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:

Студент 2 курса 9 группы

Хуторцов Кирилл Владимирович

Преподаватель: Ромыш А.С.

2025, Минск

**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**

**Задание 1.**Разобрать и разработать генератор подмножеств заданного множества. Реализация функций показана в листинге 1, 2.

Листинг 1 – Прототипы функций Combi.h

|  |
| --- |
| // 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  };  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 2 – Код файла Combi.cpp

|  |
| --- |
| // 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)); };    short permutation::ntx(short i) { return this->sset[i]; };    unsigned \_\_int64 permutation::count() const { return fact(this->n); };  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);  };  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;  };  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;  };  }; |

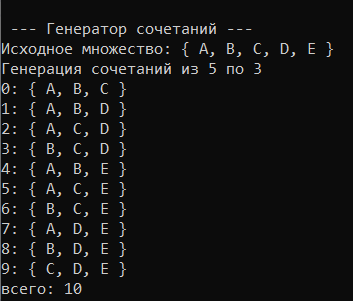


Рисунок 1.1 Результаты работы программы

**Задание 2.** Разобрать и разработать генератор сочетаний.

Задание под номером 2 показано в листинге 1,2.

**Задание 3.** Разобрать и разработать генератор перестановок.

Задание под номером **3** показано в листинге 1,2.

**Задание 4.** Разобрать и разработать генератор размещений.

Задание под номером **4** показано в листинге 1,2.

**Задание 5.** Решить в соответствии с вариантом задачу и результат занести в отчет(задача коммивояжёра)

Решить задачу и результат занести в отчет.

Задача коммивояжера (расстояния сгенерировать случайным образом: 10 городов, расстояния 10 – 300 км, 3 расстояния между городами задать бесконечными);

Задание под номером 5 показано в листинге 3.

Листинг 3 – Код файла Combinatorial\_algorithms.cpp

|  |
| --- |
| // Main  //#include "stdafx.h"  #include <iostream>  #include "Combi.h"  #include <time.h>  #include <iomanip>  #include "Auxil.h"  #include "Salesman.h"  // Определите макросы для включения/выключения заданий  #define TASK\_SUBSET\_GENERATOR 0  #define TASK\_COMBINATION\_GENERATOR 1  #define PERMUTATION 0  #define ACCOMODATION 0  #define SALESMAN 1  #define SALESMAN\_TIME 1  #define NN (sizeof(c)/sizeof(int))  #define MM 3  #define SPACE(n) std::setw(n)<<" "  #define N 13  #define R 4  #define F 3  int main()  {  setlocale(LC\_ALL, "rus");  #if TASK\_SUBSET\_GENERATOR  // Генератор множества подмножеств  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;  #endif  #if TASK\_COMBINATION\_GENERATOR  // Генератор сочетаний  char AAComb[][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(AAComb) / 2; i++)  std::cout << AAComb[i] << ((i < sizeof(AAComb) / 2 - 1) ? ", " : " ");  std::cout << "}";  std::cout << std::endl << "Генерация сочетаний ";  combi::xcombination xc(sizeof(AAComb) / 2, 3);  std::cout << "из " << xc.n << " по " << xc.m;  int nComb = xc.getfirst();  while (nComb >= 0)  {  std::cout << std::endl << xc.nc << ": { ";  for (int i = 0; i < nComb; i++)  std::cout << AAComb[xc.ntx(i)] << ((i < nComb - 1) ? ", " : " ");  std::cout << "}";  nComb = xc.getnext();  };  std::cout << std::endl << "всего: " << xc.count() << std::endl;  #endif  #if PERMUTATION  char AAPermut[][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(AAPermut) / 2; i++)  std::cout << AAPermut[i] << ((i < sizeof(AAPermut) / 2 - 1) ? ", " : " ");  std::cout << "}";  std::cout << std::endl << "Генерация перестановок ";  combi::permutation p(sizeof(AAPermut) / 2);  \_\_int64 nPermut = p.getfirst();  while (nPermut >= 0)  {  std::cout << std::endl << std::setw(4) << p.np << ": { ";  for (int i = 0; i < p.n; i++)  std::cout << AAPermut[p.ntx(i)] << ((i < p.n - 1) ? ", " : " ");  std::cout << "}";  nPermut = p.getnext();  };  std::cout << std::endl << "всего: " << p.count() << std::endl;  #endif  #if ACCOMODATION  char AAAcc[][2] = { "A", "B", "C", "D" };  std::cout << std::endl << " --- Генератор размещений ---";  std::cout << std::endl << "Исходное множество: ";  std::cout << "{ ";  for (int i = 0; i < R; i++)  std::cout << AAAcc[i] << ((i < R - 1) ? ", " : " ");  std::cout << "}";  std::cout << std::endl << "Генерация размещений из " << R << " по " << F;  combi::accomodation acc(R, F);  int nAcc = acc.getfirst();  while (nAcc >= 0)  {  std::cout << std::endl << std::setw(2) << acc.na << ": { ";  for (int i = 0; i < 3; i++)  std::cout << AAAcc[acc.ntx(i)] << ((i < nAcc - 1) ? ", " : " ");  std::cout << "}";  nAcc = acc.getnext();  };  std::cout << std::endl << "всего: " << acc.count() << std::endl;  #endif  #if SALESMAN  int d[N][N] = { //0 1 2 3 4  { 0, 45, INF, 25, 50}, // 0  { 45, 0, 55, 20, 100}, // 1  { 70, 20, 0, 10, 30}, // 2  { 80, 10, 40, 0, 10}, // 3  { 30, 50, 20, 10, 0} }; // 4  int r[N]; // результат  int s = salesman(  N, // [in] количество городов  (int\*)d, // [in] массив [n\*n] расстояний  r // [out] массив [n] маршрут 0 x x x x  );  std::cout << std::endl << "-- Задача коммивояжера -- ";  std::cout << std::endl << "-- количество городов: " << N;  std::cout << std::endl << "-- матрица расстояний : ";  for (int i = 0; i < N; i++)  {  std::cout << std::endl;  for (int j = 0; j < N; j++)  if (d[i][j] != INF) std::cout << std::setw(3) << d[i][j] << " ";  else std::cout << std::setw(3) << "INF" << " ";  }  std::cout << std::endl << "-- оптимальный маршрут: ";  for (int i = 0; i < N; i++) std::cout << r[i] << "-->"; std::cout << 0;  std::cout << std::endl << "-- длина маршрута : " << s;  std::cout << std::endl;  #endif  #if SALESMAN\_TIME  int dSalesmanTime[N \* N + 1], rSalesmanTime[N];  auxil::start();  for (int i = 0; i <= N \* N; i++) {  dSalesmanTime[i] = auxil::iget(10, 100);  }  std::cout << std::endl << "-- Задача коммивояжера -- ";  std::cout << std::endl << "-- количество ------ продолжительность -- ";  std::cout << std::endl << " городов вычисления ";  clock\_t t1, t2;  for (int i = 7; i <= N; i++)  {  t1 = clock();  salesman(i, (int\*)dSalesmanTime, rSalesmanTime);  t2 = clock();  std::cout << std::endl << SPACE(7) << std::setw(2) << i  << SPACE(15) << std::setw(5) << (t2 - t1);  }  std::cout << std::endl;  #endif  system("pause");  return 0;  } |

Результат работы программы представлен на рисунке 1.1.

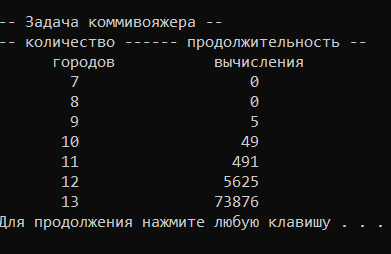


Рисунок 1.2 Результаты работы программы

**Задание 6.**

Исследовать зависимость времени вычисления необходимое для решения задачи коммивояжера от размерности задачи и результат в виде графика с небольшим пояснением занести в отчет:

Измерение скорости выполнения функции генерации случайных чисел. Результаты измерений и соответствующий график приведены на рисунке 1.2.

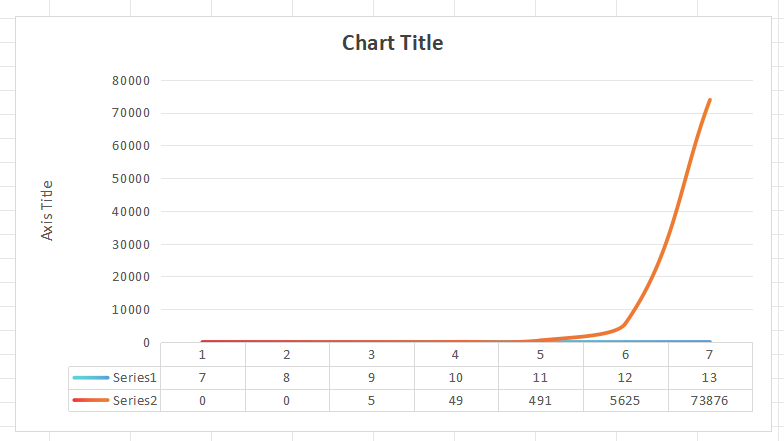


Рисунок 1.2 Результаты измерений и их график

**Вывод**: скорость выполнения программы не линейно зависит от количества итераций цикла. Похоже на факториал.

**Вывод:** Скорость выполнения программы возрастает экспоненциально с увеличением порядкового номера числа Фибоначчи. Такая зависимость обусловлена особенностями алгоритма вычисления последовательности Фибоначчи.

Вывод: Разработаны генераторы подмножеств, сочетаний, перестановок и размещений. Реализовано решение задачи коммивояжера для 13 городов с учетом случайных расстояний и бесконечных маршрутов. Проведено исследование зависимости времени вычисления от размерности задачи. Установлено, что время выполнения задачи коммивояжера растет экспоненциально с увеличением количества городов, что подтверждает NP-сложность задачи. Приобретены навыки работы с комбинаторными алгоритмами и их применения для решения задач оптимизации.