**Лабораторная работа 2. Комбинаторные алгоритмы решения оптимизационных задач**

**ЦЕЛЬ РАБОТЫ:** приобрести навыки разработки генераторов подмножеств, перестановок, сочетаний и размещений на С++; научиться применять разработанные генераторы для решения задач о рюкзаке (упрощенную, коммивояжера, об оптимальной загрузке судна и об оптимальной загрузке судна с центровкой.

**Задание 5.**  Решить в соответствии с вариантом задачу и результат занести в отчет (Вариант распределяется по списку):

1, 5, 9, 13) коммивояжера (расстояния сгенерировать случайным образом: 10 городов, расстояния 10 – 300 км, 3 расстояния между городами задать бесконечными);

Ниже в листингах представлены заголовочные файлы и файлы .cpp с реализацией поставленной задачи.

#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

};

};

Листинг 2.1 – Combi.h

#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); };

}

Листинг 2.2 – Combi.cpp

//-- 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

);

Листинг 2.3 – Salesman.h

// -- 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;

}

Листинг 2.4 – Salesman.cpp

#pragma once

namespace auxil {

void start();

double dget(double min, double max);

int iget(int min, int max);

}

Листинг 2.5 – Auxil.h

#include <iostream>

#include <random>

namespace auxil {

void start() {

srand((unsigned)time(NULL));

}

double dget(double min, double max) {

return ((double)rand() / (double)RAND\_MAX) \* (max - min) + min;

}

int iget(int min, int max) {

return (int)dget((double)min, (double)max);

};

Листинг 2.6 – Auxil.cpp

// -- main

#include "stdafx.h"

#include "Auxil.h"

#include <iostream>

#include <iomanip>

#include <time.h>

#include "Salesman.h"

#include <tchar.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;

}

Листинг 2.7 – Main.cpp

Зависимость времени вычисления от количества городов представлена на рисунке 2.8

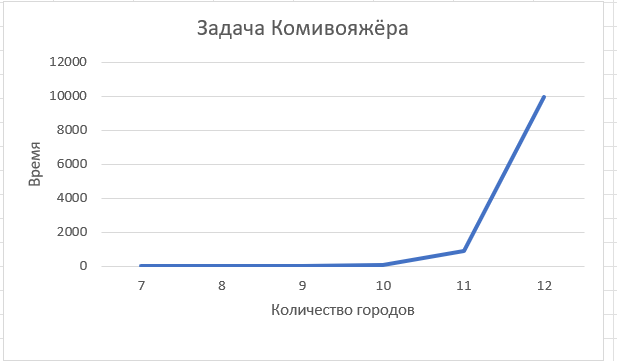


Рисунок 2.8 – Зависимость времени вычисления от количества городов

Вывод: исходя из полученных данных и графика, можно заметить, что скорость выполнения программы экспоненциально возрастает при увеличении количества городов.