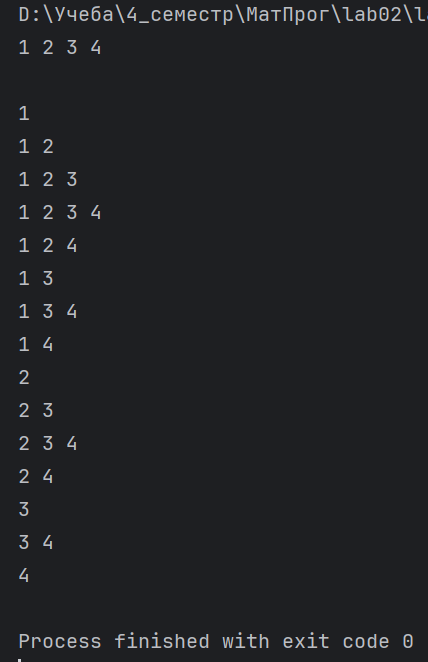
**Лабораторная работа 2. Комбинаторные алгоритмы решения оптимизационных задач**

**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 ДЛЯ ВЫПОЛНЕНИЯ:**

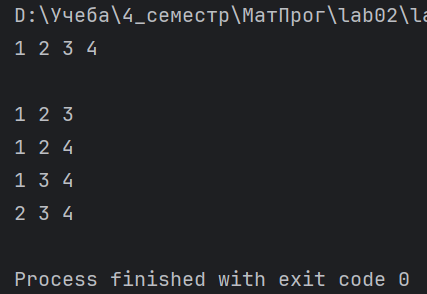
**Задание 1.** Разобрать и разработать генератор подмножеств заданного множества.

#include <iostream>  
#include <vector>  
  
using namespace std;  
  
void printSet(const vector<int>& set) {  
 for (int num : set) {  
 cout << num << " ";  
 }  
 cout << endl;  
}  
  
// Функция для генерации подмножеств  
void generateSubsets(const vector<int>& set, vector<int>& subset, int index) {  
 // Выводим текущее подмножество  
 for (int num : subset) {  
 cout << num << " ";  
 }  
 cout << endl;  
  
 int i = index;  
 // Генерируем подмножества, начиная с текущего индекса  
 for (i; i < set.size(); i++) {  
 // Добавляем элемент в подмножество  
 subset.push\_back(set[i]);  
  
 // Рекурсивно генерируем подмножества, начиная с следующего элемента  
 generateSubsets(set, subset, i + 1);  
  
 // Удаляем последний добавленный элемент для генерации следующего подмножества  
 subset.pop\_back();  
 }  
}  
  
int main() {  
 vector<int> set = **{** 1, 2, 3, 4 **}**;  
 vector<int> subset;  
 printSet(set);  
 generateSubsets(set, subset, 0);  
  
 return 0;  
}



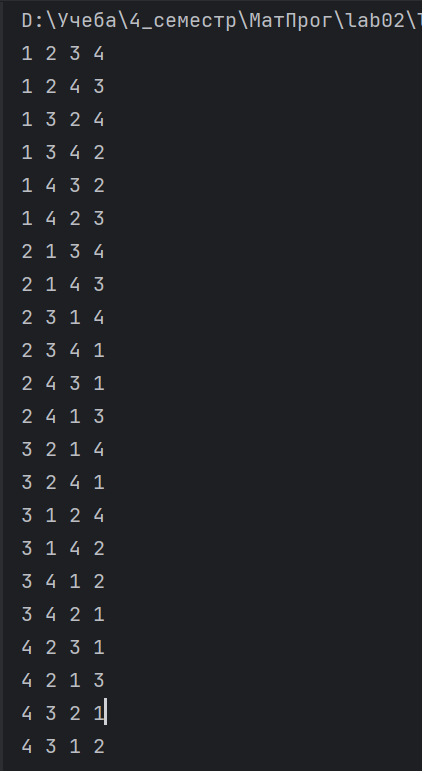
**Задание 2.** Разобрать и разработать генератор сочетаний.

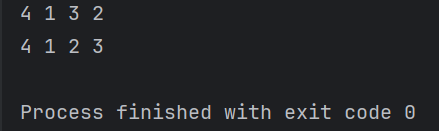
#include <iostream>  
#include <vector>  
  
using namespace std;  
  
void printSet(const vector<int>& set) {  
 for (int num : set) {  
 cout << num << " ";  
 }  
 cout << endl << endl;  
}  
  
// Функция для генерации сочетаний  
void generateCombinations(const vector<int>& set, vector<int>& combination, int index, int k) {  
 // Проверяем, достигли ли мы размера k для сочетания  
 if (combination.size() == k) {  
 for (int num : combination) {  
 cout << num << " ";  
 }  
 cout << endl;  
 return;  
 }  
  
 // Генерируем сочетания, начиная с текущего индекса  
 for (int i = index; i < set.size(); i++) {  
 // Добавляем элемент в сочетание  
 combination.push\_back(set[i]);  
  
 // Рекурсивно генерируем сочетания, начиная со следующего элемента  
 generateCombinations(set, combination, i + 1, k);  
  
 // Удаляем последний добавленный элемент для генерации следующего сочетания  
 combination.pop\_back();  
 }  
}  
  
int main() {  
 vector<int> set = **{** 1, 2, 3, 4 **}**;  
 int k = 3; // Размер сочетания  
 vector<int> combination;  
 printSet(set);  
 generateCombinations(set, combination, 0, k);  
  
 return 0;  
}



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

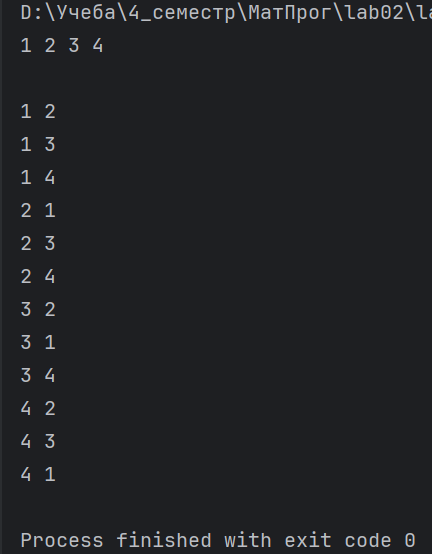
#include <iostream>  
#include <vector>  
  
using namespace std;  
  
// Функция для генерации перестановок  
void generatePermutations(vector<int>& elements, int index) {  
 // Проверяем, достигли ли мы конца вектора элементов  
 if (index == elements.size() - 1) {  
 // Выводим текущую перестановку  
 for (int num : elements) {  
 cout << num << " ";  
 }  
 cout << endl;  
 return;  
 }  
  
 // Генерируем перестановки, начиная с текущего индекса  
 for (int i = index; i < elements.size(); i++) {  
 // Меняем текущий элемент с элементом на индексе i  
 swap(elements[index], elements[i]);  
  
 // Рекурсивно генерируем перестановки, начиная со следующего индекса  
 generatePermutations(elements, index + 1);  
  
 // Возвращаем элементы в исходное состояние  
 swap(elements[index], elements[i]);  
 }  
}  
  
int main() {  
 vector<int> elements = **{** 1, 2, 3, 4 **}**;  
 generatePermutations(elements, 0);  
  
 return 0;  
}





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

#include <iostream>  
#include <vector>  
  
using namespace std;  
  
void printSet(const vector<int>& set) {  
 for (int num : set) {  
 cout << num << " ";  
 }  
 cout << endl << endl;  
}  
  
// Функция для генерации размещений  
void generatePermutations(vector<int>& elements, int index, int k) {  
 // Проверяем, достигли ли мы конца вектора элементов  
 if (index == k) {  
 // Выводим текущее размещение  
 for (int i = 0; i < k; i++) {  
 cout << elements[i] << " ";  
 }  
 cout << endl;  
 return;  
 }  
  
 // Генерируем размещения, начиная с текущего индекса  
 for (int i = index; i < elements.size(); i++) {  
 // Меняем текущий элемент с элементом на индексе i  
 swap(elements[index], elements[i]);  
  
 // Рекурсивно генерируем размещения, начиная со следующего индекса  
 generatePermutations(elements, index + 1, k);  
  
 // Возвращаем элементы в исходное состояние  
 swap(elements[index], elements[i]);  
 }  
}  
  
int main() {  
 vector<int> elements = **{** 1, 2, 3, 4 **}**;  
 int k = 2;  
 printSet(elements);  
 generatePermutations(elements, 0, k);  
  
 return 0;  
}



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

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



#include <iostream>  
#include <vector>  
#include <algorithm>  
#include <cstdlib>  
#include <ctime>  
#include <limits>  
#include <iomanip>  
  
using namespace std;  
  
const int INF = numeric\_limits<int>::max();  
  
// Генерация случайных расстояний между городами  
void generateDistances(vector<vector<int>>& distances, int n) {  
 distances.resize(n, vector<int>(n, INF));  
 for (int i = 0; i < n; i++) {  
 distances[i][i] = INF;  
 for (int j = i + 1; j < n; j++) {  
 distances[i][j] = distances[j][i] = (rand() % 291) + 10; // Случайное число от 10 до 300  
 }  
 }  
  
 // Установка случайного расстояния между 3 городами как бесконечное  
 int city1 = rand() % n;  
 int city2 = rand() % n;  
 int city3 = rand() % n;  
 while (city2 == city1) {  
 city2 = rand() % n;  
 }  
 while (city3 == city1 || city3 == city2) {  
 city3 = rand() % n;  
 }  
 distances[city1][city2] = distances[city2][city1] = INF;  
 distances[city1][city3] = distances[city3][city1] = INF;  
 distances[city2][city3] = distances[city3][city2] = INF;  
}  
  
// Решение задачи коммивояжера  
vector<int> solveTravelingSalesmanProblem(const vector<vector<int>>& distances, int& totalDistance) {  
 int n = distances.size();  
 vector<int> cities(n);  
 for (int i = 0; i < n; i++) {  
 cities[i] = i + 1;  
 }  
  
 vector<int> bestPath;  
 int bestDistance = INF;  
  
 // Генерация всех перестановок городов  
 do {  
 int distance = 0;  
 bool validPath = true;  
  
 // Вычисление длины пути  
 for (int i = 0; i < n - 1; i++) {  
 int city1 = cities[i] - 1;  
 int city2 = cities[i + 1] - 1;  
 if (distances[city1][city2] == INF) {  
 validPath = false;  
 break;  
 }  
 distance += distances[city1][city2];  
 }  
  
 // Замыкаем путь к начальному городу  
 if (validPath && distances[cities[n - 1] - 1][cities[0] - 1] != INF) {  
 distance += distances[cities[n - 1] - 1][cities[0] - 1];  
 if (distance < bestDistance) {  
 bestDistance = distance;  
 bestPath = cities;  
 }  
 }  
 } while (next\_permutation(cities.begin() + 1, cities.end()));  
  
 totalDistance = bestDistance;  
 return bestPath;  
}  
  
int main() {  
 srand(time(nullptr));  
  
 int n = 10; // Количество городов  
 vector<vector<int>> distances;  
 generateDistances(distances, n);  
  
 // Вывод номеров городов сверху  
 cout << " ";  
 for (int i = 1; i <= n; i++) {  
 cout << setw(7) << left << i;  
 }  
 cout << endl;  
  
 // Вывод матрицы расстояний и номеров городов слева  
 for (int i = 1; i <= n; i++) {  
 cout << setw(4) << left << i;  
 for (int j = 1; j <= n; j++) {  
 if (distances[i - 1][j - 1] == INF) {  
 cout << setw(7) << left << "INF";  
 } else {  
 cout << setw(7) << left << distances[i - 1][j - 1];  
 }  
 }  
 cout << endl;  
 }  
 cout << endl;  
  
 clock\_t startTime = clock();  
  
 int optimalDistance;  
 vector<int> optimalPath = solveTravelingSalesmanProblem(distances, optimalDistance);  
  
 clock\_t endTime = clock();  
 double executionTime = double(endTime - startTime) / CLOCKS\_PER\_SEC;  
  
 cout << "Optimal path: ";  
 for (int city : optimalPath) {  
 cout << city << " ";  
 }  
 cout << endl;  
 cout << "Best path: " << optimalDistance << endl;  
 cout << "Execution time: " << executionTime << " seconds" << endl;  
 cout << endl;  
  
 return 0;  
}

