Домашняя работа №12

Бредихин Александр

10 мая 2020 г.

Задача 1

Задача: Как модифицировать алгоритм Флойда-Уоршелла, чтобы он находил не только длины кратчайших путей между всеми парами вершин, но и сами пути?

Параллельно построению матрицы кратчайших расстояний (которая строится алгоритмом Флойда-Уоршелла) строим матрицу предков. Для этого инициализируем её нулями а затем на каждой итерации в ячейку (i, j) записываем предпоследнюю вершину на кратчайшем пути из i в j через вершины $1 \dots k$. Кода матрица кратчайших расстояний заполнится, то есть алгоритм сделает $|V|$ шагов мы получим матрицу предков, в которой будем хранить предпоследние вершины в кратчайшем пути между двумя любыми вершинами.

Заметим, что мы можем восстановить все эти кратчайшие пути по этой матрице следующем образом: для пути из i в j начиная с конца переходим по значениям в построенной матрице. Так как весь путь кратчайший, то любой подпуть тоже будет кратчайшим (иначе можно было бы сделать его меньшим и весь путь тоже бы уменьшился). То есть зная концы пути за $O(|V|)$ в худшем случаем мы найдём кратчайший путь между этими вершинами. Всего возможных пар $O(|V|^2)$, поэтому, чтобы найти кратчайшие пути между всеми вершинами потребуется $O(|V|^3)$ (время работы алгоритма не изменилось)

Задача 2

Задача: Как используя выходные данные алгоритма Флойда-Уоршелла проверить, что в графе есть цикл отрицательного веса?

Инициализируем кратчайшие расстояния от каждой вершины до себя нулями. Запустим алгоритм Флойда-Уоршелла и на выход получим матрицу с кратчайшими расстояниями от каждой до каждой вершины. Если в полученной матрице есть значение в диагонали, которое меньше 0, то в графе есть цикл отрицательного веса. Докажем это:
Доказываем в две стороны:

1) Если в полученной после выполнения матрице нашёлся элемент в диагонале меньше 0, то алгоритм Флойда-Уоршелла нашёл минимальный путь из этой вершины в себя же и его длина меньше 0, что по определению и есть цикл отрицательного веса

2) Если в графе есть цикл отрицательного веса, то можно пройти по этому циклу из какой-то вершины в саму себя и при этом будет отрицательный путь, поэтому алгоритм Флойда-Уоршелла выведет отрицательное число в диагонали для этой вершины (так как ищет минимальный путь)

Сложность: алгоритм Флойда-Уоршелла - $O(|V|^3)$ затем проверяем все эдементы диагонали и ищем отрицательное значение - $O(|V|)$. Суммарно: $O(|V|^3)$

Задача 3

Задача: 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.

Будем считать, что в графе нет циклов отрицательного веса (иначе кратчайшее расстояние не определено, так как мы можем ходить по этому циклу бесконечно много а затем прийти в нужную вершину, то есть для любого расстояния можно сделать меньше пройдясь по циклу ещё раз). В этом случае нужно запускать алгоритм Беллмана-Форда для определения этого цикла.

1 способ: Можем сразу найти с помощью алгоритма Беллмана-Форда, так как он корректно работает и для графов с отрицательными весами.

2 способ: Удалим ребро $(u \to v)$, возможны 2 случая: кратчайший путь проходит через это ребро, не проходит.

Запустим алгоритм Дейкстры 3 раза: найдём кратчайшее расстояние между (a, b) (без ребра $(u \to v)$) обозначим его за l_1.

Также найдём с помощью алгоритма Дейкстры кратчайшие расстояния от a до $u - l_2$ и от v до $b - l'_2$. Тогда кратчайшим расстоянием между a и b будет значение:
$min(l_1, l_2 + w(u, v) + l'_2)$

В графе после удаления ребра (u, v) нет рёбер с отрицательным весом, следовательно, алгоритм Дейкстры работает корректно. Рассматриваем всевозможные случаи и находим меньший из них: кратчайший путь проходящий через удалённое ребро и кратчайчий путь не проходящий через него. Алгоритм работает корректно.

Сложность: запускаем несколько раз алгоритм Дейкстры оптимальная реализация которого работает за $O((|V|+|E|)\cdot\log|V|)$, что для плотных графов намного быстрее, чем алгоритм Беллмана-Форда $O(|E||V|)$.

Задача 4

Задача: 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 $O(n^{\log_2 7}\log n)$?

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.

1) Используем все выражения и обозначения, как в алгоритме Штрассена из ДВП (также используем тот факт, что булевая и тропическая алгебры являются полукольцами, а не кольцами и у них нет обратного элемента по сложению)

Для булевой алгебры: возьмём выражения: $P_4 = D(G-E) \neq DG-DE$ в булевой алгебре для чисел: $D = 0$, $G = 1$, $E = 0$ (правая и левая части не будут совпадать). Следовательно, в этой алгебре левая нижняя четверть будет вычислена неверно (для этих же чисел для выражения

$P_3 = (C + D)E = CE + DE$ - всё верно). Следовательно, нельзя использовать алгоритм Штрассена.

Для тропической алгебры делаем аналогичные рассуждения и выражение $P_4 = D(G - E) \neq DG - DE$ для чисел $G = 1$, $D = 1$, $E = 5$. Левая и правая части выражений не равны в тропической алгебре, а для P_3 верно. Следовательно, левая нижняя четверть будет вычислена неверно, значит, нельзя использовать алгоритм Штрассена.

2) Используем алгоритм Штрассена для умножения матриц в обычной алгебре R (действительные числа образуют кольцо). В итоге, делая все те же действия, только перемножая матрицы быстрее получим сложность $O(n^{\log_2 7} \log n)$

Задача 5

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

Заметим, что если мы удалим все листы (висячие вершины), то центр дерева не изменится. Это следует из определения центра дерева: вершина, максимальное расстояние от которой до всех остальных минимально. То есть пусть у первоначального дерева эта вершина a, тогда у нового дерева, полученного удалением всех висячих вершин, эта вершина тоже является центром, так как максимальное расстояние от центра дерева до какой-то вершины достигается, когда она висячая (так как если это не она, то мы можем сделать ещё шаги и получить висячую). Так как мы удаляем все висячие вершины, то макимальное расстояние уменьшится на 1, но все равно останется минимальным, так как это то же дерево только без висячих вершин, расстояние от центра до которых не превосходит максимального. Получаем, что центр дерева сохранится.

Основываясь на этом факте строим следующий алгоритм: запускаем DFS и по полученному дереву поиска убираем все висячие вершины (листья дерева поиска) (и запоминаем те, которые после удаления станут листьями дерева поиска, чтобы удалить их на следующем шаге). Таким образом удаляем вершины, пока в дереве не останется 1 или 2 вершины (так как центров может быть не один, а 2, тогда будем выводить любой из них).

Из первого факта и того, что лист не может быть центром (так как можем взять не лист, тогда максимальное расстояние сократится на 1) дерева следует то, что когда останутся 1 или 2 вершины, то они будут центрами первоначального дерева (мы изменяем граф так, что вершина центр не меняется и не может удалиться на каком-то шаге). Следовательно, алгоритм сработает корректно.

Сложность: учитываем, что работаем с деревом для которого: $O(|E|) = O(|V|)$. Тогда поиск в глубину работает за $O(|E| + |V|) = O(|V|)$. Также удаляем все вершины кроме 1 (или 2х) – $O(|V|)$. Суммарно $O(|V|)$.

Задача 6

Задача: Вершинным покрытием (vertex cover) графа $G(V, E)$ называется множество его вершин $S \subseteq V$, которое содержит хотя бы один конец каждого ребра графа. Постройте линейный алгоритм, который находит минимальное (по числу вершин) вершинное покрытие для дерева (вход задачи).

Заметим, что если мы не бёрем корень дерева в вершинное покрытие, то по его определению мы должны взять всех его потомков (чтобы получилось вершинное покрытие). То есть возникает рекурсивная формула: если обозначит за $f(A)$ – минимальное вершинное покрытие дерева с корнем A если в вершинное покрытие брать этот корень, то
$f = min(f(root), f(A_1) + \ldots + f(A_n)$, где F_n - потомки корня A (минимум берём по мощности множества). Из этих рассуждений получаем следующий динамический алгоритм:
Выполняем DFS и строим дерево поиска. Начинаем подниматься от листьев к корню дерева на каждом шаге выбирая минимально возможное количество вершин, входящих в покрытие, согласно реккурентной формуле (в каждой вершине храним какое минимальное количество вершин используется, чтобы покрыть данное поддерево). В корне после выполнения алгоритма будет лежать ответ.

Алгоритм работает корректно, так как прроссматривает всё дерево на каждом шаге выбирая оптимальный вариант (работает как индукция по реккуретной формуле).

Сложность: учитываем, что в дереве $O(|V|) = O(|E|)$, поиск в глубину работат за $O(|V| + |E|) = O(|V|)$. Затем прроссматриваем всё дерево ещё раз – $O(|V|)$. В итоге: $O(|V|)$

Задача 7

Задача: Даны две последовательности $x[1] \dots x[n]$ и $y[1] \dots y[m]$ целых чисел. Постройте алгоритм, который находит максимальную длину последовательности, являющейся подпоследовательностью обеих последовательностей. Сложность алгоритма $O(nm)$.

Решаем задачу динамикой: для этого создаём массив $L[n][m]$. В ячейку $L[i][j]$ записываем максимальную длину общей подпоследовательности $x[1] \dots x[n]$ и $y[1] \dots y[m]$. Заметим, что выполняется такое рекурретное соотношение:

$$\mathrm{L}[i][j] = \begin{cases} 0, & i = 0 \text{ or } j = 0 \\ \mathrm{L}[i-1][j-1] + 1, & x[i] = y[j] \\ \max(\mathrm{L}[i][j-1], \mathrm{L}[i-1][j]), & x[i] \neq y[j] \end{cases}$$

Так как если $x[i] = y[j]$, то длина максимальной подпоследовательности увеличится на 1 (так как мы можем взять этот элемент)с предыдущего шага, то есть $L[i-1][j-1] + 1$

Если $x[i] \neq y[j]$, то этот элемент не может входить в общую подпоследовательность, поэтому на этом шаге выбираем максимум из $L[i][j-1]$ и $L[i-1][j]$.

Последовательно заполняем весь массив. В ячейке $L[n][m]$ будет лежать то, что требуется в задаче (так как на каждом шаге выбираем максимально возможное).

Сложность: заполняем весь массив, в каждой ячейке делаем $O(1)$ операций, поэтому суммарно $O(nm)$.