Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Отчет по лабораторной работе №2**

«Комбинаторные алгоритмы решения оптимизационных задач»

Студентка: Агапкина Д.С.

ФИТ 2курс 6 группа

Преподаватель: Буснюк Н.Н.

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

// 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

**};**

**};**

Рис. 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);};**

**};**

Рис. 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;**

**}**

Рис. 4. Пример применения генератора множества всех подмножеств

**Задание 2.** Разработать генератор сочетаний

На рис. 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.



Рис.14. Схема генерации сочетаний на основе множества всех подмножеств

На рис. 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

**};**

**};**

Рис. 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;**

**};**

**};**

Рис. 16. Реализация функций генератора сочетаний

// 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;**

**}**

**Задание 3.** Разработать генератор перестановок

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



Рис. 4.1. Схема работы алгоритма Джонсона – Троттера

На рис. 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

**};**

**};**

Рис. 2. Шаблон структуры генератора перестановок

Рис. 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;**

**}**

Рис. 4. Пример применения генератора перестановок

**Задание 4.** Разработать генератор размещений

На рис. 1 представлена схема построения множества размещений  из элементов множества 



Рис.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

**};**

**}**

**};**

Рис.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;**

**};**

**}**

**};**

Рис. 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;**

**}**

Рис.4. Пример использования генератора перестановок

**Задание 5.** Решить 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. 7 и 8 представлен пример реализации функции **knapsack\_s** на языке C++, которая решает задачу о рюкзаке.

// Knapsack.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

**);**

Рис.7. Прототип функции **knapsack\_s**

// Knapsack.cpp

**#include "stdafx.h"**

**#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;**

**};**

Рис.8. Реализация функции **knapsack\_s**

На рис. 9 приведен пример вызова функции **knapsack\_s** для решения задачи о рюкзаке с исходными данными для схемы, представленной на рис. 6.

// Main

**#include "stdafx.h"**

**#include <iostream>**

**#include "Combi.h"**

**#include "Knapsack.h"**

**#define NN 4**

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

**{**

**setlocale(LC\_ALL, "rus");**

**int V = 100,** // вместимость рюкзака

**v[] = {25, 30, 60, 20},** // размер предмета каждого типа

**c[] = {25, 10, 20, 30};** // стоимость предмета каждого типа

**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

**);**

**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;**

**system("pause");**

**return 0;**

**}**

Рис. 9. Пример использования функции **knapsack\_s**

Оценить зависимость продолжительности вычисления оптимальной комбинации предметов от их общего количества можно с помощью программы, изображенной на рис. 11.

// Main

**#include "stdafx.h"**

**#include <iostream>**

**#include "Combi.h"**

**#include "Knapsack.h"**

**#include <time.h>**

**#include <iomanip>**

**#define NN (sizeof(c)/sizeof(int))**

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

**{**

**setlocale(LC\_ALL, "rus");**

**int V = 600,** // вместимость рюкзака

**v[] = {25, 56, 67, 40, 20, 27, 37, 33, 33, 44, 53, 12,**

**60, 75, 12, 55, 54, 42, 43, 14, 30, 37, 31, 12},**

**c[] = {15, 26, 27, 43, 16, 26, 42, 22, 34, 12, 33, 30,**

**12, 45, 60, 41, 33, 11, 14, 12, 25, 41, 30, 40};**

**short m[NN];**

**int maxcc = 0;**

**clock\_t t1, t2;**

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

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

**std::cout<<std::endl<<"-- количество ------ продолжительность -- ";**

**std::cout<<std::endl<<" предметов вычисления ";**

**for (int i = 14; i <= NN; i++)**

**{**

**t1 = clock();**

**maxcc = knapsack\_s(V, i, v, c, m );**

**t2 = clock();**

**std::cout<<std::endl<<" "<<std::setw(2)<<i**

**<<" "<<std::setw(5)<<(t2-t1);**

**}**

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

**system("pause");**

**return 0;**

**}**

Рис. 11. Вычисление продолжительности решения задачи о рюкзаке при различном количестве предметов