2 задача: Цикл минимальной длины

1 модуль, 2 семестр

ФИВТ МФТИ, 2019

Описание by Илья Белов

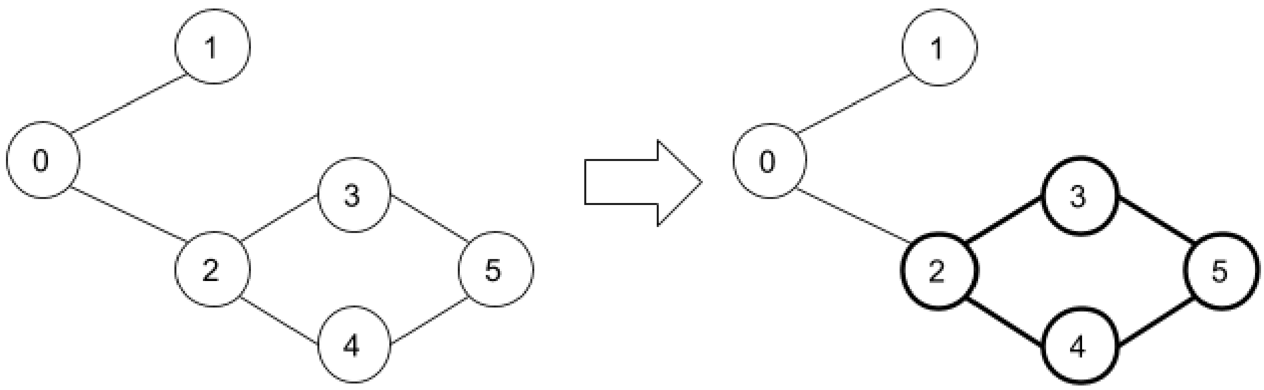
1. Текст задачи

Дан невзвешенный неориентированный граф. Найдите цикл минимальной длины.
Ввод: v:кол-во вершин(макс. 50000), n:кол-во ребер(макс. 200000), n пар реберных вершин
Вывод: одно целое число равное длине минимального цикла. Если цикла нет, то вывести -1.

Пример:

in	out
6 6 0 1 0 2 2 3 2 4 3 5 4 5	4

Иллюстрация примера:

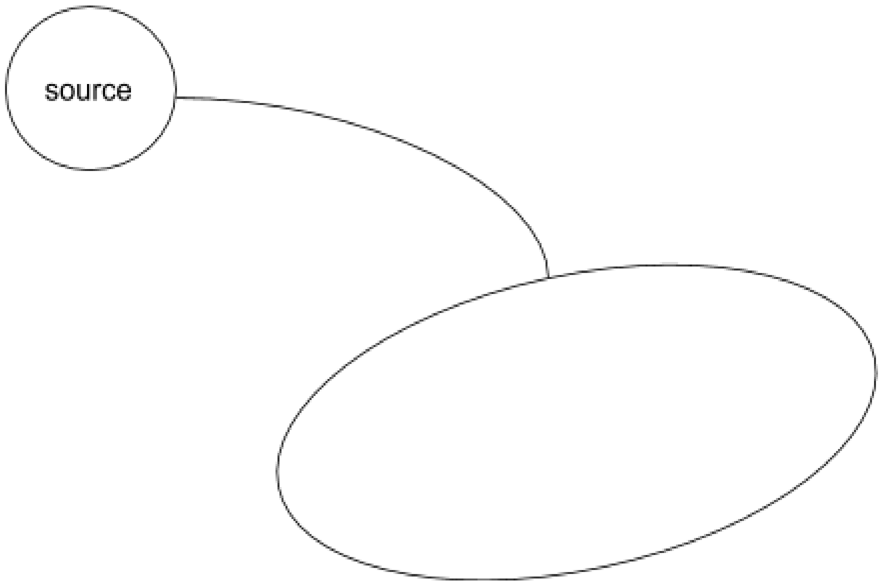


2. Описание алгоритма

Для каждой вершины запускается BFS. Как только BFS нашёл цикл, запоминаем длину цикла, которая находится по формуле length = distance[current] + distance[next] + 1. Переходим к следующему запуску BFS. Минимальная длина цикла по всем BFS будет искомой длиной минимального цикла в графе

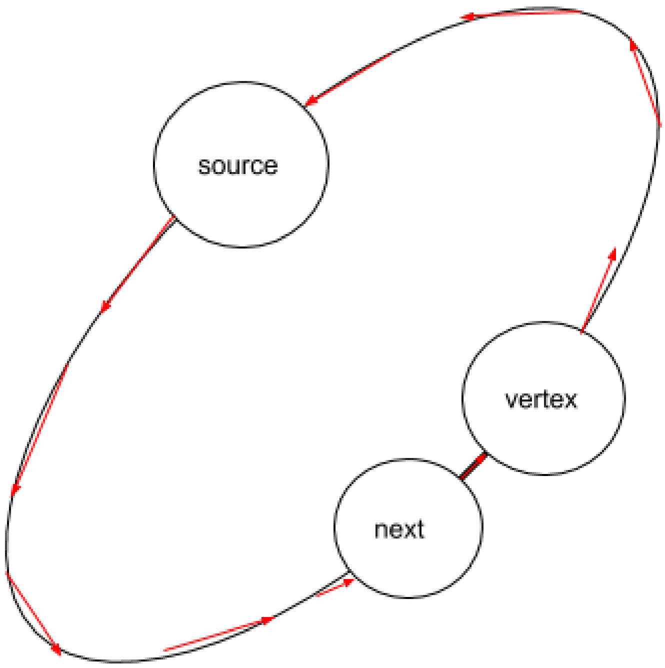
3. Доказательство корректности

а) Почему первый найденный цикл будет минимальным? Предположим обратное: минимальный цикл будет найден после первого найденного. Но у всех последующих циклов distance[current] или distance[next] будет больше, чем у первого цикла (так как BFS обрабатывает вершины в порядке увеличения расстояния), значит все такие циклы будут иметь б**о**льшую длину. Следовательно, первый найденный цикл будет минимальным и дальше продолжать поиск бессмысленно б) Почему необходимо запускать BFS от каждой вершины? Цикл будет найден и его длина определится правильно только в том случае, когда source (стартовая вершина) лежит на цикле. Иначе возможна подобная ситуация:



в) Почему если вершина лежит на минимальном цикле, BFS его найдет? Предположим обратное. Пусть он найдёт другой цикл, но это противоречит доказанному в а). Пусть он не найдёт ни одного цикла. Но ведь BFS проходит по всем вершинам и всем рёбрам, и если BFS никогда не зайдёт в уже посещённую вершину, то это значит, что он обходил дерево и циклов нет вовсе

г) Почему BFS из вершины на минимальном цикле найдёт этот цикл и найденная по вышеуказанной формуле длина будет длиной цикла? Если мы попали в уже посещённую вершину, значит в неё есть второй путь от source, значит есть цикл: из текущей вершины в source (distance[vertex]), из source в next (distance[next]) и из next в vertex (1), в итоге получаем формулу distance[vertex]+distance[next] + 1. Когда же source не лежит на цикле, к этой длине добавляется удвоенная длина “аппендикса” от цикла до sorce. Суммарная длина будет больше чем длина цикла, соответственно в ответ не попадёт



4. Время работы и дополнительная память

$T = O(V^2 + VE)$

$M = O(V)$

5. Доказательство времени работы и дополнительной памяти

а) Время работы:

Найдём время работы BFS[1]. В очередь добавляются только непосещённые вершины, операции добавления и удаления выполняются за $O(1)$, следовательно время работы с очередью составляет $O(V)$. При извлечении вершины из очереди обрабатываются все исходящие рёбра, а так как каждая вершина обрабатывается один раз, то каждое ребро так же обрабатывается единожды. Обработка ребра занимает $O(1)$, значит работа со всеми рёбрами составляет $O(E)$. Получаем, что суммарное время работы алгоритма равно $O(V + E)$.

В описываемом алгоритме BFS вызывается для каждой вершины, т. е. V раз. В итоге $T = O(V(V + E)) = O(V^2 + VE)$

б) Дополнительная память:

Дополнительная память выделяется на очередь для BFS, буфер для дочерних вершин, и расстояния до каждой вершины. Размер каждого из этих контейнеров не превышает $V \Rightarrow M = O(V)$

6. Описание модификаций алгоритма

Модификация: приспособить DFS как более очевидного кандидата на ищущий циклы алгоритм, но почему-то реализация на DFS упорно отказывается работать. Асимптотика времени тогда значительно бы улучшилась и стала бы $T = O(V + E)$

[1] Доказательство времени работы BFS приводится только в этом описании, в последующих оно будет приниматься за данное