Лабораторная работа 2. Комбинаторные алгоритмы решения оптимизационных задач

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

ЗАДАНИЕ ДЛЯ ВЫПОЛНЕНИЯ:

Задание 1. Разработать генератор подмножеств заданного множества.
Задание 2. Разработать генератор сочетаний
Задание 3. Разработать генератор перестановок
Задание 4. Разработать генератор размещений

Задание 5. Решить в соответствии с вариантом задачу:

1, 5, 9, 13) коммивояжера (расстояния сгенерировать случайным образом: 10 городов, расстояния 10 – 300 км, 3 расстояния между городами задать бесконечными);

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.);

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.);

4, 8, 12, 16) 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);

Задание 6. Исследовать зависимость времени вычисления необходимое для решения задачи (в соответствии с вариантом) от размерности задачи:

1, 5, 9, 13) коммивояжера (5–12 городов);

2, 6, 10, 14) упрощенную о рюкзаке (количество предметов 12 – 20 шт.);

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

4, 8, 12, 16) об оптимальной загрузке судна с условием центровки (количество мест на судне для контейнеров 4 – 8);

ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ:

1. Генерация подмножеств заданного множества

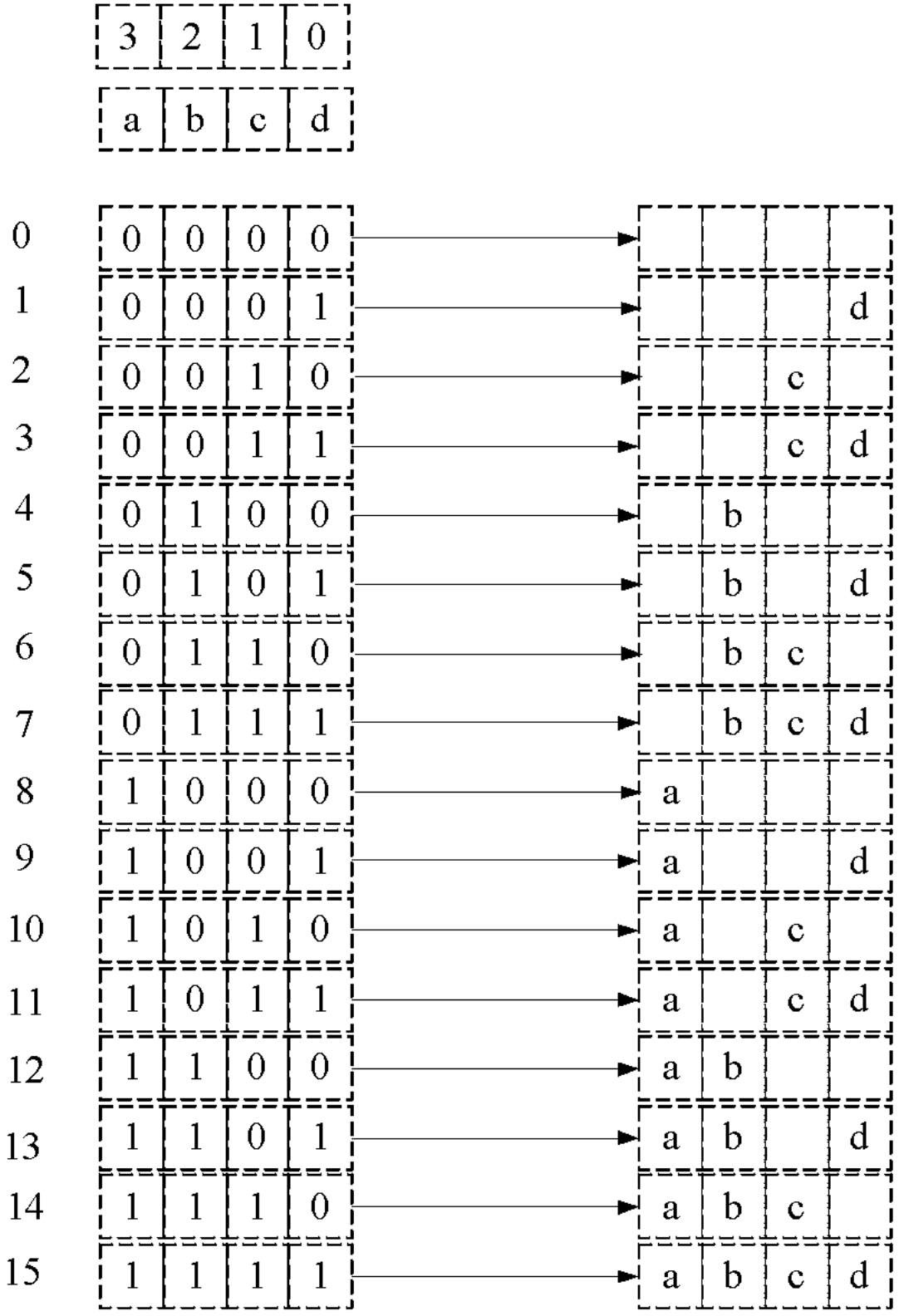

Рис. 1. Генерация множества всех подмножеств

```
// 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
      };
};
```

Рис. 2. Шаблон структуры генератора множества всех подмножеств

```
// Combi.cpp
#include "stdafx.h"
#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);};
};
```

Рис. 3. Реализация методов структуры **subset**

```
// Main
#include "stdafx.h"
#include <iostream>
#include "Combi.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  ";
 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();                    // следующее подмножество
 };
 std::cout<<std::endl<<"всего: " << s1.count()<<std::endl;
 system("pause");
 return 0;
}
```

Рис. 4. Пример применения генератора множества всех подмножеств

Решение упрощенной задачи о рюкзаке с помощью генератора множества всех подмножеств

На рис. 6 изображена схема решения задачи о рюкзаке с применением генератора множества всех подмножеств. Задача имеет следующие исходные данные:

$V = 100$ – вместимость (объем) рюкзака;

$n = 4$ – количество предметов;

$(25, 30, 60, 20)$ – вектор объемов предметов;

$(25, 10, 20, 30)$ – вектор стоимостей предметов.

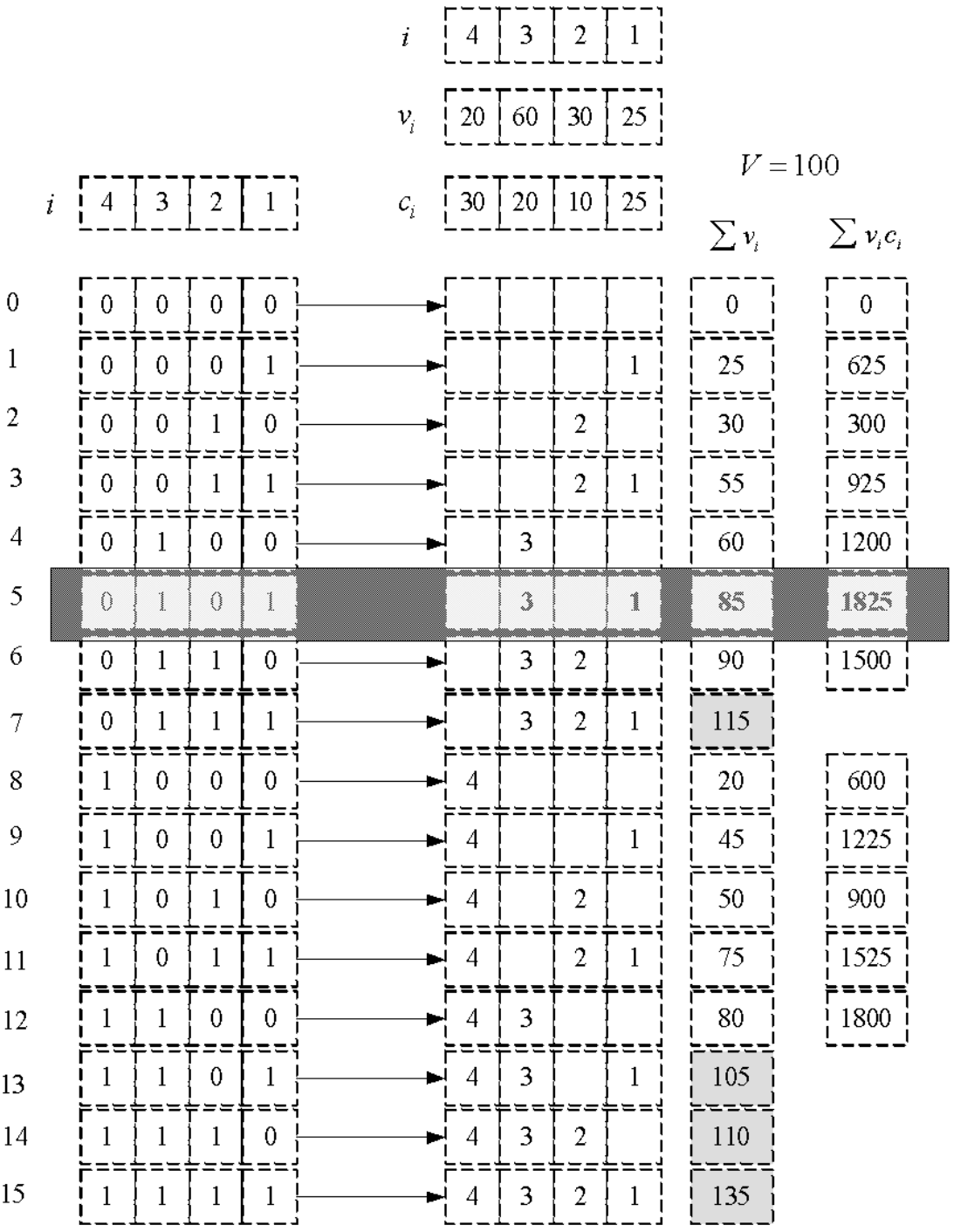

Рис. 6. Схема решения задачи о рюкзаке с применением генератора множества всех подмножеств

На рис. 7 и 8 представлен пример реализации функции **knapsack_s** на языке C++, которая решает задачу о рюкзаке.

```
// Knapsack.h
#pragma once
#include "Combi.h"

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
                );
```

Рис.7. Прототип функции **knapsack_s**

```
// Knapsack.cpp
#include "stdafx.h"
#include "Knapsack.h"
#define NINF 0x80000000     // самое малое int-число
int calcv(combi::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(combi::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(combi::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}
                )
{
 combi::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;
};
```

Рис.8. Реализация функции **knapsack_s**

На рис. 9 приведен пример вызова функции **knapsack_s** для решения задачи о рюкзаке с исходными данными для схемы, представленной на рис. 6.

```
// Main
#include "stdafx.h"
#include <iostream>
#include "Combi.h"
#include "Knapsack.h"
#define NN 4
int _tmain(int argc, _TCHAR* argv[])
{
 setlocale(LC_ALL, "rus");
 int V = 100,                  // вместимость рюкзака
 v[] = {25, 30, 60, 20},       // размер предмета каждого типа
 c[] = {25, 10, 20, 30};       // стоимость предмета каждого типа
 short m[NN];                  // количество предметов каждого типа
                                  {0,1}

 int maxcc =  knapsack_s(

                          V,    // [in]  вместимость рюкзака
                          NN,   // [in]  количество типов предметов
                          v,    // [in]  размер предмета каждого типа
                          c,    // [in]  стоимость предмета каждого
типа
                          m     // [out] количество предметов каждого
типа
                        );

 std::cout<<std::endl<<"-------- Задача о рюкзаке --------- ";
 std::cout<<std::endl<<"- количество предметов : "<< NN;
 std::cout<<std::endl<<"- вместимость рюкзака  : "<< V;
 std::cout<<std::endl<<"- размеры предметов    : ";
         for(int i = 0; i < NN; i++) std::cout<<v[i]<<" ";
 std::cout<<std::endl<<"- стоимости предметов  : ";
         for(int i = 0; i < NN; i++) std::cout<<v[i]*c[i]<<" ";
 std::cout<<std::endl<<"- оптимальная стоимость рюкзака: " << maxcc;
 std::cout<<std::endl<<"- вес рюкзака: ";
        int s = 0; for(int i = 0; i < NN; i++) s+= m[i]*v[i];
 std::cout<<s;
 std::cout<<std::endl<<"- выбраны предметы: ";
        for(int i = 0; i < NN; i++) std::cout<<" "<<m[i];
 std::cout<<std::endl<<std::endl;

 system("pause");
 return 0;
}
```

Рис. 9. Пример использования функции **knapsack_s**

Оценить зависимость продолжительности вычисления оптимальной комбинации предметов от их общего количества можно с помощью программы, изображенной на рис. 11.

```
// Main
#include "stdafx.h"
#include <iostream>
#include "Combi.h"
#include "Knapsack.h"
#include <time.h>
#include <iomanip>
#define NN (sizeof(c)/sizeof(int))
int _tmain(int argc, _TCHAR* argv[])
{
 setlocale(LC_ALL, "rus");
 int V = 600,                // вместимость рюкзака
 v[] = {25, 56, 67, 40, 20, 27, 37, 33, 33, 44, 53, 12,
        60, 75, 12, 55, 54, 42, 43, 14, 30, 37, 31, 12},
 c[] = {15, 26, 27, 43, 16, 26, 42, 22, 34, 12, 33, 30,
        12, 45, 60, 41, 33, 11, 14, 12, 25, 41, 30, 40};
 short m[NN];
 int maxcc = 0;
 clock_t t1, t2;
 std::cout<<std::endl<<"-------- Задача о рюкзаке --------- ";
 std::cout<<std::endl<<"- вместимость рюкзака  : "<< V;
 std::cout<<std::endl<<"-- количество ------ продолжительность -- ";
 std::cout<<std::endl<<"    предметов          вычисления  ";
 for (int i =  14; i <= NN; i++)
 {
 t1 = clock();
       maxcc =  knapsack_s(V, i, v, c, m );
 t2 = clock();
 std::cout<<std::endl<<"       "<<std::setw(2)<<i
                 <<"              "<<std::setw(5)<<(t2-t1);
 }
 std::cout<<std::endl<<std::endl;
 system("pause");
 return 0;
}
```

Рис. 11. Вычисление продолжительности решения задачи о рюкзаке при различном количестве предметов

2. Генерация сочетаний

На рис. 14 представлена схема построения множества сочетаний $C_{X,3}$ из элементов множества $X = \{a, b, c, d\}$. Закрашенным прямоугольником на рисунке обозначены номера (индексы) элементов битовых последовательностей B_i, $i = \overline{0,15}$ и элементов множества X. Стрелки связывают битовые последовательности, содержащие три двоичные единицы и сгенерированные сочетания множества $C_{X,3}$. Для каждой стрелки указаны индексы единичных позиций соответствующих битовых последовательностей. Эти индексы используются для выбора элементов из множества X для включения в соответствующее сочетание. Очевидно, что

такой алгоритм генерации сочетаний имеет сложность $O(2^{|X|})$, как и алгоритм генерации множества всех подмножеств.

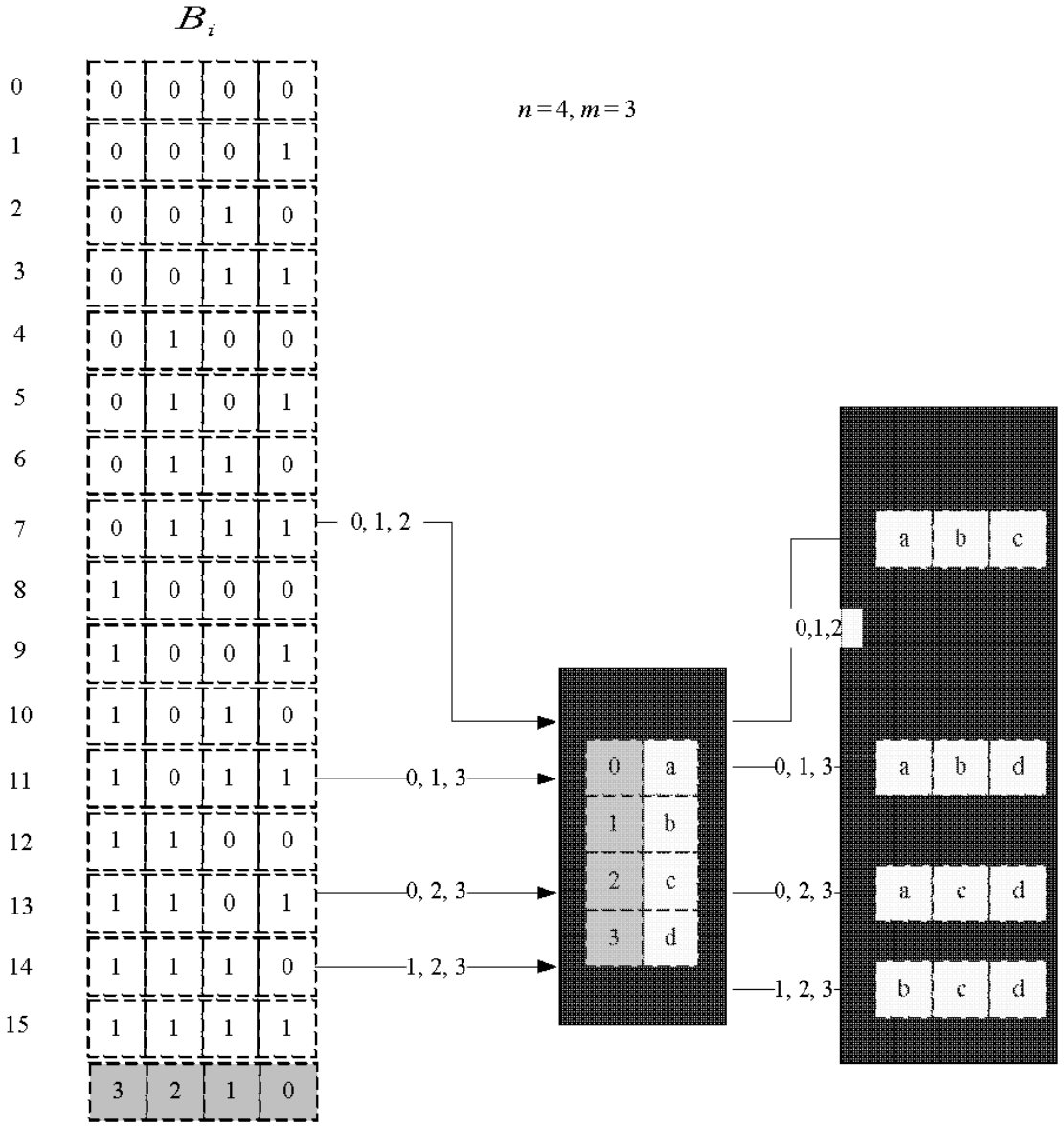

Рис.14. Схема генерации сочетаний на основе множества всех подмножеств

На рис. 15 и 16 представлена реализация генератора сочетаний на языке C++. Генератор реализован в виде структуры **xcombination**.

```
// 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
  };
};
```

Рис. 15. Шаблон структуры генератора сочетаний

```
// Combi.cpp
#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)?
```

Рис. 16. Реализация функций генератора сочетаний

```
// Main
#include "stdafx.h"
#include <iostream>
#include "Combi.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 ";
 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;
}
```

Решение задачи об оптимальной загрузке судна на основе генератора сочетаний

На рис. 19 изображена схема решения задачи с применением генератора подмножеств. Задача имеет следующие исходные данные:

$V = 1000$ – ограничение по общему весу контейнеров;

$n = 6$ – количество контейнеров;

$m = 3$ – количество свободных мест на палубе;

$(100, 200, 300, 400, 500, 150)$ – вес контейнеров;

$(10, 15, 20, 25, 30, 25)$ – доход от перевозки контейнеров.

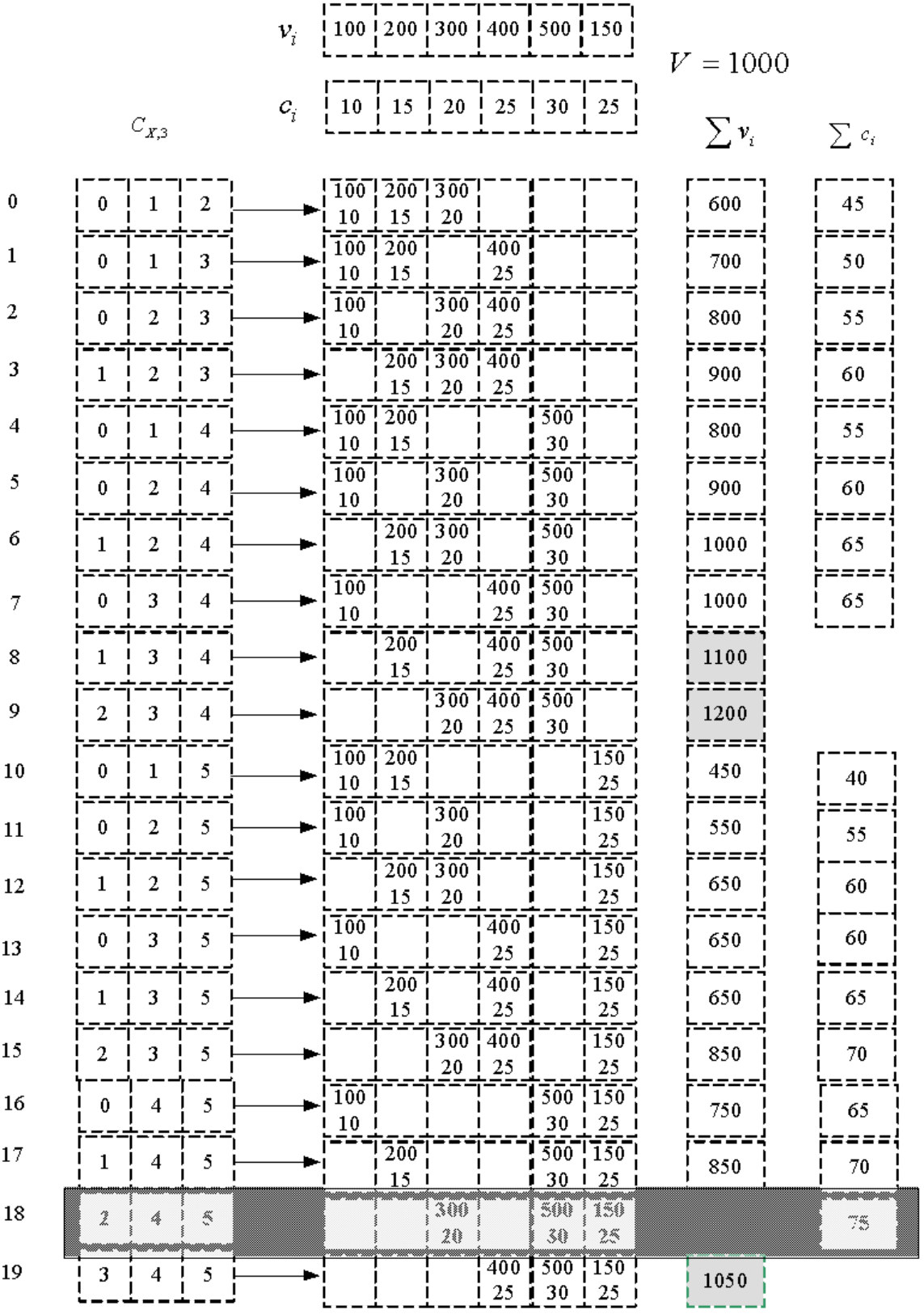

Рис. 19. Схема решения задачи об оптимальной загрузке судна

На рис. 20 и 21 представлен пример реализации на языке C++ функции **boat**, решающей задачу об оптимальной загрузке судна.

```
// --- Boat.h
// -- решение  задачи об оптимальной загрузке судна
//    функция возвращает доход  от перевози выбранных контейнеров
#pragma once
#include "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
         );
```

Рис. 20. Функция **boat**, решающая задачу об оптимальной загрузке судна

```
// --- Boat.cpp
#include "stdafx.h"
#include "Boat.h"
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;
};
```

Рис. 21. Реализация функции **boat**

```
// --- Main
#include "stdafx.h"
#include <iostream>
#include <iomanip>
#include "Boat.h"
#define NN (sizeof(v)/sizeof(int))
#define MM 3
int _tmain(int argc, _TCHAR* argv[])
{
 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;
}
```

Рис. 22. Пример решения задачи об оптимальной загрузке судна

На рис. 24 представлена программа, с помощью которой можно оценить продолжительность решения задачи о загрузке судна при разном количестве контейнеров. В программе фиксируется значение параметра **m** (количество мест для контейнеров) и вычисляется продолжительность работы функции boat в зависимости от параметра n (общее количество контейнеров).

```
// --- Main
#include "stdafx.h"
#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 _tmain(int argc, _TCHAR* argv[])
{
 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;
}
```

Рис. 24. Вычисление продолжительности решения задачи о загрузке судна при разном количестве контейнеров

3. Генерация перестановок

Схема алгоритма генерации множества всех перестановок множества $X = \{a, b, c, d\}$ приведена на рис. 1.

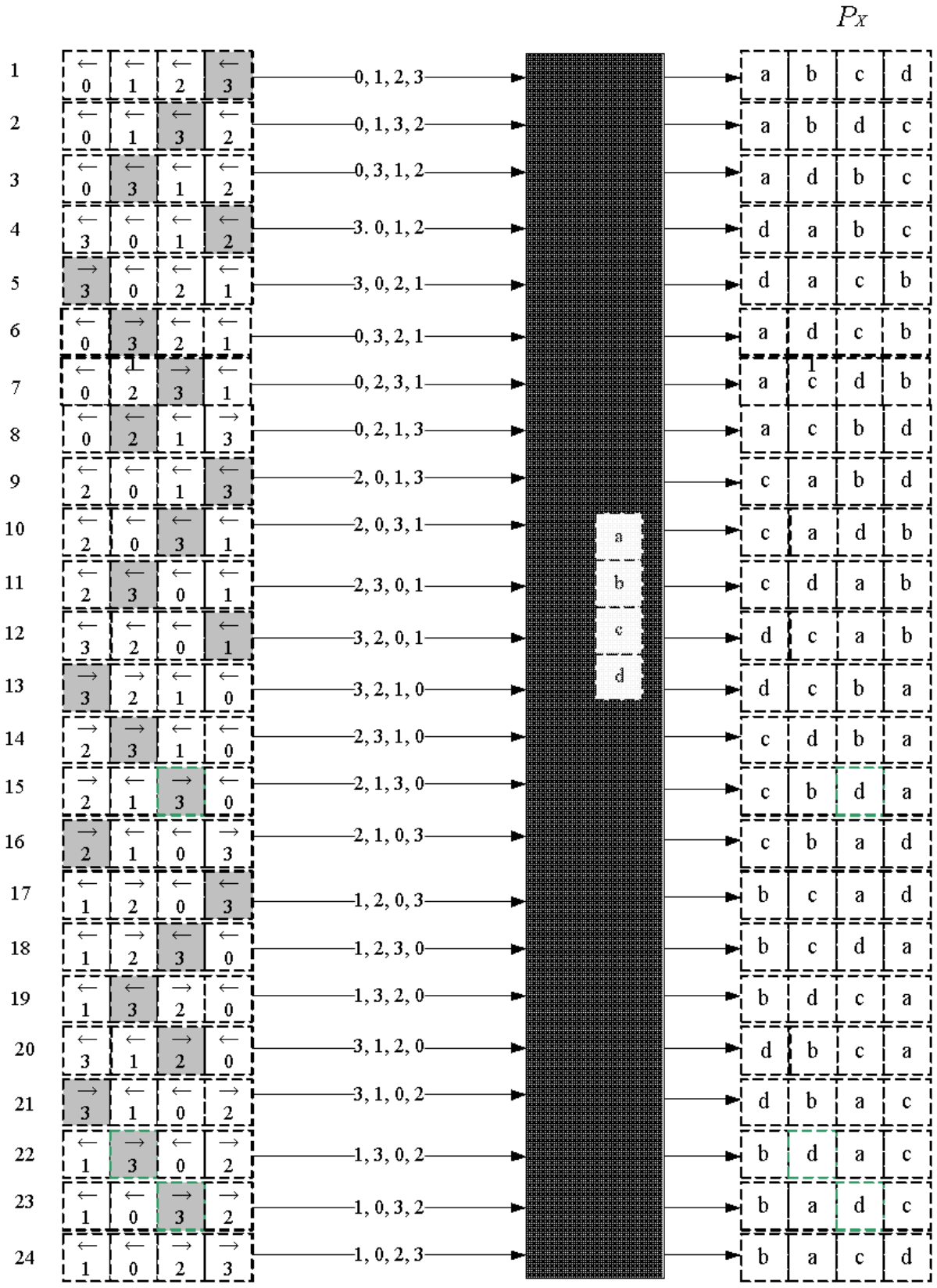

Рис. 4.1. Схема работы алгоритма Джонсона – Троттера

Реализация генератора перестановок на языке C++

На рис. 2 и 3 представлена программная реализация генератора перестановок.

```
// 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 перестановки 0,... count()-1

unsigned __int64 count() const;  // вычислить общее кол. перестановок
  };
};
```

Рис. 2. Шаблон структуры генератора перестановок

```
// Combi.cpp
#include "stdafx.h"
#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;
```

Рис. 3. Реализация функций генератора перестановок

```
// --- Main
#include "stdafx.h"
#include <iostream>
#include "Combi.h"
#include <iomanip>
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 ";
 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;
}
```

Рис. 4. Пример применения генератора перестановок

На рис. 6 изображена схема решения задачи коммивояжера с применением генератора перестановок. Задача решается для пяти городов.

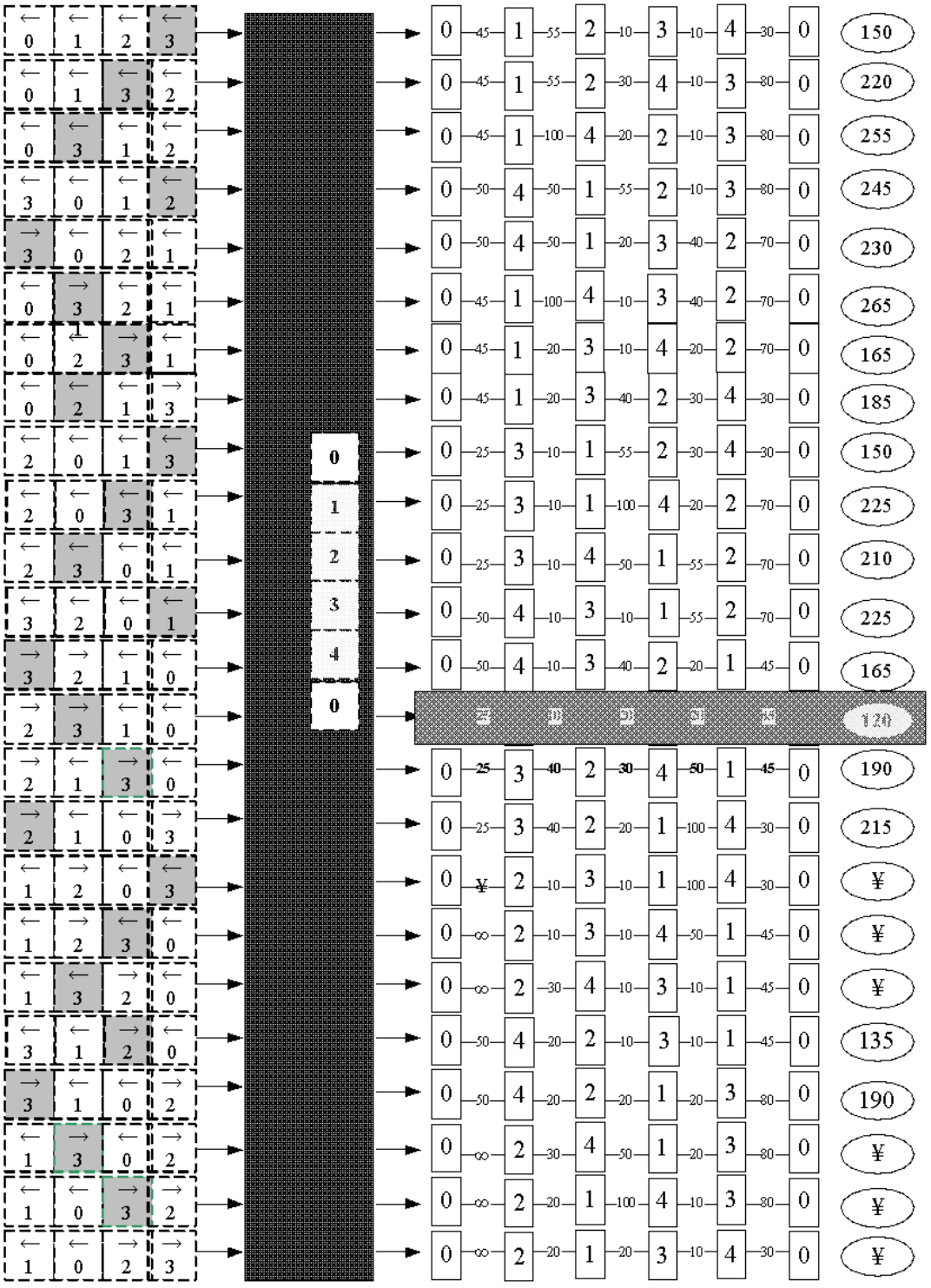

Рис. 6. Схема решения задачи коммивояжера

Расстояние между городами задается следующей матрицей A:

$$A = (a_{ij})_{5\times5} = \begin{pmatrix} 0 & 45 & \infty & 25 & 50 \\ 45 & 0 & 55 & 20 & 100 \\ 70 & 20 & 0 & 10 & 30 \\ 80 & 10 & 40 & 0 & 10 \\ 30 & 50 & 20 & 10 & 0 \end{pmatrix}.$$

На рис. 7 и 8 представлен пример реализации на C++ функции **salesman**, вычисляющей оптимальный кольцевой маршрут коммивояжера.

```
//-- 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

  );
```

Рис. 7. Функция **salesman**, решающая задачу коммивояжера

```
// -- Salesman.cpp
#include "stdafx.h"
#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;
 }
```

Рис. 8. Реализация функции **salesman**

На рис. 9 и 10 приведен пример вызова функции **salesman** для решения задачи с исходными данными к схеме на рис. 6.

```
// --- main
#include "stdafx.h"
#include <iostream>
#include <iomanip>
#include "Salesman.h"
#define N 5
int _tmain(int argc, _TCHAR* argv[])
{
 setlocale(LC_ALL, "rus");
 int d[N][N] = { //0   1    2    3     4
                {  0,  45, INF,  25,   50},    //  0
                { 45,   0,  55,  20,  100},    //  1
                { 70,  20,   0,  10,   30},    //  2
                { 80,  10,  40,   0,   10},    //  3
                { 30,  50,  20,  10,    0}};   //  4
int r[N];                        // результат
int s = salesman (
            N,          // [in]  количество городов
      (int*)d,          // [in]  массив [n*n] расстояний
            r           // [out] массив [n] маршрут 0 x x x x

      );
 std::cout<<std::endl<<"-- Задача коммивояжера -- ";
 std::cout<<std::endl<<"-- количество  городов: "<<N;
 std::cout<<std::endl<<"-- матрица расстояний : ";
 for(int i = 0; i < N; i++)
  {
    std::cout<<std::endl;
    for (int j = 0; j < N; j++)

if (d[i][j]!= INF) std::cout<<std::setw(3)<<d[i][j]<< " ";

else std::cout<<std::setw(3)<<"INF"<<" ";
  }
 std::cout<<std::endl<<"-- оптимальный маршрут: ";
 for(int i = 0; i < N; i++) std::cout<<r[i]<<"-->"; std::cout<<0;
 std::cout<<std::endl<<"-- длина маршрута     : "<<s;
 std::cout<<std::endl;
 system("pause");
return 0;
}
```

Рис. 9. Пример решения задачи коммивояжера

На рис. 11 представлена программа, позволяющая оценить продолжительность решения задачи коммивояжера в зависимости от количества городов.

```
// -- main
#include "stdafx.h"
#include "Auxil.h"
#include <iostream>
#include <iomanip>
#include <time.h>
#include "Salesman.h"
#define SPACE(n) std::setw(n)<<" "
#define N 12
int _tmain(int argc, _TCHAR* argv[])
{
 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;
}
```

Рис. 11. Вычисление продолжительности решения задачи коммивояжера при разном количестве городов

4. Генерация размещений

На рис. 1 представлена схема построения множества размещений $A_{X,3}$ из элементов множества $X = \{a, b, c, d\}$.

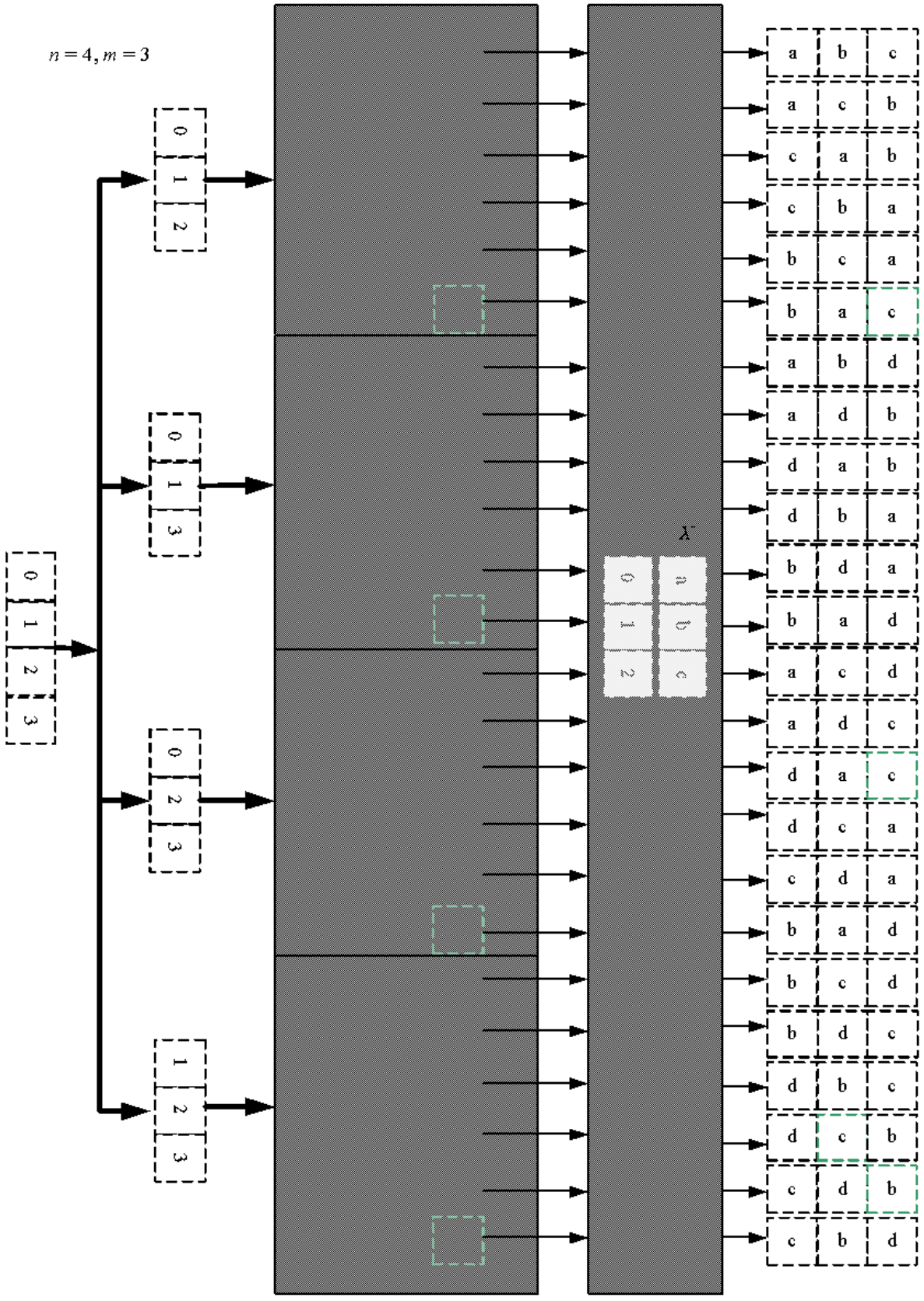

Рис.1. Схема генерации размещений

Реализация генератора размещений на языке C++

```
// Combi.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
  };
}
```

```
};
```

Рис.2. Шаблон структуры генератора размещений

```
// Combi.cpp
#include "stdafx.h"
#include "Combi.h"
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
  {
```

Рис. 3. Реализация функций генератора размещений

```
// --- main
#include "stdafx.h"
#include <iostream>
#include <iomanip>
#include "Combi.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;
 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 < 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;
}
```

Рис.4. Пример использования генератора перестановок

Решение задачи об оптимальном размещении контейнеров на судне с помощью генератора размещений

На рис. 6 изображена схема, поясняющая решение этой задачи с помощью генератора размещений. Задача имеет следующие исходные данные:

$n = 4$ – общее количество контейнеров;

$m = 3$ – количество свободных мест на палубе судна;

$\{100, 200, 300, 400\}$ – вес контейнеров $(v_i, i = \overline{1, 4})$;

$\{10, 15, 20, 25\}$ – доход от перевозки контейнеров $(c_i, i = \overline{1, 4})$;

$\{350, 250, 0\}$ – минимальный вес контейнеров ($(\coprod_i, i = \overline{1, 3})$;

$\{750, 350, 750\}$ – максимальный вес контейнеров $(\coprod v_i, i = \overline{1, 3})$.

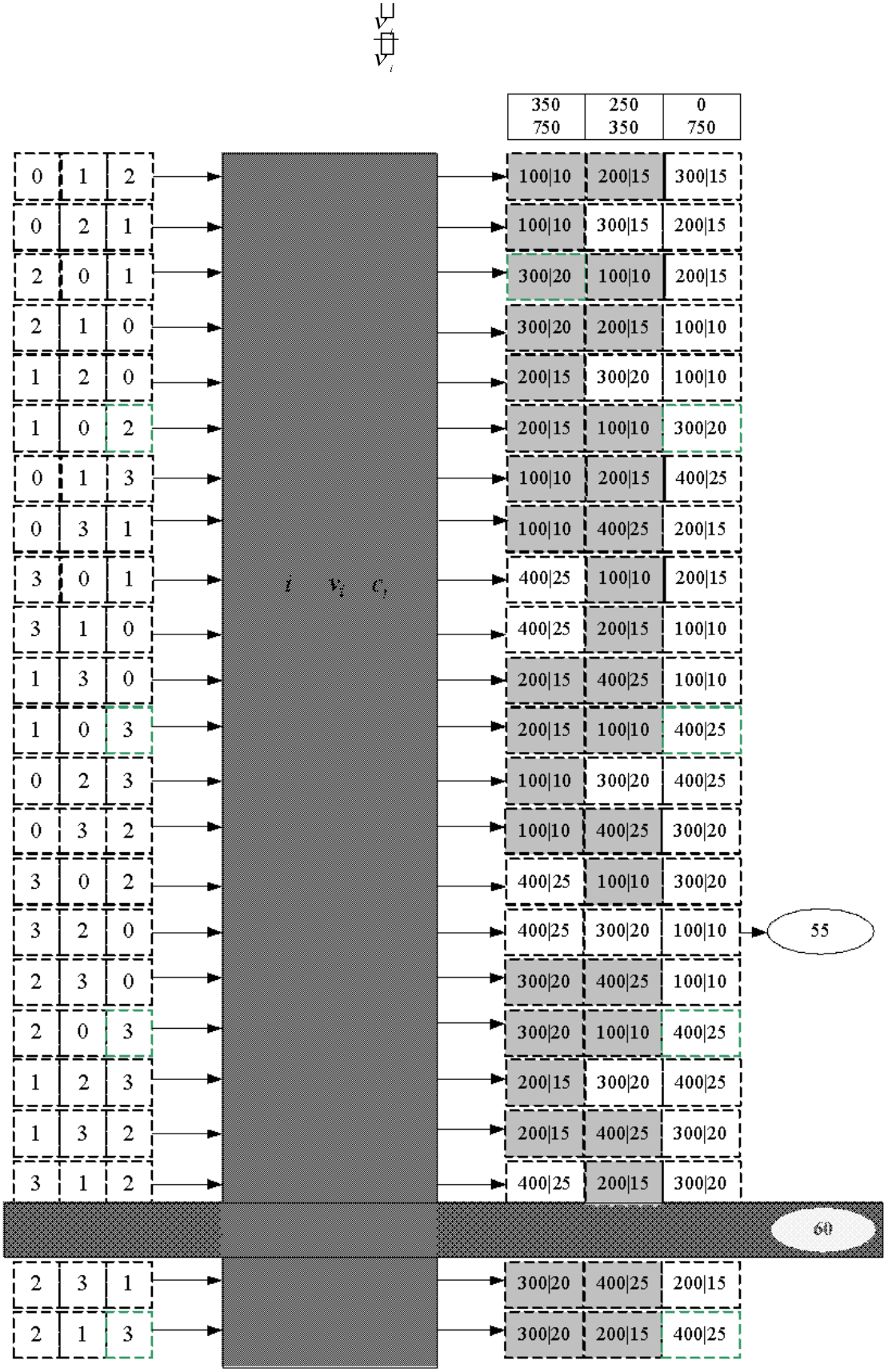

Рис. 6. Схема решения задачи об оптимальном размещении контейнеров на судне

```
// --- Boat.h
// -- решение  задачи об оптимальном  размещении контейнеров
//  функция возвращает доход  от перевозки выбранных контейнеров
int boat_c(
      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
    );
```

Рис. 7. Функция **boat_c**, решающая задачу об оптимальном размещении контейнеров на судне

```
// --- Boat.cpp
#include "stdafx.h"
#include "Boat.h"
#include "Combi.h"
namespace boatfnc
{
 bool compv( combi::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(combi::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_c(      // 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 конntейнера 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
    )
  {
  combi::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;
```

Рис. 8. Реализация функции **boat_c**

```
// -- main (решение задачи  о размещении контейнеров)
#include "stdafx.h"
#include <iostream>
#include <iomanip>
#include "Boat.h"
#define NN (sizeof(v)/sizeof(int))
#define MM 3
int _tmain(int argc, _TCHAR* argv[])
{
 setlocale(LC_ALL, "rus");
 int v[] =   {100,  200, 300,  400}; // вес
 int c[] =   { 10, 15,  20, 25}; // доход
 int minv[]  = {350,  250,  0};    // минимальный  вес
 int maxv[]  = {750,  350,  750};    // максимальный вес
 short r[MM];
 int cc = boat_c(
   MM,    // [in]  количество мест для контейнеров
   minv,  // [in]  максимальный вес контейнера на каждом  месте
   maxv,  // [in]  минимальный вес контейнера на каждом  месте
   NN,    // [in]  всего контейнеров
   v,     // [in]  вес каждого контейнера
   c,     // [in]  доход от перевозки каждого контейнера
   r      // [out] номера  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<<"- минимальный  вес контейнера  : ";
  for(int i = 0; i < MM; i++) std::cout<<std::setw(3)<<minv[i]<<" ";
std::cout<<std::endl<<"- максимальный вес контейнера  : ";
  for(int i = 0; i < MM; i++) std::cout<<std::setw(3)<<maxv[i]<<" ";
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<<std::endl;
system("pause");
return 0;
}
```

Рис. 9. Пример решения задачи об оптимальном размещении контейнеров на судне

На рис. 11 представлена программа, позволяющая оценить продолжительность решения задачи о размещении контейнеров в зависимости от количества свободных мест на палубе судна

```
#include "stdafx.h"
#include <iostream>
#include <iomanip>
#include <time.h>
#include "Auxil.h"
#include "Boat.h"
#define SPACE(n) std::setw(n)<<" "
#define NN 11
int _tmain(int argc, _TCHAR* argv[])
{
 setlocale(LC_ALL, "rus");
 int v[NN+1], c[NN+1], minv[NN+1], maxv[NN+1];
 short r[NN];
 auxil::start();
 for(int i = 0; i <= NN; i++)
 {
 v[i] = auxil::iget(50,500); c[i] = auxil::iget(10,30);
 minv[i] = auxil::iget(50,300); maxv[i] = auxil::iget(250,750);
 }
 std::cout<<std::endl<<"-- Задача о размещении контейнеров -- ";
 std::cout<<std::endl<<"-- всего контейнеров: " << NN;
 std::cout<<std::endl<<"-- количество ------ продолжительность -- ";
 std::cout<<std::endl<<"  мест     вычисления  ";
 clock_t t1, t2;
 for (int i = 4; i < NN; i++)
 {
 t1 = clock();
 boat_c(i, minv,  maxv, NN,  v,  c, r);
 t2 = clock();
 std::cout<<std::endl<<SPACE(7)<<std::setw(2)<<i
       <<SPACE(15)<<std::setw(6)<<(t2-t1);
 }
 std::cout<<std::endl; system("pause");
 return 0;
}
```

Рис. 11. Оценка продолжительности решения задачи о размещении контейнеров на судне