9.1. Проверка, является ли остовное дерево деревом кратчайших путей.

Граф взвешенный? Ответа на письмо пока не пришло, но я думаю, да

Для начала пройдем по выданному дереву обходом в ширину и проставим каждой вершине расстояние (минимальное, оно же единственное) до нее от исходной вершины S. Во время обхода будем помечать ребра, по которым мы прошли.

После обхода в графе останутся непомеченными только те ребра, которые дереву не принадлежат.

Теперь, согласно определению кратчайших путей, процедура Relax, запущенная на каждом из непомеченных ребер вида (u, v) не должна ослабить расстояние от вершины u до v. Если такое изменение произошло – данное нам дерево не является деревом кратчайших путей, иначе – является.

Сложность: O(V+E), так как суммарно обе части алгоритма прошли по всем ребрам графа (сначала по дереву, затем по остальным ребрам), а Relax делается за O(1).

Relax – это процедура обновления расстояния до очередной вершины v из u:

```
if d[v] > d[u] + w(u, v):
    d[v] = d[u] + w(u, v)
```

9.2. Найти число кратчайших путей в неориентированном графе между двумя выделенными вершинами.

<u>Directed graph:</u>

Можно слегка модифицировать алгоритм Дейкстры, добавив в процедуру Relax шаг, который обновляет массив count – массив количества путей между вершиной s и текущей.

```
if d[v] > d[u] + w(u, v):
    d[v] = d[u] + w(u, v)
    count[v] = count[u]
else if d[v] = d[u] + w(u, v):
    count[v] = count[v] + count[u]
```

Изначально этот массив проинициализирован нулем для всех вершин, кроме исходной: для нее это значение равно 1. Как обновляется массив ясно: если удалось срелаксировать расстоние до вершины, то число путей добраться в нее равно числу путей добраться до текущей вершины u; а если пути равны, то число способов добраться до v есть их сумма.

Ответ: значение count[v].

Сложность = сложность алг. Дейкстры O(V*V + E) или O((V+E) * log V), смотря какой граф.

<u>Undirected graph:</u>

А тут можно использовать слегка модифицированный BFS. Запустим его из первой вершины и пусть в каждой вершине графа хранится дополнительное поле count, которое считает число путей из начальной вершины в текущую. Изначально всем вершинам припишем ноль, кроме начальной — там будет единица.

Далее, как только BFS начнет обрабатывать вершину v, проверим всех соседей этой вершины и запишем в поле count вершины v сумму всех count'ов всех соседей, уровень которых в дереве поиска в ширину ровно на единицу меньше уровня вершины v (другими словами сумму всех родителей).

Понятно почему это работает: все кратчайшие пути от вершины s до вершины v, лежащей на уровне k будут длины k, и каждая вершина лежащая между ними, скажем, i, будет лежать ровно на уровне i в дереве поиска в ширину.

Сложность не больше BFS, несмотря на то, что мы у каждой вершины просматриваем всех ее соседей, однако число таких просмотров будет ровно 2E, значит общее время O(V+E).

9.3. Найти кратчайшие пути во взвешенном ориентированном графе, проходящие через выделенную вершину s.

Найдем все кратчайшие пути из вершины s во все вершины, которые достижимы из s. – это делает алгоритм Дейкстры. Затем сделаем то же самое, только в инвертированном графе – это нам даст значения всех кратчайших путей *до* вершины s изо всех, из которых она достижима. Дальше несложно получить ответ: длина пути будет сумма значений d[v]+d[u] (если какого-то числа не будет (будет начальное значение – бесконечность), значит вершина недостижима). Заведя массив предков несложно получить и сам путь.

Сложность – дважды алгоритм Дейкстры плюс время на инвертирование ребер.

Для разных графов применима оценка, которая давалась в лекции:

Для плотных графов можно использовать в качестве «очереди» обычный массив – сложность будет O(V*V).

В случае разреженных графов бинарная куча: сложность O((V+E)log V).

В промежуточном варианте: d-ичная куча.

9.4. Дан взвешенный ориентированный граф и выделенная вершина. Построить дерево «кратчайших путей» из выделенной вершины во все остальные по двум метрикам.

а) Вес пути определяется весом самого тяжелого ребра.

Эту задачу решает слегка модифицированный алгоритм Дейкстры: процедура обновления будет несколько иной:

*foreach **u** in Adj[v]:*
*d[**u**] = min(d[**u**], max(d[v], w(v, **u**)))*

Обновление логичное: мы стараемся уменьшить вес до очередной вершины, причем новый вес есть максимум из весов до текущей вершины и нового ребра (только оно может увеличить «расстоние» между вершинами).

Все остальное аналогично обычному алгоритму Дейкстры, время работы соответствующее.

б) Вес пути определяется парой весов самого тяжелого ребра и следующего за ним. Пары сравнивается по элементно: сначала первые элемент; если они равны, то вторые.

Ее стоит решать так же, как и в предыдущем пункте, только нужно корректно обновлять вес до новой вершины. В предыдущем пункте это был просто максимум между текущим весом и весом до новой вершины.

В этой же задаче вес определяется двумя максимальными весами в пути, поэтому в каждой вершине стоит хранить два максимальных веса в пути до нее из начальной вершины. Затем для каждой вершины u расстоние до нее (то есть пара весов) обновляется так: если вес ребра больше первого веса в паре, то он становится самым тяжелым ребром и заменяет первой вес, в противном случае аналогичное сравнение происходит со вторым весом в паре.

Я вот подумал: а что такое «следующее за ним»? Следующее по весу или следующее в пути до очередной вершины?

9.5. Сломается ли алгоритм Дейкстры в графе, где отрицательные ребра могут выходить только из одной выделенной (начальной) вершины?

- Да, если есть циклы отрицательной длины (это очевидно).
- Нет, если циклов отрицательный длины нет.

Это следует из анализа доказательства корректности алгоритма Дейкстры: единственным местом в доказательстве где нам нужна была неотрицательность весов ребер было рассмотрение случая, а нет ли другого пути через узел, который мы еще не посещали.

Или, по Кормену: когда доказывается корректность (а именно, сохранение инварианта цикла: в начале каждой итерации цикла while множество S содержит вершины, для которых уже точно вычислены кратчайшие расстояния и они не могут меняться), предполагается обратное и единственное место, где нужна неотрицательность, это неравенство D(s, y) <= D(s, u), где s – исток, u – текущая обрабатываемая вершина, в предположении, что u – первая такая добавленная вершина в S, для который вышесказанный инвариант не выполняется, а y – первая вершина на пути от истока к u, которая не лежит в S. Но так как отрицательные ребра выходят непосредственно из истока, то это никак не вредит доказательству и это неравенство корректно работает, т.к. на пути от s до u все ребра имеют неотрицательный вес.

Значит, в таком случае алгоритм Дейкстры не может сломаться на таком графе.