**Лекция 4**

**КОМБИНАТОРНЫЕ МЕТОДЫ РЕШЕНИЯ ОПТИМИЗАЦИОННЫХ ЗАДАЧ**

**4. Генерация размещений**

Множество размещений  из  по  можно получить, перестановкой элементов всех сочетаний множества . Другими словами, для получения множества размещений  требуется сначала сгенерировать все сочетания по  элементов из множества  а затем все перестановки элементов для каждого сочетания.

На рис. 1 представлена схема построения множества размещений  из элементов множества 

Генерация размещений осуществляется в три этапа:

1. Для множества  формируется множество всех сочетаний по три элемента. Таких сочетаний будет 

2. Для элементов каждого сочетания ,  генерируется все перестановки  Каждому сочетанию будет соответствовать  перестановок. Таким образом, всего будет построено  перестановок, которые в совокупности представляют собой все размещения множества 

3. Применяя полученные на предыдущем шаге размещения в качестве индексов для элементов множества  формируется множество  всех размещений по 3.

Для того чтобы проследить этапы генерации множества  на схеме, ее следует рассматривать слева направо. В крайней левой позиции изображено множество , на основе которого формируются четыре сочетания.

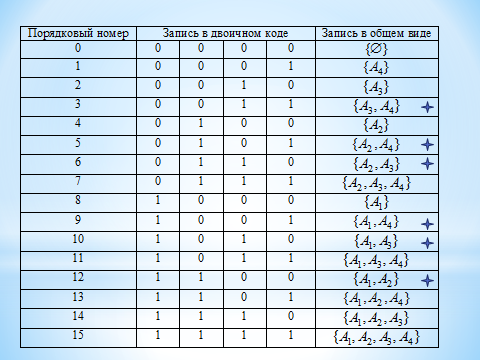
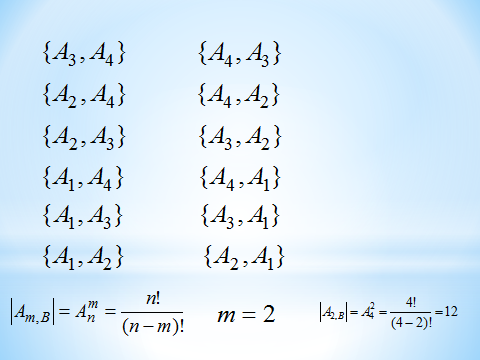
Затем каждое сочетание рассматривается как отдельное множество, состоящее из трех элементов. Для каждого множества формируется по  перестановок. В итоге в третьем слева столбце на схеме отображены  перестановки множества 

На последнем этапе формируется множество  всех перестановок элементов множества  Элементы множества  на рис. 1 отображены в крайнем справа столбце.



Рис.1. Схема генерации размещений

Пример. Имеется множество из 4-х элементов. Необходимо получить все возможные размещения по 2 элемента. Сначала получаем множество всех подмножеств – 16. Затем берем все сочетания по 2 – их будет 6, а потом получившиеся сочетания переставляем – их получается 12.



**Реализация генератора размещений на языке С++**

На рис. 2 и 3 представлена реализация программы генератора перестановок, а на рис. 4 и 5 приведен пример построения множества размещений с помощью этой программы и результат выполнения программы соответственно.

// 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**

**{**

**return (this->n >= this->m)?**

**fact(this->n)/fact(this->n - this->m):0;**

**};**

**}**

**};**

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

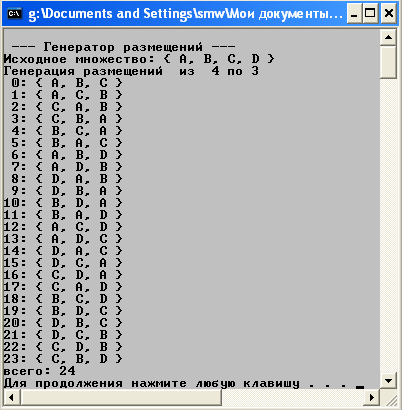
****

Рис. 5. Результат выполнения программы представленной на рис. 4

Как и в предыдущих случаях, генератор размещений тоже реализован в виде структуры. Структура **accomodation** имеет один конструктор. С помощью двух параметров конструктору передается размерность исходного множества (параметр **n**) и размерность генерируемых размещений (параметр **m**).

В своей работе генератор размещений использует два встроенных генератора: сочетаний (указатель **cgen**) и перестановок (указатель **pgen**).

Текущее состояние генератора определяется состоянием используемых генераторов, а также значением четырех переменных: **n** (количество элементов исходного множества), **m** (размерность генерируемых размещений), **sset** (указатель на массив индексов) и **na** (номер текущего размещения). Все переменные, включая указатели генераторов, инициализируются в конструкторе. Значение **na** увеличивается на единицу после генерации очередного размещения. Элементы масcива **sset** меняются при каждом цикле работы генератора.

Кроме конструктора, структура **accomodation** содержит еще пять функций.

Функция **getfirst** не имеет параметров и предназначена для формирования первого размещения. Первое размещение совпадает с первым сочетанием, сформированным функцией **getfirst** встроенного генератора сочетаний.

Функция **reset** позволяет сбросить текущее состояние генератора для того, чтобы начать его работу сначала. Функция выполняет сброс встроенных генераторов (функция **reset**), устанавливает значение нумератора размещений (переменная **na**) в нуль и выполняет функцию **getfirst** встроенногогенератора сочетаний.

Функция **getnext** формирует массив индексов следующего размещения и увеличивает значение переменной **na** на единицу. В своей работе функция использует функции **getnext** или **getfirst** встроенных генераторов, а также функцию **getfirst** генератора размещений.

Функция **ntx** возвращает значение элемента массива индексов по индексу этого элемента и служит для сокращения записи при переборе элементов массива.

Функция **count** вычисляет и возвращает общее количество размещений из **n** по **m** элементов.

Пример использования генератора размещений демонстрирует формирование множества размещений по три из четырех элементов массива.

Исходные данные примера аналогичны данным схемы на рис. 1.

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

Изменим формулировку задачи об оптимальной загрузке судна и продемонстрируем способ ее решения с помощью генератора размещений. Новая формулировка следующая.

На палубе судна имеется  мест для размещения грузовых контейнеров. Выбрать  контейнеров для погрузки на судно можно из  имеющихся в наличии. Каждый контейнер  характеризуется весом  и доходом   от его перевозки. На каждое место  можно разместить контейнер, если его вес  где  и  – заданные величины.

Необходимо выбрать  контейнеров из  имеющихся таким образом, чтобы доход от перевозки был максимально возможным.

Математическая модель задачи выглядит следующим образом:

где  – неизвестные (номера выбранных контейнеров), которые требуется найти.

Решением задачи будет вектор . Каждый элемент этого вектора может принимать целое значение из отрезка  и при этом все значения  должны быть разными.

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

 – общее количество контейнеров;

 – количество свободных мест на палубе судна;

– вес контейнеров 

 – доход от перевозки контейнеров 

 – минимальный вес контейнеров (

 – максимальный вес контейнеров 





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

Слева на схеме (рис. 6) показана таблица, содержащая все размещения по три элемента множества  Эти размещения могут быть получены с помощью генератора размещений. Количество различных способов разместить контейнеры на палубе равно:



Элементы размещений, полученных с помощью генератора используются в качестве индексов в массивах, содержащих вес и доход от транспортировки контейнеров.

Строки таблицы, представленной на схеме справа, содержат планы всех размещений контейнеров по свободным местам на палубе судна. При этом в первой заголовочной строке таблицы указаны ограничения на вес контейнера для соответствующего места (два числа, обозначающие минимальный и максимальный вес), а в остальных ячейках для каждого места приведены вес соответствующего контейнера и доход от его перевозки в формате . Закрашенные ячейки таблицы обозначают нарушение ограничений по весу контейнеров для соответствующих мест. План размещения, имеющий хотя бы одну закрашенную ячейку, является недопустимым.

На схеме только два плана размещения контейнеров являются допустимыми. Для этих планов вычисляется доход от перевозки выбранных контейнеров. Доход указан в овальной рамке рядом со строкой плана. Оптимальный план размещения контейнеров на палубе судна выделен рамкой.

На рис. 7 и 8 представлен пример реализации на языке С++ функции **boat\_c**, решающей задачу о размещении контейнеров.

// --- Вoat.h

// -- решение задачи об оптимальном размещении контейнеров

// функция возвращает доход от перевозки выбранных контейнеров

**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

**);**

Рис. 7. Функция **boat\_c**, решающая задачу об оптимальном размещении контейнеров на судне

// --- Вoat.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\_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

**)**

**{**

**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**

Функция имеет семь параметров, определяющих условие задачи: **m** (количество мест для установки контейнеров), **minv** (массив размерностью **m**, содержащий минимальный вес контейнера для каждого места), **maxv** (массив размерностью **m**, содержащий максимальный вес контейнера для каждого места), **n** (количество контейнеров), **v** (массив размерностью **n**, содержащий вес контейнеров), **c** (массив размерностью **n**, содержащий величину дохода от перевозки каждого контейнера), а также один возвращаемый параметр **r** (массив размерностью **m**,содержащий номера выбранных контейнеров). Если решение задачи существует, то функция возвращает положительное значение, равное величине дохода, иначе возвращается нуль.

В функции **boat\_c** применяется генератор размещений (тип **combi::accomodation**). Конструктору генератора передается два параметра: **n** (количество контейнеров) и **m** (количество мест для установки контейнеров).

Кроме того, функция вызывает три вспомогательные функции.

Функция **boatfnc::compv** позволяет проверить допустимость размещения контейнеров. Если вес всех контейнеров в размещении удовлетворяет заданным ограничениям, то функция возвращает **true**, иначе возвращается **false**.

Функция **boatfnc::calcc** дает возможность вычислить доход от перевозки контейнеров при заданном размещении.

Функция **boatfnc::copycomb** предназначена для сохранения (копирования) наиболее доходного на данном шаге допустимого размещения.

Функция **boat\_c** в цикле генерирует все возможные размещения контейнеров (функции **getfirst** и **getnext** генератора размещений), проверяет их допустимость (функция **boatfnc::compv**), вычисляет для допустимых размещений доход от перевозки контейнеров (функция **boatfnc::calcc**), фиксирует оптимальное размещение (функция **boatfnc::copycomb**) и возвращает доход от оптимального размещения или нуль, если решения нет.

На рис. 9 и 10 приведен пример вызова функции **boat\_c** для решения задачи с исходными данными к схеме, представленной на рис. 6.

// -- 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\_с(**

**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

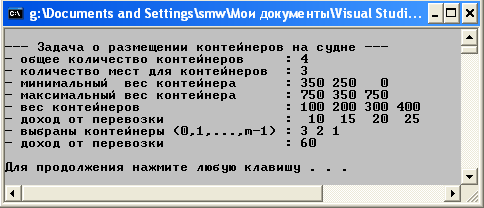


Рис. 10. Результат выполнения программы, представленной на рис. 5.9

Программа на рис. 9 сначала подготавливает массивы данных: **v** (вес каждого контейнера), **c** (доход от перевозки каждого контейнера), **minv** (нижняя граница веса контейнера для каждого свободного места на палубе судна), **maxv** (верхняя граница веса контейнера для каждого свободного места), а затем вызывает функцию **boat\_c**.

В результате выполнения функция возвращает доход от оптимального размещения контейнеров на палубе судна, а также параметр (массив **r**), содержащий перечень номеров выбранных контейнеров.

На рис. 11 представлена программа, позволяющая оценить продолжительность решения задачи о размещении контейнеров в зависимости от количества свободных мест на палубе судна, а на рис. 12 – результат ее выполнения.

#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\_с(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

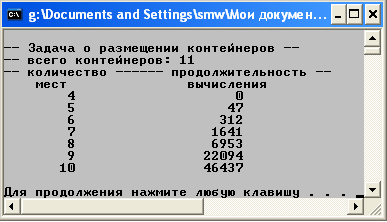


Рис. 12. Результат выполнения программы, представленной на рис. 11

Здесь применяются функции **auxil::start** и **auxil::iget**, позволяющие сгенерировать случайные последовательности целых чисел в заданных пределах.

На рис. 13 изображен график, отражающий зависимость продолжительности решения задачи от количества свободных мест на палубе судна при общем количестве контейнеров, равном 11.

**Зависимость продолжительности вычисления**

**от количества свободных мест при общем**

**количестве контейнеров, равном 11**

0

5

10

15

20

25

30

35

40

45

50

4

5

6

7

8

9

10

**Количество свободных мест на палубе судна**

**Продолжительность**

**вычисления, с**

Рис. 13. График, построенный по результатам вычислений, представленных на рис. 12

Вид графика, приведенного на рис. 13, вполне согласуется с оценкой  сложности алгоритма генерации размещений  из 11 элементов.

Сложность 