Основные алгоритмы 12

Ковалев Алексей

1. Заведем дополнительный двумерный массив p размера $n \times n$, в котором будем хранить, через какую вершину проходит кратчайший путь. Изначально этот массив заполним так: $p[i][j]$ равно i, если есть ребро (i, j), иначе равно -1. Во время работы алгоритм Флойда на каждой итерации выбирает, не менять путь или пройто через некоторую дополнительную вершину k. Именно это k и будем хранить в $p[i][j]$ при поиске пути из i в j. Тогда после завершения работы алгоритмы Флойда пути можно восстановить рекурсивно следующим способом: путь между i и j – это объединение путей от i до $p[i][j]$ и от $p[i][j]$ до j. Рекурсивно продолжаем искать пути, пока $p[u][v]$ не окажется равно u или v.

2. Определить наличие циклов отрицательного веса в графе можно так: для всех пар вершин u, v, $u \neq v$ найдем сумму расстояний от u до v и от v до u. Если хотя бы для одной пары вершин это сумма отрицательна, то в графе есть цикл отрицательного веса, причем вершины u и v лежат на нем. Такая проверка займет у нас $O\big(n^2\big)$ времени, что меньше времени работы алгоритма Флойда $O\big(n^3\big)$, то есть на общее время работы алгоритма такая проверка не повлияет.

3.

1. При поиске транзитивного замыкания нами используется булева алгебра $\{\vee, \wedge\}$, которая не является кольцом, так как 1 не имеет обратного по дизъюнкции ($1 \vee x = 0$ не имеет решений), значит оно не может быть найдено с помощью алгоритма Штрассена. Аналогично тропическая алгебра $\{\min, +\}$ не является кольцом, так как не умеет нейтрального элемента по min, поэтому не может быть использована в алгоритме Штрассена для поиска кратчайших путей между всеми парами вершин.
2. Матрицу транзитивного замыкания все же можно построить с помощью алгоритма Штрассена за $O\big(n^{\log_2 7} \log n\big)$. Для этого при умножении матриц $AB = C$ будем считать не $\sum_{k=1}^{n} a_{ik} \wedge a_{kj}$, а $\sum_{k=1}^{n} a_{ik} \cdot b_{kj}$ по модулю, большему n, после чего проверять, равна ли эта сумма 0. Если равна, то $c_{ij} = 0$, иначе $c_{ij} = 1$. Это корректно, поскольку вычисляемая сумма равна количеству истинных конъюнкций при произведении строк булевой матрицы, но теперь вычисления проводятся над элементами кольца.

4. Будем сразу строить онлайн алгоритм, решающий эту задачу. Заведем переменные $\texttt{sum} = 0, \texttt{ans}, \texttt{l} = 0, \texttt{r} = 0, \texttt{pos} = -1$. После каждого нового считывания будет делать следующее:

- $\texttt{sum} = \texttt{sum} + \texttt{x[i]}$, где `x[i]` – считанное только что число.
- если $\texttt{sum} > \texttt{ans}$, то $\texttt{ans} = \texttt{sum}$, $\texttt{l} = \texttt{pos} + 1$, $\texttt{r} = \texttt{i}$, где `i` – номер последнего считанного элемента.
- если $\texttt{sum} < 0$, то $\texttt{sum} = 0$, $\texttt{pos} = \texttt{i}$, где `i` – номер последнего считанного элемента.

Также дополнительно сразу после первого считывания присовоим $\texttt{ans} = \texttt{x[0]}$. После выполнения алгоритма в перменной `ans` будет максимальная сумма на подотрезке, а в перменных `l` и `r` – искомые границы отрезка с максимальной суммой. Алгоритм корректен, так как в `ans` всегда находится максимальная из возможных сумм на подотрезке, `sum` обнуляется, только если сумма всех элементов до данного стала отрицательной, в `pos` хранится позиция элемента, сумма всех элементов до которого отрицательна, то есть эти элементы не умеет смысла брать в итоговую сумму.

5. Сначала найдем две наибольшие возрастающий подпоследовательности: в исходном массиве и в записанном в обратном порядке исходном массиве. Это делается с помощью алгоритма поиска НВП, работающего за $O\big(n^2\big)$. Можно считать, что этот алгоритм возвращает нам массив d, в котором d_i равно длине наибольшей возрастающей подпоследовательности, заканчивающейся в x_i. Тогда два заупска этого алгоритма дадут нам два массива a и b, такие что a_i – длине наибольшей возрастающей подпоследовательности, заканчивающейся в x_i, и b_i – длина наибольшей убывающей подпоследовательности, начианющейся в x_i. Тогда одновременно идя по этим массивом и выбирая максимум из $a_i + b_i$ мы получим длину наибольшей унимодальной подпоследовательности исходного массива. Сложность такого алгоритма – $O\left(n^2\right)$, так как именно такое время займет два запуска выбранного НВП, а затем линейное время займет поиск ответа.

6. Пусть $\mathrm{LCS}(i, j)$ – длина наибольшей общей подстроки, заканчивающейся в позициях i и j данных строк. Тогда составим такую динамику:

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

Ее можно посчитать за $O(nm)$, если n и m – длины данных строк, если сохранять все вычисленные ранее значения. Это даелается так: сначала считаем $\mathrm{LCS}(0, i)$ для всех i, затем $\mathrm{LCS}(1, i)$ для всех i, и так далее до $\mathrm{LCS}(n-1, i)$. Если i или j меньше 0, то есть $\mathrm{LCS}(i, j)$ не имеет смысла, можно считать ее за 0. Длина наибольшей подстроки – наибольшее из посчитанных значений $\mathrm{LCS}(i, j)$. Алгоритма корректен по построению, а его сложность, как сказано выше, составляет $O(nm)$.

7. Пусть $\mathrm{cut}(s)$ – стоимость разрезания строки по данным индексам. Тогда

$$\mathrm{cut}(w) = \min_{0 \leqslant i \leqslant k} (\mathrm{cut}(u_1 \dots u_i) + \mathrm{cut}(u_{i+1} \dots u_k))$$

Если считать $\mathrm{cut}(u_i) = 0$ и $\mathrm{cut}(u_i u_{i+1}) = |u_i| + |u_{i+1}|$, то найти $\mathrm{cut}(w)$ можно за $O\left(k^2\right)$, если сохранять все промежуточные вычисления. Объясняется это тем, что сначала выбирается один разрез из не более чем k возможных, затем один из не более чем $k-1$ возможных и так далее. Восстановление последовательность разрезов не вызывает затруднений, если при сворачивании рекурсии запоминать, откуда пришло минимальное на данном уровне значение.