МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА 25

КУРСОВАЯ РАБОТА (ПРОЕКТ)   
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доцент, канд. тех. наук |  |  |  | Е. М. Линский |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ |
| ЗАДАЧА КОММИВОЯЖЕРА |
| по дисциплине: ОСНОВЫ ПРОГРАММИРОВАНИЯ |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | 2352 |  |  |  | Мельников Н.Д |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

**СОДЕРЖАНИЕ**

[ПОСТАНОВКА ЗАДАЧИ 3](#_Toc185872450)

[ОПИСАНИЕ АЛГОРИТМА 4](#_Toc185872451)

[Пошаговое выполнение жадного алгоритма с примером 6](#_Toc185872452)

[Пошаговое выполнение алгоритма ветвей и границ с примером 9](#_Toc185872453)

[Псевдокод: 10](#_Toc185872454)

[ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЯ 14](#_Toc185872455)

[ТЕСТОВЫЕ ПРИМЕРЫ 15](#_Toc185872456)

[СПИСОК ЛИТЕРАТУРЫ 19](#_Toc185872457)

# ПОСТАНОВКА ЗАДАЧИ

Задачей данной курсовой работы является разработка программы, которая решает задачу коммивояжера с использованием двух алгоритмов: метода ветвей и границ и жадного алгоритма. Программа должна принимать на вход матрицу смежности, представляющую взвешенный граф, и генерировать файл в формате Graphviz, визуализирующий оптимальный маршрут и его стоимость. Основная цель программы — сравнить эффективность указанных алгоритмов по качеству найденного решения и времени выполнения на тестовых данных.

**Пример**:

Пусть дана матрица смежности:

​**0 10 15 20​**

**10 0 35 25​**

**15 35 0 30​**

**20 25 30 0​**

*Возможные маршруты: (метод ветвей и границ)*

**A → B → C → D → A: стоимость 10 + 35 + 30 + 20 = 95**

**A → C → B → D → A: стоимость 15 + 35 + 25 + 20 = 95**

**A → D → C → B → A: стоимость 20 + 30 + 15 + 10 = 75 (оптимальное решение).**

*Жадный алгоритм:*

**A → B → D → C → A: стоимость 10 + 25 + 30 + 15 = 80**

# ОПИСАНИЕ АЛГОРИТМА

**Жадный алгоритм**

Жадный алгоритм строит маршрут, выбирая на каждом шаге ребро с минимальной стоимостью, соединяющее текущую вершину с непосещенной. Такой подход обеспечивает быстрое нахождение приближенного решения, хотя оно не всегда является оптимальным.

**Описание работы:**

**1)Начинаем с заданной стартовой вершины.**

1. Алгоритм получает стартовую вершину в качестве входных данных. Например, если начальная вершина — 0, маршрут будет строиться от неё.

2. Создаётся список для хранения порядка посещения вершин *route*.

Инициализация:

*vector<pair<int, int>> route;* ({вершина, стоимость перехода}).

*route.push\_back({start, 0});* (Добавляем стартовую вершину с весом 0).

*MinDist = 0;* (Общее расстояние, которое мы пройдем за весь путь).

**2)Помечаем её как посещенную.**

Используется массив посещённых вершин *vector<int> visited[],* где *visited[i] = 1* означает, что вершина посещена, соответственно *visited[i] = 0* – не посещена.

Начальную вершину уже можно считать посещенной, т.к мы начинаем обход с нее, поэтому *visited[start] = 1;* (*visited[0] = 1* – в нашем случае).

**3)Выбираем ближайшую непосещенную вершину и добавляем её в маршрут.**

Среди всех возможных переходов из текущей вершины, выбирается вершина с минимальной стоимостью перехода, которая ещё не была посещена.

Для этого просматриваются значения в строке матрицы смежности для текущей вершины.

Например для текущей вершины 0 ищем минимальное значение среди *matrix[0][i],* где *visited[i] = 0.*

Допустим, ближайшая вершина — 2 с весом 5.

Переход выполняется в вершину 2, она добавляется в маршрут, прибавляем пройденное расстояние к MinDist.

Текущая ситуация после перехода.

Текущая вершина: 2;

*Route : [{0,5}];*

*Visited[2] = 1*;

*MinDist* =5.

**4)Повторяем до тех пор, пока все вершины не будут посещены.**

Алгоритм продолжает выбирать ближайшую непосещённую вершину на каждом шаге.

После каждого перехода:

* Обновляется текущая вершина.
* Добавляется {текущая вершина, стоимость перехода} в route.
* Текущая вершина помечается как посещённая.
* Обновляет MinDist.

Этот процесс повторяется до тех пор, пока все вершины графа не будут включены в маршрут.

**5)Возвращаемся в начальную вершину.**

После посещения всех вершин, алгоритм добавляет ребро от последней посещённой вершины к стартовой вершине, также не забываем добавлять пройденное расстояние к MinDist.

## **Пошаговое выполнение жадного алгоритма с примером**

Входные данные:

Стартовая вершина: 0.

Общее количество вершин: 4.

INF 10 15 20

10 INF 35 25

15 35 INF 30

20 25 30 INF

Шаг 1: Инициализация

1.visited[0] = 1;

2.route = { {0, 0} }- начало маршрута, переход из 0 с весом 0.

3.MinDist = 0;

Шаг 2: Поиск ближайшей вершины из 0

1.Просматриваем первую строку в матрице: matrix[0] = [INF, 10, 15, 20]

2. Минимальное расстояние = 10 --> visited[1] = 1; route.push\_back({ 0, 10 });

3. Обновляем текущую вершину current = 1;

4. MinDist += 10; Обновляем итоговую стоимость пути.

Шаг 3: Поиск ближайшей вершины из 1

1.Просматриваем вторую строку в матрице: matrix[1] = [10, INF, 35, 25]

2. Минимальное расстояние = 25, т.к вершину 0 уже посещена.

visited[3] = 1, route.push\_back({ 1, 25 });

3. Обновляем текущую вершину current = 3;

4. MinDist += 25; Обновляем итоговую стоимость пути.

Шаг 4: Поиск ближайшей вершины из 3

1.Просматриваем четвертую строку в матрице: matrix[3] = [20, 25, 30, INF]

2. Минимальное расстояние = 30, т.к остальные вершины уже посещены --> visited[2] = 1; route.push\_back({ 3, 30 });

3. Обновляем текущую вершину current = 2;

4. MinDist += 30; Обновляем итоговую стоимость пути.

Шаг 5: Возвращение в начальную вершину

1.Все вершины посещены, возвращаемся в начальную вершину 0.

(matrix[2][0] = 15);

2. route.push\_back({ 2, 15 });

3. route = { {0, 0}, {0, 10}, {1, 25}, {3, 30}, {2, 15} } - Итоговый маршрут.

4.MinDist += 15;

5. MinDist = 80 (Итоговая стоимость пути)

**Метод ветвей и границ**

Метод ветвей и границ используется для поиска оптимального решения задачи о коммивояжёре. Он гарантирует нахождение оптимального маршрута, но имеет большую вычислительную сложность. Основная идея алгоритма — исследовать возможные пути, отсекать те, которые заведомо не могут привести к лучшему решению, и минимизировать текущие затраты**.**

**Описание работы**

**1.Инициализация данных:**

Входные данные:

1. матрица стоимости переходов между вершинами ‘SolutionArray‘,
2. массив посещённых вершин visited;
3. общее количество вершин v;
4. стартовая вершина start;

Начальная настройка:

1. curcountnodes: счётчик посещённых вершин;
2. curcost: текущая стоимость пути;
3. curnode: текущая вершина;
4. mincost: минимальная стоимость пути (обновляется в процессе);
5. curPath: текущий путь;
6. bestPath: лучший найденный путь.

**2.Поиск решения с использованием рекурсии:**

На каждом шаге проверяется, достигнут ли конечный случай: все вершины посещены и можно вернуться в стартовую вершину. Если так, обновляется mincost и сохраняется bestPath.

Для каждого непосещённого соседа текущей вершины выполняются следующие действия:

1. Отмечается как посещённая.
2. Добавляется в текущий путь curPath.
3. Вызывается рекурсивная функция с обновлёнными параметрами.
4. После возврата снимаются изменения (отмена выбора вершины).

**3.Рекурсивное отсечение:**

Отсечение происходит, если текущая стоимость curcost плюс стоимость возврата больше, чем mincost.

**4.Возврат оптимального решения:**

Алгоритм возвращает маршрут bestPath и минимальную стоимость mincost.

## **Пошаговое выполнение алгоритма ветвей и границ с примером**

Входные данные:

Стартовая вершина: 0.

Общее количество вершин: 4.

INF 10 15 20

10 INF 35 25

15 35 INF 30

20 25 30 INF

Шаг 1: Инициализация

Устанавливаем стартовую вершину: start = 0.

Visited[start] = 1 — начальную вершину отмечаем как посещённую.

curPath = { {0, 0} } — начинаем с вершины 0.

mincost = INF — изначально минимальная стоимость не определена.

bestPath = {} — оптимальный маршрут пока пуст.

Шаг 2: Исследование соседей вершины 0

1. matrix[0] = [INF, 10, 15, 20].
2. Выбираем вершину 1 с минимальной стоимостью перехода 10.

visited[1] = 1.

curPath = { {0, 0}, {1, 10} }.

curcost = 10.

1. Переходим к следующей вершине (рекурсивно).

Шаг 3: Исследование соседей вершины 1

1. matrix[1] = [10, INF, 35, 25].
2. Выбираем вершину 3 с минимальной стоимостью перехода 25.

Visited[3] = 1;

curPath = { {0, 0}, {1, 10}, {3, 25} }.

curcost = 35.

1. Переходим к следующей вершине (рекурсивно).

Шаг 4: Исследование соседей вершины 3

1. matrix[3] = [20, 25, 30, INF].
2. Выбираем вершину 2 с минимальной стоимостью перехода 30.

Visited[2] = 1;

curPath = { {0, 0}, {1, 10}, {3, 25}, {2, 30} }.

curcost = 65.

1. Переходим к следующей вершине (рекурсивно).

Шаг 5: Завершение маршрута

1.Все вершины посещены. Возвращаемся в стартовую вершину 0.

matrix[2][0] = 15.

2. bestPath = { {0, 0}, {1, 10}, {3, 25}, {2, 30}, {0, 15} }.

mincost = 80.

## **Псевдокод:**

void Input(SolutionArray, filename){

ОТКРЫТЬ файл filename ДЛЯ чтения (fstream in)

СЧИТАТЬ начальную вершину (start)

СЧИТАТЬ количество вершин (v)

ДЛЯ i от 0 до v - 1:

ДЛЯ j от 0 до v - 1:

СЧИТАТЬ значение SolutionArray[i][j]

ЕСЛИ i == j:

SolutionArray[i][j] = INF

ЗАКРЫТЬ файл (in.close())

}

pair<vector<pair<int, int>>, int> SolutionGreedy(SolutionArray, v, start){

ИНИЦИАЛИЗИРОВАТЬ mindist = 0

ИНИЦИАЛИЗИРОВАТЬ route как пустой список пар (вершина, стоимость)

ИНИЦИАЛИЗИРОВАТЬ visited как массив размера v, заполненный нулями

УСТАНОВИТЬ current = start

ПОКА ИСТИНА:

visited[current] = 1

cur = { -1, INF }

ДЛЯ i от 0 до v - 1:

ЕСЛИ visited[i] == 0 И SolutionArray[current][i] < cur.second:

cur = { i, SolutionArray[current][i] }

ЕСЛИ cur.first == -1:

ВЫЙТИ ИЗ ЦИКЛА

ДОБАВИТЬ { current, cur.second } в route

mindist += cur.second

current = cur.first

mindist += SolutionArray[current][start]

ДОБАВИТЬ { current, SolutionArray[current][start] } в route

ДОБАВИТЬ { start, INF } в route

ВЕРНУТЬ { route, mindist }

}

void GraphvizGreedy(SolutionArray, v, start, filename) {

ОТКРЫТЬ файл filename ДЛЯ записи (ofstream g)

ВЫЗВАТЬ SolutionGreedy(SolutionArray, v, start) и сохранить результат в res

g << "digraph G {" << endl

current = start

ДЛЯ i от 1 до res.first.size() - 1:

g << current << " -> " << res.first[i].first << " [label=\"" << res.first[i - 1].second << "\"];" << endl

current = res.first[i].first

g << "}" << endl

ВЫВЕСТИ "after greedy alg mincost = " << res.second

ЗАКРЫТЬ файл (g.close())

}

void SolutionBAB(SolutionArray, visited, v, start, curcountnodes, curcost, curnode, mincost, curPath, bestPath){

ЕСЛИ curcountnodes == v И SolutionArray[curnode][start] > 0:

ЕСЛИ curcost + SolutionArray[curnode][start] < mincost:

mincost = curcost + SolutionArray[curnode][start]

bestPath = curPath

ДОБАВИТЬ { start, SolutionArray[curnode][start] } в bestPath

ВЕРНУТЬСЯ

ДЛЯ next от 0 до v - 1:

ЕСЛИ visited[next] == 0 И SolutionArray[curnode][next] > 0:

visited[next] = 1

ДОБАВИТЬ { next, SolutionArray[curnode][next] } в curPath

ВЫЗВАТЬ SolutionBAB(SolutionArray, visited, v, start, curcountnodes + 1,

curcost + SolutionArray[curnode][next], next, mincost, curPath, bestPath)

visited[next] = 0

УДАЛИТЬ последний элемент из curPath

}

void GraphvizBAB(SolutionArray, v, start, filename){

ОТКРЫТЬ файл filename ДЛЯ записи (ofstream g)

curPath = пустой список пар (вершина, стоимость)

bestPath = пустой список пар (вершина, стоимость)

visited = массив размера v, заполненный нулями

mincost = INF

visited[start] = 1

ДОБАВИТЬ { start, 0 } в curPath

ВЫЗВАТЬ SolutionBAB(SolutionArray, visited, v, start, 1, 0, start, mincost, curPath, bestPath)

g << "digraph G {" << endl

current = start

ДЛЯ i от 1 до bestPath.size():

g << current << " -> " << bestPath[i].first << " [label=\"" << bestPath[i].second << "\"];" << endl

current = bestPath[i].first

g << "}" << endl

ВЫВЕСТИ "after BAB mincost = " << mincost

ЗАКРЫТЬ файл (g.close())

}

**Временная сложность алгоритмов**

1)Жадный алгоритм – O(v\*v)

2)Метод ветвей и границ – O(v\*v!)

# ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЯ

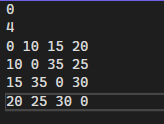
На вход подается текстовый файл input.txt, на выходе два файла с расширением dot.

Для того чтобы отрисовать два полученных решения, нужно в терминале прописать данную команду – dot -Tpng название.dot -o название.png (для каждого полученного файла dot)

# ТЕСТОВЫЕ ПРИМЕРЫ

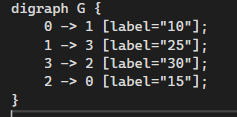
Тест 1

Входные данные:

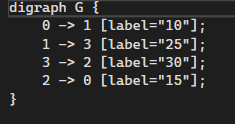


Выходные данные:

После жадного алгоритма.



После метода ветвей и границ.



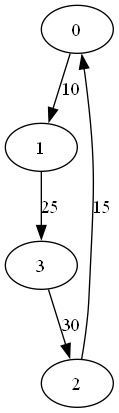
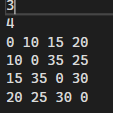


Рисунок 1. Жадный алгоритм; метод ветвей и границ

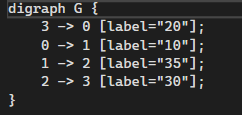
Тест 2

Входные данные:

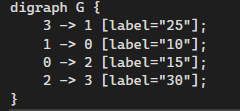


Выходные данные:

После жадного алгоритма.



После метода ветвей и границ.



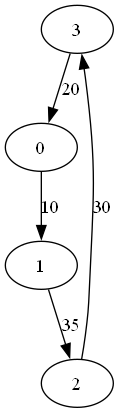


Рисунок 2. Жадный алгоритм

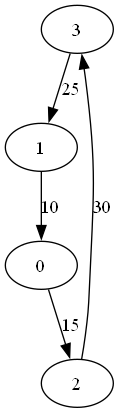
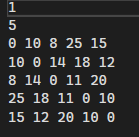


Рисунок 3. Метод ветвей и границ.

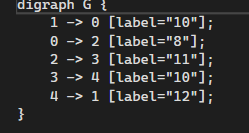
Тест 3

Входные данные:

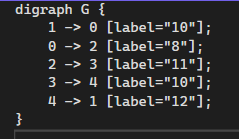


Выходные данные:

После жадного алгоритма.



После метода ветвей и границ.



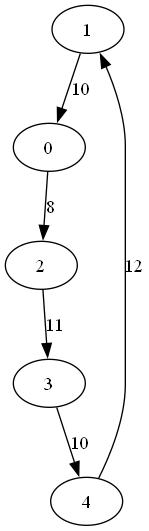


Рисунок 4. Жадный алгоритм; метод ветвей и границ

СПИСОК ЛИТЕРАТУРЫ

1. Э. Рейнгольд , Ю. Нивергельт, Н. Део, Комбинаторные алгоритмы, Мир, 1980 г. <http://www.uic.unn.ru/~zny/hu/contents.pdf>
2. <https://ru.hexlet.io/courses/algorithms-graphs/lessons/traveling-salesman-problem/theory_unit> , ru.hexlet.io