

Федеральное агентство по образованию

**ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ  
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)**

Кафедра автоматизации обработки информации

Утверждаю:

Зав. каф. АОИ

профессор

\_\_\_\_\_ Ю.П. Ехлаков

«\_\_» \_\_\_\_\_ 2007 г.

**Методические указания**

к выполнению практических работ по дисциплине

*«Дискретная математика»*

для студентов специальности 230102 –

«Автоматизированные системы обработки информации и управления»

Разработчики:

ст. преподаватель каф. АОИ

\_\_\_\_\_ З.А. Смыслова

ст. преподаватель каф. АОИ

\_\_\_\_\_ Н.В. Пермякова

**Содержание**

Практическое занятие №1 .....	3
Практическое занятие № 2 .....	6
Практическое занятие № 3 .....	9
Практическое занятие № 4 .....	17
Практическое занятие № 5 .....	21
Практическое занятие № 6 .....	22
Рекомендуемая литература .....	28

## **Практическое занятие № 1. «Получение булеана конечного множества»**

### ***1.1. Цель работы***

Программно реализовать один из алгоритмов генерации булеана конечного множества.

### ***1.2. Определение булеана конечного множества***

Булеаном множества  $M$  называется множество всех подмножеств множества  $M$ . Для конечного множества мощности  $n$  мощность булеана равна  $2^n$ .

### ***1.3. Алгоритмы генерации булеана конечного множества***

#### ***1.3.1. Алгоритм генерации всех подмножеств $n$ -элементного множества***

В памяти компьютера целые числа представляются кодами в двоичной системе счисления, причем число  $2^n - 1$  представляется кодом, содержащим  $n$  единиц.

Тогда, число 0 является представлением пустого множества, число 1 является представлением подмножества, состоящего из первого элемента и т.д..

Описанный ниже алгоритм перечисляет все двоичные коды чисел от 0 и до  $2^n - 1$ , т.е. все подмножества конечного множества мощности  $n$ .

1. Задать  $n$ , и множество  $A$ , состоящее из  $n$  элементов

2. Цикл  $(i = \overline{0; 2^n - 1})$

2.1.  $M$  – пустое подмножество

2.2. Цикл  $(j = \overline{0; n - 1})$

2.2.1. Получить значение  $j$ -го разряда числа  $i$

2.2.2. Если полученное значение равно 1, то включить в подмножество  $M$  элемент  $A[j]$

2.3. Конец цикла

2.4. Вывести полученное подмножество  $M$ .

2.5. Конец цикла

3. Конец

### 1.3.2. Алгоритм построения бинарного кода Грея

Описанный далее алгоритм генерирует последовательность всех подмножеств  $n$ -элементного множества таким образом, что каждое последующее множество получается из предыдущего добавлением или удалением одного элемента.

Код Грея называется так же отраженным кодом. Рассмотрим построение кода на примере  $n = 4$ .

Будем считать старшим разрядом нулевой разряд. Он может принимать значения 0 и 1.

0000

0001 Далее старший разряд первый, который принимает значения 1, а младший разряд (нулевой) принимает значения в обратном порядке от предыдущего

0011

0010 Далее аналогичным образом: (старший разряд выделен размером, отражаемая часть жирным шрифтом)

0110

0111

0101

0100

1100

1101

1111

1110

1010

1011

1001

Пронумеруем полученные наборы от 0 до 14.

0	0000	7	0100
1	0001	8	1100
2	0011	9	1101
3	0010	10	1111
4	0110	11	1110
5	0111	12	1010
6	0101	13	1011
		14	1001

В первом наборе инвертировался разряд с номером 0, во втором – разряд с номером 1, в третьем – разряд с номером 0, в четвертом разряд с номером 2 и т.д.. Разложим числа от 1 до 14 на простые множи-

тели и подсчитаем количество двоек в разложении числа. В итоге получена та же самая последовательность.

Число	Разложение	Количество двоек в разложении числа	Число	Разложение	Количество двоек в разложении числа
1	1	0	8	$2*2*2*2$	3
2	2	1	9	$3*3$	0
3	3	0	10	$2*5$	1
4	$2*2$	2	11	11	0
5	5	0	12	$2*2*3$	2
6	$2*3$	1	13	13	0
7	7	0	14	$2*7$	1

1. Задать  $A$  – множество из  $n$  элементов.
2. Задать  $M = [000..]$ - подмножество булеана.
3. Вывести  $M$ ;
4. Цикл  $(i = \overline{1; 2^n - 1})$ 
  - 4.1. Найти  $k$  – количество двоек в разложении числа  $i$  ;
  - 4.2. Если  $M[k] = 0$  То  $\{M[k] = 1$   
Иначе  $M[k] = 0$ ;
  - 4.3. Вывести  $M$ ;
5. Конец цикла
6. Конец

### 1.3.3. Реализация кода Грея с помощью стека.

1. СТЕК  $\leftarrow$  пустой стек
2. Цикл  $(j = \overline{n-1; 0})$ 
  - 2.1.  $g_j = 0$
  - 2.2. СТЕК  $\leftarrow j$
3. Конец цикла
4. Печать  $(g_{n-1}, g_{n-2}, \dots, g_0)$
5. Пока (СТЕК не пуст);
  - 5.1. СТЕК  $\rightarrow a$
  - 5.2.  $g_a := \overline{g_a}$  {Инвертировать  $g_a$  }
  - 5.3. Печать  $(g_{n-1}, g_{n-2}, \dots, g_0)$
  - 5.4. Цикл  $(j = \overline{a; 0})$

5.4.1. СТЕК  $\leftarrow j$

5.5. Конец цикла

6. Конец цикла

### ***1.4. Задание на выполнение***

Дано множество  $A$ , состоящее из  $n$  элементов. Значение  $n$  и значения элементов множества вводятся с клавиатуры. Предусмотреть обработку некорректно введенных данных. Написать программу, выводящую на экран булеан множества  $A$ .

1. Использовать алгоритм генерации всех подмножеств.
2. Использовать алгоритм получения кода Грея.
3. Использовать реализацию кода Грея с помощью стека. Стек организовать в виде массива.
4. Использовать реализацию кода Грея с помощью стека. Стек организовать в виде линейного однонаправленного списка.

## **Практическое занятие № 2. «Представление множеств упорядоченными списками»**

### ***1.1. Цель работы***

Программно реализовать представление множеств упорядоченными списками и основные операции над множествами.

### ***1.2. Представление множеств упорядоченными списками***

Если рассматриваемое множество не велико, то с точки зрения экономии памяти, множества достаточно часто представляются в виде списков элементов. Элемент списка в этом случае представляется записью с двумя полями: информационным и указателем на следующий элемент. Весь список описывается указателем на первый элемент.

Эффективная реализация операций над множествами, представленными в виде упорядоченных списков, основана на общем алгоритме типа слияния. Общая идея алгоритма типа слияния состоит в следующем: алгоритм параллельно просматривает два множества, пред-



// в этом случае  $Pa.i = Pb.i$ , можно взять любой  
// элемент

2.2. Добавить  $d$  в конец списка  $c$ .

3. Конец цикла

4. Если ( $Pa \neq NULL$ )

То добавить в конец списка  $c$  оставшиеся элементы из  $A$

5. Если ( $Pb \neq NULL$ )

То добавить в конец списка  $c$  оставшиеся элементы из  $B$

6. Конец

### 1.5. Вычисление пересечения слиянием

Даны пересекаемые множества  $A$  и  $B$ , которые заданы указателями  $a$  и  $b$ .

1.  $Pa = a; Pb = b; c = NULL;$

2. Пока ( $Pa \neq NULL$  and  $Pb \neq NULL$ )

2.1. Если ( $Pa.i < Pb.i$ ) // сравнение текущих информационных  
// полей.

То  $Pa = Pa.n$  // элемент множества  $A$  не принадлежит  
// пересечению

Иначе Если ( $Pa.i > Pb.i$ )

То  $Pb = Pb.n$  // элемент множества  $B$  не  
// принадлежит пересечению

Иначе  $d = Pb.i; Pa = Pa.n; Pb = Pb.n$  // в этом случае  
//  $Pa.i = Pb.i$ , можно взять любой элемент

2.2. Добавить  $d$  в конец списка  $c$ .

3. Конец цикла

4. Конец

### 1.6. Задание на выполнение

Даны два множества  $A$  и  $B$ . Организовать представление множеств в виде линейных однонаправленных списков. Мощность множеств и элементы множеств задавать с клавиатуры. В программе выполнить проверку списка на упорядоченность и на уникальность элементов.

1. Проверить, включено ли множество  $A$  во множество  $B$ .

2. Найти пересечение множеств  $A$  и  $B$ .

3. Найти объединение множеств  $A$  и  $B$ .



## Практическое занятие №3. «Алгоритмы порождения комбинаторных объектов»

### 1.1. Цель работы

Ознакомиться с алгоритмами, генерирующими комбинаторные объекты и программно реализовать их.

### 1.2. Генерация сочетаний

#### 1.2.1. Генерация сочетаний в лексикографическом порядке

Будем рассматривать в качестве множества  $X = \{1, 2, \dots, n\}$ . Требуется сгенерировать все подмножества мощности  $k$ , ( $0 \leq k \leq n$ ) множества  $X$ .

Определим отношение лексикографического порядка ( $E$ ) следующим образом. Пусть  $a = (a_1, a_2, \dots, a_n)$ ,  $b = (b_1, b_2, \dots, b_m)$ . Будем говорить, что набор  $a$  предшествует набору  $b$ :  $a E b \Leftrightarrow \exists r \geq 1 : a_r < b_r$  и  $\forall i = \overline{1, r-1} ; a_i = b_i$ .

Будем рассматривать сочетания  $k$  элементов из множества  $X$  как вектор  $(c_1, c_2, \dots, c_k)$ , компоненты которого расположены в порядке возрастания слева направо (т.е.  $c_i < c_{i+1}$  для любого  $i$ ). Начиная с сочетания  $(1, 2, \dots, k)$ , следующие будем строить, просматривая текущее справа налево, чтобы найти самый первый элемент, не достигший максимального значения; этот элемент увеличим на единицу, а всем элементам справа от него присвоим номинальные наименьшие значения.

Лексикографический порядок порождения сочетаний не является алгоритмом с минимальными изменениями.

1.  $c_0 := -1$
2. Цикл ( $i := \overline{1, k}$ )
  - 2.1.  $c_i := i$
3. Конец цикла
4.  $j := 1$
5. Пока ( $j \neq 0$ )
  - 5.1. Печать  $(c_1, c_2, \dots, c_k)$
  - 5.2.  $j := k$

5.3. Пока ( $c_j = n - k + j$ )

5.3.1  $j := j - 1$

5.4. Конец цикла

5.5.  $c_j := c_j + 1$

5.6. Цикл ( $i := \overline{j+1, k}$ )

5.6.1.  $c_i := c_{i-1} + 1$

5.7. Конец цикла

6. Конец цикла

7. Конец

### 1.2.2. Генерация сочетаний с помощью кодов Грея

При генерации сочетаний из  $n$  элементов по  $k$  наименьшим возможным изменением при переходе от текущего сочетания к следующему является замена одного элемента другим. В терминах Грея это означает, что мы хотим выписать все  $n$ -разрядные кодовые слова, содержащие ровно  $k$  единиц, причем последовательные наборы отличаются ровно в двух разрядах (в одном из разрядов 0 заменяется на 1, а в другом — 1 на 0).

Пусть  $G(n)$  — двоично-отраженный код Грея, а  $G(n, k)$  ( $0 \leq k \leq n$ ) — последовательность кодовых слов ровно с  $k$  единицами:

$$G(n, k)^T = \left( G(n, k)_1, G(n, k)_2, \dots, G(n, k)_{C_n^k} \right)^T.$$

Эту последовательность можно рекурсивно определить следующим образом:

$$G(n, 0) = (0 \ 0 \ \dots \ 0);$$

$$G(n, n) = (1 \ 1 \ \dots \ 1);$$

$$G(n, k) = \begin{pmatrix} 0 & G(n-1, k) \\ 1 & G(n-1, k-1) \end{pmatrix}, \quad (1.1)$$

где  $0$  — вектор-столбец размерности  $C_{n-1}^k \times 1$ , состоящий из нулей;

$1$  — вектор-столбец размерности  $C_{n-1}^{k-1} \times 1$ , состоящий из единиц;

$G(n-1, k)$  — матрица  $C_{n-1}^k \times (n-1)$  кодовых слов, содержащих ровно  $k$  единиц;

$\overline{G(n-1, k-1)}$  — матрица  $C_{n-1}^{k-1} \times (n-1)$  кодовых слов, содержащих ровно  $k-1$  единиц, причем кодовые слова записаны в порядке, обратном порядку  $G(n-1, k-1)$  ( $\overline{G}$  — «перевернутая» матрица  $G$ ).

На рис. 1.1 приведен пример построения кодовых слов Грея для генерации сочетаний из 4 элементов по 2.

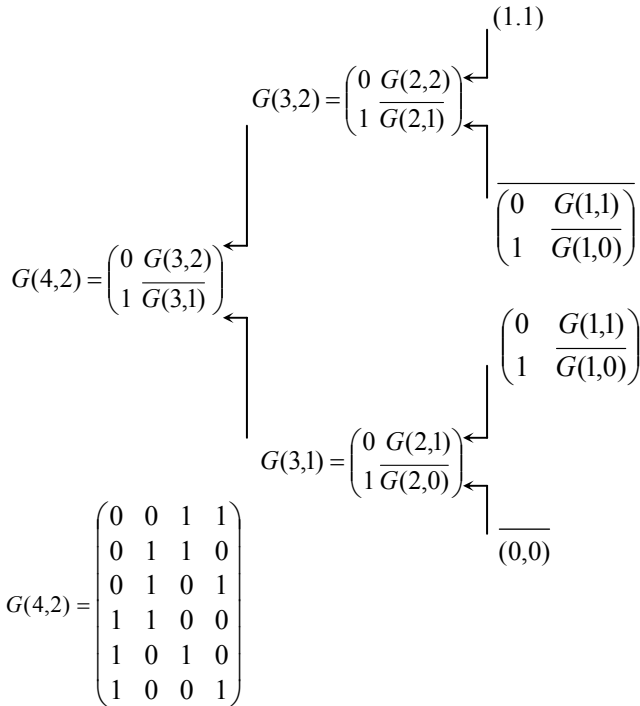


Рис. 1.1. Кодовые слова Грея для сочетаний из 4 по 2

Индукцией по  $n$  доказывается, что последовательность кодовых слов  $G(n, k)$  получается удалением из кода Грея  $G(n)$  всех кодовых слов с числом единиц, не равным  $k$ , причем в этой последовательности любые два соседних кодовых слов различаются только в двух позициях.

### 1.3. Генерация перестановок

#### 1.3.1. Генерация перестановок в лексикографическом порядке

Будем рассматривать исходное множество  $X = \{1, 2, \dots, n\}$ , и в качестве начальной перестановки возьмем  $\pi' = (1, 2, \dots, n)$ . Условие окончания работы — порождение перестановки  $\pi'' = (n, n-1, \dots, 2, 1)$ , которая является последней в лексикографическом смысле среди всех перестановок множества  $X$ . Переход от текущей перестановки  $\pi = (\pi_1, \pi_2, \dots, \pi_n)$  к следующей за ней будем осуществлять таким образом:

1) просматривая перестановку  $\pi$  справа налево, ищем самую первую позицию  $i$  такую, что  $\pi_i < \pi_{i+1}$  (если такой позиции нет, значит текущая подстановка  $\pi = \pi''$  и процесс генерации завершается);

2) просматривая  $\pi$  от  $\pi_i$  слева направо, ищем наименьший из элементов  $\pi_j$  такой, что  $\pi_i < \pi_j (i < j)$ ;

3) меняем местами элементы  $\pi_i$  и  $\pi_j$ ; затем все элементы  $\pi_{i+1}, \pi_{i+2}, \dots, \pi_n$  записываем в обратном порядке (т.е. меняем местами симметрично расположенные элементы  $\pi_{i+1+t}$  и  $\pi_{n-t}$ ).

*Пример.* Пусть текущая перестановка  $\pi$  имеет вид  $\pi = (3, 5, 7, 6, 4, 2, 1)$ . На первом шаге найдены  $\pi_i = 5, i = 2$ ; на втором —  $\pi_j = 6, j = 4$ ; на третьем шаге меняем местами  $\pi_i$  и  $\pi_j$ :  $(3, 6, 7, 5, 4, 2, 1)$  и меняем местами элементы, начиная с третьей позиции:  $(3, 6, 1, 2, 4, 5, 7)$  — получили подстановку, следующую за текущей в лексикографическом порядке.

#### 1.3.2. Генерация перестановок с помощью вложенных циклов

Будем говорить, что перестановка  $\pi = \begin{pmatrix} 1 & 2 & \dots & n \\ a_1 & a_2 & \dots & a_n \end{pmatrix}$  является

циклом длины  $k$  степени  $d$ , если ее элементы  $a_i, i = \overline{1, k}$ , получены из  $1, 2, \dots, k$  циклическим сдвигом вправо на  $d$  позиций, остальные  $n - k$  элементов стационарны. Например, подстановка

$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 1 & 2 & 3 & 5 & 6 \end{pmatrix}$  является циклом длины 4 степени 1.

Алгоритм порождения подстановок с помощью вложенных циклов основан на следующей теореме.

**Теорема 1.** Любую подстановку  $\pi$  на множестве  $X = \{1, 2, \dots, n\}$  можно представить в виде композиции

$$\pi = \rho_n \circ \rho_{n-1} \circ \dots \circ \rho_1 \quad (1.2)$$

где  $\rho_i$  — циклическая подстановка порядка  $i$ .

*Пример.* Представим в виде (2.1) подстановку  $\pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 4 & 1 \end{pmatrix}$ ,

т.е. запишем

$$\begin{aligned} \pi = \rho_4 \circ \rho_3 \circ \rho_2 \circ \rho_1 &= \begin{pmatrix} 1 & 2 & 3 & 4 \\ a_1 & a_2 & a_3 & a_4 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 \\ b_1 & b_2 & b_3 & 4 \end{pmatrix} \circ \\ &\begin{pmatrix} 1 & 2 & 3 & 4 \\ c_1 & c_2 & 3 & 4 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 \\ d_1 & 2 & 3 & 4 \end{pmatrix}. \end{aligned}$$

Очевидно, последний цикл является тождественной подстановкой. Определим  $\rho_4$ : т.к.  $\rho_3(4) = 4, \rho_2(4) = 4$ , то  $\rho_4(4) = 1$  следовательно

$$\rho_4 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 1 & 4 \end{pmatrix} \text{ — цикл порядка 4.}$$

Т.к.  $\rho_2(3) = 3$ , то  $3 = \pi(1) = \rho_3(2) = \rho_3(\rho_4(1)) = \rho_2(2)$  и

$$\rho_2 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 3 & 4 \end{pmatrix}.$$

Разложение подстановки  $\pi$  имеет вид:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 4 & 1 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 1 & 4 \end{pmatrix} \circ \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 3 & 4 \end{pmatrix} \quad (1.3)$$

Диаграмма композиции (1.3) приведена на рис. 2.2.

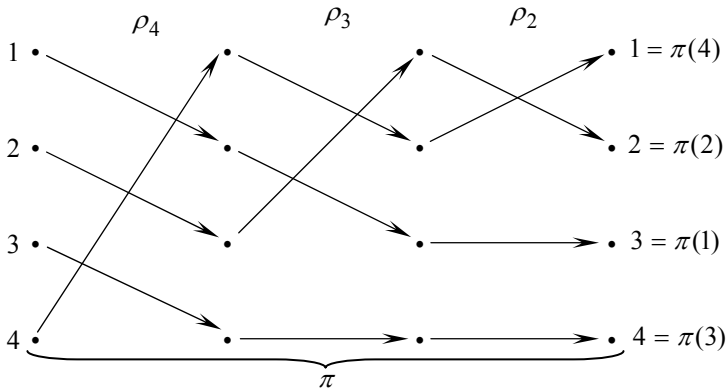


Рис. 2.2. Разложение в произведение вложенных циклов

Из теоремы 1 следует, что все перестановки можно получить систематическим перебором циклических сдвигов. В качестве начальной перестановки берем  $\pi' = (1, 2, \dots, n)$  и сдвигаем на одну позицию вправо все элементы до тех пор, пока вновь не получим  $\pi'$ ; теперь сдвигаем циклически первые  $n-1$  элементов и снова повторяем сдвиг всех  $n$  элементов на одну позицию до тех пор, пока не получим уже имеющуюся перестановку; сдвигаем циклически ее первые  $n-2$  элементов... и т.д., пока не переберем все  $n!$  перестановок. Ниже приведен алгоритм вложенных циклов.

1. Цикл ( $i = \overline{1, n}$ )
  - 1.1.  $\pi_i := i$ ;
2. Конец цикла
3.  $k := 0$
4. Пока ( $k \neq 1$ )
  - 4.1. Печать  $\pi = (\pi_1, \pi_2, \dots, \pi_n)$
  - 4.2.  $k := n$
  - 4.3. Сдвиг первых  $k$  элементов на одну позицию
  - 4.4. Пока ( $\pi_k = k$  и  $k > 0$ )
    - 4.4.1.  $k := k - 1$
    - 4.4.2. Сдвиг первых  $k$  элементов на одну позицию
  - 4.5. Конец цикла

5. Конец цикла

6. Конец

Этот алгоритм не является эффективным, т.к. на каждом шаге требует большого количества (не меньше  $n$ ) транспозиций (транспозиция — обмен местами двух элементов).

### 1.3.3. Транспозиция соседних элементов

Описанные выше алгоритмы генерации перестановок не являются алгоритмами с минимальными изменениями. Минимальным изменением при переходе от текущей перестановки к следующей является транспозиция двух элементов. Дадим рекурсивное описание такого алгоритма.

Если  $n = 1$ , то существует единственная перестановка  $\pi^{(1)} = (1)$ . Пусть  $n > 1$  и последовательность перестановок  $\pi^{(1)}, \pi^{(2)}, \dots, \pi^{(r)}$ ,  $r = (n-1)!$  на множестве  $(1, 2, \dots, n-1)$  построена. Для получения перестановок на множестве  $(1, 2, \dots, n)$  будем вставлять элемент  $n$  на «промежутке» между элементами перестановки  $\pi^{(i)}$  по следующему правилу: если номер  $i$  подстановки  $\pi^{(i)}$  — нечетное число, то элемент  $n$  вставляется в промежутки справа налево, если  $i$  — четное число, то элемент  $n$  вставляется в промежутки между элементами  $\pi^{(i)}$  слева направо.

Пример генерации перестановки при  $n = 4$  приведен на рис. 1.3.

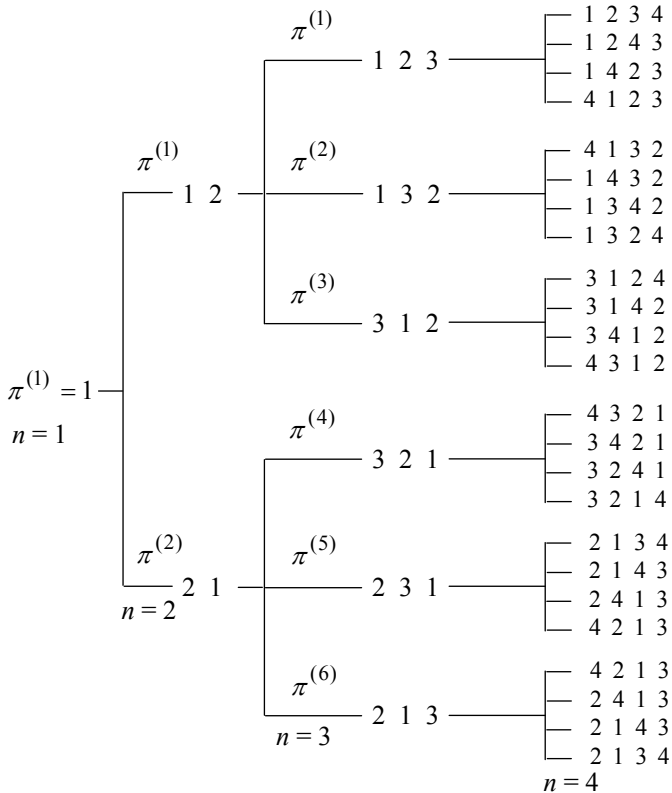


Рис. 1.3. Генерация перестановок транспозиций соседних элементов ( $n = 4$ )

#### 1.4. Задание на выполнение

Задать с клавиатуры мощность множества  $n$ , элементы множества и мощность выборки  $k$ . Реализовать алгоритм генерации сочетания

1. в лексикографическом порядке;
2. с помощью кодов Грея.



Задать с клавиатуры мощность множества  $n$  и элементы множества. Реализовать алгоритм генерации перестановок:

3. в лексикографическом порядке;
4. с помощью вложенных циклов;
5. транспозицией соседних элементов.

## **Практическое занятие № 4. «Машинное представление графов»**

### ***1.1. Цель работы***

Разработка и реализация алгоритмов преобразования различных форм представления графов.

### ***1.2. Машинные способы представления графов***

#### ***1.2.1. Матрица смежности***

Матрицей смежности неориентированного графа  $G=(X,U)$ ,  $|X|=n, |U|=m$  называется квадратная матрица  $A[n \times n]$ , элементы которой задаются по правилу 1.4.

$$a_{i,j} = \begin{cases} 1, & \text{если вершина } x_i \text{ смежна вершине } x_j; \\ 0, & \text{если вершина } x_i \text{ не смежна вершине } x_j; \end{cases} \quad (1.4)$$

Матрицей смежности ориентированного графа  $G=(X,U)$ ,  $|X|=n, |U|=m$  называется квадратная матрица  $A[n \times n]$ , элементы которой задаются по правилу 1.5.

$$a_{i,j} = \begin{cases} 1, & \text{если вершина } x_i \text{ смежна вершине } x_j; \\ -1, & \text{если вершина } x_j \text{ смежна вершине } x_i; \\ 0, & \text{если вершина } x_i \text{ не смежна вершине } x_j; \end{cases} \quad (1.5)$$

### 1.2.2. Матрица инцидентности

Матрицей инцидентности неориентированного графа  $G=(X,U)$ ,  $|X|=n, |U|=m$  называется матрица  $B[n \times m]$ , элементы которой задаются по правилу 1.6.

$$b_{i,j} = \begin{cases} 1, & \text{если вершина } x_i \text{ инцидентна ребру } u_j; \\ 0, & \text{если вершина } x_i \text{ не инцидентна ребру } u_j. \end{cases} \quad (1.6)$$

Матрицей инцидентности ориентированного графа  $G=(X,U)$ ,  $|X|=n, |U|=m$  называется матрица  $B[n \times m]$ , элементы которой задаются по правилу 1.7.

$$a_{i,j} = \begin{cases} 1, & \text{если вершина } x_i \text{ начало дуги } u_j; \\ -1, & \text{если вершина } x_i \text{ конец дуги } u_j; \\ 2, & \text{если } u_j \text{ - петля при вершине } x_i; \\ 0, & \text{если вершина } x_i \text{ не инцидентна дуге } u_j. \end{cases} \quad (1.7)$$

### 1.2.3. Список ребер

Списком ребер неориентированного графа  $G=(X,U)$ ,  $|X|=n, |U|=m$  называются два массива  $N[2m]$  и  $K[2m]$ , элементы которых задаются по правилу 1.8:

$$\begin{aligned} n_i & - \text{вершина, являющаяся началом ребра с номером } i. \\ k_i & - \text{вершина, являющаяся концом ребра с номером } i. \end{aligned} \quad (1.8)$$

Списком ребер ориентированного графа  $G=(X,U)$ ,  $|X|=n, |U|=m$  называются два массива  $N[m]$  и  $K[m]$ , элементы которых задаются по правилу 1.9:

$$\begin{aligned} n_i & - \text{вершина, являющаяся началом дуги с номером } i. \\ k_i & - \text{вершина, являющаяся концом дуги с номером } i. \end{aligned} \quad (1.9)$$

### 1.2.4. Структура смежности

Структурой смежности графа  $G=(X,U)$ ,  $|X|=n, |U|=m$  называется массив списков

$$x_i : x_k, x_{k1}, \dots, x_{kz}, \quad i = \overline{1, n},$$

где  $x_k, x_{k1}, \dots, x_{kz}$  - все вершины, смежные вершине  $x_i$ .

Пример реализации структуры смежности предложен в следующей программе:

```
#include <iostream.h>
#include <conio.h>
```

```
void main()
{
struct List {
int Number;
List *Next;
};
```

```
List *Smegn; // массив вершин
int n; // количество вершин графа
```

```
clrscr();
cout << "Введите количество вершин графа: ";
cin >> n;
// Выделение памяти под массив вершин
Smegn = new List [n];
for (int i=0; i<n; i++)
{
Smegn[i].Number = i+1;
Smegn[i].Next = NULL;
}
// Ввод структуры смежности
cout << "Признак окончания ввода - 0" << endl;
for(i = 0; i<n; i++)
{
cout << "Вводите вершины смежные вершине " << i+1 << " : ";
int d = 1;
List* Cur = &Smegn[i];
while (d!=0)
{
cout << "# вершины: ";
cin >> d;
if (d==0) continue;
Cur->Next = new List; // выделение памяти под новый элемент
Cur = Cur->Next;
Cur->Next = NULL;
```

```

        Cur->Number = d;
    }
}
// Печать структуры смежности
for (i=0;i<n;i++)
{
    cout << Smegn[i].Number << ": "; // номер вершины
    List *Cur = &Smegn[i];
    if (Cur->Next == NULL) {cout << endl; continue;} // Если смежных нет, то
                                                    //перейти на следующую
                                                    //вершину.
    do // пока не пройдены все смежные вершины,
        //выводить на экран номера вершин
        {
            if (Cur->Next->Next == NULL) continue;
            Cur = Cur->Next;
            cout << Cur->Number << ", ";
        }
        while (Cur->Next->Next != NULL);
        Cur = Cur->Next;
        cout << Cur->Number << "." << endl; // вывод последней вершины
    }
    cout << endl;
    // удаление структуры смежности из памяти.
    for (i=0;i<n;i++)
    {
        List *Top;
        Top = &Smegn[i];
        List* Cur = Top->Next;
        delete Top;
        Top = Cur;
        while (Cur!=NULL)
        {
            Cur = Top->Next;
            delete Top;
            Top = Cur;
        }
        delete [] Smegn;
    }
}
getch();
}

```

### 1.3. Задание на выполнение

Написать программу по заданному варианту. Ввод данных выполнить с клавиатуры или из текстового файла. Предусмотреть ошибки ввода – некорректные данные, отсутствие файла и т.д..

## Практическое занятие № 5. «Построение матрицы достижимости»

### 1.1. Цель работы

Реализовать алгоритмы построения матрицы достижимости.

### 1.2. Основные определения

Матрицей достижимости графа  $G=(X,U)$ ,  $|X|=n,|U|=m$  называется квадратная матрица  $D[n \times n]$ , элементы которой задаются по правилу 1.10:

$$d_{i,j} = \begin{cases} 1, & \text{если вершина } x_j \text{ достижима из } x_i; \\ 0, & \text{если вершина } x_j \text{ не достижима из } x_i; \end{cases} \quad (1.10)$$

### 1.3. Алгоритмы получения матрицы достижимости

#### 1.3.1. Алгоритм построчного сложения

Пусть граф  $G=(X,U)$ ,  $|X|=n,|U|=m$  задан матрицей смежности

А. Получим матрицу достижимости  $D$  графа следующим способом:

1.  $D = A$ ;

2. Цикл ( $i = \overline{0; n-1}$ )

2.1. Пока (в строке  $i$  существует хотя бы одна не просмотренная 1)

2.1.1. Если ( $d_{i,j} = 1$ ) То прибавить к строке  $i$

строку  $j$ .

2.2. Конец цикла

3. Конец цикла

4. Печать  $D$

5. Конец

### 1.3.2. Алгоритм булевого сложения матриц

Пусть граф  $G=(X,U)$ ,  $|X|=n, |U|=m$  задан матрицей смежности

А. Получим матрицу достижимости  $D$  графа следующим способом:

1.  $D^{(0)} = E$ ; //  $E$  – единичная матрица

2.  $C = A$

3.  $D^{(1)} = C$

4.  $i=1$ ;

5. Пока ( $D^{(i)} \neq D^{(i-1)}$ )

5.1.  $C = C \cdot A$

5.2.  $i = i+1$

5.3.  $D^{(i)} = D^{(i-1)} + C$

6. Конец цикла

7. Конец

### 1.4. Задание на выполнение

Граф задан матрицей смежности. Найти матрицу достижимости графа.

## Практическое занятие № 6. «Алгоритмы на графах»

### 1.1. Цель работы

Программная реализация основных алгоритмов на графах

### 1.2. Алгоритмы обходов графа

Дан граф  $G=(X,U)$ ,  $|X|=n, |U|=m$ .

1. Цикл ( $i = \overline{1, n}$ )

1.1.  $M[x_i] = 0$ ; // Все вершины не отмечены

2. Конец цикла

3. Выбрать произвольную вершину  $v = x_i \in X$

4.  $v \rightarrow T$  // Записать выбранную вершину в структуру  $T$

5.  $M[v] = 1$  // Отмечаем вершину, как пройденную

6. Пока ( $T$  не пуста)

6.1.  $u \leftarrow T$  // извлекаем вершину из структуры

6.2. Печать  $u$

- 6.3. Цикл(для всех вершин  $w$ , смежных с  $u$ )
  - 6.3.1. Если  $(M[w]=0)$  То  $w \rightarrow T$ ;  $M[w]=1$
- 6.4. Конец цикла
7. Конец цикла
8. Конец

Если структуру  $T$  определить как стек, то алгоритм обхода будет выполняться «в глубину». Если же  $T$  определена как очередь, будет выполняться обход «в ширину».

### 1.3. Алгоритмы поиска путей на графах

#### 1.3.1. Алгоритм Дейкстры

Алгоритм Дейкстры находит кратчайший путь между двумя заданными вершинами в орграфе.

Дан граф  $G=(X,U)$ ,  $|X|=n$ ,  $|U|=m$ . Граф задан матрицей весов  $C$ ,  $s$  – начальная вершина обхода,  $t$  – конечная вершина обхода.

На выходе алгоритма создаются два массива:

- $T$  – если вершина  $v$  лежит на кратчайшем пути, то  $T[v]$  – длина кратчайшего пути от  $s$  к  $v$ .
- $H$  –  $H[v]$  – вершина, непосредственно предшествующая  $v$  на кратчайшем пути.

#### 1. Цикл ( $x = \overline{1, n}$ )

- 1.1.  $T[x] = \infty$  // кратчайший путь не известен
- 1.2.  $M[x] = 0$  // все вершины не отмечены

2. Конец цикла
3.  $H[s] = 0$  //  $s$  ничего не предшествует
4.  $T[s] = 0$  // кратчайший путь имеет длину 0
5.  $M[s] = 1$  // вершина  $s$  пройдена
6.  $v = s$  // зафиксируем текущую вершину
7. Цикл (1)

#### 7.1. Цикл(для $\forall u$ смежных с $v$ )

- 7.1.1. Если  $(M[u] = 0)$  и  $T[u] > T[v] + C[v, u]$   
То  $T[u] = T[v] + C[v, u]$ ; // найден более  
// короткий путь из  $s$  в  $u$  через  $v$   
 $H[u] = v$ ;

- 7.2. Конец цикла
- 7.3.  $t = \infty$

7.4.  $v=0$

7.5. Цикл ( $x = \overline{1, n}$ )

7.5.1. Если ( $M[x] = 0$  и  $T[x] < t$ )

То  $v=u$ ;  $t=T[u]$  // вершина  $v$  заканчивает

// кратчайший путь из  $s$ .

7.6. Конец цикла

7.7. Если ( $v=0$ ) То «Нет пути из  $s$  в  $t$ »; Закончить выполнение цикла.

7.8. Если ( $v=t$ ) То «Путь найден»;

Печать  $T[t]$ ;

Печать  $H[t]$ .

7.9.  $M[v]=1$

8. Конец цикла

9. Конец

### 1.3.2. Алгоритм Форда

Алгоритм Форда позволяет найти расстояние от заданной вершины  $s$  до всех вершин  $T[v]=d(s,v)$ , ориентированного графа. Исходными данными для этого алгоритма является матрица весов дуг  $C$ . В отличие от алгоритма Дейкстры алгоритм может быть использован на графах с отрицательными длинами дуг.

Дан граф  $G=(X,U)$ ,  $|X|=n, |U|=m$ . Граф задан матрицей весов  $C$ .  
 $s$  – вершина начала пути.

1. Цикл (для всех вершин  $x$  графа  $G$ )

1.1.  $D[x]:=C[s,x]$ ;

2. Конец цикла

3.  $D[s]:=0$ ;

4. Цикл ( $k = \overline{1, n-2}$ )

// последовательно перебрать все ребра

4.1. Цикл (для всех вершин  $v$  графа,  $v \neq s$ )

// применить процесс поиска кратчайшего

//пути для всех ребер

4.1.1. Цикл (для всех вершин  $u$  графа,  $u \neq v$ )

4.1.1.1 Если ( $D[v] > D[u] + C[u,v]$ )

То  $D[v] = D[u] + C[u,v]$

4.1.2. Конец цикла

4.2. Конец цикла

// закончить алгоритм досрочно



5. Если нет изменений в  $D$ , То  $k=n-2$
6. Конец цикла
7. Печать  $D$ .
8. Конец

#### 1.3.4. Алгоритм ближайшего соседа

Существует другой метод получения кратчайшего остовного дерева, который не требует ни сортировки ребер, ни проверки на цикличность на каждом шаге, - так называемый *алгоритм ближайшего соседа*. Просмотр начинается с некоторой произвольной вершины  $a$  в заданном графе. Пусть  $(a,b)$ - ребро с наименьшим весом, инцидентное  $a$ ; ребро  $(a,b)$  включается в остов. Затем среди всех ребер, инцидентных либо  $a$ , либо  $b$ , выбираем ребро с наименьшим весом и включаем его в частично построенное дерево. В результате этого в дерево добавляется новая вершина, например,  $c$ . Повторяя процесс, ищем наименьшее ребро, соединяющее  $a$ ,  $b$  или  $c$  с некоторой другой вершиной графа. Процесс продолжается до тех пор, пока все вершины из  $G$  не будут включены в дерево, то есть пока дерево не станет остовным.

Дан граф  $G=(X,U)$ ,  $|X|=n, |U|=m$ . Граф задан матрицей весов  $C$ .

1.  $T$  – пустое множество ребер остовного дерева
2.  $a$  – произвольная вершина графа
3.  $a \rightarrow T$
4. Пока ( $T \neq X$ )
  - 4.1. Найти ребро  $(u,v)$  с минимальным весом, такое, что  $u \in T, v \notin T$ .
  - 4.2.  $v \rightarrow T$
5. Конец цикла
6. Печать  $T$
7. Конец

#### 1.3.5. Алгоритм Краскала

Алгоритм Краскала относится к семейству «жадных» алгоритмов. Алгоритм ищет кратчайший остов в связном графе  $G=(X,U)$ ,  $|X|=n, |U|=m$ . Граф задан списком ребер  $U$  с длинами. На выходе алгоритма создается список  $T$  ребер кратчайшего остова.

1.  $T$  – пустой список
2. Упорядочить список  $U$  в порядке возрастания длин

3.  $k = 1$  // номер рассматриваемого ребра
4. Цикл( $i = \overline{1, m-1}$ )
  - 4.1. Пока (добавление ребра  $E[k]$  образует цикл в  $T$ )
    - 4.1.1.  $k=k+1$
  - 4.2. Конец цикла
  - 4.3.  $E[k] \rightarrow T$
5. Конец цикла
6. Печать  $T$
7. Конец

### 1.3.6. Волновой алгоритм

Волновой алгоритм является алгоритмом поиска кратчайшего пути между двумя вершинами в неориентированном ненагруженном графе.

Пусть дан граф  $G=(X,U)$ ,  $|X|=n, |U|=m$ . Вершина  $a$  – начало пути, вершина  $b$  – конец пути. Запишем вершину  $a$  во фронт волны нулевого уровня  $FW_0$ . Введем переменную  $i=0$ . Далее найдем все вершины, смежные вершинам  $v \in FW_i$  и ранее не пройденные. Запишем эти вершины во фронт волны следующего уровня  $FW_{i+1}$ . Увеличим  $i - i = i + 1$ . Если среди найденных вершин есть вершина  $b$ , то длина кратчайшего пути найдена и равна  $i$ , если нет, то процесс продолжается до тех пор пока не будет найдена конечная вершина пути, либо, пока не будут просмотрены все вершины графа. В этом случае пути из  $a$  в  $b$  нет.

1. Цикл( $i = \overline{1, n}$ )
  - 1.1.  $M[x_i] = -1$  // отметим все вершины, как не пройденные
2. Конец цикла
3.  $M[a]=0$ ; // вершина пройдена
4.  $i=1$
5.  $a \cup FW_{i-1}$
6. Пока ( $b \notin FW_{i-1}$  И  $\exists x \in X, M[x] = -1$ )
  - 6.1. Цикл (для вершин  $v \in X, M[v] = -1$  и смежных вершинам  $\notin FW_{i-1}$ )
    - 6.1.1.  $v \cup FW_i$
    - 6.1.2.  $i = i + 1$
  - 6.2. Конец цикла

7. Конец цикла
8. Если  $b \in FW_{i-1}$  То «Путь найден», длина пути равна  $i-1$ .  
Иначе «Путь не существует»
9. Конец

### 1.3.7. Алгоритм Уоршалла

Алгоритм вычисления транзитивного замыкания для орграфа  $G=(X,U)$ ,  $|X|=n, |U|=m$ . Граф задан матрицей смежности.

1.  $S = R$  // вспомогательная матрица
2. Цикл( $i = \overline{1, n}$ )
  - 2.1. Цикл( $j = \overline{1, n}$ )
    - 2.1.1. Цикл( $k = \overline{1, n}$ )
 

$T[i, j] = S[j, k] \vee S[j, i] \& S[i, k]$  // вычисление  
// матрицы  
// транзитивного  
// замыкания
  - 2.2. Конец цикла
3. Конец цикла
4. Печать  $T$ .
5. Конец

### 1.3.8. Алгоритм построения эйлеровой цепи

Если граф имеет цикл 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.

Для того, чтобы в графе существовал эйлеров цикл граф должен быть связанным, для неориентированных графов число ребер в каждой вершине должно быть четным.

Дан эйлеров граф  $G=(X,U)$ ,  $|X|=n, |U|=m$ , заданный структурой смежности.  $\Gamma[v]$ - множество вершин, смежных с вершиной  $v$ .

1.  $S$  – пустой стек // стек для хранения вершин
2. Выбрать произвольную вершину  $x \in X$
3.  $x \rightarrow S$  // положить  $x$  в стек

4. Пока (Стек не пуст)
  - 4.1.  $v \leftarrow S$
  - 4.2.  $v \rightarrow S$
  - 4.3. Если  $\Gamma[v]$ - пустое множество То  $v \leftarrow S$  ; Печать  $v$   
 Иначе взять  $u \in \Gamma[v]$ ; // взять первую  
 // вершину, смежную  $v$   
 $u \rightarrow S$   
 //удалить ребро  $(v,u)$   
 $\Gamma[v] := \Gamma[v] \setminus u$ ;  
 $\Gamma[u] := \Gamma[u] \setminus v$ ;
6. Конец цикла
7. Конец

#### ***1.4. Задание на выполнение***

Программно реализовать один из описанных алгоритмов. Программа должна позволять пользователю вводить данные о графе с клавиатуры, либо считывать из текстового файла.

#### ***Рекомендуемая литература***

1. **Павловская**, Татьяна Александровна. С/С++: Программирование на языке высокого уровня: Учебник для вузов - СПб.: Питер, 2002.
2. **Новиков**, Федор Алексеевич. Дискретная математика для программистов: Учебник для вузов - СПб.: Питер, 2000.
3. Кнут, Дональд Эрвин. Искусство программирования: - М.: Вильямс, 2005.
4. Линский В. Комбинаторика для программистов. –М.:Мир, 1988.
5. Рейнгольд Э., Нивергельт Ю., Део Н. Комбинаторные алгоритмы: теория и практика. –М.:Мир, 1980.
6. Касьянов В.Н., Сабельфельд В.К. Сборник заданий по практикуму на ЭВМ. –М.:Наука, 1986.