1. Как модифицировать алгоритм Флойда-Уоршелла?

Чтобы алгоритм Флойда помимо длин кратчайших путей находил и сами кратчайшие пути, необходимо завести двумерный массив направленных ребер Array[from][to], хранящий в каждой ячейке номер вершины, в которую необходимо направиться для нахождения кратчайшего пути из from в to. Таким образом, на каждой итерации, если мы находим более короткий путь, мы записываем в ячейку массива номер следующей вершины.

Соответственно после работы алгоритма мы просто идем по этому массиву, пока не дойдем до финишной вершины.

Сложность: такая же, как и у алгоритма Флойда – $\Theta(N^3)$ по времени и по памяти.

Корректность: действительно, если мы будем запоминать, в какие вершины надо заходить, чтобы пойти по кратчайшему пути, то в результате получим искомый кратчайший путь (если он существует).

2. Цикл отрицательного веса?

Поскольку алгоритм Флойда ищет расстояния между любыми двумя вершинами a, b, где $a, b \in V$ – множеству вершин, то мы можем найти расстояние от a до самой себя. Если алгоритм выдаст, что $\exists$ такая вершина $a \in V$, что расстояние от a до a отрицательное, то тогда в графе обязательно найдется цикл отрицательного веса.

3. В ориентированном взвешенном графе есть ровно одно ребро $(u \to v)$ с отрицательным весом. Описать эффективный алгоритм поиска кратчайшего пути между заданной парой вершин (a, b) — вход задачи: матрица весов и вершины a и b.

Сначала проверяем в графе наличие отрицательного цикла (как это делать, было разобрано на прошлой неделе).

Далее убираем ребро с отрицательным весом и ищем длину кратчайшего пути из a в b по алгоритму Дейкстры.

Аналогично ищем длины кратчайших путей из a в u и из v в b.

Складываем длины этих двух путей и вычитаем абсолютное значение веса удаленного раннее ребра.

Сравниваем полученную сумму с длиной пути из a в b, найденного ранее.

Берем легчайший из этих двух путей.

Сложность алгоритма равна сложности алгоритма Дейсксатры, который мы применили 3 раза. То есть $O(V^2)$.

Корректность: поскольку мы знаем, что алгоритм Дейкстры работает корректно при отсутствии ребер отрицательного веса, а в нашем алгоритме мы использовали его исключительно при таких условиях, то мы победили – алгоритм корректен!

4. В Главе 2 [ДПВ] (раздел 2.5) приведён алгоритм Штрассена для умножения матриц сложностью $O(n^{\log_2 7})$.

1. Объясните почему с его помощью нельзя ускорить алгоритмы поиска транзитивного замыкания (поиска вершин, достижимых из каждой вершины) и поиска кратчайших путей, основанных на быстром возведении в степень до $O(n^{\log_2 7} \log n)$?

Потому что алгоритм Штрассена ничуть не быстрее наивного алгоритма умножения матриц (а для поиска транзитивного замыкания нужно как раз перемножать матрицы).

Алгоритм Штрассена использует метод "разделяй и влавствуй деля большую задачу на несколько подзадач. А именно он делит исходную матрицу на 4 одинаковые по размеру матрицы и считает произведения маленьких матриц.

В результате получается рекурсивный алгоритм сложностью $T(n) = 8T(\frac{n}{2}) + O(n^2)$, что по master-теореме дает нам $\theta(n^3)$ – такую же сложность, что и у наивного алгоритма умножения матриц.

Указание: булева алгебра $\{\vee, \wedge\}$ *и тропическая алгебра* $\{\min, +\}$ *не являются кольцами.*

2. Постройте алгоритм, который на основе алгоритма Штрассена находит матрицу транзитивного замыкания ($a_{ij} = 1$ тогда и только тогда, когда j достижима из i) оптимальнее, чем за $O(n^3 \log n)$ и оцените его сложность.

На самом деле мы можем ускорить алгоритм Штрассена с помощью небольшого алгебраического трюка, перемножив не 8 подматриц, а 7! (это не факториал, а восклицательный знак)

Сейчас посмотрим! После умножения матрицы X на матрицу Y получаем результат.

$$\begin{Vmatrix} P_4 + P_5 + P_6 - P_2 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{Vmatrix}$$

где

$P_1 = A(F - H), P_2 = (A + B)H, P_3 = (C + D)E, P_4 = D(G - E)$

$P_5 = (A + D)(E + H), P_6 = (B - D)(G + H), P_7 = (A - C)(E + F)$

В результате формула для вычисления времени работы немного изменилась: $T(n) = 7T(\frac{n}{2}) + O(n^2)$.

По master-теореме получаем время работы алгоритма $\theta(n^{log_2 7})$

5. Предложите $O(|V| + |E|)$ алгоритм, который находит центр дерева (вершину, максимальное расстояние от которой до всех остальных минимально). Докажите его корректность и оцените асимптотику.

Сначала обходим дерево с помощью DFS. – сложность $O(|V| + |E|)$

Удаляем все листья дерева. – сложность

Повторяем удаление листьев, пока не останется 1 или 2 вершины.

Оценка асимптотики: обход дерева стоит $O(|V| + |E|)$, удаление листьев суммарно – $O(|V)|$. В итоге получаем нужную нам асимптотику $O(|V| + |E|)$

Корректность: действительно, с каждой иттерацией по удалению листьев мы приближаемся все ближе к центру и когда-нибудь мы обязательно до него дойдем. Также стоит отметить, что дерево (по одному из определений) – это граф, в котором нет циклов, поэтому центральных вершин в дереве не может быть больше чем 2 (АЛКТГ ван лав).

6. Что такое вершинное покрытие мы знаем благодаря ДиМаСиКу ГуЩиНу. Постройте линейный алгоритм, который находит минимальное (по числу вершин) вершинное покрытие для дерева (вход задачи).

1. Берем любую вершину $u : u$ находится на расстоянии 1 от одного из листьев.
2. Добавляем вершину u в решение.
3. Удаляем все ребра, инцидентные u.
4. Решаем аналогично задачу для дерева с мЕньшим количеством вершин.

Сложность алгоритма – линейная, т.к. в худшем случае нам необходимо будет взять в решение $\frac{n}{2}$ вершин (двудольный граф), т.к. у дерева по определению $n-1$ ребро, а каждая вершина может покрыть два ребра.

Корректность алгоритма тривиальна и может быть доказана по индукции.

7[Шень 1.3.3]. Даны две последовательности `x[1]` ... `x[n]` и `y[1]` ... `y[m]` целых чисел... Сложность алгоритма $O(nm)$.

Заведем матрицу $n \times m$ и в ячейку $a_{i,j}$ будем писать максимальное из значений среди ячеек $a_{i,j-1}$ и $a_{i-1,j}$. В случае, если $a_{i,j} = a_{i-1,j-1}$, то максимальное из указанных двух выше значений сравниваем со значением $a_{i-1,j-1} + 1$ и присваиваем ячейке $a_{i,j}$ максимальное из них.

В итоге каждой из диагоналей, параллельных главной, мы сопоставляем сращивание двух последовательностей со сдвигом на некоторое количество символов. Максимальный сдвиг будет соответствовать максимальной общей части.