**Лабораторная работа №2**

Комбинаторные алгоритмы решения оптимизационных задач

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

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

#include "Module.h"

void printSubset(const vector<int>& subset) {

cout << "{ ";

for (int num : subset) {

cout << num << " ";

}

cout << "}\n";

}

void generateSubsets(const vector<int>& set, vector<int>& subset, int index) {

if (index == set.size()) {

printSubset(subset);

return;

}

subset.push\_back(set[index]);

generateSubsets(set, subset, index + 1);

subset.pop\_back();

generateSubsets(set, subset, index + 1);

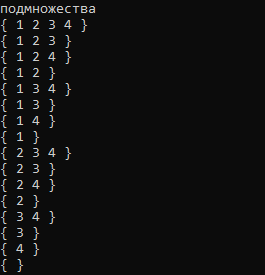
}

void generateAllSubsets(const vector<int>& set) {

vector<int> subset;

generateSubsets(set, subset, 0);

}



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

#include "Module.h"

void print(const vector<int>& arr) {

for (int num : arr) {

cout << num << " ";

}

cout << endl;

}

void generateCombinations(const vector<int>& elements, vector<int>& combination, int index, int k) {

if (combination.size() == k) {

print(combination);

return;

}

for (int i = index; i < elements.size(); ++i) {

combination.push\_back(elements[i]);

generateCombinations(elements, combination, i + 1, k);

combination.pop\_back();

}

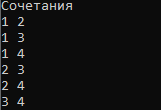
}

void generateAllCombinations(const vector<int>& elements, int k) {

vector<int> combination;

generateCombinations(elements, combination, 0, k);

}



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

#include "Module.h"

void generatePermutations(vector<int>& permutation, int index) {

if (index == permutation.size()) {

print(permutation);

return;

}

for (int i = index; i < permutation.size(); ++i) {

swap(permutation[index], permutation[i]);

generatePermutations(permutation, index + 1);

swap(permutation[index], permutation[i]);

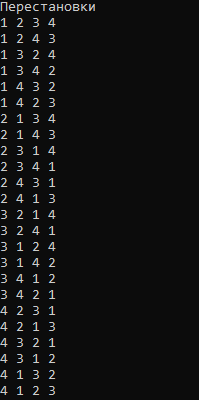
}

}

void generateAllPermutations(vector<int>& permutation) {

generatePermutations(permutation, 0);

}



**Задание 4.**

Разобрать и разработать генератор размещений.

void Revers(vector<int>& arr) {

for (int i = 0; i \* 2 < arr.size(); i++) {

int temp = arr[i];

arr[i] = arr[arr.size() - 1 - i];

arr[arr.size() - 1 - i] = temp;

}

}

void generateAllArrangements(const vector<int>& elements, int k) {

vector<int> arrangement1(elements);

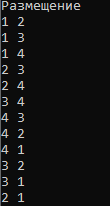
vector<int> arrangement2(elements);

Revers(arrangement2);

generateAllCombinations(arrangement1, k);

generateAllCombinations(arrangement2, k);

}



**Задание 5.**  Решить в соответствии с вариантом задачу и результат занести в отчет :

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);

#include "Module.h"

const int NUM\_SLOTS = 5;

const int NUM\_CONTAINERS = 8;

const int MIN\_CONTAINER\_WEIGHT = 100;

const int MAX\_CONTAINER\_WEIGHT = 200;

const int MIN\_PROFIT = 10;

const int MAX\_PROFIT = 100;

const int MIN\_SLOT\_MIN\_WEIGHT = 50;

const int MAX\_SLOT\_MIN\_WEIGHT = 120;

const int MIN\_SLOT\_MAX\_WEIGHT = 150;

const int MAX\_SLOT\_MAX\_WEIGHT = 850;

class Container

{

public:

int weight;

int profit;

Container(int w, int p) {

weight = w;

profit = p;

}

};

int RandomGenerator(int from, int to) {

return rand() % (to - from + 1) + from;

}

int CalculateTotalProfit(vector<Container>& containers) {

int totalProfit = 0;

for (const auto& container : containers) {

totalProfit += container.profit;

}

return totalProfit;

}

int CalculateTotalWeight(vector<Container>& containers) {

int totalWeight = 0;

for (const auto& container : containers) {

totalWeight += container.weight;

}

return totalWeight;

}

vector<Container> bestContainers;

int bestProfit = 0;

void generatePermutationsBoat(vector<Container>& permutation, int index) {

if (index == permutation.size()) {

if (isValidPlacement(permutation)) {

int currentProfit = CalculateTotalProfit(permutation);

if (currentProfit > bestProfit) {

bestProfit = currentProfit;

bestContainers = permutation;

}

}

return;

}

for (int i = index; i < permutation.size(); ++i) {

swap(permutation[index], permutation[i]);

generatePermutationsBoat(permutation, index + 1);

swap(permutation[index], permutation[i]);

}

}

bool isValidPlacement(const vector<Container>& containers) {

vector<int> minWeights(NUM\_SLOTS, 0);

vector<int> maxWeights(NUM\_SLOTS, 0);

for (int i = 0; i < NUM\_SLOTS; ++i) {

minWeights[i] = RandomGenerator(MIN\_SLOT\_MIN\_WEIGHT, MAX\_SLOT\_MIN\_WEIGHT);

maxWeights[i] = RandomGenerator(MIN\_SLOT\_MAX\_WEIGHT, MAX\_SLOT\_MAX\_WEIGHT);

}

for (int i = 0; i < containers.size(); ++i) {

if (containers[i].weight < minWeights[i % NUM\_SLOTS] || containers[i].weight > maxWeights[i % NUM\_SLOTS]) {

return false;

}

}

return true;

}

vector<Container> optimizeLoading(int num\_Containers) {

vector<Container> bestContainers;

int bestProfit = 0;

vector<Container> containers;

for (int i = 0; i < num\_Containers; ++i) {

int weight = RandomGenerator(MIN\_CONTAINER\_WEIGHT, MAX\_CONTAINER\_WEIGHT);

int profit = RandomGenerator(MIN\_PROFIT, MAX\_PROFIT);

containers.push\_back(Container(weight, profit));

}

generatePermutationsBoat(containers, 0);

return bestContainers;

}

void GenereteOptimizeLoad(int num\_Containers) {

vector<Container> optimalContainers = optimizeLoading(num\_Containers);

cout << "Оптимальная загрузка:\n";

for (const auto& container : optimalContainers) {

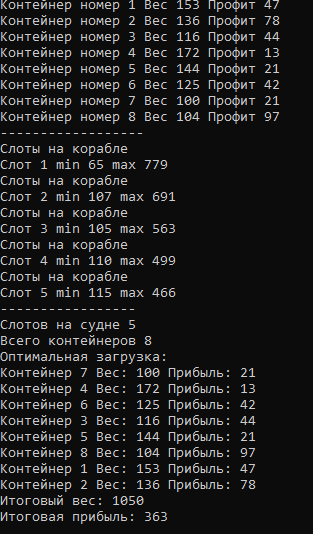
cout << "Вес: " << container.weight << " Прибыль: " << container.profit << std::endl;

}

cout << "Итоговый вес: " << CalculateTotalWeight(optimalContainers) << std::endl;

cout << "Итоговая прибыль: " << CalculateTotalProfit(optimalContainers) << std::endl;

}



**Задание 6.** Исследовать зависимость времени вычисления необходимое для решения задачи от размерности задачи и результат в виде графика с небольшим пояснением занести в отчет:

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

clock\_t t1 = 0, t2 = 0;

for (int i = 4; i <= 8; i++) {

cout << i << "Конетйнеров" << endl;

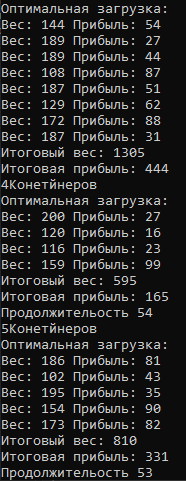
t1 = clock();

GenereteOptimizeLoad(i);

t2 = clock();

cout << "Продолжительость " << t2 - t1 << endl;

}



Исходные данные:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Количество Контейнеров | 4 | 5 | 6 | 7 | 8 |
|  |  |  |  |  |  |
| Проц. время | 60 | 61 | 61 | 71 | 106 |

Как видно из графика время выполнения программы решения задачи о судне с условием центровки растет полиномиально согласно количеству контейнеров

Вывод: В процессе выполнения данной работы вы приобрели навыки разработки генераторов подмножеств, перестановок, сочетаний и размещений на языке C++. Вы также научились применять эти генераторы для решения различных задач, включая задачи о рюкзаке, коммивояжере, оптимальной загрузке судна и оптимальной загрузке судна с центровкой.