



Министерство науки и высшего образования Российской Федерации
Федеральное государственное
бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ

«Фундаментальные Науки»

КАФЕДРА

ФН-12 «Математическое моделирование»

ОТЧЕТ

ПО РК 1:

Метод перебора

Студент:

дата, подпись

Мацеевский И. М.

Ф.И.О.

Преподаватель:

дата, подпись

Волкова Л. Л.

Ф.И.О.

Москва, 2023

Содержание

Введение	2
1 Аналитическая часть	2
1.1 Задача коммивояжера	2
1.2 Метод полного перебора	2
2 Конструкторская часть	2
3 Технологическая часть	3
Заключение	8

Введение

Цель: Решить подзадачу задачи коммивояжера, получить навык реализации алгоритмов. Реализовать метод полного перебора в графе: дан граф на N вершин, требуется перебрать все возможные решения задачи коммивояжера — кортежи N городов.

1 Аналитическая часть

1.1 Задача коммивояжера

Задача коммивояжера — это классическая задача комбинаторной оптимизации, которая формулируется следующим образом: имеется граф, в котором вершины представляют города, а рёбра — пути между городами. Каждому ребру сопоставлен вес, который обычно представляет собой расстояние или стоимость перемещения между соответствующими городами. Задача состоит в том, чтобы найти самый выгодный (кратчайший или наименее затратный) маршрут, проходящий через каждый город ровно один раз и возвращающийся в исходный город.

1.2 Метод полного перебора

Метод полного перебора — это способ решения задачи коммивояжера, при котором алгоритм перебирает все возможные варианты посещения вершин графа и выбирает оптимальный. Для задачи коммивояжера метод полного перебора проверяет все возможные порядки посещения городов и находит тот, который является минимальным по суммарному пройденному расстоянию. Преимуществом этого метода является простота его написания, но он неэффективен из-за проверки всех возможных маршрутов.

2 Конструкторская часть

1. Создать два вектора: *free* и *ans* — в первом хранятся свободные вершины, то есть те, которые пока не включены в данный маршрут, а во втором те, которые уже входят в него, изначально вектор *free* пуст, а в векторе *ans* хранятся вершины в любой последовательности. Инициализируется переменная *last*, изначально $last = ans[N - 1]$. *ans* добавляется в файл всех перестановок
2. Далее следует найти вершину с минимальным номером, который при этом больше, чем *last*.
3. Если такая вершина нашлась, она добавляется в вектор *ans*, после чего в него в порядке возрастания записываются оставшиеся вершины, вектор *free* очищается, вектор *ans* записывается в файл со всеми маршрутами.

4. Если такой вершины нет и при этом вектор *ans* пуст, все маршруты найдены, программа завершается. Если же вектор *ans* не пуст, то последний элемент переносится из вектора *ans* в вектор *free*, при этом переменная *last* принимает его значение.

3 Технологическая часть

Для реализации выбран язык C++. Функции, реализованные в проекте:

1. *show_one_dim* — выводит на экран одномерный вектор;
2. *write_to_f* — записывает вектор в файл.

Для реализации выбран язык C++. На листинге 1 представлена реализация программы (Реализация 1).

Листинг 1 – Исходный код

```
#include <iostream>
using namespace std;
#include <vector>
#include <fstream>

void show_one_dim(const vector<int>& vec) {
    for (int i = 0; i < vec.size(); i++) {
        cout << vec[i] + 1 << " ";
    }
}

void write_to_f (vector<int>&vec, fstream &file) {
    for (int i = 0; i < vec.size(); i++) {
        file << vec[i] + 1 << " ";
    }
    file << endl;
}

int main() {
    fstream file("/Users/ilya/DownloadsТипы/ и структуры данных 2 курс, 1
семестр/complet_search/permutations.txt");
    int n;
    cin >> n;
    vector<int> free;
    vector<int> ans;
    for (int i = 0; i < n; i++) {
        ans.push_back(i);
    }
    int last;
    last = ans[n - 1];
    write_to_f(ans, file);
    ans.pop_back();
    free.push_back(last);
    while (1) {
        int new_el = n + 1;
        for (int i = 0; i < free.size(); i++) {
```

```

        if (free[i] > last) {
            new_el = min(free[i], new_el);
        }
    }
    if (new_el == n + 1) {
        if (ans.empty()){
            //show_one_dim(free);
            write_to_f(ans, file);
            break;
        }
        last = ans[ans.size() - 1];
        ans.pop_back();
        free.push_back(last);
    }
    else {
        ans.push_back(new_el);
        erase(free, new_el);
        sort(free.begin(), free.end());
        for (int i = 0; i < free.size(); i++) {
            ans.push_back(free[i]);
        }
        free.clear();
        write_to_f(ans, file);
    }
}
file.close();
return 0;
}

```

Примеры работы.

1. $N = 1$, то есть дерево из 1 вершины.

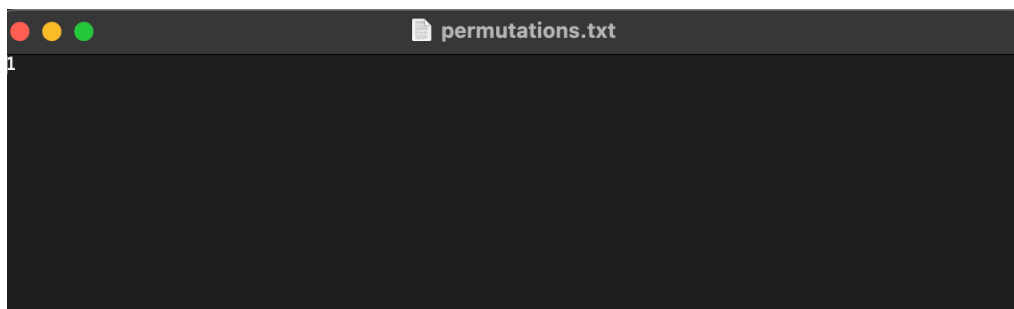


Рис. 1 – Пример работы 1

2. $N = 2$, то есть дерево из 2 вершин.

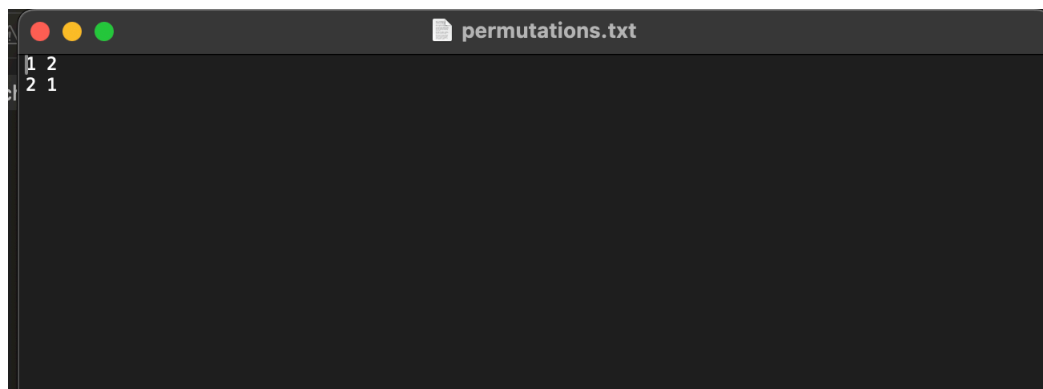


Рис. 2 – Пример работы 2

3. $N = 3$, то есть дерево из 3 вершины.

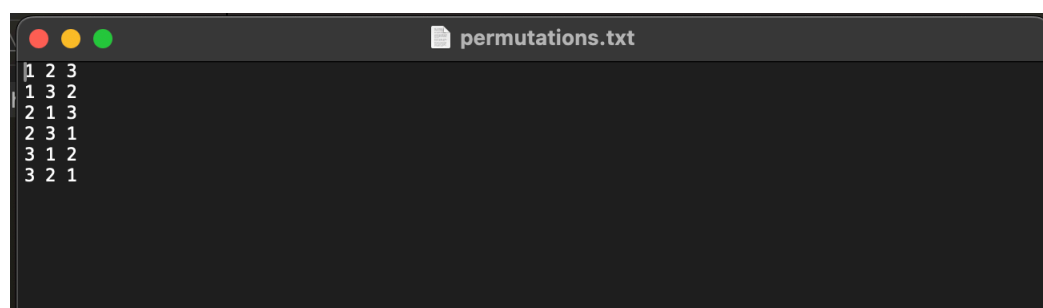
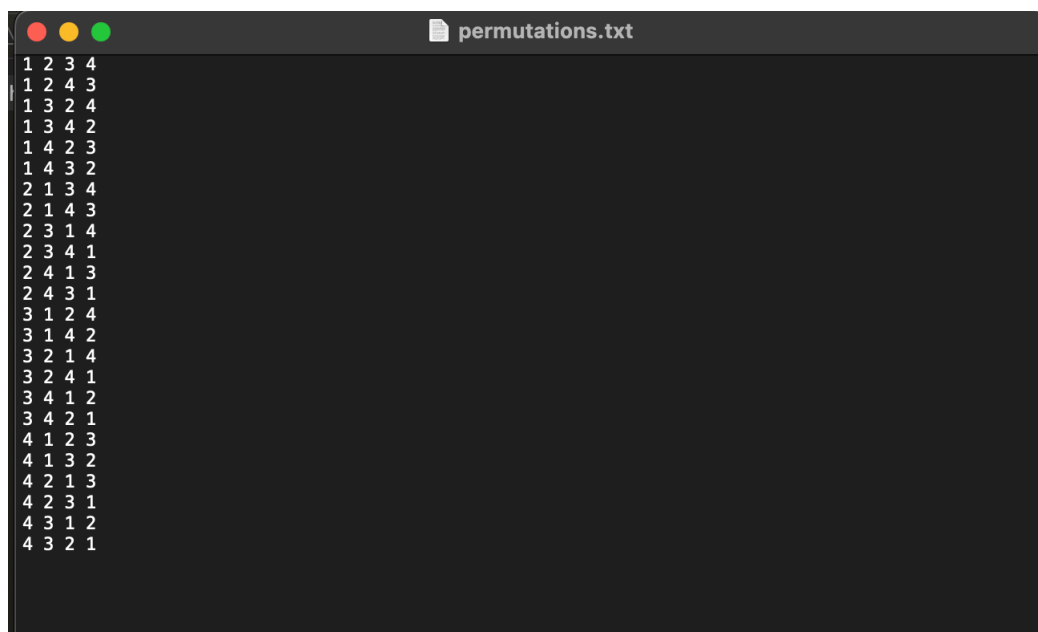


Рис. 3 – Пример работы 3

4. $N = 4$, то есть дерево из 4 вершины.



```
permutations.txt
1 2 3 4
1 2 4 3
1 3 2 4
1 3 4 2
1 4 2 3
1 4 3 2
2 1 3 4
2 1 4 3
2 3 1 4
2 3 4 1
2 4 1 3
2 4 3 1
3 1 2 4
3 1 4 2
3 2 1 4
3 2 4 1
3 4 1 2
3 4 2 1
4 1 2 3
4 1 3 2
4 2 1 3
4 2 3 1
4 3 1 2
4 3 2 1
```

Рис. 4 – Пример работы 4

Заключение

Цель достигнута: изучен и реализован метод полного перебора, проведен анализ эффективности. Метод полного перебора предполагает перебор всех возможных комбинаций вершин, что является вычислительно затратной операцией. Однако он гарантирует нахождение оптимального решения в контексте задачи обхода вершин графа.