МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«Белорусский государственный технологический университет»

Отчет по лабораторной работе № 2

Комбинаторные алгоритмы решения оптимизационных задач

Выполнил: Лепеш Роман Дмитриевич

Проверил: Берников Владислав Олегович

Минск 2021

**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.

**Теоретическое введение:**

1. **Генерация подмножеств заданного множества**



Рисунок 1 - Генерация множества всех подмножеств

// Combi.h

**#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

**};**

**};**

Рисунок 2 - Шаблон структуры генератора множества всех подмножеств

// Combi.cpp

**#include "stdafx.h"**

**#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);};**

**};**

Рисунок 3 - Реализация методов структуры **subset**

// Main

**#include "stdafx.h"**

**#include <iostream>**

**#include "Combi.h"**

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

**{**

**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;**

**}**

Рисунок 4 - Пример применения генератора множества всех подмножеств

1. **Генерация сочетаний**

На рисунке 14 представлена схема построения множества сочетаний  из элементов множества  Закрашенным прямоугольником на рисунке обозначены номера (индексы) элементов битовых последовательностей   и элементов множества  Стрелки связывают битовые последовательности, содержащие три двоичные единицы и сгенерированные сочетания множества  Для каждой стрелки указаны индексы единичных позиций соответствующих битовых последовательностей. Эти индексы используются для выбора элементов из множества для включения в соответствующее сочетание. Очевидно, что такой алгоритм генерации сочетаний имеет сложность  как и алгоритм генерации множества всех подмножеств.



Рисунок 14 - Схема генерации сочетаний на основе множества всех подмножеств

На рисунках 15 и 16 представлена реализация генератора сочетаний на языке С++. Генератор реализован в виде структуры **xcombination**.

// 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

**};**

**};**

Рисунок 15 - Шаблон структуры генератора сочетаний

// Combi.cpp

**#include "stdafx.h"**

**#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;**

**};**

**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;**

**};**

**};**

Рисунок 16 - Реализация функций генератора сочетаний

Продолжение рисунка 16.

// Main

**#include "stdafx.h"**

**#include <iostream>**

**#include "Combi.h"**

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

**{**

**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;**

**}**

**3. Генерация перестановок**

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



Рисунок 4.1 - Схема работы алгоритма Джонсона – Троттера

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

На рисунках 2 и 3 представлена программная реализация генератора перестановок.

// 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

**};**

**};**

Рисунок 2 - Шаблон структуры генератора перестановок

Рисунок 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;**

**}**

Рисунок 4 - Пример применения генератора перестановок

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



Рисунок 6 - Схема решения задачи коммивояжера

Расстояние между городами задается следующей матрицей 



На рисунках 7 и 8 представлен пример реализации на C++ функции **salesman**, вычисляющей оптимальный кольцевой маршрут коммивояжера.

//-- 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

**);**

Рисунок 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;**

**}**

Рисунок 8. Реализация функции **salesman**

На рисунках 9 и 10 приведен пример вызова функции **salesman** для решения задачи с исходными данными к схеме на рисунка 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;**

**}**

Рисунок 9 - Пример решения задачи коммивояжера

На рисунке 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;

}

Рисунок 11 - Вычисление продолжительности решения задачи коммивояжера при разном количестве городов

**4. Генерация размещений**

На рисунке 1 представлена схема построения множества размещений  из элементов множества 



Рисунок 1 - Схема генерации размещений

**Реализация генератора размещений на языке С++**

// Combi.h

**#pragma once**

**namespace combi**

**{**

**struct accomodation** // генератор размещений

**{**

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

**m,** // количество элементов в размещении

**\*sset;** // массив индесов текущего размещения

**xcombination \*cgen;** // указатель на генератор сочетаний

**permutation \*pgen;** // указатель на генератор перестановок

**accomodation(short n = 1, short m = 1);** // конструктор

**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 na;** // номер размещения 0, ..., count()-1

**unsigned \_\_int64 count() const;** // общее количество размещений

**};**

**}**

**};**

Рисунок 2 - Шаблон структуры генератора размещений

// Combi.cpp

**#include "stdafx.h"**

**#include "Combi.h"**

**namespace combi**

**{**

**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]; };**

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;**

**};**

**}**

**};**

Рисунок 3 - Реализация функций генератора размещений

// --- main

**#include "stdafx.h"**

**#include <iostream>**

**#include <iomanip>**

**#include "Combi.h"**

**#define N (sizeof(AA)/2)**

**#define M 3**

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

**{**

**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();**

**while (n >= 0)**

**{**

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

**for (int i = 0; i < 3; 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 - Пример использования генератора перестановок

**Ход Работы**

**Задание 1:** Разработать генератор подмножеств заданного множества.

Сложность алгоритма : 

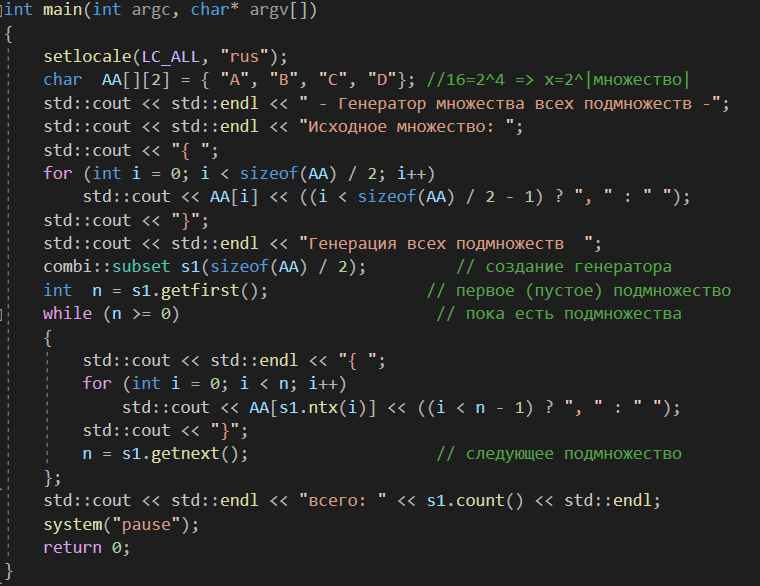


Рисунок 5 – Тестирование функции генерации подмножеств

Результат выполнения:

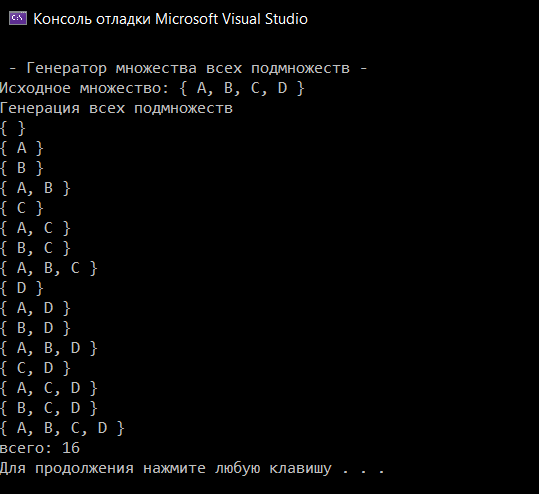


Рисунок 6 – Вывод задания 1

**Задание 2:** Разработать генератор сочетаний. Сложность алгоритма: 



Рисунок 7 – Тестирование функции генерации сочетаний

Результат выполнения:

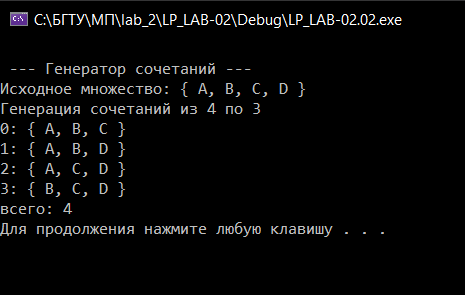


Рисунок 8 – Вывод задания 2

**Задание 3:** Разработать генератор перестановок. Сложность алгоритма: O(n!)

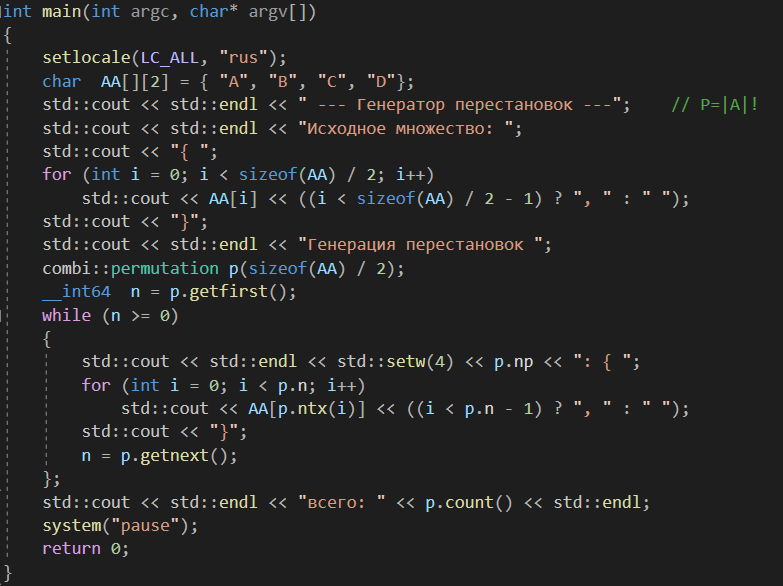


Рисунок 9 – Тестирование функции генерации перестановок

Результат выполнения:

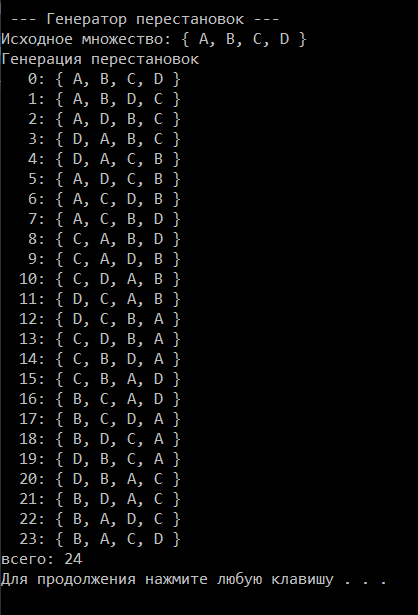


Рисунок 10 – Вывод задания 3

**Задание 4:** Разработать генератор размещений

Сложность алгоритма: O(n!/(r!(n-r)!))

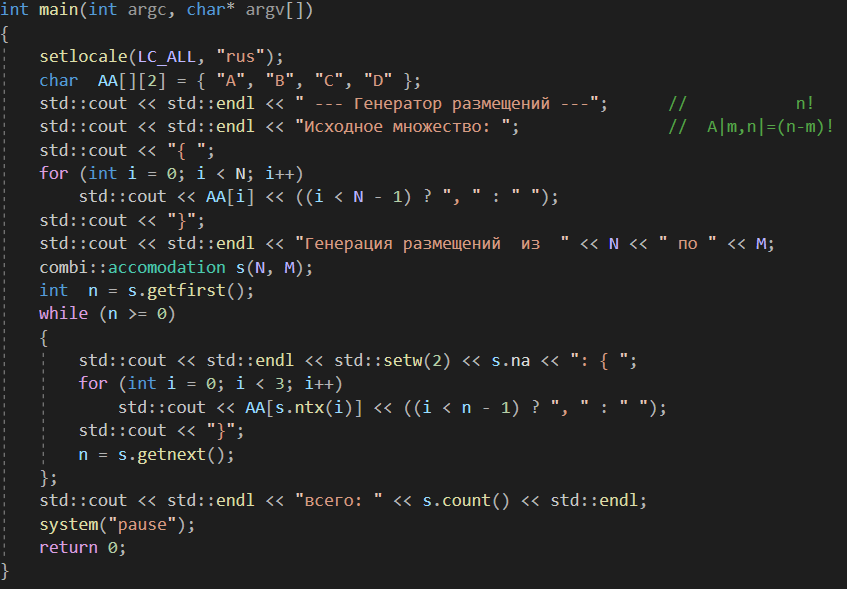


Рисунок 11 – Тестирование функции генерации размещений

Результат выполнения:

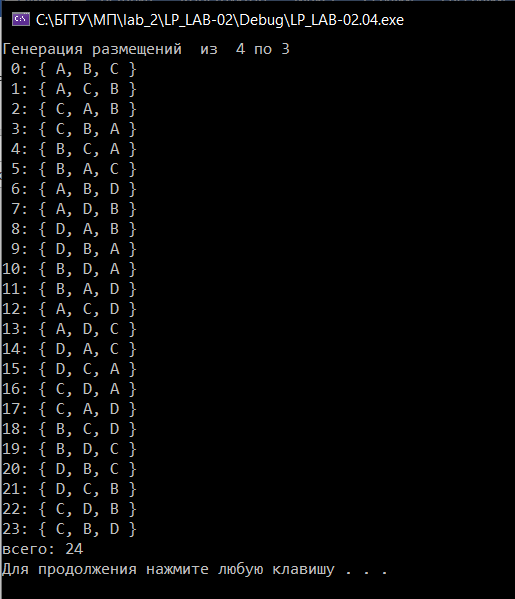


Рисунок 12 – Вывод задания 4

**Задание 5:** Решить упрощенную задачу о рюкзаке (веса предметов и их стоимость сгенерировать случайным образом: вместимость рюкзака 300 кг, веса предметов 10 – 300 кг, стоимость предметов 5 – 55 у.е.; количество предметов – 18 шт.);

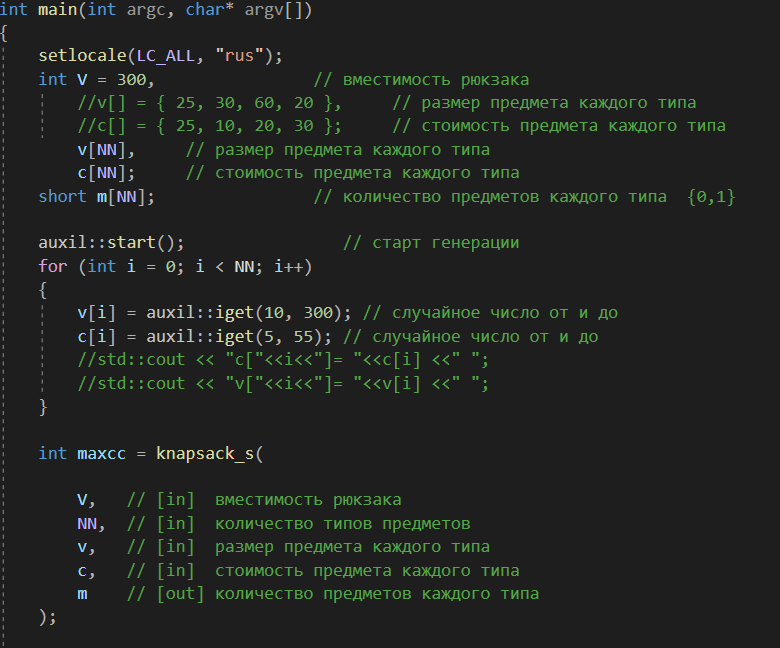


Рисунок 13.1 – Тестирование задачи о рюкзаке

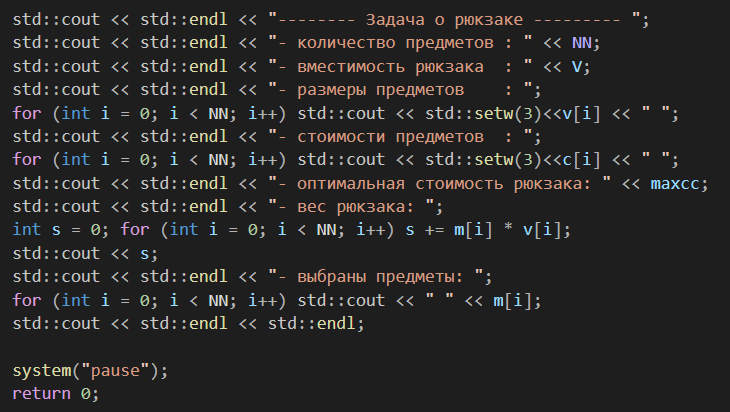


Рисунок 13.2 – Тестирование задачи о рюкзаке

Результат выполнения:

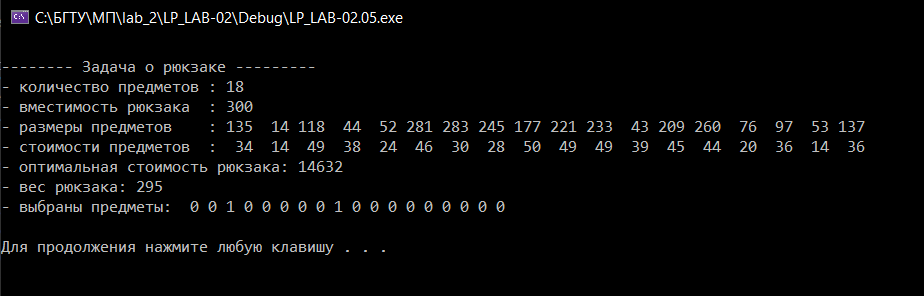


Рисунок 14 – Вывод задания 5

**Задание 6:** Исследовать зависимость времени вычисления от размерности, необходимое для решения упрощенной задачи о рюкзаке (количество предметов 12 – 20 шт.);

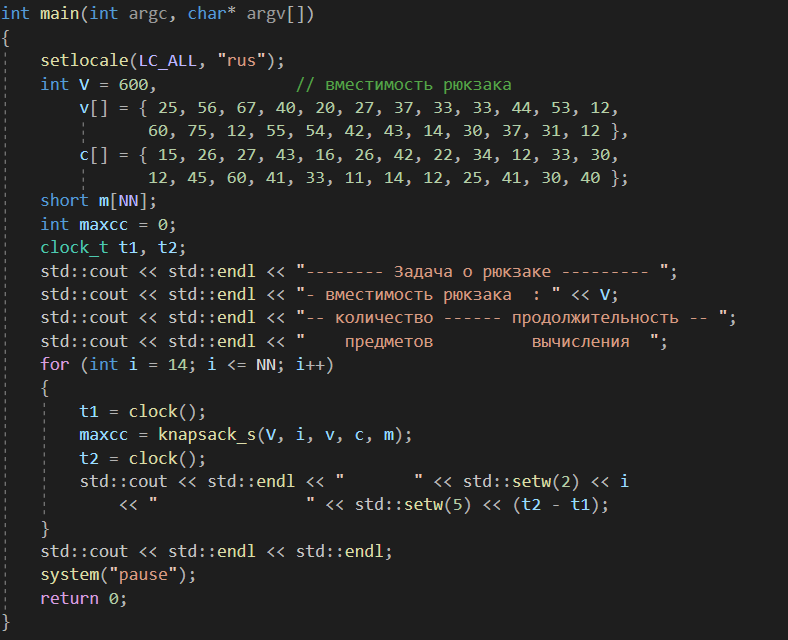


Рисунок 15 – Тестирование упрощенной задачи о рюкзаке

Результат выполнения:

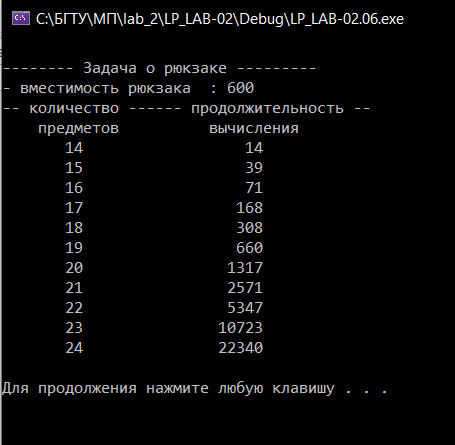


Рисунок 16 – Вывод задания 6

**Вывод**

В ходе данной лабораторной работы я изучил алгоритмы генераций подмножеств заданного множества, генерации сочетаний, генерации перестановок и генерации размещений, а также изучил алгоритм решения задачи коммивояжёра. И применил их на практике используя язык программирования С++.