Волжский Университет им. Татищева.

# Лекции по «ДИСКРЕТНОЙ МАТЕМАТИКЕ»

Составитель: доцент С.В. Каверин.

Тольятти 2002.

## Оглавление.

# Введение.

Данный курс лекций составлен в соответствии с государственным стандартом по специальностям 071900- «Информационные системы» , 220100-«Вычислительные машины, системы, комплексы и сети».

При составлении курса были использованы примеры, алгоритмы и выкладки в основном из работ [1,2]. По поводу обозначений приведем соответствующий фрагмент из книги [1].

Данная книга [1] предназначена для программистов, то есть предполагается, что читатель не испытывает затруднений в понимании текстов программ на языке высокого уровня. Именно поэтому в качестве нотации для записи алгоритмов используется некоторый неспецифицированный язык программирования, похожий по синтаксису на Паскаль (но, конечно, Паскалем не являющийся).

В программах широко используются математические обозначения, которые являются самоочевидными в конкретном контексте. Например, конструкция

**for** x∈M **do**
  P(x)
**end for**

означает применение процедуры **P** ко всем элементам множества M.

«Лишние» разделители систематически опускаются, функции могут возвращать несколько значений, и вообще, разрешаются любые вольности, которые позволяют сделать тексты программ более краткими и читабельными.

Особого упоминания заслуживают два «естественных» оператора, аналоги которых в явном виде редко встречаются в настоящих языках программирования. Оператор

**select** m∈*M*

означает выбор произвольного элемента m из множества M. Этот оператор часто необходим в «переборных» алгоритмах. Оператор

**yield x**

означает возврат значения **x**, но при этом выполнение функции не прекращается, а продолжается со следующего оператора. Этот оператор позволяет очень просто записать «генерирующие» алгоритмы, результатом которых является некоторое заранее неизвестное множество значений.

# Раздел 1. Множества и отношения.

## §1.1. Множества и их спецификации.

Понятия **"множество" и "элемент множества"** считаются первичными и поэтому не имеют строгого определения, хотя в каждом конкретном случае интуитивно должно быть ясно, что из себя представляет множество и из каких элементов оно состоит. Предполагается, что для данных элемента $a$ и множества **A** всегда можно определить, принадлежит элемент $a$ множеству **A** (пишется: $a \in A$) или элемент $a$ не принадлежит множеству **A** (пишется: $a \notin A$).

Множества, как правило, обозначаются большими латинскими буквами (A,B,C,…), а его элементы маленькими (a,b,c,…).

Множество не содержащее ни одного элемента называется **пустым** и обозначается $\emptyset$. Множество **U** содержащее все элементы, рассматриваемые в данной задаче, называется **универсальным** множеством (универсум). Множество называется **конечным,** если оно содержит конечное число элементов.

Чтобы задать множество, нужно указать, какие элементы ему принадлежат. Это можно сделать следующими способами:

1. перечислением элементов: $A=\{a_1,a_2,\ldots,a_m\}$;
2. характеристическим предикатом: $A=\{x \mid P(x)\}$, т.е. в A входят только те **x**, которые удовлетворяют условию **P(x)**;
3. A={словесное описание элементов множества};
4. порождающей процедурой: A={x| x:=f}.

Последний из перечисленных способов лежит в основе так называемого конструктивизма — направления в математике, в рамках которого рассматриваются только такие объекты, для которых известны процедуры их порождения (алгоритмы). В конструктивной математике исключаются из рассмотрения некоторые понятия и методы классической математики, чреватые возможными парадоксами.

**Примеры**

**1.** A={1,2,3,4,5,6,7,8,9};

A={n|1≤n≤9 и n-целое}

A={состоит из арабских цифр от 1 до 9}.

**2.** $A=\{(x,y) \mid x^2+y^2=1,\ x \in R,\ y \in R\}$;

A={точки единичной окружности на плоскости}

**3.** A={автомобили ВАЗ, выпущенные в 2000 году}

**4.** A={n| **for** n **from** 1 **to** 9 **print** n} – порождающая процедура.

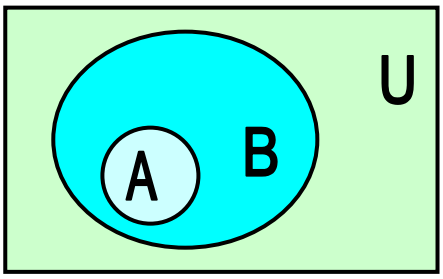
**Замечание.** Очевидно, что перечислением можно задать лишь конечные множества, которые имеют «небольшое количество» элементов.

## §1.2. Подмножества.

Если все элементы множество A являются также элементами множества B, то говорят, что множество A является **подмножеством** множества B, или A содержится в B (пишется: $A \subset B$ или $A \subseteq B$).

**Замечание.** Считается, что пустое множество $\emptyset$ является подмножеством любого другого множества.

Для пояснения различных соотношений между множествами используются **диаграммы Венна (круги Эйлера)**, где множества изображаются в виде совокупностей точек на плоскости ограниченных некоторой замкнутой кривой. Так диаграмма Венна для изображения подмножества ($A \subseteq B$) может выглядеть следующим образом




Два множества называются **равными,** если они состоят из одних и тех же элементов (обозначается A=B). Другое определение равенства множеств: $A=B \Leftrightarrow A \subseteq B$ и $B \subseteq A$.

Если $A \subset B$ и $A \neq B$, то A называется **собственным подмножеством** множества B.

Множество, состоящее из всех подмножеств множества **A,** называется **булеаном** и обозначается $\mathbf{2^A}$**.** Такое обозначение булеана можно объяснить на основе следующего факта. Если множество A конечно и состоит из n элементов (обозначается |A|=n), то булеан множества A состоит из $2^n$ элементов (т.е. $|2^A|=2^n$).

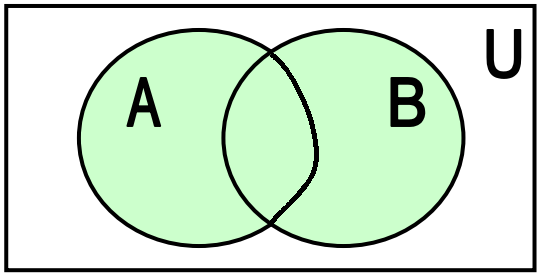
**Пример.** Пусть A={a,b,c}. Тогда |A|=3 и

$2^A=\{\emptyset,\{a\},\{b\},\{c\},\{a,b\},\{a,c\},\{b,c\},\{a,b,c\}\}$, $|2^A|=2^3=8$.

## §1.3. Операции над множествами.

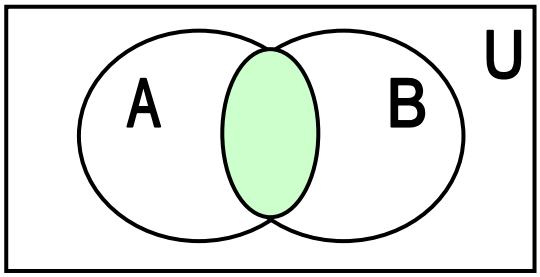
Самого по себе понятия множества еще недостаточно — нужно определить способы конструирования новых множеств из уже имеющихся, то есть ввести операции над множествами.

**1. Объединением** двух множеств A и B (обозначается **A∪B**) называется множество элементы которого принадлежат либо A, либо B либо обоим множествам сразу.



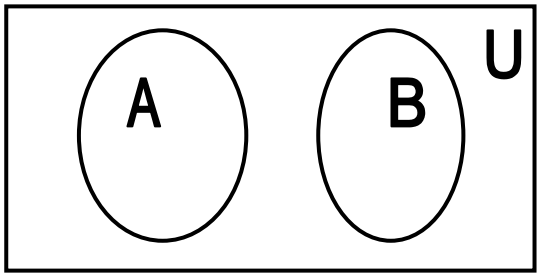

$A \cup B$

**2. Пересечением** двух множеств A и B (обозначается **A∩B**) называется множество элементы которого принадлежат и A и B одновременно.



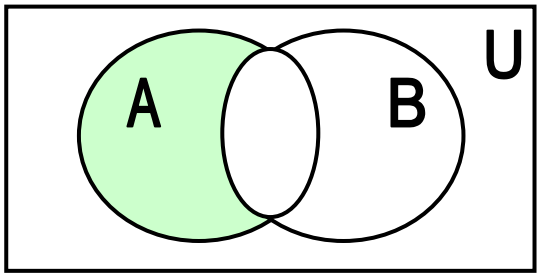

$A \cap B$

Два множества называются непересекающимися, если они не имеют общих элементов, т.е. $A \cap B = \emptyset$.




$A \cap B = \emptyset$

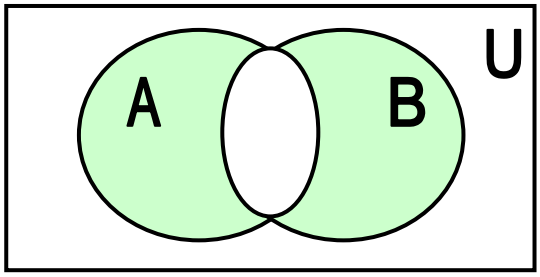
**3. Разностью** двух множеств A и B (обозначается **A\B**) называется множество элементы которого принадлежат A, но не принадлежат B.




$A \setminus B$

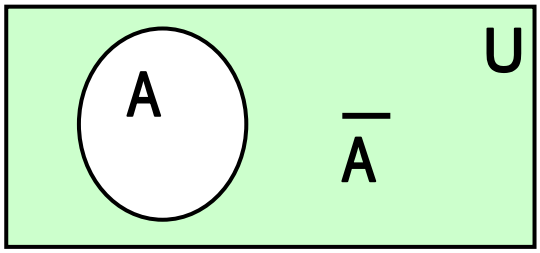
Очевидно, что если множества A и B пересекаются, то $A \setminus B \neq B \setminus A$..

**4. Симметричной разностью** двух множеств A и B (обозначается AΔB или A⊕B) называется множество элементы которого принадлежат A, или B, но не обоим сразу.




AΔB

**5. Дополнением** множества A (обозначается $\overline{A}$) называется множество, элементы которого не принадлежат A.




**Утверждение.**

1. AΔB= (A\B)∪(B\A).
2. AΔB= (A∪B)\(B∩A).
3. $\overline{A} = U \setminus A$ .
4. если множества A и B не пересекаются, то A\B=A..

## §1.4. Свойства операций над множествами

Пусть задан универсум **U**. Тогда для любых множеств A,B,C ⊆ **U** выполняются следующие свойства.

1. идемпотентность: A∪A=A, A∩A=A;
2. коммутативность: AUB=BUA, A∩B = B∩A;
3. ассоциативность:
   A∪(B∪C) = (A∪B)∪C, A∩(B∩C) = (A∩B)∩C;
4. дистрибутивность:
   A∪(B∩C) = (A∪B)∩(A∪C), A∩(B∪C) = (A∩B)∪(A∩C);
5. поглощение: (A∩B)∪A=A, (A∪B)∩A=A;
6. свойства нуля (∅): A∪∅=A, A∩∅=∅;
7. свойства единицы (**U**): A∪**U**=**U**, A∩**U**=A;

8. инволютивностъ: $\overline{\overline{A}} = A$

9. законы де Моргана: $\overline{A \cap B} = \overline{A} \cup \overline{B}$ , $\overline{A \cup B} = \overline{A} \cap \overline{B}$;

10. свойства дополнения: $A \cup \overline{A} = U$ , $A \cap \overline{A} = \varnothing$.

В справедливости перечисленных свойств можно убедиться различными способами. Например, нарисовать диаграммы Венна для левой и правой частей равенства и убедиться, что они совпадают. Можно также провести формальное рассуждение для каждого равенства. Рассмотрим для примера первое равенство: A∪A = A. Возьмем произвольный элемент **x** принадлежащий левой части равенства, x∈A∪A. По определению операции объединения ∪ имеем x∈A или x∈A. В любом случае x∈A. Взяв произвольный элемент из множества в левой части равенства, обнаружили, что он принадлежит множеству в правой части. Отсюда по определению включения множеств получаем, что A∪A⊆A. Пусть теперь x∈A.. Тогда, очевидно, верно x∈A или x∈A. Отсюда по определению операции объединения имеем x∈A∪A. Таким образом, A⊆A∪A. Следовательно, по определению равенства множеств, A∪A = A. Аналогичные рассуждения нетрудно провести и для остальных равенств.

## §1.5. Декартово произведение.

Упорядоченной n-кой называется набор из n элементов $(x_1,x_2,\ldots,x_n)$, причем порядок следования элементов фиксирован (имеет значение). Если *a* и b два объекта, то через (*a*,b) обозначают **упорядоченную пару.** Вообще говоря (*a*,b)≠(b,*a*). Равенство упорядоченных пар определяется следующим образом:

$$(a,b)=(c,d) \Leftrightarrow a=c \text{ и } b=d.$$

Например, на плоскости точка (2,1) отличается от точки (1,2).

**Прямым или декартовым произведением** двух множеств A и B называется множество упорядоченных пар, в котором первый элемент каждой пары принадлежит множеству A, а второй принадлежит B:

$$A \times B = \{(a,b) \mid a \in A,\ b \in B\}.$$

Аналогично определяется декартово произведение n множеств:

$$A_1 \times A_2 \times \ldots \times A_n = \{(a_1,a_2,\ldots,a_n) \mid a_1 \in A_1, a_2 \in A_2, \ldots, a_n \in A_n\}.$$

**Отступление.** Точка на плоскости может быть задана упорядоченной парой координат, то есть двумя точками на координатных осях. Таким образом, $R^2=R \times R$. Метод координат ввел в употребление Рене Декарт (1596-1650), отсюда и название «декартово произведение».

**Степенью** множества A называется его прямое произведение самого на себя. Обозначение: $A^n=A\times A\times\ldots\times A$. Соответственно, $A^1=A$, $A^2=A\times A$, $A^3=A\times A\times A$ и т.д.

**Пример.** Пусть A={1,2}, B={x,y}. Тогда

$A\times B=\{(1,x), (2,x), (1,y), (2,y)\}$, $B\times A=\{(x,1), (x,2), (y,1), (y,2)\}$,

$A^2=A\times A=\{(1,1), (1,2), (2,1), (2,2)\}$.

## §1.6. Отношения.

**Определение.** n-местным (n-арным) **отношением** R на множествах $A_1,A_2,\ldots,A_n$ называется подмножество декартового произведения $A_1\times A_2\times\ldots\times A_n$ , т.е. $R\subseteq A_1\times A_2\times\ldots\times A_n$.

При n=1 отношение R является подмножеством множества A и называется **унарным** отношением или свойством.

При n=2 отношение R называется **бинарным** отношением (т.е. $R\subseteq A\times B$ – бинарное отношением)**.**

Из определения отношения R следует, что R является множеством и поэтому, во-первых, может быть задано как множество, а во-вторых все теоретико-множественные операции и рассуждения распространяются и на отношения.

Для бинарных отношений обычно используется инфиксная форма записи; $a$Rb={($a$,b)|$R\subseteq A\times B$}. Если A=B, то говорят, что R есть отношение на множестве A.

**Примеры_1.6.**

1. Если A={2,3,4,5,6,7,8}, то бинарное отношение P={(x,y)|x,y∈A, x делит y и x<4} можно записать в виде P={(2,2), (2,4), (2,6), (2,8), (3,3), (3,6)}.
2. Отношение быть отцом на множестве группы людей. Здесь запись $a$Rb означает, что $a$ отец b.
3. Отношение параллельности на множестве всех прямых на плоскости. Здесь запись $a$Rb означает, что прямая $a$ параллельна прямой b.
4. Числа x,y связаны отношением R, если они являются корнями уравнения sinx=0,5, т.е. R={x| sinx=0,5 и x действительное число}.
5. Многоместные отношения используются, например, в теории баз данных. Само название «реляционная» база данных происходит от слова relation (отношение). В качестве примера рассмотрим базу данных телефонного справочника по г. Тольятти, которая состоит из трех полей (множеств): A1={фамилия, имя, отчество абонента}, A2={номера телефонов}, A3={адрес абонента}. Декартовое произведение A1×A2×A3 лишь обозначает объект

показывая, что в записи на первом поле будет выведено фамилия, имя, отчество абонента, на втором поле номер телефона абонента, а на третьем адрес абонента. Отношение R задает базу данных, т.к. из всех возможностей однозначно определяет, какому абоненту соответствует тот или иной номер телефона.

**Замечание.** По сути, все связи между различными объектами являются отношениями.

**Определение.** Пусть $R \subseteq A \times B$ бинарное отношение. Множество $X=\{x|\ x \in A$ и $(x,y) \in R$ для некоторого $y \in B\}$ называется **областью определения** отношения R. Множество $Y=\{y|\ y \in B$ и $(x,y) \in R$ для некоторого $x \in A\}$ называется **областью значений** отношения R.

**Определения.** Пусть R бинарное отношение на множестве A.

$R^{-1}=\{(x,y)|(y,x) \in R\}$ – называется **обратным** отношением.

$I_A=\{(x,x)|x \in A\}$ – называется **тождественным** отношением.

$U_A=\{(x,y)|x \in A$ и $y \in A\}=A \times A$ – называется **универсальным** отношением.

**Определение.** Пусть $P1 \subseteq A \times B$, $P2 \subseteq B \times C$ – два бинарных отношения. **Композицией** отношений P1 и P2 (обозначается $P1 \circ P2$) называется множество $P1 \circ P2=\{(x,y)|x \in A$, $y \in C$, и найдется элемент $z \in B$ такой, что $(x,z) \in P1$ и $(z,y) \in P2\}$ (см.рис.1.6.1.).

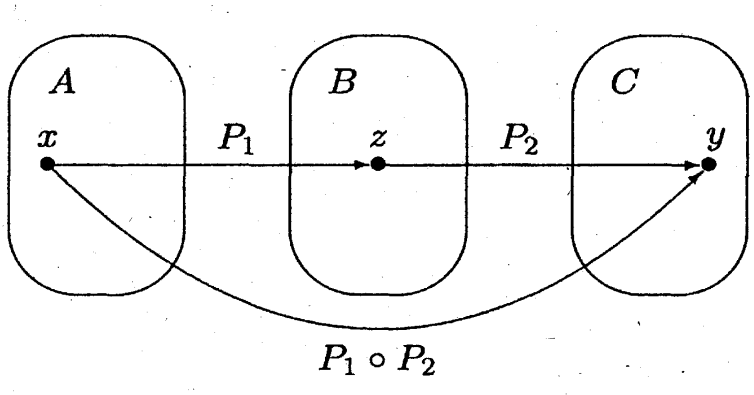



**Рисунок 1.6.1.**

## §1.7. Графическое представление бинарных отношений.

Бинарные отношения $R \subseteq A \times B$ иногда удобно изображать графически. Рассмотрим три таких способа.

**1-способ.** Начертим пару взаимно перпендикулярных осей (Ox — горизонтальная ось, Oy — вертикальная ось), на каждой оси отметим точки, представляющие элементы множеств A и B соответственно. Отметив на плоскости точки с координатами (x,y) такие, что $(x,y) \in R$, получаем, множество, соответствующее отношению R.

**Пример_1.7.1.** A={a,b,c,d}, B={1,2}, R={(a,1), (a,2), (c,2), (d,2)}. На рис.1.7.1. представлено графическое изображение R.

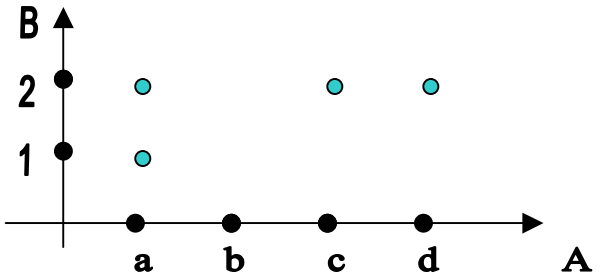



Рис.1.7.1.

**2-способ.** Другой способ состоит в том, что элементы x∈A и y∈B, связанные отношением R, соединяются стрелками.

**Пример_1.7.2.** A={a,b,c}, B={1,2,3}, R={(a,2), (b,1), (c,2)}. На рис.1.7.2а) представлено графическое изображение R.

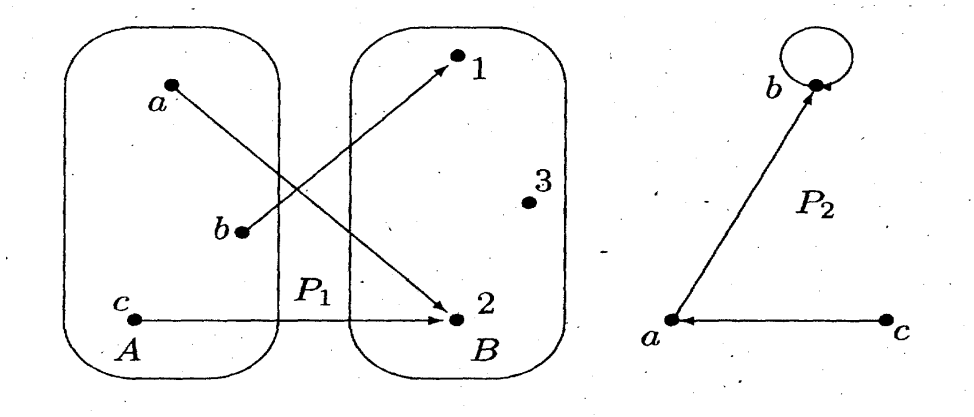



**Рис.1.7.2.** **а)** **б)**

**3-способ.** Данный способ применим только к отношениям заданным на множестве A, т.е. R⊂A×A. Точки множества A изображаются точками на плоскости. Тогда упорядоченной паре (a,b) соответствует стрелка идущая от a к b. Так получается диаграмма отношения R.

**Пример_1.7.3.** A={*a*,b,c}, B={1,2,3}, R={(*a*,b), (b,b), (c,*a*)}. На рис.1.7.2б). представлено графическое изображение R.

## §1.8. Свойства отношений

Пусть $R \subset A^2$ – бинарное отношение на множестве A. Тогда отношение R называется

- **рефлексивным**, если для любого $a \in A$ следует, что $(a,a) \in R$;
- **антирефлексивным,** если для любого $a \in A$ следует, что $(a,a) \notin R$;
- **симметричным,** если для любых $a,b \in A$ связанных отношением R (т.е. $(a,b) \in R$), следует, что $(b,a) \in R$;

- **антисимметричным,** если для любых $a,b\in A$ таких, что $(a,b)\in R$ и $(b,a)\in R$, следует, что $a=b$;
- **транзитивным,** если для любых $a,b,c\in A$ таких, что $(a,b)\in R$ и $(b,c)\in R$, следует, что $(a,c)\in R$;

**Теорема 1.8.1.** Пусть $R\subset A^2$ – бинарное отношение на А. Тогда:

**1.** R рефлексивно $\Leftrightarrow I_A\subseteq R$, где $I_A=\{(x,x)|x\in A\}$ – тождественное отношение;

**2.** R симметрично $\Leftrightarrow R^{-1}=R$;

**3**. R транзитивно $\Leftrightarrow R\circ R\subset R$;

**4.** R антисимметрично $\Leftrightarrow R^{-1}\cap R\subset I_A$;

**5.** R антирефлексивно $\Leftrightarrow I_A\cap R=\emptyset$.

В более общей ситуации мы можем интерпретировать рассмотренные выше характеристики отношения путем построения диаграммы:

а) отношение рефлексивно тогда и только тогда, когда для каждого узла (точки) на диаграмме существует стрелка, которая начинается и заканчивается на этом узле (петля);

б) отношение антирефлексивно тогда и только тогда, когда для каждого узла на диаграмме не существует петли;

в) отношение симметрично тогда и только тогда, когда для каждой стрелки, соединяющей два узла, существует также стрелка, соединяющая эти узлы в обратном направлении;

г) отношение транзитивно тогда и только тогда, когда для каждой пары узлов **x** и **y**, связанных последовательностью стрелок от **x** к $\boldsymbol{a_1}$, от $\boldsymbol{a_1}$ к $\boldsymbol{a_2}$, ..., от $\boldsymbol{a_{m-1}}$ к $\boldsymbol{a_m}$, от $\boldsymbol{a_m}$ к **y**, существует также стрелка от **x** к **y;**

д) отношение антисимметрично тогда и только тогда, когда не существует двух различных узлов, связанных парой стрелок, т.е. если есть стрелка (x,y), то нет стрелки (y,x).

**Пример_1.8.1.** Отношение на рис.1.8.1. является антирефлексивным и только.

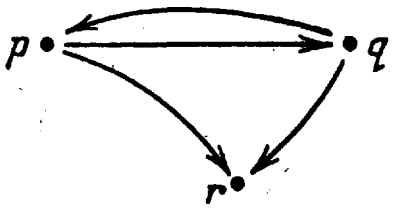



**Рис.1.8.2.**

## §1.9. Матрица бинарного отношения.

Рассмотрим конечное множество $A=\{a_1,a_2,\dots a_n\}$, и бинарное отношение P на A: $P\subset A^2$. Определим матрицу $[P]=(p_{ij})$ порядка n бинарного отношения P по следующему правилу:

$$p_{ij}=\begin{cases}1, & \text{если } (a_i,a_j)\in P\\ 0, & \text{если } (a_i,a_j)\notin P\end{cases}$$

Полученная матрица содержит полную информацию о связях между элементами и позволяет представлять эту информацию на компьютере. Заметим, что любая матрица, состоящая из нулей и единиц, является матрицей некоторого бинарного отношения.

**Пример 1.9.1.** Матрица бинарного отношения $P\subset A^2$, $A=\{1,2,3\}$, заданного на рис.1.9.1., имеет вид

$$[P]=\begin{Vmatrix}1 & 1 & 1\\ 0 & 0 & 1\\ 1 & 0 & 0\end{Vmatrix}$$

**Таб.1.9.1.**

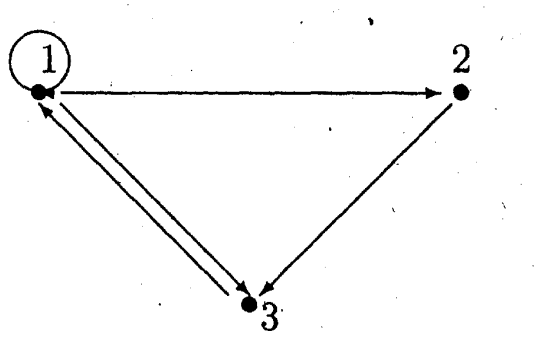



**Рис 1.9.1**

Отметим основные свойства матриц бинарных отношений заданных на множестве A.

**1.** Если $P,Q\subset A\times B$, $[P]=(p_{ij})$, $[Q]=(q_{ij})$, то $[P\cup Q]=(p_{ij}+g_{ij})$ и $[P\cap Q]=(p_{ij}\cdot qij)$, где сложение (дизъюнкция) осуществляется по правилам 0+0=0, 1+1=1+0=0+1=1, а умножение — обычным образом. Итак, $[P\cup Q]=[P]+[Q]$, а матрица $[P\cap Q]$ получается перемножением соответствующих элементов из [P] и [Q]: $[P\cap Q]=[P]*[Q]$.

**Пример 1.9.1.** Пусть $[P]=\begin{pmatrix}011\\110\\010\end{pmatrix}$, $[Q]=\begin{pmatrix}001\\010\\110\end{pmatrix}$ – матрицы отношений P и Q. Тогда

$$[P\cup Q]=\begin{pmatrix}011\\110\\110\end{pmatrix},\quad [P\cap Q]=\begin{pmatrix}001\\010\\010\end{pmatrix}.$$

**2.** Если $P\subset A\times B$, $Q\subset B\times C$, то $[P\circ Q]=[P]\times[Q]$, где умножение матриц [P] и [Q] производится по обычному правилу умножения матриц («строка на столбец»), но произведение и сумма (дизъюнкция) элементов из [P] и [Q] — по определенным в п. 1 правилам.

**Пример 1.9.2.** Пусть [P], [Q] – матрицы отношений P и Q из примера 1.9.1. Тогда

$$[P\circ Q]=\begin{pmatrix} 0+0+0 & 0+1+1 & 0+0+0 \\ 0+0+0 & 0+1+0 & 1+0+0 \\ 0+0+0 & 0+1+0 & 1+0+0 \end{pmatrix}=\begin{pmatrix} 010 \\ 011 \\ 011 \end{pmatrix}$$

**3.** Матрица обратного отношения $P^{-1}$ равна транспонированной матрице отношения P: $[P^{-1}]=[P]^t$.

**Пример 1.9.3.** Пусть $[P]=\begin{pmatrix} 011 \\ 110 \\ 010 \end{pmatrix}$, $[P^{-1}]=\begin{pmatrix} 010 \\ 111 \\ 100 \end{pmatrix}$.

**4.** Если $P\subset Q$, $[P]=(p_{ij})$, $[O]=(q_{ij})$, то $p_{ij}\le q_{ij}$.

**5.** Матрица тождественного отношения $[I_A]$ - единична.

**Теорема 1.9.1.** Пусть R — бинарное отношение на множестве A, т.е. $R\subset A^2$.

**1.** R рефлексивно $\Leftrightarrow$ $[I_A]\subseteq[R]$, т.е. на главной диагонали матрицы [R] стоят единицы ;

**2.** R симметрично $\Leftrightarrow$ [R]-симметричная матрица, т.е. $[R]=[R]^t$;

**3**. R транзитивно $\Leftrightarrow$ $[R\circ R]\subset[R]$;

**4.** R антисимметрично $\Leftrightarrow$ $[R^{-1}\cap R]\subset[I]_A$, т.е. вне главной диагонали все элементы равны нулю;

**5.** R антирефлексивно $\Leftrightarrow$ на главной диагонали матрицы [R] стоят только нули.

## §1.10. Отношение эквивалентности.

Пусть R — бинарное отношение на множестве A, т.е. $R\subset A^2$.

Отношение R называется **отношением эквивалентности (эквивалентностью)** если оно рефлексивно, симметрично и транзитивно.

Эквивалентность часто обозначают символом ~ (тильда), т.е. вместо записи xRy пишут x~y и говорят «x (икс) эквивалентен y (игрек)». Важность отношения эквивалентности объясняется наличием следующего утверждения.

**Теорема 1.10.1.** Если на множестве A задано отношение эквивалентности R, то множество A разбивается на непересекающиеся множества A1,A2,..., эквивалентных друг другу элементов. Множества A1,A2,... называются **классами эквивалентности.**

Множество, состоящее из классов эквивалентности, называется фактор-множеством и обозначается A/R (или A/~).

Обозначим $M[x]=\{y|x\sim y, y\in A\}$-множество элементов эквивалентных x.

**Теорема 1.10.2.**

1. Если $x\sim y$, то $M[x]=M[y]$. Т.о., класс эквивалентности однозначно определяется любым своим элементом.
2. Если x не эквиалентен y, то $M[x]\cap M[y]=\emptyset$.

**Примеры.**

1. Разбиение студентов на группы является отношением эквивалентности. Здесь два студента считаются эквивалентными, если они учатся в одной группе. Фактор-множество - множество групп ВУЗа.
2. Отношение равенства x=y является эквивалентностью на любом множестве A.
3. Отношение подобия на множестве треугольников.
4. Отношение R, заданное на множестве A={1,2,3,4,5} графически (см. рис.1.10.1), является отношением эквивалентности, т.к. оно рефлексивно, симметрично и транзитивно. Фактор множество A/R={{1,4},{2,3,5}}.

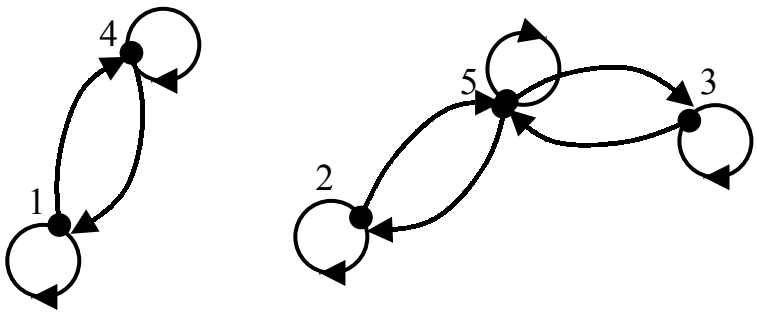



Рис.1.10.1.

## §1.11. Отношение порядка.

Пусть R — бинарное отношение на множестве A, т.е. $R\subset A^2$.

Отношение R называется **отношением строгого порядка**, если оно антирефлексивно, антисимметрично и транзитивно. Обычно обозначается "<"или ">".

Отношение R называется **отношением частичного порядка (нестрогого порядка)**, если оно рефлексивно, антисимметрично и транзитивно. Обычно обозначается ≤ или ≥.

Множество A с заданным частичным порядком ≤ называется **частично упорядоченным множеством** (ч.у.м.) и обозначается (A,≤). Отметим, что в частично упорядоченном множестве не все элементы могут быть связаны отношением ≤ (сравнимы между собой). Если все элементы ч.у.м. (A,≤) сравнимы между собой, то множество

называется **линейно упорядоченным**, а порядок ≤ в этом случае называется **линейным.**

Очевидно, что отношение строгого порядка не является отношением частичного порядка.

**Пример_1.11.1.** Линейным порядком является обычное отношение ≤ на множестве натуральных чисел N, т.к. для любых чисел x,y выполняется или x≤y или y≤x.

**Пример_1.11.2.** Отношение, изображенное на рис. 1.11.1, является частичным порядком

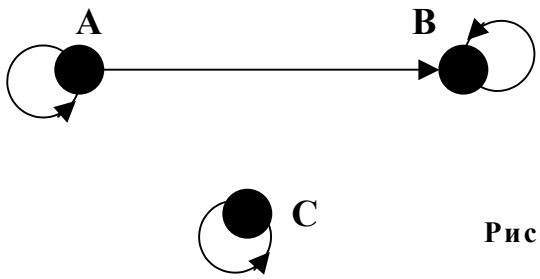



**Рис.1.11.1.**

**Пример_1.11.3.** Отношение включения ⊆ на булеане $2^U$ образует частичный порядок. Очевидно, что если множества A и B не пересекаются, то они не сравнимы между собой.

**Пример_1.11.4.** Определим на множестве $A^2$ отношение ≤ условием: (a,b)≤(a1,b1) ⇔ a≤a1 и b≤b1. Отношение ≤ есть отношение частичного порядка. Оно называется отношением Парето. Так вектора (0,1) и (2,1) сравнимы между собой: (0,3)≤(2,3), т.к. 0≤2 и 3≤3. Очевидно, что вектора (2,1) и (0,3) не сравнимы между собой.

Элемент ***a*∈A** частично упорядоченного множества U=(A,≤) называется **максимальным (минимальным)**, если для всех x∈A из *a*≤x (x≤*a*) следует x=*a*. Элемент ***a***∈A называется **наибольшим (наименьшим)**, если x≤*a* (*a*≤x) для всех x∈A. Наибольший (наименьший) элемент ч.у.м. U (если он существует) обозначается через maxU (minU). Наибольший элемент часто называют единицей, а наименьший – нулем множества U.

**Пример_1.11.5.** Ч.у.м. ({A,B,C},≤), изображенное на рис.1.11.2., имеет наибольший элемент B, минимальные элементы A,C, но не имеет наименьшего элемента.

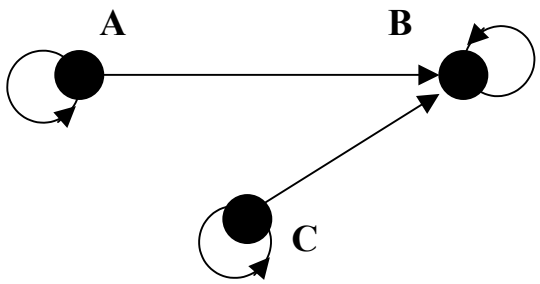



**Рис.1.11.2.**

**Теорема_1.11.1.** Во всяком конечном непустом частично упорядоченном множестве существует минимальный элемент.

**Теорема_1.11.2.** Всякий частичный порядок на конечном множестве может быть дополнен до линейного.

**Доказательство** теорема_1.11.2. См. алгоритм 1.11.1.

Линейный порядок ≤ на множестве A называется **полным,** если каждое непустое подмножество множества A имеет наименьший элемент. Пара (A,≤), в которой отношение ≤ является полным порядком на множестве A, называется **вполне упорядоченным множеством** (сокращенно в.у.м.).

**Алгоритм 1.11.1.** Алгоритм топологической сортировки

**Вход:** частично упорядоченное множество U.

**Выход:** вполне упорядоченное множество W.

```
while U≠∅ do
  m:=M(U) {функция M возвращает минимальный элемент такой
           элемент m существует по теореме_1.11.1.}
  yield m    {вывод элемента m}
     U:=U\{m}
end while.
```

Рассмотрим непустое множество символов X={x,y,z,...}, называемое алфавитам. Конечные, наборы написанных друг за другом символов из X называются словами (например, x, y, xy, yx, zxx, xyyz и т. д.). Элемент $x_i$ слова $x_1,x_2,...x_n$ называется его i-й координатой. Число n называется длиной слова $x_1,x_2,...x_n$. Множество слов алфавита X обозначим через W(X). При этом будем считать, что W(X) содержит слово ∧, не имеющее символов и называемое пустым словом. Длина пустого слова ∧ по определению равна нулю.

Пусть ≤ отношение порядка на множестве X. Определим на множестве W(X) отношение **лексикографического порядка L** по следующему правилу: $x_1,x_2,...x_m L y_1,y_2,...y_n$ если выполняется одно из двух условий

1. $m \le n$ и $x_i = y_i$ для всех $1 \le i \le n$

2. существует $i \le m$ такое, что $x_i < y_i$, и для всех $j < i$ выполняется $x_j = y_j$)•

**Утверждение 1.11.1.** Если (X,≤) — линейно упорядоченное множество (л.у.м.), то (W(X),L) — л.у.м.

**Пример_1.11.6.** Рассмотрим множество букв русского алфавита, которое обозначим через X. Определим на X полный порядок ≤ в соответствии с обычным упорядочением букв по алфавиту. Отношение L на множестве слов W(X) определяет упорядочение слов, по которому составляются словари русского языка.

Рассмотрим частично упорядоченное множество (A,≤). Говорят, что элемент **y** покрывает элемент **x**, если **x≤y** и не существует такого элемента **z**, что **x≤z≤y.** Если множество A конечно, то частично

упорядоченное множество (A,≤) можно представить в виде схемы, в которой каждый элемент изображается точкой на плоскости, и если **y** докрывает **x** то точки **x** и **y** соединяют отрезком, причем точку, соответствующую **x**, располагают ниже **y.** Такие схемы называются **диаграммами Хассе.**

**Пример 1.11.7.** Рассмотрим частично упорядоченное множество $(2^A,\subseteq)$, где A={1,2,3}. Множество $2^A$ содержит восемь элементов: {∅,{1},{2},{3},{1,2},{1,3},{2,3},{1,2,3}}. На рис._1.11.3а изображена диаграмма Хассе, соответствующая $(2^A,\subseteq)$.

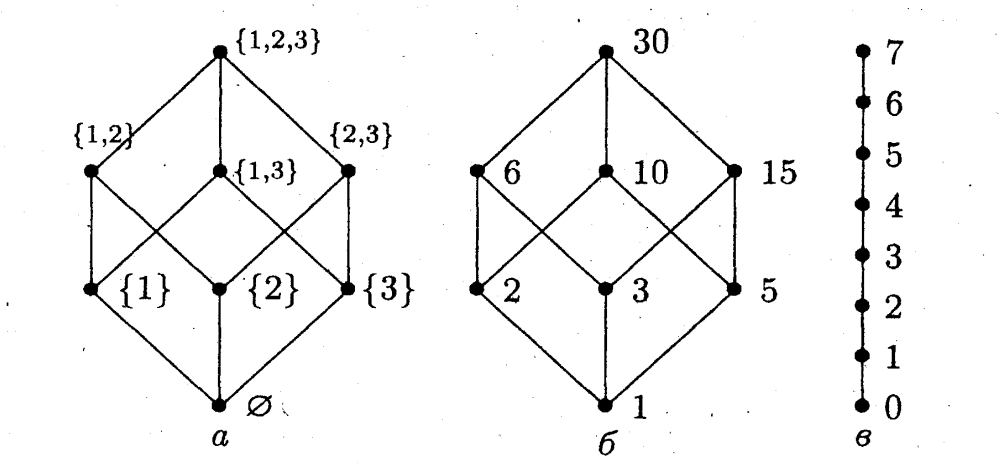



**Рис.1.11.3.**

**Пример 1.11.8.** Пусть A={1,2,3,5,6,10,15,30}. Рассмотрим отношение частичного порядка ≤ на множестве A, задаваемое по правилу: x≤y ⇔ y делится на x. Диаграмма Хассе для частично упорядоченного множества (A,≤) изображена на рис_1.11.3б.

**Пример 1.11.9.** На рис. 1.11.3в изображена диаграмма Хассе линейно упорядоченного множества ({0,1,2,3,4,5,6,7},≤) с обычным отношением порядка на множестве натуральных чисел, не превосходящих семи.

Заметим, что диаграммы Хассе первых двух отношений, изображенных на рис.1.11.3., совпадают. Это означает, что эти частично упорядоченные множества имеют одинаковую структуру, причем отличную от структуры третьего частично упорядоченного множества, хотя оно тоже содержит восемь элементов. Формально такая общность структуры определяется понятием изоморфизма.

## §1.12. Функции.

Пусть задано бинарное отношение f⊆A×B.

Определение_1.12.1. Отношение f⊆A×B называется **многозначным отображением,** если f задает правило, по которому

некоторым (не обязательно всем) элементам $x \in A$ ставится в соответствие **один или несколько** элементов $y \in B$.

Определение 1.12.2. Отношение $f \subseteq A \times B$ называется **функцией (однозначным отображением),** если f задает правило, по которому каждому элементу $x \in A$ ставится в соответствие **только один** элемент $y \in B$.

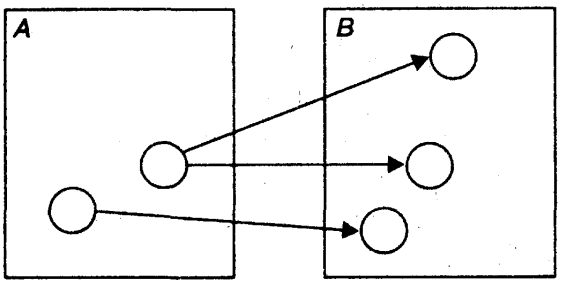
**Замечание_1.** Если отношение $f \subseteq A \times B$ задает правило, по которому некоторым (не обязательно всем) элементам $x \in A$ ставится в соответствие только один элемент $y \in B$, то оно называется **частичной функцией**.

Обозначается функция следующим образом: $f: A \to B$. Часто на практике используется и префиксная форма записи – $y=f(x)$, в этом случае **x** называют **аргументом**, а **y** — **значением** функции f.

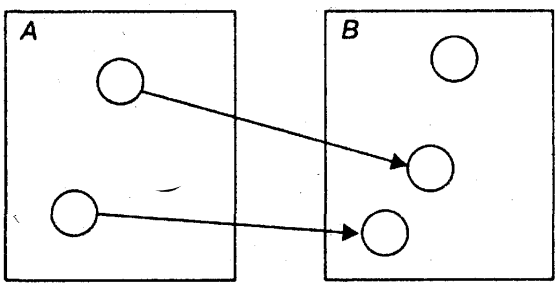
**Замечание_2.** Если в определении функции f множество $A=A_1 \times A_2 \times \ldots \times A_n$, то функция f называется **функцией n аргументов** или n местной функцией.

**Определение 1.12.3.** Пусть $f: A \to B$ функция. Тогда функция f называется:

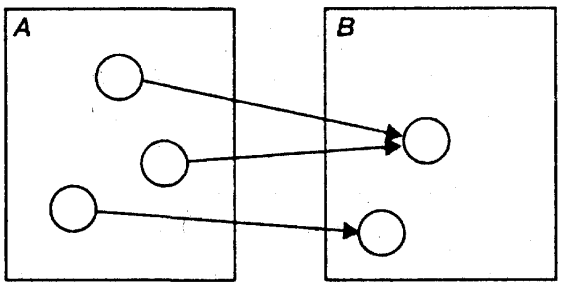
- **иньективной,** (разнозначной), если разным x1 и x2 ($x1 \neq x2$) изA, соответствуют разные y1 и y2 ($y1 \neq y2$) из B;
- **сюрьективной** (отображение «на»), если для любого элемента $b \in B$ найдется элемент $a \in A$ такой, что $b=f(a)$;
- **биективной** (взаимнооднозначной), если она инъективная и сюръективная.



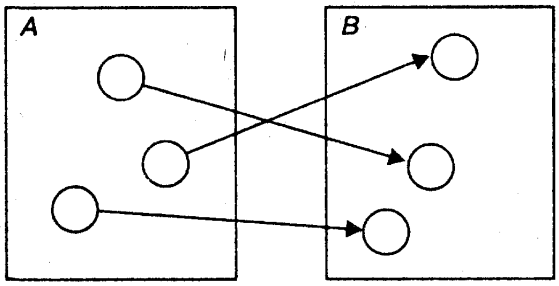

Отношение, но не функция



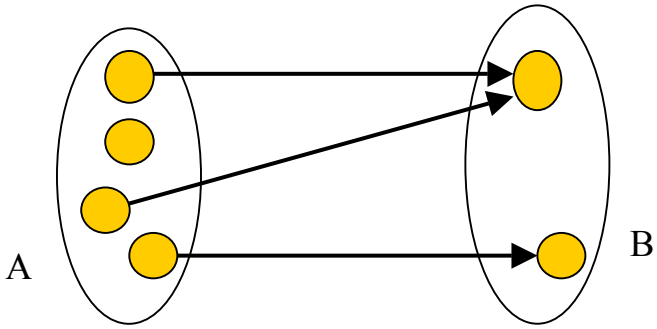

Инъекция, но не сюръекция




Сюръекция, но не инъекция




Биекция




**Частично определенная функция**

**Рис.1.12.1.**

**Теорема_1.12.1**. Если f: A→B — биекция, то отношение $f^{-1}$:B×A (**обратная функция**) является биекцией.

**Теорема 1.12.2**. Если f: A→B — инъекция, но не сюръекция, то отношение $f^{-1}$:B×A (обратная функция) является частично определенной функцией.

**Теорема_1.12.3.** Пусть A и B конечные множества. Биекция f:A→B существует тогда и только тогда, когда множества A и B имеют одинаковое число элементов.

**Определение 1.12.3.** Пусть f:A→B функция.

- **Образом** множества A1⊆A называется множество
  f(A1)={b|b∈B и существует x∈A1, что b=f(x)}.
- **Прообразом** множества B1⊆B называется множество
  $f^{-1}$(B1)={a|a∈A и существует y∈B1, что y=f(a)}.
- **Областью определения** (частичной) функции f называется множество элементов из A, для которых существуют образы в B;
- **Областью значений** функции называется множество элементов из B, для которых существуют прообразы в A.

**Пример.** Пусть заданы множества A={1,2,3,4,5}, B={a,b,c,d} и функция f:A→B, изображенная на рис.1.12.2.

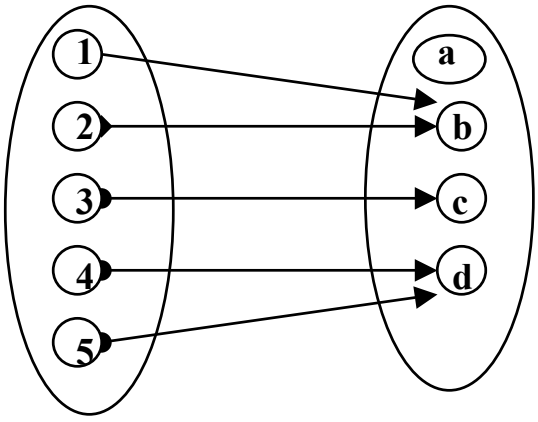



**Рис.1.12.2.**

$f(\{1,2\})=\{b\}$, $f(\{4\})=\{d\}$, $f(\{2,5\})=\{b,d\}$, $f(\{2,3,5\})=B$,
$f^{-1}(\{b\})=\{1,2\}$, $f^{-1}(\{b,d\})=\{1,2,4,5\}$, $f^{-1}(\{a\})=\emptyset$,
Область определения$=\{1,2,3,4,5\}=A$, Область значений$=\{b,c,d\}\subset B$.

## §1.13. Мощность множеств.

Понятие мощности возникает при сравнении множеств по числу элементов.

**Определение.** Множества A и B назовем эквивалентными (обозначается $A\sim B$), если существует биекция $f:A\to B$.

**Утверждение.** Введенное понятие эквивалентности является отношением эквивалентности, т.е. оно рефлексивно, симметрично и транзитивно.

Доказательство.

1. Рефлексивность ($A\sim A$): здесь в качестве биекции следует взять тождественное отношение (функцию).
2. Симметричность ($A\sim B\Rightarrow B\sim A$). Пусть $f:A\to B$ исходная биекция. Тогда обратная функция $f^{-1}:B\to A$ является искомой биекцией.
3. Транзитивность ($A\sim B$ и $B\sim C \Rightarrow A\sim C$). Пусть $f:A\to B$ и $g:B\to C$ исходные биекции. Тогда композиция $f\circ g:A\to C$ является искомой биекцией. Утверждение доказано.

**Определение.** Мощностью конечного множества является число его элементов.

Множество, не являющееся конечным, называется **бесконечным.** Определим теперь мощность некоторых бесконечных множеств.

**Определение.** Множество A называется **счетным,** если $A\sim N$, где N множество натуральных чисел.

**Определение.** Говорят, что множество A имеет мощность континуума**,** если $A\sim[0,1]$, где $[0,1]$ – множество точек отрезка $[0,1]$.

Мощность множества A **обозначается |A|.**

**Примеры.**

1. Пусть $A=\{1,2,3,5\}$, $B=\{a,b,c,f\}$. Тогда $|A|=4$, $|B|=4$, т.е. множества A и B имеют одинаковую мощность (равномощны)
2. Счетными являются следующие множества:
   - Множество четных чисел;
   - Множество целых чисел;
   - Множество рациональных чисел.
3. Следующие множества имеют мощность континуума:
   - Множество точек на отрезке $[0,5]$;
   - Множество точек квадрата $[1,10]\times[1,10]$;
   - Множество точек пространства $R^3$.

**Замечание_1.** На мощность множества A можно смотреть и как на новый объект, называемый **кардинальным числом** или кардиналом. Очевидно, что кардинальное число конечного множества есть число его элементов.

**Замечание_2.** Существование биекции между двумя эквивалентными множествами позволяет переносить изучение свойств с одного множества на другое, когда природа элементов не важна. Например, если |A|=n, то с элементами множества A можно работать как с числами 1,2,...,n.

## §1.14. Представление множеств в ЭВМ.

**1.** Пусть универсум U — конечный, и число элементов в нем не превосходит разрядности ЭВМ. Элементы универсума нумеруются: $U=\{u_1,u_2,...,u_n\}$. Подмножество A универсума U представляется **кодом (машинным словом или битовой шкалой)** $C(A)=\{c_1,c_2,...,c_n\}$, в котором:

$$c_i = \begin{cases} 1, & \text{если } u_i \in A \\ 0, & \text{если } u_i \notin A \end{cases}.$$

здесь $c_i$— это i-й разряд кода C(A).

Код пересечения множеств A и B есть поразрядное логическое произведение кода множества A и кода множества B. Код объединения множеств A и B есть поразрядная логическая сумма кода множества A и кода множества B. Код дополнения множества A есть инверсия кода множества A. Таким образом, операции над небольшими множествами выполняются весьма эффективно.

**2.** Если универсум очень велик (или бесконечен), а рассматриваемые подмножества универсума не очень велики, то представление с помощью битовых шкал не является эффективным с точки зрения экономии памяти. В этом случае множества обычно представляются **списками элементов.** Элемент списка при этом представляется записью с двумя полями: информационным и указателем на следующий элемент. Весь список представляется указателем на первый элемент.

```
elem=record
    i: info;      { информационное поле }
    n:↑ elem      { указатель на следующий элемент }
    end record.
```

**3. Представление функций в ЭВМ.** Пусть **f:A→B,** причем множество A конечно и не очень велико, |A|=n. Наиболее общим представлением такой функции является массив **array [A] of B**, где

A—тип данных, значения которого представляют элементы множества A, а B—тип данных, значения которого представляют элементы множества B. Если среда программирования допускает массивы только с натуральными индексами, то элементы множества A нумеруются (то есть $A=\{a_1,a_2,\ldots,a_n\}$) и функция представляется с помощью массива **array [l..n] of B**. Функция нескольких аргументов представляется многомерным массивом.

## §1.15. Алгоритмы на множествах.

**1.** Рассмотрим алгоритм типа слияния, который определяет, является ли множество **A подмножеством множества B.**

**Алгоритм 1.15.1.** Проверка включения слиянием

**Вход:** проверяемые множества A и B, которые заданы указателями a и b.

**Выход:** 1, если $A \subset B$, в противном случае 0.

```
pa:=a; pb:=b
while pa≠nil & pb≠nil do
    if pa.i < pb.i then
        return 0    {элемент множества A отсутствует в множестве B}
    else if pa.i > pb.i then
        pb:=pb.n    {элемент множества A, может быть, присутствует в
                     множестве B}
    else
        pa:=pa.n    { здесь pa.i=pb.i, то есть }
        pb:=pb.n    {элемент множества A точно присутствует в
                    множестве B }
    end if
end while
return pa=nil
```

**Обоснование.** На каждом шаге основного цикла возможна одна из трех ситуаций: текущий элемент множества A меньше, больше или равен текущему элементу множества B. В первом случае текущий элемент множества A заведомо меньше, чем текущий и все последующие элементы множества B, а потому он не содержится в множестве B, и можно завершить выполнение алгоритма. Во втором случае происходит продвижение по множеству B в надежде отыскать элемент, который совпадает с текущим элементом множества A. В третьем случае, найдены совпадающие элементы, и происходит продвижение сразу в обоих множествах. По завершении основного цикла возможны два случая: либо pa=**nil,** либо pa≠**nil.** Первый случай означает, что для всех элементов множества A удалось найти совпадающие элементы в множестве B. Второй случай означает, что множество B закончилось раньше, то есть не для всех элементов множества A удалось найти совпадающие элементы в множестве B.

**2.** Рассмотрим алгоритм типа слияния, который вычисляет **пересечение двух множеств,** представленных упорядоченными списками.

**Алгоритм 1.15.2.** Вычисление пересечения слиянием

**Вход:** пересекаемые множества A и B, заданные указателями **a** и **b.**

**Выход:** пересечение C=A∩B, заданное указателем **c**.

```
pa:=a; pb:=b; c:=nil; e:=nil
while pa≠nil & pb≠nil do
   if pa.i < pb.i then
      pa:=pa.n {элемент множества A не принадлежит пересечению}
   else if pa.i > pb.i then
      pb:=pb.n {элемент множества B не принадлежит пересечению}
   else {здесь pa.i=pb.i — данный элемент принадлежит пересечению}
      Append(c,e,pa.i); pa:=pa.n; pb:=pb.n
   end if
end while
```

**Обоснование.** На каждом шаге основного цикла возможна одна из трех ситуаций: текущий элемент множества A меньше, больше или равен текущему элементу множества *B*. В первом случае текущий элемент множества A не принадлежит пересечению, он пропускается и происходит продвижение в этом множестве, во втором то же самое производится с множеством B. В третьем случае найдены совпадающие элементы, один экземпляр элемента добавляется в результат и происходит продвижение сразу в обоих множествах. Таким образом, в результат попадают все совпадающие элементы обоих множеств, причем ровно один раз.

**3.** Вспомогательная процедура **Append(c,e,d)** присоединяет элемент d к хвосту e списка c.

**Алгоритм 1.15.3.** Процедура Append

**Вход:** указатель **c** на первый элемент списка, указатель **e** на последний элемент списка, добавляемый элемент d.

**Выход:** указатель **c** на первый элемент списка, указатель **e** на последний элемент списка.

```
q:=new(elem); q.i:=d; q.n:=nil {новый элемент списка}
if c=nil then
   c:=q
else
   e.n:=g
end if
e:=q.
```

**Обоснование.** До первого вызова функции Append имеем: **c=nil** и **e=nil.** После первого вызова указатель **c** указывает на первый элемент

списка, а указатель **e** — на последний элемент (который после первого вызова является тем же самым элементом). Если указатели **c** и **e** не пусты и указывают на первый и последний элементы правильного списка, то после очередного вызова функции Append это свойство сохраняется, поскольку из текста основной программы видно, что, кроме инициализации, все остальные манипуляции с этими указателями выполняются только с помощью функции Append.

# Раздел 2. Теория графов.

При изучении дискретных систем, в частности, систем преобразования информации, требуется строить математические модели, описывающие структуру или функционирование таких систем.

Одним из наиболее важных понятий, относящихся к структуре дискретных систем, является понятие графа. Это понятие, описывающее структуру связей между отдельными частями системы, в силу своей общности используется во многих математических моделях.

Графы очень часто используются в приложениях, поскольку они возникают как модель при изучении многих объектов. Например, блок-схема алгоритма представляет собой орграф, в котором вершинами являются отдельные операторы, а дуги указывают переходы между ними. Другие примеры графов: схема дорог, узлы и соединения в электрической цепи, множество предприятий с указанием двухсторонних связей между ними, структура управления с указанием объектов и их подчиненности друг другу и т.д.

## Глава 1. Графы. Основные понятия.

### §1.1. Определение графов.

**Определение**. Графом G (в широком понимании) называется любая пара (V,E), где $V=\{v_1,v_2,\ldots,v_n\}$ – конечное множество элементов любой природы, а $E=\{e_1,e_2,\ldots,e_m\}$ - семейство пар элементов из V, причем допускаются пары вида $(v_i, v_i)$ и одинаковые пары. Если пары в V рассматриваются как неупорядоченные, то граф называется **неориентированным,** если как упорядоченные, то граф называется **ориентированным (орграфом).**

Элементы множества V называются **вершинами** графа, а пары из E – его **ребрами;** в орграфе они называются ориентированными ребрами или **дугами.** Пара вида $(v_i, v_i)$ называется **петлей** в вершине $v_i$. Если пара $(v_i, v_j)$ встречается в E более одного раза, то говорят, что $(v_i, v_j)$ – **кратное ребро**. Говорят, что ребро e=(u,v) в неориентированном графе соединяет вершины u и v, а в ориентированном графе дуга e=(u,v) идет из вершины u в вершину v.

Дадим понятие графа в узком смысле, которое и будем использовать в дальнейшем.

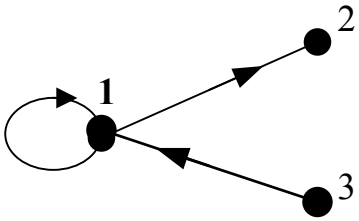
**Определения. 1.** Граф (в широком понимании) с кратными ребрами и петлями называют **псевдографом** (псевдоорграфом).

**2**. Граф (в широком понимании) с кратными ребрами, но без петель называют **мультиграфом** (псевдомультиграфом).

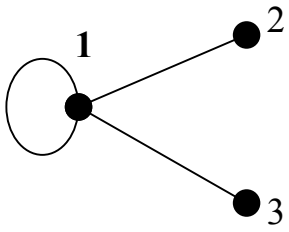
**3. Графом** (орграфом) называется граф (в широком понимании) без петель и кратных ребер.

Графы (орграфы) можно графически изображать следующим образом. Вершины будем изображать точками, а каждое ребро (дугу) $(v_i,v_j)$ - линией, соединяющей точки $v_i$ и $v_j$. Если $(v_i,v_j)$ - дуга, то на этой линии будем указывать стрелку от $v_i$ к $v_j$.

**Примеры. 1**. Псевдографы (в вершине 1 – петля).



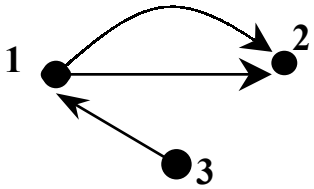

а) ориентированный псевдограф



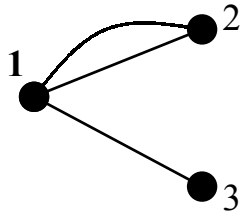

б) неориентированный псевдограф

**рис.1.1.1.**

**2**. Мультиграфы (из вершины 1 идут две дуги/ребра в вершину 2).



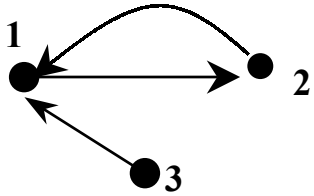

а) ориентированный мультиграф



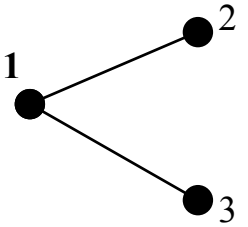

б) неориентированный мультиграф

**рис.1.1.2.**

**3. Графы** (нет петель и кратных ребер/дуг: дуги (1,2) и (2,1) в орграфе считаются разными).




а) ориентированный граф




б) неориентированный граф

**рис.1.1.3.**

**Пример 1.1.1. а)** Ориентированный псевдограф на рис.1.1.1.а) аналитически задается следующим образом: G=(V,E), где

V={1,2,3}, E={(1,1), (1,2), (3,1)}.

**б)** Неориентированный псевдограф на рис.1.1.1.а) аналитически задается следующим образом: G=(V,E), где

V={1,2,3}, E={(1,1), (1,2), (1,3)} или V={1,2,3}, E={(1,1), (2,1), (1,3)}, т.к. в неупорядоченной паре (2,1)=(1,2).

**г)** Ориентированный мультиграф на рис.1.1.2.а) аналитически задается следующим образом: G=(V,E), где

V={1,2,3}, E={(1,1), (1,2), (1,2), (3,1)}.

## §1.2. Смежность, инцидентность, степени.

Две **вершины** в графе (орграфе) G называются **смежными,** если существует ребро(дуга), которая их соединяет. Пусть v, w вершины, а e=(v,w) – ребро (дуга) их соединяющая. Тогда вершина v и ребро (дуга) e являются **инцидентными,** вершина w и ребро (дуга) e также являются инцидентными. Два **ребра** инцидентные одной вершине называются **смежными.**

**Пример 1.2.1.** Рассмотрим неориентированный граф на рис.1.1.1.в).

- Вершины 1,2 –смежные, а вершины 2,3 не являются смежными.
- Вершина 1 инцидентна ребру (1,2). Ребро (2,3) инцидентно вершине 2. Вершина 1 не инцидентна ребру (3,2).
- Ребра (1,2) и (3,2) смежные, т.к. у них есть общая вершина.

**Пример 1.2.2.** Рассмотрим ориентированный граф на рис.1.1.1.в).

- Вершины 1,2 –смежные, а вершины 2,3 не являются смежными.
- Вершина 1 инцидентна дуге (1,2). Дуга (2,3) инцидентна вершине 2. Вершина 1 не инцидентна дуге (3,2).

**Степенью** вершины v графа G называется число $\delta$(v) ребер инцидентных v. Вершина графа, имеющая степень 0, называется **изолированной,** а вершина степени 1 называется **висячей.**

**Полустепенью исхода** вершины v орграфа G называется число $\delta^-$(v) дуг исходящих из вершины v (т.е. v является началом этих дуг). **Полустепенью захода** вершины v орграфа G называется число $\delta^+$(v) дуг входящих в вершину v (т.е. v является концом этих дуг).

**Замечание.** В случае неориентированного псевдографа вклад каждой петли при расчете степени инцидентной вершины равен 2.

**Пример 1.2.3.** Рассмотрим ориентированный псевдограф, изображенный на рис.1.1.1.а): $\delta^-(1)=2$, $\delta^+(1)=2$, $\delta^-(2)=0$, $\delta^+(2)=1$.

**Пример 1.2.4.** Рассмотрим неориентированный псевдограф, изображенный на рис.1.1.1.а: $\delta(1)=4$, $\delta(2)=1$, $\delta(3)=1$.

## §1.3. Маршруты, пути, циклы.

Пусть G=(V,E) граф с **p** вершинами и **q** ребрами, т.е. $V=\{v_1,v_2,\ldots,v_p\}$, $E=\{e_1,e_2,\ldots,e_q\}$.

**Маршрутом,** соединяющим вершины $v_{n_0}, v_{n_i}$ в графе G, называется чередующаяся последовательность вершин и ребер, в которой любые два соседних элемента инцидентны: $v_{n_0} e_{n_1} v_{n_1} e_{n_2} v_{n_2} \dots e_{n_i} v_{n_i}$. Вершины $v_{n_0}, v_{n_i}$ называются начальной и конечной вершинами маршрута, остальные называются внутренними. Число ребер в маршруте называется **длиной маршрута.**

**Замечание_1.** Это определение подходит также для псевдо-, мульти- и орграфов. Для «обычного» графа достаточно указать только последовательность вершин или только последовательность ребер, т.к. оставшиеся компоненты в этом случае восстанавливаются однозначно.

Если начальная и конечная вершины маршрута совпадают $(v_{n_0} = v_{n_i})$, то маршрут называется **замкнутым**, иначе – открытым.

Если все ребра в маршруте различны, то маршрут называется **цепью.** Если все вершины (а значит, и ребра) в маршруте различны, то маршрут называется **простой цепью**. В цепи начальная и конечная вершины называются концами цепи. Говорят, что цепь с концами u и v соединяет вершины u и v. Цепь, соединяющая вершины u и v, обозначается <u,v>. Очевидно, что если есть цепь, соединяющая вершины u и v, то есть и простая цепь, соединяющая эти вершины. Замкнутая цепь называется **циклом;** замкнутая простая цепь называется **простым циклом.** Граф без циклов называется ациклическим.

**Замечание 2.** Часто в записи замкнутого маршрута перечисляют только вершины входящие в маршрут, тем самым, подчеркивая, что в этом случае неважно какую вершину считать начальной

Для орграфов цепь называется **путем,** а цикл — **контуром.** Двигать в орграфе можно только в направлении «стрелок»

**Пример_1.3.1.** В графе, диаграмма которого приведена на рис.1.3.1.:

1. $v_1 v_3 v_1 v_4$ — маршрут длины 3, но не цепь;
2. $v_1 v_3 v_5 v_2 v_3 v_4$ — цепь длины 5, но не простая цепь;
3. $v_1 v_4 v_3 v_2 v_5$ — простая цепь длины 4;
4. $v_1 v_3 v_5 v_2 v_3 v_4 v_1$ — цикл длины 6, но не простой цикл;
5. $v_1 v_3 v_4$ — простой цикл длины 3.

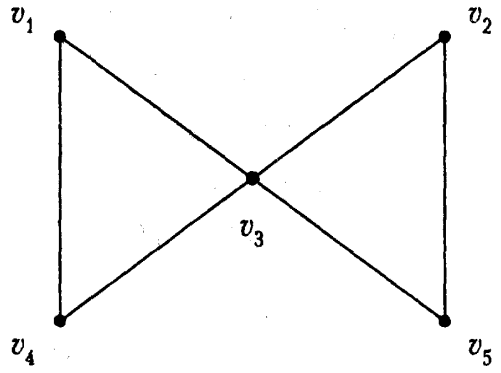



**рис.1.3.1.**

**Определения.**

1. **Расстоянием** между вершинами u и v называется длина кратчайшей цепи <u,v>.
2. Наименьшее из расстояний между вершинами графа называется **радиусом**, а наибольшее – **диаметром.**
3. Множество вершин находящихся на одинаковом расстоянии от вершины v называется **ярусом** вершины v.

## §1.4. Изоморфизм графов.

Говорят, что два графа G1(V1,E1) и G(V2,E2) **изоморфны** (обозначается G1~G2), если существует биекция h:V1→V2, сохраняющая смежность:

(a,b)∈E1-ребро графа G1 ⇔ (h(a),h(b))∈E2-ребро графа G2.

**Утверждение_1.4.1.** Изоморфизм графов есть отношение эквивалентности.

Действительно, изоморфизм обладает всеми необходимыми свойствами:

**1.** рефлексивность: G~G, где требуемую биекцию устанавливает тождественная функция;

**2.** симметричность: если G1~G2 с биекцией h, то G2~G1 с биекцией $h^{-1}$;

**3.** транзитивность: если G1~G2 с биекцией h и G2~G3 с биекцией g, то G1~G3 с биекцией h∘g, являющейся композицией h и g.

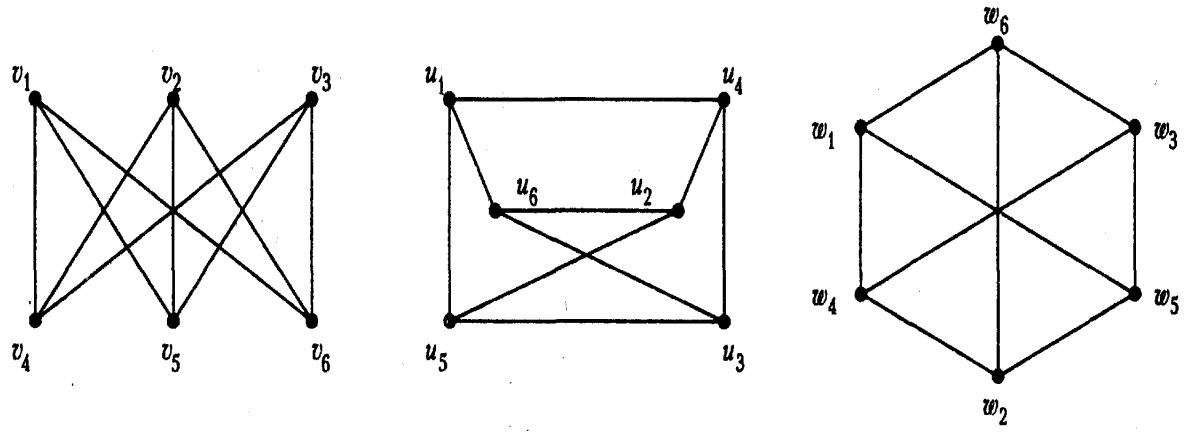
**Замечание_1.4.1.** Из определения изоморфизма графов следует, что изоморфные графы (орграфы) отличаются лишь обозначением вершин и “их расположением на плоскости”.

**Замечание_1.4.2.** Графы рассматриваются с точностью до изоморфизма, то есть рассматриваются классы эквивалентности по отношению изоморфизма.

**Утверждение_1.4.2.** У изоморфных графов одинаково

- число вершин;
- число ребер;
- число вершин одинаковой степени (полустепени).

**Пример_1.4.1.** Три внешне различные диаграммы, приведенные на рис.1.4.1., являются диаграммами одного и того же графа $K_{3,3}$. Действительно, если вершины графа G1 переобозначить так – $v_1$ на $u_1$, $v_2$ на $u_2$,..., $v_6$ на $u_6$, затем расположить в местах, где расположены вершины графа G2, потом восстановить связи (ребра) как в первом графе, то получится граф G2.




**а) граф G1** **б) граф G2** **в) граф G3**

**Рис.1.5.1.**

**Замечание_1.4.3.** Количество вершин, ребер и количество смежных вершин для каждой вершины не определяют граф. На рис.1.5.2. представлены диаграммы графов, у которых указанные инварианты совпадают, но графы при этом не изоморфны.

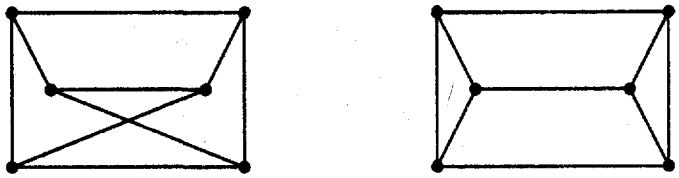


Рис.1.5.2.

## §1.5. Представление графов в ЭВМ.

Следует еще раз подчеркнуть, что конструирование структур данных для представления в программе объектов математической модели — это основа искусства практического программирования. Мы приводим четыре различных базовых представления графов. Выбор наилучшего представления определяется требованиями конкретной задачи. Более того, при решении конкретных задач используются, как правило, некоторые комбинации или модификации указанных представлений, общее число которых необозримо. Но все они так или иначе основаны на тех базовых идеях, которые описаны в этом разделе.

### А) Матрица смежности.

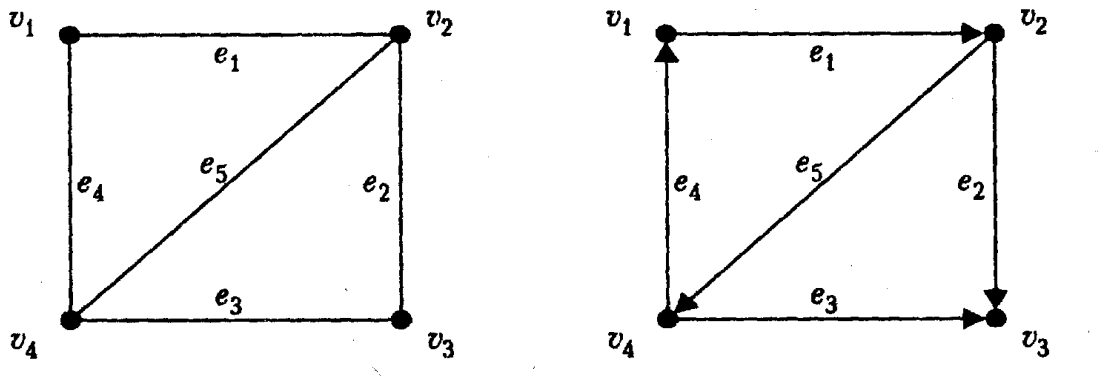
Пусть G=(V,E) граф (орграф) с p-вершинами и q ребрами, т.е. $V=\{v_1,v_2,\ldots,v_p\}$, $E=\{e_1,e_2,\ldots,e_q\}$.

**Матрицей смежности** графа (орграфа) G называется квадратная матрица A(G) порядка p элементы которой равны

$$a_{ij} = \begin{cases} 1, & \text{если } (v_i, v_j) \in E \\ 0, & \text{если } (v_i, v_j) \notin E. \end{cases}$$

**Замечание_1.** Матрица смежности **A:array**[1..p,1..p]**of**0..1, отражает смежность вершин.

**Пример 1.5.1.** Граф G и орграф D изображены на рис.1.5.1. Найдем их матрицы смежности.




а) граф G б) орграф D

**рис.1.5.1.**

A(G)=

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|
| | 0 | 1 | 0 | 1 |
| | 1 | 0 | 1 | 1 |
| | 0 | 1 | 0 | 1 |
| | 1 | 1 | 1 | 0 |

A(D)=

| $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |

**Замечание_2.** Матрица смежности для неоринтированного графа является симметричной.

**Замечание_3.** Матрицу смежности можно ввести для мультиграфов: элемент $a_{ij}$ матрицы A равен числу ребер (дуг) идущих из вершины $v_i$ к вершине $v_j$.

**Б) Матрица инцидентности.**

Пусть G=(V,E) граф (орграф) с p-вершинами и q ребрами, т.е. $V=\{v_1,v_2,\ldots,v_p\}$, $E=\{e_1,e_2,\ldots,e_q\}$.

**Матрицей инцидентности** B(G) орграфа G называется матрица порядка p×q, элементы которой вычисляются следующим образом:

$$b_{ij} = \begin{cases} 1, & \text{если } v_i \text{ является концом дуги } e_j \\ -1, & \text{если } v_i \text{ является началом дуги } e_j \\ 0, & \text{если } v_i \text{ не инцидентна дуге } e_j. \end{cases}$$

**Матрицей инцидентности** B(G) графа G называется матрица порядка p×q, элементы которой вычисляются следующим образом:

$$b_{ij} = \begin{cases} 1, & \text{если } v_i \text{ инцидентна ребру } e_j \\ 0, & \text{если } v_i \text{ не инцидентна ребру } e_j. \end{cases}$$

**Замечание_4.** В каждом столбце матрицы инцидентности только два элемента не равны нулю (они равны ±1).

**Замечание_5.** Матрицу инцидентности можно ввести для псевдографов: например, элемент $a_{ij}$ матрицы A равен +1, если вершина $v_i$ является началом и концом дуги $e_j$, все остальные элементы в этом столбце равны нулю.

**Пример 1.5.2.** Граф G и орграф D изображены на рис.1.5.1. Найдем их матрицы инцидентности.

A(G)=

| | $e_1$ | $e_2$ | $e_3$ | $e_4$ | $e_5$ |
|---|---|---|---|---|---|
| $v_1$ | 1 | 0 | 0 | 1 | 0 |
| $v_2$ | 1 | 1 | 0 | 0 | 1 |
| $v_3$ | 0 | 1 | 1 | 0 | 0 |
| $v_4$ | 0 | 0 | 1 | 1 | 1 |

A(D)=

| $e_1$ | $e_2$ | $e_3$ | $e_4$ | $e_5$ |
|---|---|---|---|---|
| -1 | 0 | 0 | 1 | 0 |
| 1 | -1 | 0 | 0 | -1 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | -1 | -1 | 1 |

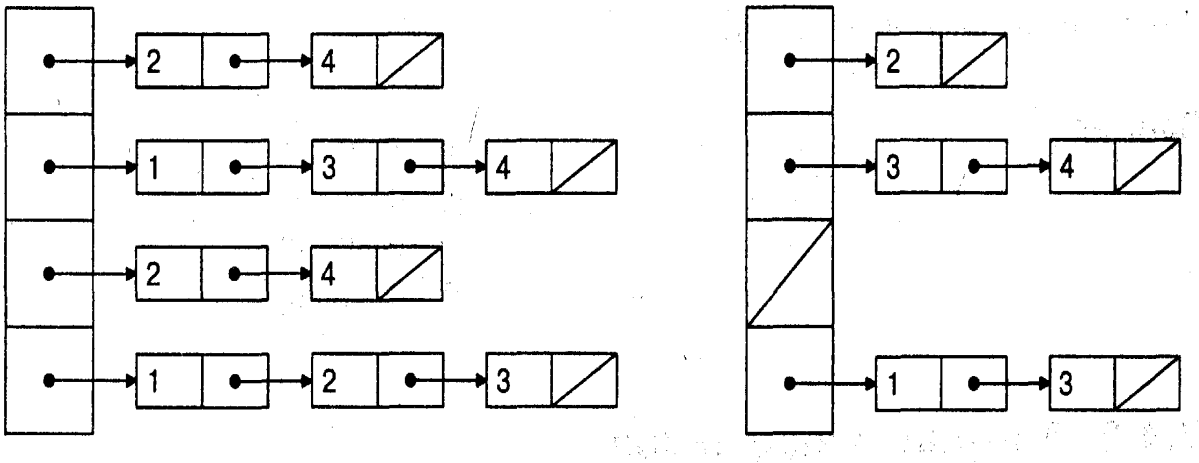
**В) Списки смежности.**

Представление графа с помощью списочной структуры, отражающей смежность вершин и состоящей из массива указателей **Г**:**array**_[1..p]_of ↑ N на списки смежных вершин называется **списком смежности.** Элемент списка представлен структурой N:

```
N:record_v:1..p;_
          n: ↑ N_
  endrecord.
```

**Пример 1.5.1.** Граф G и орграф D изображены на рис.1.5.1. Найдем их списки смежности.

1. Способ (см. рис.1.5.2)




**а) граф G** **б) орграф D**

**рис.1.5.2.**

2. Способ.

Граф G

| v | Γ(v) |
|---|---|
| 1 | 2,4 |
| 2 | 1,3,4 |
| 3 | 2,4 |
| 4 | 1,2,3 |

Орграф D

| v | Γ(v) |
|---|---|
| 1 | 2 |
| 2 | 3,4 |
| 3 | ∅ |
| 4 | 1,3 |

## §1.6. Полные графы и двудольные графы.

Граф, состоящий из одной вершины, называется **тривиальным.**

Граф, в котором каждая пара вершин смежна, называется **полным.** Другими словами, в полном графе каждая вершина, соединяется ребрами со всеми другими вершинами. Полный граф с **p** вершинами обозначается $\mathbf{K_p}$, он имеет максимально возможное число ребер (см. рис.1.6.1).

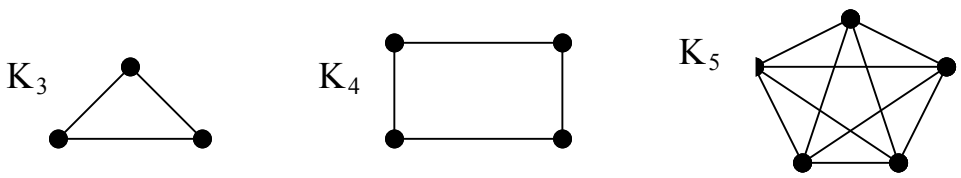



**Рис.1.6.1. Полные графы $K_p$.**

**Двудольный граф** (или биграф, или четный граф) — это граф G(V,E), такой что множество вершин V разбито на два непересекающихся множества V1 и V2 (т.е. V1∪V2=V, V1∩V2=∅), причем всякое ребро из E инцидентно вершине из V1 и вершине из V2

(то есть ребро соединяет вершину из V1 с вершиной из V2). Множества V1 и V2 называются долями двудольного графа. Если в двудольном графе, каждая вершина из V1 соединяется ребрами со всеми вершинами из V2, то граф называется **полным двудольным графом**. Если |V1|=n, и |V2|=m, то полный двудольный граф обозначается $K_{n,m}$.

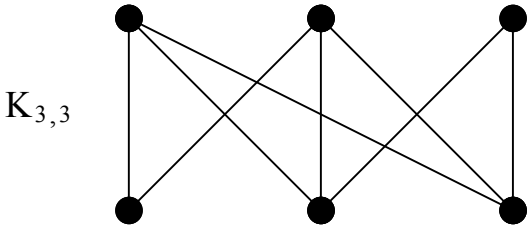



**Рис.1.6.2. Полный двудольный граф $K_{3,3}$**

## §1.7. Операции с графами.

Очевидно, что каждый неориентированный граф можно сделать ориентированным, заменив каждое ребро парой дуг идущих в противоположных направлениях (см. рис.1.7.1.). В связи с этим, можно рассматривать «смешанные» графы, которые могут содержать и ребра и дуги. Очевидно, что графы и орграфы являются частными случаями «смешанных» графов. В том параграфе под словом «граф» понимается «смешанный» граф, а под словом «ребро», понимается либо ребро, которое обозначаться будет [u,v], либо дуга, которая обозначаться будет (u,v).

**рис.1.7.1. Замена ребра на две дуги.**

Напомним, что граф состоит из пары множеств, для которых теоретико-множественные операции, рассмотрены в §1.3. раздела 1.

Граф G1=(V1,E1) называется **подграфом** графа G=(V,E), если $V1 \subseteq V$ и $E1 \subseteq E$. Подграф G1 называется **собственным** подграфом графа G, если G1 не совпадает с G.

**Определения.** Пусть даны два графа G1=(V1,E1) и G2=(V2,E2).

1. **Объединением** двух графов G1 и G2 называется граф G=(V,E), где $V=V1 \cup V2$, $E=E1 \cup E2$. Обозначение: $G=G1 \cup G2$.
2. **Пересечением** двух графов G1 и G2 называется граф G=(V,E), где $V=V1 \cap V2$, $E=E1 \cap E2$. Обозначение: $G=G1 \cap G2$.
3. **Симметрической разность** (или **суммой по модулю 2**) двух графов

G1 и G2 называется граф G=(V,E), где V=V1∪V2, E=E1⊕E2 (E=E1ΔE2). Обозначение: G=G1⊕G2.

4. Пусть задан граф G=(V,E) с |V|=n (т.е. с n вершинами) и без петель. На множестве вершин V построим полный граф $K_n$=(V,M). **Дополнением** графа без петель G(V,E) называется граф $\overline{G} = (V, M \setminus E)$.

**Пример 1.7.1.** Пусть даны графы $G_1$=(V1,E1) и $G_2$=(V2,E2), где V1={$a_1$,$a_2$,$a_3$}, E1={[$a_1$,$a_2$],($a_2$,$a_3$)}, V2={$a_1$,$a_2$,$a_4$}, E2={($a_1$,$a_2$),($a_4$,$a_1$)} (см. рис.1.7.2.)

1. Найдем G=$G_1$∪$G_2$=(V,E), где V={$a_1$,$a_2$,$a_3$}∪{$a_1$,$a_2$,$a_4$}={$a_1$,$a_2$,$a_3$,$a_4$}, E={[$a_1$,$a_2$],($a_2$,$a_3$)}∪{($a_1$,$a_2$),($a_4$,$a_1$)}={[$a_1$,$a_2$],($a_2$,$a_3$),($a_4$,$a_1$)} (рис.1.7.2).
2. Найдем G=$G_1$∩$G_2$=(V,E), где V={$a_1$,$a_2$,$a_3$}∩{$a_1$,$a_2$,$a_4$}={$a_1$,$a_2$}, E={[$a_1$,$a_2$],($a_2$,$a_3$)}∩{($a_1$,$a_2$),($a_4$,$a_1$)}={($a_1$,$a_2$)} (рис.1.7.2).
3. Найдем G=$G_1$⊕$G_2$=(V,E), где V={$a_1$,$a_2$,$a_3$}∪{$a_1$,$a_2$,$a_4$}={$a_1$,$a_2$,$a_3$,$a_4$}, E={[$a_1$,$a_2$],($a_2$,$a_3$)}⊕{($a_1$,$a_2$),($a_4$,$a_1$)}={($a_2$,$a_1$),($a_2$,$a_3$),($a_4$,$a_1$)} (рис.1.7.2).

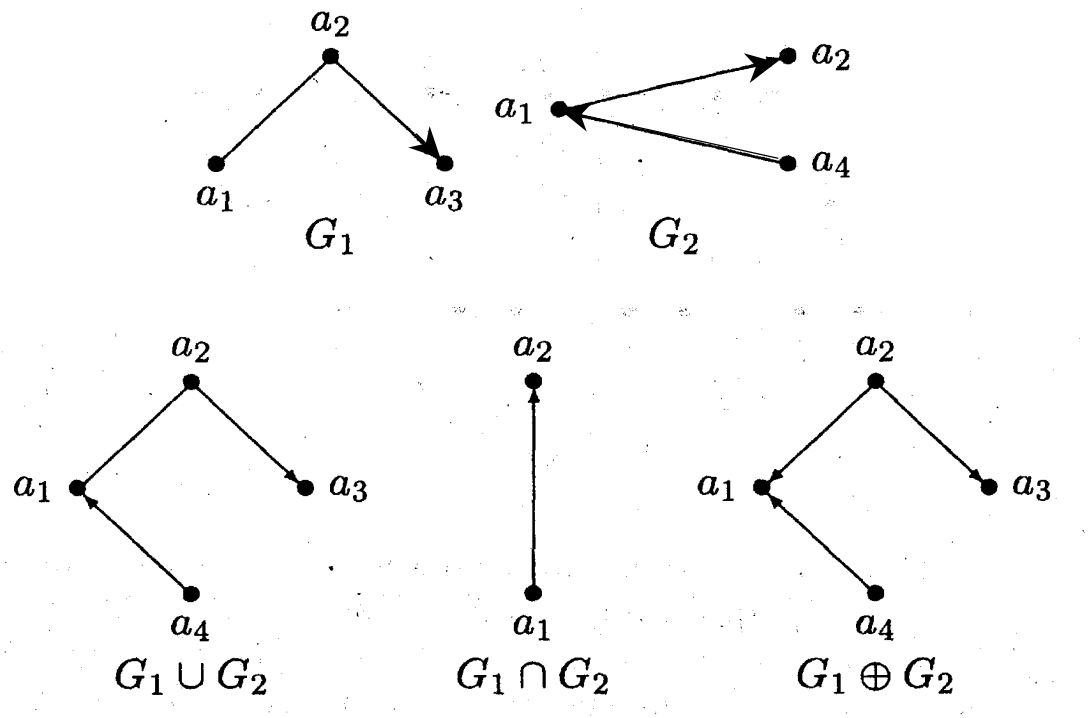



**Рис.1.7.2. Операции над графами.**

**Пример_1.7.3.** Дополнением графа G, изображенного на рис.1.7.3., является граф $\overline{G}$, изображенный на рис.1.7.4.

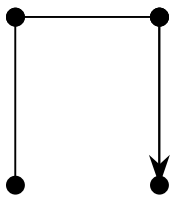


**Рис.1.7.3.**

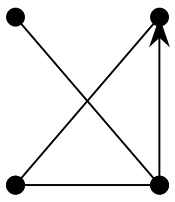


**рис.1.7.4.**

# Глава 2. Связность.

В этой главе все понятия и рассуждения справедливы для псевдо-, мульти-, и просто графов, т.е. для графов в широком смысле.

## §2.1. Связность в неориентированных графах.

Говорят, что две вершины u, v в графе G связаны отношением **достижимости,** если существует соединяющая их цепь <u,v>. Граф, в котором все вершины достижимы, называется **связным.** Тривиальный граф, состоящий из изолированной вершины, по определению считается связным.

**Утверждение 2.1.1.** Отношение достижимости вершин является отношением эквивалентности.

**Компонентой связности** графа G называется его связный подграф, который не является собственным подграфом никакого другого связного подграфа графа G.

Из утверждения 2.1.1 следует, что классы эквивалентности по отношению достижимости, являются разбиением множества вершин графа G, на подмножества вершин входящих в одну компоненту связности. Для построения компоненты связности достаточно взять один класс эквивалентности (вершин) и перенести на множество его вершин связи (ребра) из графа G.

Число компонент связности графа G обозначим c(G). Граф G является связным тогда и только тогда, когда c(G)=1. Если c(G)>1, то G — несвязный граф. Граф, состоящий только из изолированных вершин, называется вполне несвязным.

**Пример 2.1.1.** Граф, показанный на рис.2.1.1., имеет две компоненты связности с множеством вершин V1={1,2,3,4} и V2={5,6,7}.

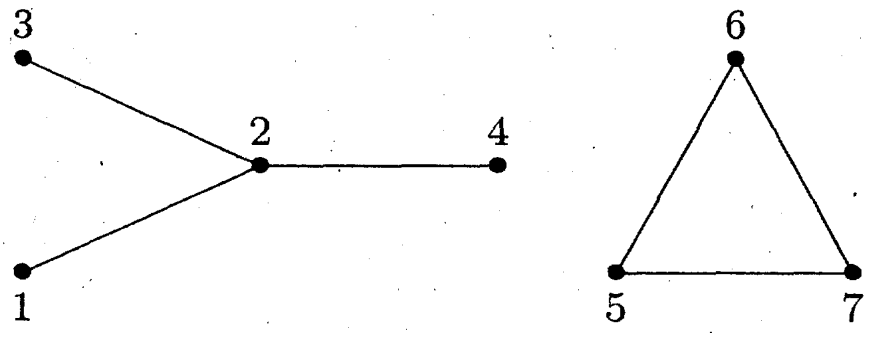



**Рис.2.1.1.**

## §2.2. Связность в орграфах.

Пусть G=(V,E) ориентированный граф (орграф).

**А) Сильная связность.**

Говорят, что две вершины **u, v** в орграфе G связаны отношением **двусторонней достижимости**, если существует маршрут из **u** в **v**, а также маршрут из **v** в **u**. Орграф, в котором любые две вершины двусторонне достижимы, называется **сильно связным.** Тривиальный граф, состоящий из изолированной вершины, по определению считается сильно связным.

**Утверждение 2.2.1.** Отношение двусторонней достижимости вершин является отношением эквивалентности.

**Компонентой сильной связности** орграфа G называется его сильно связный подграф, который не является собственным подграфом никакого другого сильно связного подграфа орграфа G.

Из утверждения 2.2.1 следует, что классы эквивалентности по отношению двусторонней достижимости, являются разбиением множества вершин орграфа G на подмножества вершин, входящих в одну компоненту сильной связности. Для построения компоненты сильной связности достаточно взять один класс эквивалентности и перенести на множество его вершин связи (дуги) из орграфа G.

Число компонент сильной связности орграфа G обозначим k(G). Орграф G является сильно связным тогда и только тогда, когда k(G)=1. Если k(G)>1, то G — не является сильно связным орграфом.

**Пример 2.2.1.** Граф, показанный на рис.2.2.1. не является сильно связным, т.к. имеет три компоненты сильной связности с множеством вершин V1={1,2,3}, V2={4} и V3={5,6,7}. Сами компоненты сильной связности изображены на рис.2.2.2.

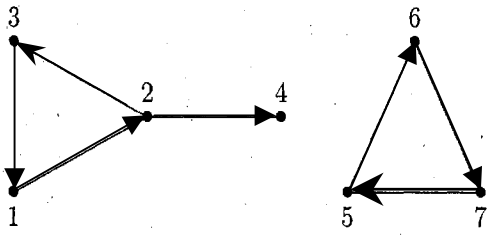



**Рис.2.2.1.**

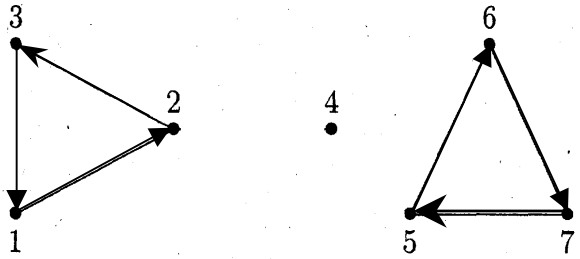



**Рис.2.2.2.**

**Б) Односторонняя связность.**

Орграф G называется **односторонне связанным,** если для любых двух вершин **u, v** существует маршрут хотя бы в одну сторону, т.е. либо маршрут из **u** в **v**, либо маршрут из **v** в **u**.

Очевидно, что если орграф является сильно связным, то он будет и односторонне связным.

**Пример 2.2.2.** Граф, показанный на рис.2.2.1., не является односторонне связанным, т.к. скажем из вершины 1 нет пути в вершину 5 и наоборот, из вершины 5 нет пути в вершину 1.

**Пример 2.2.3.** Граф на рис.2.2.3., не является сильно связным, но является односторонне связным.

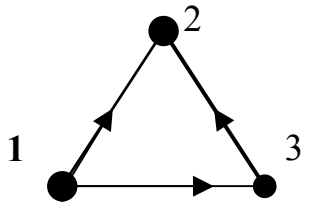



**Рис.2.2.3.**

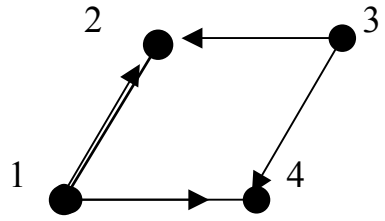



**Рис.2.2.4.**

**Пример 2.2.4.** Граф на рис.2.2.4., не является односторонне связным, т.к. для вершин 1 и 3 не существует пути <1,3> и не существует пути <3,1>.

**В) Слабая связность.**

Псевдографом ассоциированным с орграфом G=(V,E) называется псевдограф G1=(V1,E1), в котором E1 получается из E заменой всех дуг (упорядоченных пар) на ребра (неупорядоченные пары).

Орграф G=(V,E) называется **слабо связным,** если ассоциированный с ним неориентированный граф является связным.

**Пример 2.2.5.** Граф на рис.2.2.4., является слабо связным.

## §2.3. Нахождение компонент связности на ЭВМ.

Пусть G=(V,E) граф (орграф) с **n** вершинами и **q** ребрами, т.е. $V=\{v_1,v_2,\ldots,v_n\}$, $E=\{e_1,e_2,\ldots,e_q\}$.

Следующая теорема позволяет по матрице смежности A(G) исследовать маршруты данного графа (орграфа) G.

**Теорема 2.3.1.** Если A=A(G) — матрица смежности графа (орграфа) G, то $a_{ij}$ элемент матрицы $A^k$ есть число маршрутов из $v_i$ в $v_j$ длины k.

**Следствие 2.3.1.** В n вершинном графе (орграфе) G тогда и только тогда существует маршрут из $v_i$ в $v_j$ ($i\neq j$), когда (i,j)-й элемент матрицы $A+A^2+\ldots+A^{n-1}$ не равен нулю.

**Следствие 2.3.2.** В n вершинном графе (орграфе) G тогда и только тогда существует цикл; содержащий вершину $a_i$, когда (i,i)-й элемент матрицы $A+A^2+\ldots+A^{n-1}+A^n$ не равен нулю.

Образуем из матрицы $B=E+A+A^2+\ldots+A^{n-1}+A^n$ матрицу $C=(c_{ij})$ порядка n по следующему правилу:

$$c_{ij} = \begin{cases} 1, & \text{если } b_{ij} \neq 0 \\ 0, & \text{если } b_{ij} = 0 \end{cases}, \quad \text{где } b_{ij} \text{ элементы матрицы B.}$$

Матрица C называется **матрицей связности**, если G •— неориентированный граф, и **матрицей достижимости**, если G — орграф. В графе G тогда и только тогда существует маршрут из $v_i$ в $v_j$ ($i \neq j$) когда $c_{ij}=1$. Таким образом, в матрице C содержится информация о существовании связей между различными элементами графа посредством маршрутов. Если G — связный неориентированный граф, то все элементы матрицы связности C равны единице. В общем случае матрица связности неориентированного графа является матрицей отношения эквивалентности, соответствующего разбиению множества вершин графа на компоненты связности.

Пусть $C^t$ транспонированная матрица C. В случае неориентированного графа $C=C^t$, и является матрицей связности. В случае ориентированного графа $C^t$ называется **матрицей контрдостижимости,** т.к. если (i,j)-й элемент матрицы C показывает наличие пути из $v_i$ в $v_j$, то (i,j)-й элемент матрицы $C^t$ показывает наличие обратного пути из $v_j$ в $v_i$.

Матрицы достижимости и контрдостижимости можно использовать для нахождения сильных компонент орграфа. Рассмотрим матрицу $S=C*C^t$, где операция * означает поэлементное произведение матриц C и $C^t$. Элемент $s_{ij}$ матрицы S равен 1 тогда и только тогда, когда вершины $v_i$ и $v_j$ взаимно достижимы, т. е. $v_j$ достижима из $v_i$ и $v_i$ достижима из $v_j$. Следовательно, сильная компонента, содержащая вершину $v_i$ состоит из элементов $v_j$, для которых $s_{ij}=1$.

**Пример 2.3.1.** Найти компоненту сильной связности орграфа G, изображенного на рис.2.3.1., содержащую вершину 2

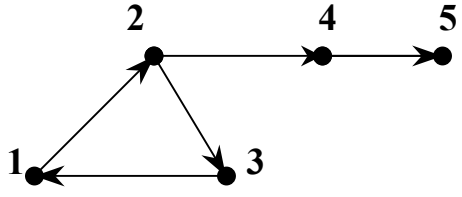



**Рис.2.3.1.**

Решение. Матрицы достижимости C и контрдостижимости $C^t$ орграфа, изображенного на рис.2.3.1., имеют вид

$$\mathbf{C}=\begin{Vmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{Vmatrix} \qquad \mathbf{C}^{\mathbf{t}}=\begin{vmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{vmatrix}$$

$$\mathbf{S}=\begin{Vmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{Vmatrix}$$

Из второй строки матрицы S находим, что сильная компонента, содержащая вершину 2, состоит из вершин {1,2,3}.

## Глава 3. Обходы графов.

Обход графа — это некоторое систематическое перечисление его вершин (и/или ребер). Наибольший интерес представляют обходы, использующие локальную информацию (списки смежности). Среди всех обходов наиболее известны поиск в ширину и в глубину. Алгоритмы поиска в ширину и в глубину лежат в основе многих конкретных алгоритмов на графах.

**Алгоритм 2.3.1. Поиск в ширину и в глубину.**

**Вход:** граф G{V,E), представленный **списками смежности Г.**
**Выход:** последовательность вершин обхода.

```
for v∈V do x[v]:=0 { вначале все вершины не отмечены }
end for
select v∈V              { начало обхода — произвольная вершина }
v→T                     { помещаем v в структуру данных T}
x[v]:=1                 {... и отмечаем вершину v }
repeat
   u←T          {извлекаем вершину из структуры данных T: T:=T\{u}...}
   yield u {и возвращаем ее в качестве очередной пройденной вершины }
      for w∈Γ(u) do
         if x[w]=0 then
            w→T         {помещаем w в структуру данных T ... }
            x[w]:=1      { ... и отмечаем вершину w }
         end if
      end for
```

**until** T=∅

Если T – это стек (LIFO — Last In First Out), то обход называется **поиском в глубину**. Если T — это очередь (FIFO — First In First Out), то обход называется **поиском в ширину.**

**Пример_2.3.1.** В следующей таблице показаны протоколы поиска в глубину и в ширину для графа, диаграмма которого приведена на рис. 2.3.1. Слева в таблице протокол поиска в глубину, а справа — в ширину. Предполагается, что начальной является вершина 1.

| в глубину | | в ширину | |
|---|---|---|---|
| u | T | u | T |
| 1 | 2.4 | 1 | 2,4 |
| 4 | 2,3 | 2 | 4,3 |
| 3 | 2 | 4 | 3 |
| 2 | ∅ | 3 | ∅ |

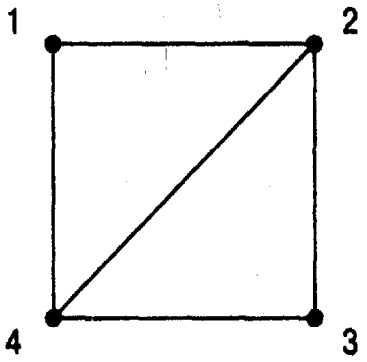



**Рисунок 2.3.1.**

**Теорема 2.3.1.** Если граф G связен (и конечен), то поиск в ширину и поиск в глубину обойдут все вершины по одному разу.

Доказательство. 1. Единственность обхода вершины. Обходятся только вершины, попавшие в T. В T попадают только неотмеченные вершины. При попадании в T вершина отмечается. Следовательно, любая вершина будет обойдена не более одного раза.

2. Завершаемость алгоритма. Всего в T может попасть не более p вершин. На каждом шаге одна вершина удаляется из T. Следовательно, алгоритм завершит работу не более чем через p шагов.

3. Обход всех вершин. От противного. Пусть алгоритм закончил работу, и вершина w не обойдена. Значит, w не попала в T. Следовательно, она не была отмечена. Отсюда следует, что все вершины, смежные с w, не были обойдены и отмечены. Аналогично, любые вершины, смежные с неотмеченными вершинами, сами не отмечены (после завершения алгоритма). Но G связен, значит, существует путь из v в w. Следовательно, вершина v не отмечена. Но она была отмечена на первом шаге. Противоречие.

**Замечание.** При поиске в ширину, вершины обходятся в порядке возрастания расстояния от начальной вершины (отсюда и название – поиск в ширину).

# Глава 4. Графы и бинарные отношения.

Целью этой главы является установление связи теории графов с другими разделами дискретной математики.

## §4.1. Графы и отношения

Любой орграф G(V,E) с петлями, но без кратных дуг, задает бинарное отношение E на множестве V, и обратно. А именно, пара элементов принадлежит отношению $(a,b)\in E\subset V\times V$ тогда и только тогда, когда в графе G есть дуга (a,b).

Полный граф соответствует универсальному отношению. Граф (неориентированный) соответствует симметричному отношению. Дополнение графов есть дополнение отношений. Изменение направления всех дуг соответствует обратному отношению и т. д.

Таким образом, имеется полная аналогия между орграфами и бинарными отношениями — фактически, это один и тот же класс объектов, только описанный разными средствами. Отношения (в частности, функции), являются базовыми средствами для построения подавляющего большинства математических моделей, используемых при решении практических задач. С другой стороны, графы допускают наглядное представление в виде диаграмм. Это обстоятельство объясняет широкое использование диаграмм различного вида (которые суть представления графов или родственных объектов) при кодировании и особенно при проектировании в программировании.

## §4.2. Достижимость и частичное упорядочение

В качестве примера связи между орграфами и бинарными отношениями рассмотрим отношения частичного порядка с точки зрения теории графов.

Вершина u в орграфе G(V,E) достижима из вершины v, если существует путь из v в u. Путь из v в и обозначим <u,v>.

Отношение достижимости можно представить матрицей T:**array**[1..p,1..p] **of** 0..1, где T[i,j]=1, если вершина v(j) достижима из вершины v(i), и T[i,j] = 0, если не достижима.

Рассмотрим отношение строгого порядка >, которое характеризуется следующими аксиомами:

- антирефлексивность: для любого $v\in V$ неверно, что v>v;
- антисмметричность: для любых $u,v\in V$ неверно, что u>v и v>u;
- транзитивность: для любых $u,v,w\in V$ таких, что u>v и v>w следует, что u>w;

Отношению строгого порядка >, заданному на множестве V, можно сопоставить граф G(V,E), в котором $u>v \Leftrightarrow (u,v)\in E$.

**Теорема 4.2.1.** Если отношение E есть отношение строгого порядка, то орграф G(V, E) не имеет контуров.

**Доказательство.** От противного. Пусть в G есть контур. Рассмотрим любую дугу (a,b) в этом контуре. Тогда имеем a>b, но b>a по транзитивности, что противоречит строгости упорядочения.

**Теорема 4.2.2.** Если орграф G(V,E) не имеет контуров, то достижимость есть отношение строгого порядка.

Доказательство. 1. Нет циклов, следовательно, нет петель, следовательно, достижимость антирефлексивна.

2. Если существуют пути из **v** в **w** и из **w** в **u**, то существует и путь из **v** в **u**. Следовательно, достижимость транзитивна.

3. Пусть достижимость — это не строгое упорядочение. Тогда существуют **u,v** такие, что **u**>**v** и **v**>**u**, то есть существует путь из v в u и из u в v. Следовательно, существует контур <u,v>∪<v,u>, что противоречит условию.

**Теорема 4.2.3.** Если орграф не имеет контуров, то в нем есть вершина, полустепень захода которой равна 0.

Доказательство. От противного. Пусть такой вершины нет, тогда для любой вершины найдется вершина, из которой есть дуга в данную вершину. Следовательно, имеем контур против направления стрелок.

Замечание. Теорема_4.2.3. позволяет найти минимальный элемент в конечном частично упорядоченном множестве, который требуется в алгоритме топологической сортировки (алгоритм 1.11.1). А именно, минимальный элемент — это источник, то есть вершина, которой в матрице смежности соответствует нулевой столбец.

## Глава 5. Нахождение кратчайших маршрутов.

Пусть G= {V,X} — взвешенный граф, имеющий *n* вершин и матрицу весов $W=(w_{ij})$, $w_{ij} \in R$. Опишем некоторые методы нахождения взвешенного расстояния от фиксированной вершины $a_i \in M$ (называемой **источником**) до всех вершин графа G.

Мы будем предполагать, что в G отсутствуют контуры с отрицательным весом, поскольку, двигаясь по такому контуру достаточное число раз, можно получить маршрут, имеющий вес, меньший любого заведомо взятого числа, и тем самым задача нахождения расстояния становится бессмысленной.

Определим **алгоритм Форда-Беллмана.**

1. Выбираем источник – вершину $a_i$ .
2. Зададим строку $D^1= (d_1^1, d_2^1,\ldots,d_n^1)$–i-я строка в матрице W.
3. Последовательно определяем строки $D^s= (d_1^s, d_2^s,\ldots,d_n^s)$, где
   $d_j^s=\min\{ d_j^{s-1}, d_k^{s-1}+ w_{kj}\}_{k=1,2,\ldots n}$, j=1,2,…n.

Нетрудно видеть, что $d_j^s$ – минимальный из весов маршрутов из $a_i$ в $a_j$, состоящих не более чем из s дуг.

4. Искомая строка W-расстояний получается при s = n-1.

Действительно, на этом шаге из весов всех маршрутов из $a_i$ в $a_j$, содержащих не более n-1 дуг, выбирается наименьший, а каждый маршрут более чем с n-1 дугами содержит контур, добавление которого к маршруту не уменьшает w-расстояния, так как мы предположили отсутствие контуров отрицательного веса.

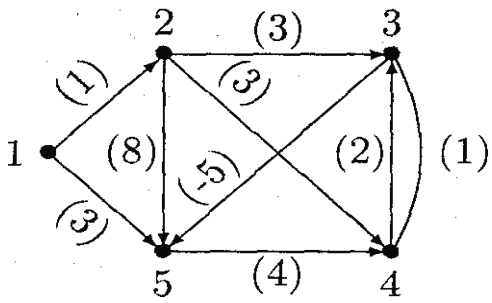
**Замечание.** Работу алгоритма можно завершить на шаге **k**, если $D^k = D^{k+1}$.

**Пример 5.1.** Продемонстрируем работу алгоритма Форда-Беллмана на примере взвешенного графа, показанного на рис. 5.1, с матрицей весов

**Таблица 5.1.**

| | | | | | |
|---|---|---|---|---|---|
| W= | 0 | 1 | ∞ | ∞ | 3 |
| | ∞ | 0 | 3 | 3 | 8 |
| | ∞ | ∞ | 0 | 1 | -5 |
| | ∞ | ∞ | 2 | 0 | ∞ |
| | ∞ | ∞ | ∞ | 4 | 0 |

**рис 5.1.**




В качестве источника выберем вершину 1. Тогда

$D^{(1)}=(0,1,\infty,\infty,3)$, $D^{(2)}=(0,1,4,4,3)$, $D^{(3)}=(0,1,4,4,-1)$, $D^{(4)}=(0,1,4,3,-1)$.

Вектор $D^{(4)}$ определяет минимальные расстояния от источника 1 до вершин 1,2,3,4,5 соответственно.

Отметим, что, зная минимальное расстояние от источника **a**, до всех остальных вершин графа, можно восстановить минимальный маршрут по формуле

$$\rho(a,b)=\rho(a,c)+w_{cb}, \text{ где}$$

$\rho(a,b)$ – минимальное расстояние от вершины a до вершины b.

Так кратчайший (1,4)-маршрут задается последовательностью вершин (1,2,3,5,4).

**Алгоритм Дейкстры** является более эффективным, чем алгоритм Форда-Беллмана, но используется только для взвешенных графов, в которых веса всех дуг неотрицательны.

1. Выбираем источник – вершину $a_i$.
2. Зададим строку $D^1= (d_1^1, d_2^1,\ldots,d_n^1)$–i-я строка в матрице W.

Обозначим $T_1=V\backslash\{a_i\}$.

3. Предположим, что на шаге **s** уже определены строка $D^s=(d_1^s,d_2^s,\ldots,d_n^s)$ и множество вершин $T_s$.

4. Выбираем вершину $a_j\in T_s$ такую, что $d_j^s=\min\{ d_k^s \mid a_k\in T_s\}$. Положим $T_{s+1}=T_s\setminus\{ a_j \}$, $D^{s+1}= (d_1^{s+1},d_2^{s+1},\ldots,d_n^{s+1})$, где $d_k^{s+1}=\min\{d_k^s,d_j^s+w_{jk}\}$, если $a_k\in T_{s+1}$, и $d_k^{s+1}= d_k^s$, если $a_k\notin T_{s+1}$.

5. На шаге s=n-1 образуется строка $D^{n-1}$ w-расстояний между источником $a_i$; и остальными вершинами графа: $d_k^{n-1}= \rho(a_i, a_k)$.

Отметим, что для реализации алгоритма Форда-Беллмана требуется порядка $n^3$ операций, тогда как в алгоритме Дейкстры необходимо выполнить порядка $n^2$ операций.

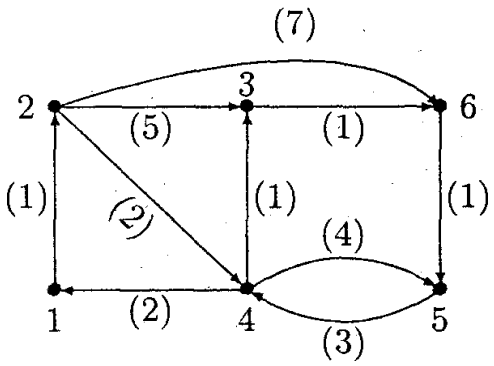
**Пример 5.2.** Рассмотрим взвешенный граф G с матрицей весов W и источником 1 (рис. 5.2).

**Таблица 5.2.**

W=

| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | ∞ | ∞ | ∞ | ∞ |
| ∞ | 0 | 5 | 2 | ∞ | 7 |
| ∞ | ∞ | 0 | ∞ | ∞ | 1 |
| 2 | ∞ | 1 | 0 | 4 | ∞ |
| ∞ | ∞ | ∞ | 3 | 0 | ∞ |
| ∞ | ∞ | ∞ | ∞ | 1 | 0 |

**рис.5.2.**




'По алгоритму Дейкстры находим

$T^1=\{2,3,4,5,6\}$, $D^1=\{0,\underline{1},\infty,\infty,\infty,\infty\}$,

$T^2=\{3,4,5,6\}$, $D^2=\{0,1,6,\underline{3},\infty,\infty\}$,

$T^3=\{3,5,6\}$, $D^3=\{0,1,\underline{4},3,7,8\}$,

$T^4=\{5,6\}$, $D^4=\{0,1,4,3,7,\underline{5}\}$,

$T^5=\{6\}$, $D^5=\{0,1,4,3,6,5\}$.

В $D^s$ подчеркнута величина $d_j^s$, для которой $T_{s+1}=T_s\setminus\{a_j\}$.

Отметим, что если в приведенных алгоритмах ∞ заменить на -∞, min на max, то получим алгоритмы, которые позволяют в графах, не имеющих контуров положительных весов, находить маршруты наибольшей длины.

## Глава 6. Деревья.

Деревья заслуживают отдельного и подробного рассмотрения по двум причинам.

Деревья являются в некотором смысле простейшим классом графов. Для них выполняются многие свойства, которые не всегда выполняются для графов в общем случае. Применительно к деревьям многие доказательства и рассуждения оказываются намного проще.

Выдвигая, какие-то гипотезы при решении задач теории графов, целесообразно сначала их проверять на деревьях.

Деревья являются самым распространенным классом графов, применяемых в программировании, причем в самых разных ситуациях.

## §6.1.Свободные деревья

Вершины в графе часто называют **узлами**.

Связный граф без циклов называется (свободным) **деревом.** Граф состоящий из нескольких деревьев называется **лесом**. Таким образом, компонентами связности леса являются деревья.

**Пример 6.1.1.** На рис.6.1.1. показаны диаграммы всех различных (свободных) деревьев с 5 вершинами, а на рис.6.1.2. — диаграммы всех различных (свободных) деревьев с 6 вершинами.

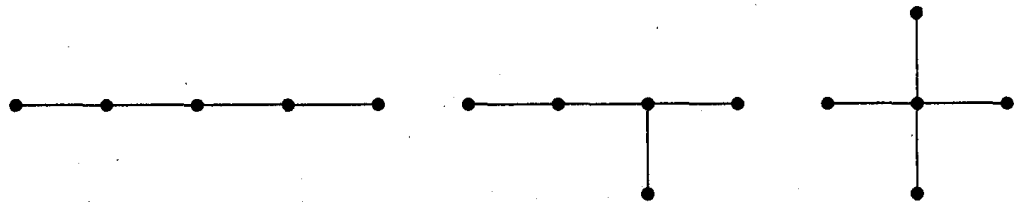


**Рисунок 6.4.1. . Свободные деревья с 5 вершинами**

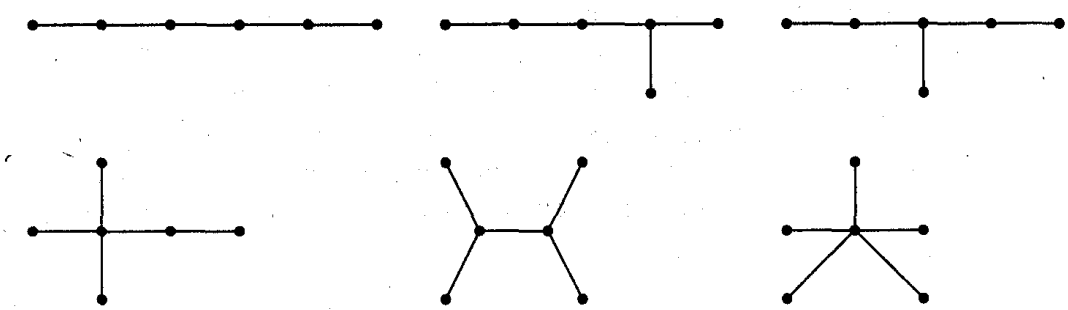


**Рисунок 6.1.5. Свободные деревья с 6 вершинами.**

**Теорема 6.1.1.** Пусть G(V, E) — граф с p вершинами, q ребрами. Следующие утверждения эквивалентны:

1. G — дерево, то есть связный граф без циклов;
2. любые две вершины соединены в G единственной простой цепью;
3. G — связный граф, и любое ребро есть мост, т.е. его удаление делает граф не связным;
4. G — связный граф и q=p-1;
5. G — связный граф и нет простых циклов.

**Следствие.** В любом дереве имеются по крайней мере две висячие вершины.

## §6.2. Ориентированные деревья.

**Ориентированным деревом** (или ордеревом, или корневым деревом) называется орграф со следующими свойствами:

1. существует единственный узел, полустепень захода которого равна 0. Он называется **корнем** ордерева;

2. полустепень захода всех остальных узлов равна 1;

3. каждый узел достижим из корня.

**Пример_6.2.1.** На рис. 6.2.1. приведены диаграммы всех различных ориентированных деревьев с 3 узлами, а на рис. 6.2.2. показаны диаграммы всех различных ориентированных деревьев с 4 узлами.

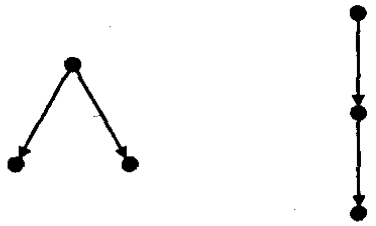


**Рисунок 6.2.1.**

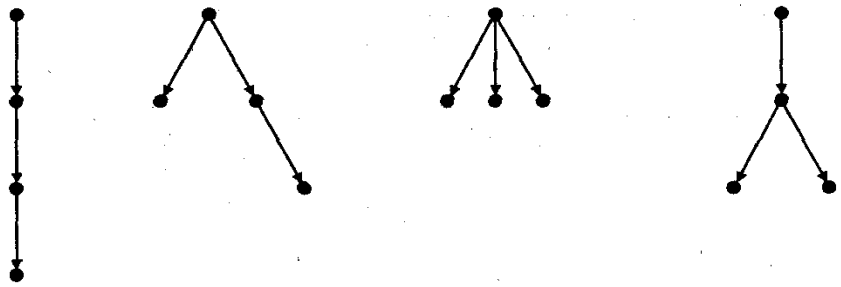


**Рисунок 6.2.2.**

**Теорема 6.2.1.** Ордерево G(V, E) с p вершинами и q дугами, обладает следующими свойствами:

1. q=p-1;

2. если в ордереве отменить ориентацию ребер, то получится свободное дерево;

3. в ордереве нет контуров;

4. для каждого узла существует единственный путь, ведущий в этот узел из корня;

5. подграф, определяемый множеством узлов, достижимых из узла v, является ордеревом с корнем v (это ордерево называется поддеревом узла v);

6. если в свободном дереве любую вершину назначить корнем, то получится ордерево, при этом дуги будут последовательно ориентированы от корня обходом в глубину.

Концевая вершина (полустепень исхода равна 0) ордерева называется **листом.** Путь из корня в лист называется **ветвью.** Длина наибольшей ветви ордерева называется **высотой. Уровень** узла ордерева — это расстояние от корня до узла. Сам корень имеет уровень 0. Узлы одного уровня образуют **ярус** дерева.

**Замечание.** Наряду с «растительной» применяется еще и «генеалогическая» терминология. Узлы, достижимые из узла **u**, называются **потомками** узла **u** (потомки образуют поддерево). Если в дереве существует дуга **(u,v),** то узел **u** называется **отцом** (или родителем*)* узла **v**, а узел **v** называется **сыном** узла **u**. Сыновья одного узла называются **братьями**.

## §6.3. Упорядоченные деревья

**Упорядоченное дерево T** — это конечное множество узлов, таких что:

1. имеется один узел **r,** называемый корнем данного дерева;
2. остальные узлы (исключая корень) содержатся в **k** попарно непересекающихся множествах $T_1,T_2,\ldots,T_k$, каждое из которых является ордеревом (множества $T_1,T_2,\ldots,T_k$ являются поддеревьями);
3. относительный порядок поддеревьев $T_1,T_2,\ldots,T_k$ фиксирован.

Другими словами, **упорядоченное** дерево T это ордерево, в котором множество сыновей каждого узла упорядочено. Порядок, как правило, устанавливается слева на право по старшинству.

Исходя из определения, упорядоченное дерево обозначается в виде $T=\{r,T_1,T_2,\ldots,T_k\}$.

Покажем на примере, как изображаются упорядоченные деревья графически.

**Пример_6.3.1.** Упорядоченные ориентированные деревья используются

1. Для представления выражений языков программирования. Пример представления выражения a+b∗c показан на рис. 6.3.1.а). Отметим, что если, например, операция «∗» не обладает свойством коммутативности, то вершины «b» и «c» переставлять местами нельзя, иначе результат вычисления не будет соответствовать данному выражению. Поэтому данный граф является упорядоченным.

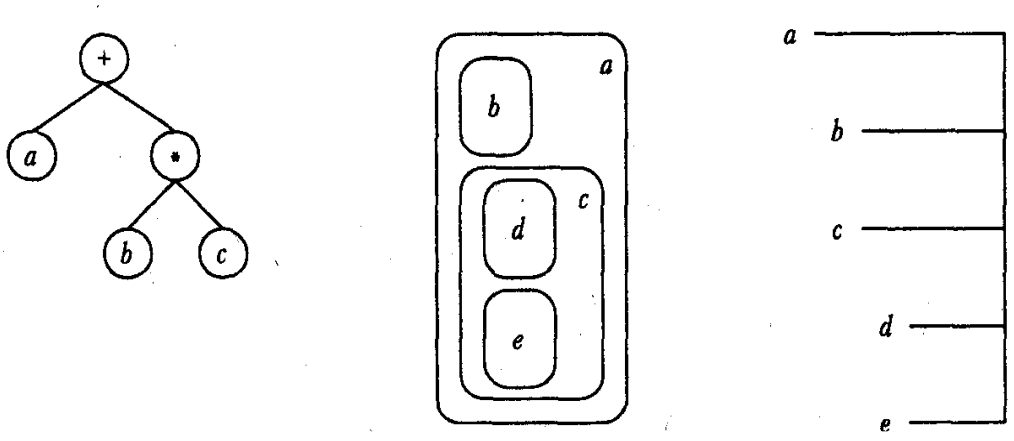
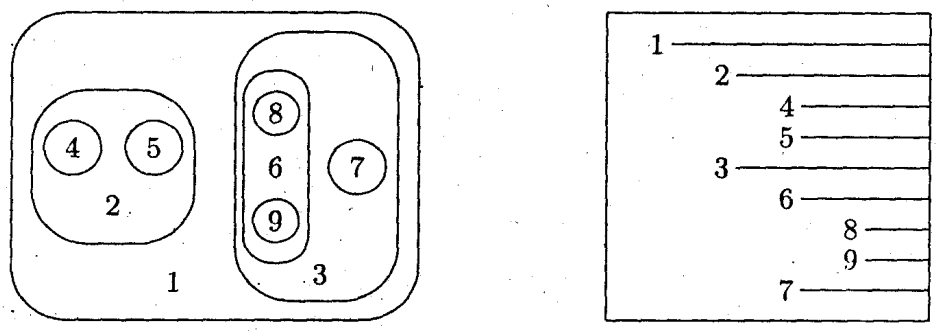



**Рис.6.5.1. а), б), в).**

2. Для представления блочной структуры программа, часто используется ориентированное дерево (может быть, неупорядоченное, так, как порядок определения переменных в блоке в большинстве языков программирования считается несущественным). На рис. 6.6.1.б) в центре показана структура областей определения идентификаторов *a, b, c, d, e*, причем для отображения структуры дерева использована альтернативная техника.

3. Для представления иерархической структуры вложенности элементов данных и/или операторов управления часто используется техника отступов, показанная на рис. 6.3.1.в).

**Пример_6.3.2**. Пусть задано упорядоченное дерево (1,(2,(4),(5)),(3,(6,(8),(9)),(7))). Здесь r=1, $T_1$=(2,(4),(5)), $T_2$=(3,(6,(8),(9)),(7)). Данному упорядоченному дереву соответствует система множеств, изображенная на рис.6.3.2.а), и уступчатый список изображенный на рис.6.3.2.б). Из данного изображения видно, что объекты 2 и 3 являются братьями, причем объект 3 старше 2, т.к. расположен правее (на графе вершина 3 будет также расположена правее вершины 2).




**а) Рисунок 6.3.2. б)**

Согласно следующему тезису любая схема, в которой заданы определенные приоритеты между элементами, может рассматриваться как некоторое упорядоченное дерево.

**Тезис.** Любая иерархическая классификационная схема интерпретируется некоторым упорядоченным деревом.

**Замечание.** Общепринятой практикой при изображении деревьев является соглашение о том, что корень находится наверху и все стрелки дуг ориентированы сверху вниз, поэтому стрелки можно не изображать. Таким образом, диаграммы свободных, ориентированных и упорядоченных деревьев оказываются графически неотличимыми, и требуется дополнительное указание, дерево какого класса изображено на диаграмме. В большинстве случаев это ясно из контекста.

**Пример 6.3.3.** На рис.6.3.3. приведены три диаграммы деревьев, которые внешне выглядят различными. Обозначим дерево слева — (1), в центре — (2) и справа — (3). Как упорядоченные деревья, они все различны: (1)≠(2), (2)≠(3), (3)≠(1) (значок означает «не изоморфно»). Как ориентированные деревья (1)=(2), но (2)≠(3). Как свободные деревья, они все изоморфны: (1)=(2)=(3).

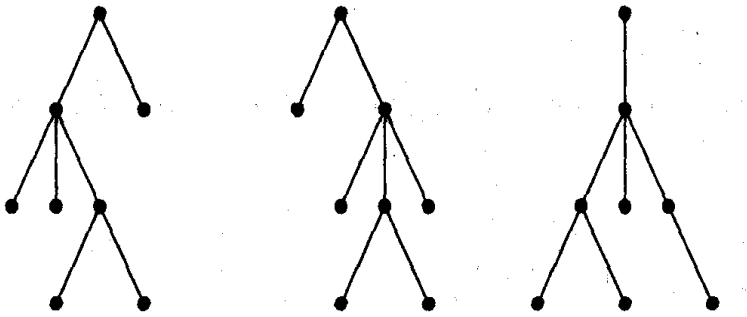


**Рисунок 6.3.3.**

## §6.4. Бинарные деревья

**Бинарное дерево** — это конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев — левого и правого.

Из определения следует, что в бинарном дереве каждый узел может иметь максимум две связи, причем связь влево (если она имеется) ведет к младшему сыну, а связь вправо (если она имеется) ведет к ближайшему по старшинству брату.

На рис.6.4.1. приведены две диаграммы деревьев, которые изоморфны как упорядоченные, ориентированные и свободные деревья, но не изоморфны как бинарные деревья.

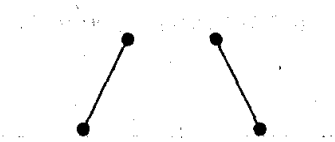


Рис. 6.4.1.

Напомним, что всякое свободное дерево можно ориентировать, назначив один из узлов корнем. Всякое ордерево можно произвольно упорядочить. Всякий упорядоченный лес (дерево) можно представить бинарным деревом, действуя по следующему правилу:

1. корень самого левого упорядоченного дерева является корнем бинарного дерева, остальные корни – его братья;
2. связи от узлов проводят так: правую связь к ближайшему старшему брату, а левую — к младшему сыну.

**Утверждение.** Всякому упорядоченному лесу (дереву) взаимно однозначно соответствует некоторое бинарное дерево.

**Пример_6.4.1.** На рис.6.4.2. приведены диаграммы упорядоченного и соответствующего ему бинарного деревьев.

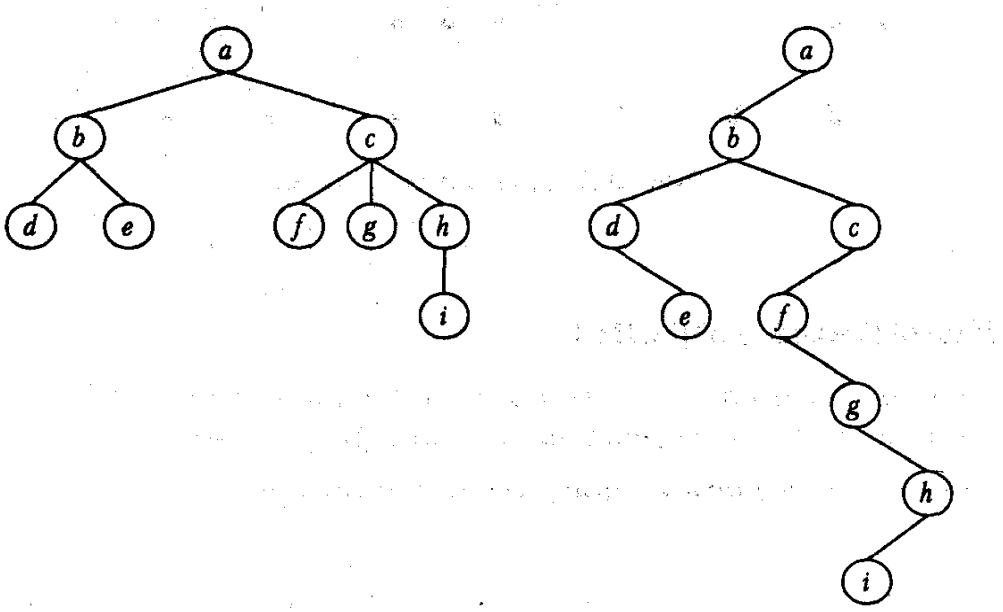


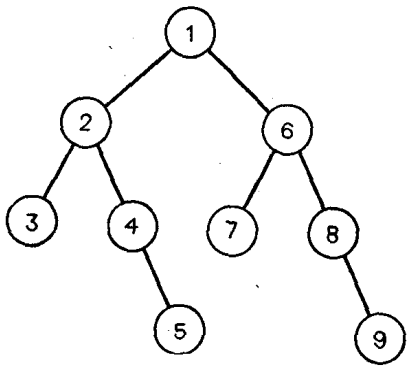

**Рис.6.4.2. а) упорядоченное дерево** **б) бинарное дерево**

Из утверждения 6.4.1. следует, что для представления деревьев в ЭВМ, достаточно рассмотреть представление в ЭВМ бинарных деревьев, т.к. они имеют более простое устройство.

**Списочные структуры:** каждый узел представляется записью типа N, содержащей два поля (**L** и **R**) с указателями на левый и правый узлы и еще одно поле **i** для хранения указателя на информацию об узле. Дерево представляется указателем на корень. Тип N обычно определяется следующим образом:

```
N = record
        i info;
        L,R :↑ N;
    end record.
```

На рисунке 6.4.3. представлено двоичное дерево и его представление в виде списочной структуры.




| i | L | R |
|---|---|---|
| 1 | 2 | 6 |
| 2 | 3 | 4 |
| 3 | 0 | 0 |
| 4 | 0 | 5 |
| 5 | 0 | 0 |
| 6 | 7 | 8 |
| 7 | 0 | 0 |
| 8 | 0 | 9 |
| 9 | 0 | 0 |

**Рис.6.4.3.**

## §6.5. Обходы бинарных деревьев

Большинство алгоритмов работы с деревьями основаны на обходах. Возможны следующие основные обходы бинарных деревьев:

**Прямой (левый) обход:**
попасть в корень,
обойти левое поддерево,
обойти правое поддерево.

**Обратный обход (симметричный или внутренний порядок):**
обойти левое поддерево,
попасть в корень,
обойти правое поддерево.

**Концевой (правый) обход:**
обойти левое поддерево,
обойти правое поддерево,
попасть в корень.

Данное описание обходов фактически дает алгоритм обхода с рекурсией. Приведем алгоритм симметричного обхода без рекурсии.

**Алгоритм. Нумерация узлов двоичного дерева в соответствии с обходом по внутреннему порядку.**

**Вход**. Двоичное дерево, представленное массивами ЛЕВЫЙ-СЫН и ПРАВЫЙ БРАТ.

**Выход.** Массив, называемый НОМЕР, такой, что НОМЕР(i) — номер узла i во внутреннем порядке.

**Метод.** Кроме массивов ЛЕВЫЙСЫН, ПРАВЫЙБРАТ и НОМЕР, алгоритм использует глобальную переменную СЧЕТ, значение которой — номер очередного узла в соответствии с внутренним порядком. Начальным значением переменной СЧЕТ является 1. Параметр УЗЕЛ вначале равен корню. При прохождении дерева в стеке запоминаются все узлы, которые еще не были занумерованы и которые лежат на пути из корня в узел, рассматриваемый в данный момент. При переходе из узла **v** к его левому сыну узел **v** запоминается в стеке. После нахождения левого поддерева для **v** узел **v** нумеруется и выталкивается из стека. Затем нумеруется правое поддерево для **v.** При переходе из **v** к его правому сыну, узел **v** не помещается в стек, поскольку после нумерации правого поддерева мы не хотим возвращаться в **v,** Более того, мы хотим вернуться к тому предку узла **v,** который еще не занумерован (т.е. к ближайшему предку **w** узла **v**, такому, что **v** лежит в левом поддереве для **w**).

```
begin
      СЧЕТ⟵1;
      УЗЕЛ⟵КОРЕНЬ;
      СТЕК⟵пустой;
левый:  while ЛЕВЫЙСЫН[УЗЕЛ] ≠ 0 do
              begin
                  затолкнуть УЗЕЛ в СТЕК;
                  УЗЕЛ ← ЛЕВЫЙСЫН[УЗЕЛ]
              end;
центр:    НОМЕР[УЗЕЛ]⟵СЧЕТ;
          СЧЕТ ⟵ СЧЕТ+1;
        if ПРАВЫЙБРАТ[УЗЕЛ] ≠ 0 then
            begin
                УЗЕЛ ⟵ ПРАВЫЙБРАТ[УЗЕЛ];
                goto левый
            end;
        if СТЕК не пуст then
            begin
              УЗЕЛ ⟵ элемент в вершине СТЕКа;
              вытолкнуть из СТЕКа;
              goto центр
            end
        end
```

**Пример_6.5.1.** На рис.6.5.1. изображено двоичное дерево, узлы которого пронумерованы в соответствии с прохождением его по внутреннему порядку (рис.6.5.1).

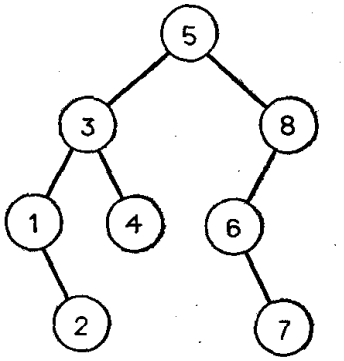



**Рис. 6.5.1.**

Если все узлы занумерованы в порядке посещения, то рассмотренные нумерации обладают рядом интересных свойств.

Номера узлов двоичного дерева, соответствующие внутреннему порядку, обладают тем свойством, что номера узлов в левом поддереве для **v** меньше и, а в правом поддереве больше **v**. Таким образом, чтобы найти узел с номером **w,** надо сравнить **w** с корнем **r**. Если **w=r**, то искомый узел найден. Если **w<r**, то надо повторить этот процесс для левого поддерева; если **w>r**, то повторить процесс для правого поддерева. В конце концов, узел с номером **w** будет найден.

## §6.6. Деревья сортировки

В этом разделе обсуждается одно конкретное применение деревьев в программировании, а именно деревья сортировки.

В практическом программировании для организации хранения данных и доступа к ним часто используется механизм, который обычно называют **ассоциативной памятью**. При использовании ассоциативной памяти данные делятся на порции (называемые *записями),* и с каждой записью ассоциируется ключ. **Ключ** — это значение из некоторого вполне упорядоченного множества, а записи могут иметь произвольную природу и различные размеры. Доступ к данным осуществляется по значению ключа, который обычно выбирается простым, компактным и удобным для работы.

**Примеры**. Ассоциативная память используется во многих областях жизни.

1. Толковый словарь или энциклопедия: записью является словарная статья, а ключом — заголовок словарной статьи (обычно его выделяют жирным шрифтом).

2. Адресная книга: ключом является имя абонента, а записью — адресная информация (телефон(ы), почтовый адрес и т. д.).

3. Банковские счета: ключом является номер счета, а записью — финансовая информация (которая может быть очень сложной).

Таким образом, ассоциативная память должна поддерживать по меньшей мере три основные операции:

1. добавить (ключ, запись);
2. найти (ключ): запись;
3. удалить (ключ).

**Эффективность каждой операции зависит от структуры данных**, используемой для представления ассоциативной памяти.

Для представления ассоциативной памяти используются следующие основные структуры данных:

1. неупорядоченный массив;
2. упорядоченный массив;
3. **дерево сортировки** — бинарное дерево, каждый узел которого содержит ключ и обладает следующим свойством: значения ключа во всех узлах левого поддерева меньше, а во всех узлах правого поддерева больше, чем значение ключа (см. обход бинарного дерева по внутреннему порядку).

При использовании неупорядоченного массива алгоритмы реализации операций ассоциативной памяти очевидны:

1. операция «добавить (ключ, запись)» реализуется добавлением записи в конец массива;
2. операция «найти (ключ): запись» реализуется проверкой в цикле всех записей в массиве;
3. операция «удалить (ключ)?» реализуется поиском удаляемой записи, а затем перемещением всех последующих записей на одну позицию вперед.

## §6.7. Алгоритмы на дереве сортировки.

**Алгоритм 6.7.1.** Поиск узла в дереве сортировки

**Вход:** дерево сортировки T, заданное указателем на корень; ключ *a*.
**Выход:** указатель **p** на найденный узел или **nil**, если в дереве нет такого ключа.

```
begin
   p:=T { указатель на проверяемый узел }
   while p≠ nil do
      if a<p.i  then
         p:=p.l { продолжаем поиск слева }
      else if a> p.i then
         p:=p.r { продолжаем поиск справа }
      else
         return p { нашли узел }
      end if
   end while
```

Этот алгоритм работает в точном соответствии с определением дерева сортировки. Если текущий узел не искомый, то в зависимости

от того, меньше или больше искомый ключ по сравнению с текущим ключом, нужно продолжать поиск соответственно слева или справа.

**Алгоритм 6.7.2. Вставка узла в дерево сортировки.**

**Вход:** дерево сортировки T, заданное указателем на корень; ключ *a*.
**Выход:** модифицированное дерево сортировки T.

```
begin
    if T == nil then
       T: == NewNode(a) { первый узел в дереве }
       return T
    end if
    p:=T{указатель на текущих узел}
    M : { анализ текущего узла }
    if a< p.i then
        if p.l = nil then
           q:=NewNode(a) {создаем новый узел }
           p.l:= q {и подцепляем его к p слева}
           return T
        else
           p:=p.l {продолжаем поиск места для вставки слева}
           goto M
        end if
    end if
    if a > p.i then
        if p.l = nil then
           q:=NewNode(a) { создаем новый узел }
           p.r:=q { и подцепляем его к p справа }
           return T
        else
           p:=p.r {продолжаем поиск места для вставки справа}
           goto M
        end if
    end if
    return T {сюда попали, если уже есть такой ключ!}
procedure NewNode.
  Вход: ключ a. Выход: указатель p на созданный узел.
  begin
        new(p); p.i:=a; p.l:=nil; p.r:=nil
  return p
```

Алгоритм вставки, в сущности, аналогичен алгоритму поиска: в дереве ищется такой узел, имеющий свободную связь для подцепления нового узла, чтобы не нарушалось условие дерева сортировки. А именно, если новый ключ меньше текущего, то либо его можно подцепить слева (если левая связь свободна), либо нужно найти слева подходящее место. Аналогично, если новый ключ больше текущего.

**Алгоритм 6.7.3. удаления из дерева сортировки.**

Удаление узла производится перестройкой дерева сортировки. При этом возможны три случая (не считая тривиального случая, когда удаляемого узла нет в дереве и ничего делать не нужно).

1. Правая связь удаляемого узла **p** пуста (см. рис. 6.7.1. слева). В этом случае левое поддерево 1 узла **p** подцепляется к родительскому узлу **q** *c* той же стороны, с которой был подцеплен узел **p**. Условие дерева сортировки, очевидно, выполняется.

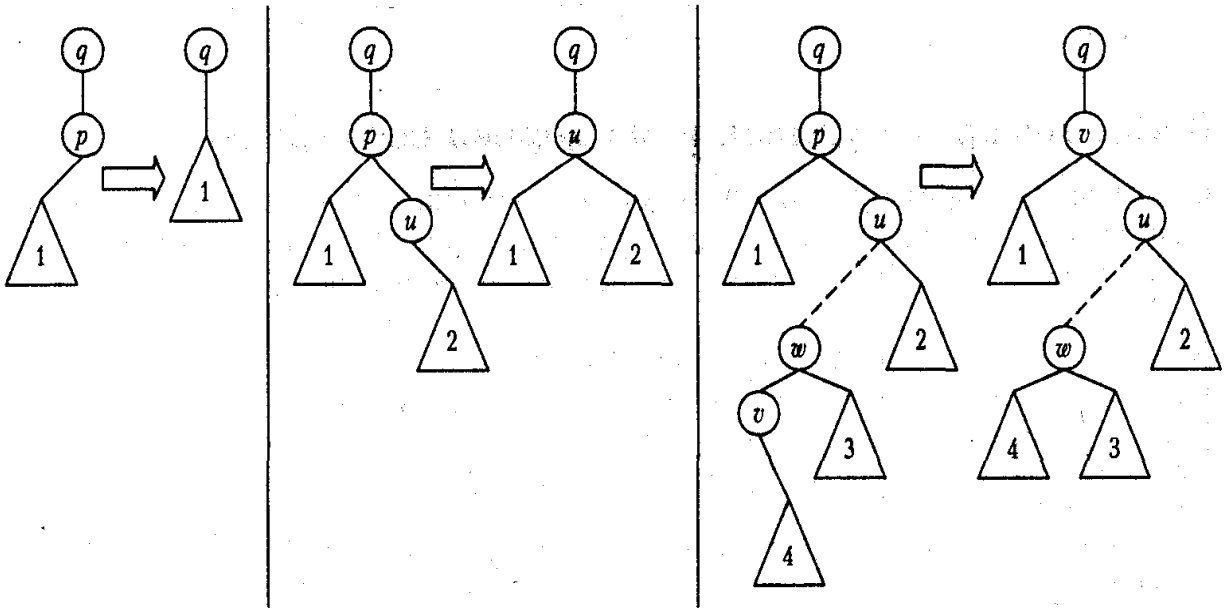



**Рис.7.7.1. Иллюстрация к алгоритму удаления из дерева сортировки.**

2. Правая связь удаляемого узла **p** не пуста и ведет в узел ***u***, левая связь которого пуста (см. рис. 6.7.1. в центре). В этом случае левое поддерево 1 узла **p** подцепляется к узлу ***u*** слева, а сам узел ***u*** подцепляется к родительскому узлу ***q*** *c* той же стороны, с которой был подцеплен узел ***p***. Нетрудно проверить, что условие дерева сортировки выполняется и в этом случае.

3. Правая связь удаляемого узла ***p*** не пуста и ведет в узел ***u***, левая связь которого не пуста. Поскольку дерево сортировки конечно, можно спуститься от узла ***u*** до узла ***v***, левая связь которого пуста (см. рис. 6.7.1. справа). В этом случае выполняются два преобразования дерева. Сначала информация в узле ***p*** заменяется на информацию узла ***v***. Поскольку узел ***v*** находится в правом поддереве узла **p** и в левом поддереве узла ***u***, имеем ***p.i < v.i < u.i.*** Таким образом, после этого преобразования условие дерева сортировки выполняется. Далее правое поддерево 4 узла *v* подцепляется слева к узлу ***w***, а сам узел ***v*** удаляется. Поскольку поддерево 4 входило в левое поддерево узла ***w,*** условие дерева сортировки также сохраняется

## §6.8. Сравнение представлений ассоциативной памяти

Пусть n — количество элементов в ассоциативной памяти. Тогда сложность операций для различных представлений ограничена сверху следующим образом.

| | неупорядочный массив | упорядочный массив | дерево сортировки |
|---|---|---|---|
| добавить | O(1) | O(n) | $O(\log_2(n))..O(n)$ |
| найти | O(n) | $O(\log_2(n))$ | $O(\log_2(n))..O(n)$ |
| удалить | O(n) | O(n) | $O(\log_2(n))..O(n)$ |

Эффективность операций с деревом сортировки ограничена сверху высотой дерева.

Ордерево называется **выровненным,** если все узлы, степень которых меньше 2, располагаются на одном или двух последних уровнях. Выровненное дерево имеет наименьшую возможную для данного числа узлов р высоту h.

## §6.9. Кратчайший остов.

Пусть G(V,E) — граф. **Остовной подграф** графа G(V,E) — это подграф, содержащий все вершины. Остовный подграф, являющийся деревом, называется **остовом.**

Несвязный граф не имеет остова. Связный граф может иметь много остовов.

Если задать длины ребер, то можно поставить задачу нахождения **кратчайшего** остова. Эта задача имеет множество практических интерпретаций. Например, пусть задано множество аэродромов и нужно определить минимальный (по сумме расстояний) набор авиарейсов, который позволил бы перелететь с любого аэродрома на любой другой. Решением этой задачи будет кратчайший остов полного графа расстояний между аэродромами.

**Алгоритм**, решающий задачу **нахождения остова минимального веса** во взвешенном графе G(V,E), заключается в следующем.

1. Строим граф T(1), состоящий из множества вершин V и единственного ребра e(i), которое имеет минимальный вес.

2. Если граф T(k) уже построен и k < n-1, где n=|V|, то строим граф T(k+1), добавляя к множеству ребер графа T(k) ребро e(i+1)

имеющее минимальный вес среди ребер, не входящих в T(k) и не составляющих циклов с ребрами из T(k).

Приведенный алгоритм, в частности, позволяет находить остов в невзвешенном графе, положив для всех ребер вес равный 1.

**Пример 6.9.1**. На рис.6.9.1 показан остов минимального веса взвешенного графа. Вес остова равен 9.

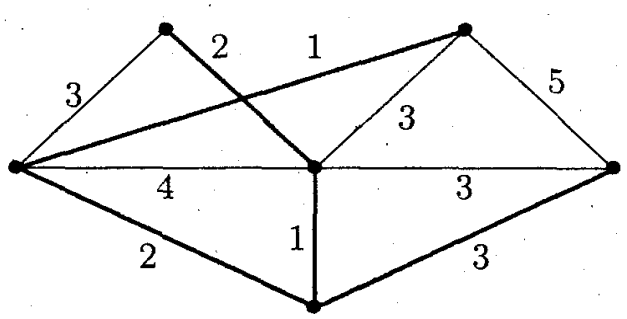



**Рис.6.9.1. Толстыми линиями выделен минимальный остов.**

# Глава 7. Раскраски графов. Планарные графы.

## §7.1. Планарные графы.

Неориентированный граф *G* называется **планарным,** если его можно изобразить на плоскости так, что никакие два ребра не пересекаются. Такое изображение графа на плоскости называется **плоским.** Таким образом, если граф имеет плоское изображение, то он является планарным.

Пример 7.1.1. Граф $K_4$. (рис.7.1.1.а) планарен, поскольку может быть изображен, как показано на рис.7.1.1.*б*.

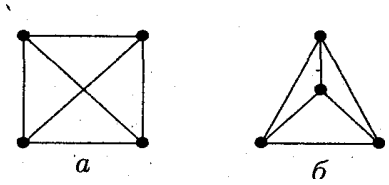



**Рисунок 8.1.1.**

Рассмотрим операцию **подразбиения ребра** в графе G = (V,E). После подразбиения ребра (a,b)∈E получается граф G' = (V',E'), где $V'=V\cup\{c^*\}$, т.е. в множество вершин добавляется новая вершина $c^*$, $E'=(E\setminus\{(a,b)\})\cup\{(a,c^*),(c^*,b)\}$, т. е. ребро (a,b) заменяется на два ребра $(a,c^*),(c^*,b)$. Аналогично, определяется операция надразбиения: удаляется вершина v степени 2 вместе с ребрами (v,u), (v,w) и добавляется ребро (u,w).

Два графа называются **гомеоморфными,** если один из них можно получить (с точностью до изоморфизма) из другого с помощью последовательного применения операций подразбиений и надразбиений ребер.

Не всякий граф является планарным. Критерий планарности описывает

**Теорема_7.1.1.**(теорема Понтрягина — Куратовского). Граф G планарен тогда и только тогда, когда G не содержит подграфа, гомеоморфного $K_5$ или $K_{3,3}$ (рис.7.1.2).

**Следствие.** Графы $K_5$ и $K_{3,3}$ не являются планарными.

**Замечание.** Любой конечный граф может быть изображен в трехмерном пространстве $R^3$ без пересечения дуг вне их концов.

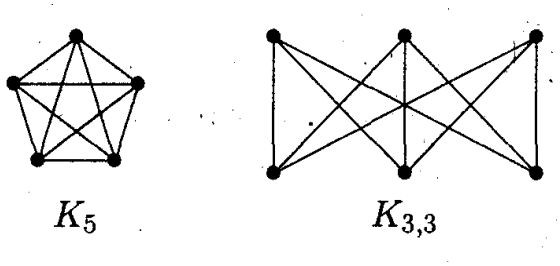



**Рисунок 7.1.2.**

Если граф *G* непланарен, то для его геометрической реализации удаляют отдельные ребра (переносят на другую плоскость). Минимальное число ребер, которое необходимо удалить из графа для получения его плоского изображения, называется **числом планарности графа G.** При вынесении этих ребер на вторую плоскость получают часть графа, которая также может оказаться неплоской. Тогда вновь решают задачу вынесения отдельных ребер на следующую плоскость и т. д. Минимальное число плоскостей **m**, при котором граф *G* разбивается на плоские части G1,G2,…,Gm (разбиение ведется по множеству ребер), называется **толщиной графа** G.

**Теорема 7.1.2.** Нижняя оценка толщины t(G) n-вершинного графа G дается неравенством

$$t(G) \geq 1 + \left[ \frac{\sum_{i=1}^{n} \delta(i) - 2}{6 \cdot (n-2)} \right]$$

где [*a*]-целая часть числа *a,* $\delta$(i) – степень вершины i.

**Теорема_7.1.3.** (Эйлер). В связном планарном графа G справедливо равенство: **p-q+r=2,**
где p-число вершин графа G, q- число ребер графа G, r- число областей на которые разбивается плоскость плоским изображением графа G.

**Следствие_1.** Если G связный планарный граф (p>3), то q≤3p-6.

**Следствие_2.** В любом планарном графе существует вершина, степень которой не больше 5.

## §7.2. Раскраска графов.

Раскраска вершин (ребер) графа. **k** красками называется правильной, если смежные вершины (ребра) окрашены в разные цвета. **Хроматическим числом h(G)** (хроматическим классом H(G)) графа G называется минимальное число красок **k**, для которого граф *G* имеет правильную раскраску вершин (ребер). Граф, хроматическое число которого равно **k**, называется **k-хроматическим.** Граф *G* называется **бихроматическим,** если h(G)=2.

Между понятиями "хроматическое число" и "хроматический класс" существует тесная связь, которая осуществляется следующим образом.

Пусть G=(V,X) исходный граф. Построим граф G1=(V1,X1) так: вершины V1 графа G1 взаимно однозначно соответствуют ребрам X графа G и вершины графа G1 смежные тогда и только тогда, когда смежные соответствующие ребра графа G. Хроматический класс H=(G) графа G совпадает с хроматическим числом h(G1) графа G1. Т.о. задача определения хроматического класса сводится к задаче определения хроматического числа,

**Пример_7.2.1**. Так как в полном графе $K_n$ любые две различные вершины связаны ребром, то $h(K_n)=n$.

Многие практические задачи сводятся к построению раскрасок графов.

**Пример_7.2.2.** Рассмотрим задачу составления расписания. Предположим, что нужно прочесть несколько лекций за кратчайшее время. Чтение каждой лекции в отдельности занимает один час, но некоторые лекции не могут читаться одновременно (например, их читает один и тот же лектор). Построим граф G, вершины которого биективно соответствуют лекциям и две вершины смежны тогда и только тогда, когда соответствующие им лекции нельзя читать одновременно. Очевидно, что любая раскраска этого графа определяет допустимое расписание: лекции, соответствующие вершинам одного цвета, могут читаться одновременно. Напротив, любое допустимое расписание определяет раскраску графа *G*. Оптимальные расписания соответствуют раскраскам с минимальным числом цветов, а число часов, необходимое для прочтения всех лекций, равно h(G).

**Пример 7.2.3.** Рассмотрим граф G, вершины которого — страны, а ребра соединяют страны, имеющие общую границу. Числу h(G)

соответствует наименьшее число красок, необходимых для раскраски карты так, чтобы никакие две соседние страны не были окрашены в один цвет.

Существуют и практические задачи, связанные с раскраской ребер в мультиграфе.

**Пример 7.2.4.** Проводится монтаж аппаратуры. Чтобы не перепутать проводники, необходимо их окрасить таким образом, чтобы два проводника, идущие к одной плате, имели разные цвета. В этом случае вершинами являются платы, а ребрами — проводники.

**Теорема_7.2.1.(Кенита)** Пусть G — граф без петель, имеющий хотя бы одно ребро. Тогда следующие условия эквивалентны:

1) G — бихроматический граф;
2) G — двудольный граф;
3) G не содержит циклов нечетной длины.

**Следствие_1.** Если G — лес, то $h(G)\le 2$.

**Теорема_7.2.2.** Для любого графа G без петель выполняется неравенство $h(G)\le \deg(G)+1$ , где deg(G) максимальная степень вершин графа G.

**Теорема_7.2.3.(теорема о четырех красках)** Если граф G планарен, то $h(G)\le 4$.

## §7.3. Алгоритмы раскраски графов.

Рассмотрим простой алгоритм построения раскраски, который во многих случаях приводит к раскраскам, близким к минимальным.

**Алгоритм последовательной раскраски.**

1. Произвольная вершина **v(1)** графа G принимает цвет 1.
2. Если вершины v(1),…,v(i), раскрашены m цветами (1,2,…,m) $m\le i$, то новой произвольно взятой вершине v(i+1) припишем минимальный цвет, не использованный при раскраске вершин из множества вершин смежных с v(i+1): {v(j)|расстояние[v(i+1),v(j)]=1, j<i}.

Для некоторых классов графов последовательная раскраска является минимальной. В общем случае это не так.

**Алгоритм минимальной раскраски вершин графа G=(V,X).**

1. Множество вершин смежных вершине v обозначим T(v). Для каждой пары смежных вершин v, w подсчитываем функционал

$$\phi(v,w)=\frac{\left|T(v)\cap T(w)\right|}{|T(v)|+|T(w)|}$$, где |T(v)|-мощность множества v,

и определяем на какой паре (v,W) этот функционал принимает наибольшее значение (таких пар может быть несколько, тогда выбирать можно любую из них).

2. Найденную пару раскрашиваем в разные цвета, и им взаимно однозначно сопоставляем столбцы в двумерной таблице, строкам которой соответствуют вершины, смежные хотя бы с одной раскрашенной вершиной; в клетке (i,j) ставим 1, если i-я и j-я" вершины смежны, и 0 в противном случае.

3. Выбираем строку с наибольшим числом единиц.

4. Если для выделенной в п.3 **k**-ой строки найдется столбец S, на пересечении с которым находится 0 (т.е. клетка (k,S) равна 0), то соответствующую k-ой строке вершину раскрасим в краску, которой раскрашивается "S-й столбец" и произведем склеивание соцветных по этой краске вершин. В противном случае "k-ю" вершину раскрасим в новую краску, увеличивая количество столбцов исходной таблицы.

5. Если осталась хотя бы одна неокрашенная вершина, то переходим к п.3 в противном случае к п.6.

6. Хроматическое число h(G) равно количеству столбцов в итоговой таблице. Конец.

# Глава 8. Циклы.

## §8.1. Циклы и коциклы.

Цикл может входить только в одну компоненту связности графа, поэтому далее без ограничения общности граф считается связным. Цикл (простой) рассматривается как множество ребер.

**Разрезом** связного графа называется множество ребер, удаление которых делает граф несвязным. **Простым разрезом** называется минимальный разрез, то есть такой, никакое собственное подмножество которого разрезом не является.

В этом параграфе рассматриваются только простые циклы и разрезы, далее слово «простые» опускается.

Между циклами и разрезами существует определенная двойственность, поэтому разрезы иногда называют **коциклами.**

**Замечание.** Чем больше в графе циклов, тем труднее его разрезать. В дереве, напротив, каждое ребро само по себе является разрезом.

## §8.2. Независимые множества циклов и коциклов.

Рассмотрим операцию $\oplus$ сложения по модулю 2 или симметрической разности над множествами ребер:

$M1 \oplus M2 = \{e \mid (e \in M1 \& e \notin M2) \vee (e \notin M1 \& e \in M2)\}$.

Другими словами, множество $M1 \oplus M2$ состоит только из тех ребер которые входят только в одно из множеств M1или M2.

Множество M называется **зависимым** или **линейной комбинацией** множеств $\{M_k, k=1,2,\ldots,n\}$, если M является суммой некоторых $M_k$, т.е. $M=\oplus M_k$. Множество циклов $\{Z_k, k=1,2,\ldots,n\}$ называется **независимым**, если ни один из циклов $Z_k$, не является линейной комбинацией остальных. Множество разрезов $\{S_k, k=1,2,\ldots,n\}$ называется **независимым,** если ни один из разрезов $S_k$ не является линейной комбинацией остальных.

Максимальное независимое множество циклов (или минимальное множество циклов, от которых зависят все остальные) называется **фундаментальной системой циклов.** Циклы фундаментальной системы называются **фундаментальными,** а количество циклов в (данной) фундаментальной системе называется циклическим рангом, или **цикломатическим числом,** графа G и обозначается **m(G)**. Максимальное независимое множество коциклов (разрезов) (или минимальное множество коциклов, от которых зависят все остальные) называется **фундаментальной системой коциклов (разрезов).** Коциклы (разрезы) фундаментальной системы называются **фундаментальными,** а количество коциклов в (данной) фундаментальной системе называется коциклическим рангом**,** или **коцикломатическим числом,** графа G и обозначается **m*(G).**

**Теорема 8.2.1.** Если G — граф с **p(G)** вершинами, **q(G)** ребрами и **c(G)** компонентами связности, то цикломатическое число графа G вычисляется по формуле

$$m(G)=q(G)-p(G)+c(G).$$

**Следствие.** Если G — связный граф, то

$$m(G)=q(G)-p(G)+1.$$

## §8.3. Фундаментальные циклы.

Пусть G=(V,X)—связный граф, имеющий n вершин, q ребер и одну компоненту связности, T — остов графа G. Тогда T имеет $v^*\{G\}=n-1$ ребер u(1),...,u(n-1), которые будем называть ветвями остова T. Оставшиеся q-n+1 ребер v(1),...,v(q-n+1) не входящие в T, называются **хордами** остова T. Если к дереву T добавить произвольную хорду v(i), то в полученном графе найдется ровно один цикл, который обозначим через C(i). Цикл C(i), состоит из хорды v(i) и ветвей остова T, которые принадлежат единственной простой цепи, соединяющей вершины хорды v(i). Цикл C(i) называется **фундаментальным циклом** графа G относительно хорды v(i) остова T. Множество {C(1),...,C(q-n+1)} всех фундаментальных циклов относительно хорд остова T называется фундаментальным множеством циклов графа G относительно остова T. Отметим, что мощность

фундаментального множества циклов равна цикломатическому числу **m(G)**.

Обозначим через (w(1),w(2),...,w(q)) последовательность {v(1),...,v(q-n+1),u(1),...,u(n-1)} всех ребер графа G. Тогда фундаментальному циклу C(i) соответствует вектор $a_i=(a_{i1},a_{i2},\ldots,a_{im})$ определенный по следующему правилу:

$$a_{ij} = \begin{cases} 1, & \text{если } w(j) \in C(i) \\ 0, & \text{если } w(j) \notin C(i). \end{cases}$$

Теперь фундаментальное множество циклов можно задать с помощью матрицы **C** фундаментальных циклов, строками которой являются вектора $a_1,a_2,\ldots,a_{m(G)}$.

Так как каждый фундаментальный цикл C(i) содержит ровно одну хорду, а именно v(i), то матрица **C** имеет вид

$$C = \begin{pmatrix} 100\ldots0 \mid a_{1,m(G)} \ldots\ldots\ldots\ldots a_{1q} \\ 010\ldots0 \mid a_{2,m(G)} \ldots\ldots\ldots\ldots a_{2q} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ 00\ldots01 \mid a_{m(G),m(G)} \ldots a_{m(G),q} \end{pmatrix}$$

Т. о. C=(C1,C2), где C1 — единичная матрица порядка m(G).

**Пример 8.3.1**. Найдем матрицу фундаментальных циклов графа G, изображенного на рис.8.3.1.

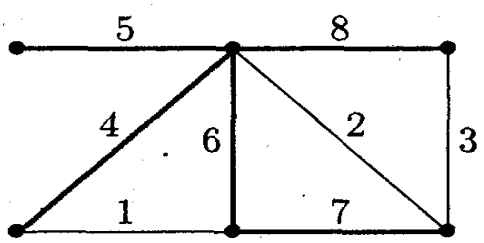



**Рисунок 8.3.1.**

Так как m(G)=8-6+1=3, то для получения остова удаляем из графа три ребра. Сопоставим этим ребрам номера 1, 2, 3. Ребрам, входящим в остов, поставим в соответствие номера 4, 5; 6, 7, 8. Легко видеть, что фундаментальный цикл C(1) соответствующий хорде 1, состоит из ребер 1, 4, 6, цикл C(2) — из ребер 2, 6, 7, цикл C(3) — из ребер 3, 6, 7, 8. Поэтому матрица .фундаментальных циклов **C** имеет вид

| | | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|---|
| | C(1) | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| C= | C(2) | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| | C(3) | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |

Следующие две теоремы дают информацию о связи остовов с разрезами, а также циклов с разрезами.

**Теорема 8.3.1.** В связном графе остовное дерево имеет по крайней мере одно общее ребро с любым из разрезов графа.

**Теорема 8.3.2.** В связном графе любой цикл имеет с любым разрезом четное число общих ребер.

## §8.4. Фундаментальные разрезы.

В условиях предыдущего параграфа рассмотрим неориентированный граф G с остовом Г. Снова пусть u(1),...,u(n-1) — ветви остова Г. Удаляя из остова Т произвольную ветвь u(i), получаем лес с 2 компонентами связности M1, M2 (рис.8.4.1.). В графе G могут найтись еще какие-то ребра $v(i_1),v(i_2),\ldots,v(i_k)$ (являющиеся хордами Г), которые соединяют вершины из M1 и M2. Множество $K(i)=\{u(i),v(i_1),v(i_2),\ldots,v(i_k)\}$ образует простой разрез, который называется фундаментальным разрезом графа G относительно ветви u(i) остова Г. Множество {K(1),K(2),...,K(n-1)} всех фундаментальных разрезов графа G называется фундаментальным множеством коциклов графа G относительно остова Г.

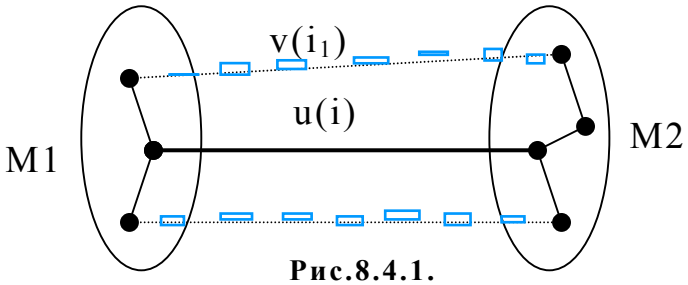



**Рис.8.4.1.**

Аналогично фундаментальным циклам фундаментальному разрезу K(i) ставится в соответствие вектор $\mathbf{b_i=(b_{i1},b_{i2},\ldots,b_{im})}$ определенный по следующему правилу:

$$b_{ij} = \begin{cases} 1, & \text{если } w(j) \in K(i) \\ 0, & \text{если } w(j) \notin K(i). \end{cases}$$

Фундаментальное множество коциклов задается матрицей фундаменталъных разрезов К, строки, которой являются векторами $\mathbf{b_i}$.

**Пример 8.4.1.** Найдем матрицу фундаментальных разрезов графа G=(V,E), изображенного на рис. 8.3.1. Ребру 4 соответствует коцикл K(1)={1,4}. Аналогично ребру 5 соответствует коцикл K(2)={5}, ребру 6 — коцикл K(3)={1,2,3,6}, ребру 7 — коцикл K(4)={2,3,7}, ребру 8 — коцикл K(5)={3,8}. Следовательно, матрица фундаментальных разрезов имеет вид

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| | K(1) | **1** | 0 | 0 | **1** | 0 | 0 | 0 | 0 |
| | K(2) | 0 | 0 | 0 | 0 | **1** | 0 | 0 | 0 |
| K= | K(3) | **1** | **1** | **1** | 0 | 0 | **1** | 0 | 0 |
| | K(4) | 0 | **1** | **1** | 0 | 0 | 0 | **1** | 0 |
| | K(5) | 0 | 0 | **1** | 0 | 0 | 0 | 0 | **1** |

## §8.5. Эйлеровы циклы.

Если граф имеет цикл (не обязательно простой), содержащий все ребра графа по одному разу, то такой цикл называется **эйлеровым циклом,** а граф называется эйлеровым графом. Если граф имеет цепь (не обязательно простую), содержащую все вершины по одному разу, то такая цепь называется **эйлеровой цепью**, а граф называется полуэйлеровым графом.

**Теорема_8.5.1.** Если граф G связен и нетривиален, то следующие утверждения эквивалентны:

1. G — эйлеров граф;
2. каждая вершина G имеет четную степень;
3. множество ребер G можно разбить на простые циклы.

В предыдущей теореме был установлен эффективный способ проверки наличия эйлерова цикла в графе. А именно, для этого необходимо и достаточно убедиться, что степени всех вершин четные, что нетрудно сделать при любом представлении графа. Следующий алгоритм находит эйлеров цикл в графе, если известно, что граф эйлеров.

**Алгоритм 8.5.1.** Алгоритм построения эйлерова цикла

**Вход:** Эйлеров граф G(V, E), заданный списками смежности (Γ(v)-список вершин, смежных с вершиной v).

**Выход:** последовательность вершин эйлерова цикла.

```
S:=∅  {стек для хранения вершин}
select v∈V  {произвольная вершина}
v→S  {положить v в стек S}
while S≠∅ do
    v←S; v→S    {v-верхний элемент стека}
    if Γ[v]=∅ then
        v←S; yield v
    else
       select u∈Γ[v] {взять первую вершину из списка смежности }
       u→S        {положить u в стек }
       Γ[v]:=Γ[v]\{u}; Γ[u]:=Γ[u]\{v}  {удалить ребро (v, и)}
```

**end if**
**end while**

**Обоснование.** Принцип действия этого алгоритма заключается в следующем. Начиная с произвольной вершины, строим путь, удаляя ребра и запоминая вершины в стеке, до тех пор пока множество смежности очередной вершины не окажется пустым, что означает, что путь удлинить нельзя. Заметим, что при этом мы с необходимостью придем в ту вершину, с которой начали. В противном случае это означало бы, что вершина v имеет нечетную степень, что невозможно по условию. Таким образом, из графа были удалены ребра цикла, а вершины цикла были сохранены в стеке S. Заметим, что при этом степени всех вершин остались четными. Далее вершина v выводится в качестве первой вершины эйлерова цикла, а процесс продолжается, начиная с вершины, стоящей на вершине стека.

## §8.6. Гамильтоновы графы

Если граф имеет простой цикл, содержащий все вершины графа (по одному разу), то такой цикл называется **гамильтоновым циклом**, а граф называется **гамильтоновым графом**. Гамильтонов цикл не обязательно содержит все ребра графа. Ясно, что гамильтоновым может быть только связный граф.

**Теорема_8.6.1.** (Дирак) Если в n-вершинном графе G(V, E) степень любой вершины больше $\frac{n}{2}$, то граф G является гамильтоновым.

Рассмотрим следующую задачу, известную как задача коммивояжера. Имеется **n** городов, расстояния между которыми известны. Коммивояжер должен посетить все **n** городов по одному разу, вернувшись в тот, с которого начал. Требуется найти такой маршрут движения, при котором суммарное пройденное расстояние будет минимальным.

Очевидно, что задача коммивояжера — это задача отыскания кратчайшего гамильтонова цикла в полном графе. Можно предложить следующую простую схему решения задачи коммивояжера: сгенерировать все **n** возможных перестановок вершин полного графа, подсчитать для каждой перестановки длину маршрута и выбрать из них кратчайший (алгоритм перебора всех возможностей). Очевидно, такое вычисление потребует O(n!) шагов.

Как известно, n! с ростом n растет быстрее, чем любой полином от n, и даже быстрее, чем $2^n$. Таким образом, решение задачи коммивояжера описанным методом полного перебора оказывается

практически неосуществимым даже для сравнительно небольших n. Более того, известно, что задача коммивояжера принадлежит к числу так называемых NP-полных задач. Вкратце суть проблемы NP-полноты сводится следующему. В различных областях дискретной математики, комбинаторики, логики и т. п. известно множество задач, принадлежащих к числу наиболее фундаментальных, для которых, несмотря на все усилия, не удалось найти алгоритмов решения, имеющих полиномиальную (степенную) трудоемкость. Более того, если бы удалось отыскать эффективный алгоритм решения хотя бы одной из этих задач, то из этого немедленно следовало бы существование эффективных алгоритмов для всех остальных задач данного класса. На этом основано общепринятое мнение, что таких алгоритмов не существует.

Отметим, что задача отыскания гамильтонова цикла или эквивалентная задача коммивояжера являются практически востребованными, но для них неизвестен (а скорее всего не существует) эффективный алгоритм решения.

# Раздел 3. Конечные автоматы.

Одним из применений конечных автоматов является задача идентификации (распознавания образов). Например, идентификация цепочек символов входит как составная часть во многие задачи, связанные с редактированием текстов, поиском данных и символьной обработкой. В типичной задаче идентификации цепочек задаются цепочка-текст **x** и множество цепочек-образов {Y1,Y2,...,Yn}. Требуется найти либо одно, либо все вхождения цепочек-образов в цепочке-тексте **x.**

## §3.1. Определение автомата.

**Конечным автоматом** называется совокупность пяти объектов $Z=(A,B,Q,\varphi,\psi)$, где

1) A – конечное множество, называемое **входным алфавитом**;

2) B – конечное множество, называемое **выходным алфавитом**,

3) Q – конечное множество (**множество состояний** автомата) с отмеченным элементом q0, называемым начальным состоянием автомата;

4) $\varphi$: Q×A→Q – **функция перехода;**

5) $\psi$: Q×A→B – **функция выхода**.

Опишем работу конечного автомата. Автомат работает в дискретные моменты времени (условно t=0,1,2,3,...). В начальный момент времени t=0 автомат находится в состоянии q0. В момент времени t=1 на вход автомата подается некоторая буква (сигнал) из входного алфавита A, которую условно обозначим $\boldsymbol{a}(1)$ (т.е. $a(1)\in A$). В результате этого автомат переходит в состояние $q(1)=\varphi(q0,a(1))$ (возможно совпадающее с q0), а на его выходе появляется сигнал $b(1)=\psi(q0,a(1))$ из множества выходного алфавита B, описываемый функцией $\psi$. В момент времени t==2 на вход канала подается буква $a(2)$. Тогда в момент t=2 автомат перейдет в состояние $q(2)=\varphi(q(1),a(2))$, а на его выходе появляется буква $b(2)=\psi(q(1),a(2))$. Работа автомата продолжается до тех пор, пока на его вход подаются буквы (сигналы) из алфавита A.

Конечная последовательность входных сигналов $a(1),a(2),...,a(t)$ образует входное слово, а последовательность b(1),b(2),...,b(t) - выходное слово.

Таким образом определенные автоматы (как совокупность пяти объектов $Z=(A,B,Q,\varphi,\psi)$) называют **автоматами Мили.**

**Замечание.** Если множество $A=A_1\times A_2\times\ldots\times A_n$, а $B=B_1\times B_2\times\ldots\times B_m$, то говорят, что задан конечный автомат с n входами и m выходами. Символами такого автомат служат сигналы вида $a=(a_1,a_2,\ldots,a_n)$, $b=(b_1,b_2,\ldots,b_m)$, где $a_k\in A_k$ $b_j\in B_j$ $k=1,2,\ldots n$, $j=1,2,\ldots,m$. Тогда функция выходов является вектор-функцией, т.к. в каждый момент времени на выходе имеется m сигналов.

Блок-схема конечного автомата Мили приведена на рис.3.1.1.

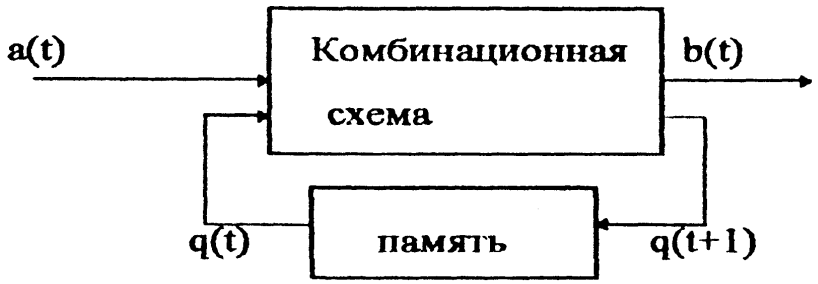



**Рисунок 3.1.1.**

Определим теперь автомат Мура, или автомат без выхода.

**Определение. Автоматом Мура** называется тройка $V=(A,Q,\varphi)$ где

1) A – конечное множество, называемое **входным алфавитом**;

2) Q – конечное множество (**множество состояний** автомата) с отмеченным элементом q0 называемым начальным состоянием автомата;

3) $\varphi$: $Q\times A\to Q$ – **функция перехода;**

Равенство $\varphi(q,a)=q1$ означает, что если автомат находится в состоянии q и принимает символ $a$, то должен осуществиться его переход в состояние q1.

Покажем, что на самом деле в автомате Мили можно "избавиться" от выходного алфавита B, увеличив соответствующим образом множество состояний Q, т.е. **автомат Мили эквивалентен в определённом смысле подходящему автомату Мура.** Пусть дан автомат Мили $Z=(A,B,Q,\varphi,\psi)$. Построим автомат Мура $V=(A^*,Q^*,\varphi^*)$. В качестве множества состояний нового автомата возьмём $Q^*=Q\times B$, входной алфавит сохраним прежним: $A^*=A$, а функцию $\varphi 1$ определим следующим образом:

$$\varphi^*(q^*,a)=\varphi^*((q,b),a)=(\varphi(q,a),\psi(q,a)),\quad \text{где}\quad q^*=(q,b)\in Q^*.$$

Автомат Мура V=(A*,Q*,$\varphi$*) эквивалентен автомату Мили Z=(A,B,Q,$\varphi$,$\psi$) в том смысле, что V "работает" так же, как Z, но реакцией на входной сигнал является не выходной символ **b**, а изменённое более сложным образом состояние, в котором фактически "зашифрован" выходной символ **b**.

**В дальнейшем мы будем рассматривать только автоматы Мили.**

## §3.2. Задание автоматов.

Конечные автоматы могут быть описаны следующими способами: аналитически, при помощи уравнений; таблицами переходов и выходов; с помощью графов.

**1-способ.** Работа автомата Z=(A,B,Q,$\varphi$,$\psi$) может быть описана системой **канонических уравнений**

$$\begin{cases} \boldsymbol{q}(t+1) = \varphi(\boldsymbol{q}(t), \boldsymbol{a}(t)) \\ \boldsymbol{b}(t) = \psi(\boldsymbol{q}(t), \boldsymbol{a}(t)) \end{cases},$$

которая должна быть дополнена начальным условием q(0)=q0.

**2-способ. Граф автомата** состоит из вершин, в которых записаны внутренние состояния $q_i$, ($q_i \in Q$). Вершины соединены дугами в соответствии с переходами автоматов. У автоматов Мили на дугах записываются в числителе входные сигналы (элементы из алфавита A), а в знаменателе выходные сигналы (элементы из алфавита B), которые соответствуют этим переходам. Если из одного состояния $q_i$ можно перейти в другое $q_k$, под воздействием нескольких входных сигналов, то в соответствующей дуге их можно объединять в виде дизъюнкции (символ $\vee$ - или).

**3-способ.** Функции $\varphi$ и $\psi$ можно представить в виде **общей таблицы переходов**, строки которых соответствуют состояниям (т.е. элементам из алфавита Q), а столбцы символам из входного алфавита A. В клетках этой таблицы записывается пара символов: в числителе записывается символ следующего состояния, а в знаменателе символ выходного сигнала.

**4-способ. Матрица соединения** автомата Z представляет собой квадратную таблицу в, которой номерам строк и столбцов соответствуют номера состояний (если состояния предварительно перенумеровать, начиная с 0). Клетка матрицы на пересечении i-й строки и j-го столбца заполняется дизъюнкцией пар "вход/выход", которая приписана дуге графа исходящей из i-й в j-ю вершину. При отсутствии такой дуги клетка остается свободной.

## §3.3. Общие задачи теории автоматов.

**1. Задача синтеза.** По данному отображению множества слов над алфавитом A в множество слов над алфавитом B требуется построить автомат Z, реализующий данное отображение.

**2. Задача анализа.** По данному графу конечного автомата Z найти отображение слов над его входным алфавитом A в множество слов над его выходным алфавитом B, которое реализует этот автомат.

**3. Задача декомпозиции.** Выяснить, при помощи соединения каких более простых автоматов, можно построить данный автомат Z и вычертить структурную **схему** соединения таких автоматов. В общем случае эта задача имеет неоднозначное решение и поэтому ищется декомпозиция, оптимальная в том или ином смысле. Например, с уменьшением числа состояний уменьшается и количество требуемых элементов памяти, что может привести к усложнению комбинационной схемы автомата.

## §3.4. Минимизация числа состояний автомата.

Минимизация основана на разбиении множества состояний на эквивалентные, с последующей заменой эквивалентных состояний одним состоянием. Более точно, пусть автоматы Z1 и Z2, находятся в состояниях q(i) и q(k) соответственно и под воздействием любой входной последовательности выдают одинаковые выходные последовательности. Такое отношение между состояниями одного и того же автомата или двух различных автоматов является отношением эквивалентности, а состояния являются эквивалентными (в противном случае они являются различимыми).

**Теорема 3.4.1.**

**1)** Состояния q(i) и q(k) автомата различимы, если различаются соответствующие им строки в таблице выходов (знаменатели в общей таблице переходов).

**2)** Состояния q(i) и q(k) автомата эквивалентны, если соответствующие им строки в общей таблице переходов одинаковы или становятся одинаковыми при замене каждого символа q(i), на символ q(k).

## §3.5. Структурный синтез.

Задачей структурного синтеза является построение схемы автомата из логических элементов. Структурный синтез, при котором используются элементарные автоматы с памятью и комбинационные

логические элементы, называется каноническим. Рассмотрим канонический метод структурного синтеза автоматов на элементах единичной задержки.

Для перехода от абстрактной модели к структурной необходимо закодировать входные и выходные сигналы, а также внутренние состояния. Можно поступить следующим образом. Элементы множеств A,B,Q по отдельности пронумеровать порядковыми числами, начиная с нуля. Потом каждому порядковому номеру поставить в соответствие его двоичный код. Следовательно, вместо использования любого конечного алфавита достаточно использовать алфавит {0,1}, что позволяет представить канонические уравнения автомата Z через булевы функции алгебры логики. Отметим, что при синтезе комбинационных схем из универсальных элементов в качестве исходных выражений часто используют минимальную ДНФ и минимальную КНФ.

Известно, что всякий автомат может быть реализован схемой над таким множеством, которое содержит схемы реализующие;

а) элемент единичной задержки;

б) элементы, порожденные функциями из некоторой полной в булевой алгебре системы.

## §3.6. Пример.

**Пример_3.6.1.** Построим автомат Мили, который «распознает» слова начинающиеся на «**а**» и оканчивающиеся на «**я**». Например, такие слова как акация, акция и т.п.

Опишем один из возможных автоматов.

1. **Входной алфавит** A = {а, я, δ, *}, где символ $\delta$ означает произвольную букву отличную от букв «а» и «я», а символ * - означает разделитель между словами (пробел).
2. **Множество состояний** Q={НС, ОНС, q(А), q(я)},

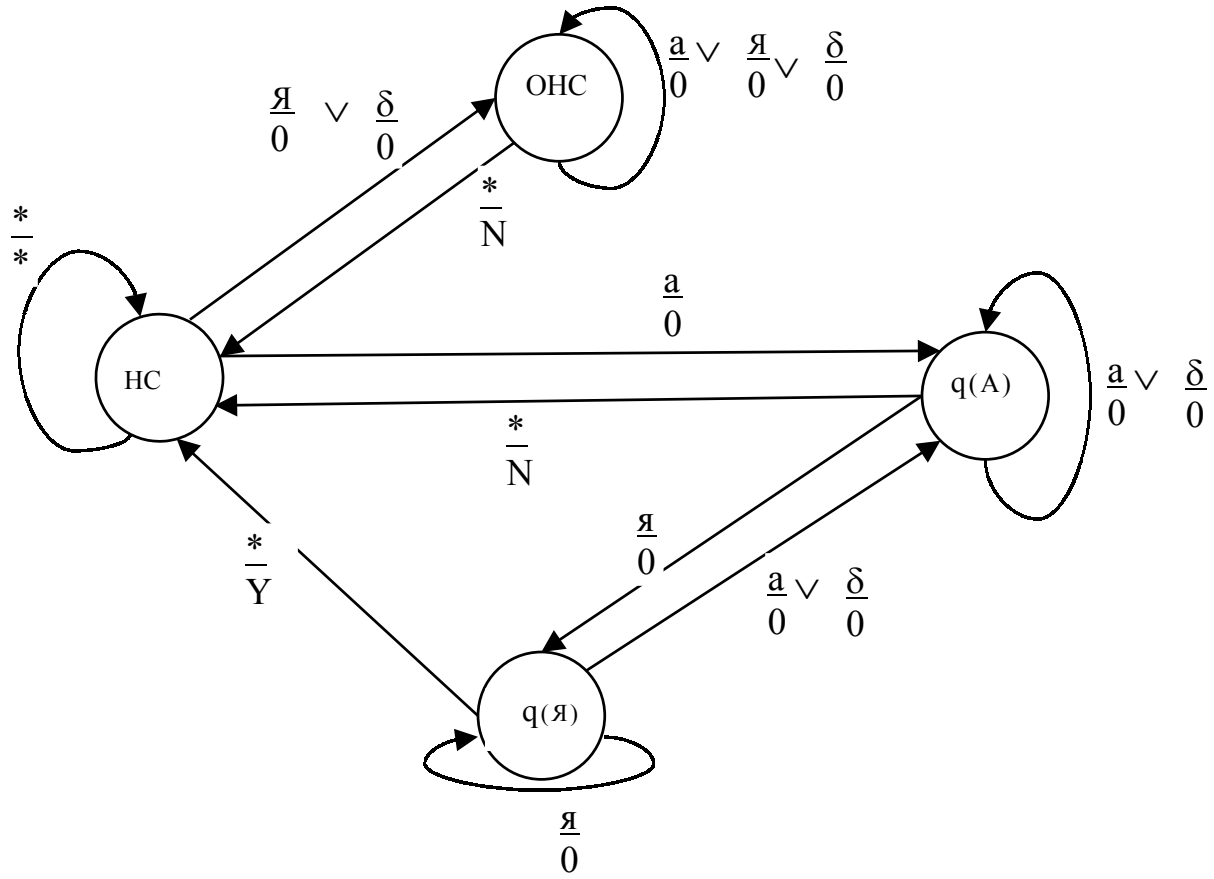
   где НС={новое слово}; ОНС={Ожидание нового слова};

   q(А)={первая буква нового слова – буква **а**};

   q(я)={последняя принятая (на данном шаге) буква – буква **я**}.
3. **Выходной алфавит** B={*, O, N, Y}, где

   *={разделитель между словами}; O={ожидание ответа т.е. слово еще не обработано}; N={слово не удовлетворяет требования, т.е. не **а…я**}; Y={слово начинается на **а** и заканчивается на **я**}.

Построим граф автомата Мили.




Общая таблица переходов (в числителе – следующее состояние, в знаменателе – выходной символ, другими словами числитель описывает функцию φ, а знаменатель функцию ψ.).

| a(t) / Q(t) | а | я | δ | * |
|---|---|---|---|---|
| НС | $\frac{q(A)}{0}$ | $\frac{OHC}{0}$ | $\frac{OHC}{0}$ | $\frac{HC}{*}$ |
| ОНС | $\frac{OHC}{0}$ | $\frac{OHC}{0}$ | $\frac{OHC}{0}$ | $\frac{HC}{N}$ |
| Q(А) | $\frac{q(A)}{0}$ | $\frac{q(Я)}{0}$ | $\frac{q(A)}{0}$ | $\frac{HC}{N}$ |
| Q(Я) | $\frac{q(A)}{0}$ | $\frac{q(Я)}{0}$ | $\frac{q(A)}{0}$ | $\frac{HC}{Y}$ |

Матрица соединения автомата (вход/выход).

| Q(t) \ Q(t+1) | НС | ОНС | q(А) | q(я) |
|---|---|---|---|---|
| НС | $\frac{*}{*}$ | $\frac{я}{0} \vee \frac{\delta}{0}$ | $\frac{a}{0}$ | – |
| ОНС | $\frac{*}{N}$ | $\frac{a}{0} \vee \frac{я}{0} \vee \frac{\delta}{0}$ | — | — |
| Q(А) | $\frac{*}{N}$ | — | $\frac{a}{0} \vee \frac{\delta}{0}$ | $\frac{я}{0}$ |
| Q(Я) | $\frac{*}{Y}$ | — | $\frac{a}{0} \vee \frac{\delta}{0}$ | $\frac{я}{0}$ |

Для построения канонических уравнений, преобразуем множества A, B, Q следующим образом: поставим каждому элементу число в двоичной форме записи (кодировка).

Коды для A: $A=\{(a_1, a_2) \mid a_1 \in \{0,1\}, a_2 \in \{0,1\}\}$.

$a \leftrightarrow 00$; $я \leftrightarrow 01$; $\delta \leftrightarrow 11$; $* \leftrightarrow 10$.

Коды для B: $B=\{(b_1, b_2) \mid b_1 \in \{0,1\}, a_2 \in \{0,1\}\}$.

$* \leftrightarrow 00$; $O \leftrightarrow 01$; $N \leftrightarrow 10$; $Y \leftrightarrow 11$.

Коды для Q: $Q=\{(q_1, q_2) \mid q_1 \in \{0,1\}, q_2 \in \{0,1\}\}$.

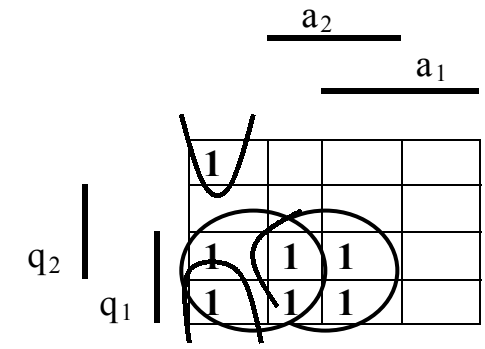
$НС \leftrightarrow 00$; $ОНС \leftrightarrow 01$; $q(A) \leftrightarrow 11$, $q(Я) \leftrightarrow 10$.

Теперь общая таблица переходов примет вид (в каждой клетке $\frac{b_1(t)\ b_2(t)}{q_1(t+1)\ q_2(t+1)}$)

| $q_1(t)$ $q_2(t)$ \ $a_1(t)$ $a_2(t)$ | 0 0 | 0 1 | 1 1 | 1 0 |
|---|---|---|---|---|
| 0 0 | $\frac{11}{01}$ | $\frac{01}{01}$ | $\frac{01}{01}$ | $\frac{00}{00}$ |
| 0 1 | $\frac{01}{01}$ | $\frac{01}{01}$ | $\frac{01}{01}$ | $\frac{00}{10}$ |
| 1 1 | $\frac{11}{01}$ | $\frac{10}{01}$ | $\frac{11}{01}$ | $\frac{00}{10}$ |
| 1 0 | $\frac{11}{01}$ | $\frac{10}{01}$ | $\frac{11}{01}$ | $\frac{00}{11}$ |

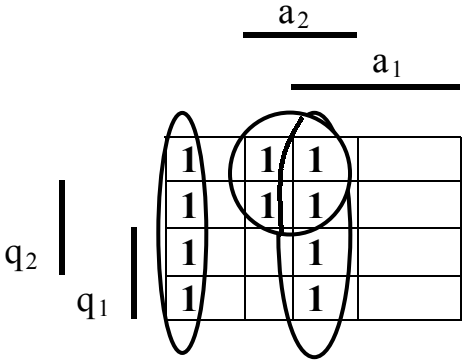
Выпишем карты Карно для соответствующих переменных:

**$b_1t$):**



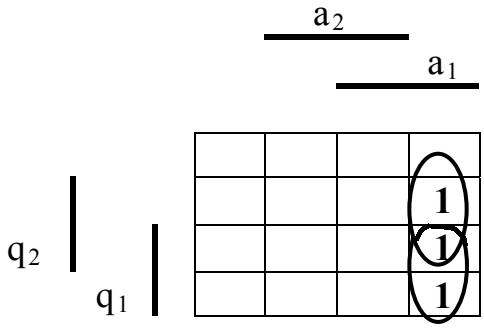

$$b_1(t)= q_1 \cdot a_2 \vee q_1 \cdot \overline{a_1} \vee \overline{q_2} \cdot \overline{a_1} \cdot \overline{a_2}$$

**$b_2(t)$:**



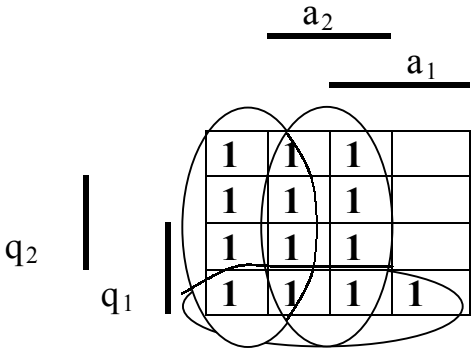

$$b_2(t)= \overline{q_1} \cdot a_2 \vee \overline{a_1} \cdot \overline{a_2} \vee a_1 a_2$$

$q_1(t+1)$




$$q_1(t+1) = q_1 \cdot a_1 \cdot \overline{a_2} \vee q_2 \cdot a_1 \cdot \overline{a_2}$$

$q_2(t+1)$:




$$q_2(t+1) = \overline{a_1} \vee a_2 \vee q_1 \cdot \overline{q_2}$$

Итак, система канонических уравнений имеет вид:

$$\begin{cases} b_1(t) = q_1(t) \cdot a_2(t) \vee q_1(t) \cdot \overline{a_1(t)} \vee \overline{q_2(t)} \cdot \overline{a_1(t)} \cdot \overline{a_2(t)} \\ b_1(t) = \overline{q_1(t)} \cdot a_2(t) \vee \overline{a_1(t)} \cdot \overline{a_2(t)} \vee a_1(t) a_2(t) \\ q_1(t+1) = q_1(t) \cdot a_1(t) \cdot \overline{a_2(t)} \vee q_2(t) \cdot a_1(t) \cdot \overline{a_2(t)} \\ q_2(t+1) = \overline{a_1(t)} \vee a_2(t) \vee q_1(t) \cdot \overline{q_2(t)} \end{cases}$$

С начальным состоянием q(0)=(0,0).

**Пример_3.6.2.** На автомат из примера_3.6.1. подается последовательность слов (входное слово):

«*трап*удав*ария*ток*акция*».

Выпишем выходное слово:

«*0000N0000N0000Y000N00000Y».

Из выходной последовательности видно, что третье и пятое, введенные в автомат слова, начинаются на «а» и заканчиваются на «я».

**Пример_3.6.3.** Построить граф автомата Мура для примера_3.6.1.

Решение. Граф автомата Мура изображен на рис.3.6.1.

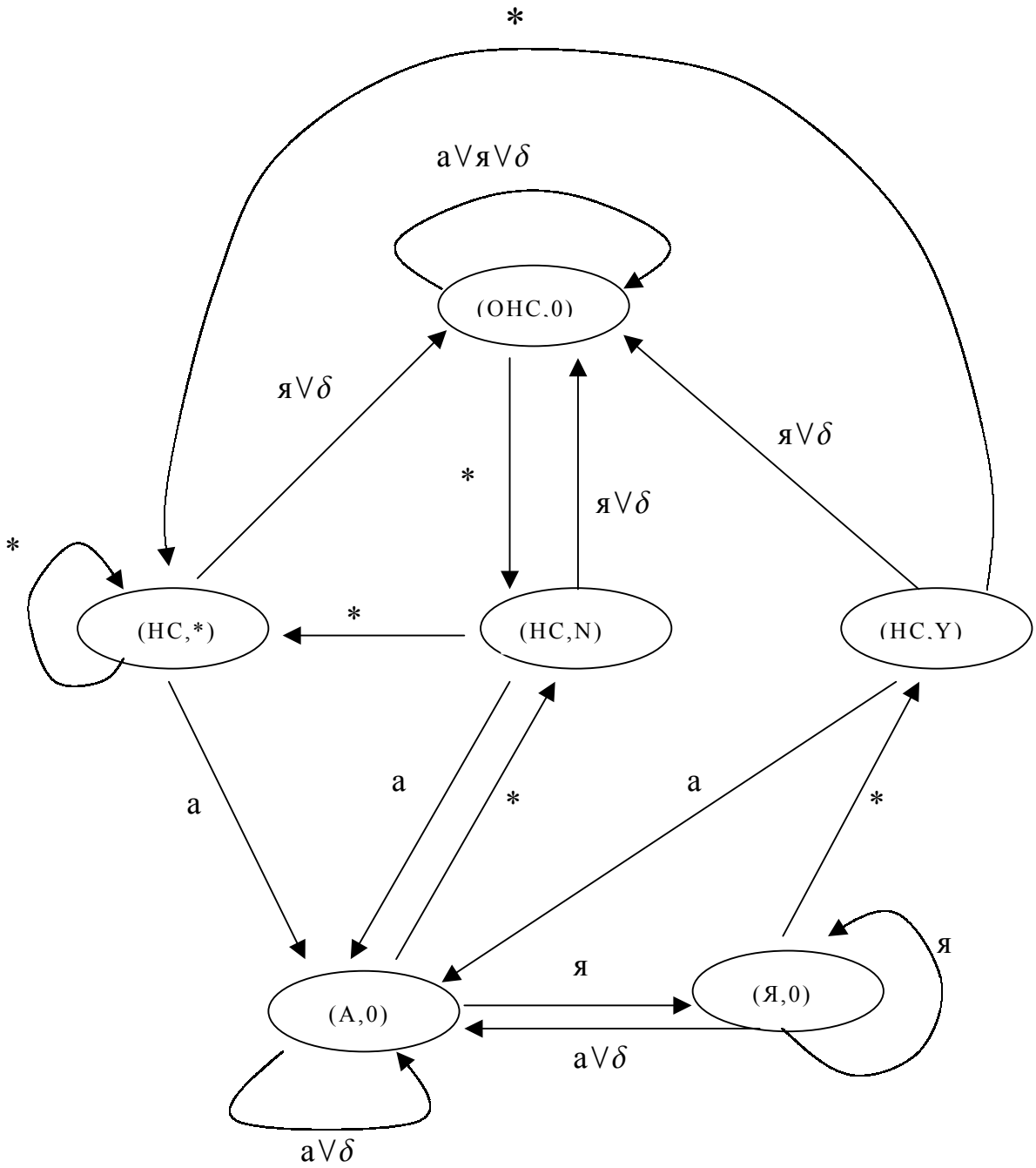



**Рис.3.6.1. Граф автомата Мура.**

# Раздел 4. Элементы теории алгоритмов.

## §4.1. Определение алгоритма.

Характерной чертой современности является широкое использование компьютеров при решении проблем (задач) в различных областях человеческой деятельности. Однако предварительно задача должна быть решена алгоритмически, т.е. должно быть задано формальное предписание, следуя которому можно получить итоговый результат для решения всех задач определенного типа (это интуитивное, не строгое понятие алгоритма).

Отметим, что если наличие алгоритма само по себе служит доказательством существования алгоритма, то для доказательства его отсутствия необходимо иметь строгое математическое определение алгоритма.

Попытки формализовать понятие алгоритма привели к созданию **машины Тьюринга**, как некоего воображаемого устройства, реализующего алгоритм. Ещё одним шагом в определении понятия алгоритма стало появление **рекурсивных функций,** как функций, формализующих понятие алгоритма и реализующих интуитивное понятие вычислимости. Вскоре было установлено, что множество рекурсивных функций совпадает с множеством функций, вычислимых на машинах Тьюринга. Появлявшиеся затем новые понятия, претендующие на объяснение понятия алгоритма, оказывались эквивалентными функциям, вычислимым на машинах Тьюринга, а значит, и рекурсивным функциям. Итогом развернувшейся дискуссии о том, что такое алгоритм, стало утверждение, называемое сейчас тезисом Чёрча.

**Тезис Чёрча.** Понятие алгоритма, или вычислимости некоторым механическим устройством, совпадает с понятием вычислимости на машинах Тьюринга

Это утверждение нельзя считать математической теоремой. Это есть некоторый естественнонаучный тезис, принятый большинством исследователей.

## §4.2. Машина Тьюринга.

Машина Тьюринга представляет собой (абстрактное) устройство, состоящее из ленты, управляющего устройства и считывающей головки.

Лента разбита на ячейки. Во всякой ячейке в точности один символ из **внешнего алфавита** $A=\{a_0,a_1,\ldots,a_n\}$. Некоторый символ (будем обозначать его $\Lambda$) алфавита A называется пустым, а любая

ячейка, содержащая в данный момент пустой символ, называется пустой ячейкой (в этот момент). Лента предполагается потенциально неограниченной в обе стороны.

**Управляющее устройство** в каждый момент времени находится в некотором состоянии $q_j$, принадлежащем множеству $\mathbf{Q=\{q_0, q_1,..., q_m\}}$ (m=l). Множество Q называется **внутренним алфавитом**. Одно из таких состояний (обычно $\mathbf{q_0}$) называется заключительным, а некоторое другое (обычно $\mathbf{q_1}$) -начальным.

Считывающая головка перемещается вдоль ленты так, что в каждый момент времени она обозревает ровно одну ячейку ленты. Головка может считывать содержимое обозреваемой ячейки и записывать в нее вместо обозреваемого символа некоторый новый символ из внешнего алфавита **A** (возможно тот же самый).

В процессе работы управляющее устройство в зависимости от состояния, в котором оно находится и символа, обозреваемого головкой, изменяет свое внутреннее состояние **q**. Затем выдает головке приказ напечатать в обозреваемой ячейке определенный символ из внешнего алфавита **A,** а потом приказывает головке либо остаться на месте, либо сдвинуться на одну ячейку вправо, либо сдвинуться на одну ячейку влево. Попав в заключительное состояние, машина прекращает работу.

**Конфигурацией на ленте (или машинным словом)** называется совокупность, образованная:

1) последовательностью $\mathbf{a_{i(1)}, a_{i(2)},..., a_{i(s)}}$ символов из внешнего алфавита **A**, записанных в ячейках ленты, где $\mathbf{a_{i(1)}}$.-символ записанный в первой ячейке слева и т.д. (любая такая последовательность называется **словом),**

2) состоянием внутренней памяти $q_r$;

3) номером **k** воспринимаемой ячейки.

Конфигурацию машины будем записывать так:

$$a_{i(1)}, a_{i(2)},..., a_{i(r-1)}\ \frac{a_{i(r)}}{q_r}\ a_{i(r+1)}, a_{i(r+2)},..., a_{i(s)}$$

здесь **r**-воспринимаемая ячейка, выделяется в виде дроби.

Если машина, находясь во внутреннем состоянии $\mathbf{q_i}$ , воспринимает ячейку с символом $\mathbf{a_u}$ , записывает к следующему моменту времени в эту ячейку символ $\mathbf{a_r}$ , переходит во внутреннее состояние $\mathbf{q_s}$ и сдвигается по ленте, то говорят, что машина выполняет

команду $q_i a_u \to q_s a_r S$ , где символ S может принять одно из значений: -1 – сдвиг головки влево; +1 - сдвиг головки вправо; 0 – головка остается на месте. Список всех команд (пятерок), определяющих работу машины Тьюринга, называется **программой** этой машины. Программу машины часто задают в виде таблицы. Так для описанной выше ситуации в таблице на пересечении строки $a_i$ и столбца $q_i$ будет стоять $q_s a_r S$ ( см. таб.4.2.1)

**Таб.4.2.1.**

| | $q_0$ | … | $q_i$ | … | $q_m$ |
|---|---|---|---|---|---|
| … | | | | | |
| $a_u$ | | | $q_s a_r S$ | | |
| … | | | | | |

Если в программе машины для пары $(q_i, a_u)$ пятерка отсутствует, то в таблице на пересечении строки $a_u$, и столбца $q_i$ ставится прочерк.

Итак, **машина Тьюринга есть, по определению**, набор **M=(A,Q,П)**, где **A** — внешний алфавит, **Q** — алфавит внутренних состояний, **П** — программа.

Если машина, начав работу с некоторым словом P, записанным на ленте, придет в заключительное состояние, то она называется **применимой к этому слову**. Результатом ее работы считается слово, записанное на ленте в заключительном состоянии. В противном случае, машина называется не применимой к слову P.

**Пример** 4.2.1. Построим машину Тьюринга, складывающую натуральные числа, записанные в унарной системе счисления (т.е. записанные при помощи одного символа **|.** Например 5=|||||.).

Решение. Рассмотрим алфавит $A = \{|, +, \wedge\}$.

Машина определяется следующей программой:

| | $q_1$ | $q_2$ | $q_3$ |
|---|---|---|---|
| \| | $\| q_1 + 1$ | $\wedge q_3 - 1$ | $\| q_3 - 1$ |
| + | $\| q_1 + 1$ | | |
| $\wedge$ | $\wedge q_2 - 1$ | | $\wedge q_0 + 1$ |

Выпишем последовательно возникающие при работе этой машины конфигурации на исходном слове ||+|||, т.е. 2+3. Здесь при

записи конфигурации будем использовать следующее соглашение: состояние, в котором находится машина, записывается в круглых скобках справа за обозреваемой буквой.

0) $\cdots\wedge|(q_1)|+|||\wedge\ldots$
1) $\cdots\wedge||(q_1)+|||\wedge\ldots$
2) $\cdots\wedge||+(q_1)|||\wedge\ldots$
3) $\cdots\wedge||||(q_1)||\wedge\ldots$
4) $\cdots\wedge|||||(q_1)|\wedge\ldots$
5) $\cdots\wedge||||||(q_1)\wedge\ldots$
6) $\cdots\wedge||||||\wedge(q_1)\ldots$
7) $\cdots\wedge||||||(q_2)\wedge\ldots$
8) $\cdots\wedge||||||(q_3)\wedge\ldots$
9) $\cdots\wedge||||(q_3)|\wedge\ldots$
10) $\cdots\wedge|||(q_3)||\wedge\ldots$
11) $\cdots\wedge||(q_3)|||\wedge\ldots$
12) $\cdots\wedge|(q_3)||||\wedge\ldots$
13) $\cdots\wedge(q_3)|||||\wedge\ldots$
14) $\cdots\wedge|(q_0)||||\wedge\ldots$

◄

**Пример 4.2.2.** Построить машину Тьюринга, удваивающую натуральные числа, записанные в унарной системе счисления.

**Решение.** Искомую машину построим в алфавите A={|, α, ∧}. Программа такой машины может выглядеть так:

| | $q_1$ | $q_2$ | $q_3$ |
|---|---|---|---|
| \| | $\alpha\, q_1 + 1$ | $\| q_2 - 1$ | $\| q_3 + 1$ |
| α | | $\| q_3 + 1$ | |
| ∧ | $\wedge\, q_2 - 1$ | $\wedge\, q_0 + 1$ | $\| q_2 - 1$ |

Применим полученную машину к слову ||.

0) $\cdots\wedge|(q_1)|+\ldots$
1) $\cdots\wedge\alpha|(q_1)\wedge\ldots$
2) $\cdots\wedge\alpha\alpha\wedge(q_1)\ldots$
3) $\cdots\wedge\alpha\alpha(q_2)\wedge\ldots$
4) $\cdots\wedge\alpha|\wedge(q_3)\ldots$
5) $\cdots\wedge\alpha|(q_2)|\wedge\ldots$
6) $\cdots\wedge\alpha|(q_2)|\wedge\ldots$
7) $\cdots\wedge\alpha(q_2)||\wedge\ldots$
8) $\cdots\wedge|(q_3)||\wedge\ldots$
9) $\cdots\wedge|(q_2)|||\wedge\ldots$
13) $\cdots\wedge|(q_2)|||\wedge\ldots$
14) $\cdots\wedge(q_2)||||\wedge\ldots$
15) $\cdots\wedge|(q_0)|||\wedge\ldots$

◄

Введение новой буквы α и замена исходных | на α позволяет различить исходные | и новые (приписанные) |. Состояние $\mathbf{q_1}$ обеспечивает замену | на α, состояние $\mathbf{q_2}$ обеспечивает поиск α, предназначенных для замены на |, и останов машины в случае, когда α не обнаружено, $\mathbf{q_3}$ обеспечивает дописывание | в случае, когда произошла замена α на |.

# Раздел 5. Комбинаторика

## §5.1. Основные комбинаторные функции

**Определение.** Функция f(n), для которой f(0)=1, f(n+1)=(n+1)f(n) при всех целых неотрицательных n, называется **n−факториалом** и обозначается n!

Таким образом: 0!=1, 1!=1, 2!=1·2, …, n! = 1·2·…·n.

Для приближенного вычисления n! при больших n справедлива формула Стирлинга:

$$n! \approx \left(\frac{n}{e}\right)^n \sqrt{2\pi n}.$$

**Определение**. Для всех целых неотрицательных **n** и **k** функция

$$C_n^k = \begin{cases} \dfrac{n!}{k!(n-k)!}, & для \quad 0 \le k \le n \\ 0, & для \quad 0 \le n < k \end{cases}$$

называется **биномиальным коэффициентом.** Читается: Це из n(эн) по k(ка). Область определения биномиальных коэффициентов можно расширить. А именно, для всех действительных ***a*** и для всех целых k≥0 функция

$$\binom{a}{k} = \begin{cases} \dfrac{a(a-1)(a-2)...(a-k+1)}{k!}, & для \quad k > 0 \\ 1, & для \quad k = 0 \end{cases}$$

также называется **биномиальным коэффициентом.**

**Примеры:**

1) $4! = 1\cdot 2\cdot 3\cdot 4 = 24$

2) $C_5^3 = \dfrac{5!}{3!(5-3)!} = \dfrac{1\cdot 2\cdot 3\cdot 4\cdot 5}{1\cdot 2\cdot 3\cdot 1\cdot 2} = \dfrac{4\cdot 5}{2} = 10$

3) $\binom{-\frac{1}{2}}{2} = \dfrac{-\frac{1}{2}\left(-\frac{1}{2}-1\right)}{2!} = \dfrac{3}{8}$

Свойства биномиальных коэффициентов

1. Для целых ***a*** имеем $C_a^k = \binom{a}{k}$.

2. $C_a^k = C_n^{n-k}, n, k$ - целые неотрицательные числа.

3. Для действительных ***a*** имеет место соотношение

$$\binom{a}{k} + \binom{a}{k+1} = \binom{a+1}{k+1}$$

**Определение**. Определим для всех натуральных n и всех наборов неотрицательных целых чисел [$k_1$, $k_2$, …, $k_r$], для которых $k_1 + k_2 + \ldots + k_r = n$, функцию

$$C_n(k_1, k_2, \ldots, k_r) = \frac{n!}{k_1! k_2! \ldots \cdot k_r!},$$

которая называется **полиномиальным коэффициентом.**

Примеры.

$$C_6(2,0,4) = \frac{6!}{2!0!4!} = 15;$$

$$C_6(1,1,3,1) = \frac{6!}{1!1!3!1!} = 4 \cdot 5 \cdot 6 = 120.$$

## §5.2. Формулы бинома и полинома

Для всех действительных ***a, b*** и всех натуральных **n** справедлива формула бинома Ньютона:

$$(a+b)^n = \sum_{k=0}^{n} C_n^k a^k b^{n-k}.$$

Утверждения:

$$1) \quad \sum_{k=0}^{n} C_n^k = 2, \quad при \quad a = b = 1$$

$$2) \quad \sum_{k=0}^{n} (-1)^k C_n^k = 0, \quad при \quad a = -1, b = 1.$$

Для любых отличных от нуля действительных чисел $a_1, a_2, \ldots, a_r$ и любого натурального n справедлива формула полинома:

$$(a_1 + a_2 + \ldots + a_r)^n = \sum_{k_1 + k_2 + \ldots + k_r = n} C_n(k_1, k_2, \ldots, k_r) \cdot a_1^{k_1} \cdot a_2^{k_2} \cdot \ldots \cdot a_r^{k_r}$$

где суммирование распространяется на все наборы неотрицательных чисел ($k_1$,$k_2$,…,$k_r$), для которых $k_1 + k_2 + \ldots + k_r = n$.

**Пример.**

$(a+b+c)^3 = C_3(3,0,0)a^3 + C_3((2,1,0))a^2b + C_3(2,0,1)a^2c + C_3(1,2,0)ab^2 + C_3(1,0,2)ac^2 + C_3(1,1,1)abc + C_3(0,3,0)b^3 + C_3(0,2,1)b^2c + C_3(0,1,2)bc^2 + C_3(0,0,3)c^3$.

## §4.3. Перестановки. Размещения. Сочетания

Каждая последовательность k различных предметов с учетом порядка называется перестановкой этих предметов. Из определения следует, что перестановки из k элементов множества M отличаются друг от друга только порядком входящих в них элементов.

**Утверждение 5.3.1.** Число $P_k$ всех перестановок из k различных элементов равно: $P_k = k!$

Пример1. Сколькими способами можно расставить на полке 5 различных книг? Ответ: $5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120$ - различных способов.

Пример 2. Число всех подстановок (взаимно- однозначных отображений конечного множества, состоящего из n элементов, на себя) равно n!.

Пусть множество M состоит из элементов: $x_1, \ldots, x_n$. Выборкой объема r из n элементов, или (n,r)- выборкой, назовем некоторый набор элементов $x_{i_1}, x_{i_2}, \ldots, x_{i_r}$ из множества X. Организацию выборки можно представить в виде урновой схемы следующим образом. Так как множество X содержит конечное число элементов, то их можно перенумеровать, а затем поставить во взаимнооднозначное соответствие с перенумерованными соответствующим образом шарами. Например, элементу с номером 1 соответствует шар с таким же номером. Шары помещаются в урну, перемешиваются, а затем случайным образом последовательно вынимается r шаров. Причем здесь возможны два варианта. Первый вариант - шар, вынутый из урны, обратно в урну не возвращается. Таким образом, получается выборка без повторения, а соответствующая урновая схема называется урновой схемой без возвращения. Второй вариант - вынутый на определенном шаге шар, после запоминания номера, возвращается обратно в урну, и на следующем шаге опять может быть вынут из урны (урновая схема с возвращением). Получаемая таким образом выборка называется (n,r)- выборкой с повторениями.

Выборка называется упорядоченной, если порядок следования элементов в ней задан, т.е. является существенным; в противном случае выборка называется неупорядоченной.

**Определение.** Упорядоченная (n,r)- выборка, в которой элементы могут повторяться, называется (n,r)- **размещением с повторением.** Если элементы упорядоченной (n,r)- выборки попарно различны, то она называется (n,r) - **размещением без повторения** (или просто (n,r)- размещением).

**Определение.** Неупорядоченная (n,r)- выборка с повторение называется (n,r)- **сочетанием с повторением.** Если элементы неупорядоченной (n,r)- выборки попарно различны, то она называется (n,r)- **сочетанием без повторений** (или просто (n,r)- сочетанием).

**Утверждение 5.3.2.**

а) Число (n,r)- размещений с повторением $\overline{A}$ равно:

$$\overline{A}_n^r = n^r .$$

б) Число (n,r)- размещений (обозначим $A_n^r$) равно:

$$A_n^r = \frac{n!}{(n-r)!} = n(n-1)\cdot\ldots\cdot(n-r+1).$$

**Утверждение 5.3.3.**

а) Число (n,r)- сочетаний с повторениями (обозначим $\overline{C}_n^r$ ) равно:

$$\overline{C}_n^r = C_{n+r+1}^r$$

б) Число (n,r)- сочетаний равно $C_n^r$.

**Примеры.**

1) Сколькими способами можно выбрать 5 номеров из 36? Нас интересует количество неупорядоченных (36,5)−сочетаний. По утверждению 5.3.3(б) их $C_{36}^5$ , что равно 376992.

2) Имеется $A_8^3 = 336$ различных способов распределения трех первых мест при восьми командах, участвующих в турнире.

3) Имеется 30 монет достоинством 1, 2, 3 у.е. Сколько существует различных комбинаций монет (например, 20 монет - по 1 у.е., 5- по 2 у.е., 5- по 3 у.е.)? Представим урновую схему, состоящую из трех шаров с номерами 1, 2, 3, что соответствует достоинству монет. Выборка с возвращением. Количество неупорядоченных (3,30)-выборок с повторением, по утверждению_5.3.3(а) равно

$$C_{3+30-1}^{30} = C_{32}^{30} = \frac{32\cdot 31}{2} = 496.$$

## §4.4. Правило суммы, произведения

Если объект X может быть выбран n способами, а объект Y - другими m способами, то выбор "либо X, либо Y" (но не оба сразу) может быть осуществлен n+m способами. Это так называемое правило суммы, которое очевидным образом может быть перенесено на более чем два объекта.

Правило произведения. Если объект X может быть выбран n способами, и после каждого из таких выборов объект Y в свою очередь, в независимости от выбора X может быть выбран m способами, то выбор упорядоченной пары (X,Y) может быть осуществлен mn способами.

Пример. В скольких случаях при игре в "Спортлото" (5 из 36) будут правильно выбраны а) ровно 4 номера; б) ровно 5 номеров; в) не менее 4 номеров.

Решение. а) Предположим, что у нас имеется две урны, в которых содержится 5 "выигрышных" номеров, а в другой содержится 31 "невыигрышных" номеров. 4 "выигрышных" шара из первой урны можно выбрать n= $C_5^4$ способами (объект X), а один оставшийся "невыигрышный" шар – $m = C_{31}^1$ (объект Y). Далее по правилу произведения получаем, что искомое число равно mn= $C_5^4 \cdot C_{31}^1 = 155;$

б) $C_5^5 \cdot C_{31}^0 = 1;$

в) Не менее четырех номеров может быть угадано, когда либо будет угадано ровно 4 номера (объект X), либо будет угадано ровно 5 номеров (объект Y). Тогда по правилу суммы, используя результаты п. (а), (б), получаем: 155+1=156.

## ЛИТЕРАТУРА.


1. Ф.А.Новиков. Дискретная математика для программистов./ Санкт-Петербург: Питер, 2001г., 304С.
2. С.В.Судоплатов, Е.В.Овчинникова. Элементы Дискретной математики./ М., ИНФРА-М, Новосибирск, Изд-во НГТУ, 2002г., 208С.
3. Я.М.Ерусалимский. Дискретная математика/ М., «Вузовская книга», 2001г.,279С.
4. А.Ахо, Дж.Хопкрофт, Дж.Ульман. Построение и анализ вычислительных алгоритмов. / М., Мир, 1979г., 536С.
5. В.Н.Нефедов, В.А.Осипова Курс дискретной математики./ М., Изд-во МАИ, 1992г., 264С.
6. Д.Кук, Г.Бейз. Компьютерная математика./ М., Наука, 1990, 384С.
7. В.А.Евстигнеев. Применение теории графов в программировании / М., Наука, 1985.
8. Д.Кнут. Искусство программирования для ЭВМ. / М., Мир, 1977, т.1,2.