**Белорусский государственный технологический университет**

**Факультет информационных технологий**

**Специальность программная инженерия**

Отчёт по лабораторной работе №2

По дисциплине «Математическое программирование»

На тему «Комбинаторные алгоритмы»

Выполнил:

Студент 2 курса 8 группы

Кулешов Артём Алексеевич

Преподаватель: асс. Ромыш А.С.

2025, Минск

**Вариант 5**

**ЦЕЛЬ РАБОТЫ:** приобрести навыки разработки генераторов подмножеств, перестановок, сочетаний и размещений на С++; научиться применять разработанные генераторы для решения задачи коммивояжера для 10 городов.

**Задание 1.** Разобрать и разработать генератор сочетаний.

Генератор сочетаний – это инструмент или алгоритм, который помогает создать все возможные комбинации элементов заданного множества определенного размера. Сочетания не зависят от порядка элементов.

Листинг 1.1 – Файл Combi.h

#pragma once

namespace combi

{

struct xcombination // генератор сочетаний (эвристика)

{

short n, // количество элементов исходного множества

m, // количество элементов в сочетаниях

\* sset; // массив индексов текущего сочетания

xcombination(

short n = 1, // количество элементов исходного множества

short m = 1 // количество элементов в сочетаниях

);

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

short getfirst(); // сформировать первый массив индексов

short getnext(); // сформировать следующий массив индексов

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

unsigned \_\_int64 nc; // номер сочетания 0,..., count()-1

unsigned \_\_int64 count() const; // вычислить количество сочетаний

};

};

#include "Combi.h"

#include <algorithm>

namespace combi

{

xcombination::xcombination(short n, short m)

{

this->n = n;

this->m = m;

this->sset = new short[m + 2];

this->reset();

}

void xcombination::reset() // сбросить генератор, начать сначала

{

this->nc = 0;

for (int i = 0; i < this->m; i++) this->sset[i] = i;

this->sset[m] = this->n;

this->sset[m + 1] = 0;

};

Листинг 1.2 – Файл Combi.cpp

#include <iostream>

#include "Combi.h"

int main() // тоже битовая маска, но берем только подмножества с N двоичными единицами

{

setlocale(LC\_ALL, "rus");

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

std::cout << std::endl << " --- Генератор сочетаний ---";

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 сочетаний ";

combi::xcombination xc(sizeof(AA) / 2, 3);

std::cout << "из " << xc.n << " по " << xc.m;

int n = xc.getfirst();

while (n >= 0)

{

std::cout << std::endl << xc.nc << ": { ";

for (int i = 0; i < n; i++)

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

std::cout << "}";

n = xc.getnext();

};

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

system("pause");

return 0;

}

void xcombination::reset() // сбросить генератор, начать сначала

{

this->nc = 0;

for (int i = 0; i < this->m; i++) this->sset[i] = i;

this->sset[m] = this->n;

this->sset[m + 1] = 0;

};

short xcombination::getfirst()

{

return (this->n >= this->m) ? this->m : -1;

};

short xcombination::getnext() // сформировать следующий массив индексов

{

short rc = getfirst();

if (rc > 0)

{

short j;

for (j = 0; this->sset[j] + 1 == this->sset[j + 1]; ++j)

this->sset[j] = j;

if (j >= this->m) rc = -1;

else {

this->sset[j]++;

this->nc++;

};

}

return rc;

};

short xcombination::ntx(short i)

{

return this->sset[i];

};

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

unsigned \_\_int64 xcombination::count() const

{

return (this->n >= this->m) ?

fact(this->n) / (fact(this->n - this->m) \* fact(this->m)) : 0;

};

}

Листинг 1.3 – Файл Main.cpp

std::cout << "}";

n = xc.getnext();

};

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

system("pause");

return 0;

}

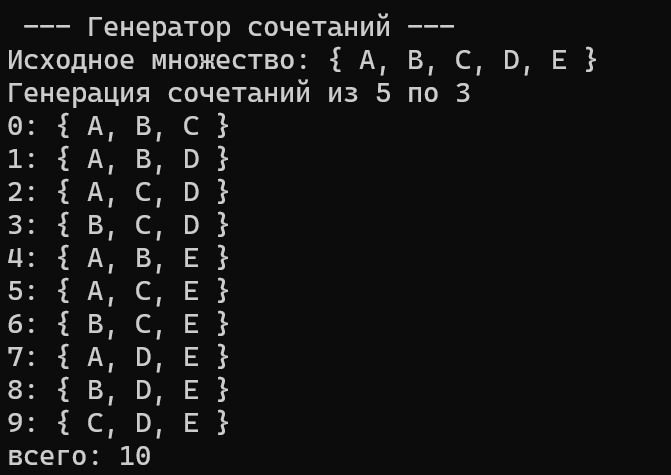


Рисунок 1 – Результат работы программы

**Задание 2.** Разобрать и разработать генератор подмножеств заданного множества.

Генератор подмножеств – это инструмент или алгоритм, который помогает создать все возможные подмножества заданного множества. Подмножество – это набор элементов, которые принадлежат исходному множеству, но не обязательно включают все элементы этого множества.

#pragma once

namespace combi

{

struct subset // генератор множества всех подмножеств

{

short n, // количество элементов исходного множества < 64

sn, // количество элементов текущего подмножества

\* sset; // массив индексов текущего подмножества

unsigned \_\_int64 mask; // битовая маска

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

short getfirst(); // сформормировать массив индексов по битовой маске

short getnext(); // ++маска и сформировать массив индексов

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

unsigned \_\_int64 count(); // вычислить общее количество подмножеств

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

};

};

Листинг 2.1 – Файл Combi.h

Листинг 2.2 – Файл Combi.cpp

#include "Combi.h"

#include <algorithm>

namespace combi

{

subset::subset(short n)

{

this->n = n;

this->sset = new short[n];

this->reset();

};

void subset::reset()

{

this->sn = 0;

this->mask = 0;

};

short subset::getfirst()

{

\_\_int64 buf = this->mask;

this->sn = 0;

for (short i = 0; i < n; i++)

{

if (buf & 0x1) this->sset[this->sn++] = i;

buf >>= 1;

}

return this->sn;

};

short subset::getnext()

{

int rc = -1;

this->sn = 0;

if (++this->mask < this->count()) rc = getfirst();

return rc;

};

short subset::ntx(short i)

{

return this->sset[i];

};

unsigned \_\_int64 subset::count()

{

return (unsigned \_\_int64)(1 << this->n);

};

};

#include <iostream>

#include "Combi.h"

int main() // с помощью инкременирования битовой маски

{

setlocale(LC\_ALL, "rus");

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

std::cout << std::endl << " - Генератор множества всех подмножеств -";

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 всех подмножеств ";

combi::subset s1(sizeof(AA) / 2); // создание генератора

int n = s1.getfirst(); // первое (пустое) подмножество

while (n >= 0) // пока есть подмножества

{

std::cout << std::endl << "{ ";

for (int i = 0; i < n; i++)

std::cout << AA[s1.ntx(i)] << ((i < n - 1) ? ", " : " ");

std::cout << "}";

n = s1.getnext(); // cледующее подмножество

};

std::cout << std::endl << "всего: " << s1.count() << std::endl;

system("pause");

return 0;

}

Листинг 2.3 – Файл Main.cpp

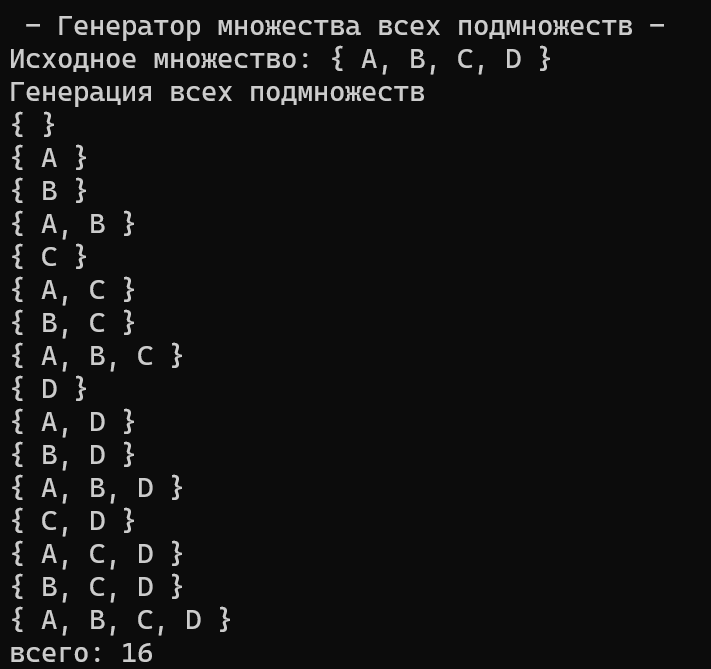


Рисунок 2 – Результат работы программы

**Задание 3.** Разобрать и разработать генератор перестановок.

Генератор перестановок – это инструмент или алгоритм, который создаёт все возможные перестановки (порядки) элементов заданного множества. Перестановка представляет собой упорядоченное расположение элементов множества.

#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 3.1 – Файл Combi.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 3.2 – Файл Combi.cpp

#include <iostream>

#include "Combi.h"

#include <iomanip>

int main() // алгоритм Джонсона – Троттера

{

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();

\_\_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 3.3 – Файл Main.cpp

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;

}

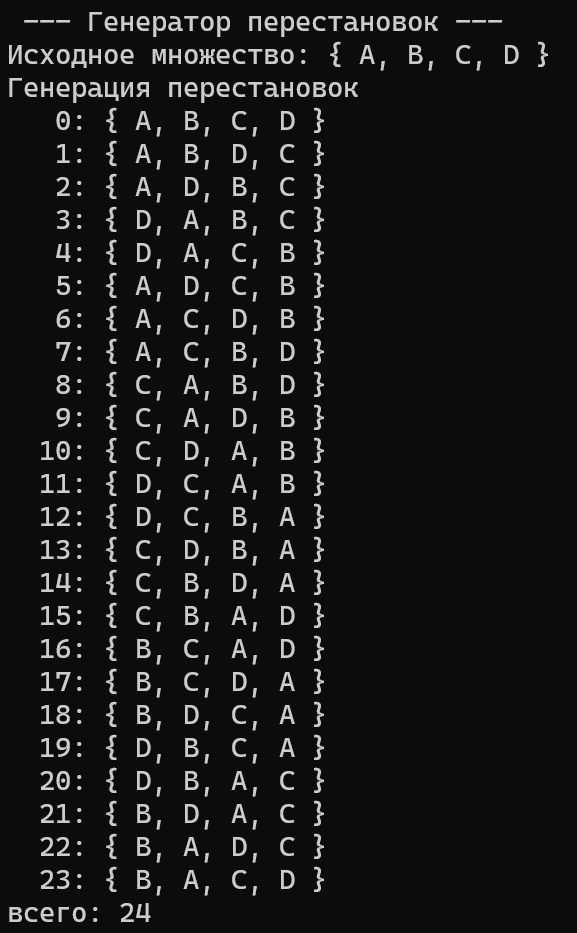


Рисунок 3 – Результат работы программы

**Задание 4.** Разобрать и разработать генератор размещений.

Генератор размещений – это инструмент или алгоритм, который помогает создать все возможные размещения элементов заданного множества определенного размера.

Размещение – это упорядоченная комбинация элементов множества, где важен порядок и количество элементов в каждом размещении. В отличие от сочетаний, размещения зависят от порядка и могут повторять элементы (если это допускается).

#pragma once

namespace combi

{

struct xcombination

{

short n,

m,

\* sset;

xcombination(

short n = 1,

short m = 1

);

void reset();

short getfirst();

short getnext();

short ntx(short i);

unsigned \_\_int64 nc;

unsigned \_\_int64 count() const;

};

struct permutation

{

const static bool L = true;

const static bool R = false;

short n,

\* sset;

bool\* dart;

permutation(short n = 1);

void reset();

\_\_int64 getfirst();

\_\_int64 getnext();

short ntx(short i);

unsigned \_\_int64 np;

unsigned \_\_int64 count() const;

};

struct accomodation

{

short n,

m,

\* sset;

xcombination\* cgen;

permutation\* pgen;

accomodation(short n = 1, short m = 1);

void reset();

short getfirst();

short getnext();

short ntx(short i);

unsigned \_\_int64 na;

unsigned \_\_int64 count() const;

};

}

Листинг 4.1 – Файл Combi.h

#include "Combi.h"

#include <algorithm>

#define NINF ((short)0x8000)

namespace combi

{

// ================================== ACCOMODATION ==================================

accomodation::accomodation(short n, short m)

{

this->n = n;

this->m = m;

this->cgen = new xcombination(n, m);

this->pgen = new permutation(m);

this->sset = new short[m];

this->reset();

}

void accomodation::reset()

{

this->na = 0;

this->cgen->reset();

this->pgen->reset();

this->cgen->getfirst();

};

short accomodation::getfirst()

{

short rc = (this->n >= this->m) ? this->m : -1;

if (rc > 0)

{

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

this->sset[i] = this->cgen->sset[this->pgen->ntx(i)];

};

return rc;

};

short accomodation::getnext()

{

short rc;

this->na++;

if ((this->pgen->getnext()) > 0) rc = this->getfirst();

else if ((rc = this->cgen->getnext()) > 0)

{

this->pgen->reset(); rc = this->getfirst();

};

return rc;

};

short accomodation::ntx(short i)

{

return this->sset[i];

};

// факториал из структуры accomodation

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

unsigned \_\_int64 accomodation::count() const

{

return (this->n >= this->m) ?

fact(this->n) / fact(this->n - this->m) : 0;

};

accomodation(short n = 1, short m = 1);

void reset();

short getfirst();

short getnext();

short ntx(short i);

unsigned \_\_int64 na;

unsigned \_\_int64 count() const;

};

}

// ================================== PERMUTATION ==================================

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)

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 permutation::count() const { return fact(this->n); };

xcombination::xcombination(short n, short m)

{

this->n = n;

this->m = m;

this->sset = new short[m + 2];

this->reset();

}

Листинг 4.2 – Файл Combi.cpp

#include <iostream>

#include <iomanip>

#include "Combi.h"

#define N (sizeof(AA)/2)

#define M 3

int main()

{

setlocale(LC\_ALL, "rus");

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

std::cout << std::endl << " --- Генератор размещений ---";

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

std::cout << "{ ";

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

std::cout << AA[i] << ((i < N - 1) ? ", " : " ");

std::cout << "}";

std::cout << std::endl << "Генерация размещений из " << N << " по " << M;

combi::accomodation s(N, M);

int n = s.getfirst();

void xcombination::reset()

{

this->nc = 0;

for (int i = 0; i < this->m; i++) this->sset[i] = i;

this->sset[m] = this->n;

this->sset[m + 1] = 0;

};

short xcombination::getfirst()

{

return (this->n >= this->m) ? this->m : -1;

};

short xcombination::getnext()

{

short rc = getfirst();

if (rc > 0)

{

short j;

for (j = 0; this->sset[j] + 1 == this->sset[j + 1]; ++j)

this->sset[j] = j;

if (j >= this->m) rc = -1;

else

{

this->sset[j]++;

this->nc++;

}

}

return rc;

};

short xcombination::ntx(short i)

{

return this->sset[i];

};

unsigned \_\_int64 xcombination::count() const

{

return (this->n >= this->m) ?

fact(this->n) / (fact(this->n - this->m) \* fact(this->m)) : 0;

};

}

Листинг 4.3 – Файл Main.cpp

while (n >= 0)

{

std::cout << std::endl << std::setw(2) << s.na << ": { ";

for (int i = 0; i < M; i++)

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

std::cout << "}";

n = s.getnext();

};

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

system("pause");

return 0;

}



Рисунок 4 – Результат работы программы

**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);

#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)

~permutation(); //деструктор

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 5.1 – Файл Combi.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;

};

Листинг 5.2 – Файл Salesman.cpp

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;

}

#include "Combi.h"

namespace combi

{

permutation::permutation(short n) : n(n), np(0)

{

sset = new short[n];

dart = new bool[n];

for (short i = 0; i < n; i++)

{

sset[i] = i;

dart[i] = L; // Все стрелки направлены влево

}

}

permutation::~permutation()

{

delete[] sset;

delete[] dart;

}

Листинг 5.3 – Файл Combi.cpp

#include <tchar.h>

#include <iostream>

#include <iomanip>

#include "Salesman.h"

#define N 5

int main(int argc, char\* 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;

}

\_\_int64 permutation::getfirst()

{

np = 0;

return np;

}

\_\_int64 permutation::getnext()

{

if (++np < count())

return np;

else

return -1; // Нет следующей перестановки

}

unsigned \_\_int64 permutation::count() const

{

unsigned \_\_int64 result = 1;

for (short i = 2; i <= n; i++)

result \*= i;

return result;

}

}

Листинг 5.4 – Файл Main.cpp

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;

}

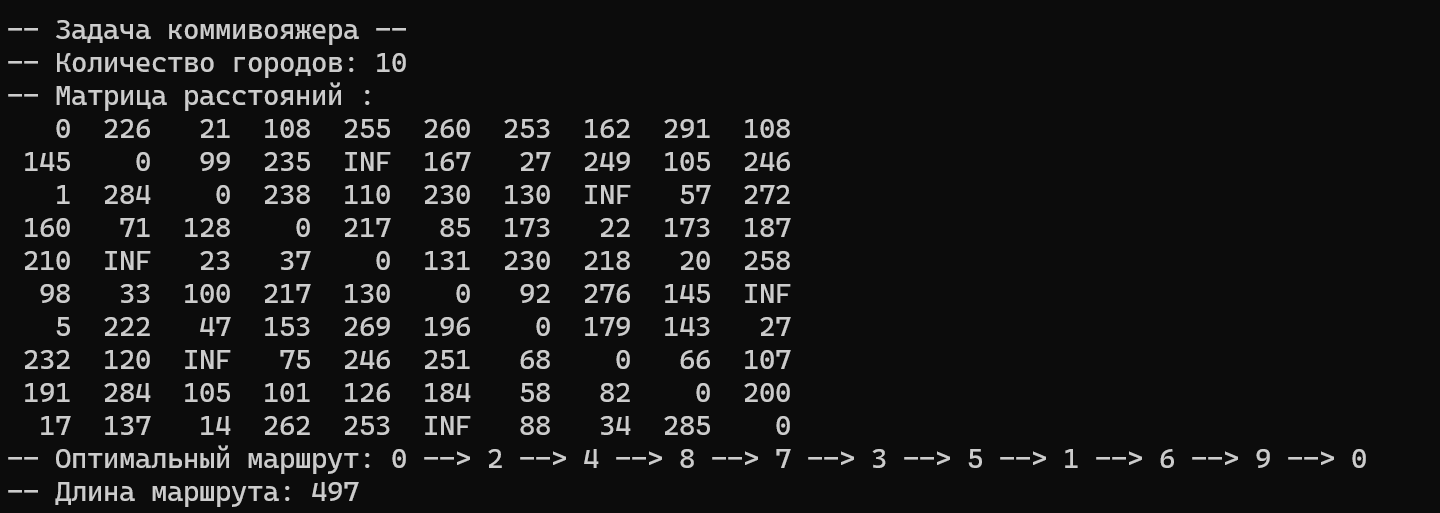


Рисунок 5 – Результат работы программы

**Задание 6.** Исследовать зависимость времени вычисления необходимое для решения задачи (в соответствии с вариантом) от размерности задачи и результат в виде графика с небольшим пояснением занести в отчет:

(1, 5, 9, 13) коммивояжера (6 – 12 городов);

#include <tchar.h>

#include <iostream>

#include <iomanip>

#include <ctime>

#include <chrono>

#include <cstdlib>

#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];

srand(time(0));

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

d[i] = rand() % 91 + 10; // (rand() % (100 - 10 + 1)) + 10

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

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

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

Листинг 6.1 – Файл Main.cpp

clock\_t t1, t2;

for (int i = 6; 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;

}

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 6 – Результат работы программы



Рисунок 7 – График зависимости времени вычисления от количества городов