Лабораторная работа № 9 по курсу дискретного анализа: Графы

Выполнил студент группы 08-208 МАИ *Куликов Алексей*.

Условие

Разработать программу на языке C или C++, реализующую указанный алгоритм. Формат входных и выходных данных описан в варианте задания. Первый тест в проверяющей системе совпадает с примером.

В качестве конкретного задания предлагается решить следующую задачу (вариант 4):

Задан взвешенный неориентированный граф, состоящий из n вершин и m ребер. Вершины пронумерованы целыми числами от 1 до n. Необходимо найти длину кратчайшего пути из вершины с номером `start` в вершину с номером `finish` при помощи алгоритма Дейкстры. Длина пути равна сумме весов ребер на этом пути. Граф не содержит петель и кратных ребер.

Метод решения

Решением является стандартный алгоритм Дейкстры с использованием очереди с приоритетами.

На каждом этапе выбирается такая вершина `minDistVert`, которая не использовалась ранее и длина пути `dist[minDistVert]` от которой до стартовой вершины наименьший (такую вершину можно быстро получить из очереди с приоритетом). Далее для каждой из смежных с ней вершин `to` запускаем процедуру релаксации. Релаксация – по сути, улучшение текущего решения, если это возможно. Процедура релаксации: `dist[to] = min(dist[to], dist[minDistVert] + len)`, где `len` – длина дуги от вершины `minDistVert` до рассматириваемой смежной `to`.

Релаксации повторяются до тех пор, пока не будут пройдены все вершины, либо пока не останется вершин, удаленных от стартовой менее чем на бесконечность. Это будет означать, что такие вершины по-просту никак не связаны со стартовой и из нее до подобных вершин не добраться.

В результате в `dist[i]` будет храниться длина кратчайшего пути от вершины `start` до `i`-й вершины.

Описание программы

Программа состоит из единственного исходного файла. В нем записано решение согласно алгоритму, описанному выше.

Дневник отладки

1. 07.05 Не проходит тест номер 14.
   РЕШЕНИЕ: Пришел к выводу, что после удачной релаксации одного из путей ломалась куча из-за того, что функция-компаратор использовала актуальные данные о кратчайших расстояниях, которые в свою очередь успевали меняться до того, как до соответствующей вершины могла бы дойти очередь. Переписал с использованием данных на момент добавленийя. Т.о. вновь добавляемое состояние для вершины может иметь только меньшее значение расстояния, чем предыдущее, и значит, окажется выше в куче, будет взята раньше, а каждую вершину обрабатываем только раз (помечаем в векторе `used`). Т.о. алгоритм всегда выбирает вершину с наименьшим текущим расстоянием от старта.

2. 07.05 Не проходит тест номер 7.
   РЕШЕНИЕ: Ошибся с компаратором и строил кучу не по убыванию, а по возрастанию. Заменил компаратор на `std::greater`.

Выводы

В решении данной задачи самым трудным оказалось понять ошибку с функцией-компаратором, использующей актуальный вектор дистанций. Оказвается свежее не всегда лучшее.

Алгоритм Дейкстры может быть использован везде, где нужно найти кратчайший путь от вершины до вершины (вершин). Например, в для нахождения кратчайших маршрутов между точками на карте. Так же может быть использован в системах коммуникации для IP маршрутизаци и т.д. и т.п.