Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа № 2

По дисциплине «Математическое программирование»

На тему «Комбинаторные алгоритмы решения оптимизационных задач»

Выполнила:

Студентка 2 курса 10 группы

Рублевская Маргарита Владимировна

Преподаватель: асс. Ромыш А.С.

2025, Минск

**Вариант 10**

**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.

**Ход работы**

## **Написание кода генератора подмножеств заданного множества**

Генератор подмножеств заданного множества – это вывод всех возможных комбинаций элементов множества. Код файла Combi.h приведен в листинге 1.1.

#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 1.1 — Содержание файла Combi.h

Код файла Combi.cpp приведен в листинге 1.2.

#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);

};

};

Листинг 1.2 — Содержание файла Combi.cpp

1. **Написание кода генератора сочетаний**

Сочетания выбор элементов из множества, в котором порядок не важен. Код файла Combi.h приведен в листинге 2.1.

#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

};

};

Листинг 2.1 — Содержание файла Combi.h

Код файла Combi.cpp приведен в листинге 2.2.

#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;

};

};

Листинг 2.2 — Содержание файла Combi.cpp

1. **Написание кода генератора перестановок**

Код файла Combi.h приведен в листинге 3.1.

#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 — Содержание файла Combi.h

Код файла Combi.cpp приведен в листинге 3.2.

#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 — Содержание файла Combi.cpp

1. **Написание кода генератора размещений**

Код файла Combi.h приведен в листинге 4.1.

#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

};

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 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

};

}

Листинг 4.1 — Содержание файла Combi.h

Код файла Combi.cpp приведен в листинге 4.2.

#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;

};

// ================ 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)

{

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::xcombination(short n, short m)

{

this->n = n;

this->m = m;

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

this->reset();

}

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 4.2 — Содержание файла Combi.cpp

1. **Решение задачи упрощенную о рюкзаке и исследование зависимости времени вычисления необходимого для решения задачи от размерности задачи**

Код файла Combi.h приведен в листинге 5.1.

#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 5.1 — Содержание файла Combi.h

Код файла Combi.cpp приведен в листинге 5.2.

#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);

};

};

Листинг 5.2 — Содержание файла Combi.cpp

Код файла Knapsack.h приведен в листинге 5.3.

#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

);

Листинг 5.3 — Содержание файла Knapsack.h

Код файла Knapsack.cpp приведен в листинге 5.4.

#include "Knapsack.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;

};

Листинг 5.4 — Содержание файла Knapsack.cpp

Код файла main.cpp приведен в листинге 5.5.

#include <iostream>

#include "Combi.h"

#include "Knapsack.h"

#include <ctime>

#define NN 18

int main()

{

setlocale(LC\_ALL, "rus");

int V = 300, // вместимость рюкзака

v[] = { 25, 30, 60, 20, 21, 14, 36, 85, 45, 35, 74, 58, 67, 25, 19, 46, 36, 89}, // размер предмета каждого типа

c[] = { 25, 10, 20, 30, 5, 15, 55, 36, 47, 50, 32, 52, 12, 11, 30, 40, 7, 21}; // стоимость предмета каждого типа

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

clock\_t t1 = 0, t2 = 0;

t1 = clock();

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

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);

std::cout << std::endl;

system("pause");

return 0;

}

Листинг 5.5 — Содержание файла main.cpp

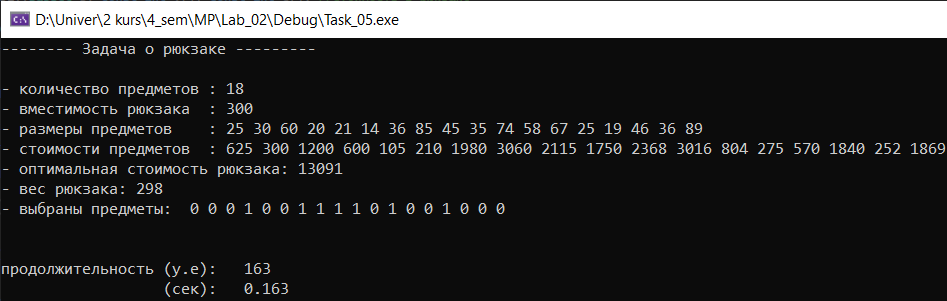
Результат работы программы представлен на рисунке 5.1. 

Рисунок 5.1. Результат работы программы

Исследование зависимости представлено на рисунке 5.2.



Рисунок 5.2. Исследование зависимости

