Учреждение образования

«Белорусский государственный технологический университет»

**Лабораторная работа №2**

**Комбинаторные алгоритмы решения оптимизационных задач**

Выполнила:

Студентка 2 курса 1 группы ФИТ

Самсоник Анастасия Ивановна

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

**Подмножество**

В терминологии теории множеств говорят, что множество B является подмножеством множества A, если каждый элемент B является в то же время и элементом множества A. Обозначается это знаком включения: B ⊂ A.

**Перестановки**

Число объектов остается неизменными, меняется только их порядок

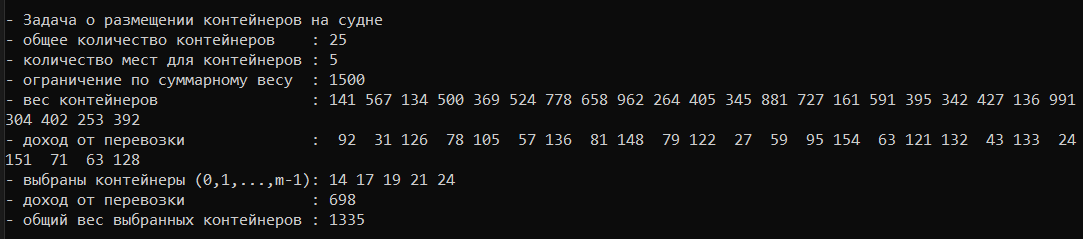
**Размещения**

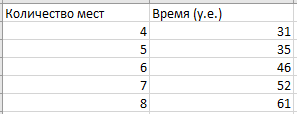
Меняется и состав выбранных объектов, и их порядок

**Сочетания**

Меняется состав выбранных объектов, но порядок не важен

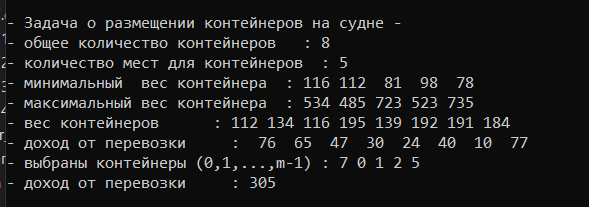
*Результат выполнения задачи о судне без центровки:*

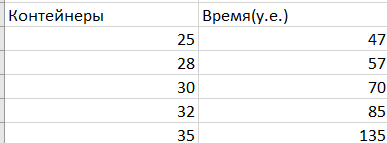




Зависимость является линейной

*Результат выполнения задачи о судне с центровкой:*





**Листинги**

Combi1.cpp

#include "Combi1.h"

namespace combi1

{

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);

};

};

Combi2.cpp

#include "Combi2.h"

namespace combi2

{

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;

};

};

Combi3.cpp

#include "Combi3.h"

#include <algorithm>

#define NINF ((short)0x8000)

namespace combi3

{

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); };

};

Combi4.cpp

#include "Combi4.h"

#include <algorithm>

#include <ctime>

#define NINF ((short)0x8000)

namespace combi4

{

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;

};

}

*Судно без центровки:*

Main.cpp:

#include <iostream>

#include <iomanip>

#include "Header.h"

#define NN (sizeof(v)/sizeof(int))

#define MM 6

#define K 35

int main()

{

setlocale(LC\_ALL, "rus");

clock\_t t1 = 0, t2 = 0;

srand(unsigned(time(NULL)));

int V = 1500,

v[K],

c[K];

short r[MM];

t1 = clock();

for (int i = 0; i < K; i++) {

v[i] = rand() % 900 + 100;

}

for (int i = 0; i < K; i++) {

c[i] = rand() % 150 + 10;

}

int cc = boat(

V, // [in] максимальный вес груза

MM, // [in] количество мест для контейнеров

NN, // [in] всего контейнеров

v, // [in] вес каждого контейнера

c, // [in] доход от перевозки каждого контейнера

r // [out] результат: индексы выбранных контейнеров

);

std::cout << std::endl << "- Задача о размещении контейнеров на судне";

std::cout << std::endl << "- общее количество контейнеров : " << NN;

std::cout << std::endl << "- количество мест для контейнеров : " << MM;

std::cout << std::endl << "- ограничение по суммарному весу : " << V;

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

for (int i = 0; i < NN; i++) std::cout << std::setw(3) << v[i] << " ";

std::cout << std::endl << "- доход от перевозки : ";

for (int i = 0; i < NN; i++) std::cout << std::setw(3) << c[i] << " ";

std::cout << std::endl << "- выбраны контейнеры (0,1,...,m-1): ";

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

std::cout << std::endl << "- доход от перевозки : " << cc;

std::cout << std::endl << "- общий вес выбранных контейнеров : ";

int s = 0; for (int i = 0; i < MM; i++) s += v[r[i]]; std::cout << s;

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

t2 = clock();

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

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

system("pause");

return 0;

}

Boat.cpp:

#include "Header.h"

namespace boatfnc

{

int calcv(combi::xcombination s, const int v[]) // вес

{

int rc = 0;

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

return rc;

};

int calcc(combi::xcombination s, const int c[]) // доход

{

int rc = 0;

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

return rc;

};

void copycomb(short m, short\* r1, const short\* r2) // копировать

{

for (int i = 0; i < m; i++) r1[i] = r2[i];

};

}

int boat(

int V, // [in] максимальный вес груза

short m, // [in] количество мест для контейнеров

short n, // [in] всего контейнеров

const int v[], // [in] вес каждого контейнера

const int c[], // [in] доход от перевозки каждого контейнера

short r[] // [out] результат: индексы выбранных контейнеров

)

{

combi::xcombination xc(n, m);

int rc = 0, i = xc.getfirst(), cc = 0;

while (i > 0)

{

if (boatfnc::calcv(xc, v) <= V)

if ((cc = boatfnc::calcc(xc, c)) > rc)

{

rc = cc; boatfnc::copycomb(m, r, xc.sset);

}

i = xc.getnext();

};

return rc;

};

Source.cpp:

#include "Header1.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;

};

};

boat.h:

#pragma once

// -- решение задачи об оптимальной загрузке судна

// функция возвращает доход от перевози выбранных контейнеров

#pragma once

#include "Combi.h"

int boat(

int V, // [in] максимальный вес груза

short m, // [in] количество мест для контейнеров

short n, // [in] всего контейнеров

const int v[], // [in] вес каждого контейнера

const int c[], // [in] доход от перевозки каждого контейнера

short r[] // [out] результат: индексы выбранных контейнеров

);

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

};

};

*Судно с центровкой:*

// --- Вoat.cpp

#include "Boat.h"

#include "Combi.h"

namespace boatfnc

{

bool compv(combi::accomodation s, const int ming[],

const int maxg[], const int v[])

{

int i = 0;

while (i < s.m && v[s.ntx(i)] <= maxg[i] && v[s.ntx(i)] >= ming[i])i++;

return (i == s.m);

};

int calcc(combi::accomodation s, const int c[])

{

int rc = 0;

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

return rc;

};

void copycomb(short m, short\* r1, const short\* r2)

{

for (int i = 0; i < m; i++) r1[i] = r2[i];

};

}

int boat\_с( // функция возвращает доход от перевозки контейнеров

short m, // [in] количество мест для контейнеров

int minv[], // [in] минимальный вес контейнера на каждом месте

int maxv[], // [in] максимальный вес коннтейнера каждом месте

short n, // [in] всего контейнеров

const int v[],// [in] вес каждого контейнера

const int c[],// [in] доход от перевозки каждого контейнера

short r[] // [out] номера выбранных контейнеров

)

{

combi::accomodation s(n, m);

int rc = 0, i = s.getfirst(), cc = 0;

while (i > 0)

{

if (boatfnc::compv(s, minv, maxv, v))

if ((cc = boatfnc::calcc(s, c)) > rc)

{

rc = cc; boatfnc::copycomb(m, r, s.sset);

}

i = s.getnext();

};

return rc;

};

void boat()

{

setlocale(LC\_ALL, "rus");

int v[8]; // вес

int c[8] = {}; // доход

int minv[MM] = {}; // минимальный вес

int maxv[MM] = {}; // максимальный вес

short r[MM];

clock\_t t1 = 0, t2 = 0;

int rmin = 100;

int rmax = 200;

int min\_min\_weight = 50;

int min\_max\_weight = 120;

int max\_min\_weight = 150;

int max\_max\_weight = 850;

int min\_doxod = 10;

int max\_doxod = 100;

int rand\_weight;

srand(unsigned(time(NULL)));

for ( int i = 0; i < 8; i++)

{

rand\_weight = (int)auxil::iget(rmin, rmax);

v[i] = rand\_weight;

rand\_weight = (int)auxil::iget(min\_doxod, max\_doxod);

c[i] = rand\_weight;

}

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

{

rand\_weight = (int)auxil::iget(min\_min\_weight, min\_max\_weight);

minv[i] = rand\_weight;

rand\_weight = (int)auxil::iget(max\_min\_weight, max\_max\_weight);

maxv[i] = rand\_weight;

}

int cc = boat\_с(

MM, // [in] количество мест для контейнеров

minv, // [in] максимальный вес контейнера на каждом месте

maxv, // [in] минимальный вес контейнера на каждом месте

NN, // [in] всего контейнеров

v, // [in] вес каждого контейнера

c, // [in] доход от перевозки каждого контейнера

r // [out] номера выбранных контейнеров

);

t1 = clock();

std::cout << std::endl << "- Задача о размещении контейнеров на судне -";

std::cout << std::endl << "- общее количество контейнеров : " << NN;

std::cout << std::endl << "- количество мест для контейнеров : " << MM;

std::cout << std::endl << "- минимальный вес контейнера : ";

for (int i = 0; i < MM; i++) std::cout << std::setw(3) << minv[i] << " ";

std::cout << std::endl << "- максимальный вес контейнера : ";

for (int i = 0; i < MM; i++) std::cout << std::setw(3) << maxv[i] << " ";

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

for (int i = 0; i < NN; i++) std::cout << std::setw(3) << v[i] << " ";

std::cout << std::endl << "- доход от перевозки : ";

for (int i = 0; i < NN; i++) std::cout << std::setw(3) << c[i] << " ";

std::cout << std::endl << "- выбраны контейнеры (0,1,...,m-1) : ";

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

std::cout << std::endl << "- доход от перевозки : " << cc;

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

t2 = clock();

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

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

system("pause");

}

**Вывод**: приобретены навыки разработки генераторов подмножеств, перестановок, сочетаний и размещений на С++; получены навыки применения разработанных генераторов для решения конкретных задач (об оптимальной загрузке судна);