Учреждение образования

«Белорусский государственный технологический университет»

**Лабораторная работа №2**

**Комбинаторные алгоритмы**

Выполнил:

Студент 2 курса 1 группы ФИТ

Натынчик Семен Сергеевич

**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 1.** Разработать генератор подмножеств заданного множества.

Combi1.h – объявление структуры подмножества

|  |
| --- |
| #pragma once  namespace combi1  {  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(кол-во эл-ов исх. мн-ва)  short getfirst(); // сформ. 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 кол-во подмножеств  void reset(); // сбросить генератор, начать сначала  };  }; |

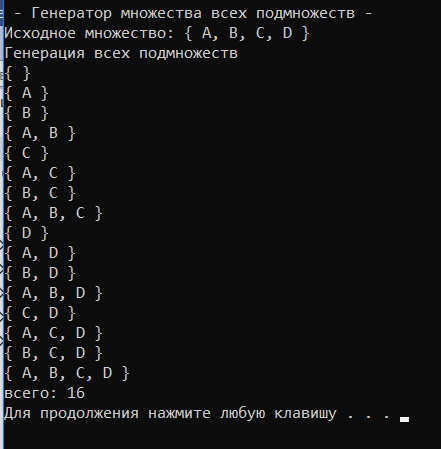
Combi1.cpp – реализация функций структуры

|  |
| --- |
| #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);  };  }; |

Main.cpp – применение структуры

|  |
| --- |
| #include <iostream>  #include "Combi1.h"  #include <tchar.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 ";  combi1::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;  } |

Рез-тат выполнения.



**Задание 2.** Разработать генератор сочетаний

Combi2.h

|  |
| --- |
| #pragma once  namespace combi2  {  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  };  } |

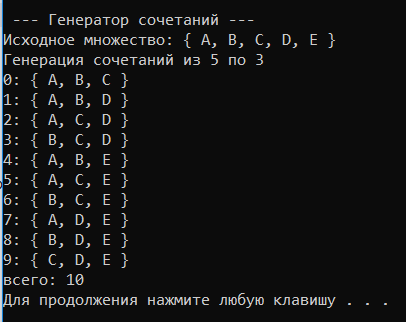
Combi2.cpp

|  |
| --- |
| #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]; };  } |

Main.cpp

|  |
| --- |
| #include <iostream>  #include <tchar.h>  #include "Combi2.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 ";  combi2::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;  } |

**Рез-тат выполнения**



**Задание 3.** Разработать генератор перестановок

Combi3.h

|  |
| --- |
| #pragma once  namespace combi3  {  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 (кол-во эл-ов исх. мн-ва)  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  };  }; |

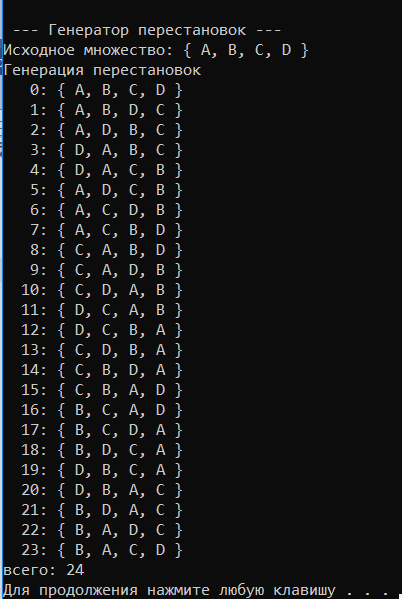
Combi3.cpp

|  |
| --- |
| #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); };  }; |

Main.cpp

|  |
| --- |
| #include <iostream>  #include "Combi3.h"  #include <iomanip>  #include <tchar.h>  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 ";  combi3::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;  } |

**Рез-тат выполнения:**

****

**Задание 4.** Разработать генератор размещений

Combi4.h

|  |
| --- |
| #pragma once  namespace combi4  {  struct xcombination // генератор сочетаний (эвристика)  {  short n, // кол-во 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(); // сформ. первый массив индексов  short getnext(); // сформ. следующий массив индексов  short ntx(short i); // получить i-й эл-т массива индексов  unsigned \_\_int64 nc; // номер сочетания 0,..., count()-1  unsigned \_\_int64 count() const; // вычислить кол-во 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 (кол-во эл-ов исх. мн-ва)  void reset(); // сбросить генератор, начать сначала  \_\_int64 getfirst(); // сформ. первый массив индексов  \_\_int64 getnext(); // сформ. 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  };  } |

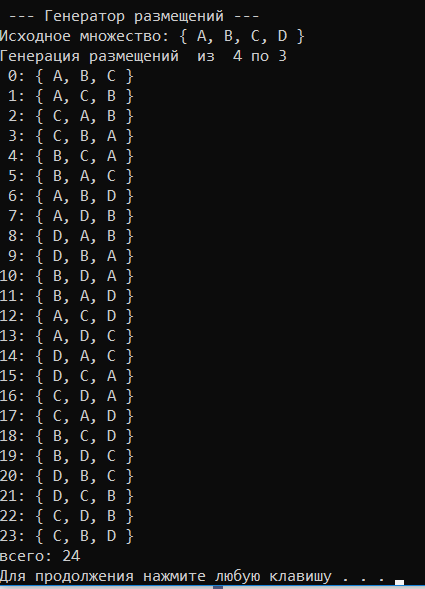
Combi4.cpp

|  |
| --- |
| #pragma once  namespace combi4  {  struct xcombination // генератор сочетаний (эвристика)  {  short n, // кол-во 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(); // сформ. первый массив индексов  short getnext(); // сформ. следующий массив индексов  short ntx(short i); // получить i-й эл-т массива индексов  unsigned \_\_int64 nc; // номер сочетания 0,..., count()-1  unsigned \_\_int64 count() const; // вычислить кол-во 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 (кол-во эл-ов исх. мн-ва)  void reset(); // сбросить генератор, начать сначала  \_\_int64 getfirst(); // сформ. первый массив индексов  \_\_int64 getnext(); // сформ. 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  };  } |

Main.cpp

|  |
| --- |
| #include <iostream>  #include <iomanip>  #include <tchar.h>  #include "Combi4.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;  combi4::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;  } |

**Рез-тат выполнения :**



**Задание 5.**  Решить в соответствии с вариантом задачу:

**Вариант 9** - коммивояжера (расстояния сгенерировать случайным образом: 10 городов, расстояния 10 – 300 км, 3 расстояния между городами задать бесконечными);

Salesman.h

|  |
| --- |
| #pragma once  //-- Salesman.h  // -- решение задачи коммивояжера перебором вариантов  #define INF 0x7fffffff // бесконечность  #include "Combi.h"  int salesman( // функция возвращает длину оптимального маршрута  int n, // [in] количество городов  const int\* d, // [in] массив [n\*n] расстояний  int\* r // [out] массив [n] маршрут 0 x x x x  ); |

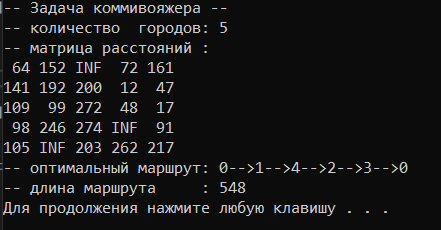
Salesman.cpp

|  |
| --- |
| // -- Salesman.cpp  #include "Salesman.h"  int sum(int x1, int x2) // суммирование с учетом бесконечности  {  return (x1 == INF || x2 == INF) ? INF : (x1 + x2);  };  int\* firstpath(int n) // формирование 1го маршрута 0,1,2,..., n-1, 0  {  int\* rc = new int[n + 1]; rc[n] = 0;  for (int i = 0; i < n; i++) rc[i] = i;  return rc;  };  int\* source(int n) // формирование исходного массива 1,2,..., n-1  {  int\* rc = new int[n - 1];  for (int i = 1; i < n; i++) rc[i - 1] = i;  return rc;  };  void copypath(int n, int\* r1, const int\* r2) // копировать маршрут  {  for (int i = 0; i < n; i++) r1[i] = r2[i];  };  int distance(int n, int\* r, const int\* d) // длина маршрута  {  int rc = 0;  for (int i = 0; i < n - 1; i++) rc = sum(rc, d[r[i] \* n + r[i + 1]]);  return sum(rc, d[r[n - 1] \* n + 0]); //+ последняя дуга (n-1,0)  };  void indx(int n, int\* r, const int\* s, const short\* ntx)  {  for (int i = 1; i < n; i++) r[i] = s[ntx[i - 1]];  }  int salesman(  int n, // [in] количество городов  const int\* d, // [in] массив [n\*n] расстояний  int\* r // [out] массив [n] маршрут 0 x x x x  )  {  int\* s = source(n), \* b = firstpath(n), rc = INF, dist = 0;  combi::permutation p(n - 1);  int k = p.getfirst();  while (k >= 0) // цикл генерации перестановок  {  indx(n, b, s, p.sset); // новый маршрут  if ((dist = distance(n, b, d)) < rc)  {  rc = dist; copypath(n, r, b);  }  k = p.getnext();  };  return rc;  } |

Lab\_2\_5.cpp

|  |
| --- |
| #include "Auxil.cpp"  #include <iostream>  #include <iomanip>  #include "Salesman.h"  #define N 5  int main()  {  setlocale(LC\_ALL, "rus");  int d[N \* N + 1], r[N];  auxil::start();  for (int i = 0; i <= N \* N; i++) d[i] = auxil::iget(10, 300);  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;  system("pause");  return 0;  } |

**Рез-тат выполнения :**

****

**Задание 6.** Исследовать зависимость времени вычисления необходимое для решения задачи (в соответствии с вариантом) от размерности задачи и результат в виде графика с небольшим пояснением занести в отчет

**Вариант 9** - коммивояжера (6 – 12 городов)

Lab\_2\_6.cpp

|  |
| --- |
| // -- main  #include "Auxil.h"  #include <iostream>  #include <iomanip>  #include <time.h>  #include "Salesman.h"  #define SPACE(n) std::setw(n)<<" "  #define N 12  int main()  {  setlocale(LC\_ALL, "rus");  int d[N \* N + 1], r[N];  auxil::start();  for (int i = 0; i <= N \* N; i++) d[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\*)d, r);  t2 = clock();  std::cout << std::endl << SPACE(7) << std::setw(2) << i  << SPACE(15) << std::setw(5) << (t2 - t1);  }  std::cout << std::endl;  system("pause");  return 0;  } |

Рез-тат при разном количестве контейнеров:

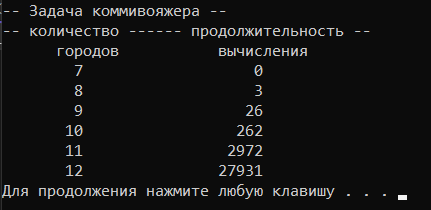
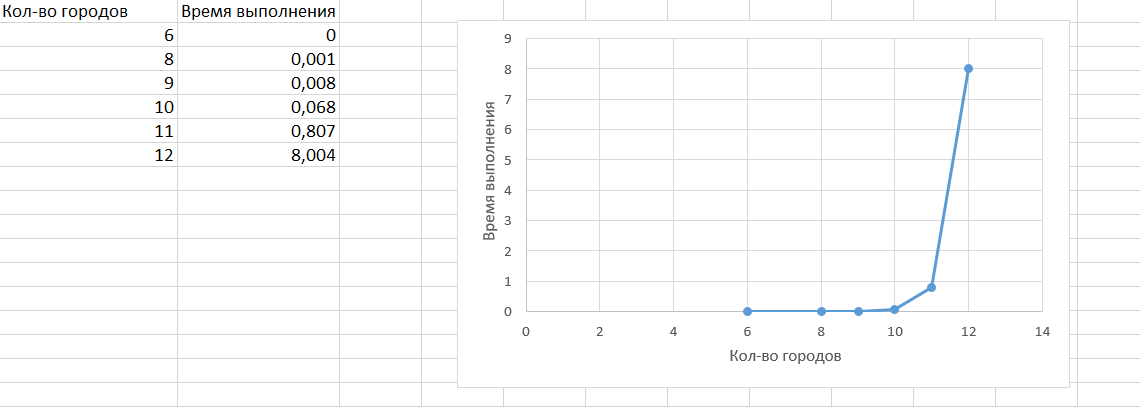


График зависимости времени выполнения от количества городов:



**Вывод:** Скорость выполнения алгоритма по нахождению оптимального маршрута между городами увеличивалась экспоненциально при увеличении кол-ва городов.