Задача A. Бинарная строка

Заметим, что, если $s_i = s_{i+1}$ ($s_i \neq s_{i+1}$), то это равенство (неравенство) может измениться только при применении операции к префиксу, у которого последний символ это s_i. Значит, необходимо применить операцию для каждого i, для которого $s_i \neq s_{i+1}$. После этого строка будет состоять из одинаковых символов. Если на конце «1», то необходимо применить ещё 1 операцию, чтобы инвертировать всю строку.

Задача B. Загадочный массив

При применении любой операции мы получаем возможность добавить единицу к любому элементу с большим индексом – после выполнения каждой операции будем добавлять 1 единицу, которую мы можем прибавить далее в некоторый счётчик. Рассмотрим элементы с начала до конца и будем хранить число единиц, которое мы можем добавить к любому элементу далее. Если очередной элемент больше 1, то будем вычитать из него 2, пока можем (и для последующих операций запомним, сколько раз мы сможем добавить единицу за счёт этого). Это является оптимальным, так как далее мы не сможем использовать этот элемент никак. Если он после этого равен 1 и мы можем добавить к нему 1, добавим и выполним ещё операцию – это также выгодно, так как мы получаем то же 1 прибавление назад и выполняем дополнительную операцию. Теперь рассмотрим случай, когда элемент равен 0, но мы можем добавить ещё 2 единицы, чтобы выполнить операцию. Оказывается, что в этом случае добавление 2 единиц является оптимальным.

Рассмотрим все единицы, которые мы будем куда-либо добавлять. Они либо не принесут никаких операций, либо принесут 1 операцию, комбинируясь с единичкой, первоначально находившейся в некотором месте, либо принесут 1 операцию, комбинируясь с другой прибавленной единичкой. Однако второй из этих случаев возможен лишь тогда, когда мы, рассматривая новую позицию в нашем алгоритме, рассматриваем нечётное число и при этом у нас есть хотя бы 1 единица, которую можно применить. Все остальные добавления единиц должны комбинироваться между собой – при этом мы тратим 2 операции прибавления и получаем одну, а значит, вне зависимости от того, когда мы будем тратить единицы таким образом, мы не будем никак влиять на возможность потратить единицы способом 2. Тогда выгоднее тратить их как можно раньше, так как полученные прибавления при этом можно будет применить на больший участок массива.

Таким образом, алгоритм, описанный выше, является корректным.

Задача C. Игра

Для решения введём функцию от n, которая равна 1, если для этого n делающий ход игрок выигрывает, иначе 0. Тогда если мы для некоторого n можем перейти в проигрышное состояние, значение функции 1, иначе – 0 (так как для любого хода в этом случае у соперника есть выигрышная стратегия).

Количество делителей n не превышает $\mathcal{O}(\sqrt{n})$, а число переходов не превышает числа различных простых делителей числа (можно грубо оценить как $\mathcal{O}(\log n)$), умноженного на максимальную степень простого числа (также не превышает $\mathcal{O}(\log n)$). Для каждого n переходы рассмотрим только 1 раз, а затем запомним значение функции для данного n. Тогда получим решение с асимптотикой $\mathcal{O}(\sqrt{n}\log^2 n)$, что является достаточным.

Задача D. Сокровище

Пусть кладоискатели остановятся в городе i, тогда наибольшая прибыль, которую они получат, это сумма k максимумов префикса массива a до i включительно минус сумма массива c до i не включительно. Ответом будет наибольший ответ среди этих случаев. Тогда для решения достаточно поддерживать k максимумов на префиксе, что можно сделать, используя структуры, поддерживающие поиск минимума, добавление и удаление (`std::multiset` в C++). Такое решение будет работать за $\mathcal{O}(n \log n)$.

Задача E. Лабиринт времени

Если бы в задаче не было k дополнительных добавлений времени, то задачу можно было бы решить обычным алгоритмом Дейкстры. Для того, чтобы учитывать дополнительные k условий, будем

использовать пары $(u, mask)$ для алгоритма Дейкстры. Такая пара будет означать минимальное расстояние до вершины u, если при этом мы посетили хотя бы 1 ребро из условия i, $0 \leqslant i \leqslant k-1$ для всех i таких, что бит под номером i в маске равен «1». Тогда при переходе из такой пары нужно, как и в графе, перебрать исходящие из u рёбра и соответствующие конечные вершины v. Если эти рёбра не участвуют в условиях, то переходим в $(v, mask)$. Если ребро участвует в условии, то нужно либо добавить «1» в соответствующий бит маски, чтобы далее при возможном посещении второго ребра учесть дополнительное условие, либо, если там была единица, добавить к потраченному времени время из соответствующего условия.

В таком графе мы будем рассматривать $\mathcal{O}(m2^k)$ переходов, а значит асимптотика решения будет $\mathcal{O}(m2^k(\log m + k))$.