

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Владимирский государственный университет

им. А. Г. и Н. Г. Столетовых» (ВлГУ)

А. В. ШУТОВ

Ю. А. МЕДВЕДЕВ

**СТРУКТУРЫ И АЛГОРИТМЫ КОМПЬЮТЕРНОЙ ОБРАБОТКИ ДАННЫХ**

Часть2. Лабораторный практикум

по дисциплине «Структуры и алгоритмы компьютерной обработки данных» для студентов, обучающихся по направлению 010500 «Математическое обеспечение и администрирование информационных систем»

ВЛАДИМИР – 2013

УДК 004.31

ББК 32.988 – 5 я7

Ш 97

**Шутов А. В., Медведев Ю. А.**

Структуры и алгоритмы компьютерной обработки данных. Часть 2 (Лабораторный практикум). – Владимир: ВлГУ, 2013. – 109 с.

Учебное пособие адресовано студентам, обучающимся по направлению 010500 «Математическое обеспечение и администрирование информационных систем». Содержит теоретический материал, необходимый для выполнения лабораторных занятий, а также задания для самостоятельной работы студентов.

Практикум включает 18 лабораторных работ по 2 темам: алгоритмы на графах, алгоритмы комбинаторного перебора. Материал систематизирован и может быть использован студентами физико-математических факультетов.

**Рецензенты:** доктор технических наук, профессор **Монахов М. Ю.**, зав. кафедрой информатикии защитыинформацииВлГУ;

доктор физико-математических наук, профессор ВлГУ

**Алхутов Ю. А.**

*Печатается по решению Редакционно-*

*издательского совета ВлГУ*

© ФГБОУ ВПО «Владимирский государственный университет», 2013

© Шутов А. В., Медведев Ю. А., 2013

**ВВЕДЕНИЕ**

Одним из важнейших аспектов подготовки будущего специалиста в области компьютерных технологий является знакомство с базовыми алгоритмами информатики, а также со структурами данных, позволяющими реализовывать эти алгоритмы. Среди этих алгоритмов особое место занимают алгоритмы решения задач, связанных с графами, а также методы организации перебора. Рассматриваемые алгоритмы полезны для решения огромного числа проблем от анализа экономических систем до проектирования компьютерных сетей.

Следует отметить, что для полноценного изучения алгоритма недостаточно ознакомления с соответствующим теоретическим материалом. Необходимо написание компьютерной программы, реализующей алгоритм, а также самостоятельное использование данной программы для решения практических задач.

Учебное пособие содержит 9 лабораторных работ, посвященных алгоритмам на графах, а также 9 работ, связанных с алгоритмами комбинаторного перебора. Лабораторные работы содержат изложение теоретического материала, необходимого для их выполнения, описание хода работ, а также задания для реализации на компьютере и вопросы для самопроверки.

Учебное пособие предназначено для проведения лабораторных работ по дисциплине «Структуры и алгоритмы компьютерной обработки данных» для студентов вузов, обучающихся по направлению «Математическое обеспечение и администрирование информационных систем». Отдельные работы из данного пособия также могут быть использованы при изучении дисциплин «Программирование», «Теоретические основы информатики» студентами вузов, обучающимися на физико-математических факультетах по направлению «Педагогическое образование», а также в школах при проведении факультативов по информатике и при подготовке учащихся к олимпиадам по программированию.

**Тема 1. Алгоритмы на графах (18 часов).**

**Цель**: Практическое изучение базовых алгоритмов на графах и способов их реализации на компьютере.

**ЛАБОРАТОРНАЯ РАБОТА №1-2**

МАТРИЧНЫЕ СПОСОБЫ ПРЕДСТАВЛЕНИЯ ГРАФОВ

1 ЦЕЛЬ РАБОТЫ

Целью работы является изучение матричных способов представления графов.

2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

В последнее время теория графов стала простым, доступным и мощным средством решения вопросов, относящихся к широкому кругу проблем. Это проблемы проектирования интегральных схем и схем управления, исследования автоматов, логических цепей, блок-схем программ, экономики и статистики, химии и биологии, теории расписаний и дискретной оптимизации.

2.1 ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

*Граф G* задается множеством точек или вершин *x1*,*x2,...,xn* (которое обозначается через *X*) и множеством линий или *ребер* *a1,a2,…,an* (которое обозначается символом *А*), соединяющих между собой все или часть этих точек. Таким образом, граф *G* полностью задается (и обозначается) парой (*X, А*).

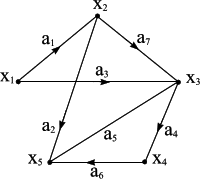
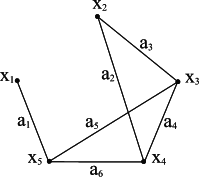
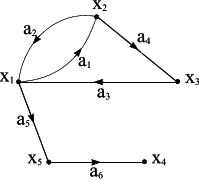
Если ребра из множества *А* ориентированы, что обычно показывается стрелкой, то они называются *дугами*, и граф с такими ребрами называется *ориентированным* графом (рисунок 1(а)). Если ребра не имеют ориентации, то граф называется *неориентированным* (рисунок 1(б)). В случае когда *G=*(*X, А*) является ориентированным графом и мы хотим пренебречь направленностью дуг из множества *А*, то неориентированный граф, соответствующий *G*, будем обозначать как *G=*(*X, А*).

(в)

Рисунок 1 (a) – ориентированный граф; (б) – неориентированный граф;  
 (в) – смешанный граф.

(б)

(a)



Если дуга обозначается упорядоченной парой, состоящей из *начальной* и *конечной* вершин (т. е. двумя *концевыми* вершинами дуги), ее направление предполагается заданным от первой вершины ко второй. Так, например, на рисунке 1(а) обозначение (*x1*,*x2*)относится к дуге *a1*, а (*x2*,*x1*) *-* к дуге *a2*.

Другое, употребляемое чаще описание ориентированного графа *G* состоит в задании множества вершин *Х* и *соответствия* Г, которое показывает, как между собой связаны вершины. Соответствие Г называется *отображением* множества *Х* в *Х*, а граф в этом случае обозначается парой *G=*(*X,*Г).

Для графа на рисунке 1(а) имеем Г(*x1*)={*x2*,*x5*}, т. е. вершины *x2* и *x5* являются конечными вершинами дуг, у которых начальной вершиной является *x1*.

Г(*x2*)={*x1*,*x3*}, Г(*x3*)={*x1*}, Г(*x4*)= *-* пустое множество, Г(*x5*)={*x4*}.

В случае неориентированного графа или графа, содержащего и дуги, и неориентированные ребра (см., например, графы, изображенные на рисунках 1(б) и 1(в)), предполагается, что соответствие Г задает такой эквивалентный ориентированный граф, который получается из исходного графа заменой каждого неориентированного ребра двумя противоположно направленными дугами, соединяющими те же самые вершины. Так, например, для графа, приведенного на рисунке 1(б), имеем Г(*x5*)={*x1*,*x3,x4*}, Г(*x1*)={*x5*}и др.

Поскольку *прямое соответствие* или *образ* вершины Г(*xi*)представляет собой множество таких вершин *xj*X, для которых в графе *G* существует дуга (*xi*,*xj*), то через Г-1(*xi*) естественно обозначить множество вершин *xk*, для которых в *G* существует дуга (*xk*,*xi*). Такое отношение принято называть *обратным* *соответствием* или *прообразом* вершины. Для графа, изображенного на рисунке 1(а), имеем

Г-1(*x1*)={*x2*,*x3*}, Г-1(*x2*)={*x1*} и т. д.

Вполне очевидно, что для неориентированного графа Г-1(*xi*)=Г(*xi*) для всех *xi*X.

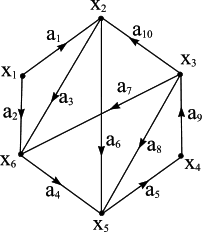
Когда отображение Г действует не на одну вершину, а на множество вершин *Xq*={*x1*,*x2*,...,*xq*}, то под Г(*Xq*)понимают объединение Г(*x1*)Г(*x2*)...Г(*xq*), т. е. Г(*Xq*)является множеством таких вершин *xj**X*, что для каждой из них существует дуга (*xi*,*xj*) в *G*, где *xi**Xq*. Для графа, приведенного на рисунке 1(а), Г({*x2*,*x5*})={*x1*,*x3*,*x4*} и Г({*x1*,*x3*})={*x2*,*x5*,*x1*}.

Отображение Г(Г(*xi*)) записывается как Г2(*xi*). Аналогично "тройное" отображение Г(Г(Г(*xi*))) записывается как Г3(*xi*) и т. д. Для графа, показанного на рисунке 1(а), имеем:

Г2(*x1*)=Г(Г(*x1*))=Г({*x2*,*x5*})={*x1*,*x3*,*x4*};

Г3(*x1*)=Г(Г2(*x1*))=Г({*x1*,*x3*,*x4*})={*x2*,*x5*,*x1*} и т. д.

Аналогично понимаются обозначения Г-2(*xi*), Г-3(*xi*) и т. д.

Дуги *a=*(*xi*,*xj*), *xi**xj*, имеющие общие концевые вершины, называются *смежными*. Две вершины *xi* и *xj* называются смежными, если какая-нибудь из двух дуг (*xi*,*xj*) и (*xj*,*xi*) или обе одновременно присутствуют в графе. Так, например, на рисунке 2 дуги *a1*, *a10*, *a3* и *a6* как и вершины *x5* и *x3*, являются смежными, в то время как дуги *a1* и *a5* или вершины *x1* и *x4* не являются смежными.

Число дуг, которые имеют вершину *xi* своей начальной вершиной, называется *полустепенью* *исхода* вершины *xi*, и, аналогично, число дуг, которые имеют *xi* своей конечной вершиной, называется *полустепенью захода* вершины *xi*.

Рисунок 2.

Таким образом, на рисунке 2 полустепень исхода вершины *x3*, обозначаемая через *deg+*(*x3*), равна Г(*x3*)=3, и полустепень захода вершины *x3*, обозначаемая через *deg-*(*x3*), равна Г-1(*x3*)=1.

Очевидно, что сумма полустепеней захода всех вершин графа, а также сумма полустепеней исхода всех вершин равны общему числу дуг графа *G*, т. е.

, (1)

где *n* - число вершин и *m* - число дуг графа *G*.

Для неориентированного графа *G=*(*X,*Г) *степень* вершины *xi* определяется аналогично - с помощью соотношения *deg*(*xi*) Г(*xi*)=Г-1(*xi*).

*Петлей* называется дуга, начальная и конечная вершины которой совпадают. На рисунке 3, например, дуги *a3* и *a10* являются петлями.

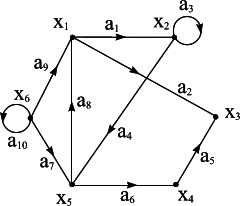
2.2 МАТРИЧНЫЕ ПРЕДСТАВЛЕНИЯ

2.2.1 МАТРИЦА СМЕЖНОСТИ

Пусть дан граф *G*, его матрица смежности обозначается через **A***=*[*aij*]и определяется следующим образом:

*aij*=1, если в *G* существует дуга (*xi*,*xj*),

*aij*=0, если в *G* нт дуги (*xi*,*xj*).

Таким образом, матрица смежности графа, изображенного на рисунке 3, имеет вид

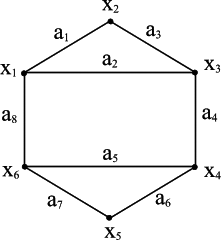
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | x1 | x2 | x3 | x4 | x5 | x6 |
|  | x1 | **0** | **1** | **1** | **0** | **0** | **0** |
|  | x2 | **0** | **1** | **0** | **0** | **1** | **0** |
| A= | x3 | **0** | **0** | **0** | **0** | **0** | **0** |
|  | x4 | **0** | **0** | **1** | **0** | **0** | **0** |
|  | x5 | **1** | **0** | **0** | **1** | **0** | **0** |
|  | x6 | **1** | **0** | **0** | **0** | **1** | **1** |

Рисунок 3.

Матрица смежности полностью определяет структуру графа. Например, сумма всех элементов строки *xi* матрицы дает полустепень исхода вершины *xi*, а сумма элементов столбца *xi* - полустепень захода вершины *xi*. Множество столбцов, имеющих 1 в строке *xi* есть множество Г(*xi*), а множество строк, которые имеют 1 в столбце *xi* совпадает с множеством Г-1(*xi*).

Петли на графе представляют собой элементы, имеющие 1 на главной диагонали матрицы, например *a22*, *a66* для графа, изображенного на рисунке 3.

В случае неориентированного графа матрица смежности является симметричной относительно главной диагонали (рисунок 4).



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | x1 | x2 | x3 | x4 | x5 | x6 |
|  | x1 | **0** | **1** | **1** | **0** | **0** | **1** |
|  | x2 | **1** | **0** | **1** | **0** | **0** | **0** |
| A= | x3 | **1** | **1** | **0** | **1** | **0** | **0** |
|  | x4 | **0** | **0** | **1** | **0** | **1** | **1** |
|  | x5 | **0** | **0** | **0** | **1** | **0** | **1** |
|  | x6 | **1** | **0** | **0** | **1** | **1** | **0** |

Рисунок 4.

2.2.2 МАТРИЦА ИНЦИДЕНТНОСТИ

Пусть дан граф *G* с *n* вершинами и *m* дугами. *Матрица инцидентности* графа *G* обозначается через **B**=[*bij*] и является матрицей размерности *n x m*, определяемой следующим образом:

*bij*=1, если *xi* является начальной вершиной дуги *aj*;

*bij*=-1, если *xi* является конечной вершиной дуги *aj*;

*bij*=0, если *xi* не является концевой вершиной дуги *aj*.

Для графа, приведенного на рисунке 3, матрица инцидентности имеет вид:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | a1 | a2 | a3 | a4 | a5 | a6 | a7 | a8 | a9 | a10 |
|  | x1 | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **-1** | **-1** | **0** |
|  | x2 | **-1** | **0** | **±1** | **1** | **0** | **0** | **0** | **0** | **0** | **0** |
| B= | x3 | **0** | **-1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
|  | x4 | **0** | **0** | **0** | **0** | **-1** | **-1** | **0** | **0** | **0** | **0** |
|  | x5 | **0** | **0** | **0** | **-1** | **1** | **1** | **-1** | **1** | **0** | **0** |
|  | x6 | **0** | **0** | **0** | **0** | **0** | **0** | **1** | **0** | **1** | **±1** |

Поскольку каждая дуга инцидентна двум различным вершинам (за исключением случая, когда дуга образует петлю), то каждый столбец содержит один элемент, равный 1, и один - равный -1. Петля в матрице инцидентности не имеет адекватного математического представления (в программной реализации допустимо задание одного элемента *bij*=1).

Если *G* является неориентированным графом (рисунок 4), то его матрица инцидентности определяется следующим образом:

*bij*=1, если *xi* является концевой вершиной дуги *aj*;

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | a1 | a2 | a3 | a4 | a5 | a6 | a7 | a8 |
|  | x1 | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **1** |
|  | x2 | **1** | **0** | **1** | **0** | **0** | **0** | **0** | **0** |
| B= | x3 | **0** | **1** | **1** | **1** | **0** | **0** | **0** | **0** |
|  | x4 | **0** | **0** | **0** | **1** | **1** | **1** | **0** | **0** |
|  | x5 | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **0** |
|  | x6 | **0** | **0** | **0** | **0** | **1** | **0** | **1** | **1** |

*bij*=0, если *xi* не является концевой вершиной дуги *aj*.

Матрица инцидентности, как способ задания графов, успешно применяется при описании мультиграфов (графов, в которых смежные вершины могут соединяться несколькими параллельными дугами).

3 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

3.1 Получить задание у преподавателя в виде одного из двух способов матричного представления графа:

а) матрица смежности; б) матрица инцидентности

3.2 Составить алгоритм программы, реализующей перевод из заданного способа матричного представления графа в другой, учитывая при этом исходный тип графа (неориентированный, ориентированный, смешанный).

3.3 Создать программу, реализующую перевод из заданного способа матричного представления графа в другой. Предусмотреть консольный ввод исходных данных и вывод результатов работы программы на экран.

4 СОДЕРЖАНИЕ ОТЧЕТА ПО РАБОТЕ

4.1 Исходное задание и цель работы.

4.2 Блок-схема программы по п.3.2.

4.3 Распечатка текста программы по п.3.3.

4.4 Контрольный пример и результаты машинного расчета.

4.5 Выводы по работе.

5 КОНТРОЛЬНЫЕ ВОПРОСЫ

5.1 Перечислите основные способы представления графов.

5.2 Покажите на примере прямое и обратное соответствие для заданной вершины.

5.3 Чему равна сумма степеней всех вершин неориентированного графа?

5.4 В чем отличия матричного представления ориентированных и неориентированных графов?

5.5 В чем особенности представления графа матрицей смежности?

5.6 В чем особенности представления графа матрицей инцидентности?

**ЛАБОРАТОРНАЯ РАБОТА № 3-4**

ПОИСК КРАТЧАЙШИХ ПУТЕЙ НА ГРАФАХ

1 ЦЕЛЬ РАБОТЫ

Целью работы является изучение алгоритмов поиска кратчайших путей на графах на примере метода динамического программирования.

2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1 ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

*Путем* (или *ориентированным маршрутом*) ориентированного графа называется последовательность дуг, в которой конечная вершина всякой дуги, отличной от последней, является начальной вершиной следующей. Так на рисунке 5 последовательности дуг *μ1=*{*a6*, *a5*, *a9*, *a8*, *a4*}, *μ2=*{*a1*, *a6*, *a5*, *a9*}, *μ3=*{*a1*, *a6*, *a5*, *a9*, *a10*, *a6*, *a4* } являются путями.

*Ориентированной* *цепью* (*орцепью*) называется такой путь, в котором каждая дуга используется не больше одного раза. Так, например, приведенные выше пути *μ1* и *μ2* являются орцепями, а путь *μ3* не является таким, поскольку дуга *a6* в нем используется дважды.

*Маршрут* есть неориентированный "двойник" пути, и это понятие рассматривается в тех случаях, когда можно пренебречь направленностью дуг в графе.

Таким образом, маршрут есть последовательность ребер a*1*, a*2*,...,a*q*, в которой каждое ребро a*i*, за исключением, возможно, первого и последнего ребер, связано с ребрами *ai-1* и *ai+1* своими двумя концевыми вершинами.

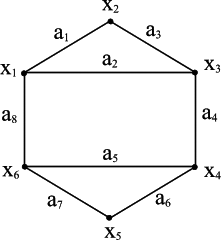
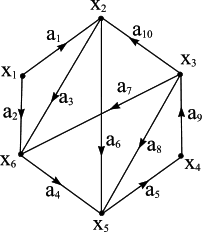
 

Рисунок 5. Рисунок 6.

Последовательности дуг на рисунке 6

*μ4=*{a*2*,a*4,* a*8,* a*10*}, *μ5=*{a*2*,a*7,* a*8,* a*4,* a*3*} и *μ6=*{a*10*,a*7,* a*4,* a*8,* a*7,* a*2*}

являются маршрутами.

*Контуром (простой цепью)* называется такой путь (маршрут), в котором каждая вершина используется не более одного раза. Например, путь *μ2* является контуром, а пути *μ1* и *μ3* - нет. Очевидно, что контур является также цепью, но обратное утверждение неверно. Например, путь *μ1* является цепью, но не контуром, путь *μ2* является цепью и контуром, а путь *μ3* не является ни цепью, ни контуром.

Аналогично определяется простая цепь в неориентированных графах. Так, например, маршрут *μ4* есть простая цепь, маршрут *μ5* - цепь, а маршрут *μ6* не является цепью.

Путь или маршрут можно изображать также последовательностью вершин. Например, путь *μ1* можно представить также: *μ1*={*x2, x5, x4, x3, x5, x6*} и такое представление часто оказывается более полезным в тех случаях, когда осуществляется поиск контуров или простых цепей.

Иногда дугам графа *G* сопоставляются (приписываются) числа - дуге (*xi*,*xj*) ставится в соответствие некоторое число *cij*, называемое *весом*, или *длиной*, или *стоимостью* (*ценой*) дуги. Тогда граф *G* называется *взвешенным*. Иногда веса (числа *vi*) приписываются вершинам *xi* графа.

При рассмотрении пути *μ*, представленного последовательностью дуг (a*1*, a*2*,...,a*q*), за его *вес* (или *длину*, или *стоимость*) принимается число *L*(*µ*), равное сумме весов всех дуг, входящих в *μ*, т. е.

 (2)

Таким образом, когда слова "длина", "стоимость", "цена" и "вес" применяются к дугам, то они эквивалентны по содержанию, и в каждом конкретном случае выбирается такое слово, которое ближе подходит по смыслу.

*Длиной* (или *мощностью*) пути и называется число дуг, входящих в него.

2.2 ЗАДАЧА О КРАТЧАЙШЕМ ПУТИ

Пусть дан граф G=(X,Г), дугам которого приписаны веса (стоимости), задаваемые матрицей С=[сij]. *Задача о кратчайшем пути* состоит в нахождении кратчайшего пути от заданной начальной вершины (*истока*) s до заданной конечной вершины (*стока*) t, при условии, что такой путь существует:

Найти *µ*(*s,t*) при *L*(*µ*)→*min*, s,tХ, tR(s),

где R(s) - множество, достижимое из вершины s.

В общем случае элементы сij матрицы весов С могут быть положительными, отрицательными или нулями. Единственное ограничение состоит в том, чтобы в G не было циклов с суммарным отрицательным весом. Отсюда следует, что дуги (ребра) графа G не должны иметь отрицательные веса.

Почти все методы, позволяющие решить задачу о кратчайшем (s-t)-пути, дают также (в процессе решения) и все кратчайшие пути от s к xi(∀xiХ). Таким образом, они позволяют решить задачу с небольшими дополнительными вычислительными затратами.

Допускается, что матрица весов С не удовлетворяет условию треугольника, т. е. не обязательно сijсik +сkj для всех i, j и k.

Если в графе G дуга (xi, xj) отсутствует, то ее вес полагается равным .

Ряд задач, например, задачи нахождения в графах путей с максимальной надежностью и с максимальной пропускной способностью, связаны с задачей о кратчайшем пути, хотя в них характеристика пути (скажем, вес) является не суммой, а некоторой другой функцией характеристик (весов) дуг, образующих путь. Такие задачи можно переформулировать как задачи о кратчайшем пути и решать их соответствующим образом.

Существует множество методов решения данной задачи, отличающиеся областью применимости и трудоемкостью (Дейкстры, Флойда, динамического программирования). Среди них большое распространение получили частные алгоритмы, применяющиеся при решении частных задач, и имеющие меньшую трудоемкость. Эти частные случаи встречаются на практике довольно часто (например, когда сij являются расстояниями), так что рассмотрение этих специальных алгоритмов оправдано.

На практике задачу кратчайшего пути часто требуется решать для класса ориентированных ациклических графов. Такая задача успешно решается с помощью метода динамического программирования.

2.3 МЕТОД ДИНАМИЧЕСКОГО ПРОГРАММИРОВАНИЯ

*Прямая итерация.* Пусть вершины пронумерованы так, что дуга (*xi,xj*) всегда ориентирована от вершины *xi* к вершине *xj*, имеющей больший номер. Для ациклического графа такая нумерация всегда возможна и производится очень легко. При этом начальная вершина *s* получает номер *1*, а конечная *t* - номер *n*.

Пусть *λ*(*xi*) – пометка вершины *xi*, равная длине кратчайшего пути от *1* до *xi*, *s* – начальная вершина (источник), *t* – конечная вершина (сток).

Шаг 1. Положить *λ*(*s*)=*0, λ*(*xi*) =  для всех вершин *xi* ∈ *X/s;* *i=1;*

Шаг 2. *i=i+1.* Присвоим вершине *xj* пометку *λ*(*xj*),  равную длине кратчайшего пути от *1* до *xj*, используя для этого соотношение

 (3)

Шаг 3. Повторить п.2. до тех пор, пока последняя вершина *n* не получит пометку *λ*(*t*).

Необходимо отметить, что если вершина *xj* помечена, то пометки *λ*(*xi*) известны для всех вершин *xi*Г-1(*xj*), так как в соответствии со способом нумерации это означает, что *xi**xj* и, следовательно, вершины *xi* уже помечены в процессе применения алгоритма.

Пометка *λ*(*t*) равна длине самого короткого пути от *s* до *t*. Сами дуги, образующие путь, могут быть найдены способом последовательного возвращения. А именно дуга (*xi,xj*), согласно (3), принадлежит пути тогда и только тогда, когда

*λ*(*xj*)*= λ*(*xi*)*+cij* (4)

*Обратная итерация*: начиная с вершины *t,* имеющей номер *n*, полагаем на каждом шаге *xj* равной такой вершине (скажем, *xj\**), для которой выполняется соотношение (4), и так продолжаем до тех пор, пока не будет достигнута начальная вершина (т.е. пока не будет *xj\*≡s*).

Совершенно очевидно, что пометка *λ*(*xj*) вершины *xj* дает длину кратчайшего пути *μ* от *s* до *xj*.

2.4 АЛГОРИТМ ТОПОЛОГИЧЕСКОЙ СОРТИРОВКИ

В некоторых случаях исходный граф является ациклическим, но имеет неправильную нумерацию – содержит дуги (*xj,xi*), ориентированные от вершины *xj* к вершине *xi*, имеющей меньший номер (*j*>*i*). Для успешного нахождения кратчайшего пути с помощью метода динамического программирования к такому графу сначала применяется алгоритм топологической сортировки вершин.

Алгоритм топологической сортировки вершин очень простой. Он позволяет не только правильно перенумеровать вершины графа, но и определить его ацикличность.

Шаг 1. Положить *i=n*, где *n* – число вершин графа *G.*

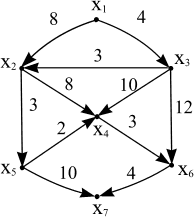
Шаг 2. В графе определяется вершина *xk*, для которой выполняется условие |*Г*(*xk*)|=∅ (т.е., вершина, из которой не выходит ни одна дуга). Вершина *xk* получает порядковый номер *i* (перенумеруется) и исключается из дальнейшего рассмотрения вместе со всеми входящими в нее инцидентными дугами. *i=i–1.*

Шаг 3. Повторять п.2. до тех пор, пока не будет выполнено одно из условий:

1. *i=1* – достигнута начальная вершина. Вершины графа получили правильную нумерацию.
2. Невозможно определить вершину, для которой выполнялось бы условие |*Г*(*xk*)|=∅. В графе имеется цикл.

В последнем случае алгоритм динамического программирования неприменим. Для поиска кратчайших путей на таком графе необходимо использовать более эффективные методы, например, алгоритм Дейкстры [].

2.5 КОНТРОЛЬНЫЙ ПРИМЕР

Для графа, изображенного на рисунке 7, определим кратчайший путь между вершинами *x1*  и *x7*, используя метод динамического программирования.

Так как граф содержит дуги (*x3 ,x2*) и (*x5 ,x4*), имеющие неправильную нумерацию (от большего к меньшему), необходимо перенумеровать вершины графа, применяя алгоритм топологической сортировки вершин.

В нашем случае из графа будут последовательно исключаться вершины *x7*, *x6* , *x4*, *x5*, *x2*, *x3*, *x1.* Соответственно, вершины графа получат новую нумерацию, и граф будет иметь вид, представленный на рисунке 8 (старая нумерация вершин сохранена в скобках).

Рисунок 7.

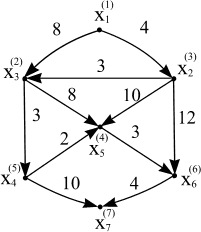
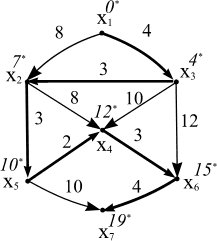
На первом шаге оценка *λ*(*x1*)=*0*. Переходим к вершине *x2*. Множество Г-1(*x2*) включает только одну вершину *x1*. Следовательно, оценка для вершины *x2* определяемая по формуле (3), будет *λ*(*x2*)=*min*{*0*+*4*}=*4*. Переходим к вершине *x3*. Для вершины *x3* множество Г-1(*x3*)={*x1, x2*}. В этом случае, оценка будет выбираться как минимальная из двух возможных: *λ*(*x3*)=*min*{*0*+*8*, *4*+*3*}=*7.* Для вершины *x4* оценка определяется снова однозначно: *λ*(*x4*)=*min*{*7*+*3*}=*10*. Однако при переходе к вершине *x5* мы получаем сразу три входящих дуги (*x2, x5*), (*x3, x5),* (*x4, x5*). Применяя формулу (3), определяем оценку для вершины *x5*:

Рисунок 8.

*λ*(*x5*)=*min*{*4*+*10*, *7*+*8*, *10*+*2*}=*12.*

Далее, аналогичным образом вершина *x6* получает оценку *λ*(*x6*)=*min*{*4*+*12*, *12*+*3*}=*15* и, наконец, вершина *x7* получает оценку *λ*(*x7*)=*min*{*10*+*10*, *15*+*4*}=*19*.

Таким образом, конечная вершина *x7*  пути *µ*(x1,x7) достигнута, длина пути равна *L*(*µ*)=*19.*

Применяя выражение (4), последовательно определяем вершины, которые входят в кратчайший путь. Перемещаясь от конечной вершины *x7*, выбираем последовательность вершин, для которой выражение (4) принимает значение «истинно»: *x6, x5, x4, x3, x2, x1*, т.е. кратчайший путь проходит последовательно через все вершины графа.

Рисунок 9.

Действительно, легко убедиться в истинности выражений:

*λ*(*x7*)= *λ*(*x6*)+ *c67*,(*19*=*15*+*4*);

*λ*(*x6*)= *λ*(*x5*)+ *c56*, (*15*=*12*+*3*);

*λ*(*x5*)= *λ*(*x4*)+ *c45*, (*12*=*10*+*2*);

*λ*(*x4*)= *λ*(*x3*)+ *c34*, (*10*=*7*+*3*);

*λ*(*x3*)= *λ*(*x2*)+ *c23*,(*7*=*4*+*3*);

*λ*(*x2*)= *λ*(*x1*)+ *c12*, (*4*=*0*+*4*);

Произведя перенумерацию вершин графа на исходную, окончательно определяем кратчайший путь:

*µ*(x1,x7)={*x1, x3, x2, x5, x4, x6, x7*}.

Задача решена.

Результат решения в виде выделенного пути изображен на рисунке 9. Курсивом «со звездочкой» отмечены значения пометок вершин *λ*(*xi*).

3 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

3.1 Получить задание у преподавателя в виде исходного ориентированного графа.

3.2 Составить блок-схему программы, определяющей кратчайший путь на графе от заданной начальной вершины *s* до заданной конечной вершины *t* с помощью метода динамического программирования.

3.3 Составить блок-схему программы, реализующей алгоритм топологической сортировки с произвольной нумерацией вершин графа.

3.4 Создать программу, реализующую метод динамического программирования и алгоритм топологической сортировки вершин. Исходный граф задается в виде матрицы смежности, вводимой построчно с помощью консоли. Указание: для определения вершин, входящих в множество Г-1(*xi*) используйте *j-*йстолбец матрицы смежности.

4 СОДЕРЖАНИЕ ОТЧЕТА ПО РАБОТЕ

4.1 Исходное задание и цель работы.

4.2 Блок-схема программы по п.3.2.

4.3 Блок-схема программы по п.3.3.

4.3 Распечатка текста программы по п.3.4.

4.4 Контрольный пример и результаты машинного расчета.

4.5 Выводы по работе.

5 КОНТРОЛЬНЫЕ ВОПРОСЫ

5.1 Дайте определение пути, маршрута, цепи, контура.

5.2 Какой граф называется взвешенным?

5.3 Как определяется длина пути графа?

5.4 Задача нахождения кратчайшего пути на графе.

5.5 Реализация метода динамического программирования для нахождения кратчайшего пути на графе.

5.6 Ограничения применения метода динамического программирования для нахождения кратчайшего пути на графе.

5.7 Что называется правильной нумерацией вершин графа?

5.8 Применение алгоритма топологической сортировки для перенумерации вершин графа.

**ЛАБОРАТОРНАЯ РАБОТА № 5**

ПОСТРОЕНИЕ КРАТЧАЙШИХ ОСТОВЫХ ДЕРЕВЬЕВ ГРАФА

1 ЦЕЛЬ РАБОТЫ

Целью работы является изучение метода построения кратчайших остовых деревьев графа на примере алгоритма Прима-Краскала.

2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1 ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

Одним из наиболее важных понятий теории графов является *дерево*.

*Неориентированным деревом* называется связанный граф, не имеющий циклов.

*Суграфом* графа G является подграф Gp, содержащий все вершины исходного графа.

Если G=(X,A) - неориентированный граф с n вершинами, то связный суграф Gp, не имеющий циклов,называется *остовым деревом* (*остовом*) *графа* G.

Для остового дерева справедливо соотношение:

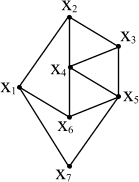
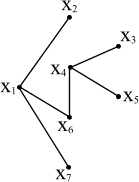
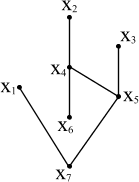
Gp=(Xp,Ap) ⊆ G, где Xp = X, Ap ⊆ A (5)

Легко доказать, что остовое дерево имеет следующие свойства:

1. остовое дерево графа с n вершинами имеет n-1 ребро (|Xp|=|Ap|-1);
2. существует единственный путь, соединяющий любые две вершины остова графа:

∀xi,xj ∈Xp (i≠j) → ∃! μ(xi,xj).

Например, если G - граф, показанный на рисунке 10(a), то графы на рисунках 10(б,в) являются остовами графа G. Из сформулированных выше определений вытекает, что остов графа G можно рассматривать как минимальный связанный остовый подграф графа G.

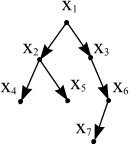
  

(а) (б) (в)

Рисунок 10 – Представление графа в виде остовых деревьев.

Понятие дерева как математического объекта было впервые предложено Кирхгофом в связи с определением фундаментальных циклов, применяемых при анализе электрических цепей. Приблизительно десятью годами позже Кэли вновь (независимо от Кирхгофа) ввел понятие дерева и получил большую часть первых результатов в области исследования свойств деревьев. Большую известность получила его знаменитая теорема:

*Теорема Кэли*. На графе с n вершинами можно построить nn-2 остовых деревьев.

*Ориентированное дерево* представляет собой ориентированный граф без циклов, в котором полустепень захода каждой вершины, за исключением одной (вершины r), равна единице, а полустепень захода вершины r (называемой корнем этого дерева) равна нулю.

На рисунке 11 показан граф, который является ориентированным деревом с корнем в вершине *x1*. Из приведенного определения следует, что ориентированное дерево с n вершинами имеет n-1 дуг и связно.

Рисунок 11.

Неориентированное дерево можно преобразовать в ориентированное: надо взять его произвольную вершину в качестве корня и ребрам приписать такую ориентацию, чтобы каждая вершина соединялась с корнем (только одной) простой цепью.

"Генеалогическое дерево", в котором вершины соответствуют лицам мужского пола, а дуги ориентированы от родителей к детям, представляет собой хорошо известный пример ориентированного дерева. Корень в этом дереве соответствует "основателю" рода (лицу, родившемуся раньше остальных).

2.2 КРАТЧАЙШИЙ ОСТОВ ГРАФА

В лабораторной работе исследуется метод прямого построения *кратчайших остовых деревьев* во взвешенном графе (в котором веса приписаны дугам). Кратчайшее остовое дерево графа находит применение при прокладке дорог (газопроводов, линий электропередач и т. д.), когда необходимо связать n точек некоторой сетью так, чтобы общая длина "линий связи" была минимальной. Если точки лежат на евклидовой плоскости, то их можно считать вершинами полного графа G с весами дуг, равными соответствующим "прямолинейным" расстояниям между концевыми точками дуг. Если "разветвление" дорог допускается только в заданных n точках, кратчайшее остовое дерево графа G будет как раз требуемой сетью дорог, имеющей наименьший вес.

Рассмотрим взвешенный связный неориентированный граф G=(X,А); вес ребра (xi,xj) обозначим cij. Из большого числа остовов графа нужно найти один, у которого сумма весов ребер наименьшая. Такая задача возникает, например, в том случае, когда вершины являются клеммами электрической сети, которые должны быть соединены друг с другом с помощью проводов наименьшей общей длины (для уменьшения уровня наводок). Другой пример: вершины представляют города, которые нужно связать сетью трубопроводов; тогда наименьшая общая длина труб, которая должна быть использована для строительства (при условии, что вне городов "разветвления" трубопроводов не допускаются), определяется кратчайшим остовом соответствующего графа.

Задача построения кратчайшего остова графа является одной из немногих задач теории графов, которые можно считать полностью решенными.

2.3 АЛГОРИТМ ПРИМА-КРАСКАЛА

Этот алгоритм порождает остовое дерево посредством разрастания только одного поддерева, например Xp, содержащего больше одной вершины. Поддерево постепенно разрастается за счет присоединения ребер (xi,xj), где xiXp и xjXp; причем добавляемое ребро должно иметь наименьший вес cij. Процесс продолжается до тех пор, пока число ребер в Ap не станет равным n-1. Тогда поддерево Gp=(Xp,Ap) будет требуемым остовым деревом. Впервые такая операция была предложена Примом и Краскалом (с разницей – в способе построения дерева), поэтому данный алгоритм получил название Прима-  
Краскала.

Алгоритм начинает работу с включения в поддерево начальной вершины. Поскольку остовое дерево включает все вершины графа G, то выбор начальной вершины не имеет принципиального значения. Будем каждой очередной вершине присваивать пометку **(xi)=1, если вершина xi принадлежит поддереву Xp и **(xj)=0 – в противном случае.

Алгоритм имеет вид:

Шаг 1. Пусть Xp={x1}, где x1 - начальная вершина, и Ap= (Ap является множеством ребер, входящих в остовое дерево). Вершине x1 присвоить пометку **(x1)=1. Для каждой вершины xiXp присвоить **(xi)=0.

Шаг 2. Из всех вершин xjГ(Xp), для которых **(xj)=0, найти вершину xj\* такую, что

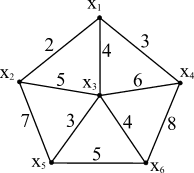
, где xiXp и xjXp. (6)

Шаг 3. Обновить данные: Xp= Xp {xj\*}; Ap= Ap . Присвоить **(xj\*)=1.

Шаг 4. Если |Xp|=n, то остановиться. Ребра в Ap образуют кратчайший остов графа.   
Если |Xp|<n, то перейти к шагу 2.

2.4 КОНТРОЛЬНЫЙ ПРИМЕР

Для примера рассмотрим граф, изображенный на рисунке 12. Найдем для него кратчайшее остовое дерево, используя для этой цели рассмотренный выше алгоритм Прима-Краскала. Обозначим множество смежных вершин, не входящих в порожденное поддерево, как Г \*(Xp). Таким образом, для всех вершин, входящих в это множество, оценка **(xj)=0, ∀xj ∈ Г \*(Xp). Вектор B представляет собой множество оценок **(xi) для всех вершин графа G: ∀xi ∈X. Длина порожденного поддерева обозначается как L.

Итерация 1: Xp={x1}; Ap=; Г\*(Xp)={*x2, x3, x4*};

*c*(*x1, x2\**)=*2*; *B=*{*1,1,0,0,0,0*}; L*=2*.

Итерация 2: Xp={x1*, x2*}; Ap={(x1*, x2*)};

Г \*(Xp)={*x3, x4, x5*}; *c*(*x1,x4\**)=*3*;

*B=*{*1,1,0,10,0*}; L*=2+3=5*.

Итерация 3: Xp={x1*, x2, x4*}; Ap={(x1*, x2*); (x1*, x4*)};

Г \*(Xp)={*x3, x5, x6*}; *c*(*x1,x3\**)=*4*;

Рисунок 12.

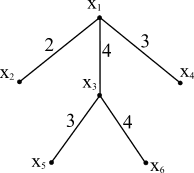
*B=*{*1,1,1,1,0,0*}; L*=5+4*.

Итерация 4: Xp={x1, x2*, x3, x4*}; Ap={(x1*, x2*); (x1*, x4*); (*x1,x3*)};

Г \*(Xp)={*x5, x6*};  *c*(*x3, x5\**)=*2*; *B=*{*1,1,1,1,1,0*}; L*=9+3=12*.

Итерация 5: Xp={x1, x2*, x3, x4,*x5}; Ap={(x1*, x2*); (x1*, x4*); (*x1,x3*); (*x3,x5*)};

Г \*(Xp)={*x6*}; *c*(*x3,x6\**)=*4*; *B=*{*1,1,1,1,1,1*}; L*=12+4=16*.



Задача решена. Полученные множества вершин Xp и ребер Ap составляют кратчайшее остовое дерево:

Xp={x1, x2*, x3, x4,*x5*,*x6};

Ap={(x1*, x2*); (x1*, x4*); (*x1,x3*); (*x3,x5*); (*x3,x6*)};

Суммарная длина кратчайшего остового дерева L*=16.*

Рисунок 13.

Результат решения задачи представлен на рисунке 13.

3 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

3.1 Получить задание у преподавателя в виде исходного неориентированного графа:

3.2 Составить блок-схему программы, определяющей кратчайшее остовое дерево графа с помощью алгоритма Прима-Краскала.

3.3 Создать программу, реализующую алгоритм Прима-Краскала. Исходный граф задается в виде матрицы смежности, вводимой построчно с помощью консоли. Программа должна вывести список ребер, входящих в кратчайшее остовое дерево.

4 СОДЕРЖАНИЕ ОТЧЕТА ПО РАБОТЕ

4.1 Исходное задание и цель работы.

4.2 Блок-схема программы по п.3.2.

4.3 Распечатка текста программы по п.3.3.

4.4 Контрольный пример и результаты машинного расчета.

4.5 Выводы по работе.

5 КОНТРОЛЬНЫЕ ВОПРОСЫ

5.1 Дайте определение дерева; ориентированного дерева.

5.2 Какое дерево называется остовым?

5.3 Свойства остовых деревьев. Теорема Кэли.

5.4 Что называется корнем дерева?

5.5 Как преобразовать неориентированное дерево в ориентированное?

5.6 Сколько ребер содержит остовое дерево графа?

5.7 Задача нахождения кратчайшего остова графа.

5.8 Приведите практические примеры нахождения кратчайшего остова графа.

5.9 Реализация алгоритма Прима-Краскала для нахождения кратчайшего остова графа.

**ЛАБОРАТОРНАЯ РАБОТА №6**

РАСКРАСКА ГРАФА

1 ЦЕЛЬ РАБОТЫ

Целью работы является изучение способа правильной раскраски графа на основе эвристического алгоритма.

2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1 ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

Разнообразные задачи, возникающие при планировании производства, составлении графиков осмотра, хранении и транспортировке товаров и т.д., могут быть представлены часто как задачи теории графов, тесно связанные с так называемой "задачей раскраски". Графы, рассматриваемые в данной лабораторной работе, являются неориентированными и не имеют петель.

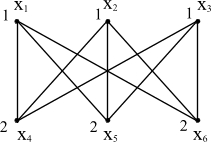
Граф *G* называют *r*-*хроматическим*, если его вершины *могут быть раскрашены* с использованием *r* цветов (красок) так, что не найдется двух смежных вершин одного цвета. Наименьшее число *r*, такое, что граф *G* является *r*-*хроматическим*, называется *хроматическим числом* графа G и обозначается (*G*). Задача нахождения хроматического числа графа называется *задачей о раскраске* (или *задачей раскраски*) графа. Соответствующая этому числу раскраска вершин разбивает множество вершин графа на *r* подмножеств, каждое из которых содержит вершины одного цвета. Эти множества являются независимыми, поскольку в пределах одного множества нет двух смежных вершин.

Задача нахождения хроматического числа произвольного графа явилась предметом многих исследований в конце XIX и в XX столетии. По этому вопросу получено много интересных результатов.

Хроматическое число графа нельзя найти, зная только числа вершин и ребер графа. Недостаточно также знать степень каждой вершины, чтобы вычислить хроматическое число графа. При известных величинах *n* (число вершин), *m* (число ребер) и *deg*(*x*1),...,*deg*(*x*n) (степени вершин графа) можно получить только верхнюю и нижнюю оценки для хроматического числа графа.

Пример раскраски графа приведен на рисунке 14. Этот граф является одной из запрещенных фигур, используемых для определения планарности. Цифрами “1” и “2” обозначены цвета вершин.

*Максимальное число независимых вершин* графа (*G*), равное мощности наибольшего множества попарно несмежных вершин, совпадает также с мощностью наибольшего множества вершин в *G*, которые могут быть окрашены в один цвет, следовательно:



, (7)

где *n -* число вершин графа *G*, а  обозначает наибольшее целое число, не превосходящее *x*.

Еще одна нижняя оценка для (*G*) может быть получена следующим образом:

Рисунок 14 – Двудольный бихроматический граф Кенига.

. (8)

Верхняя оценка хроматического числа может быть вычислена по формуле:

. (9)

Применение оценок для хроматического числа значительно сужает границы решения. Для определения оценки хроматического числа также могут использоваться другие топологические характеристики графа, например, свойство планарности.

Граф, который можно изобразить на плоскости так, что никакие два его ребра не пересекаются между собой, называется *планарным*.

*Теорема о пяти красках.* Каждый планарный граф можно раскрасить с помощью пяти цветов так, что любые две смежные вершины будут окрашены в разные цвета, т. е. если граф *G* - планарный, то (*G*)  *5*.

*Гипотеза о четырех красках (недоказанная).* Каждый планарный граф можно раскрасить с помощью четырех цветов так, что любые две смежные вершины будут окрашены в разные цвета, т. е. (*G*)  *4*, если граф *G* - планарный.

В 1852 г. о гипотезе четырех красок говорилось в переписке Огюста де Моргана с сэром Вильямом Гамильтоном. С тех пор эта "теорема" стала, наряду с теоремой Ферма, одной из самых знаменитых нерешенных задач в математике.

*Полный граф Gn* всегда раскрашивается в *n* цветов, равных количеству его вершин.

2.2 ЭВРИСТИЧЕСКИЕ АЛГОРИТМЫ РАСКРАШИВАНИЯ

Точные методы раскраски графа сложны для программной реализации. Однако существует много эвристических процедур раскрашивания, позволяющих находить хорошие приближения для определения хроматического числа графа. Такие процедуры также могут с успехом использоваться при раскраске графов с большим числом вершин, где применение точных методов не оправдано ввиду высокой трудоемкости вычислений.

Из эвристических процедур раскраски следует отметить последовательные методы, основанные на упорядочивании множества вершин.

В одном из простейших методов вершины вначале располагаются в порядке убывания их степеней. Первая вершина окрашивается в цвет 1; затем список вершин просматривается по убыванию степеней и в цвет 1 окрашивается каждая вершина, которая не является смежной с вершинами, окрашенными в тот же цвет. Потом возвращаемся к первой в списке неокрашенной вершине, окрашиваем ее в цвет 2 и снова просматриваем список вершин сверху вниз, окрашивая в цвет 2 любую неокрашенную вершину, которая не соединена ребром с другой, уже окрашенной в цвет 2 вершиной. Аналогично действуем с цветами 3, 4 и т. д., пока не будут окрашены все вершины. Число использованных цветов будет тогда приближенным значением хроматического числа графа.

Эвристический алгоритм раскраски вершин графа имеет следующий вид:

Шаг 1. Сортировать вершины графа по степеням убывания:

*deg*(*xi*) ≥ *deg*(*xj*), ∀xi,xj ∈ *G*; Установить текущий цвет *p=1; i=1.*

Шаг 2. Выбрать очередную не раскрашенную вершину из списка и назначить ей новый цвет: *col*(*xi*)=*p*; Xp={*xi*}*.*

Шаг 3. *i=i+1.* Выбрать очередную не раскрашенную вершину *xi* и проверить условие смежности: *xi* ∩Г(Xp)=∅, где Xp – множество вершин, уже раскрашенных в цвет *p.* Если вершина *xi* не является смежной с данными вершинами, то также присвоить ей цвет *p: col*(*xi*)=*p.*

Шаг 4. Повторить п.3, пока не будет достигнут конец списка (*i=n*).

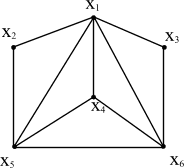
Шаг 5. Если все вершины графа раскрашены, то – конец алгоритма;

Иначе: *p=p+1; i=1.* Повторить п.2.

2.4 КОНТРОЛЬНЫЙ ПРИМЕР

Раскрасим граф *G*, изображенный на рисунке 15. Промежуточные данные для решения задачи будем записывать в таблицу. Отсортируем вершины графа по убыванию их степеней. В результате получается вектор отсортированных вершин *X*\*={x1, x5*, x6, x4,*x2*,*x3}.

Соответствующие данным вершинам степени образуют второй вектор:

*D*={5, 4*, 4, 3, 2, 2*}. В первой строке таблицы запишем вектор *X*\*; во второй – вектор *D*. Последующие строки отражают содержание вектора раскраски *col*(*X*\*)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Номера вершин *X*\* | x1 | x5 | *x6* | *x4* | x2 | x3 |
| Степени вершин *D* | 5 | 4 | 4 | 3 | 2 | 2 |
| *p=1* | 1 | - | - | - | - | - |
| *p=2* | 1 | 2 | - | - | - | 2 |
| *p=3* | 1 | 2 | 3 | - | 3 | 2 |
| *p=4* | 1 | 2 | 3 | 4 | 3 | 2 |

Рисунок 14.

Таким образом, данный граф можно раскрасить не менее чем в четыре цвета, т.е. (*G*)=*4*.

3 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

3.1 Получить задание у преподавателя в виде исходного неориентированного графа:

3.2 Составить блок-схему программы, определяющей раскраску графа с помощью эвристического алгоритма, указанного в п.2.2.

3.3 Создать программу, реализующую эвристический алгоритм раскраски графа. Исходный граф задается в виде матрицы смежности, вводимой построчно с помощью консоли. Программа должна вывести список полученных цветов для всех вершин графа.

4 СОДЕРЖАНИЕ ОТЧЕТА ПО РАБОТЕ

4.1 Исходное задание и цель работы.

4.2 Блок-схема программы по п.3.2.

4.3 Распечатка текста программы по п.3.3.

4.4 Контрольный пример и результаты машинного расчета.

4.5 Выводы по работе.

5 КОНТРОЛЬНЫЕ ВОПРОСЫ

5.1 Сформулируйте задачу раскраски графа.

5.2 Какой граф называется r-хроматическим?

5.3 Что называется хроматическим числом графа?

5.4 Как определяются нижняя и верхняя оценки хроматического числа?

5.5 Какой граф называется планарным? Во сколько цветов можно его раскрасить?

5.6 Во сколько цветов можно раскрасить полный граф?

5.7 Эвристический алгоритм раскраски графа.

5.9 Реализация эвристического алгоритма для нахождения хроматического числа графа.

**ЛАБОРАТОРНАЯ РАБОТА № 7-8**

АЛГОРИТМ ФОРДА-ФАЛКЕРСОНА

1 ЦЕЛЬ РАБОТЫ

Целью работы является изучение матричных способов *:* научиться находить максимальный поток и строить минимальный разрез в сети с использованием алгоритма Форда- Фалкерсона.

2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

*Теорема Форда-Фалкерсона 1 (о максимальном потоке и минимальном разрезе).*

В любой сети существует максимальный поток. Величина максимального потока равна пропускной способности минимального разреза.

*Теорема Форда-Фалкерсона 2.*

Поток, вычисленный с помощью алгоритма Форда-Фалкерсона имеет максимальную величину, а разрез , где -множество вершин, помеченных при последнем помечивании, имеет минимальную пропускную способность.

*Перечень умений*

|  |  |  |
| --- | --- | --- |
| **№** | **Умение** | **Алгоритм** |
| 1. | *Нахождение максимального потока и построение минимального разреза в сети с использованием алгоритма Форда-Фалкерсона* | *Описание алгоритма*  В данной задаче основным параметром на дугах сети является – *пропускная способность*. Пропускная способность показывает, сколько единиц потока может быть передано по дугам сети. Таким образом, *потоком в сети**D* = [*N*, *M*] называется неотрицательная вещественная функция, удовлетворяющая условиям:  1. *ограниченности*: поток по любой дуге сети не превосходит пропускной способности этой дуги ;  2. *сохранения*: суммарный поток, заходящий в любую вершину сети (кроме истока и стока), равен суммарному потоку, выходящему из этой вершины.  Дуга сети называется *насыщенной*, если поток по этой дуге равен пропускной способности этой дуги, т. е. .  *Разрезом сети*называется множество дуг, удаление которых из сети приводит к тому, что исток и сток оказываются несвязанными.  *Пропускной способностью разреза*называется число, равное сумме пропускных способностей дуг этого разреза. Разрез называется *минимальным*, если имеет наименьшую пропускную способность.  Отыскание минимального разреза – одна из основных задач анализа транспортных сетей. В силу конечности графа минимальный разрез может быть найден перебором всех разрезов, но этот путь, конечно, неприемлем для достаточно больших графов.  Минимальный разрез можно отыскать при помощи теоремы Форда – Фалкерсона: в любой транспортной сети величина любого *максимального потока*равна пропускной способности любого *минимального разреза*.  Для нахождения максимального потока в сети разработан *алгоритм Форда – Фалкерсона*. Перед началом выполнения алгоритма все вершины сети нумеруются произвольным образом, кроме источника и стока (источник получает минимальный номер 1, сток – максимальный , где – число узлов).  Алгоритм состоит из следующих основных шагов:  1. Определить начальный поток в сети, сложив потоки по дугам, выходящим из источника.  2. Вершинам сети присвоить целочисленные метки, а дугам – знаки «+» и «–» по следующим правилам:  а) вершине-истоку присвоить метку ;  б) находим *непомеченную*вершину , *смежную помеченной*вершине . Если поток по соединяющей вершины дуге меньше пропускной способности этой дуги, то происходит *помечивание*, иначе переходим к рассмотрению следующей вершины. Если вершина является *образом*помеченной вершины , то происходит *прямое помечивание* **(***дуга в прямом направлении допустима***)**: вершина получает метку, равную номеру вершины , соединяющая вершины дуга получает метку «+», переходим к рассмотрению следующей вершины. Если вершина не имеет ни одного помеченного прообраза, поток по дуге в прямом направлении больше 0, то происходит *обратное помечивание* **(***дуга допустима в обратном направлении***)**: вершина получает метку, равную номеру вершины (являющейся в данном случае ее образом), соединяющая вершины дуга получает метку «–», происходит переход к рассмотрению следующей вершины. Процесс помечивания продолжается до тех пор, пока все удовлетворяющие этим условиям вершины не получат метку.  3. Если в результате процедуры помечивания вершина-сток метки не получила, то текущий поток – максимальный, переход к шагу 5. В противном случае перейти к пункту 4.  4. Рассмотреть последовательность вершин *L*, метка каждой из которых равна номеру следующей за ней вершины, и множество дуг *МL*, соединяющих соседние вершины из *L*.  *Построение нового потока по схеме:*  а) Если дуга принадлежит множеству *МL* (смотри выше) и имеет знак «+», то *новый поток по этой дуге* **=** *старый поток по этой дуге* **+** Δ (схему нахождения смотри далее).  б) Если дуга принадлежит множеству *МL* и имеет знак «–», то *новый поток по этой дуге* **=** *старый поток по этой дуге* **–** Δ.  в) Если дуга не принадлежит множеству *МL*, то поток по дуге оставляем без изменения.  Схема нахождения Δ:  I. **,** *где для нахождения*рассматриваются все дуги, принадлежащие множеству *МL* и имеющие знак «+», и для каждой такой дуги вычисляется разность между пропускной способностью дуги и потоком по этой дуге (). Затем из этих значений разностей выбирается минимальное значение и присваивается .  II. *Для нахождения*рассматриваются все дуги, принадлежащие множеству *МL* и имеющие знак «–». Затем из этих дуг выбирается дуга с минимальным потоком (), и значение потока по этой дуге присваивается .  Перейти к шагу 2.  5. Определяем максимальный поток, складывая начальный поток и все полученные изменения потока.  В оптимальном решении, т. е. когда найден максимальный поток, минимальный разрез образуется насыщенными дугами. |

*Тренинг умений*

*Пример*выполнения упражнения тренинга*на умение № 1.*

Постановка задачи поиска максимального потока: найти максимальный поток из в для транспортной сети (рисунок) с помощью алгоритма Форда – Фалкерсона:



*Решение:*

|  |  |  |
| --- | --- | --- |
| *№* | *Алгоритм* | *Конкретные действия* |
| 1. | *1-я итерация* | 1. .  2.1 (Второй шаг, первая итерация – подобное обозначение идет далее для всех шагов алгоритма).  Производим помечивание вершин и дуг, результат показан на рисунке. Вершина 6 получила метку . |
| 2. | *2-я итерация* | 3.1.  .  4.1. ;  ;  .  2.2. Заново осуществляется помечивание. Вершина 6 снова получает метку  (смотри рисунок). |
| 3. | *3-я итерация* | 3.2. .  4.2. ;  ;  .  2.3. Осуществляется помечивание. При этом из вершины 3 прямых допустимых дуг не выходит, однако дуга 2–3 является допустимой в обратном направлении, и вершина 2 получает метку . Вершина 6 получает метку  (смотри рисунок). |
| 4. | *4-я итерация* | 3.3. .  4.3. ;  ;  ;  ;  .  2.4. Осуществляется помечивание. При этом из вершины 3 допустимые дуги не выходят. Вершина 6 не получает метку (смотри рисунок). Переходим к шагу 5. |
| 5. |  | 5. .  Минимальный разрез образуют насыщенные дуги 3–6 и 5–6. Пропускная способность минимального разреза . Условия теоремы Форда – Фалкерсона выполняются   задача решена правильно.  Алгоритм Форда – Фалкерсона используется при решении многих практических задач. Одна из них – *задача об источниках и потребителях*. |

3 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

3.1 Получить задание у преподавателя по определению максимального потока.

3.2 Составить алгоритм программы, реализующей нахождение максимального потока в сети методом Форда-Фалкерсона.

3.3 Создать программу, реализующую реализующей нахождение максимального потока в сети методом Форда-Фалкерсона..

*Задание.*

Дана сеть:

****

Определить максимальный поток в сети при начальных значениях дуговых потоков: , , , , , , .

Варианты значений пропускных способностей дуг для задания:



4 СОДЕРЖАНИЕ ОТЧЕТА ПО РАБОТЕ

4.1 Исходное задание и цель работы.

4.2 Блок-схема программы по п.3.2.

4.3 Распечатка текста программы по п.3.3.

4.4 Контрольный пример и результаты машинного расчета.

4.5 Выводы по работе.

5 КОНТРОЛЬНЫЕ ВОПРОСЫ

* 1. Дайте определение сети и потока в сети.
  2. Дайте определение разреза.
  3. Сформулируйте задачу о нахождении максимального потока.
  4. Сформулируйте теорему Форда-Фалкерсона.
  5. Опишите общий алгоритм поиска максимального потока.
  6. Опишите метод нахождения увеличивающего пути.

**ЛАБОРАТОРНАЯ РАБОТА № 9**

ЦИКЛЫ В ГРАФАХ

1 ЦЕЛЬ РАБОТЫ

Целью работы является изучение алгоритмов поиска эйлеровых и гамильтоновых циклов в графе.

2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Эйлеровы циклы

Эйлеров цикл — это такой цикл, который проходит ровно один раз по каждому ребру.

Теорема. Связный неориентированный граф G содержит эйлеров цикл тогда и только тогда, когда число вершин нечет­ной степени равно нулю.

Не все графы имеют эйлеровы циклы, но если эйлеров цикл существует, то это означает, что, следуя вдоль этого цикла, можно нарисовать граф на бумаге, не отрывая от нее каранда­ша. Дан граф G, удовлетворяющий условию теоремы. Требует­ся найти эйлеров цикл. Используется просмотр графа методом поиска в глубину, при этом ребра удаляются. Порядок просмот­ра (номера вершин) запоминается. При обнаружении вершины, из которой не выходят ребра, мы их удалили, ее номер записы­вается в стек, и просмотр продолжается от предыдущей верши­ны. Обнаружение вершин с нулевым числом ребер говорит о том, что найден цикл. Его можно удалить, четность вершин (количество выходящих ребер) при этом не изменится. Процесс продолжается до тех пор, пока есть ребра. В стеке после этого будут записаны номера вершин графа в порядке, соответствующем эйлерову циклу.

*Procedure Search(v:Integer);{\*Глобальные*

*переменные: А - матрица смежности, CV - стек;*

*yk - указатель стека. \*}*

*Var j:Integer;*

*Begin*

*For j:=1 To N Do*

*If A[v,j]<>0 Then Begin*

*A[v,j] :=0;A[j,v] :=0;*

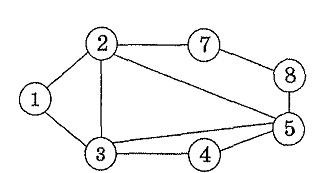
*Search (j)*

*End;*

*Inc (yk);Cv[yk]:=v;*

*End;*

Пример графа и содержимое сте­ка Cv после работы процедуры Search

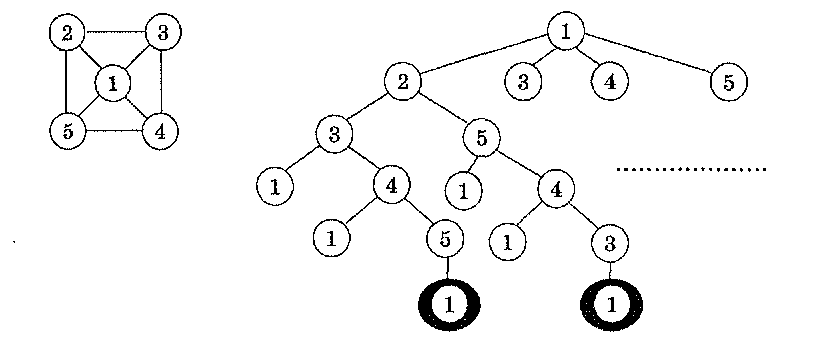
**

Cv: 1356725432.

**Гамильтоновы циклы**

Граф называется гамильтоновым, если в нем имеется цикл, содержащий каждую вершину этого графа. Сам цикл также на­зывается гамильтоновым. Не все связные графы гамильтоновы. На рисунке дан пример графа, не имеющего гамильтонова цикла.

Дан связный неориентированный граф G. Требуется найти все гамильтоновы циклы графа, если они есть. Метод реше­ния — перебор с возвратом (backtracking). Начинаем поиск ре­шения, например, с первой вершины графа. Предположим, что уже найдены первые k компонент решения. Рассматриваем ребра, выходящие из последней вершины. Если есть такие, что идут в ранее не просмотренные вершины, то включаем эту вер­шину в решение и помечаем ее как просмотренную. Получена компонента решения. Если такой вершины нет, то воз­вращаемся к предыдущей вершине и пытаемся найти ребро из нее, выходящее в другую вершину. Решение получено при про­смотре всех вершин графа и возможности достичь из последней первой вершины. Решение (цикл) выводится, и продолжается процесс нахождения следующих циклов. Пример графа и часть дерева перебора вариантов, показывающего механизм работы данного метода, приведен на рисунке.



*Procedure Gm(к:Integer); {\* к - номер итерации.*

*Глобальные переменные: А - матрица смежности; St -массив для хранения порядка просмотра вершин*

*графа; Nnew - массив признаков: вершина*

*просмотрена или нет. \*}*

*Var j,v:Integer;*

*Begin*

*v:=St [k-1]; {\*Номер последней вершины. \*}*

*For j:=1 To N Do*

*If (A[v,j]<>0) Then {\*Есть ребро между*

*вершинами с номерами v и j . \*}*

*If (k=N+l) And (j=l) Then <вывод цикла>*

*Else*

*If Nnew[j] Then Begin {^Вершина*

*не просмотрена. \*}*

*St[k]:=j;*

*Nnew[j]:=False;*

*Gm(k+1) ;*

*Nnew[j]:=True;*

*End;*

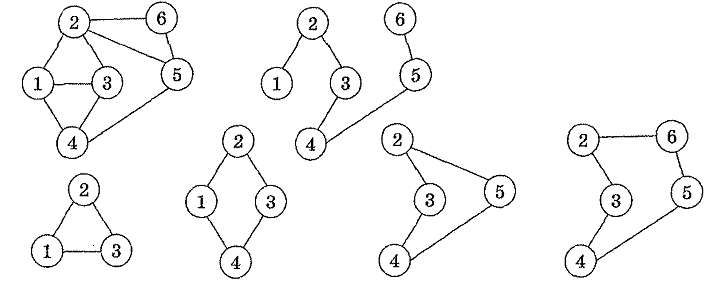
*End;*

Фрагмент основной программы. St[l] :=l;Nnew[l]: =False; Gm(2)

**Фундаментальное множество циклов**

Каркас *(V,T)* связного неориентированного графа *G=<V,E>* содержит *N-1* ребро, где *N* — количество вершин *G.* Каждое ребро, не принадлежащее *Т,* т. е. любое ребро из *Е-Т,* порожда­ет в точности один цикл при добавлении его к *Т.* Такой цикл является элементом фундаментального множества циклов гра­фа *G* относительно каркаса *Т.* Поскольку каркас состоит из *N—1* ребра, в фундаментальном множестве циклов графа *G* от­носительно любого каркаса имеется М-ЛЧ^циклов, где *М —* количество ребер в *G.*

Пример графа, его каркаса и множества фундаментальных циклов приведен на рисунке



Поиск в глубину является естественным подходом, исполь­зуемым для нахождения фундаментальных циклов. Строится каркас, а каждое обратное ребро порождает цикл относительно этого каркаса. Для вывода циклов необходимо хранить поря­док обхода графа при поиске в глубину (номера вершин) — мас­сив *St,* а для определения обратных ребер вершины следует «метить» (массив *Gnum)* в той очередности, в которой они про­сматриваются. Если для ребра *<v,j>* оказывается, что значение метки вершины с номером *j* меньше, чем значение метки вер­шины с номером *v,* то ребро обратное и найден цикл.

Начальная инициализация переменных

*пит:=0;ук:=0;*

*For j:=2 Го N Do Gnum[j ] :=0;*

Основная логика.

*Procedure Circl(v:integer);{\*Глобальные*

*переменные: А - матрица смежности графа; St -массив для хранения номеров вершин графа в том порядке, в котором они используются при построении каркаса; ук - указатель записи в массив St; Gnum -для каждой вершины в соответствующем элементе массива фиксируется номер шага (num.) , на котором она просматривается при поиске в глубину. \*}*

*Var j:Integer;*

*Begin*

*Inc(yk);St[yk]:=v;*

*Inc (num);Gnum[v]:=num;*

*For j :=1 To N Do*

*If A[v,j]<>0 Then*

*If Gnum[j]=0 Then Circl[j] {\*Вершина j*

*не просмотрена.\*}*

*Else*

*If (j<>St[yk-l]) And (Gnum[j]<Gnum[v]) Then*

*<вывод цикла из St>(\*j не предыдущая*

*вершина при просмотре, и она была*

*просмотрена ранее. \*};*

*Dec (yk);*

*End;*

Название «фундаментальный» связано с тем, что каждый цикл графа может быть получен из циклов этого множества. Для произвольных множеств *A* и *В* определим операцию симметриче­ской разности *.* Известно, что произвольный цикл графа G можно однозначно представить как симметриче­скую разность некоторого числа фундаментальных циклов. Одна­ко не при всех операциях симметрической разности получаются циклы (вырожденный случай).

Исследовательская работа — разработать программу нахождения всех циклов графа.

3 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

3.1 Получить задание у преподавателя по определению эйлерова и гамильтонова цикла.

3.2 Составить алгоритм программы, реализующей нахождение эйлерова и гамильтонова цикла в графе.

3.3 Создать программу, реализующую реализующей нахождение эйлерова и гамильтонова цикла в графе.

4 СОДЕРЖАНИЕ ОТЧЕТА ПО РАБОТЕ

4.1 Исходное задание и цель работы.

4.2 Блок-схема программы по п.3.2.

4.3 Распечатка текста программы по п.3.3.

4.4 Контрольный пример и результаты машинного расчета.

4.5 Выводы по работе.

5 КОНТРОЛЬНЫЕ ВОПРОСЫ

5.1. Дайте определение пути и цикла в графе.

5.2 Дайте определение эйлерова пути и эйлерова цикла.

5.3 Дайте определение гамильтонова пути и гамильтонова цикла.

5.4. Сформулируйте теорему Эйлера.

5.5 Опишите алгоритмы нахождения эйлерова и гамильтонова циклов.

**Тема 2. Алгоритмы комбинаторного перебора (18 часов).**

**Цель**: Практическое изучение базовых алгоритмов комбинаторной генерации и комбинаторного перебора, а также способов их реализации на компьютере.

**ЛАБОРАТОРНАЯ РАБОТА № 10-11**

ГЕНЕРАЦИЯ ПЕРЕСТАНОВОК

1 ЦЕЛЬ РАБОТЫ

Целью работы является изучение различных алгоритмов генерации всех перестановок.

2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

**Перестановки**

*Первая задача.* Научимся вычислять число перестановок для заданного значения N. Известно, что число перестановок равно факториалу числа



Рекурсивное определение:



Функция имеет ограниченную область применения.

*Function Fac (k:LongInt): LongInt;*

*Var r,i: LongLnt;*

*Begin*

*r:=1;*

*For i:=l To к Do r:=r\*i;*

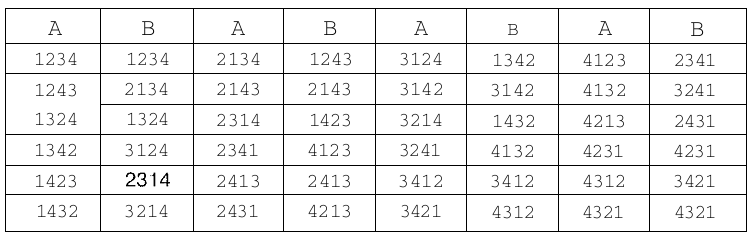
*Faс:=r;*

*End;*

Функция работоспособна при N, меньшем или равном 12.Для больших значений N требуется использовать «длинную» арифметику: умножение «длинного» числа на короткое; вывод «длинного» числа

*Вторая и третья задачи*. Необходимо перечислить или сгенерировать все перестановки для заданного значения N, что требует введения отношения порядка на множестве перестановок. Только в этом случае задача имеет смысл. Предполагаем, что перестановка хранится в массиве Р длины N. Лексикографический порядок на множестве всех перестановок определяется следующим образом P1<P2 тогда и только тогда, когда существует такое t>=1, что P1[t]<P2[t] и P1[i]<P2[i] для всех i<t. При решении данной задачи решается и задача получения следующей в лексикографическом порядке перестановки. Аналогично определяется антилексикографический порядок.

*Пример* (N=4). Столбцы, обозначенные буквой А, означают генерацию перестановок в лексикографическом порядке, буквой В — антилексикографическом.



*Пример*. Перестановка 11, 8, 5, 1, 7, 4, 10, 9, 6, 3, 2. Какая следующая перестановка идет в лексикографическом порядке? Находим «скачок» — 4 меньше 10. Затем «в хвосте» перестановки находим первый элемент с конца перестановки, больший значения 4, на рисунке это значение 6. Меняем местами элементы 4 и 6 и «хвост» перестановки ухудшаем максимально, т. е. расставляем элементы в порядке возрастания. Процедура получения следующей в лексикографическом порядке перестановки имеет вид:

*Procedure GetNext;*

*Var i,j:Integer;*

*Begin {\*Для перестановки N, N-1,...,1 процедура*

*не работает, проверить. \*}*

*i : =N;*

*While (i>l) And (P [i] <P [i-1] ) DoDec(i);*

*{\*Находим "скачок".\*}*

*j:=N;*

*While P[j]<P[i-1] Do Dec(j); {\*Находим первый*

*элемент, больший значения P[i-1].\*}*

*Swap (P[i-1], P[j]) ;*

*For j:=0 To (N-i+1) Div 2 - 1 Do Swap (P[i+j] ,*

*P[N-j]) ; {\*Переставляем элементы "хвоста"*

*перестановки в порядке возрастания.\*}*

*End;*

*Procedure Swap (Var a,b:Integer);*

*Var t:Integer;*

*Begin*

*t:=a; a:=b; b:=t;*

*End;*

Фрагмент общей схемы генерации:

*Init;{\*Инициализация: в массиве Р - первая*

*перестановка, в массиве Last - последняя.\*}*

*Print;{\*Вывод перестановки.\*}*

*While Not(Eq(P,Last)) Do Begin {\*Функция Eq*

*определяет равенство текущей перестановки*

*и последней; если не равны, то генерируется*

*следующая \*}*

*GetNext;*

*Print;*

*End;*

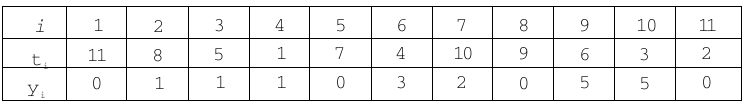
Изменим задачу. Необходимо перечислить все перестановки чисел от 1 до N так, чтобы каждая следующая перестановка получалась из предыдущей с помощью транспозиции двух соседних элементов.

Например:



Возьмем перестановку p1,…,pN  и сопоставим с ней следующую последовательность целых неотрицательных чисел у1,…,yN. Для любого i от 1 до N найдем номер позиции s, в которой стоит значение i в перестановке, т. е. такое s, что ps=i, и подсчитаем количество чисел, меньших i среди p1,…,pN . Это количество и будет значением yi

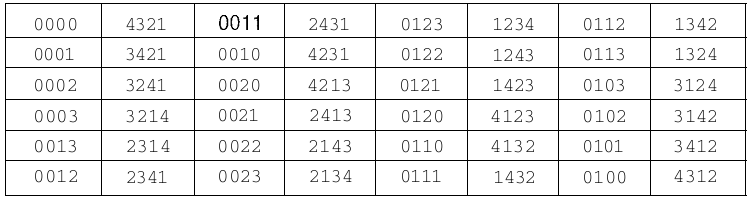
*Пример*



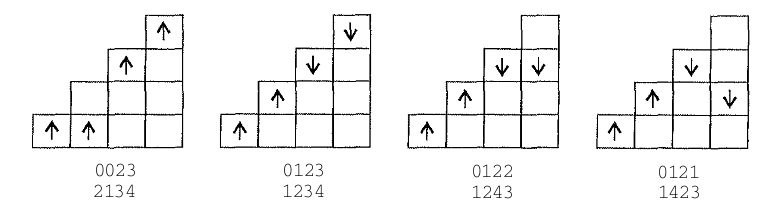
Очевидно, что 0=<yi<i. Всего таких последовательностей N!, т. е. множества перестановок и последовательностей чисел совпадают по мощности. Построение последовательности чисел из перестановки очевидно, как и то, что разным перестановкам соответствуют разные последовательности чисел. Эта таблица называется таблицей инверсий. М. Холл установил, что таблица инверсий единственным образом определяет соответствующую перестановку — обратное построение.

*Пример*. Последовательность 0111. Четверка записана на втором месте, ибо только одна цифра в перестановке «стоит» левее ее; имеем \*4\*\*. Тройка с учетом занятого места находится на третьем месте — \*43\*. Двойке с учетом занятых мест принадлежит четвертое место, т. е. получаем перестановку 1432.

Таблица последовательностей чисел и соответствующих перестановок для N=4.



Зрительный образ предлагаемой схемы генерации. Пусть есть доска в форме лестницы. Высота i вертикали равна i. В первоначальном состоянии шашки стоят на первой горизонтали. Стрелки указывают направление движения шашек (вверх). За один ход можно передвинуть одну шашку в направлении стрелки. Если шашку передвинуть нельзя, то осуществляется переход к следующей слева шашке. После того как найдена шашка, которую можно передвинуть помимо передвижения шашки, осуществляется смена направления движения всех шашек справа от найденной. Ниже на рисунке приведена смена состояний шашек на доске при N=4, начиная с последовательности чисел 0023. Под последовательностями чисел записаны соответствующие им перестановки.



Оказывается, что изменение на единицу одного из элементов последовательности соответствует транспозиции двух соседних элементов перестановки. Увеличение y[i] на единицу соответствует транспозиции числа i с правым соседом, уменьшение — с левым соседом. Итак, помимо генерации последовательностей необходимо уметь находить число i в перестановке и менять его местами с одним из «соседей», но это уже чисто техническая

задача.

Опишем используемые данные.

*ConstNmax=12;*

*Var P:Array[1. .Nmax] Of 0 . .Nmax; {\*Хранение*

*перестановки.\*}*

*Y:Array[1..Nmax] Of 0..Nmax-1; {\*Хранение*

*последовательности чисел.\*}*

*D:Array[1..Nmax] Of-1..1; {\*Направление*

*движения "шашек". \*}*

Процедура формирования начальной перестановки и начальных значений в массивах Y и D.

*Procedure First;*

*Var i:Integer;*

*Begin*

*For i:=1 To N Do Begin*

*P[i]:=N-i+1; Y[i]:=0; D[i]:=1;*

*End;*

*End;*

Поиск номера «шашки», которую можно передвинуть. При D[i]=l и Y[i]=i-1 или D[i]=-l и Y[i]=0 шашку с номером i нельзя передвинуть.

*Function Ok:Integer; {\*Если значение функции равно*

*1, то нет ни одной шашки, которую можно*

*передвинуть.\*}*

*Var i:Integer;*

*Begin*

*i:=N;*

*While (i>1) And (((D[i]=1) And (Y[i]=i-1)) Or*

*((D[i]=-1) And (Y[i]=0))) DoDec(i);*

*Ok: =i ;*

*End;*

Основная логика генерации перестановок имеет вид:

*Begin*

*First;*

*рр:=True;*

*While pp Do Begin*

*Solve (pp) ;*

*If pp Then Print;*

*End;*

*End;*

Уточним процедуру генерации следующей перестановки.

*Procedure Solve(Var q:Boolean);*

*Var i, j, dj:Integer;*

*Begin*

*i:=Ok; q:=(i>l); {\*Haходим номер шашки, которую*

*можно передвинуть. \*}*

*If i>1 Then Begin*

*Y[i]:=Y[i]+D[i]; {\*Передвигаем шашку - изменяем*

*последовательность чисел.\*}*

*For j:=i+1 To N Do D[j]:=-D[j]; {\*Изменяем*

*направление движения шашек, находящихся*

*справа от передвинутой.\*}*

*j:=Who\_i(i); {\*Находим позицию в перестановке,*

*в которой записано число i.\*}*

*dj:=j+D[i]; {\*Определяем соседний элемент*

*перестановки для выполнения транспозиции.\*}*

*Swap(P[j], P[dj]); {\*Транспозиция.\*}*

*End;*

*End;*

Остался последний фрагмент логики, требующий уточнения, — поиск в перестановке позиции, в которой находится элемент i. Он достаточно прост.

*Function Who\_i (i : Integer): Integer;*

*Var j: Integer;*

*Begin*

*j:=N;*

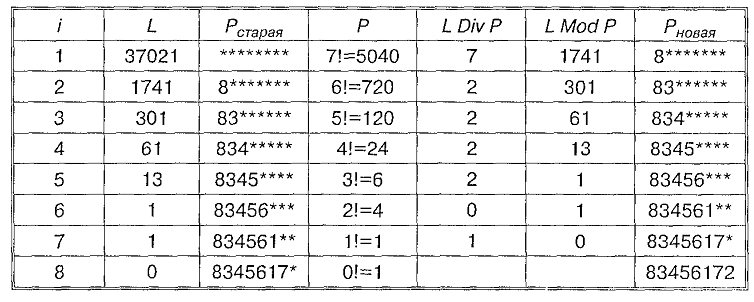
*While (j>0) And (P[j]<>i) Do Dec(j);*

*Who\_i :=j;*

*End;*

*Четвертая задача*. Как по номеру определить перестановку относительно того порядка, разумеется, который введен на множестве перестановок? Остановимся на лексикографическом порядке. Рассмотрим идею решения на примере. Пусть N равно 8 и дан номер L, равный 37021. Найдем соответствующую перестановку. Пусть на первом месте записана единица. Таких перестановок 7!, или 5040 (1\*\*\*\*\*\*\*). При 2 тоже 5040 (2\*\*\*\*\*\*\*). Итак, 37021 Div 5040=7. Следовательно, первая цифра в перестановке 8. Новое значение L (37021 Mod 5040=1741) 1741. Продолжим рассуждения. Оформим, как обычно, их в виде таблицы. Обратим внимание на третью строку, в которой на третье

место записывается цифра 4. То, что записываются не цифры 1 и 2, очевидно: их требуется пропустить. Цифра три «занята», поэтому записываем 4. Точно так же в строках 4, 5 и 7.



О программной реализации. Пусть N=<12, таким образом,

значение L не превосходит максимального значения типа LongInt. Определим следующие константы.

*Const Nsmall=12;*

*ResSm: Array[0..Nsmall] Of LongInt*

*=(1,1,2,6,24,120, 720,5040,40320,362880,3628800,3991*

*6800,479001600) ; {\*3начения N! для N от 0 до 12.\*}*

И процедура.

*Procedure GetPByNum(L:LongInt);*

*Var Ws: Set Of Byte;*

*i,j,sc: Integer;*

*Begin*

*Ws:=[]; {\*Множество для фиксации цифр,*

*задействованных в перестановке.\*}*

*For i:=1 То N Do Begin*

*Sc:=L Div ResSm[N-i]; L:=L Mod ResSm[N-i]; j:=1;*

*While (sc<>0) Or (j In Ws) Do Begin*

*If Not (j In Ws) Then Dec(sc);*

*Inc (j) ;*

*End;*

*Ws:=Ws+[j]; {\*Нашли очередную цифру. \*}*

*P[i]:=j;*

*End;*

*End;*

*Пятая задача*. По перестановке получить ее номер, не выполняя генерацию множества перестановок.

Начнем с примера. Пусть N равно 8 и дана перестановка 53871462.

Схема:

7!\*<количество цифр в перестановке на 1-м месте, идущих до цифры 5, с учетом занятых цифр — ответ 4>

6!\*< количество цифр в перестановке на 2-м месте, идущих до цифры 3, с учетом занятых цифр — ответ 2>

5!\*< количество цифр в перестановке на 3-м месте, идущих до цифры 8, с учетом занятых цифр — ответ 5>

4!\*< количество цифр в перестановке на 4-м месте, идущих до цифры 7, с учетом занятых цифр — ответ 4>

3!\*< количество цифр в перестановке на 5-м месте, идущих до цифры 1, с учетом занятых цифр — ответ 0>

2!\*< количество цифр в перестановке на 6-м месте, идущих до цифры 4, с учетом занятых цифр — ответ 1>

1!\*< количество цифр в перестановке на 7-м месте, идущих до цифры 6, с учетом занятых цифр — ответ 1>

Итак,

7!\*4+6!\*2+5!\*5+4!\*4+3!\*0+2!\*1+1!\*1=4\*5040+2\*720+5\*120+4\*24+0\*6+1\*2+1\*1=22305.

*Function GetNumByP: LongInt;*

*Var ws:Set Of Byte;*

*i,j,sq:Integer; sc:LongInt;*

*Begin*

*ws:=[]; {\*Множество цифр перестановки.\*} sc:=1;*

*{\*Номер перестановки.\*}*

*For i:=1 To N Do Begin*

*j:=1; sq:=0; {\*Определяем количество цифр.\*}*

*While j<P[i] Do Begin*

*If Not (j In ws) Then Inc (sq);*

*Inc (j);*

*End;*

*ws:=ws+[P[i]]; {\*Цифра P[i] задействована.\*}*

*sc:=sc+ResSm[N-i]\*sq; {\*Умножаем число*

*перестановок на значение текущей цифры.\*}*

*End;*

*GetNumByP:=sс;*

*End;*

3 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

3.1. Составить алгоритмы программ, реализующих генерацию перестановок рассмотренными способами.

3.2. Создать программы, реализующие генерацию перестановок рассмотренными способами.

4 СОДЕРЖАНИЕ ОТЧЕТА ПО РАБОТЕ

4.1 Исходное задание и цель работы.

4.2 Блок-схема программы по п.3.1.

4.3 Распечатка текста программы по п.3.2.

4.4 Контрольный пример и результаты машинного расчета.

4.5 Выводы по работе.

5 КОНТРОЛЬНЫЕ ВОПРОСЫ

5.1 Чему равно число перестановок множества из N элементов?

5.2. Определите лексикографический порядок на множестве перестановок.

5.3. Как задать перестановку при помощи инверсий?

5.4. Опишите известные Вам алгоритмы генерации всех перестановок.

5.5. Как получить номер заданной перестановки?

5.6. Как получить перестановку по ее номеру?

**ЛАБОРАТОРНАЯ РАБОТА № 12-13**

ГЕНЕРАЦИЯ РАЗМЕЩЕНИЙ

1 ЦЕЛЬ РАБОТЫ

Целью работы является изучение различных алгоритмов генерации всех размещений по M различным местам М из N предметов.

2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

*Первая задача.* Подсчитать количество размещений для небольших значений N и М можно с помощью следующих простых функций.

*Function Plac(n,m: LongInt): LongInt;*

*Var i,z: LongInt;*

*Begin*

*z: =1 ;*

*For i:=0 To m-1 Do z :=z\* (n-i);*

*Plac:=z;*

*End;*

Рекурсивный вариант реализации.

*Function Plac (n,m:LongInt): LongInt;*

*Begin*

*If m=0 Then Plac:=1*

*Else Plac:=(n-m+1)\*Plac (n,m-1);*

*End;*

Функции дают правильный результат при N=<12. Для больших значений N необходимо использовать «длинную» арифметику.

*Вторая задача*. Требуется сгенерировать все размещения для заданных значений N и М в лексикографическом порядке. Пример. N=4, М=3. В таблице приведены размещения в лексикографическом порядке.



Текст решения имеет вид:

*Program GPlac;*

*Const n=4; т=3; {\*Значения пит взяты в качестве*

*примера.\*}*

*Var A:Array[1..m] Of Integer; {\*Массив для*

*хранения элементов размещения.\*}*

*S:Set Of Byte; {\*Для хранения использованных*

*в размещении цифр.\*}*

*Procedure Solve (t:Integer); {\*Параметр t*

*определяет номер позиции в размещении.\*}*

*Var i:Byte;*

*Begin*

*For i:=1 To n Do {\*Перебираем цифры и находим*

*1-ю неиспользованную.\*}*

*If Not (i In S) Then Begin*

*42 2. Комбинаторные алгоритмы*

*S:=S+[i]; {\*Запоминаем её в множестве занятых*

*цифр.\*}*

*A[t] :=i; {\*Записываем её в размещение.\*}*

*If t<m Then Solve (t+1) Else Print; {\*Если*

*размещение не получено, то идем*

*к следующей позиции, иначе*

*выводим очередное размещение.\*}*

*S: =S- [i]; {\*Возвращаем цифру в число*

*неиспользованных.\*}*

*End;*

*End;*

Фрагмент основной программы.

*Begin S: = []; Solve (1) ; End.*

Для генерации размещений можно воспользоваться процедурой генерации перестановок из предыдущей работы. Требуется только уметь «вырезать» из очередной перестановки первые М позиций и исключать при этом повторяющиеся последовательности.

*Третья задача*. По заданному размещению найти следующее за ним в лексикографическом порядке.

Свободными элементами размещения назовем те элементы из множества от 1 до N, которых нет в текущем размещении (последовательности из М элементов). Пример. N=5, М=3 и размещение 1 3 4. Свободными элементами являются 2 и 5. Первый вариант решения заключается в том, чтобы дописать в размещение свободные элементы в убывающем порядке (1 3 4 5 2), сгенерировать следующую перестановку (1 3 5 2 4) и отсечь первые М элементов (1 3 5). Получаем следующее размещение.

Реализация этой же идеи, но без обращения к генерации следующей перестановки приводится ниже по тексту.

Её суть:

• Начинаем просмотр размещения с последнего элемента (М) и идем, если необходимо, до первого.

• Находим первый свободный элемент, который больше, чем элемент в рассматриваемой позиции размещения.

• Если такой элемент найден, то заменяем им текущий, а в «хвост» размещения записываем свободные элементы в порядке возрастания и заканчиваем работу.

• Если такого элемента не найдено, то переходим к следующей позиции.

*Const n=4; т=3;*

*Var A:Array[1 . .т] Of Byte;*

*Procedure GetNext;*

*Var i,j,k,q:Integer;*

*Free:Array[1..n] Of Boolean; {\*Массив для*

*хранения признаков занятости элементов*

*в размещении.\*}*

*Function FreeNext(t:Byte):Byte; {\*Функция поиска*

*первого не занятого элемента.\*}*

*Begin*

*While (t<=n) And Not (Free [t]) Do Inc(t);*

*{\*Находим первый свободный элемент.\*}*

*If t>n Then FreeNext:=0 {\*Если такого элемента*

*нет, то значение функции равно нулю.\*}*

*Else FreeNext:=t; {\* Номер первого свободного*

*элемента.\*}*

*End;*

*Begin*

*For i:=1 To n Do Free [i]:=True; {\*Массив*

*свободных элементов.\*}*

*For i:=1 To m Do Free [A[i]]:=False; {\*По*

*размещению исключаем занятые элементы.\*}*

*i:=m; {\*Начинаем с конца размещения.*

*Предполагаем, что анализируемое*

*размещение не является последним.\*}*

*While (i>0) And (FreeNext(A[i])=0) Do Begin*

*{\*Пока не найти позицию в размещении,*

*элемент в которой допускается изменить,*

*выполняем действия из цикла. При нашем*

*предположении такая позиция обязательно*

*существует.\*}*

*Free[A[i]]:=True;{\*Освобождаем элемент,*

*записанный в позиции i.\*}*

*Dec(i); {\*Переходим к следующей позиции.\*}*

*End;*

*Free[A[i]]:=True; {\*Переводим текущий элемент*

*в найденной позиции в свободные.\*}*

*q:=FreeNext(A[i]+1); {\*Находим свободный*

*элемент, больший текущего.\*}*

*Free[q]:=False; {\*Считаем его занятым.\*}*

*A[i]:=q; {\*3аписываем его в размещение.\*}*

*к:=1; {\*Формируем "хвост" размещения.\*}*

*For j:=i+1 To m Do Begin {\*Co следующей позиции*

*до конца размещения.\*}*

*While (k<=n) And Not(Free[k]) Dо {\*Пока не*

*найдем первый свободный элемент.\*}*

*If k>=п Then k:=1 Else Inc(k);*

*A[j]:=k; {\*Записываем найденный элемент*

*в размещение.\*}*

*Free[k]:=False; {\*Считаем его занятым.\*}*

*End;*

*End;*

*Четвертая задача*. При заданных значениях N и М по номеру размещения L определить соответствующее размещение (упорядочены в лексикографическом порядке).

Рассмотрим размещения из 5 по 3 элемента, их 60. В таблице часть размещений, вместе с их номерами, приводится в лексикографическом порядке. Первому размещению соответствует номер 0.



Значение (в общем случае ) равно 12. Количество размещений с фиксированным значением в первой позиции равно 12, а это уже путь к решению. Пусть нам задан номер 32 и массив свободных элементов 1 2 3 4 5 (номера элементов массива считаются с 0). Вычисляя 32 Div 12, получаем 2, следовательно, в первой позиции записана цифра, стоящая на 2-м месте в массиве свободных элементов, а это цифра 3. Оставшееся количество номеров 32 Mod 12 =8, массив свободных элементов 1 2 4 5. Продолжим процесс вычисления - 8 Div 3=2 , следующая цифра 4. Оставшееся количество номеров 8 Mod 3 =2, массив свободных номеров 1 2 5. Продолжим - 2 Div 1 =2, что соответствует цифре 5.

*Const n=5; m=3;*

*Var A:Array[1. .т] Of Integer;*

*L:LongInt; {\*Номер размещения.\*}*

*Procedure GetPByNum(L:LongInt); {\*B процедуре*

*использована функция Plac - вычисления*

*количества размещений.\*}*

*Var i,j,q,t:LongInt;*

*Free:Array[1..n] Of Byte; {\*Массив свободных*

*элементов размещения.\*}*

*Begin*

*For i:=1 To n Do Free [i]:=i ; {\*Начальное*

*формирование.\*}*

*For i:=1 To m Do Begin {\*i - номер позиции*

*в размещении.\*}*

*t:=Plac(n-i,m-i); {\*Количество размещений,*

*приходящихся на один фиксированный*

*элемент в позиции i.\*}*

*q:=L Div t; { \*Вычисляем номер свободного*

*элемента размещения.\*}*

*A[i]:=Free[q+1]; {\*Формируем элемент*

*размещения.\*}*

*For j:=q+1 To n-i Do Free [j] :=Free [j+1];*

*{\*Сжатие, найденный элемент*

*размещения исключаем из свободных.\*}*

*L:=L Mod t; {\*Изменяем значение номера*

*размещения (переход к меньшей*

*размерности).\*}*

*End;*

*Пятая задача.* По размещению определить его номер (размещения упорядочены в лексикографическом порядке).

Поясним идею решения на примере. Дано размещение 345. Количество свободных элементов, меньших 3, равно 2 (12\*2=24). Количество свободных номеров, меньших 4, также 2 — 24+3\*2=30. И наконец, меньших пяти, также 2. Ответ: 12\*2+3\*2+1\*2=32.

*Function GetNumByP:LongInt; {\*Используется*

*процедура Plac вычисления количества*

*размещений.\*}*

*Var L,i,j,num: LongInt;*

*ws:Set Of Byte; {\*Множество элементов*

*размещения.\*}*

*Begin*

*ws: = [] ;*

*L: =0 ;*

*For i:=1 To m Do Begin {\*i - номер позиции*

*размещения.\*}*

*num:=0; {\*Счетчик числа незанятых элементов.\*}*

*For j:=1To A[i]-1 Do*

*If Not (j In ws) Then Inc (num); {\*Если элемента*

*j нет в ws, то увеличиваем значение*

*счетчика числа незанятых элементов,*

*которые встречаются до значения A[i].\*}*

*ws:=ws+[A[i]]; {\*Элемент A[i] задействован*

*в размещении.\*}*

*L:=L+num\*Plac(n-i,m-i); {\*Значение счетчика*

*умножаем на количество размещений,*

*приходящихся на одно значение*

*в позиции с номером i.\*}*

*End;*

*GetNumByP:=L;*

*End;*

3 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

3.1. Составить алгоритмы программ, реализующих генерацию всех размещений по M различным местам М из N предметов.

3.2. Создать программы, реализующие генерацию всех размещений по M различным местам М из N предметов.

4 СОДЕРЖАНИЕ ОТЧЕТА ПО РАБОТЕ

4.1 Исходное задание и цель работы.

4.2 Блок-схема программы по п.3.1.

4.3 Распечатка текста программы по п.3.2.

4.4 Контрольный пример и результаты машинного расчета.

4.5 Выводы по работе.

5 КОНТРОЛЬНЫЕ ВОПРОСЫ

5.1 Чему равно число всех размещений по M различным местам М из N предметов?

5.2. Определите лексикографический порядок на множестве размещений.

5.3. Опишите известные Вам алгоритмы генерации всех размещений по M различным местам М из N предметов.

5.4. Как получить номер заданного размещения?

5.5. Как получить размещение по его номеру?

**ЛАБОРАТОРНАЯ РАБОТА № 14-15**

ГЕНЕРАЦИЯ СОЧЕТАНИЙ

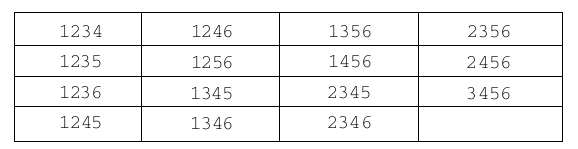
1 ЦЕЛЬ РАБОТЫ

Целью работы является изучение различных алгоритмов генерации всех сочетаний, то есть способов выбора M из N предметов.

2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Между k-элементными подмножествами N-элементного множества и возрастающими последовательностями длины k с элементами из множества целых чисел от 1 до N (сочетаниями) существует взаимно однозначное соответствие. Так, например, подмножеству [2, 6, 1, 3] соответствует последовательность чисел 1, 2, 3, 6. Таким образом, решая задачи подсчета и генерации всех сочетаний, мы решаем аналогичные задачи для k-элементных подмножеств. Лексикографический порядок на множестве последовательностей определяется так же, как и для перестановок.

*Пример *



*Первая задача*. Попробуем подсчитать количество сочетаний для данных значений N и k.

Использование формулы



явно не продуктивно. Факториал представляет собой быстровозрастающую функцию, поэтому при вычислении числителя и знаменателя дроби может возникнуть переполнение, хотя результат — число сочетаний — не превышает, например, значения MaxInt. Воспользуемся для подсчета числа сочетаний формулой



Получаем знаменитый треугольник Паскаля. Просматривается явная схема вычислений. Очевидно, что если в задаче не требуется постоянно использовать значения для различных значений N и k, а требуется только подсчитать одно значение, то хранить двумерный массив в памяти компьютера нет необходимости.

Приведем процедуры вычисления ** для того и другого случая.

Подсчет всех значений **

*Const MaxN=100;*

*Var SmallSc:Array[0..MaxN,0..MaxN] Of LongInt;*

*{\*Нулевой столбец и нулевая строка необходимы,*

*это позволяет обойтись без анализа на выход*

*за пределы массива.\*}*

*Procedure FillSmallSc;*

*Var i,j :Integer;*

*Begin*

*FillChar (SmallSc,SizeOf(SmallSc),0);*

*For i:=0 To N Do SmallSc [i,0]:=1;*

*For i:=1 To N Do*

*For j:=1 To k Do*

*If SmallSc[i-1,j]\*1. 0+SmallSc[i-1,j-1]>*

*MaxLonglnt Then SmallSc[i,j]:=MaxLongInt*

*Else SmallSc[i,j]:=SmallSc[i-1,j]+SmallSc*

*[i-1,j-1]; {\*Умножение на 1.0 переводит*

*целое число в вещественное, поэтому*

*переполнения при сложении не происходит.*

*Стандартный прием, обязательный при "игре"*

*на границах диапазона целых чисел.\*}*

*End;*

Второй вариант реализации процедуры.

*Type SmallSc=Array[0..MaxN] Of LongInt;*

*Function Res(k:Integer): LongInt;*

*Var A,B:SmallSc;*

*i,j:Integer;*

*Begin*

*FillChar (A,SizeOf (A),0) ;*

*FillChar(B,SizeOf(B),0) ;*

*A[0]:=1;A[1]=1;*

*For i:=1 To N Do Begin*

*B[0] :=1; B[1] :=1;*

*For j:=1 To k Do В [j]:=A[j]+A[j-1]; {\*Если число*

*больше максимального значения типа LongInt,*

*то необходимо работать с "длинной"*

*арифметикой. Данная операция должна быть*

*изменена - вызов процедуры сложения двух*

*"длинных" чисел.\*}*

*А:=В;*

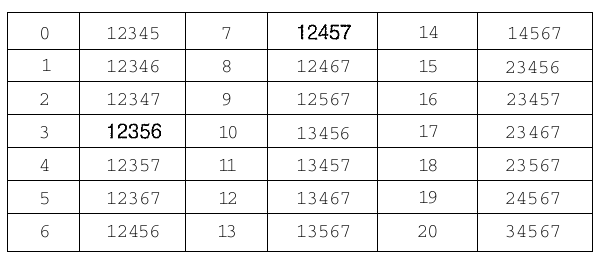
*End;*

*Res:=A[k];*

*End;*

*Вторая задача.* Перейдем к алгоритму генерирования всех сочетаний в лексикографическом порядке.

Пример при N=7 и k=5. Число сочетаний равно 21.



Начальное сочетание равно 1, 2, ..., k, последнее — N-k+1, N-k+2, ..., N. Переход от текущего сочетания к другому осуществляется по следующему алгоритму.

Просматриваем сочетание справа налево и находим первый элемент, который можно увеличить. Увеличиваем этот элемент на единицу, а часть сочетания (от этого элемента до конца) формируем из чисел натурального ряда, следующих за ним.

*Procedure GetNext;{\*Предполагается, что текущее*

*сочетание (хранится в массиве С) не является*

*последним.\*}*

*Var i,j:Integer;*

*Begin*

*i:=k;*

*2. Комбинаторные алгоритмы 49*

*While (C[i]+k-i+1>N) DoDec(i); {\*Находим*

*элемент, который можно увеличить.\*}*

*Inc(С[i]); {\*Увеличиваем на единицу.\*}*

*For j:=i+1 To k Do С[j]:=С[j-1]+1; {\*Изменяем*

*стоящие справа элементы.\*}*

*End;*

*Третья задача.* По номеру L определяем соответствующее сочетание.

Для решения этой задачи необходимо иметь (вычислить) массив SmallSc, т. е. использовать первую схему вычисления числа сочетаний (первая задача) для данных значений N и k. Порядок на множестве сочетаний лексикографический. В таблице, приведенной выше, номера с 0 по 14 имеют сочетания, в которых на первом месте записана единица. Число таких сочетаний — («израсходован» один элемент из N и один элемент из k). Сравниваем число L со значением Если значение L больше , то на первом месте в сочетании записана не единица, а большее число. Вычтем из L значение и продолжим сравнение.

*Procedure GetWhByNum (L:LongInt);*

*Var i,j,sc,ls:Integer;*

*Begin*

*sc:=n;*

*ls:=0; {\*Цифра сочетания.\*}*

*For i:=1 To k Do Begin {\*i - номер элемента*

*в сочетании; k-i - число элементов*

*в сочетании.\*}*

*j:=1; {\*sc-j - число элементов (п), из которых*

*формируется сочетание.\*}*

*While L-SmallSc[sc-j,k-i]>=0 Do Веgiп {\*Для*

*данной позиции в сочетании и числа*

*элементов в сочетании находим тот элемент*

*массива SmallSc, который превышает*

*текущее значение L.\*}*

*Dec (L,SmallSc[sc-j ,k-i]) ;*

*Inc (j); {\*Невыполнение условия цикла While*

*говорит о том, что мы нашли строку*

*таблицы SmallSc, т.е. то количество*

*элементов, из которых формируется*

*очередное сочетание, или тот интервал*

*номеров сочетаний (относительно*

*предыдущей цифры), в котором находится*

*текущее значение номера L.\*}*

*End;*

*С[i]:=ls+j; {\*Предыдущая цифра плюс приращение.\*}*

*Inc(ls,j); {\*Изменяем зна чение текущей цифры*

*сочетания.\*}*

*Dec (sc,j); {\*Изменяем количество элементов,*

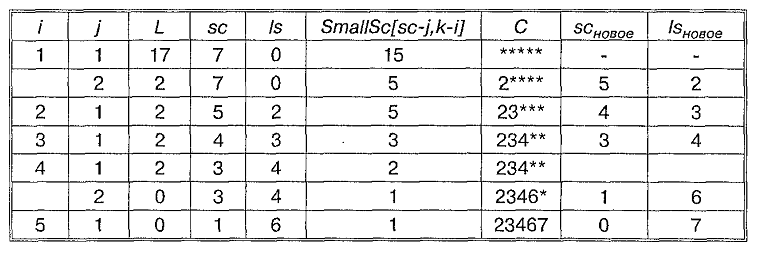
*из которых формируется остаток*

*сочетания.\*}*

*End;*

*End;*

*Пример (N=7, k=5, L=17).*

Для полного понимания логики выполните ручную трассировку при N=9, k=5, L=78. Ответ — 23479

*Четвертая задача*. Требуется по сочетанию получить его номер.

Как и в предыдущей задаче, необходимо иметь массив SmallSc. Эта задача проще предыдущей. Требуется аккуратно просмотреть часть массива SmallSc. Эта часть массива определяется сочетанием, для которого ищется номер. Например, на первом месте в сочетании стоит цифра 3 (N=7, k=4). Ищется сумма — количество сочетаний, в которых на первом месте записана цифра 1, цифра 2 и цифра 3. Аналогично для следующих цифр сочетания. Предполагаем, что С[0] равно 0 — технический прием, позволяющий избежать лишних проверок условий.

*Function GetNumByWh:LongInt;*

*Var sc:LongInt;*

*i, j; Integer;*

*Begin*

*sc:=1;*

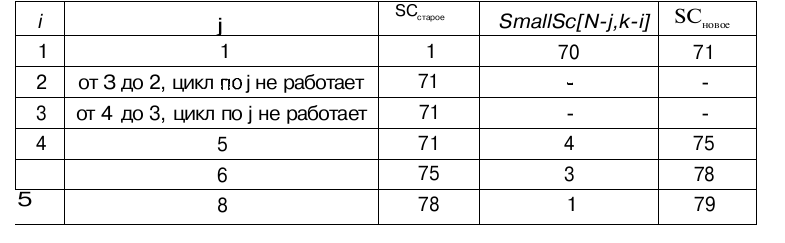
*For i:=1 To к Do*

*For j:=C[i-1]+1 To C[i]-1 Do Inc(sc, SmallSc[N-j,k-i]) ;*

*GetNumByWh:=sc ;*

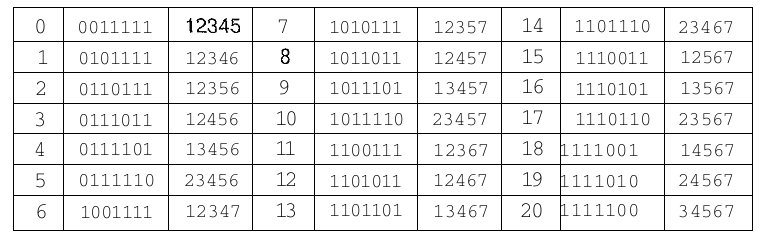
*End;*

Пример (N=9, k=5, C=023479).



Обратите внимание на то, что нумерация сочетаний в задачах 3 и 4 различна. В задаче 3 сочетания нумеруются с 0, а в задаче 4 — с 1.

*Пятая задача*. Выберем для представления сочетания другой вид — последовательность из 0 и 1. Последовательность храним в массиве (С) из N элементов со значениями 0 или 1, при этом С[i]=1 говорит о том, что элемент (N-i+1) есть в сочетании (подмножестве). Пусть N=7, k=5. В таблице представлены последовательности в той очередности, в которой они генерируются, и соответствующие им сочетания.



Принцип генерации последовательностей. Начинаем с конца последовательности. Находим пару C[i]=0 и C[i+1]=1. Если такой пары нет, то получена последняя последовательность. Если есть, то заменяем на месте i нуль на единицу и корректируем последовательность справа от i. Последовательность должна быть минимальна с точки зрения введенного порядка, поэтому вначале записываются 0, а затем 1, при этом количество 1 в последовательности должно быть сохранено.

Опишем основные фрагменты реализации этой логики.

*Const MaxN=...;*

*Type MyArray=Array[1..MaxN] Of 0. .1;*

*Var Curr, Last: MyArray; {\*Текущая и последняя*

*последовательности.\*}*

*i, j,Num, N, k: Integer;*

*Begin ...*

*For i:=1 To N Do Begin Curr[i]:=0;Last[i];=0;End;*

*For i:=1 To k Do Begin*

*Curr[N-i+1]:=1; Last[i]:=1; End; {\*Формируем начальную*

*и последнюю последовательности.\*}*

*While Not Eq(Curr,Last) Do Begin*

*i:=n-1;*

*While (i>0) And Not ((Curr [i]=0) And*

*(Curr [i + 1] =1) ) Do Dec (i); {\*Находим пару 0-1, она*

*есть, так как последовательность*

*не последняя.\*}*

*Num:=0;*

*For j:=i+1 То n Do Inc (Num,Curr [j] );*

*{\*Подсчитываем количество единиц.\*}*

*Curr[i]:=1;*

*For j:=i+1 To n-Num+1 Do Curr[j]:=0; {\*Записываем*

*нули.\*}*

*For j:=n-Num+2 To n Do Curr[j]:=1; {\*3аписываем*

*единицы.\*}*

*End;*

*End;*

3 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

3.1. Составить алгоритмы программ, реализующих генерацию всех сочетаний M из N различных предметов.

3.2. Создать программы, реализующие генерацию всех сочетаний M из N различных предметов.

4 СОДЕРЖАНИЕ ОТЧЕТА ПО РАБОТЕ

4.1 Исходное задание и цель работы.

4.2 Блок-схема программы по п.3.1.

4.3 Распечатка текста программы по п.3.2.

4.4 Контрольный пример и результаты машинного расчета.

4.5 Выводы по работе.

5 КОНТРОЛЬНЫЕ ВОПРОСЫ

5.1 Чему равно число всех сочетаний M из N различных предметов?

5.2. Определите лексикографический порядок на множестве сочетаний.

5.3. Опишите известные Вам алгоритмы генерации всех сочетаний M из N различных предметов.

5.4. Как получить номер заданного сочетания?

5.5. Как получить сочетание по его номеру?

**ЛАБОРАТОРНАЯ РАБОТА № 16**

ГЕНЕРАЦИЯ РАЗБИЕНИЙ

1 ЦЕЛЬ РАБОТЫ

Целью работы является изучение различных алгоритмов генерации всех разбиений числа на слагаемые.

2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

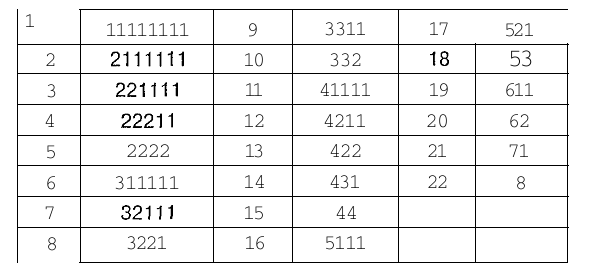
Дано натуральное число N. Его можно записать в виде суммы натуральных слагаемых:



где все a1,…,ak больше нуля. Будем считать суммы эквивалентными, если они отличаются только порядком слагаемых. Класс эквивалентных сумм приведенного вида однозначно представляется последовательностями b1,…,bk упорядоченными по невозрастанию. Каждую такую последовательность b1,…,bk  назовем разбиением числа N на k слагаемых.

Как обычно, решаем четыре задачи: подсчет количества различных разбиений числа; генерация разбиений относительно определенного отношения порядка, введенного на множестве разбиений; получение разбиения по его номеру и задача, обратная третьей.

Первая задача. Приведем пример для N=8, а затем рассуждения, позволяющие написать программный код.

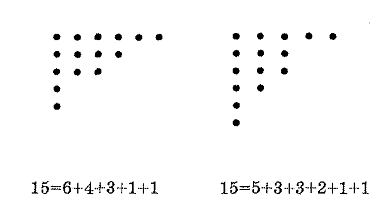


Число разбиений числа N на k слагаемых обозначим через P(N,k), общее число разбиений — P(N). Очевидно, что значение P(N) равно сумме P(N,k) по значению k. Считаем, что Р(0)=Р(1)=1.

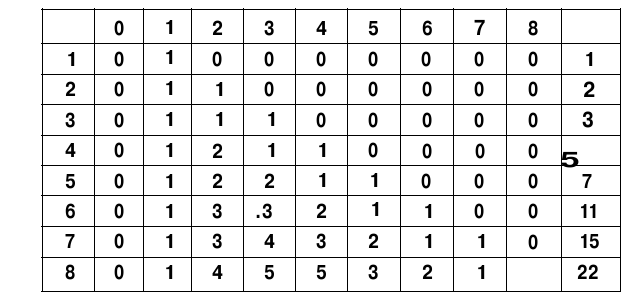
Известен следующий факт.

**Теорема.** Число разбиений числа N на k слагаемых равно числу разбиений числа N с наибольшим слагаемым, равным k.

Теорема доказывается простыми рассуждениями на основе диаграмм Феррерса. Пример такой диаграммы приведен ниже.



Главное, что следует из этого факта, — реальный путь подсчета числа разбиений. Число разбиений P(i,j) равно сумме P(i-j,k) по значению k, где k изменяется от 0 до j. Другими словами, j «выбрасывается» из i и подсчитывается число способов разбиения числа i-j на k слагаемых. Проверьте эту логику с помощью следующей таблицы.



Например, Р(8,3)=Р(5,0)+Р(5,1)+Р(5,2)+Р(5,3).

Определим данные.

*Const MaxN=100;*

*Var N:LongInt;*

*SmallSc:Array[0..MaxN, 0..MaxN] Of LongInt;*

*Procedure FillSmallSc;*

*Var i, j, k:Integer;*

*Begin*

*FillChar(SmallSc,SizeOf(SmallSc),0);*

*SmallSc[0,0]:=1;*

*For i:=1 To N Do*

*For j:=1 To i Do*

*For k:=0 To j Do Inc (SmallSc [i, j ] ,*

*SmallSc[i-j,k]);*

*SmallSc[0,0]:=0;*

*End;*

Таблица значений сформирована, она нам потребуется для решения других задач. Для подсчета же числа разбиений осталось написать простую процедуру.

*Procedure Calc;*

*Var i;Integer;*

*Res:LongInt;*

*Begin*

*Res:=SmallSc[N,1];*

*For i:=2 To N Do Res:=Res+SmallSc [N , i ] ;*

*WriteLn (Res:0:0);*

*End;*

*Вторая задача.* Генерация разбиений.

Во-первых, необходима структура данных для хранения разбиения. По традиции используем массив.

*Var Now:Array[0..MaxN] Of Integer;*

В Now[0] будем хранить длину разбиения, т. е. число задействованных элементов массива Now. Возьмем из первой таблицы одно из разбиений числа 8, например «41111», следующее «4211». В каком случае можно увеличить Now[2]? Видим, что Now[1] должно быть больше Now[2], т. е. в текущем разбиении находится «скачок» - Now[i-1]>Now[i], Now[i] увеличивается на единицу, а все следующие элементы разбиения берутся минимально возможными. Всегда ли возможна данная схема изменения разбиения? Например, разбиение «44», следующее — «5111». Таким образом, или «скачок», или i равно 1. Кроме того, изменяемый элемент не может быть последним, ибо уве-личение без уменьшения невозможно.

*Procedure GetNext;*

*Var i,j,sc:Integer;*

*Begin*

*i:=Now[0]-1; {\*Предпоследний элемент разбиения.\*}*

*sc:=Now[i+1]-1;Now[i+1]:=0; {\*B sc накапливаем*

*сумму, это число представляется затем*

*минимально возможными элементами.\*}*

*While (i>1) And (Now[i]=Now[i-1]) Do Begin*

*sc:=sc+Now[i]; Now[i]:=0; Dec(i); {\*Находим*

*"скачок".\*}*

*End;*

*Inc (Now[i]); {\*Увеличиваем найденный элемент*

*на единицу.\*}*

*Now[0] :=i+sc; {\*Изменяем длину разбиения.\*}*

*For j :=1 To sc Do Now[j+i] :=1; {\*3аписываем*

*минимально возможные элементы,*

*т.е. единицы.\*}*

*End;*

А будет ли работать данная логика для последнего разбиения (в нашем примере для «8»)? Оказывается, что нет. Проверьте.

Считаем, что в Now хранится первое разбиение, а в массиве Last — последнее. Они формируются, например, в процедуре типа In.it. И схема генерации....

*While Not(Eq(Now,Last) ) Do Begin {\*Фунция Eq дает*

*"истину", если текущее распределение*

*совпадает с последним, и "ложь"*

*в противном случае, можно сделать проще -While Now[0]<>1 Do ...\*}*

*GetNext;*

*Print; {\*Вывод разбиения.\*}*

*End;*

*Третья задача.* По номеру разбиения (L) получить само разбиение (Now). Нумерация разбиений осуществляется относительно введенного отношения порядка. Пусть L равно 0. Сделаем «набросок» логики.

*sc:=N; i:=1; {\*sc - число, которое представлено*

*разбиением, i - номер позиции в разбиении.\*}*

*While scOO Do Begin {\*Пока число не исчерпано.\*}*

*j:=1; {\*Число, которое должно записываться*

*на место i в разбиении.\*}*

*??????????? {\*3начение j определяется по значению*

*L, пока не знаем как.\*}*

*Now[i]:=j; sc:=sc-j; Inc(i); {\*Формируем текущую*

*позицию разбиения и готовимся к переходу для*

*формирование новой.\*}*

*End;*

Как это ни странно, но если вместо знаков «???» оставить пустое место («заглушку»), то при L, равном 0, данный набросок будет работать — на выходе логики мы получим начальное разбиение, состоящее из единиц. Пусть L равно 1. Обратимся ко второй таблице данного материала. Проверяем SmallSc[8,1]. Если L больше или равно этому значению, то мы уменьшим значение L на SmallSc[8,1], увеличим значение j на единицу и, в конечном итоге, получим разбиение 2111111.

Итак, что нам необходимо сделать? Да просто просмотреть строку массива SmallSc с номером sc и определить последний элемент строки, его номер в j, который меньше текущего значения L. Мы последовательно отсекаем разбиения числа sc, начинающиеся с чисел j. Ниже приведен полный текст процедуры.

*Procedure GetWhByNum (L:LongInt);*

*Var x,j,sc:Integer;*

*Begin*

*sc:=N; i:=1;*

*While sc<>0 Do Begin j:=1;*

*While L>=SmallSc[sc,j] Do Begin*

*Dec(L,SmallSc[sc,j]); Inc(j);*

*End;*

*Now[i]:=j; sc:=sc-j; Inc (i);*

*End;*

*Now[0] :=i-1; {\*Корректируем длину разбиения.\*}*

*End;*

*Четвертая задача.* По разбиению (Now) получить его номер. Принцип прост. Брать текущее число из разбиения, начиная с первого, и смотреть на его «вклад» в количество номеров.

Например, разбиение «22211». Определяем, сколько номеров приходится на первую 2, затем на вторую и т. д.

*Function GetNumByWh:LongInt;*

*Var i,jk,p:Integer;*

*sc:LongInt;*

*Begin*

*sc:=1; {\*Формируем номер разбиения.\*}*

*jk:=N; {\*Номер строки в массиве SmallSc.\*}*

*For i:=1 To Now[0] Do Begin {\*Просматриваем*

*разбиение.\*}*

*For p:=0 To Now[i]-1 Do sc:=sc+SmallSc[jk,p];*

*{\*3начение числа из разбиения определяет*

*число столбцов в массиве SmallSc.\*}*

*jk:=jk-Now[i]; {\*Изменяем номер строки.\*}*

*End;*

*GetNumByWh:=sc;*

*End;*



Итак, номер определен, он равен 4.

3 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

3.1. Составить алгоритмы программ, реализующих генерацию всех разбиений натурального числа на слагаемые.

3.2. Создать программы, реализующие генерацию всех разбиений натурального числа на слагаемые.

4 СОДЕРЖАНИЕ ОТЧЕТА ПО РАБОТЕ

4.1 Исходное задание и цель работы.

4.2 Блок-схема программы по п.3.1.

4.3 Распечатка текста программы по п.3.2.

4.4 Контрольный пример и результаты машинного расчета.

4.5 Выводы по работе.

5 КОНТРОЛЬНЫЕ ВОПРОСЫ

5.1. Дайте определение разбиения натурального числа на слагаемые.

5.2. Что такое диаграмма Феррерса?

5.3. Определите лексикографический порядок на множестве разбиений.

5.4. Опишите известные Вам алгоритмы генерации всех разбиений натурального числа на слагаемые.

5.5. Как получить номер заданного разбиения?

5.6. Как получить разбиение по его номеру?

**ЛАБОРАТОРНАЯ РАБОТА № 17**

ГЕНЕРАЦИЯ ПОДМНОЖЕСТВ

1 ЦЕЛЬ РАБОТЫ

Целью работы является изучение различных алгоритмов генерации всех подмножеств заданного множества.

2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Проблема. Научиться работать с подмножествами iV-элементного множества натуральных чисел от 1 до N такими, что каждое следующее подмножество отличалось бы от предыдущего только на один элемент. Решаем традиционные четыре задачи: подсчет количества подмножеств; генерация подмножеств; определение подмножества по номеру и определение номера по подмножеству.

*Первая задача*. Количество подмножеств множества из N элементов равно 2N. Эта задача проста, если значение N не очень большое. Однако уже при N=15 количество подмножеств 32768, поэтому тип данных Integer требуется использовать осторожно. Пусть N меньше или равно 100. При таких значениях N необходимо использовать элементы «длинной» арифметики, а именно: умножение «длинного» числа на короткое и вывод «длинного» числа.

Процедура вычисления количества подмножеств множества из N элементов. Есть небольшой «нюанс». Возводить двойку в степень N — утомительное занятие, сократим его по возможности.

*Procedure Calc;*

*Var i: Integer;*

*Tmp: TLong;*

*Begin*

*i :=N; { \*Информация для размышления. Основанием*

*системы счисления в нашей задаче является*

*10000, а два в степени 13 равно 8192.\*}*

*Res[1]:=1; Res[0]:=1;{\*Два в степени 0 равно !•\*}*

*While i>0 Do Begin*

*If i>13 Then Begin Mul(Res, 1 shl 13, Tmp);*

*i:=i-13; End {\*Напомним, что команда shl -сдвиг влево, итак 1 сдвигается на 13*

*разрядов влево, т.е. мы умножаем на 213*

*А почему нельзя умножать на 214?\*}*

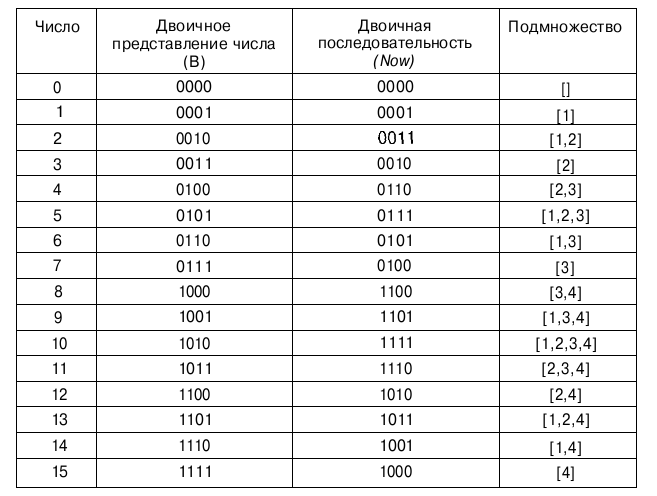
*Else Begin Mul (Res, 1 shl i, Tmp); i:=0; End;*

*Res:=Tmp;*

*End;*

*End;*

Вторая задача. Подмножества N-элементного множества натуральных чисел от 1 до N будем представлять двоичными последовательностями длины N — Now[l], Now[2],..., Now[N] (массив Now). Элемент Now[i], равный единице, говорит о том, что N—i+1 принадлежит подмножеству.



Во втором столбце таблицы записаны двоичные представления чисел от 0 до 24-1. Каждое из чисел мы подвергли преобразованию (кодирование по Грею), заменив каждую двоичную цифру, кроме первой, на ее сумму с предыдущей цифрой по модулю 2, таким образом, получив элементы массива Now. Т. е., Now[l]:~B[l] и для всех i от 2 до N. В четвертом столбце таблицы записаны подмножества, построенные по единичным элементам массива Now. Обратите внимание на то, что каждое следующее отличается от предыдущего только на один элемент. Запишем формирование массива Now в виде процедуры.

*Procedure TransferB;*

*Var i: Integer;*

*Begin*

*FillChar (Now, SizeOf (Now) , 0) ;*

*Now[l]:=B[1];*

*For i:=2 To N Do Now[i] :=B[i] Xor B[i-1] ;*

*End;*

*Procedure TransferToB;*

*Var i, j: Integer;*

*Begin*

*FillChar(B, SizeOf(B), 0);*

*B[l] :=Now[l] ;*

*For i:=2 To N Do B[i] :=Now[i] Xor B[i-1] ;*

*End;*

Воспринимая данное преобразование как факт, осмысление которого лежит за пределами данного материала, генерацию подмножеств можно осуществлять с использованием массива В. В качестве материала для самостоятельного осмысления предлагается следующая изящная процедура генерации очередного разбиения.

*Procedure GetNext;*

*Var i: Integer;*

*Begin*

*i:=N+l;*

*Repeat*

*Dec (i) ;*

*B[i] :=B[i] Xor 1;*

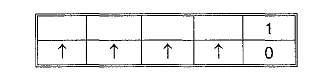
*Until B[i]=l;*

*Now[i]:=Now[i] Xor 1;*

*End; {\*Будет ли работать данная процедура для*

*последовательности 1111 (N=4) ?\*}*

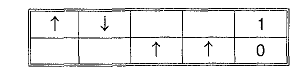
Можно решать задачу по-другому. Вспомним зрительный образ «лесенки» со стрелками, который мы использовали при генерации перестановок, таких, что каждая следующая отличалась от предыдущей только одной транспозицией соседних элементов. В данной задаче следует использовать не лесенку, а прямоугольное поле 2\*N. Шашки в начальный момент находятся в первой строке (она соответствует нулевым значениям в двоичной последовательности) и двигаются в направлении стрелок. Определяется первая справа шашка, которую можно сдвинуть в направлении стрелки. Сдвигаем ее, а у шашек, находящихся справа от нее, меняем направление движения. Последовательности 0000 при N=4 соответствует следующая «картинка».



А последовательности 0100:



Следующее состояние поля будет иметь вид (последовательность 1100):



и процесс генерации окончится при состоянии (последовательность 1000)



*Третья задача*. Получение последовательности В по номеру разбиения на подмножества L соответствует записи числа L в двоичной системе счисления с использованием массива В. Двоичный разряд числа L — это значение соответствующего элемента массива В.

*Procedure GetWhByNum (L: Longlnt);*

*Var i: Integer;*

*Begin*

*For i:=l To N Do Begin*

*B[N-i+l] :=L And 1; {\*Выделяем младший разряд*

*числа L. \*} •*

*L:=L shr 1; {\*'Уменьшаем L в два раза (деление*

*на 2) . \*}*

*End;*

*End;*

*Четвертая задача.* Логика получения номера разбиения на подмножества по последовательности соответствует в нашем случае переводу числа из двоичной системы счисления в десятичную.

*Function GetNumByWh: Longlnt;*

*Var sc: Longlnt;*

*i: Integer;*

*Begin*

*sc:=0;*

*For i:=l To N Do sc:=sc\*2 + B[i] ;*

*GetNumByWh:=sc+l;*

*End;*

3 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

3.1. Составить алгоритмы программ, реализующих генерацию всех подмножеств заданного множества.

3.2. Создать программы, реализующие генерацию всех подмножеств заданного множества.

4 СОДЕРЖАНИЕ ОТЧЕТА ПО РАБОТЕ

4.1 Исходное задание и цель работы.

4.2 Блок-схема программы по п.3.1.

4.3 Распечатка текста программы по п.3.2.

4.4 Контрольный пример и результаты машинного расчета.

4.5 Выводы по работе.

5 КОНТРОЛЬНЫЕ ВОПРОСЫ

5.1. Сколько существует подмножеств N-элементного множества?

5.2. Как кодируются подмножества заданного множества?

5.3. Что такое код Грэя?

5.4. Опишите известные Вам алгоритмы генерации всех подмножеств заданного множества.

5.5. Как получить номер заданного подмножества?

5.6. Как получить подмножество по его номеру?

**ЛАБОРАТОРНАЯ РАБОТА № 18**

ГЕНЕРАЦИЯ СКОБОЧНЫХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ

1 ЦЕЛЬ РАБОТЫ

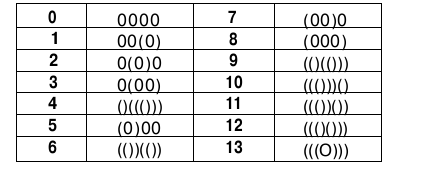
Целью работы является отработка рассмотренных методов построения алгоритмов комбинаторной генерации на примере задачи генерации всех правильных расстановок заданного числа пар скобок.

2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Последовательность из 2\*N скобок правильная, если для любого значения i (l<i<2\*N), число открывающих скобок, например «(», больше или равно числу закрывающих скобок «)» и общее число открывающих скобок в последовательности равно числу закрывающих.

Пример. Последовательность (())()() — правильная, последовательность (()))(() — не является правильной.

Мы по-прежнему решаем наши четыре задачи. Однако изменим порядок их рассмотрения. Начнем с генерации последовательностей. Пусть N равно 4. В нечетных столбцах таблицы приведены номера последовательностей, а в соседних — соответствующие последовательности.



Что же мы делаем? Какое отношение порядка введено на множестве последовательностей? Сколько существует таких последовательностей для заданного значения N? Вопросов много, ответов пока нет.

Пусть наша последовательность хранится в переменной строкового типа Now. Мы просматриваем строку справа налево до первой комбинации скобок типа «)(». Она обязательно должна встретиться, ибо, считаем, что текущая последовательность не является последней. Заменяем эту комбинацию скобок на «()». Подсчитываем с начала строки (последовательности), на сколько число открывающих скобок больше числа закрывающих, дописываем это количество закрывающих скобок, и если строка не длины 2\*N, то дополняем ее комбинациями скобок «()».

*Procedure GetNext;*

*Var i, j, sc: Integer;*

*Begin*

*i:=N\*2;*

*While (i>l) And (Now[i-1] + Now[i]<>') (')*

*Do Dec(i);{\*Поиск комбинации ") (".\*}*

*Now[i-1]: = ' ('; Now[i]: = ') ';{"Замена. \*}*

*Now:=Copy(Now,I,i);*

*sc:=0;{"Подсчет разности числа открывающих и*

*числа закрывающих скобок. \*}*

*For j:=l To i Do If Now[j ] = ' (' Then Inc(sc) Else*

*Dec (sc) ;*

*While sc<>0 Do Begin{"Дополнение*

*подпоследовательности закрывающими скобками. \*}*

*Now:=Now+')';Dec(sc) ;*

*End;*

*While Length (Now) <2\*N Do Now:=Now+' () ' ;*

*{"Дополняем строку, максимально "ухудшая"*

*ее, т.е. из остатка строки делаем ее*

*начальное значение.\*}*

*End;*

Даны две последовательности Р1 и Р2 например ()()(()) и ()(()())• Какая из них имеет больший номер? Просматриваем последовательности слева направо, сравнивая соответствующие символы. Найдем первое значение i, при котором Р1[i]<> Р2[i]. Если окажется, что Р1[i]=’)’, а Р2[i]=’(’, то Р1 имеет меньший номер, чем Р2.

Перейдем к первой задаче. Введем понятие частично правильной скобочной последовательности. Последовательность частично правильная, если для любой позиции в последовательности число открывающих скобок больше или равно числу закрывающих, естественно, что количество тех и других считается от начала последовательности. Так, последовательность «(((((» является частично правильной, а последовательность «(()))((» — нет, для позиции 5 число закрывающих больше числа открывающих скобок. Оформим наши дальнейшие рассуждения в виде таблицы. Номер строки указывает на число скобок в последовательности, номер столбца — на разность между числом открывающих и закрывающих скобок, имя таблицы ScSmall. Элемент таблицы ScSmall[i,j] равен количеству частично правильных последовательностей из i скобок, причем разность между числом открывающих и закрывающих равна j. Это ключевой момент для понимания: «разность — номер столбца». На диагонали таблицы записаны 1, число последовательностей, состоящих из одних открывающих скобок, а они попадают под определение частично правильной последовательности. Элементы таблицы ScSmall[i,j] равны 0, если i и j числа разной четности.

Примеры.

ScSmall[3,l]=2: ((), ()(.

ScSmall[4,2]=3: (((), (()(, ()((.

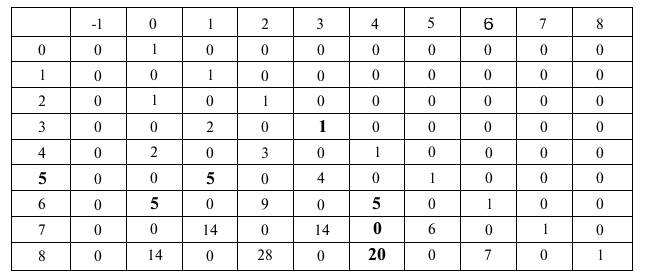
ScSmall[4,0]=2: ()(), (()).

ScSmall[5.1]=5: ()()(, (())(, ((()). (()(). ()(().

И «крохотный факт»:

ScSmall[5,l]=ScSmall[4,0]+ScSmall[4,2].

Дописываем в конец последовательностей из ScSmall[4,0] открывающую скобку, а в конец ScSmall[4,2] — закрывающую. А это уже реальный путь подсчета числа последовательностей. Ответом задачи будет ScSmall[2\*N,0].



Ограничимся при вычислениях диапазоном типа Longlnt.

*Const SmallN=37;*

*Var ScSmall: Array[-1. .SmallN + 1, -1. .SmallN + 1]*

*Of Longlnt;*

Процедура формирования таблицы вряд ли нуждается в пояснениях.

*Procedure FillSmallSc;*

*Var i , j : Integer;*

*Begin*

*FillChar (ScSmall, SizeOf(ScSmall), 0) ;*

*ScSmall[0, 0]:=l;*

*For i:=l To SmallN-1 Do*

*For j:=0 To SmallN Do ScSmall[i,j]:=ScSmall*

*[i~l,j-l]+ScSmall[i-l,j+l];*

*End;*

Дополним ее функцией определения количества частично правильных последовательностей из массива SCSmall.

*Function GetSmallSc (N, Up: Longlnt): Longlnt;*

*Begin*

*If (N<0) Or (Up<0) Then GetSmallSc:=0*

*Else*

*If (N>SmallN) Or (Up>SmallN) Then GetSmallSc*

*:=MaxLongInt*

*Else GetSmallSc:=ScSmall[N, Up];*

*End;*

Вычисление последовательности по номеру и обратное преобразование можно вынести на самостоятельную часть работы.

Ниже приведены соответствующие процедура и функция. В переменной Up, как и выше, формируется текущая разность между числом открывающих и закрывающих скобок.

*Procedure GetWhByNumfL: Longlnt);*

*Var i, Up: Integer;*

*P: Longlnt;*

*Begin*

*Now:=’’; Up:=0;*

*For i:=l To N\*2 Do Begin*

*P:=GetSmallSc(N\*2-i, Up-1);*

*If (L>=P) Then Begin Now:=Now+'('; Inc (Up);*

*L:=L-P; End*

*Else Begin Dec (Up); Now:=Now+')'; End;*

*End;*

*End;*

*Function GetNumByWh: Longlnt;*

*Var sc: Longlnt;*

*i, Up: Integer;*

*Begin*

*sc:=l; Up:=0;*

*For i:=l To N\*2 Do*

*If Now[i]='(' Then Begin sc:=sc+GetSmallSc*

*(N\*2-i,Up-l) ;Inc (Up) ;End*

*Else Dec(Up);*

*GetNumByWh:=sc;*

*End;*

3 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

3.1. Составить алгоритмы программ, реализующих генерацию всех правильных скобочных последовательностей.

3.2. Создать программы, реализующие генерацию всех правильных скобочных последовательностей..

4 СОДЕРЖАНИЕ ОТЧЕТА ПО РАБОТЕ

4.1 Исходное задание и цель работы.

4.2 Блок-схема программы по п.3.1.

4.3 Распечатка текста программы по п.3.2.

4.4 Контрольный пример и результаты машинного расчета.

4.5 Выводы по работе.

5 КОНТРОЛЬНЫЕ ВОПРОСЫ

5.1. Что такое правильная и полуправильная скобочные последовательности?

5.2. Сколько существует правильных скобочных последовательностей заданной длины?

5.3. Как ввести отношение порядка на множестве правильных скобочных последовательностей?

5.4. Опишите известные Вам алгоритмы генерации правильных скобочных последовательностей.

5.5. Как получить номер заданной скобочной последовательности?

5.6. Как получить скобочную последовательность по ее номеру?

**СОДЕРЖАНИЕ**

**ВВЕДЕНИЕ**………………………………………………………….………3

**Тема 1**. **АЛГОРИТМЫ НА ГРАФАХ**…………………………….…………4

Лабораторная работа № 1-2. МАТРИЧНЫЕ СПОСОБЫ

ПРЕДСТАВЛЕНИЯ ГРАФОВ………….……..4

Лабораторная работа № 3-4. ПОИСК КРАТЧАЙШИХ

ПУТЕЙ НА ГРАФАХ……..……..…….…..12

Лабораторная работа № 5. ПОСТРОЕНИЕ КРАТЧАЙШИХ

ОСТОВЫХ ДЕРЕВЬЕВ ГРАФА…..……...22

Лабораторная работа № 6. РАСКРАСКА ГРАФА………………...………29

Лабораторная работа № 7-8. АЛГОРИТМ

ФОРДА-ФАЛКЕРСОНА…………………………35

Лабораторная работа № 9. ЦИКЛЫ НА ГРАФАХ……………………..…46

**Тема 2**. **АЛГОРИТМЫ КОМБИНАТОРНОГО ПЕРЕБОРА**………..…53

Лабораторная работа № 10-11. ГЕНЕРАЦИЯ ПЕРЕСТАНОВОК….…....53

Лабораторная работа № 12-13. ГЕНЕРАЦИЯ РАЗМЕЩЕНИЙ……...…..64

Лабораторная работа № 14-15. ГЕНЕРАЦИЯ СОЧЕТАНИЙ………….....73

Лабораторная работа № 16. ГЕНЕРАЦИЯ РАЗБИЕНИЙ……………..….83

Лабораторная работа № 17. ГЕНЕРАЦИЯ ПОДМНОЖЕСТВ……..…….91

Лабораторная работа № 18. ГЕНЕРАЦИЯ СКОБОЧНЫХ

ПОСЛЕДОВАТЕЛЬНОСТЕЙ…………...……98

**БИБЛИОГРАФИЧЕСКИЙ СПИСОК**………………………………...91

**БИБЛИОГРАФИЕСКИЙ СПИСОК.**

1. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ. М.: Вильямс, 2007.
2. Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы. М.: Вильямс, 2007.
3. Окулов С.М. Программирование в алгоритмах. М.: Бином, 2007.
4. Кнут Д. Искусство программирования. Т. 4 Вып.2. Генерация всех кортежей и перестановок. М.: Вильямс, 2008.
5. Кнут Д. Искусство программирования. Т. 4 Вып.2. Генерация всех сочетаний и разбиений. М.: Вильямс, 2008.
6. Кнут Д. Искусство программирования. Т. 4 Вып.2. Генерация всех деревьев. История комбинаторной генерации. М.: Вильямс, 2008.
7. Седжвик Р. Фундаментальные алгоритмы на С++. М.: Вильямс, 2011.
8. Ху Т.Ч., Шинг М.Т. Комбинаторные алгоритмы Нижний Новгород: Изд-во Нижегородского госуниверситета им. Н.И. Лобачевского, 2004.
9. Новиков Ф.А. Дискретная математика для программистов. СПб.: Питер, 2012.

**Шутов Антон Владимирович**

**Медведев Юрий Алексеевич**

СТРУКТУРЫ И АЛГОРИТМЫ КОМПЬЮТЕРНОЙ

ОБРАБОТКИ ДАННЫХ

Лабораторный практикум

Издается в авторской редакции

|  |  |
| --- | --- |
| Подписано в печать 20.12.2013  Усл. п. л. – 6,9  Заказ 08 - 13 | Формат 84 x 108 1/32  Уч. –изд. л. – 7,1  Тираж 50 экз. |

Отпечатано в отделе оперативной полиграфии ВГГУ

600014, г. Владимир, ул. Университетская, 2, тел. 33-87-40