Лабораторная работа № 9 по курсу дискретного анализа: Графы

Выполнил студент группы 8О-207Б-17 МАИ *Ваньков Денис.*

Условие

Разработать алгоритм решения задачи, определяемой своим вариантом; Доказать его корректность, оценить время работы алгоритма и объем затраченной оперативной памяти.

Реализовать программу на C или $C++$, реализующую построенный алгоритм. Формат входных и выходных данных описан в варианте задания.

Вариант задания: 5. Поиск кратчайшего пути между парой вершин алгоритмом Беллмана-Форда

Задан взвешенный ориентированный 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 Беллмана-Форда. Длина пути равна сумме весов ребер на этом пути. Обратите внимание, что в данном варианте веса ребер могут быть отрицательными, поскольку алгоритм умеет с ними работать. Граф не содержит петель, кратных ребер и циклов отрицательного веса.

Входные данные:

В первой строке заданы $1 \leq n \leq 10^5$, $1 \leq m \leq 3*10^5$, $1 \leq start \leq n$ и $1 \leq finish \leq n$. В следующих m строках записаны ребра. Каждая строка содержит три числа – номера вершин, соединенных ребром, и вес данного ребра. Вес ребра – целое число от -10^9 до 10^9.

Выходные данные:

Необходимо вывести одно число - длину кратчайшего пути между указанными вершинами. Если пути между указанными вершинами не существует, следует вывести строку "No solution"(без кавычек).

Метод решения

В отличие от алгоритма Дейкстры, этот алгоритм применим также и к графам, содержащим рёбра отрицательного веса. Мы считаем, что граф не содержит цикла отрицательного веса.

Заведём массив расстояний $d[0 \ldots n-1]$, который после отработки алгоритма будет содержать ответ на задачу(В каждом индексе будет храниться кратчайший путь до вершин). В начале работы мы заполняем его следующим образом: $d[start] = 0$, а все остальные элементы равны ∞.

Алгоритм Форда-Беллмана состоит из несколько фаз. На каждой фазе просматриваются все рёбра графа, и алгоритм пытается произвести релаксацию (relax) вдоль каждого ребра $(from, to)$ стоимости $cost$ — это попытка улучшить значение $d[to]$ значением $d[from] + cost$. То есть это значит, что мы пытаемся улучшить ответ для вершины to, пользуясь ребром $(from, to)$ и текущим ответом для вершины $from$.

Утверждается, что достаточно $n - 1$ фазы алгоритма, чтобы корректно посчитать длины всех кратчайших путей в графе. Для недостижимых вершин расстояние останется равным ∞.

Описание программы

В данном алгоритме удобнее всего хранить граф в ребрах. Для этого я создал струтуру $edge$, в которой были 3 значения: два типа $uint\_32t$ $from$ и to, так как номер вершины не может быть меньше 1, и одно типа $int\_64tcost$. Алгоритм просматривает ребра и, пока мы не дошли до конца, не заканчивает цикл. Внутри цикла алгоритм сравнивает значение в массиве и если оно больше, чем сумма предыдущего и веса, то к предыдщему значению прибавляет вес ребра. Для его ускорения я создал переменную any которая проверяет изменилось ли значение в массиве $d[]$, если да, то заканчивает цикл, чтобы избежать ненужных просмотров и сравнений.

Дневник отладки

В процессе отладке удалось отследить что для решения этой типа int для значения cost не хватит, для этого был использован тип int_64t, что привело к правильной работе.

Выводы

При написании этой лабораторной я не столкнулся с трудностями, так как для нее были нужны базовые знания графов, а также я уже встречался с этим алгоритмом в курсе дискретной математики.

Корректность алгоритма:

Для недостижимых из $start$ вершин алгоритм отработает корректно: для них метка $d[]$ так и останется равной бесконечности.

Нужно доказать: после выполнения i фаз алгоритм Форда-Беллмана корректно находит все кратчайшие пути, длина которых (по числу рёбер) не превосходит i.

То есть для любой вершины $from$ обозначим через k число рёбер в кратчайшем пути до неё (если таких путей несколько, можно взять любой). Тогда это утверждение говорит о том, что после k фаз этот кратчайший путь будет найден гарантированно.

Доказательство. Рассмотрим произвольную вершину $from$, до которой существует путь из стартовой вершины $start$, и рассмотрим кратчайший путь до неё: $(p_0 = start, p_1, \ldots, p_k = from)$. Перед первой фазой кратчайший путь до вершины $p_0 = start$ найден корректно. Во время первой фазы ребро (p_0, p_1) было просмотрено, следовательно, расстояние до вершины p_1 было корректно посчитано после первой фазы. Повторяя эти утверждения k раз, получаем, что после k-ой фазы расстояние до вершины $p_k = from$ посчитано корректно, что и требовалось доказать.

Последнее, что надо заметить — это то, что любой кратчайший путь не может иметь более $n-1$ ребра. Следовательно, алгоритму достаточно произвести только $n-1$ фазу. После этого ни одна релаксация гарантированно не может завершиться улучшением расстояния до какой-то вершины.