Основные алгоритмы 10

Ковалев Алексей

1. Запустим поиск в ширину из вершины s. Это даст нам массив расстояний от вершины s до всех остальных. Затем запустим поиск в шлубину из вершины t, при обходе графа которым будем помечать те вершины, при переходе в которые из данной расстояние уменьшается на 1. Тогда все помеченные вершины и только они будут вершинами, лежащими на каком-то кратчайшем пути из s в t. Алгоритм корректен, так как поиск в ширину дает нам все корректные расстояния от вершины s до всех, а поиск в глубину из t восстанавливает кратчайшие пути, так как при обработке вершины v помечает только те вершины u, для которых `dist[`u`]` = `dist[`u`]` $- 1$. При этом все вершины, лежащие на кратчайших путях помечены, так как для любой такой вершины на расстоянии d существует хотя бы одна смежная вершина на расстоянии $d + 1$, которая будет обработана поиском в глубину. Алгоритм линеен в силу того, что и поиск в глубину, и поиск в ширину линейны.

2.

1. Корректность алгоритма следует из корректности обычного алгоритма Дейкстры, так как если в графе нет отрицательных ребер, то ни одно ребро не попадет в очередь два раза, поскольку после извлечения некоторой вершины v из очереди гарантируется, что найдено минимальное расстояние до v, а значит при любой релаксации расстояние до нее не уменьшится.

2. Обычный алгоритм Дейкстры является жадным, поэтому не работает на графах с ребрами отрицательного веса. Приведенная модификация алгоритма будет работать на таких графах, так как после релакцсации некоторой вершины v эта вершина вновь будет добавлена в кучу, а значит позже будет извлечена из нее и все ее соседи вновь прорелаксируются. То есть извлечение вершины из кучи еще не гарантирует, что кратчайший путь до нее уже обнаружен, но он обязательно будет обнаржен, поскольку некоторая уже извлеченная вершина все равно будет прорелаксированна всеми входящими в нее ребрами, что приведет, возможно, к уменьшению пути до нее.
   Вообще, если в неориентированном графе есть ребро отрицательного веса (допустим, граф связен), то все расстояния в нем равны минус бесконечности, потому что можно дойти от любой вершины до отрицательного ребра и бесконечно ходить по нему туда-сюда. Если граф не связен, то то же самое верно для компонент связности, в которых есть отрицательное ребро, а утверждение пункта 1 верно для остальных компонент связности. Именно это и будет делать модифицированный алгоритм: если вес (u, v) отрицателен, то он будет релаксировать v через u, потом u через v, потом v через u и так далее, то есть расстояния действительно окажутся минус бесконечность.
   Сложность модифицированного алгоритма – $O\big(|V|^2 \log|V|\big)$, потому что в худшем случае каждая вершина будет релаксироваться каждой другой вершиной. Сложность алгоритма Форда-Беллмана – $O(|V||E|)$, что может быть и меньше, и больше сложности модифицированного алгоритма.

3. Цикл отрицательного веса в графе можно обнаружить, если после релаксации всех вершин всеми их соседями по одному разу очередь не пуста. То есть если после $|V|^2$ операций извлечения или ранее очередь не опустела, то в графе есть цикл отрицательного веса.

3. Такой подход некорректен. Рассмотрим граф на трех вершинах с тремя ребрами с весами -1, -2, 1. Пусть вершины называются a, b, c, веса ребер `cost(`a, b`)` $= -1$, `cost(`b, c`)` $= -2$, `cost(`c, a`)` $= 1$. Тогда длина кратчайшего пути от a до b равна $-\infty$, а сам путь будет бесконечно раз обходить цикл. Если добавить к весам константу так, чтобы все веса стали положительными, то кратчайший путь от a до b будет состоять лишь из ребра (a, b).

4. Для поиска кратчайших путей от некоторой вершины до всех остальных в таком графе можно использовать 0-1`bfs` работающий за линейной от размеров графа время.

5. Для начала транспонируем все ребра графа, то есть ребро (u, v) заменим на (v, u), сохраняя веса. Затем домножим все веса в графе на -1. Пусть была дана некоторая вершина s, а правильный ответ на задачу – вершина t, то есть расстояние от t до s максимально в исходном графе. Тогда в новом графе минимально расстояние от s до t. Теперь воспользуемся алгоритмом Беллмана-Форда, запустив его из вершины s. Это даст нам расстояние от вершины s до всех остальных, причем циклы отрицательного веса будут обработаны правильно. Тогда вершина, расстояние до которой минимально в новом графе – это и есть искомая вершина t. Корректность алгоритма следует из корректности алгоритма Беллмана-Форда. Сложность алгоритма также совпадает со сложностью алгоритма Беллмана-Форда, то есть $O(|V||E|)$, так как мы делаем лишь его и несколько линейных обходов.

6. Запустим поиск в ширину в этом дереве. Это даст нам массив расстояний, то есть мы научимся по вершине понимать, в каком слое она лежит (вершина v лижет в k-ом слое, если $\texttt{dist}[v] = k$). Далее задача решается так: обозначим за $m(v)$ мощность максимального множества вершин в дереве, подвешенном за вершину v, в котором есть только те вершины u, для которых $\texttt{dist}[v] \leqslant \texttt{dist}[u]$. Тогда мы можем выразить

$$m(v) = \max(1 + \sum_{w:\,\texttt{dist}[w]=\texttt{dist}[v]+2} m(w), \sum_{w:\,\texttt{dist}[w]=\texttt{dist}[v]+1} m(w))$$

Значит значение $m(v)$ можно рекурсивно посчитать за линейное от размеров графа время, так как каждая вершина и кадое ребро будет обрабатываться алгоритмом лишь константное число раз. Если при вычислении $m(v)$ также помечать вершины, для которых упомянутая сумма оказалась максимальна, то можно найти и само независимое множество. Корректность алгоритма объясняется тем, что в дерево можно представить как двудольный граф, в котором нет циклов.