Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Математическое программирование

Студент: Трубач Д. С.

ФИТ 2 курс 5 группа

Лаборант: Степанова Л. П.

Минск 2023

Содержание

[Лабораторная работа №1. Вспомогательные функции 3](#_Toc130157676)

[Лабораторная работа 2. Комбинаторные алгоритмы решения оптимизационных задач 7](#_Toc130157677)

[Лабораторная работа 3. Метод ветвей и границ. Задача коммивояжера и методы её решения. 25](#_Toc130157678)

[Лабораторная работа 4 Динамическое программирование 33](#_Toc130157679)

# Лабораторная работа №1. Вспомогательные функции

**Цель работы:** приобретение навыков составления и отладки программ с использованием пользовательских функций для замера продолжительности процесса вычисления.

***Задание 1.*** Разработайте три функции (start, dget и iget), используя следующие спецификации

|  |
| --- |
| #include "stdafx.h"  #include "Auxil.h"  #include <ctime>  namespace auxil  {  void start() // старт генератора сл. чисел  {  srand((unsigned)time(NULL));  };  double dget(double rmin, double rmax) // получить случайное число  {  return ((double)rand() / (double)RAND\_MAX) \* (rmax - rmin) + rmin;  };  int iget(int rmin, int rmax) // получить случайное число  {  return (int)dget((double)rmin, (double)rmax);  };  } |

Листинг 1.1 — Файл Auxil.cpp

|  |
| --- |
| #pragma once  #include <cstdlib>  namespace auxil  {  void start(); // старт генератора сл. чисел  double dget(double rmin, double rmax); // получить случайное число  int iget(int rmin, int rmax); // получить случайное число  }; |

Листинг 1.2 — Файл Auxil.h

***Задание 2***

1. Реализовать пример 2.
2. Для проверки работоспособности разработанных функций и приобретения навыков замера продолжительности процесса вычисления реализуйте программу, приведенную в примере 2.

|  |
| --- |
| #include "stdafx.h"  #include "Auxil.h" // вспомогательные функции  #include "Fibonachi.h"  #include <iostream>  #include <ctime>  #include <locale>  #define CYCLE 1000000 // количество циклов  int main(int argc, char\* argv[])  {  double av1 = 0, av2 = 0;  clock\_t t1 = 0, t2 = 0;  setlocale(LC\_ALL, "rus");  auxil::start(); // старт генерации  t1 = clock(); // фиксация времени  for (int i = 0; i < CYCLE; i++)  {  av1 += (double)auxil::iget(-100, 100); // сумма случайных чисел  av2 += auxil::dget(-100, 100); // сумма случайных чисел  }  t2 = clock(); // фиксация времени  std::cout << std::endl << "количество циклов: " << CYCLE;  std::cout << std::endl << "среднее значение (int): " << av1 / CYCLE;  std::cout << std::endl << "среднее значение (double): " << av2 / CYCLE;  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 1.3 — Пример 2

**Задание 3**

Проведите необходимые эксперименты и постройте график зависимости (Excel) продолжительности процесса вычисления от количества циклов в примере 2. Проанализируйте характер зависимости. Проведите исследование любого другого рекурсивного алгоритма, например, вычисления факториала или генератора чисел Фибоначчи (прим. – например вычислите каким будет 100-е, 200-е, 300-е и т.д число), и включите в отчет график.

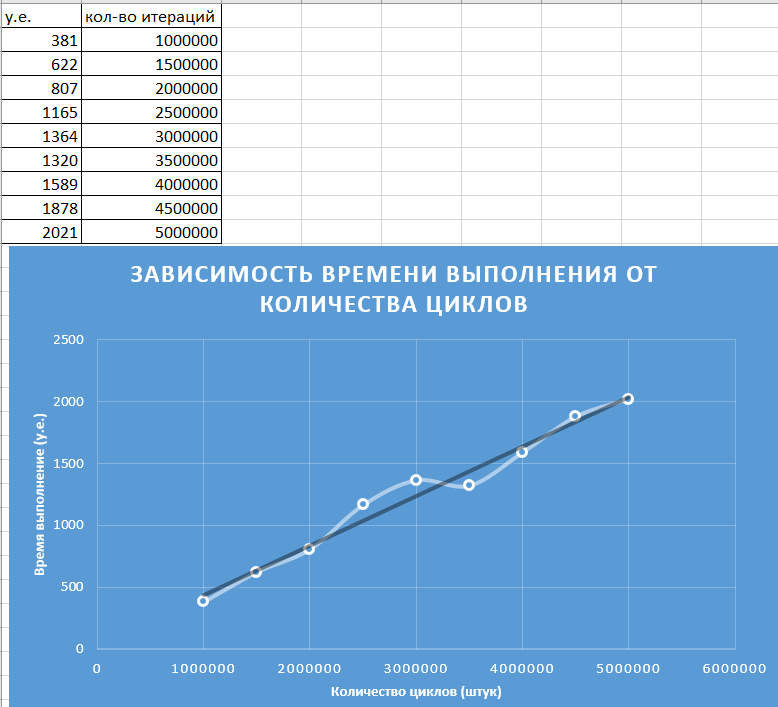


Рисунок 1.1 — График зависимости продолжительности процесса вычисления от количества циклов

Найдем коэффициент корреляции для данной зависимости:



Рисунок 1.2 — Коэффициент корреляции

Так как коэффициент приблизительно равен единице, можем сделать вывод о том, что зависимость продолжительности процесса вычисления от количества цикла имеет линейный вид.

**Числа Фибоначчи:**

Последовательность чисел Фибоначчи определяется формулой Fn = Fn-1 + Fn-2. То есть, следующее число получается как сумма двух предыдущих.

Первые два числа равны 1, затем 2(1+1), затем 3(1+2), 5(2+3) и так далее: 1, 1, 2, 3, 5, 8, 13, 21....

Реализацию данного алгоритма и график зависимости времени выполнения от заданного числа можно посмотреть в листинге 1.4 и на рисунке 1.3., соответственно.

|  |
| --- |
| //-- Fibonachi.cpp  #include "Fibonachi.h"  #include <iostream>  // Посчитать N-ое число Фибоначчи  long double fibonachi(int n)  {  if (n == 0)  return 0;  if (n == 1)  return 1;  return fibonachi(n - 1) + fibonachi(n - 2);  }  //-- Lab1.cpp  std::cout << "=============\n\nФункция чисел Фиббоначи";  clock\_t t3 = 0, t4 = 0; int a;  std::cout << std::endl << "Введите N-ое число: ";  std::cin >> a;  t3 = clock();  long double result = fibonachi(a);  std::cout << std::endl << a << " число Фиббоначи = " << result;  t4 = clock();  std::cout << std::endl << "продолжительность (у.е): " << (t4 - t3);  std::cout << std::endl << " (сек): "  << ((double)(t4 - t3)) / ((double)CLOCKS\_PER\_SEC);  std::cout << std::endl; |

Листинг 1.4 — Реализация алгоритма чисел Фибоначчи и тестирование

Рисунок 1.3 — График зависимости продолжительности процесса вычисления от количества циклов

Вывод: скорость выполнения программы линейно зависит от количества итераций цикла. Скорость нахождения N-го числа Фибоначчи имеет вид показательной зависимости.

# 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.

**Задание 1.** Разобрать и разработать генератор подмножеств заданного множества.

Ниже в листингах представлены заголовочные файлы и файлы .cpp с реализацией поставленной задачи.

|  |
| --- |
| // 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 2.1 — файл Combi.h

|  |
| --- |
| // Combi.cpp  #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 2.2 — файл Combi.cpp

|  |
| --- |
| // Main  #include <iostream>  #include "Combi.h"  int main() // с помощью инкременирования битовой маски  {  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;  } |

Листинг 2.3 — файл Subsets(Task1).cpp

Результат работы генератора подмножеств заданного множества предоставлен на рисунке 2.1.

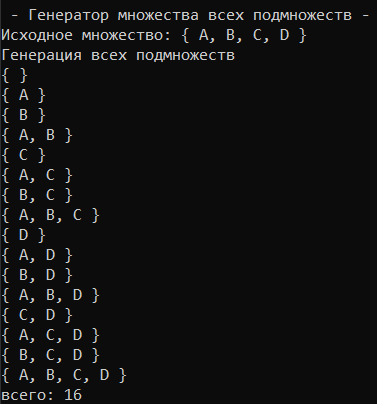


Рисунок 2.1 — результат работы генератора подмножеств заданного множества

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

Ниже в листингах представлены заголовочные файлы и файлы .cpp с реализацией поставленной задачи.

|  |
| --- |
| // 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 2.4 — файл Combi.h

|  |
| --- |
| // Combi.cpp  #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 2.5 — файл Combi.cpp

|  |
| --- |
| // Main  #include <iostream>  #include "Combi.h"  int main() // тоже битовая маска, но берем только подмножества с N двоичными единицами  {  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;  } |

Листинг 2.6 — файл Combinations(Task2).cpp

Результат работы генератора сочетаний предоставлен на рисунке 2.2.

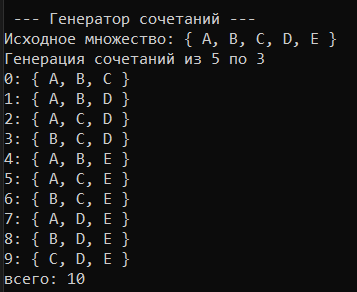


Рисунок 2.2 — результат работы генератора сочетаний

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

Ниже в листингах представлены заголовочные файлы и файлы .cpp с реализацией поставленной задачи.

|  |
| --- |
| // 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 2.7 — файл Combi.h

|  |
| --- |
| // Combi.cpp  #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 2.8 — файл Combi.cpp

|  |
| --- |
| // --- Main  #include <iostream>  #include "Combi.h"  #include <iomanip>  int main() // алгоритм Джонсона – Троттера  {  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;  } |

Листинг 2.9 — файл Permutations(Task3).cpp

Результат работы генератора перестановок предоставлен на рисунке 2.3.

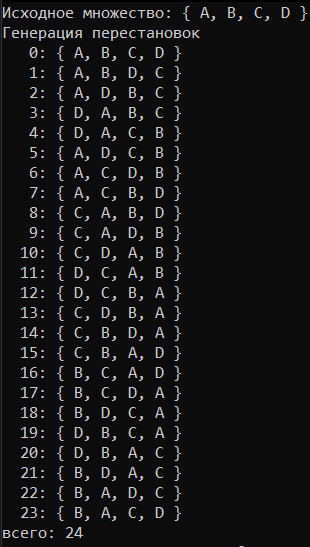


Рисунок 2.3 — результат работы генератора перестановок

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

Ниже в листингах представлены заголовочные файлы и файлы .cpp с реализацией поставленной задачи.

|  |
| --- |
| // 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  };  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 2.10 — файл Combi.h

|  |
| --- |
| // Combi.cpp  #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() // ñáðîñèòü ãåíåðàòîð, íà÷àòü ñíà÷àëà  {  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() // ñôîðìèðîâàòü ñëåäóþùèé ìàññèâ èíäåêñîâ  {  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 2.11 — файл Combi.cpp

|  |
| --- |
| // --- main  #include <iostream>  #include <iomanip>  #include "Combi.h"  #define N (sizeof(AA)/2)  #define M 3  int main()  {  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 < M; 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;  } |

Листинг 2.12 — файл Accommodations(Task4).cpp

Результат работы генератора размещений предоставлен на рисунке 2.4.

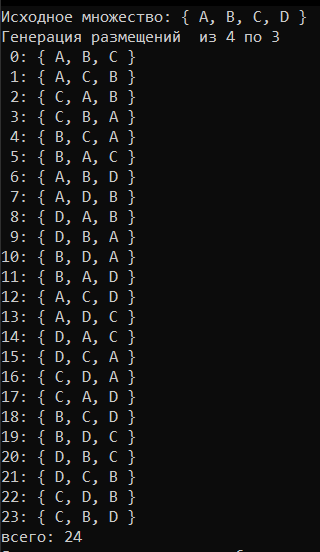


Рисунок 2.4 — результат работы генератора размещений

**Задание 5.**  Решить в соответствии с вариантом задачу и результат занести в отчет (Вариант распределяется по списку):

3, 7, 11, 15) об оптимальной загрузке судна (веса контейнеров сгенерировать случайным образом: ограничение по общему весу – 1500 кг., количество мест на судне для контейнеров – 5, количество контейнеров 25, веса контейнеров 100 – 900 кг., доход от перевозки 10 – 150 у.е.);

Ниже в листингах представлены заголовочные файлы и файлы .cpp с реализацией поставленной задачи.

|  |
| --- |
| // --- Вoat.h  // -- решение задачи об оптимальной загрузке судна  // функция возвращает доход от перевози выбранных контейнеров  #pragma once  #include "../Combinations(Task2)/Combi.h"  int boat(  int V, // [in] максимальный вес груза  short m, // [in] количество мест для контейнеров  short n, // [in] всего контейнеров  const int v[], // [in] вес каждого контейнера  const int c[], // [in] доход от перевозки каждого контейнера  short r[] // [out] результат: индексы выбранных контейнеров  ); |

Листинг 2.13 — файл Boat.h

|  |
| --- |
| // --- Boat.cpp  #include "Boat.h"  #include "../Combinations(Task2)/Combi.cpp"  namespace boatfnc  {  int calcv(combi::xcombination s, const int v[]) // вес  {  int rc = 0;  for (int i = 0; i < s.m; i++) rc += v[s.ntx(i)];  return rc;  };  int calcc(combi::xcombination s, const int c[]) // доход  {  int rc = 0;  for (int i = 0; i < s.m; i++) rc += c[s.ntx(i)];  return rc;  };  void copycomb(short m, short\* r1, const short\* r2) // копировать  {  for (int i = 0; i < m; i++) r1[i] = r2[i];  };  }  int boat(  int V, // [in] максимальный вес груза  short m, // [in] количество мест для контейнеров  short n, // [in] всего контейнеров  const int v[], // [in] вес каждого контейнера  const int c[], // [in] доход от перевозки каждого контейнера  short r[] // [out] результат: индексы выбранных контейнеров  )  {  combi::xcombination xc(n, m);  int rc = 0, i = xc.getfirst(), cc = 0;  while (i > 0)  {  if (boatfnc::calcv(xc, v) <= V)  if ((cc = boatfnc::calcc(xc, c)) > rc)  {  rc = cc; boatfnc::copycomb(m, r, xc.sset);  }  i = xc.getnext();  };  return rc;  }; |

Листинг 2.14 — файл Boat.cpp

|  |
| --- |
| #include <iostream>  #include <iomanip>  #include "Boat.h"  #define NN (sizeof(v)/sizeof(int))  #define MM 3  int wmain()  {  setlocale(LC\_ALL, "rus");  int V = 1000,  v[] = { 100, 200, 300, 400, 500, 150 },  c[NN] = { 10, 15, 20, 25, 30, 25 };  short r[MM];  int cc = boat(  V, // [in] максимальный вес груза  MM, // [in] количество мест для контейнеров  NN, // [in] всего контейнеров  v, // [in] вес каждого контейнера  c, // [in] доход от перевозки каждого контейнера  r // [out] результат: индексы выбранных контейнеров  );  std::cout << std::endl << "- Задача о размещении контейнеров на судне";  std::cout << std::endl << "- общее количество контейнеров : " << NN;  std::cout << std::endl << "- количество мест для контейнеров : " << MM;  std::cout << std::endl << "- ограничение по суммарному весу : " << V;  std::cout << std::endl << "- вес контейнеров : ";  for (int i = 0; i < NN; i++) std::cout << std::setw(3) << v[i] << " ";  std::cout << std::endl << "- доход от перевозки : ";  for (int i = 0; i < NN; i++) std::cout << std::setw(3) << c[i] << " ";  std::cout << std::endl << "- выбраны контейнеры (0,1,...,m-1): ";  for (int i = 0; i < MM; i++) std::cout << r[i] << " ";  std::cout << std::endl << "- доход от перевозки : " << cc;  std::cout << std::endl << "- общий вес выбранных контейнеров : ";  int s = 0; for (int i = 0; i < MM; i++) s += v[r[i]]; std::cout << s;  std::cout << std::endl << std::endl;  system("pause");  return 0;  } |

Листинг 2.15 — файл Boat(Task5).cpp

Решение задачи об оптимальной загрузке судна показан на рисунке 2.5.

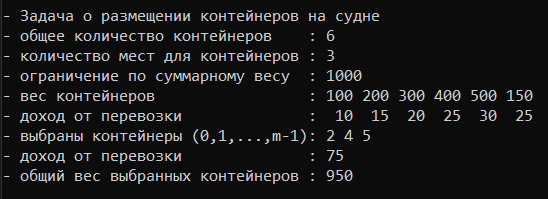


Рисунок 2.5 — решение задачи об оптимальной загрузке судна

**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:

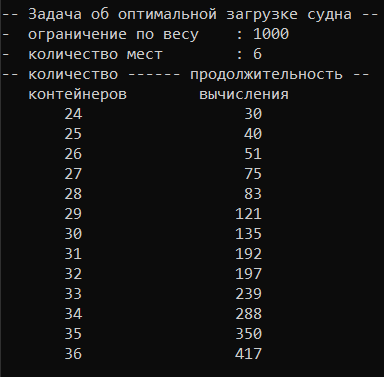
3, 7, 11, 15) об оптимальной загрузке судна (количество мест на судне для контейнеров – 6, количество контейнеров 25 – 35

Зависимость времени вычисления от количества контейнеров вычисляется с помощью кода, показанном в листинге 2.16.

|  |
| --- |
| #include <iostream>  #include <iomanip>  #include "Boat.h"  #include <time.h>  #define NN (sizeof(v)/sizeof(int))  #define MM 6  #define SPACE(n) std::setw(n)<<" "  int wmain()  {  setlocale(LC\_ALL, "rus");  int V = 1000,  v[] = { 250, 560, 670, 400, 200, 270, 370, 330, 330, 440, 530, 120,  200, 270, 370, 330, 330, 440, 700, 120, 550, 540, 420, 170,  600, 700, 120, 550, 540, 420, 430, 140, 300, 370, 310, 120 };  int c[NN] = { 15,26, 27, 43, 16, 26, 42, 22, 34, 12, 33, 30,  42,22, 34, 43, 16, 26, 14, 12, 25, 41, 17, 28,  12,45, 60, 41, 33, 11, 14, 12, 25, 41, 30, 40 };  short r[MM];  int maxcc = 0;  clock\_t t1, t2;  std::cout << std::endl << "-- Задача об оптимальной загрузке судна -- ";  std::cout << std::endl << "- ограничение по весу : " << V;  std::cout << std::endl << "- количество мест : " << MM;  std::cout << std::endl << "-- количество ------ продолжительность -- ";  std::cout << std::endl << " контейнеров вычисления ";  for (int i = 24; i <= NN; i++)  {  t1 = clock();  int maxcc = boat(V, MM, i, v, c, r);  t2 = clock();  std::cout << std::endl << SPACE(7) << std::setw(2) << i  << SPACE(15) << std::setw(5) << (t2 - t1);  }  std::cout << std::endl << std::endl;  system("pause");  return 0;  } |

Листинг 2.16

Результат выполнения кода и график зависимости приведён на рисунке 2.6.



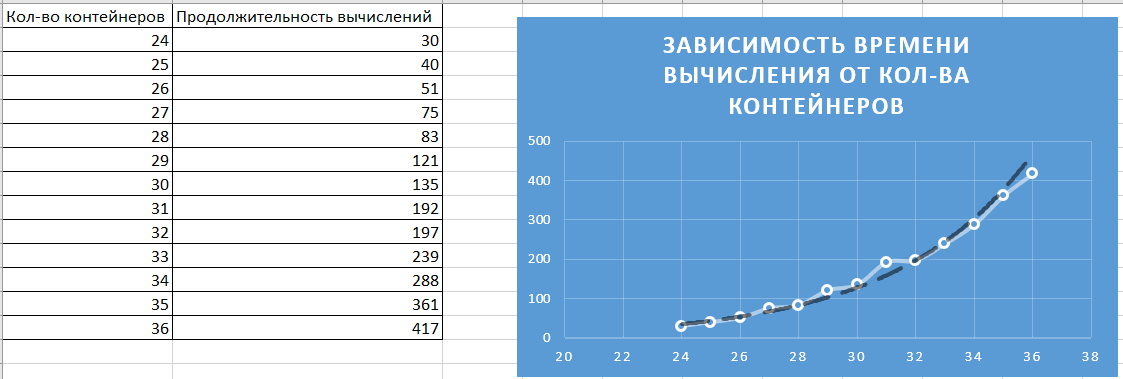


Рисунок 2.6 — Зависимость времени вычисления от кол-ва контейнеров

Вывод: исходя из полученных данных и графика, можно заметить, что скорость выполнения программы плавно возрастает при добавлении количества контейнеров.

# Лабораторная работа 3. Метод ветвей и границ. Задача коммивояжера и методы её решения.

**Цель работы:** освоить общие принципы решения задач методом ветвей и границ, решить задачу о коммивояжере данным методом, сравнить полученное решение задачи с комбинаторным методом перестановок.

**Задание 1 (вариант 11):**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Город** | **1** | **2** | **3** | **4** | **5** |
| **1** |  | 22 | 32 |  | 11 |
| **2** | 11 |  | 26 | 57 | 73 |
| **3** | 13 | 33 |  | 86 | 60 |
| **4** | 28 | 47 | 44 |  | 33 |
| **5** | 82 | 77 | 52 | 24 |  |

Задача коммивояжера с параметром заключается в поиске гамильтонова цикла минимального веса в полном взвешенном графе, заданном матрицей расстояний, при условии, что в этом цикле есть ребро с максимальным весом не больше заданного параметра.

В данном случае матрица расстояний содержит расстояния между 5 городами, заданными номерами от 1 до 5. Значение INF означает отсутствие ребра между соответствующими городами. Задача коммивояжера с параметром для данной матрицы заключается в поиске гамильтонова цикла минимального веса в графе, при условии, что в этом цикле есть ребро с максимальным весом не больше заданного параметра.

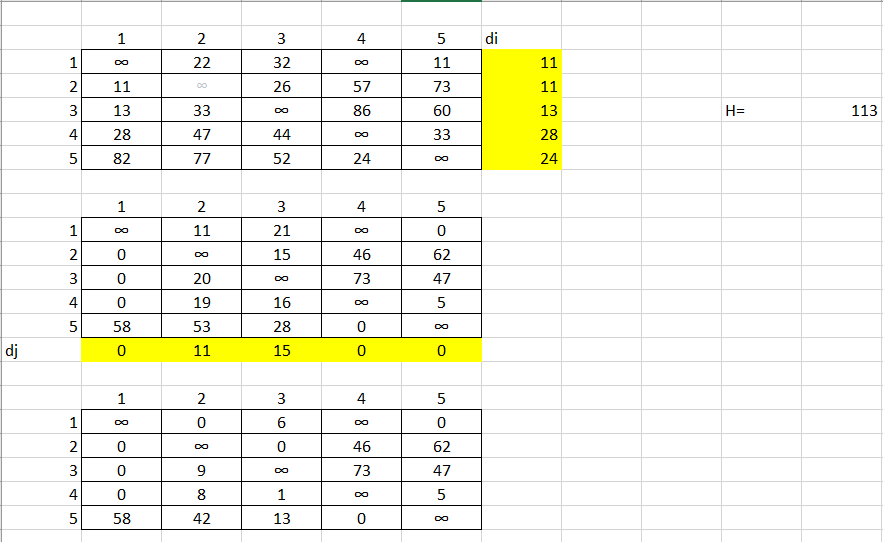
Гамильтонов цикл – такой путь, который проходит через все вершины графа ровно один раз.

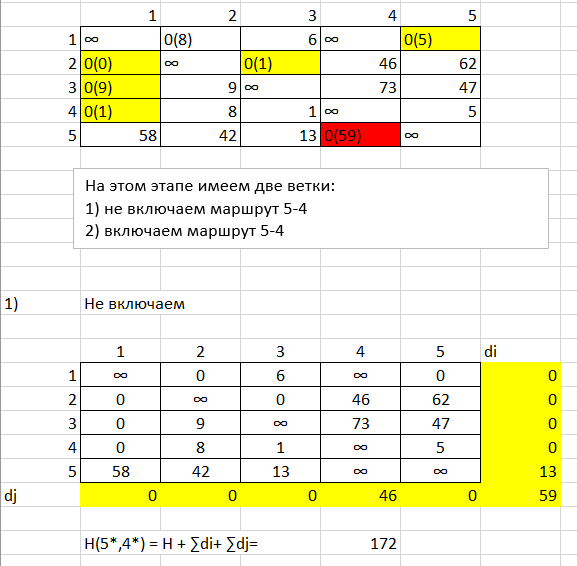
Параметр в задаче коммивояжера с параметром - это число, которое ограничивает вес ребра, которое может быть включено в найденный гамильтонов цикл.

Более конкретно, параметр определяет максимально допустимый вес ребра в гамильтоновом цикле. Если в графе есть несколько ребер, вес которых больше параметра, то цикл, содержащий любое из этих ребер, не будет оптимальным.

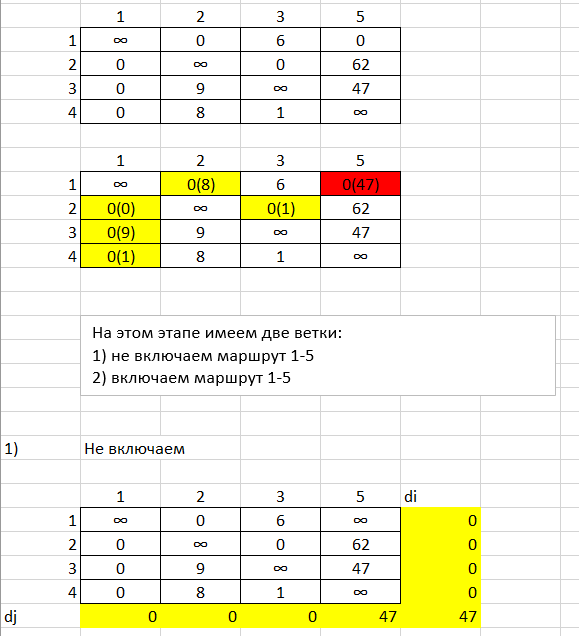
Цель задачи коммивояжера с параметром - найти гамильтонов цикл минимального веса в заданном графе, который удовлетворяет ограничению на максимальный вес ребра. Значение параметра может быть задано заранее или получено в результате вычислений, и влияет на решение задачи.

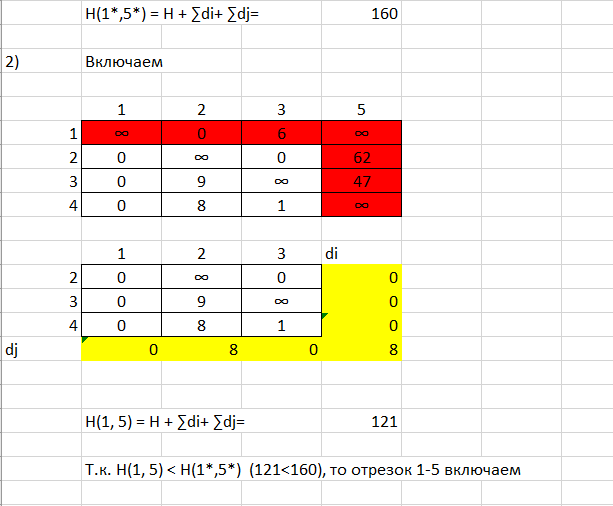
**Задание 2:**

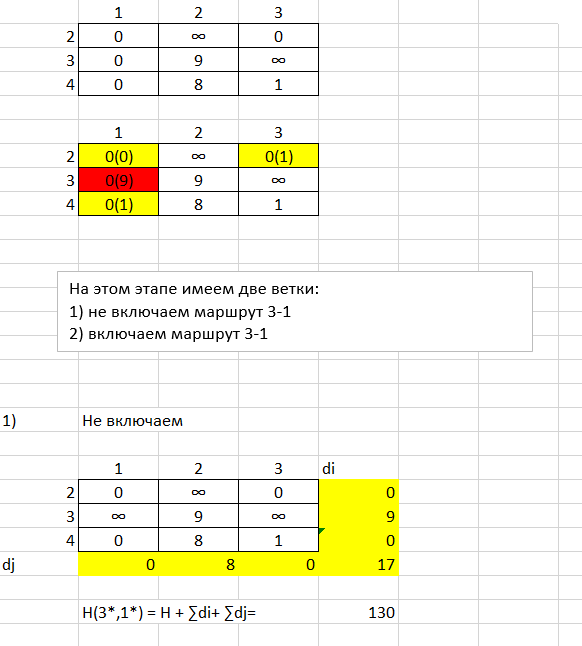
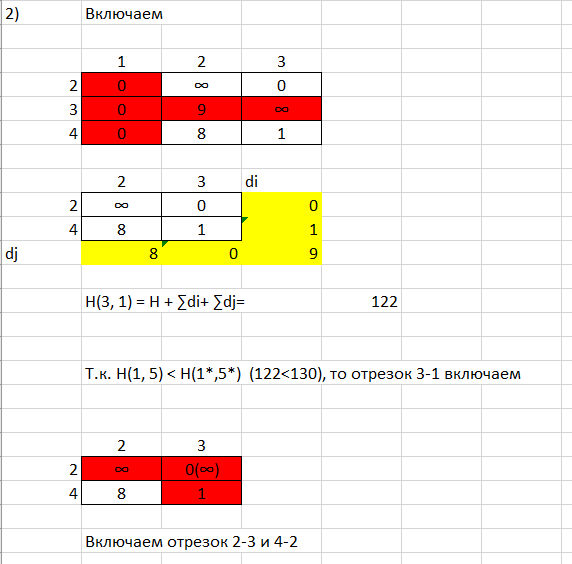












В результате мы получили следующий оптимальный маршрут: **121 (1->5->4->2->3->1)**

Граф обхода матрицы представлен на рисунке 1.

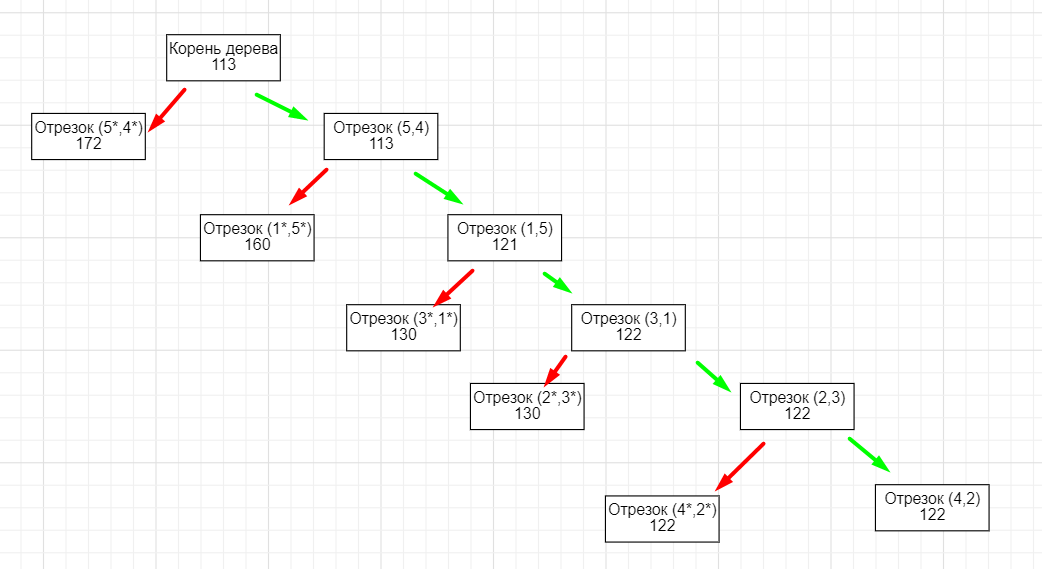


Рисунок 1 — обход исходной матрицы

**Задание 3.**

После проверки результата при помощи генератора перестановок из лабораторной работы 2, полученные ответы совпадали с исходным решением. Демонстрация выполнения программы представлена на рисунке 2.

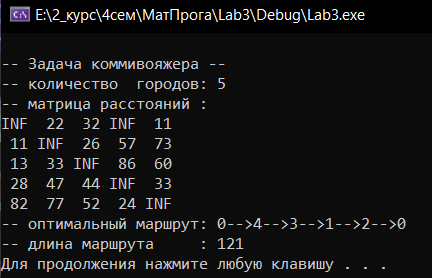


Рисунок 2 — результат выполнения программы

# Лабораторная работа 4. Динамическое программирование

**ЦЕЛЬ РАБОТЫ:** освоить общие принципы решения задач методом динамического программирования, сравнить полученные решения задач с рекурсивным методом.

**Ход выполнения работы**

**Задание 1.** На языке С++ сгенерировать случайным образом строку букв латинского алфавита  длиной  символов и длиной .

**Решение:**

Исходный код программы приведен в листинге 3.1, а результат показан на рисунке 3.1.

|  |
| --- |
| #include <iostream>  #include <ctime>  #include <Windows.h>  using namespace std;  #define FIRST\_LEN 300  #define SECOND\_LEN 250  char\* GenerateRandomString(int size)  {  char\* str = (char\*)malloc(sizeof(char) \* size);  for (int i = 0; i < size; i++) {  str[i] = rand() % 26 + 'a'; // 26 букв в алфавите  }  return str;  }  int main()  {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  srand(time(NULL));  char\* s1 = GenerateRandomString(FIRST\_LEN);  cout << "S1: " << endl;  for (int i = 0; i < FIRST\_LEN; i++) {  if (i % 50 == 0)  {  cout << "\n";  }  cout << s1[i];  }  cout << endl << endl;  srand(time(NULL) + 1);  char\* s2 = GenerateRandomString(SECOND\_LEN);  cout << "S2: " << endl;  for (int i = 0; i < SECOND\_LEN; i++) {  if (i % 50 == 0)  {  cout << "\n";  }  cout << s2[i];  }  cout << endl << endl;  } |

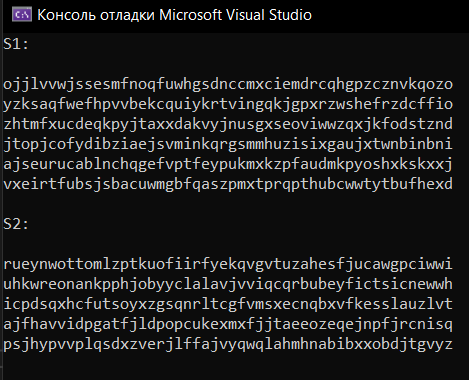
Листинг 3.1 — файл Task1(String\_generator).cpp  


Рисунок 3.1 — результат

**Задание 2.**

Вычислить двумя способами (рекурсивно и с помощью динамического программирования)  – дистанцию Левенштейна для , где - длина строки ,  - строка состоящая из первых  символов строки . (копии экрана и код вставить в отчет).

**Решение:**

Ниже приведены варианты реализации нахождения дистанции Левенштейна при помощи динамического программирования и при помощи рекурсивного алгоритма.

Исходный код реализации через динамическое программирование:

|  |
| --- |
| #include <iomanip>  #include <algorithm>  #include "Levenshtein.h"  #define DD(i,j) d[(i)\*(ly+1)+(j)]  int min3(int x1, int x2, int x3)  {  return std::min(std::min(x1, x2), x3); // минимальное из трех  }  int levenshtein(int lx, const char x[], int ly, const char y[])  {  // длина слова х, слово длиной lx  // длина слова y, слово длиной ly  int\* d = new int[(lx + 1) \* (ly + 1)];  for (int i = 0; i <= lx; i++) DD(i, 0) = i; // длина строки х = i (кол-во символов в слове x), а DD возвращ. саму строку  for (int j = 0; j <= ly; j++) DD(0, j) = j; // длина строки y = j  for (int i = 1; i <= lx; i++)  for (int j = 1; j <= ly; j++)  {  DD(i, j) = min3(DD(i - 1, j) + 1, DD(i, j - 1) + 1,  DD(i - 1, j - 1) + (x[i - 1] == y[j - 1] ? 0 : 1));  }  return DD(lx, ly); // возвращает длину слова и строки  } |

Листинг 3.2 — вычисление дистанции Левенштейна с помощью динамического программирования

Пример реализации рекурсивным методом:

|  |
| --- |
| #include <iomanip>  #include <algorithm>  #include "Levenshtein.h"  #define DD(i,j) d[(i)\*(ly+1)+(j)]  int levenshtein\_r( // рекурсия  int lx, const char x[],  int ly, const char y[]  )  {  int rc = 0; // результат, дистанция  if (lx == 0) rc = ly; // если длина x = 0, то дистанция = длине слова y  else if (ly == 0) rc = lx;  else if (lx == 1 && ly == 1 && x[0] == y[0]) rc = 0; // если длина x = длина y = 1 и первые буквы совапали, дистанция = 0  else if (lx == 1 && ly == 1 && x[0] != y[0]) rc = 1; // если длина x = длина y = 1 и первые буквы не совапали, дистанция = 1  else rc = min3(  levenshtein\_r(lx - 1, x, ly, y) + 1,  levenshtein\_r(lx, x, ly - 1, y) + 1,  levenshtein\_r(lx - 1, x, ly - 1, y) + (x[lx - 1] == y[ly - 1] ? 0 : 1)  );  return rc;  }; |

Листинг 3.3 — вычисление дистанции Левенштейна с помощью рекурсии

Полный код программы находится в листингах 3.4, 3.5, 3.6.

|  |
| --- |
| // - Levenshtein.h  // -- дистанции Левенштeйна (динамическое программирование)  int levenshtein(  int lx, // длина слова x  const char x[], // слово длиной lx  int ly, // длина слова y  const char y[] // слово y  );  // -- дистанции Левенштeйна (рекурсия)  int levenshtein\_r(  int lx, // длина строки x  const char x[], // строка длиной lx  int ly, // длина строки y  const char y[] // строка y  ); |

Листинг 3.4 — файл Levenshtein.h

|  |
| --- |
| // - Levenshtein.cpp  #include <iomanip>  #include <algorithm>  #include "Levenshtein.h"  #define DD(i,j) d[(i)\*(ly+1)+(j)]  int min3(int x1, int x2, int x3)  {  return std::min(std::min(x1, x2), x3); // минимальное из трех  }  int levenshtein(int lx, const char x[], int ly, const char y[])  {  // длина слова х, слово длиной lx  // длина слова y, слово длиной ly  int\* d = new int[(lx + 1) \* (ly + 1)];  for (int i = 0; i <= lx; i++) DD(i, 0) = i; // длина строки х = i (кол-во символов в слове x), а DD возвращ. саму строку  for (int j = 0; j <= ly; j++) DD(0, j) = j; // длина строки y = j  for (int i = 1; i <= lx; i++)  for (int j = 1; j <= ly; j++)  {  DD(i, j) = min3(DD(i - 1, j) + 1, DD(i, j - 1) + 1,  DD(i - 1, j - 1) + (x[i - 1] == y[j - 1] ? 0 : 1));  }  return DD(lx, ly); // возвращает длину слова и строки  }  int levenshtein\_r( // рекурсия  int lx, const char x[],  int ly, const char y[]  )  {  int rc = 0; // результат, дистанция  if (lx == 0) rc = ly; // если длина x = 0, то дистанция = длине слова y  else if (ly == 0) rc = lx;  else if (lx == 1 && ly == 1 && x[0] == y[0]) rc = 0; // если длина x = длина y = 1 и первые буквы совапали, дистанция = 0  else if (lx == 1 && ly == 1 && x[0] != y[0]) rc = 1; // если длина x = длина y = 1 и первые буквы не совапали, дистанция = 1  else rc = min3(  levenshtein\_r(lx - 1, x, ly, y) + 1,  levenshtein\_r(lx, x, ly - 1, y) + 1,  levenshtein\_r(lx - 1, x, ly - 1, y) + (x[lx - 1] == y[ly - 1] ? 0 : 1)  );  return rc;  }; |

Листинг 3.5 — файл Levenshtein.cpp

|  |
| --- |
| // --- main  // вычисление дистанции (расстояния) Левенштейна  #include <algorithm>  #include <iostream>  #include <ctime>  #include <iomanip>  #include "Levenshtein.h"  #include <Windows.h>  #define FIRST\_LEN 300  #define SECOND\_LEN 250  using namespace std;  char\* GenerateRandomString(int size)  {  char\* str = (char\*)malloc(sizeof(char) \* size);  for (int i = 0; i < size; i++) {  str[i] = rand() % 26 + 'a'; // 26 букв в алфавите  }  return str;  }  int main()  {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  char\* s1 = GenerateRandomString(FIRST\_LEN);  cout << "S1: " << endl;  for (int i = 0; i < FIRST\_LEN; i++) {  if (i % 50 == 0)  {  cout << "\n";  }  cout << s1[i];  }  cout << endl << endl;  srand(time(NULL) + 1);  char\* s2 = GenerateRandomString(SECOND\_LEN);  cout << "S2: " << endl;  for (int i = 0; i < SECOND\_LEN; i++) {  if (i % 50 == 0)  {  cout << "\n";  }  cout << s2[i];  }  cout << endl << endl;  clock\_t t1 = 0, t2 = 0, t3 = 0, t4 = 0;  int lx = sizeof(s1);  int ly = sizeof(s2);  int s1\_size[]{ FIRST\_LEN / 25, FIRST\_LEN / 20, FIRST\_LEN / 15, FIRST\_LEN / 10, FIRST\_LEN / 5, FIRST\_LEN / 2, FIRST\_LEN };  int s2\_size[]{ SECOND\_LEN / 25, SECOND\_LEN / 20, SECOND\_LEN / 15, SECOND\_LEN / 10, SECOND\_LEN / 5, SECOND\_LEN / 2, SECOND\_LEN };  cout << "\n\n-- расстояние Левенштейна -----";  cout << "\n\n--длина --- рекурсия -- дин.програм. ---\n";  for (int i = 0; i < min(lx, ly); i++)  {  t1 = clock();  levenshtein\_r(s1\_size[i], s1, s2\_size[i], s2);  t2 = clock();  t3 = clock();  levenshtein(s1\_size[i], s1, s2\_size[i], s2);  t4 = clock();  cout << right << setw(2) << s1\_size[i] << "/" << setw(2) << s2\_size[i]  << " " << left << setw(10) << (t2 - t1)  << " " << setw(10) << (t4 - t3) << endl;  }  system("pause");  return 0;  } |

Листинг 3.6 — файл Task2(Levenshtein).cpp

На рисунке 3.2 представлены дистанции Левенштейна вычисленные при помощи метода динамического программирования, а также рекурсивным алгоритмом.

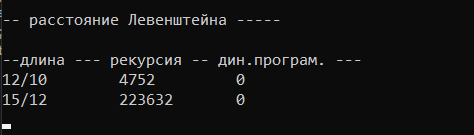


Рисунок 3.2 — результат программы

**Задание 3.** Выполнить сравнительный анализ времени затраченного на вычисление дистанции Левенштейна для двух методов решения. Построить графики зависимости времени вычисления от . (копии экрана и график вставить в отчет).

**Решение:**

На графике, представленном на рисунке 3.3, можно заметить, что выполненные с помощью динамического алгоритма, вычисления производятся в разы быстрее, чем с помощью рекурсивного алгоритма.

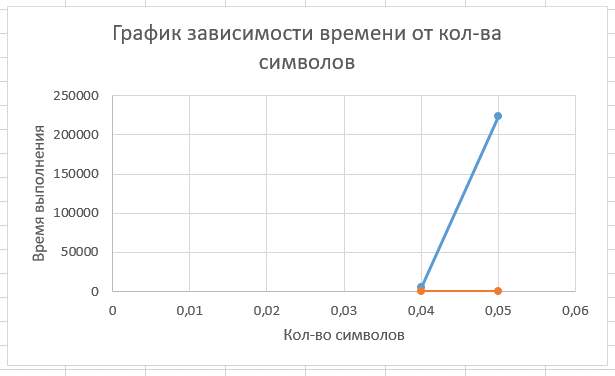


Рисунок 3.3 — сравнительный анализ времени выполнения

**Задание 4.** Реализовать вручную пример вычисления дистанции Левенштейна при помощи рекурсивного алгоритма (в соответствии с вариантом) (каждый шаг алгоритма по примеру из лекции вставить в отчет).

|  |  |
| --- | --- |
| Задание 4 (Вариант 11) | |
| Лот | Полет |

**Решение:**

1. 
2. 
3. 
4. 
5.   = 5  = 4
6.   = 4  = 3
7. 
8. 
9. 
10.   = 3  = 2
11.   = 3  = 2
12.  = 1
13.  = 2  = 1
14.   = 2 = 1
15.  =   = 1 = 1 = 0
16. 
17. 
18. 
19. 
20. 
21. 
22. 
23. 
24. 
25. 
26. 
27. 
28. 
29. 
30. 

**Задание 5. (нет решения, есть excel)**

**Нечетные варианты**. Выполнить сравнительный анализ времени затраченного на решение задачи о наибольшей общей подпоследовательности для двух методов решения (рекурсивное решение, динамическое программирование). Две последовательности взять в соответствии с вариантом. Построить графики зависимости времени вычисления от . **Отобразить ход решения в отчете**(по примеру из лекции) + код и копии экрана.

|  |  |
| --- | --- |
| Задание 5 (вариант 11) | |
| TOUEXAZ | HIEHXZ |

**Решение:**

|  |
| --- |
| // - LCS.h  // -- рекурсивное вычисление длины LCS  int lcs(int lenx, // длина последовательности X  const char x[], // последовательность X  int leny, // длина последовательности Y  const char y[] // последовательность Y  );  // -- динамическое вычисление LCS  int lcsd(const char x[], // последовательность X  const char y[], // последовательность Y  char z[] // наибольшая общая подпоследовательность  ); |

Листинг 5.1 — файл LCS.h

|  |
| --- |
| // - LCS.cpp  // -- рекурсивное вычисление длины LCS  #include <algorithm>  #include "LCS.h"  int lcs(int lenx, const char x[],  int leny, const char y[])  {  int rc = 0;  if (lenx > 0 && leny > 0)  {  if (x[lenx - 1] == y[leny - 1]) rc = 1 + lcs(lenx - 1, x, leny - 1, y);  else rc = std::max(lcs(lenx, x, leny - 1, y), lcs(lenx - 1, x, leny, y));  }  return rc; //длина LCS  }  #define LCS\_C(x1,x2) (C[(x1)\*(leny+1)+(x2)])  #define LCS\_B(x1,x2) (B[(x1)\*(leny+1)+(x2)])  #define LCS\_X(i) (x[(i)-1])  #define LCS\_Y(i) (y[(i)-1])  #define LCS\_Z(i) (z[(i)-1])  enum Dart { TOP, LEFT, LEFTTOP };  void getLCScontent(int lenx, int leny, const char x[],  const Dart\* B,  int n, int i, int j, char z[])  {  if ((i > 0 && j > 0 && n > 0))  {  if (LCS\_B(i, j) == LEFTTOP)  {  getLCScontent(lenx, leny, x, B, n - 1, i - 1, j - 1, z);  LCS\_Z(n) = LCS\_X(i);  LCS\_Z(n + 1) = 0;  }  else if (LCS\_B(i, j) == TOP)  getLCScontent(lenx, leny, x, B, n, i - 1, j, z);  else getLCScontent(lenx, leny, x, B, n, i, j - 1, z);  }  };  int lcsd(const char x[], const char y[], char z[])  {  int n;  int lenx = strlen(x), leny = strlen(x),  \* C = new int[(lenx + 1) \* (leny + 1)];  Dart\* B = new Dart[(lenx + 1) \* (leny + 1)];  memset(C, 0, sizeof(int) \* (lenx + 1) \* (leny + 1));  for (int i = 1; i <= lenx; i++)  for (int j = 1; j <= leny; j++)  if (LCS\_X(i) == LCS\_Y(j))  {  LCS\_C(i, j) = LCS\_C(i - 1, j - 1) + 1;  LCS\_B(i, j) = LEFTTOP;  }  else if (LCS\_C(i - 1, j) >= LCS\_C(i, j - 1))  {  LCS\_C(i, j) = LCS\_C(i - 1, j);  LCS\_B(i, j) = TOP;  }  else  {  LCS\_C(i, j) = LCS\_C(i, j - 1);  LCS\_B(i, j) = LEFT;  }  getLCScontent(lenx, leny, x, B, LCS\_C(lenx, leny), lenx, leny, z);  return LCS\_C(lenx, leny);  }  #undef LCS\_Z  #undef LCS\_C  #undef LCS\_B  #undef LCS\_X  #undef LCS\_Y |

Листинг 5.2 — файл LCS.cpp

|  |
| --- |
| // - main  // -- вычисления длины LCS  #include <iostream>  #include "LCS.h"  #include <Windows.h>  #include <ctime>  int main()  {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  char z[100] = "";  char X[] = "TOUEXAZ";  char Y[] = "HIEHXZ";  clock\_t t1 = 0, t2 = 0, t3 = 0, t4 = 0;  std::cout << std::endl << "-- вычисление длины LCS для X и Y(рекурсия)";  std::cout << std::endl << "-- последовательность X: " << X;  std::cout << std::endl << "-- последовательность Y: " << Y;  t1 = clock();  int s = lcs(sizeof(X) - 1, "TOUEXAZ", sizeof(Y) - 1, "HIEHXZ");  t2 = clock();  std::cout << std::endl << "-- длина LCS: " << s << std::endl;  // наибольшая общая подпоследовательность  t3 = clock();  int l = lcsd(X, Y, z);  t4 = clock();  std::cout << std::endl  << "-- наибольшая общая подпоследовательость - LCS(динамическое"  << " программирование)" << std::endl;  std::cout << std::endl << "последовательость X: " << X;  std::cout << std::endl << "последовательость Y: " << Y;  std::cout << std::endl << " LCS: " << z;  std::cout << std::endl << " длина LCS: " << l;  std::cout << std::endl;  std::cout << std::endl << "Время вычисления LCS";  std::cout << std::endl << "Рекурсия: " << (t2 - t1);  std::cout << std::endl << "Динамическое программирование: " << (t4 - t3) << std::endl;  system("pause");  return 0;  } |

Листинг 5.3 — файл Task5(Matrix)

На рисунке 5.1 представлена наибольшая общая подпоследовательность последовательностей Х и У, вычисленные при помощи метода динамического программирования, а также рекурсивным алгоритмом.

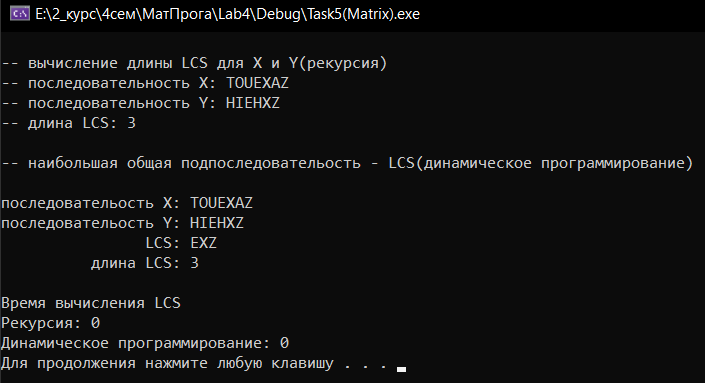


Рисунок 5.1 — результат выполнения программы

