Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет: Электротехнический

Кафедра: «Информационные технологии и автоматизированные системы» (ИТАС)

Направление: Разработка программно-информационных систем (РИС)

ОТЧЁТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ

Дисциплина: «Основы алгоритмизации и программирования»

Тема: «Задача комивояжера»

Выполнил

Студент группы РИС-23-2б

Кобзев С.И.

Проверил

Доцент кафедры ИТАС

Полякова О. А.

Г. Пермь 2024

Постановка задачи

Имеется N городов. Выезжая из исходного города А1, коммивояжер должен побывать во всех городах по одному разу и вернуться в город А1. Задача заключается в определении последовательности объезда городов, при которой коммивояжеру требуется минимизировать некоторый критерий эффективности: стоимость проезда, время пути, суммарное расстояние.

Для расчета затрат существует матрица условий, содержащая затраты на переход из каждого города в каждый, при этом считается, что можно перейти из любого города в любой, кроме того же самого. Целью решения является нахождения маршрута, удовлетворяющего всем условиям и при этом имеющего минимальную сумму затрат.

Необходимо реализовать граф и решить для него задачу Коммивояжера методом ветвей и границ.

Для решения задачи необходимо:

* Создать класс Graph
* Решить для данного графа задачу Коммивояжера
* Визуализировать граф
* Визуализировать решение задачи Коммивояжера

Анализ задачи

Для решения задачи коммивояжера составляется матрица условий, которая содержит расстояния между городами. Считается, что можно перейти из любого города в любой, кроме того же самого. Так как пути из города А в город А по условию нет, обозначим его «нн».

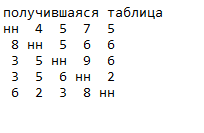


Рисунок 3.2 – Матрица условий

После построения изначальной матрицы условий проводим редукцию строк и столбцов.

Редукция строк: находим минимальное расстояние в строке, вычитаем его из всех расстояний в строке, кроме «нн».

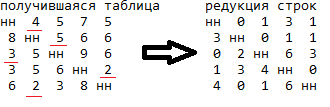


Рисунок 3.3 – Редукция строк (минимальные элементы подчеркнуты)

Редукция столбцов: находим минимальное расстояние в столбце, вычитаем его из всех расстояний в столбце, кроме «нн».

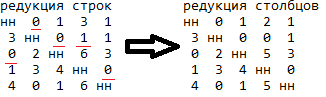


Рисунок 3.4 – Редукция столбцов (мин. элементы подчеркнуты)

Оценка нулей: каждый 0 в таблице оцениваем: считаем сумму минимального элемента в строке и в столбце и запоминаем результат.

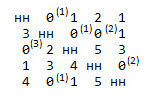


Рисунок 3.5 – Оценка нулей

Чистка карты: выбираем 0 с наибольшей оценкой, если таких несколько, выбираем любой, вычеркиваем строку и столбец, в котором находится этот 0 (вычеркивание — замена элементов строки и столбца на «нн»).

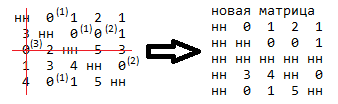


Рисунок 3.6 – Чистка карты

Координаты вычеркнутого столбца и строки запоминаем, назовем их ребрами. Повторяем редукцию столбцов, строк и чистку карты, пока последний переход не станет очевидным (сведем матрицу к размерам 2х2).

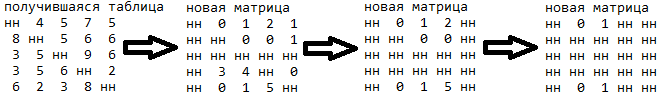


Рисунок 3.7 – Сведение исходной матрицы к матрице 2х2

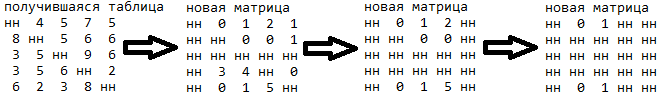


Рисунок 3.8 – Сведение исходной матрицы к матрице 2х2

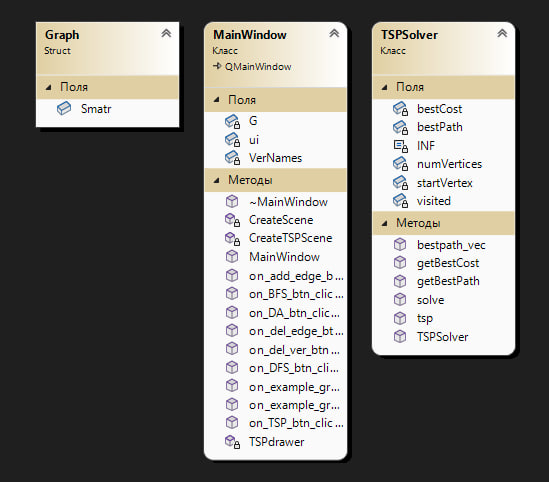
Полученные ребра выстраиваем в последовательность и получаем наименьший путь, из исходной матрицы берем расстояния, двигаясь по проложенному пути, получаем оптимальное расстояние всего пути.

Результат решения задачи коммивояжера:

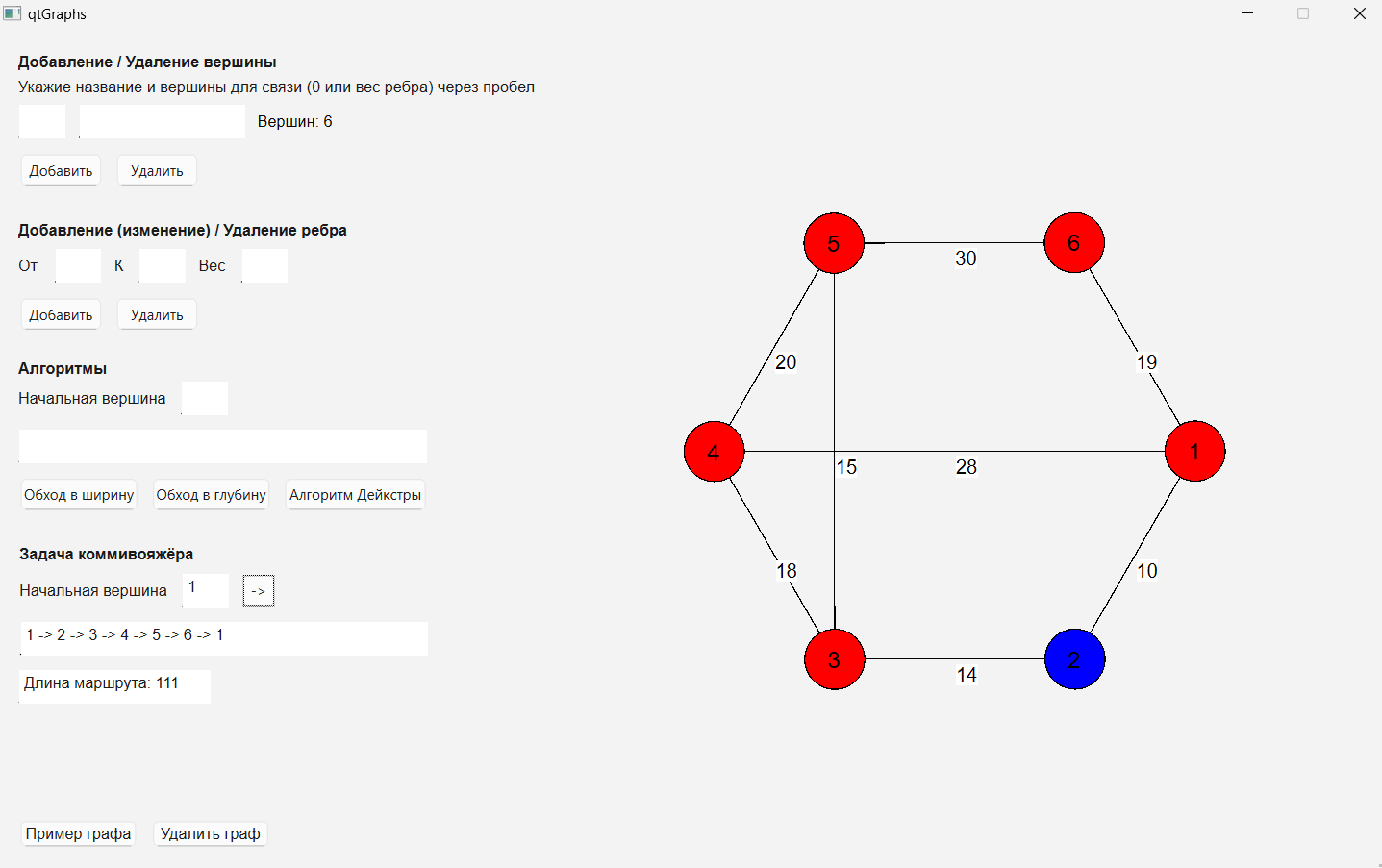
3->1->2->4->5->3

* Базовый класс Graph представлен в виде массива матрицы смежности.
* Для решения задачи комивояжера использован метод ветвей и границ, который позволяет находить оптимальное решение путём перебора всех возможных вариантов маршрутов, при этом использует бинарное дерево, чтобы эффективно сократить пространство поиска.
* Для визуализации использована библиотека QGraphics.Вершины графа отрисовываются по функции вставки эллипса по кругу, от вершины к вершине проходит ребро под необходимым вычисленным углом. Также по середине ребра располагается его вес, а на вершинах название вершины.
* Для визуализации решения задачи Комивояжера выводится кратчайший путь и его длина. На графе синим цветом отображаются вершины, которые посещаются в данный момент.

UML Диаграмма классов



Работа программы



Исходный код

Исходный код представлен в репозитории git-hub: https://github.com/Helnex/ARM-TSP/tree/main/TSP