1.

Неориентированный граф можно рассматривать как ориентированный на V вершинах с $2|E|$ ребрами: как в ориентированном графе и их транспонированных (перевернутых).

Запускаем алгоритм Дейкстры, на выходе будем иметь дерево кратчайших путей (было показано на лекции). Чтобы найти количество вершин в кратчайшем пути из s в t, необходимо найти глубину в дереве, на которой находится вершина t (s расположена в корне).

Будем искать вершину t в дереве и увеличивать счетчик на единицу, пока не дойдем до нее.

Корректность:

То, что в результате работы алгоритма Дейкстры получается дерево, было доказано на лекции. Количество вершин в пути находится по дереву путем подсчета вершин до конца – вершины s.

Сложность:

Сложность алгоритма Дейкстры: $O(|V| + |E|)$ (доказано на лекции), поиск нужной вершины в дереве: $O(\log|V|)$, так что итоговая сложность: $O(|V| + |E|)$.

2.

1. Данный алгоритм выполняет практически те же шаги, что и алгоритм Дейкстры, однако добавляет вершину в очередь только при изменении расстояния в результате процедуры релаксации и позволяет многократно обращаться к уже просмотренной вершине, что необходимо в следующем пункте. В случае отсутствия ребер отрицательного веса этот алгоритм поддерживает тот же инвариант, что и исходный алгоритм: минимум в очереди не убывает. Также гарантируется, что каждая вершина хоть раз побывает в очереди, не пропустим ни одной вершины.

2. В исходном алгоритме Дейкстры закрыв некоторую вершину, мы уже больше не могли к ней вернуться. Данный алгоритм корректно работал для графов с ребрами неотрицательно веса, ведь действительно, путем добавления любого неотрицательно числа, путь не может стать меньше по весу. Однако это неверно в случае отрицательных весов, и нужен алгоритм, который сможет оглядываться назад.

Данная модификация добавляет вершину в очередь, когда ее там нет в данный момент, что может возникнуть в двух ситуациях: мы ее вообще еще не просматривали и уже просматривали, но нашли более длинный по количеству ребер путь, но с меньшим весом за счет отрицательных ребер, тогда ее приоритет обновиться на меньший. Таким образом, гарантировано каждая вершина хоть раз побывает в очереди, не пропустим ни одной вершины, а также, найдем кратчайший путь до вершины, которая была уже однажды закрыта, но будет иметь меньший путь за счет отрицательных ребер.

Сложность:

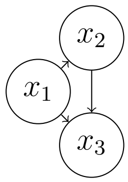
В худшем случае минимальный путь будет проходить по последнему просмотренному ребру, то есть число сборов/разборов кучи будет: $O(\sum_{u\in V} d(u)) = O(|E|)$. Число релаксаций по-прежнему равно числу ребер, так что итоговая сложность: $O(|E|\log V)$.

Сложность алгоритма Беллмана-Форда: $O(|V||E|)$. Таким образом, модификация алгоритма Дейкстры работает быстрее за счет оптимальной структуры данных.

3. Для того чтобы обнаружить цикл отрицательного веса, будем просматривать массив предков всякий раз, когда получается отрицательный приоритет у вершины. Если в массиве предков уже есть эта вершина, то это цикл, а так как приоритет отрицательный, значит, он отрицательного веса. В этом случае будет выдано сообщение об ошибке.

3.

Такой подход некорректен. Пример такого графа:



И пусть веса ребер: $w_{1-2} = -1$, $w_{2-3} = 1$, $w_{1-3} = 1$.

Согласно алгоритму прибавим константу, чтоб все веса стали положительными, например, 2, тогда $w'_{1-2} = 1$, $w'_{2-3} = 3$, $w'_{1-3} = 3$.

Кратчайший путь из 1 в 3 равен: $w_{1-2} + w_{2-3} = 0$, однако после прибавления константы получим: $w'_{1-2} + w'_{2-3} = 4$.

Следовательно, алгоритм некорректен.

4.

В отличие от стандартного алгоритма Дейкстры, в котором структура данных – очередь с приоритетом, заведем двустороннюю очередь (дек).

Сначала в вершине дека есть только исток. Если мы прошли ребро веса 0, то добавляем вершину в начало двусторонней очереди, иначе – в конец. Таким образом, мы сохраним очередь отсортированной в любой момент времени. Гарантировано, что добавляемая вершина находится либо на том же уровне, что и из которой идем, или на уровень глубже (так как веса ребер 0 или 1). То есть, очередь содержит элементы двух последовательных уровней, так что описанное добавление сохраняет в ней порядок. Остальные шаги аналогичные традиционному алгоритму Дейкстры, описанному на лекции.

Корректность:

Двусторонняя очередь позволяет сохранять порядок в структуре данных, так как гарантировано, что добавляемая вершина находится либо на том же уровне, что и из которой идем, или на уровень глубже (так как веса ребер 0 или 1). Таким образом, сверху всегда находится минимальный элемент. Остальные шаги аналогичные традиционному алгоритму Дейкстры, описанному на лекции.

Сложность:

Поиск минимума в двусторонней очереди при данном добавлении элементов: $O(1)$ (минимальный элемент всегда сверху), добавление вершины в очередь также $O(1)$. Таким образом, релаксация происходит за $O(1)$. То есть, итого на релаксацию будет потрачено $O(|E|)$ (согласно лекции). На сбор/разбор очереди – $O(|V|)$, то есть, итоговая сложность: $O(|V| + |E|)$.

5.

Изменим процедуру релаксации следующим образом:
$if\ d[v] < d[u] + w[u, v]$
$d[v] = d[u] + w[u, v]$
То есть, будем наоборот увеличивать веса путей, если на определенном шаге он меньше.

За основу возьмем модифицированный алгоритм Дейкстры из задания №2, так как, возможно, придется оглядываться назад на вершину, ведь более длинный путь будет иметь больший вес, если новые ребра не имеют отрицательный вес.

Также необходимо завести структуру данных – очередь с приоритетом, но с извлечением максимального приоритета, а не минимального. На сложность это влияния не оказывает.

Корректность:

В целом алгоритм повторяет модифицированный алгоритм Дейкстры, однако отбирает максимальные пути за счет изменения процедуры релаксации. В этом алгоритме также поддерживается инвариант: максимум в очереди не убывает. За счет модификации этот алгоритм открывает вершину даже, если она была уже однажды закрыта, то есть это позволяет просмотреть длины всевозможных путей.

Сложность:

По сути, это модифицированный алгоритм Дейкстры, его время работы: $O(|E| \log V)$.

6.

Запустим алгоритм обхода графа в глубину и будем присваивать метки вершинам через один: 1 и -1. А затем возьмем множество вершин такого класса, чья мощность больше. Однако возможна ситуация, когда максимальное множетсво вершин составляют листья, это необходимо тоже учитывать. То есть, при обходе в глубину будем запоминать в отдельный класс вершины, у которых нет соседей. Тогда множество независимых вершин – это одно из трех описанных выше множеств, чья мощность максимальная.

Корректность:

При таком алгоритме из каждого ребра мы берем одну вершину в множество (случай с листьями отдельный), таким образом обеспечиваем, что никакие две вершины не соединены ребром. Листья дерева также никогда не соединены ребром, иначе был бы цикл, что приворечило бы определению дерева, аналогично между двумя ветвями не может быть ребра.

Сложность:

Мы проделываем обход в глубину: $O(|V|+|E|)$ (согласно прошлой лекции). Мощность каждого множетства можно считать сразу по ходу обхода. Следовательно, итоговая сложность: $O(|V| + |E|)$.