**Лекция 3**

**КОМБИНАТОРНЫЕ МЕТОДЫ РЕШЕНИЯ ОПТИМИЗАЦИОННЫХ ЗАДАЧ**

**3. Генерация перестановок**

Наиболее известным методом построения множества  всех перестановок конечного множества  является ***алгоритм Джонсона – Троттера***. Алгоритм подразумевает, что все элементы множества  можно единственным способом перечислить в порядке возрастания. Отметим, что для конечного множества такой порядок всегда можно установить. В простейшем случае элементы конечного множеств  можно перенумеровать и считать, что  если  В дальнейшем будем предполагать, что  – отрезок натурального ряда. Это позволяет сравнивать элементы естественным способом и никак не мешает обобщению.

Каждый элемент исходного множества  помечается специальным символом – стрелкой, которая может быть направлена влево или вправо. Первая перестановка в алгоритме Джонсона – Троттера выглядит следующим образом:



В алгоритме используется понятие ***мобильного элемента***. Элемент  последовательности элементов множества  называется мобильным, если соответствующая ему стрелка указывает на меньший соседний элемент. В первой перестановке все элементы кроме самого левого являются мобильными.

Построение множества всех перестановок с помощью алгоритма Джонсона – Троттера сводится к следующей процедуре:

1. Построить первую перестановку. Первая перестановка – это последовательность всех элементов множества  перечисленных в порядке возрастания. Стрелки всех элементов последовательности направлены влево.
2. Найти наибольший мобильный элемент в текущей перестановке. Если в последовательности нет мобильного элемента, то построены все перестановки элементов множества  – алгоритм закончил свою работу.

3. Поменять местами наибольший мобильный элемент и элемент, на который указывает стрелка наибольшего мобильного элемента.

4. Найти все элементы, большие, чем мобильный элемент (если они есть) и изменить их стрелки на противоположное направление.

5. Перейти к пункту 2.

Схема алгоритма генерации множества всех перестановок множества  приведена на рис. 1.



Рис. 1. Схема работы алгоритма Джонсона – Троттера

Второй слева столбец на рис. 1 – массив индексов, генерируемый с помощью алгоритма Джонсона – Троттера. Каждый массив индексов является одной из перестановок элементов множества . Количество массивов составляет:



Самый правый столбец (обозначен ) – множество всех перестановок элементов  Перестановки формируются как результат индексации массива 

**Реализация генератора перестановок на языке C++**

На рис. 2 и 3 представлена программная реализация генератора перестановок, а на рис. 4 и 5 приведен пример его применения.

// Combi.h

**#pragma once**

**namespace combi**

**{**

**struct permutation** // генератор перестановок

**{**

**const static bool L = true;** // левая стрелка

**const static bool R = false;** // правая стрелка

**short n,** // количество элементов исходного множества

**\*sset;** // массив индексов текущей перестановки

**bool \*dart;** // массив стрелок (левых-L и правых-R)

**permutation (short n = 1);** // конструктор (количество элементов исходного множества)

**void reset();** // сбросить генератор, начать сначала

**\_\_int64 getfirst();** // сформировать первый массив индексов

**\_\_int64 getnext();** // сформировать случайный массив индексов

**short ntx(short i);** // получить i-й элемент масива индексов

**unsigned \_\_int64 np;** // номер перествновки 0,... count()-1

**unsigned \_\_int64 count() const;** // вычислить общее кол. перестановок

**};**

**};**

Рис. 2. Шаблон структуры генератора перестановок

Рис. 3. Реализация функций генератора перестановок

// Combi.cpp

**#include "stdafx.h"**

**#include "Combi.h"**

**#include <algorithm>**

**#define NINF ((short)0x8000)**

**namespace combi**

**{**

**permutation::permutation(short n)**

**{**

**this->n = n;**

**this->sset = new short[n];**

**this->dart = new bool[n];**

**this->reset();**

**};**

**void permutation::reset()**

**{ this->getfirst(); };**

**\_\_int64 permutation::getfirst()**

**{**

**this->np = 0;**

**for (int i = 0; i < this->n; i++)**

**{this->sset[i] = i; this->dart[i] = L;};**

**return (this->n > 0)?this->np:-1;**

**};**

**\_\_int64 permutation::getnext() //**

**{**

**\_\_int64 rc = - 1;**

**short maxm = NINF, idx = -1;**

**for(int i = 0; i < this->n; i++)**

**{**

**if ( i > 0 &&**

**this->dart[i] == L &&**

**this->sset[i] > this->sset[i-1] &&**

**maxm < this->sset[i]) maxm = this->sset[idx = i];**

**if ( i < (this->n-1)&&**

**this->dart[i] == R &&**

**this->sset[i] > this->sset[i+1]&&**

**maxm < this->sset[i]) maxm = this->sset[idx = i];**

**};**

**if (idx >= 0)**

**{**

**std::swap(this->sset[idx],**

**this->sset[idx+(this->dart[idx]== L?-1:1)]);**

**std::swap(this->dart[idx],**

**this->dart[idx+(this->dart[idx]== L?-1:1)]);**

**for (int i = 0; i < this->n; i++)**

**if (this->sset[i] > maxm) this->dart[i] = !this->dart[i];**

**rc = ++this->np;**

**}**

**return rc;**

**};**

**short permutation::ntx(short i){return this->sset[i];};**

unsigned \_\_int64 fact(unsigned \_\_int64 x){return (x == 0)?1:(x\*fact(x-1));};

**unsigned \_\_int64 permutation::count() const {return fact(this->n); };**

**}**

// --- Main

**#include "stdafx.h"**

**#include <iostream>**

**#include "Combi.h"**

**#include <iomanip>**

**int \_tmain(int argc, \_TCHAR\* argv[])**

**{**

**setlocale(LC\_ALL, "rus");**

**char AA[][2]= {"A", "B", "C", "D"};**

**std::cout<<std::endl<<" --- Генератор перестановок ---";**

**std::cout<<std::endl<<"Исходное множество: ";**

**std::cout<<"{ ";**

**for (int i = 0; i < sizeof(AA)/2; i++)**

**std::cout<<AA[i]<<((i< sizeof(AA)/2-1)?", ":" ");**

**std::cout<<"}";**

**std::cout<<std::endl<<"Генерация перестановок ";**

**combi::permutation p(sizeof(AA)/2);**

**\_\_int64 n = p.getfirst();**

**while (n >= 0)**

**{**

**std::cout<<std::endl<<std::setw(4)<< p.np <<": { ";**

**for (int i = 0; i < p.n; i++)**

**std::cout<<AA[p.ntx(i)]<<((i< p.n-1)?", ":" ");**

**std::cout<<"}";**

**n = p.getnext();**

**};**

**std::cout<<std::endl<<"всего: " << p.count()<<std::endl;**

**system("pause");**

**return 0;**

**}**

Рис. 4. Пример применения генератора перестановок

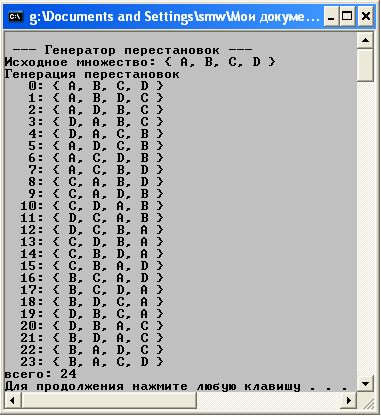


Рис. 5. Результат выполнения программы, представленной на рис. 4

Генератор реализован в виде структуры **permutation**. Структураимеет один конструктор. С помощью параметра конструктору передается размерность исходного множества.

Состояние генератора определяется значениями четырех переменных: **n** (количество элементов в исходном множестве), **sset** (указатель на массив индексов), **dart** (указатель на массив стрелок), **np** (номер текущей перестановки).Всепеременные инициализируются в конструкторе. Значение **np** увеличивается на единицу после генерации очередной перестановки, значение остальных переменных остается неизменным. Элементы массивов **sset** и **dart** меняются при каждом цикле работы генератора в соответствии с алгоритмом Джонсона –Троттера.

Кроме конструктора структура **permutation** содержит еще пять функций.

Функция **getfirst** (см.рис. 4) не имеет параметров и предназначена для формирования первой перестановки, которая представляет собой упорядоченную последовательность **n** (переменная структуры на рис. 3) неотрицательных целых чисел (см. рис. 1).

Функция **reset** позволяет сбросить текущее состояние генератора для того, чтобы начать его работу сначала. Работа функции сводится к вызову функции **getfirst**. Функция **reset** реализована главным образом для унификации интерфейсов всех генераторов.

Функция **getnext** формирует массив индексов следующей перестановки и увеличивает значение переменной **np** на единицу. При каждом вызове функции в массиве **sset** отыскивается максимальный мобильный элемент и элемент, на который указывает его стрелка, эти элементы меняются местами. Если существуют элементы **sset**,большие, чем найденный мобильный, то соответствующие им стрелки инвертируются.

Функция **ntx** возвращает значение элемента массива индексов по индексу этого элемента и служит для сокращения записи при переборе элементов массива.

Функция **count** вычисляет и возвращает общее количество перестановок **n** элементов множества.

**Решение задачи коммивояжера c использованием генератора перестановок**

Задача коммивояжера как и задача о рюкзаке, является классической задачей, решаемой с помощью перебора. Сформулируем условие задачи.

Коммивояжер (бродячий торговец) должен найти минимальный кольцевой маршрут обхода  городов. Расстояние  между каждой парой   городов считается известным.

Математическая модель задачи может быть записана следующим образом:

где  – неизвестные (номера выбранных городов), которые требуется найти.

Решением задачи будет вектор . Каждый элемент этого вектора может принимать целое значение из отрезка . При этом все значения  должны быть разными.

Решение задачи сводится к генерации всех допустимых векторов , вычислению функции  и выбору вектора, соответствующего минимальному значению функции.

На рис. 6 изображена схема решения задачи коммивояжера с применением генератора перестановок. Задача решается для пяти городов.



Рис. 6. Схема решения задачи коммивояжера

Расстояние между городами задается следующей матрицей 



Элемент  матрицы  определяет расстояние между городами  и  где  Факт отсутствия пути из города  в город  обозначается значением  (бесконечность) соответствующего элемента.

Следует отметить следующие два момента, прежде чем будет пояснена схема на рис. 6:

1. При построении оптимального маршрута коммивояжера выбор стартового (он же является и конечным, так как маршрут кольцевой) города никак не влияет на конечный результат.
2. Если задано  городов, то перебор следует осуществлять только для  городов, поскольку стартовый город можно зафиксировать.

На рис. 6 в качестве стартового города выбран город с номером . Поэтому перебор маршрутов осуществляется для городов с номерами 1, 2, 3, 4.

Левая часть схемы на рис. 6 практически идентична левой части схемы на рис. 1. Главное отличие заключается в том, что в качестве исходного массива выбран  Причем перестановкам подлежит только внутренняя, заключенная между нулями, часть исходного массива. В остальном принцип построения перестановок тот же, что и на рис 1.

В правой части схемы изображены все возможные кольцевые маршруты, которые образованы из исходного массива путем перестановок всех, кроме обрамляющих нулей, элементов. Количество маршрутов  равно количеству перестановок из четырех городов. Для каждого маршрута в округлой рамке указана длина. Длина кольцевых маршрутов, для которых не могут быть построены в силу отсутствия пути хотя бы между одной парой городов, на схеме обозначена символом  Оптимальный маршрут длиной  единиц выделен затемненной рамкой.

На рис. 7 и 8 представлен пример реализации на C++ функции **salesman**, вычисляющей оптимальный кольцевой маршрут коммивояжера. Функция имеет два входных параметра: **n** (количество городов) и **d** (двумерный массив размерностью **n** на **n**, содержащий элементы матрицы расстояний), а также один возвращаемый параметр **r** (массив размерностью **n**, содержащий оптимальный маршрут).

//-- Salesman.h

// -- решение задачи коммивояжера перебором вариантов

**#define INF 0x7fffffff** // бесконечность

**#include "Combi.h"**

**int salesman (** // функция возвращает длину оптимального маршрута

**int n,** // [in] количество городов

**const int \*d,** // [in] массив [n\*n] расстояний

**int \*r** // [out] массив [n] маршрут 0 x x x x

**);**

Рис. 7. Функция **salesman**, решающая задачу коммивояжера

// -- Salesman.cpp

**#include "stdafx.h"**

**#include "Salesman.h"**

**int sum (int x1, int x2)** // суммирование с учетом бесконечности

**{ return (x1 == INF || x2 == INF)? INF: (x1 + x2); };**

**int\* firstpath(int n)** // формирование 1го маршрута 0,1,2,..., n-1, 0

**{**

**int \*rc = new int[n+1]; rc[n] = 0;**

**for (int i = 0; i < n; i++) rc[i] = i;**

**return rc;**

**};**

**int\* source(int n)** // формирование исходного массива 1,2,..., n-1

**{**

**int \*rc = new int[n-1];**

**for (int i = 1; i < n; i++) rc[i-1] = i;**

**return rc;**

**};**

**void copypath(int n, int \*r1, const int \*r2)** // копировать маршрут

**{ for (int i = 0; i < n; i++) r1[i] = r2[i]; };**

**int distance(int n, int \*r, const int \*d)** // длина маршрута

**{**

**int rc = 0;**

**for (int i = 0; i < n-1; i++) rc = sum(rc, d[r[i]\*n+r[i+1]]);**

**return sum (rc, d[r[n-1]\*n + 0]);** //+последняя дуга (n-1,0)

**};**

**void indx(int n, int \*r, const int \*s, const short \*ntx)**

**{ for (int i = 1; i < n; i++) r[i] = s[ntx[i-1]];}**

**int salesman (**

**int n,** // [in] количество городов

**const int \*d,** // [in] массив [n\*n] расстояний

**int \*r** // [out] массив [n] маршрут 0 x x x x

**)**

**{**

**int \*s = source(n), \*b = firstpath(n), rc = INF, dist = 0;**

**combi::permutation p(n-1);**

**int k = p.getfirst();**

**while (k >= 0)** // цикл генерации перестановок

**{**

**indx(n, b, s, p.sset);** // новый маршрут

**if ((dist = distance(n,b,d)) < rc)**

**{ rc = dist; copypath(n,r,b); }**

**k = p.getnext();**

**};**

**return rc;**

**}**

Рис. 8. Реализация функции **salesman**

В функции **salesman** применяется генератор перестановок (**combi::permutation**). Кроме того она вызывает шесть вспомогательных функций: **indx** (формирование перестановки городов на основе массива идексов), **distance** (вычисление длины кольцевого маршрута), **copypath** (копирование маршрута), **source** (формирование исходного массива), **firstpath** (формирование первого маршрута) и **sum** (суммирование двух чисел с учетом того, что одно из них может быть равно бесконечности).

Функция **salesman** в цикле генерирует все возможные кольцевые маршруты, вычисляет для каждого маршрута длину (функция **distance**), фиксирует оптимальный маршрут (функция **copypath**) и возвращает длину оптимального пути или значение **INF**, что обозначает отсутствие кольцевых маршрутов.

На рис. 9 и 10 приведен пример вызова функции **salesman** для решения задачи с исходными данными к схеме на рис. 6.

// --- main

**#include "stdafx.h"**

**#include <iostream>**

**#include <iomanip>**

**#include "Salesman.h"**

**#define N 5**

**int \_tmain(int argc, \_TCHAR\* argv[])**

**{**

**setlocale(LC\_ALL, "rus");**

**int d[N][N] = {** //0 1 2 3 4

**{ 0, 45, INF, 25, 50},** // 0

**{ 45, 0, 55, 20, 100},** // 1

**{ 70, 20, 0, 10, 30},** // 2

**{ 80, 10, 40, 0, 10},** // 3

**{ 30, 50, 20, 10, 0}};** // 4

**int r[N];** // результат

**int s = salesman (**

**N,** // [in] количество городов

**(int\*)d,** // [in] массив [n\*n] расстояний

**r** // [out] массив [n] маршрут 0 x x x x

**);**

**std::cout<<std::endl<<"-- Задача коммивояжера -- ";**

**std::cout<<std::endl<<"-- количество городов: "<<N;**

**std::cout<<std::endl<<"-- матрица расстояний : ";**

**for(int i = 0; i < N; i++)**

**{**

**std::cout<<std::endl;**

**for (int j = 0; j < N; j++)**

**if (d[i][j]!= INF) std::cout<<std::setw(3)<<d[i][j]<< " ";**

**else std::cout<<std::setw(3)<<"INF"<<" ";**

**}**

**std::cout<<std::endl<<"-- оптимальный маршрут: ";**

**for(int i = 0; i < N; i++) std::cout<<r[i]<<"-->"; std::cout<<0;**

**std::cout<<std::endl<<"-- длина маршрута : "<<s;**

**std::cout<<std::endl;**

**system("pause");**

**return 0;**

**}**

Рис. 9. Пример решения задачи коммивояжера

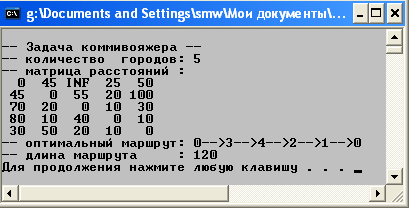


Рис. 10. Результат выполнения программы, представленной на рис. 4.9

На рис. 11 представлена программа, позволяющая оценить продолжительность решения задачи коммивояжера в зависимости от количества городов.

// -- main

#include "stdafx.h"

#include "Auxil.h"

#include <iostream>

#include <iomanip>

#include <time.h>

#include "Salesman.h"

#define SPACE(n) std::setw(n)<<" "

#define N 12

int \_tmain(int argc, \_TCHAR\* argv[])

{

setlocale(LC\_ALL, "rus");

int d[N\*N+1], r[N];

auxil::start();

for(int i = 0; i <= N\*N; i++) d [i] = auxil::iget(10,100);

std::cout<<std::endl<<"-- Задача коммивояжера -- ";

std::cout<<std::endl<<"-- количество ------ продолжительность -- ";

std::cout<<std::endl<<" городов вычисления ";

clock\_t t1, t2;

for (int i = 7; i <= N; i++)

{

t1 = clock();

salesman (i, (int\*)d, r);

t2 = clock();

std::cout<<std::endl<<SPACE(7)<<std::setw(2)<<i

<<SPACE(15)<<std::setw(5)<<(t2-t1);

}

std::cout<<std::endl;

system("pause");

return 0;

}

Рис. 11. Вычисление продолжительности решения задачи коммивояжера при разном количестве городов

В программе применяются функции **auxil::start** и **auxil::iget**, позволяющие сгенерировать расстояния между городами случайным образом, которые берутся из первой лабораторной работы.

На рис. 12 приведен результат выполнения программы, представленной на рис. 11, а на рис. 13 изображен график, построенный по этим результатам.

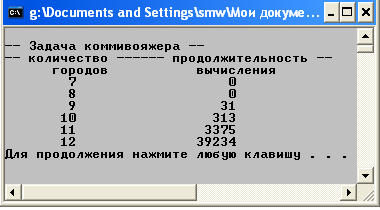


Рис. 12. Результат выполнения программы, представленной на рис. 11

**Зависимость продолжительности вычисления**

**от количества городов**

0

5

10

15

20

25

30

35

40

45

7

8

9

10

11

12

**Количество городов**

**Продолжительность**

**вычисления, с**

Рис. 13. Зависимость продолжительности вычисления функции **salesman** от значения параметра **n**

Вид графика на рис. 4.13 вполне согласуется с оценкой  сложности алгоритма генерации перестановок  элементов.

Сложность 