Brick in the Wall, Part 2

Идея: Георгий Корнеев
Разработка: Михаил Первеев

Научимся решать задачу при условии, что построенная стена должна быть горизонтальной. Далее необходимо запустить полученное решение для исходной матрицы, а также для транспонированной матрицы, чтобы учесть вертикальные стены, и выбрать наилучший из двух ответов.

Пусть мы можем строить только горизонтальные стены. Выделим в каждой строке множество наибольших по включению отрезков, состоящих только из свободных клеток. Также создадим для стартовой и финишной клеток отрезки длины 1. Теперь построим граф, в котором вершинами будут выделенные отрезки. Соединим две вершины ребром, если в отрезках, соответствующих этим вершинам, существуют две клетки, которые являются соседними по стороне.

Теперь научимся определять, можно ли построить одну стену таким образом, чтобы не существовало ни одного пути из стартовой клетки в финишную. Так как на этом этапе мы не стремимся минимизировать длину стены, будем строить стену, которая полностью перекроет один из выделенных ранее отрезков. В терминах построенного графа это означает, что некоторая вершина будет удалена, так как теперь соответствующий ей отрезок будет заполнен заблокированными клетками, а значит перемещаться по ним будет нельзя.

Мы свели задачу к классической задаче: «можно ли удалить одну вершину в графе таким образом, чтобы не существовало ни одного пути из вершины s в вершину t». Для решения этой задачи выделим компоненты вершинной двусвязности и построим Block-Cut Tree. Теперь легко заметить, что удалить одну вершину требуемым образом возможно тогда и только тогда, когда в дереве на пути строго между стартовой и финишной вершиной есть хотя бы одна точка сочленения.

Вернемся к исходной задаче и поймем, как минимизировать длину построенной стены. Переберем все точки сочленения, лежащие на пути между стартовой и финишной вершинами в дереве, и для каждой из них решим задачу независимо.

Рассмотрим некоторую точку сочленения v, а также две инцидентные ей компоненты вершинной двусвязности c_1 и c_2, которые лежат на рассмотренном пути в дереве. Рассмотрим некоторое ребро в графе (u, w). Скажем, что $seg(u, w)$ — это отрезок столбцов, в которых отрезки строк таблицы, соответствующие вершинам u и w, «соединяются». Иными словами, $seg(u, w) = [l, r]$, где $[l, r]$ — отрезок столбцов, принадлежащих обоим отрезкам, соответствующим вершинам u и w.

Определим отрезок $[l_{c_1}, r_{c_1}]$ следующим образом:

$$l_{c_1} = \min_{(v,u)\in c_1} seg(v, u).l$$

$$r_{c_1} = \max_{(v,u)\in c_1} seg(v, u).r$$

Аналогично определим отрезок $[l_{c_2}, r_{c_2}]$, рассмотрев ребра, принадлежащие компоненте c_2. Нетрудно заметить, что в качестве ответа достаточно построить стену, покрывающую отрезок столбцов $[l_{c_1}, r_{c_1}]$, или стену, покрывающую отрезок столбцов $[l_{c_2}, r_{c_2}]$, так как в каждом из этих случаев будут заблокированы все клетки, по которым можно из вершины v перейти в компоненту c_1 или в компоненту c_2. Однако, данные способы не всегда являются оптимальными. Поэтому далее необходимо рассмотреть случаи, когда оба отрезка не перекрываются полностью.

Пусть $L = \max(l_{c_1}, l_{c_2})$, а $R = \min(r_{c_1}, r_{c_2})$. Нетрудно заметить, что левая граница перекрываемого отрезка должна быть не правее L, а правая граница перекрываемого отрезка — не левее R. Если это не так, то всегда будет существовать клетка, через которую можно напрямую перейти из c_1 в c_2. Однако, перекрыть отрезок $[L, R]$ не всегда достаточно, так как могут существовать ребра, инцидентные вершине v, используя которые, можно «перепрыгнуть» заблокированный отрезок клеток и переместиться из c_1 в c_2.

Пусть помимо компонент c_1 и c_2 существуют компоненты $d_1, d_2, \ldots, d_k$, инцидентные вершине v.

Для каждой из этих компонент вычислим отрезок $[l_{d_i}, r_{d_i}]$ аналогично отрезкам, рассмотренным ранее. Теперь заметим, что для того, чтобы отрезок $[L, R]$ нельзя было «перескочить», используя компоненту d_i, необходимо и достаточно, чтобы левая граница перекрытого отрезка была не правее, чем l_{d_i}, или правая граница перекрытого отрезка была не левее r_{d_i}.

Таким образом, если перекрываемый отрезок обозначить как $[l_{ans}, r_{ans}]$, мы получаем следующий набор ограничений:

- $l_{ans} \leq L$;
- $r_{ans} \geq R$;
- Для любого $i \in [1, k]$ $l_{ans} \leq l_{d_i}$ или $r_{ans} \geq r_{d_i}$.

Переберем все способы выбрать левую границу l_{ans}, и для каждого варианта найдем минимальную границу r_{ans}, такую что отрезок $[l_{ans}, r_{ans}]$ будет удовлетворять всем ограничениям. Для этого будем перебирать l_{ans} от начала отрезка, соответствующего вершине v до L и поддерживать минимально возможную правую границу r_{ans}. При движении левой границы направо для некоторых d_i условие $l_{ans} \leq l_{d_i}$ перестает выполняться, поэтому для этих d_i необходимым становится условие $r_{ans} \geq r_{d_i}$, что не уменьшает минимально возможную правую границу ответа. Таким образом поддерживать правую границу r_{ans} при увеличении r_{ans} можно при помощи метода двух указателей, предварительно отсортировав отрезки $[l_{d_i}, r_{d_i}]$.

Суммарно нахождение ответа для всех точек сочленения v работает за $\mathcal{O}(nm \log m)$, так как отрезки, соответствующие вершинам, не пересекаются. Таким образом, все решение работает за $\mathcal{O}(nm \log m + nm \log n)$, то есть за $\mathcal{O}(nm \log nm)$.

Часть решения, в которой выполняется сортировка отрезков $[l_{d_i}, r_{d_i}]$, можно реализовать аккуратнее, получив время работы $\mathcal{O}(nm)$. Впрочем, для решения задачи это не требовалось.