Журнал квестов

Авторы задачи: Константин Бац, Даниил Орешников, Владимир Рябчун, разработчик: Владимир Рябчун

Для решения первой подгруппы достаточно было рассмотреть несколько случаев соотношений приоритетов взятых квестов. Первые два квеста никогда не будут вычеркнуты, а для оставшихся двух достаточно просимулировать описанный в условии процесс.

В третьей и пятой подгруппах было достаточно просимулировать описанный процесс полностью для каждого нового квеста. За время $\mathcal{O}(n^2)$ (в третьей) или $\mathcal{O}(n)$ (в пятой) можно было для каждого нового квеста найти два минимальных значения приоритета в очереди, и либо добавить квест в конец, либо за еще $\mathcal{O}(n)$ времени пройтись по всей очереди целиком и выписать в новую очередь все важные квесты. Суммарное время работы такого решения (при линейном поиске минимумов) равно $\mathcal{O}(n^2)$.

Для решения оставшихся подгрупп было достаточно заметить, что если квест в какой-то момент не является неважным, то он таким и останется до момента выполнения. Действительно, если перед ним не было двух квестов с меньшим приоритетом, они и не появятся, так как новые квесты добавляются только в конец очереди.

Тогда можно отдельно хранить часть очереди, состоящую из гарантированно «важных» квестов, которая никогда не будет меняться, и следующий за ней хвост очереди, начинающийся с первого неважного квеста. Если добавленный новый квест вызывает очистку журнала, достаточно пройтись по второй части очереди, и удалить неважные квесты, а все оставшиеся перенести в первую часть. При такой реализации решение будет работать за $\mathcal{O}(n \cdot \texttt{check})$, где `check` — время поиска двух минимумов и проверки на неважность одного квеста, так как каждый квест будет обработан не более трех раз: при добавлении, при переносе в первую часть и при удалении.

Ограничения второй подгруппы гарантируют, что квесты добавляются в журнал в порядке увеличения приоритета, то есть в порядке уменьшения важности. Это гарантирует, что pm_1 и pm_2 будут равны приоритетам двух последних взятых квестов, а тогда для каждого нового квеста можно за $\mathcal{O}(1)$ проверить, вызывает ли он очищение журнала. Пользуясь рассмотренным выше фактом, получаем решение за $\mathcal{O}(n)$.

Четвертая подгруппа тоже позволяет выполнять `check` за $\mathcal{O}(1)$, например, если хранить количество квестов в журнале с каждым значением приоритета.

Для полного решения достаточно было рядом с каждой из двух частей очереди хранить кучу (`heap`) из всех приоритетов. Эта структура данных позволяет за время $\mathcal{O}(\log n)$ получать минимальный элемент, и с помощью нее можно реализовать поиск двух минимальных элементов за ту же асимптотику. В языке C++ можно было воспользоваться для этого структурой `std::set` и брать `*s.begin()` и `*(++s.begin())`.

Если такое решение не проходило по времени на последней группе тестов, можно было реализовать очередь, поддерживающую запросы минимумов за $\mathcal{O}(1)$ на двух «стеках с минимумом». Время работы полного решения, таким образом — $\mathcal{O}(n \log n)$ или $\mathcal{O}(n)$.