Разбор задачи «Интернетизация»

Рассмотрим *граф*, вершинами которого являются дома, а рёбра проведены между соседними домами и имеют веса c_i. Добавим в граф ещё одну вершину и условно назовём её «Интернет». Проведём от каждого дома до «Интернета» ребро веса $min(s_i, t_i)$. Осталось заметить, что *минимальный остов* данного графа и является оптимальной схемой подключения домов к интернету. *Время работы:* $O(N \cdot \log(N))$ либо $O(N^2)$ в зависимости от выбранного алгоритма поиска минимального остовного дерева и его реализации.

Задача также допускала различные решения с применением техники *динамического программирования*. Рассмотрим некоторые из них.

Для начала сделаем важное замечание об общем виде улицы, подключённой к интернету: при **любом** способе подключения домов к интернету, дома улицы разбиваются на последовательные блоки, в каждом из которых ровно один из домов подключён к интернету (либо к спутнику, либо кабелем к провайдеру), и все дома в каждом блоке последовательно соединены между собой.

Основываясь на данном утверждении, придумаем решение, работающее за $O(N^2)$. Обозначим dp_i минимальную стоимость подключения первых i домов к интернету. Тогда при добавлении очередного i-го дома возможны две ситуации:

1. i-й дом будет подключен к предыдущему дому. В этом случае, очевидно, стоимость подключения первых i домов к интернету равна $dp_i = dp_{i-1} + c_{i-1}$.

2. i-й дом будет подключен к интернету «напрямую» (либо к спутнику, либо кабелем к провайдеру). Тогда i-й дом может образовать группу домов, подключенных к интернету, которая начинается с какого-то дома $j \leqslant i$ и заканчивается на i-м доме. Значит, $dp_i = min(s_i, t_i) + \min_{j \leqslant i} dp_{j-1} + \sum_{k=j}^{i-1} c_i$. В этом выражении dp_{j-1} отвечает за подключение всех предыдущих домов к интернету, а $\sum_{k=j}^{i-1} c_i$ за соединение домов в блоке с j-го по i-й проводами.

Таким образом, чтобы вычислить очередное значение динамики, нужно взять минимум из двух описанных выше величин. Такие вычисления займут $O(N^2)$ времени.

Оптимизируем приведённое выше решение. Для этого научимся находить $\min_{j \leqslant i} dp_{j-1} + \sum_{k=j}^{i-1} c_i$ за константное время. Пусть в момент обработки i-го дома мы знаем текущий минимум данной величины — $curMin$. После обработки этого дома мы должны обновить значение минимума. Во-первых, к нему нужно добавить c_{i-1}, т.к. блок домов, ранее являвшийся оптимальным, теперь содержит на один дом больше (а именно, в него добавится i-й дом). Также нам нужно рассмотреть новый блок, которого ранее не существовало: блок длины 1, состоящий только из i-го дома. Все описанные операции занимают константное время. Итак, актуальное значение минимума можно поддерживать «на ходу», затрачивая на его обновление $O(1)$ времени. В итоге получаем решение за $O(N)$.

Рассмотрим альтернативное решение динамическим программированием, немного отличающееся от описанного ранее. Будем обозначать all_i минимальную стоимость такого подключения первых i домов, в котором **все** i **домов** имеют доступ к интернету, а $some_i$ минимальную стоимость такого подключения домов, при котором несколько первых домов имеет доступ в интернет, а «хвост», состоящий из оставшихся домов, образует последовательный блок домов, соединённых последовательно между собой, но не подключённых к интернету. Заметим, что ответом на задачу будет значение all_{n-1}. База динамики: $all_0 = min(s_0, t_0)$, $some_0 = 0$.

Переходы: $some_i = min(some_{i-1} + c_{i-1}, all_{i-1})$, $all_i = min(all_{i-1} + c_{i-1}, some_{i-1} + c_{i-1} + min(s_i, t_i), all_{i-1} + min(s_i, t_i))$.

В последнем выражении используются следующие замечания:

- Чтобы подключить **все** i домов к интернету, необходимо рассмотреть три случая:

1. Подключаем оптимально **все первые** $i-1$ **домов** и проводим провод от $i-1$ дома до i ($all_{i-1} + c_{i-1}$).
2. Подключаем i-й дом к интернету «напрямую» ($all_{i-1} + min(s_i, t_i)$).
3. Подключаем i-й дом к ранее образованному «хвосту» и также подключаем i-й дом напрямую к интернету, тем самым предоставляя доступ в интернет и всему «хвосту» ($some_{i-1} + c_{i-1} + min(s_i, t_i)$).

- Чтобы подключить i домов так, чтобы образовался некоторый «хвост», достаточно рассмотреть два варианта:

1. Добавляем i-й дом в ранее существовавший хвост ($some_{i-1} + c_{i-1}$).
2. Подключаем **все предыдущие** $i-1$ **домов** к интернету и образуем «хвост» длины 1, состоящий только из i-го дома (all_{i-1}).

Описанное решение также имеет временную сложность $O(N)$.

Разбор задачи «Спальные мешки»

Отметим на числовой прямой все границы температур из входного файла. Тогда задачу можно переформулировать следующим образом: для каждого отрезка второго типа (места) нужно найти количество отрезков первого типа (мешки), которые полностью его покрывают.

Первое очевидное решение – просто сделаем, что от нас требуется: для каждого места переберём все мешки и посчитаем сколько мешков подходит. Время работы такого алгоритма $O(M \cdot N)$, поэтому такой алгоритм подходит не для всех значений N, M.

Более быстрое решение получается, если использовать дополнительные структурные данные для хранения информации об отрезках. Объединим все отрезки в один массив и отсортируем его по левой границе отрезков, при этом если левая граница для нескольких отрезков совпадает, то отрезки первого типа должны идти раньше.

Будем проходить по полученному массиву последовательно. Если встретился отрезок первого типа, то отметим в дополнительном массиве конец этого отрезка. Таким образом в массиве отмечены только отрезки, которые открылись раньше отрезка второго типа. Если встречается отрезок второго типа, то необходимо найти сумму от его правой границы до конца дополнительного массива. На такие запросы можно отвечать с помощью различных структур данных, например *дерево отрезков*, *дерево Фенвика* или с помощью *корневой декомпозиции*.

Также, так как в задаче стоят ограничения на координаты отрезков до 10^9, то необходимо перед ответами на запросы произвести *сжатие координат*.

Разбор задачи «Фотоохота»

Научимся определять для конкретной точки, какие памятники можно сфотографировать. Будем перебирать все памятники по очереди, и, для каждого из них, определим, можем ли его сфотографировать из данной точки или нет. Для этого, для каждой из сторон многоугольника проверим, пересекается ли отрезок этой стороны с отрезком, соединяющим наше местоположение с точкой, где находится памятник. Если наш отрезок будет пересекаться хоть с одной из сторон многоугольника, то Гена не может сфотографировать этот памятник. В данном случае, проверка *пересечения отрезков* – это стандартная геометрическая задача. Стоит отметить, что по условию задачи, касающиеся и вложенные отрезки также считаются пересекающимися. Таким образом, мы умеем находить все памятники, которые Гена может сфотографировать из конкретной точки, за время $O(nm)$.

Для того, чтобы получить ответ, достаточно посчитать количество памятников, которые Гена может увидеть в начальной и конечной точках (в случае, если Гена уходит на бесконечность, то в далёкой точке). Поймём, почему это так. Для этого зафиксируем какой-то конкретный памятник, и посмотрим, из каких точек прямой, вдоль которой идёт Гена, он может его увидеть. Несложно понять, что эта функция, в общем случае, будет иметь такой вид: если мы рассмотрим прямую с одной из сторон, то сначала этот памятник будет виден, далее, будет участок, на котором мы не будем видеть этот памятник, после чего пойдёт участок, на котором снова будет виден этот

памятник. В частном случае, эта функция может иметь более простой вид, например, у неё может не быть первого участка, но быть второй и третий. Таким образом, нам нужно посмотреть, какие памятники мы сможем увидеть на удалённых точках прямой с обеих сторон, вдоль которой шагает Гена. Однако, по условию задачи, Гена шагает в какую-то определённую сторону, таким образом, нам нужно проанализировать эту функцию на луче, вдоль которого идёт Гена. Таким образом, максимально удалённой точкой в одно сторону (в которую мы идём) является удалённая точка в этом луче, в другую – начальная точка.

Заметим, что в случае, если Гена не сможем уйти на бесконечность, а упрётся в итоге в многоугольник, то нам нужно будет найти эту точку, где Гена остановится. Причём, надо учесть, что точек пересечения с самим многоугольником может быть несколько, и нам нужно выбрать первую из них. Итоговая асимптотика решения: $O(nm)$.

Разбор задачи «Боулинг»

Рассмотрим простое решение задачи. Для каждой позиции и каждого направления будем бросать шар и, проходя по всем клеткам, которые задевает шар, подсчитывать количество кеглей, которые встретятся на пути. Ответом является максимум кеглей из всех таких бросков. Улучшим данное решение, заметив, что при переходе от одной позиции броска к другой, от текущего результата отнимается левая вертикаль (диагональ) и прибавляется правая. Таким образом, можно *предподсчитать* количество кеглей, стоящих на

- каждой вертикали;
- каждой диагонали влево под углом 45°;
- каждой диагонали вправо под углом 45°.

Теперь при переходе от одной позиции броска к другой, можно пересчитывать результат отнимая и прибавляя предподсчитанные значения за константное время.

Разбор задачи «Мировой рекорд»

Задача решается простой реализацией. Есть два различных сценария написания программы – первым начинает Вася, либо первым начинает Петя. Оба случая аналогичны, нужно лишь в обоих посчитать минимальное время и выбрать меньшее из них.

Для реализации конкретного варианта заведём две переменные, в которых будем хранить сколько секунд в данный момент осталось на отдых Пете и Васе. Назовём эти переменные r_0 и r_1. Также понадобятся переменные для хранения уже потраченного времени и сколько символов программы уже написано. В начальный момент все эти переменные имеют нулевое значение.

Будем считать, что начинает Петя. Далее в цикле будем чередовать того, кто следующим будет писать код программы. После того, как кто-то пишет программу, понятно сколько новых символов прибавляется, сколько потрачено времени на их написание и кто пишет программу следующим. Пока первый пишет, второй в этом время отдыхает – обновляем значение r_1. Чтобы перейти к следующему моменту, когда кто-то будет писать код, необходимо проверить, нужно ли второму ещё время на отдых, это эквивалентно условию $r_1 > 0$. Если да – то прибавляем к прошедшему времени время отдыха, т.е. r_1, а у обоих ребят уменьшаем значения переменных r_0 и r_1 (оба программиста отдыхают в это время! поскольку первый только что закончил писать программу, а второму ещё требуется время на отдых). После отдыха второй может писать программу, и всё происходит аналогично, только чередуются программисты и соответствующие переменные. Поэтому оформив всё в цикл, который останавливается, когда набрано более, чем N символов, получаем минимальное время для данного сценария.

Итоговое время алгоритма $O(N)$. Стоит отметить, что ответ может получиться довольно большим, поэтому для ответа нужно использовать 64-битиные переменные.