Тураев Тимур, 504 (SE)
27.10.2013

8.1. Кубик на клетчатом поле.

Пусть граф для данной задачи выглядит так: вершинами графа будут состояния кубика в каждой клетке поля, а ребрами – возможные смены состояний.

Под состоянием понимается вектор из всех 24 возможных ориентаций кубика на клетке (6 различных верхних граней и 4 поворота каждой грани), плюс для каждой ориентации можно хранить минимальное найденное число ходов, которое нужно сделать, чтобы перекатить кубик из начальной в текущую клетку.

Если стоит задача просто построить граф, то заведем 24*m*n вершины, каждой будет отвечать координата и ориентация. Из каждой вершины проведем максимум 4 ребра – т.к. существует 4 возможных перекатывания кубика (в соответствующие ориентации! – оно, очевидно, пересчитывается за O(1), хоть и не очень красиво – нужно аккуратно делать поворот кубика). Число вершин, очевидно, O(m*n), число ребер тоже O(m*n).

Чтобы решить задачу, сделаем BFS и посмотрим на число в нужной нам клетке – если там не бесконечность, то значение числа ходов и будет ответом.

8.2. Потоп в долине.

Граф построить просто: пусть вершинами будут клетки поля, а ребрами – возможные переходы между клетками. Таким образом, всего вершин будет ровно m*n, а ребер – не больше, чем 4*m*n.

Однако, ребра получаются взвешенными (их вес равен весовой функции из условия), а каждой вершине приписано некое число – высота. Плюс, в таком графе не сделаешь BFS для поиска минимального расстояния, в силу взвешенности.

Применим идею с практики: разобьем каждое ребро веса w на w ребер, введя w-1 фиктивную вершину. Получим неориентированный невзвешенный граф, в котором можно делать BFS. Число вершин в нем можно оценить так: каждое ребро (а их не больше 4*n*m) разбивается на максимум h ребер, поэтому добавляется еще O(n*m*h) вершин, значит общее их число: O(h*n*m). Реберная оценка аналогичная.

Но у нас еще есть вода, которая поднимается, ее учитывать просто: если мы во время обхода в ширину добрались до какой-либо вершины позже, чем туда прибывает вода (это легко проверять: раз у нас есть высота вершины и время, то легко вычислить высоту воды в долине), то оттуда не делаем ходов (то есть не кладем соседние вершины в очередь).

8.3. Аэропорты.

Минимальный размер бака – это максимальное “расстояние” между любыми двумя аэропортами. Под расстоянием понимается количество топлива, нужное для того, чтобы перелететь между аэропортами.

Очевидно, что это так. Пусть, максимальное расстояние между двумя аэропортами есть W (из аэропорта i в эропорт j). Тогда нельзя брать бак, размером меньше W – иначе невозможно будет перелететь из i в j. Но и больше не нужно: самолет может дозаправляться в каждом городе на всем пути следования.

Осталось найти это максимальное ребро в полном графе, вершинами которого являются аэропорты, а весами взвешенных ребер – тот объем горючего, которое нужно потратить на перелет между городами-вершинами. Это сделать просто – перебираем все ребра и поддерживаем максимум W. Всего ребер $n \cdot (n-1) = O(n^2)$. $O(\log W)$ – время, необходимо на хранение и поддержку максимума (числа W). Общая сложность – $O(n^2 \log W)$.

8.4. Граф с весами ребер 0 или 1. Найти кратчайший путь между двумя вершинами.

Кратчайший путь (в смысле количества ребер) умеет считать BFS. В нашем случае, граф взвешенный и хотелось бы, чтобы вершины, расстояние до которых от текущей равно нулю, обработались раньше, чем те, расстояние до которых равно 1. Для этого можно использовать дек: если текущее ребро имеет вес ноль, то конечную вершину кладем в начало дека, если один – то в конец. Причем класть в дек нужно лишь в том случае, если новый найденный путь для вершины оказывается лучше, чем тот, который уже нашли.

Таким образом, в деке в любой момент времени будут в начале лежать вершины, расстояние до которых k, а в конце – k или, может быть, k+1.

Легко понять, что в таком случае каждая вершина попадет в дек не более двух раз: если в первое попадание в деке не было вершин с меньшим расстоянием, то второй раз эта вершина в дек не попадет, ибо нет такой вершины, которое уменьшит расстояние до нее. Если же в деке были уже вершины, расстояние до которых меньше на единицу, то они смогут улучшить найденное расстояние один раз, и поэтому она попадет в очередь второй раз.

Исходя из этого, можно заключить, что асимптотическая сложность будет такая же, как у BFS: O(V+E).

8.5. Точки сочленения – очевидно, идет речь об этом.

Пусть, мы рассматриваем ребро (a, b). Тогда, если из вершины b и из любого ее потомка в дереве обхода в глубину нет обратного ребра в предка вершины a, то вершина a – точка сочленения, удалив ее, мы сделаем «поддерево» с корнем в b – новой компонентой связности. Тем самым проверяется, если ли другой путь из вершины a в b.

Нужно научиться быстро проверять этот факт для каждой вершины в процессе обхода графа в глубину.

Воспользуемся идеей, аналогичной используемой на практике: для каждой вершины графа заведем информационную величину, способную быстро обновляться и способную за O(1) отвечать нам на интересующий вопрос.

Для начала, будет хранить время входа в вершину a во время обхода в глубину. Вместе с ней будем хранить время t, равное минимуму среди: времени входа в эту вершину a, времени входа во все вершины, являющиеся концами обратных ребер (a, w) – [это есть время входа в самую верхнюю вершину среди концов обратных ребер] и временем t среди всех ее дочерних вершин в дереве обхода в глубину. Оно будет обновляться после выхода из вершины, поэтому времена t дочерних вершин уже будут посчитаны.

Тогда из вершины a или ее потомка есть обратное ребро в ее предка (то есть это не точка сочленения), если найдется такая дочерняя вершина, что время $t < in[a]$, где in – времена входа в алгоритме DFS.

Таким образом, если для текущего ребра (a, b) $t_b \geq in[a]$ (то есть других путей нет), то вершина a – точка сочленения.

Если же у вершины a вообще нет предков, то есть это первая вершина, с которой начался обход в глубину, то она является точкой сочленения, если у нее больше одной дочерней вершины в дереве обхода в глубину. Это достаточно очевидно: одним проходом мы не смогли обойти весь граф, значит удалив эту вершину, получим увеличение компонент связности.

8.6. Мосты – тоже очевидно, что речь идет о мостах.

Идея абсолютно такая же.

Пусть, мы рассматриваем ребро (a, b). Тогда, если из вершины b и из любого ее потомка в дереве обхода в глубину нет обратного ребра в вершину a или в предка вершины a, то текущее ребро – мост, и, удалив его, мы сделаем «поддерево» с корнем в b – новой компонентой связности. Тем самым проверяется, если ли другой путь из вершины a в b.

Информационное поле будет тем же.

Из вершины a или ее потомка есть обратное ребро в ее предка (то есть ребро не мост), если найдется такая дочерняя вершина, что время $t \leq in[a]$, где in – времена входа в алгоритме DFS.

Таким образом, если для текущего ребра (a, b) $t_b < in[a]$ (то есть других путей нет), то ребро (a, b) – мост.