Учреждение образования

«Белорусский государственный технологический университет»

**Лабораторная работа №2**

**Комбинаторные алгоритмы решения оптимизационных задач**

Выполнил:

Студент 2 курса 6 группы ФИТ

Дмитрук Илья Игоревич

**ЦЕЛЬ РАБОТЫ:** приобрести навыки разработки генераторов подмножеств, перестановок, сочетаний и размещений на С++; научиться применять разработанные генераторы для решения задач о рюкзаке (об оптимальной загрузке судна с центровкой.)

**Подмножество**

В терминологии теории множеств говорят, что множество B является подмножеством множества A, если каждый элемент B является в то же время и элементом множества A. Обозначается это знаком включения: B ⊂ A.

**Перестановки**

Число объектов остается неизменными, меняется только их порядок

**Размещения**

Меняется и состав выбранных объектов, и их порядок

**Сочетания**

Меняется состав выбранных объектов, но порядок не важен

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

Для первого задания был разработан генератор подмножеств на основание исходного множества. Код представлен в листинге 1

|  |
| --- |
| #include "Set.h"  #include "stdafx.h"  #include <algorithm>  namespace sets  {  subset::subset(short n)  {  this->n = n;  this->sset = new short[n];  this->reset();  };  void subset::reset()  {  this->sn = 0;  this->mask = 0;  };  short subset::getfirst()  {  \_\_int64 buf = this->mask;  this->sn = 0;  for (short i = 0; i < n; i++)  {  if (buf & 0x1) this->sset[this->sn++] = i;  buf >>= 1;  }  return this->sn;  };  short subset::getnext()  {  int rc = -1;  this->sn = 0;  if (++this->mask < this->count()) rc = getfirst();  return rc;  };  short subset::ntx(short i)  {  return this->sset[i];  };  unsigned \_\_int64 subset::count()  {  return (unsigned \_\_int64)(1 << this->n);  };  }; |

Листинг 1. Код Set.cpp

В листинге 2 представлен заголовочный файл для Set.h

|  |
| --- |
| #pragma once  namespace sets  {  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. Код Set.h

Результат представленного генератора можно увидеть на рисунке 1

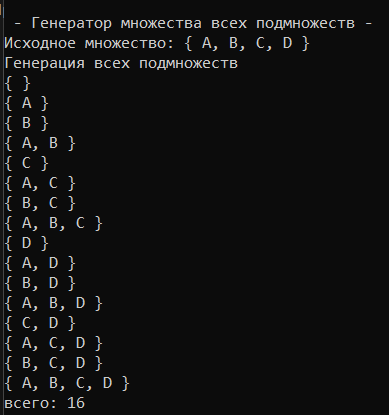


Рисунок 1. Демонстрация работы генератора

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

Для второго задания был разработан генератор сочетаний. Код представлен в листинге 3

|  |
| --- |
| #include "stdafx.h"  #include "Combi.h"  #include <algorithm>  namespace combi  {  xcombination::xcombination(short n, short m)  {  this->n = n;  this->m = m;  this->sset = new short[m + 2];  this->reset();  }  void xcombination::reset() // сбросить генератор, начать сначала  {  this->nc = 0;  for (int i = 0; i < this->m; i++) this->sset[i] = i;  this->sset[m] = this->n;  this->sset[m + 1] = 0;  };  short xcombination::getfirst()  {  return (this->n >= this->m) ? this->m : -1;  };  short xcombination::getnext() // сформировать следующий массив индексов  {  short rc = getfirst();  if (rc > 0)  {  short j;  for (j = 0; this->sset[j] + 1 == this->sset[j + 1]; ++j)  this->sset[j] = j;  if (j >= this->m) rc = -1;  else {  this->sset[j]++;  this->nc++;  };  }  return rc;  };  short xcombination::ntx(short i)  {  return this->sset[i];  };  unsigned \_\_int64 fact(unsigned \_\_int64 x) { return(x == 0) ? 1 : (x \* fact(x - 1)); };  unsigned \_\_int64 xcombination::count() const  {  return (this->n >= this->m) ?  fact(this->n) / (fact(this->n - this->m) \* fact(this->m)) : 0;  };  }; |

Листинг 3. Код Combi.cpp

В листинге 4 представлен заголовочный файл для Combi2.c

|  |
| --- |
| #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 4. Код Combi.h

Результат представленного генератора можно увидеть на рисунке 2

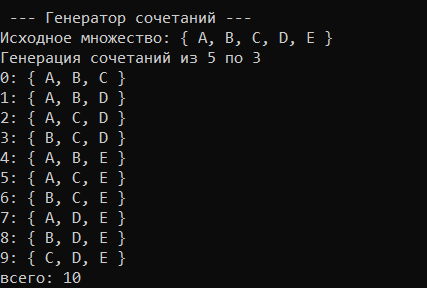


Рисунок 2. Демонстрация работы генератора

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

Для третьего задания был разработан генератор перестановок. Код представлен в листинге 5

|  |
| --- |
| #include "stdafx.h"  #include "Permutations.h"  #include <algorithm>  #define NINF ((short)0x8000)  namespace permutations  {  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 5. Код Permutations.cpp

В листинге 6 представлен заголовочный файл для Permutations.c

|  |
| --- |
| #pragma once  namespace permutations  {  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 6. Код Permutations.h

Результат представленного генератора можно увидеть на рисунке 3

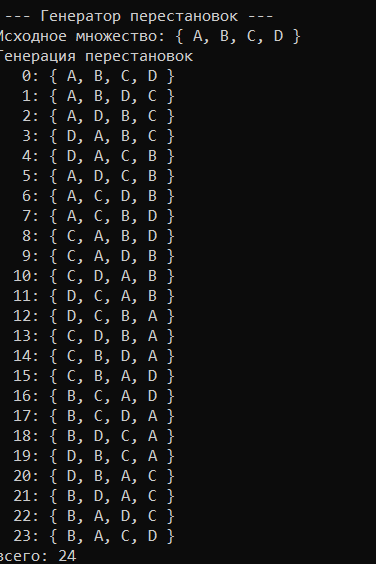


Рисунок 3. Демонстрация работы генератора

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

Для четвертого задания был разработан генератор размещений. Код представлен в листинге 7

|  |
| --- |
| #include "stdafx.h"  #include "Accomodation.h"  namespace accomodations  {  accomodation::accomodation(short n, short m)  {  this->n = n;  this->m = m;  this->cgen = new combi::xcombination(n, m);  this->pgen = new permutations::permutation(m);  this->sset = new short[m];  this->reset();  }  void accomodation::reset()  {  this->na = 0;  this->cgen->reset();  this->pgen->reset();  this->cgen->getfirst();  };  short accomodation::getfirst()  {  short rc = (this->n >= this->m) ? this->m : -1;  if (rc > 0)  {  for (int i = 0; i <= this->m; i++)  this->sset[i] = this->cgen->sset[this->pgen->ntx(i)];  };  return rc;  };  short accomodation::getnext()  {  short rc;  this->na++;  if ((this->pgen->getnext()) > 0) rc = this->getfirst();  else if ((rc = this->cgen->getnext()) > 0)  {  this->pgen->reset(); rc = this->getfirst();  };  return rc;  };  short accomodation::ntx(short i)  {  return this->sset[i];  };  unsigned \_\_int64 fact(unsigned \_\_int64 x) { return (x == 0) ? 1 : (x \* fact(x - 1)); };  unsigned \_\_int64 accomodation::count() const  {  return (this->n >= this->m) ?  fact(this->n) / fact(this->n - this->m) : 0;  };  } |

Листинг 7. Код Accomodation.cpp

В листинге 8 представлен заголовочный файл для Accomodation.cpp

|  |
| --- |
| // Combi.h  #include "Combi2.h"  #include "Combi3.h"  #pragma once  namespace combi  {  struct accomodation // генератор размещений  {  short n, // количество элементов исходного множества  m, // количество элементов в размещении  \* sset; // массив индесов текущего размещения  xcombination\* cgen; // указатель на генератор сочетаний  permutation\* pgen; // указатель на генератор перестановок  accomodation(short n = 1, short m = 1); // конструктор  void reset(); // сбросить генератор, начать сначала  short getfirst(); // сформировать первый массив индексов  short getnext(); // сформировать следующий массив индексов  short ntx(short i); // получить i-й элемент массива индексов  unsigned \_\_int64 na; // номер размещения 0, ..., count()-1  unsigned \_\_int64 count() const; // общее количество размещений  };  } |

Листинг 8. Код Accomodation.h

Результат представленного генератора можно увидеть на рисунке 4

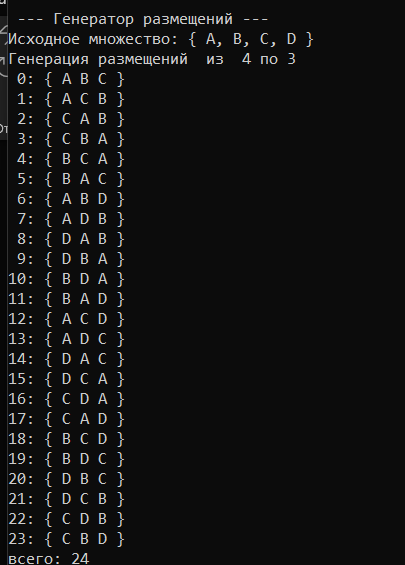


Рисунок 4. Демонстрация работы генератора

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

Решить упрощённую задачу о рюкзаке. Решение представлено в листинге 9

|  |
| --- |
| // Knapsack.cpp  #include "Knapsack.h"  #include "Set.h"  #include "Combi.h"  #include "Permutations.h"  #include "Accomodation.h"  #include "stdafx.h"  #define NINF 0x80000000 // самое малое int-число  //Множества  int calcv(sets::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(sets::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(sets::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}  )  {  sets::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;  };  void GetValue(int\* arr, int min, int max)  {  for (short i = 0; i < NN; i++)  {  arr[i] = min + rand() % (max - min + 1);  };  } |

Листинг 9. Код knapsack\_s.cpp

В листинге 10 представлен заголовочный файл для knapsack\_s.c

|  |
| --- |
| // Knapsack.h  #pragma once  //#define NN (sizeof(c)/sizeof(int))  #define NN 18  #define NN1 sizeof(c)/sizeof(int)  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  );  void GetValue(int\* arr, int min, int max); |

Листинг 10. Код knapsack\_s.h

Результат выполнения задачи о рюкзаке представлен на рисунке 5

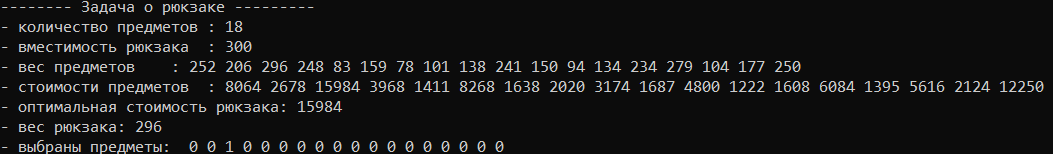


Рисунок 5. Результат работы алгоритма

**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:

Упрощённая задача о рюкзаке (количество предметов 12 – 20 шт.);

Результат работы алгоритма при разном количестве предметов, представлен на рисунке 6

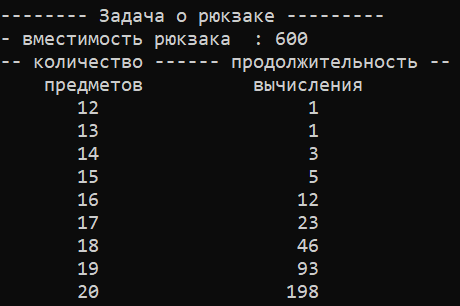


Рисунок 6. Результат работы алгоритма при разных значения

После получения данных о результате работы алгоритма, можно построить график, который отражает зависимость времени выполнения алгоритма при разных входных значениях. График представлен на рисунке 7.

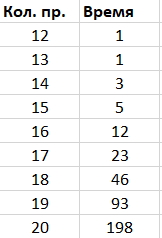


Рисунок 7. График зависимости процессорного времени от количества предметов

**Вывод**: Исходя из полученного результата можно судить, что зависимость экспоненциальная.

**Дополнение**

Решение задачи об оптимальной загрузке судна

**Условие**

об оптимальной загрузке судна с условием центровки (веса контейнеров сгенерировать случайным образом: количество мест на судне для контейнеров – 5, количество контейнеров 8, веса контейнеров 100 – 200 кг., доход от перевозки 10 – 100 у.е.; минимальный вес контейнера для каждого места 50 – 120 кг, максимальный вес контейнера для каждого места 150 – 850 кг)Код главного файла представлен в листинге 11

|  |
| --- |
| #include "stdafx.h"  #include "Boat.h"  #include "Accomodation.h"  namespace boatfnc  {  bool compv(accomodations::accomodation s, const int ming[],  const int maxg[], const int v[])  {  int i = 0;  while (i < s.m && v[s.ntx(i)] <= maxg[i] && v[s.ntx(i)] >= ming[i])i++;  return (i == s.m);  };  int calcc(accomodations::accomodation s, const int c[])  {  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)  {  for (int i = 0; i < m; i++) r1[i] = r2[i];  };  }  int boat\_с( // функция возвращает доход от перевозки контейнеров  short m, // [in] количество мест для контейнеров  int minv[], // [in] минимальный вес контейнера на каждом месте  int maxv[], // [in] максимальный вес коннтейнера каждом месте  short n, // [in] всего контейнеров  const int v[],// [in] вес каждого контейнера  const int c[],// [in] доход от перевозки каждого контейнера  short r[] // [out] номера выбранных контейнеров  )  {  accomodations::accomodation s(n, m);  int rc = 0, i = s.getfirst(), cc = 0;  while (i > 0)  {  if (boatfnc::compv(s, minv, maxv, v))  if ((cc = boatfnc::calcc(s, c)) > rc)  {  rc = cc; boatfnc::copycomb(m, r, s.sset);  }  i = s.getnext();  };  return rc;  }; |

Листинг 11. Код Boat.cpp

Заголовочный файл алгоритма представлен в листинге 12

|  |
| --- |
| #pragma once  #define SPACE(n) std::setw(n)<<" "  #define NN2 11  #define NN3 8  #define MM 5  int boat\_с(  short m, // [in] количество мест для контейнеров  int minv[], // [in] минимальный вес контейнера на каждом месте  int maxv[], // [in] максимальный вес контейнера на каждом месте  short n, // [in] всего контейнеров  const int v[],// [in] вес каждого контейнера  const int c[],// [in] доход от перевозки каждого контейнера  short r[] // [out] номера выбранных контейнеров  ); |

Листинг 12. Код Boat.h

Также для реализации алгоритма были использованы вспомогательные функции, представление в листинге 13

Результат алгоритма представлен на рисунке 8.

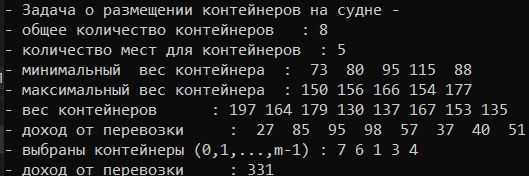


Рисунок 8. Результат работы

Проанализировать результат работы алгоритма при разном количестве контейнерах.

Результат работы представлены на рисунке 9.

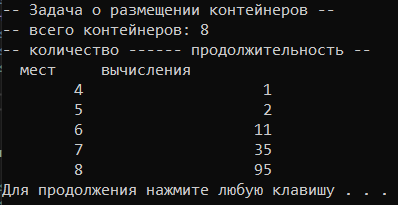


Рисунок 9. Результат работы

На основание полученных результатах, возможно построить график зависимости, времени выполнения алгоритма при разных кол-в контейнеров. График представлен на рисунке 10.



Рисунок 10. График

**Вывод**: Исходя из полученного результата можно судить, что зависимость экспоненциальная.