Задача A. Экспедиция через пустыню

Для решения задачи определим, сколько килограммов в рюкзаке каждого исследователя остается после укладки необходимой провизии. Поскольку на каждого человека требуется p килограммов воды и провизии, свободное место в одном рюкзаке составит $w - p$ килограммов. Условие задачи гарантирует, что снаряжение все-таки можно уложить, поэтому $w - p \geqslant 0$.

Далее, чтобы найти общий вес дополнительного снаряжения, которое смогут взять участники экспедиции суммарно. Умножим свободное место в одном рюкзаке на число исследователей: $n \cdot (w - p)$. Это и будет ответом на задачу.

Задача B. Палаточный лагерь

Заметим, что площадь треугольника со стороной x равна x^2 единичных треугольников. Таким образом, зная сторону треугольного участка палаточного лагеря, можно вычислить его площадь в единичных треугольниках.

Рассмотрим два случая.

1. $a+b \leqslant n$. Тогда канаты отсекают от треугольной поляны два треугольника площадями $S_1 = a^2$ и $S_2 = b^2$. А оставшаяся часть поля будет иметь площадь $S_3 = n^2 - S_1 - S_2$. Таким образом, ответ на задачу — $\max(S_1, S_2, S_3)$.

2. $a + b > n$. В таком случае канаты будут пересекаться между собой, отсекая треугольник со стороной $a + b - n$ и площадью $S_0 = (a + b - n)^2$, и два четырехугольника с площадями $S_1 = a^2 - S_0$ и $S_2 = b^2 - S_0$. Оставшаяся часть поляны будет иметь площадь $n^2 - S_0 - S_1 - S_2$. Таким образом, ответом на задачу будет $\max(S_0, S_1, S_2, n^2 - S_0 - S_1 - S_2)$.

Задача C. Караван

Найдем полную стоимость перехода пустыни с помощью каравана номер i. Она составляет $p_i + c_i + R \cdot v_i$ бэдэ.

Чтобы получить ответ на задачу, рассмотрим все караваны и выберем минимальное значение этой величины по всем i.

Время работы такого решения — $O(n)$.

Задача D. Гонки на катках

Задача может быть сведена к нахождению длины кратчайшего пути в невзвешенном неориентированном графе. Такой тип задач решается с помощью стандартного поиска в ширину (BFS). Однако создание и хранение этого графа в явном виде для данной задачи может представлять сложности, так в нем будет до 200^2 вершин и до $4 \cdot 200^2$ ребер (на пустой площадке). В решении ниже мы будем использовать этот граф неявным образом.

Будем заполнять пустые клетки площадки целыми неотрицательными числами. Если в некоторой клетке стоит число k, это будет обозначать, что до этой клетки каток может дойти минимум за k ходов (прямолинейных отрезков пути). В процессе работы алгоритма эти величины будут определяться, поэтому заполним изначально все пустые клетки площадки некоторым очень большим числом (назовем его INF $\geqslant n \cdot m$), превосходящим возможное число ходов. Это число будет выполнять роль своеобразной «бесконечности».

Организуем структуру данных «очередь», в которой будем хранить пары чисел — координаты полей на площадке. Запишем в клетку, на которой изначально стоит каток, число 0, а координаты этой клетки положим в очередь. Алгоритм решения задачи заключается в следующем. Если очередь пуста, то алгоритм прекращает работу. Иначе, вынем координаты очередной клетки из очереди. Пусть значение, записанное в этой клетке, равно k. Будем двигаться в направлении возможного движения катка (вверх, вниз, влево и вправо), пока значения, записанные в посещаемых клетках, равны INF. По мере движения заполняем посещенные клетки значением $k+1$ и кладем их координаты в очередь.

Рассмотрим числа, записанные в клетках, координаты которых в данный момент хранятся в очереди. Заметим, что все они отличаются друг от друга не более чем на единицу и не убывают в

соответствии с порядком вынимания клеток из очереди. Это утверждение легко доказывается по индукции. Заметим, что при таком движении мы можем остановиться в трех случаях:

- Мы вышли за край площадки.
- Мы перешли на занятую клетку.
- Мы перешли на ранее посещенную клетку, в которой записано число k или $k-1$.

Действительно, если мы переходим на клетку, в которой записано большее, чем $k+1$, число (но не бесконечность), то мы вытащили из очереди клетку, в которой записано число, большее чем k, раньше, чем текущую клетку, что противоречит свойству, отмеченному ранее. Если же это число меньше k, то клетка, из которой мы движемся на текущем ходу, была бы уже ранее заполнена.

Заметим, что с помощью данного алгоритма пустые клетки, достижимые из начальной, заполняются корректно, то есть, число указывает минимальное расстояние в прямых отрезках пути от начальной точки. Недостижимые же из начальной клетки остаются заполненными значением INF. С учетом сказанного, ответ равен числу, хранящемуся в конечной клетке, если оно не равно INF, и -1 в противном случае.

Время работы можно оценить как $O(m \cdot n)$. Действительно, каждая свободная клетка попадает в очередь не более одного раза, и любые другие действия с клетками также вовлекают каждую из них не более чем константное число раз.

Задача E. Артефакты

Пусть a_{min} — это минимально возможное, а a_{max} — максимально возможное фактическое количество артефактов в коллекции. Заметим, что если $a = 0$, то оно минимально возможное, а если число $a = 2^n - 1$, то оно максимально возможное. Соответственно, для первого случая $a_{min} = a$, а для второго — $a_{max} = a$.

Если a не минимальное, то чтобы найти такое a_{min}, которое получается из a изменением одного бита, необходимо заменить самую старшую единичку в битовом представлении на ноль, поскольку это гарантирует максимальную разницу между a и a_{min}. Аналогично, чтобы получить максимальную разницу между a_{max} и a, необходимо заменить самый старший ноль битового представления a на единицу. При этом необходимо учитывать, что битовое представление числа a фиксировано и имеет длину n.

Таким образом, за линейный проход по n битам числа a можно получить значения a_{min} и a_{max}. Время работы такого решения — $O(n)$.

Из-за того, что длина битового представления числа a ограничена константой ($n \leqslant 63$), допускалось и другое решение. Можно перебрать все биты i ($0 \leqslant i < n$) и заменить в числе a бит с номером i на противоположный. Для этого удобно использовать операцию битового сдвига и побитовое исключающее или (xor). Например, в языке C++ необходимое число x_i равно $a \wedge (1 << i)$. Далее задача сводится к тому, чтобы выбрать максимум и минимум среди чисел $\{n, x_0, x_1, \ldots, x_{n-1}\}$.

Задача F. Список вещей

Для решения задачи будем обрабатывать каждую операцию последовательно, модифицируя строку на каждом шаге. Поскольку длина строки и количество операций невелики (до 500), можно использовать прямой подход без оптимизаций.

При операции разворота (тип 1) нужно инвертировать символы на отрезке от l до r включительно. Для этого достаточно пройтись по индексам символов в отрезке от границы до середины и переставить символы: первый символ отрезка меняется местами с последним, второй — с предпоследним и так далее до середины отрезка.

При операции сортировки (тип 2) нужно отсортировать символы на отрезке от l до r в алфавитном порядке. Для этого необходимо выделить подстроку в отдельный список, отсортировать его, а затем записать элементы полученного массива в исходную подстроку. Для сортировки можно использовать встроенную сортировку в язык программирования или самостоятельно реализовать сортировку слиянием или цифровую сортировку.

После выполнения всех операций остается вывести символы полученной строки. Временная сложность решения составляет $O(m \cdot n \log n)$, где m — количество операций, n — длина строки, так как каждая операция требует $O(n \log n)$ действий (при использовании сортировки слиянием).

Задача G. Лагерь на деревьях

Заметим, что описанная в условии конструкция представляет из себя граф, а именно — дерево. А две искомые вершины это его листы.

С помощью поиска в глубину, найдем для каждой вершины наименее глубокий лист среди её потомков. Для этого нужно запустить поиск в глубину из вершины дерева, которая имеет больше одного соседа. В дальнейшем мы будем считать эту вершину корнем дерева.

Рассмотрим пути между двумя листами. Заметим, что поскольку мы рассматриваем дерево, то такой путь единственен и сначала идёт вверх до какого-то предка (наименьшего общего предка двух листов), а потом идёт вниз, до второго листа.

Таким образом можно найти для каждой вершины два наименее удалённых друг от друга листа, путь между которыми «перегибается» в ней. Для этого модифицируем описанный выше поиск в глубину. После посещения всех детей вершины, возьмём наименее глубокий лист каждому ребёнку. Если у вершины хотя бы два ребёнка, то ответ это сумма расстояний до найденных листов.

Ответом же на задачу будет минимум по всем вершинам из найденных наименьших расстояний. Итоговая сложность O(n).

Задача H. Магия делителей

В задаче требовалось построить последовательность магических потомков числа x, где каждый следующий элемент является суммой цифр всех делителей предыдущего элемента. Последовательность строится до тех пор, пока не встретится элемент, равный сумме цифр своих делителей (то есть $s(a_i) = a_i$), что означает, что он является неподвижной точкой), или пока не будет достигнут тысячный элемент.

Для решения задачи необходимо реализовать функцию $s(x)$, которая вычисляет сумму цифр всех делителей числа x. Для этого сначала нужно найти все делители числа x. Для нахождения делителей необходимо применить стандартный алгоритм с перебором всех чисел от 1 до $\sqrt{x}$. Для каждого из рассматриваемых чисел i нужно проверить, делится ли x на i, и если это так, то сохранить i и x/i (если они различаются). Сумма цифр сохранённых чисел — результат функции s(x).

Далее, для заданного x в явном виде строится последовательность a_i, где каждый следующий элемент вычисляется как s от предыдущего. Процесс продолжается до тех пор, пока не выполнится условие $a_i = s(a_i)$ или не будет достигнут 1000-й элемент. Если в процессе построения последовательности встретится неподвижная точка, выводится минимальное i, при котором это произошло, а также сама последовательность. В противном случае выводится -1.

Задача I. Игра с фишками

Для каждой фишки шамана оценим, через сколько ходов она сможет добраться до точки $(0, 0)$, если будет двигаться по оптимальному маршруту. За каждый ход фишка шамана может менять обе свои координаты на 1. Таким образом, фишке шамана в точке с координатами x_i и y_i понадобится $|x_i|$ ходов, чтобы оказаться на прямой $x = 0$, и $|y_i|$ ходов, чтобы оказаться на прямой $y = 0$. Соответственно в точке $(0, 0)$, в худшем для Синда случае, она сможет оказаться через $\max(|x_i|, |y_i|)$ ходов. Примем эту величину за расстояние фишки шамана до фишки Синда.

Заметим, что выгоднее всего для Синда съедать фишки шамана, расстояние до которых меньше. Отсортируем фишки шамана по увеличению расстояния.

В каком случае Синд не успеет съесть i-ю фишку шамана? В том случае, если расстояние до неё будет меньше, чем число фишек с расстоянием не больше чем у неё. Таким образом пройдемся по фишкам шамана в порядке увеличения расстояния и проверим, что $\max(|x_i|, |y_i|) \geqslant i$ (нумерация фишек с 1). Если для всех фишек это выполнено, то можно вывести их в порядке увеличения расстояний, иначе решений не существует.

Задача J. Следы на песке

Задача решается с помощью динамического программирования. Будем хранить число способов собрать первые символы слова «TENT» на каждом шаге.

Заведём массив $dp[5]$, где $dp[i]$ — число способов собрать первые i символов шаблона («T» $\to i = 1$, «TE» $\to i = 2$, «TEN» $\to i = 3$, «TENT» $\to i = 4$). Изначально $dp[0] = 1$ (пустая подпоследовательность), остальные $dp[i] = 0$.

Проходя по строке, для каждого символа c обновляем dp:

- Если c = T, то $dp[4] = dp[4] + dp[3]$ (увеличиваем количество полных «TENT»), а также $dp[1] += dp[0]$ (начинаем новую подпоследовательность с T).
- Если c = E, то $dp[2] = dp[2] + dp[1]$ (продолжаем подпоследовательность «TE»).
- Если c = N, то $dp[3] = dp[3] + dp[2]$ (продолжаем подпоследовательность «TEN»).

После обработки всей строки ответом будет $dp[4]$. Такой подход работает за $O(n)$. Важно учесть, что промежуточные значения могут быть большими, поэтому используем 64-битные целые числа.

Задача K. Карта течений

Решение сводится к проверке, можно ли посетить все отмеченные клетки, если направление движения корабля фиксировано.

Ключевая идея в том, что для каждой клетки, которую нужно посетить, должно выполняться одно из условий: либо её тип течения («H» или «V») совпадает с выбранным направлением, либо это клетка с типом «A» (где допустимо любое направление).

Проходим по всем клеткам сетки, и если встречаем «X» (целевую клетку), проверяем, допустимо ли движение в выбранном направлении. Если хотя бы одна такая клетка не удовлетворяет условию (например, выбрано направление «H», а клетка имеет тип «V» и не «A»), ответ — «NO». Если все клетки проходят проверку, ответ — «YES».

Время работы такого алгоритма — $O(m \cdot n)$.

Задача L. Дорога на Эверест

Заметим, что конечные десятичные дроби получаются, когда знаменатель после сокращения имеет только простые множители 2 и 5. Значит, нужно проверить все возможные комбинации из трёх чисел, чтобы дробь была правильной (числитель был меньше знаменателя), несократимой, и знаменатель после сокращения состоял только из 2 и 5.

Шаги решения задачи:

1. Перебрать все возможные перестановки чисел a, b, c, где первое — целая часть, второе — числитель, третье — знаменатель.
2. Для каждой перестановки проверить, что:
   - Дробь правильная. Это проверяется простым условием числитель $<$ знаменатель.
   - Дробь несократима. Для этого найдем НОД числителя и знаменателя с помощью алгоритма Евклида. НОД должен быть равен 1.
   - Знаменатель в своем представлении имеет только множители 2 и 5. Для этого можно последовательно делить на 2 и 5 до тех пор, пока это возможно, а затем проверить, что после всех делений осталось число 1.
3. Если такая комбинация найдена, вывести её.

Поскольку условие гарантирует существование ответа, то после проверки всех перестановок мы точно найдем ответ.