**Отчет по Лабораторной работе №2**

**Цель:** 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.

Разобран и разработан код генератора всех подмножества заданного множества из методического пособия.



|  |
| --- |
| #pragma once  namespace combi  {  struct subset // генератор множества всех подмножеств  {  short n, // кол-во эл-тов исх мн-ва < 64  sn, // кол-во эл-тов тек подмн-ва  \*sset; // массив индексов тек подмн-ва  unsigned \_\_int64 mask; // битовая маска  subset(short n = 1); // к-р (кол-во э-тов исх мн-ва)  short getfirst(); // сформир 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 подмн-в  void reset(); // сбросить генератор, начать сначала  };  }; |

Заголовочный файл для данного генератора

Шаблон структуры генератора множества всех подмножеств

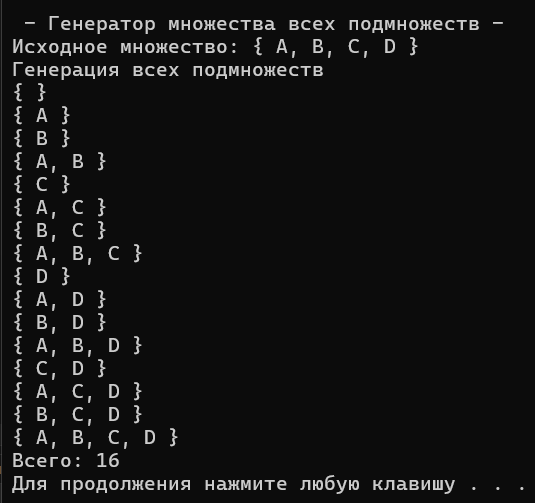
|  |
| --- |
| #include "stdafx.h"  #include "Combi.h"  #include <algorithm>  namespace combi  {  subset::subset(short n)  {  this->n = n; ///кол эл-тов исх массива  this->sset = new short[n]; ///массив индексов тек подмн-ва  this->reset();  };  void subset::reset() ///сбросить генератор  {  this->sn = 0; ///кол-во эл-тов тек подмн-ва  this->mask = 0; ///бит маска  };  short subset::getfirst() ///сформ массив индексов по бит маске  {  \_\_int64 buf = this->mask;  this->sn = 0;  for (short i = 0; i < n; i++)  {  if (buf & 0x1)  this->sset[this->sn++] = i;///массив индексов тек подмн-ва  buf >>= 1;  }  return this->sn;///кол-во эл-тов тек подмн-ва  };  short subset::getnext() ///++маска и сформ массив индексов  {  int rc = - 1;  this->sn = 0;  if (++this->mask < this->count())  rc = getfirst();  return rc;  };  short subset::ntx(short i) ///i-й эл-т массива индексов  {  return this->sset[i];  };  unsigned \_\_int64 subset::count() ///общее кол-во подмн-в  {  return (unsigned \_\_int64)(1<<this->n);  };  }; |

Исполнительный файл для данного генератора

Реализация методов структуры **subset**

|  |
| --- |
| #include <iomanip>  #include "stdafx.h"  #include <iostream>  #include "Combi.h"  using namespace std;  int main()  {  char AA[][2]= {"A", "B", "C", "D"};  cout << "\*\*\*\*\*\* Generator of all subsets \*\*\*\*\*\*\*" << endl;  cout << endl << "Primary array: ";  cout << "{ ";  for (int i = 0; i < sizeof(AA)/2; i++)  cout << AA[i] << ((i < sizeof(AA)/2-1)?", ":" ");  cout << "}" << endl;  cout << endl << "Generation of all subsets: ";  combi::subset s1(sizeof(AA)/2); // создание генератора  int n = s1.getfirst(); // первое (пустое) подмн-во    while (n >= 0) // пока есть подмн-ва  {  cout << endl << "{ ";  for (int i = 0; i < n; i++)  cout << AA[s1.ntx(i)] << ((i< n-1)?", ":" ");  cout << "}";  n = s1.getnext(); // cледующее подмн-во  };  cout << endl << "Vsego: " << s1.count() << endl;  system("pause");  return 0;  } |

Реализация генератора в функции main.



Вывод генератора на консоль

Этот алгоритм позволяет эффективно **генерировать все подмножества** множества размером n. В основе метода лежит **битовая маска**, где 1 означает, что элемент включен в текущее подмножество.

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

Разобран и разработан код генератора сочетаний для заданного множества из методического пособия.



|  |
| --- |
| 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  };  }; |

Заголовочный файл для данного генератора

Шаблон структуры генератора сочетаний

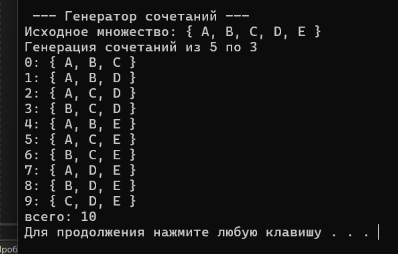
|  |
| --- |
| #include "stdafx.h"  #include "Combi.h"  #include <algorithm>  namespace combi  {  xcombination::xcombination (short n, short m)  {  this->n = n;  this->m = m;  this->sset = new short[m+2];  this->reset();  }  void xcombination::reset() // сбросить генератор, начать сначала  {  this->nc = 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;  };  }; |

Исполнительный файл для данного генератора

Реализация функций генератора сочетаний

|  |
| --- |
| #include "stdafx.h"  #include <iostream>  #include "Combi.h"  using namespace std;  int \_tmain(int argc, \_TCHAR\* argv[])  {  char AA[][2]= {"A", "B", "C", "D", "E"};  cout << "\*\*\*\*\*\* Generator of combinations \*\*\*\*\*\*\*";  cout << endl << "Primary array: ";  cout << "{ ";  for (int i = 0; i < sizeof(AA)/2; i++)  cout << AA[i] << ((i< sizeof(AA)/2-1)?", ":" ");  cout << "}" << endl;  cout << endl << "Generation of combinations ";  combi::xcombination xc(sizeof(AA)/2, 3);  cout << "iz " << xc.n << " po " << xc.m << ":";    int n = xc.getfirst();  while (n >= 0)  {  cout << endl << xc.nc << ": { ";  for (int i = 0; i < n; i++)  cout << AA[xc.ntx(i)] << ((i< n-1)?", ":" ");  cout << "}";  n = xc.getnext();  };  cout << endl << "Vsego: " << xc.count() << endl;  system("pause");  return 0;  } |

Реализация генератора в функции main.



Вывод генератора на консоль

Этот код **генерирует всевозможные комбинации** из m элементов множества из n элементов.



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

Разобран и разработан код генератора перестановок для заданного множества из методического пособия.



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

Заголовочный файл для данного генератора

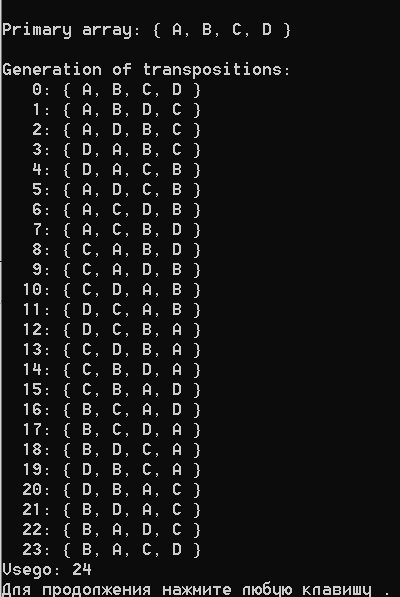
Шаблон структуры генератора перестановок

|  |
| --- |
| #include "stdafx.h"  #include "Combi.h"  #include <algorithm>  #define NINF ((short)0x8000)  using namespace std;  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)  {  swap(this->sset[idx],  this->sset[idx+(this->dart[idx]== L?-1:1)]);  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); };} |

Исполнительный файл для данного генератора

|  |
| --- |
| #include "stdafx.h"  #include <iostream>  #include "Combi.h"  #include <iomanip>  using namespace std;  int \_tmain(int argc, \_TCHAR\* argv[])  {  char AA[][2]= {"A", "B", "C", "D"};  cout << "\*\*\*\*\* Generator of transpositions \*\*\*\*\*" << endl;  cout << endl << "Primary array: ";  cout << "{ ";  for (int i = 0; i < sizeof(AA)/2; i++)    cout << AA[i] << ((i< sizeof(AA)/2-1)?", ":" ");  cout << "}" << endl;  cout << endl << "Generation of transpositions: ";  combi::permutation p(sizeof(AA)/2);  \_\_int64 n = p.getfirst();  while (n >= 0)  {  cout << endl << setw(4) << p.np << ": { ";  for (int i = 0; i < p.n; i++)  cout << AA[p.ntx(i)] << ((i< p.n-1)?", ":" ");  cout << "}";  n = p.getnext();  };  cout << endl << "Vsego: " << p.count() << endl;  system("pause");  return 0;  } |

Реализация генератора в функции main.



Вывод генератора на консоль

Этот код **генерирует все перестановки** P(n)P(n)P(n), то есть все возможные упорядоченные варианты расстановки nnn элементов. Он использует **метод Джонсона-Троттера**, который строит последовательность перестановок, изменяя порядок элементов **шаг за шагом**.



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

Разобрал и разработал код генератора всех размещений заданного множества из методического пособия.



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

Заголовочный файл для данного генератора

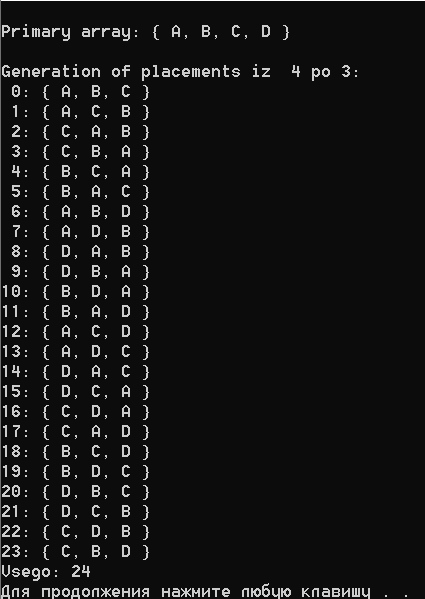
Шаблон структуры генератора размещений

|  |
| --- |
| #include "stdafx.h"  #include "Combi.h"  #include <algorithm>  #define NINF ((short)0x8000)  using namespace std;  namespace combi  {  accomodation::accomodation (short n, short m)  {  this->n = n;  this->m = m;  this->cgen = new xcombination(n,m);  this->pgen = new permutation(m);  this->sset = new short[m];  this->reset();  }  void accomodation::reset()  {  this->na = 0;  this->cgen->reset();  this->pgen->reset();  this->cgen->getfirst();  };  short accomodation::getfirst()  {  short rc = (this->n >= this->m)?this->m:-1;  if (rc > 0)  {  for (int i = 0; i <= this->m; i++)  this->sset[i] = this->cgen->sset[this->pgen->ntx(i)];  };  return rc;  };  short accomodation::getnext()  {  short rc;  this->na++;  if ((this->pgen->getnext())> 0)  rc = this->getfirst();  else if ((rc = this->cgen->getnext())> 0)  {  this->pgen->reset();  rc = this->getfirst();  };  return rc;  };  short accomodation::ntx(short i)  {  return this->sset[i];  };  unsigned \_\_int64 fact(unsigned \_\_int64 x)  {  return (x == 0)?1:(x\*fact(x-1));  };  unsigned \_\_int64 accomodation::count() const  {  return (this->n >= this->m)?  fact(this->n)/fact(this->n - this->m):0;  };  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)  {  swap(this->sset[idx],  this->sset[idx+(this->dart[idx]== L?-1:1)]);  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);  };  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; |

Исполнительный файл для данного генератора

|  |
| --- |
| #include "stdafx.h"  #include <iostream>  #include <iomanip>  #include "Combi.h"  #define N (sizeof(AA)/2)  #define M 3  using namespace std;  int \_tmain(int argc, \_TCHAR\* argv[])  {  char AA[][2]= {"A", "B", "C", "D"};  cout << "\*\*\*\*\*\* Generator of placement \*\*\*\*\*\*\*" << endl;  cout << endl << "Primary array: ";  cout << "{ ";    for (int i = 0; i < N; i++)  cout << AA[i] << ((i< N-1)?", ":" ");  cout << "}" << endl;  cout << endl << "Generation of placements iz " << N << " po " << M << ":";  combi::accomodation s(N,M);  int n = s.getfirst();    while (n >= 0)  {  cout << endl << setw(2) << s.na << ": { ";  for (int i = 0; i < 3; i++)  cout << AA[s.ntx(i)] << ((i< n-1)?", ":" ");  cout << "}";  n = s.getnext();  };  cout << endl << "Vsego: " << s.count() << endl;  system("pause");  return 0;  } |

Реализация генератора в функции main.



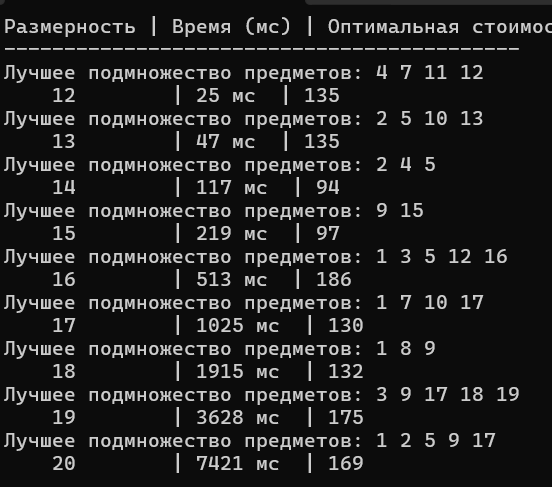
Вывод генератора на консоль

**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:

2, 6, 10, 14) упрощенную о рюкзаке (веса предметов и их стоимость сгенерировать случайным образом: вместимость рюкзака 300 кг, веса предметов 10 – 300 кг, стоимость предметов 5 – 55 у.е.; количество предметов – 18 шт.);

|  |
| --- |
| #include "backpack.h"  #include <iostream>  #include <cstdlib>  #include <ctime>  using namespace std;  // Функция полного перебора всех подмножеств  int knapsack\_s(int capacity, const vector<int>& weights, const vector<int>& values) {  int n = weights.size();  int best\_value = 0;  vector<int> best\_subset; // Сохраняем лучший набор предметов  // Перебираем все возможные подмножества (2^n)  for (int mask = 0; mask < (1 << n); ++mask) {  int total\_weight = 0, total\_value = 0;  vector<int> current\_subset; // Текущее подмножество  // Проверяем, какие предметы включены в текущее подмножество  for (int i = 0; i < n; ++i) {  if (mask & (1 << i)) { // Если предмет i включен в рюкзак  total\_weight += weights[i];  total\_value += values[i];  current\_subset.push\_back(i + 1); // Добавляем предмет в подмножество  }  }  // Если вес подмножества не превышает вместимость, обновляем максимум  if (total\_weight <= capacity && total\_value > best\_value) {  best\_value = total\_value;  best\_subset = current\_subset;  }  }  // Вывод лучшего найденного набора предметов  cout << "Лучшее подмножество предметов: ";  for (int item : best\_subset) {  cout << item << " ";  }  cout << "\n";  return best\_value;  }  // Функция генерации случайных предметов  void generate\_items(int n, vector<int>& weights, vector<int>& values) {  for (int i = 0; i < n; i++) {  weights[i] = 10 + rand() % 291;  values[i] = 5 + rand() % 51;  }  } |

Реализация генератора в функции main.



Вывод генератора на консоль

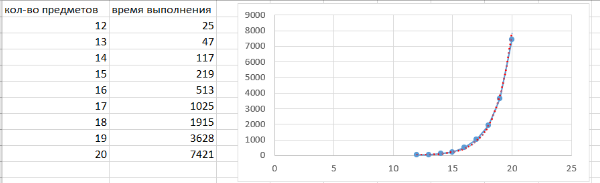


График зависимости времени от количества предметов.

Вывод: данная задача имеет экспоненциальное решение, зависимость времени вычислений от количества предметов близка к экспоненциальной. Этот алгоритм использует полный перебор всех возможных комбинаций предметов, что означает, что его временная сложность определяется количеством всех подмножеств множества из n элементов.

Поскольку у каждого предмета есть два состояния (включен в рюкзак или не включен), количество подмножеств равно:

Таким образом, временная сложность алгоритма: O(2^n)