**Выполнил**

**Слесарев И.В.**

**2 курс 5-2**

**Лабораторная работа 2. Комбинаторные алгоритмы решения оптимизационных задач**

**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**

1. **Генератор 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

**};**

**};**

Листинг 1.1 – файл Combi.h 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);};**

**};**

Листинг 1.2 – Реализация методов для генератора множества всех подмножеств

**2.Генератор сочетаний**

// 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

**};**

**};**

Листинг 2.1 – Струкутура 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;**

**};**

**};**

Листинг 2.2 – Реализация методов для генератора сочетаний

**3.Генерация 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

**};**

**};**

Листинг 3.1 – структура генератора перестановок

Листинг 3.2 – Реализация методов 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); };**

**}**

**4.Генератор размещений**

// 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

**};**

**}**

**};**

Листинг 4.1 – Структура 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;**

**};**

**}**

**};**

Листинг 4.2 – Реализация методов генератора размещений

**5. Решение задачи об оптимальной загрузке судна**

Данная задача была решена с помощью генератора размещений.

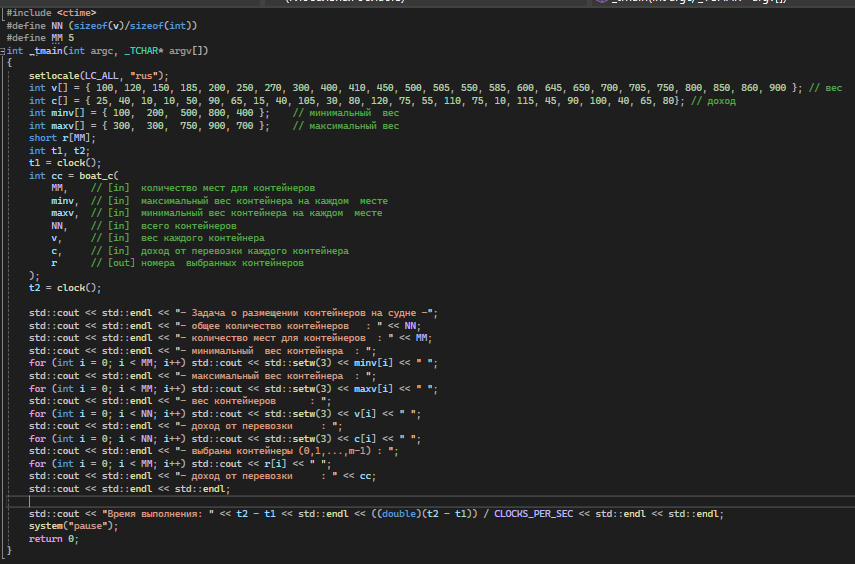


Рисунок 5.1 – Файл Main.cpp решения задачи об оптимальной загрузке

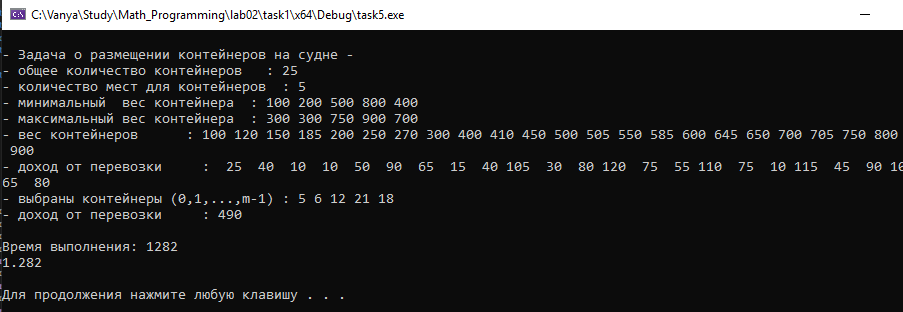


Рисунок 5.2 – Результат работы программы

**6.** **Исследование зависимости времени вычисления необходимое для решения задачи**

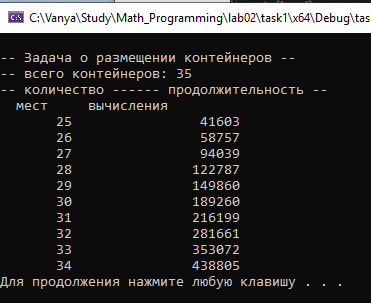
****

Рисунок 6.1 – Результат выполнения программы

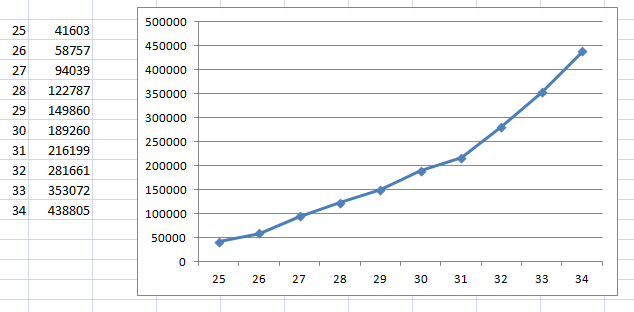


Рисунок 6.2 – График зависимости времени выполнения алгоритма от количества входных данных

Вывод: скорость выполнения программы нелинейно зависит от количества контейнеров.