МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УО «Белорусский государственный технологический университет»

Лабораторная работа №2

«Комбинаторные алгоритмы»

Выполнила студентка 6 группы 2 курса

Пунько Алина

Проверил Буснюк Николай Николаевич

Минск 2019

**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 для выполнения:**

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



Рис. 1. Генерация множества всех подмножеств

Файл 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

};

};

Файл 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);

};

};

Файл Lab2-1.cpp

#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;

}

**Задание 2.** Разработать генератор сочетаний

На рис. 2 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.



Рис.2. Схема генерации сочетаний на основе множества всех подмножеств

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

};

};

Файл Combi.cpp

#include "stdafx.h"

#include "Combi.h"

#include <algorithm>

namespace combi

{

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

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 xcombination::count() const

{

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

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

};

};

Файл Lab2-2.cpp

#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;

}

**Задание 3.** Разработать генератор перестановок

Схема алгоритма генерации множества всех перестановок множества  приведена на рис. 3.



Рис. 3. Схема работы алгоритма Джонсона – Троттера

Файл 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

};

};

Файл 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); };

}

Файл Lab2-3.cpp

#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;

}

**Задание 4.** Разработать генератор размещений

На рис. 4 представлена схема построения множества размещений  из элементов множества 



Рис.4. Схема генерации размещений

Файл 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

};

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

};

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

};

}

Файл Combi.cpp

#include "stdafx.h"

#include "Combi.h"

#include <algorithm>

#define NINF ((short)0x8000)

namespace combi

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

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 xcombination::count() const

{

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

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

};

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

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 accomodation::count() const

{

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

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

};

}

Файл Lab2-4.cpp

#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;

}

**Задание 5.**  Решить в соответствии с вариантом задачу:

2, 6, 10, 14) упрощенную 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.);

На рис. 5 изображена схема решения задачи о рюкзаке с применением генератора множества всех подмножеств.



Рис. 5. Схема решения задачи о рюкзаке с применением генератора множества всех подмножеств

Файл 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

};

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

};

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

};

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

};

}

Файл knapsack\_s.h

#pragma once

#include "Combi.h"

int knapsack\_s(

int V, // [in] вместимость рюкзака

short n, // [in] количество типов предметов

const int v[], // [in] размер предмета каждого типа

const int c[], // [in] стоимость предмета каждого типа

short m[] // [out] количество предметов каждого типа

);

Файл knapsack\_s.cpp

#include "stdafx.h"

#include "knapsack\_s.h"

#include "combi.h"

#define NINF 0x80000000 // самое малое int-число

int calcv(combi::subset s, const int v[]) // объем в рюкзаке

{

int rc = 0;

for (int i = 0; i < s.sn; i++) rc += v[s.ntx(i)];

return rc;

};

int calcc(combi::subset s, const int v[], const int c[]) //стоимость в рюкзаке

{

int rc = 0;

for (int i = 0; i < s.sn; i++) rc += (v[s.ntx(i)] \* c[s.ntx(i)]);

return rc;

};

void setm(combi::subset s, short m[]) //отметить выбранные предметы

{

for (int i = 0; i < s.n; i++) m[i] = 0;

for (int i = 0; i < s.sn; i++) m[s.ntx(i)] = 1;

};

int knapsack\_s(

int V, // [in] вместимость рюкзака

short n, // [in] количество типов предметов

const int v[], // [in] размер предмета каждого типа

const int c[], // [in] стоимость предмета каждого типа

short m[] // [out] количество предметов каждого типа {0,1}

)

{

combi::subset s(n);

int maxc = NINF, cc = 0;

short ns = s.getfirst();

while (ns >= 0)

{

if (calcv(s, v) <= V)

if ((cc = calcc(s, v, c)) > maxc)

{

maxc = cc;

setm(s, m);

}

ns = s.getnext();

};

return maxc;

};

Файл Combi.cpp

#include "stdafx.h"

#include "Combi.h"

#include <algorithm>

#define NINF ((short)0x8000)

namespace combi

{

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

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 xcombination::count() const

{

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

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

};

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

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 accomodation::count() const

{

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

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

};

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);

};

}

Файл Lab2-5.cpp

#include "stdafx.h"

#include <iostream>

#include <ctime>

#include "Combi.h"

#include "knapsack\_s.h"

#define NN 20

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

{

setlocale(LC\_ALL, "rus");

clock\_t t1 = 0, t2 = 0;

t1 = clock();

int V = 300, // вместимость рюкзака

v[NN] , // размер предмета каждого типа

c[NN] ; // стоимость предмета каждого типа

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

v[i] = rand() % 55 +5;

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

c[i] = rand() % 300 +10;

short m[NN]; // количество предметов каждого типа {0,1}

int maxcc = knapsack\_s(

V, // [in] вместимость рюкзака

NN, // [in] количество типов предметов

v, // [in] размер предмета каждого типа

c, // [in] стоимость предмета каждого типа

m // [out] количество предметов каждого типа

);

t2 = clock();

std::cout << std::endl << "-------- Задача о рюкзаке --------- ";

std::cout << std::endl << "- количество предметов : " << NN;

std::cout << std::endl << "- вместимость рюкзака : " << V;

std::cout << std::endl << "- размеры предметов : ";

for (int i = 0; i < NN; i++) std::cout << v[i] << " ";

std::cout << std::endl << "- стоимости предметов : ";

for (int i = 0; i < NN; i++) std::cout << v[i] \* c[i] << " ";

std::cout << std::endl << "- оптимальная стоимость рюкзака: " << maxcc;

std::cout << std::endl << "- вес рюкзака: ";

int s = 0; for (int i = 0; i < NN; i++) s += m[i] \* v[i];

std::cout << s;

std::cout << std::endl << "- выбраны предметы: ";

for (int i = 0; i < NN; i++) std::cout << " " << m[i];

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

std::cout << std::endl << "продолжительность (у.е): " << (t2 - t1);

std::cout << std::endl << " (сек): "

<< ((double)(t2 - t1)) / ((double)CLOCKS\_PER\_SEC);

system("pause");

return 0;

}

**Задание 6.** Исследовать зависимость времени вычисления необходимое для решения задачи (в соответствии с вариантом) от размерности задачи: 2, 6, 10, 14) упрощенную о рюкзаке (количество предметов 12 – 20 шт.);

