Задача A: Повтор

Решение 1: прочитаем строку, выведем первые два символа.

Решение 2: прочитаем число, поделим его на 101: ведь $\overline{uvuv} = \overline{uv} \cdot 100 + \overline{uv}$.

Задача B: Гадание

Решение 1: функциями, доступными в нашем языке программирования, найдём позицию первого слова «`yes`», найти позицию первого слова «`no`», а дальше сравним их. Во многих языках такая функция возвращает -1, если слово не найдено.

Решение 2: для каждой позиции в строке проверим, начинается ли в ней каждое из слов.

Задача C: Уголки

Решение 1: рекурсивное. Существует множество вариаций. Например, начнём с квадрата 8×8 — всей доски.

- Если мы стоим в квадрате $2d \times 2d$, посмотрим на четыре составляющих его квадрата $d \times d$. В том квадрате, который содержит пустую клетку, запустим решение рекурсивно. Остальные три квадрата целиком заполним соответствующим числом.
- Если мы дошли до квадрата 1×1, просто поставим туда 0.

Решение 2: числовое. Рассмотрим двоичную запись координат в нумерации с нуля: например, $(1, 4)$ превратится в $(001_2, 100_2)$. Чтобы узнать, какое число поставить в клетку (r, c), сравним её координаты с координатами пустой клетки (r_0, c_0).

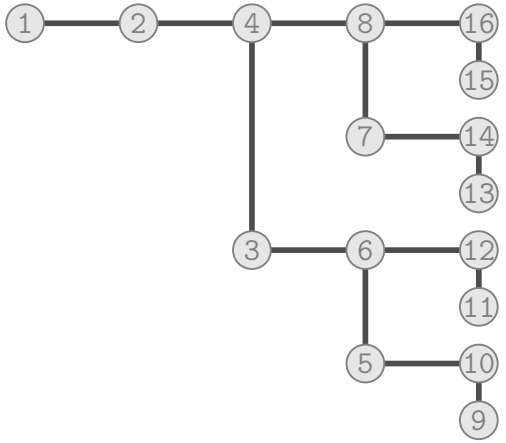
- Если они совпадают, напишем 0.
- Если они отличаются только в последнем разряде, напишем 1 (в нашем примере r равно 001_2 или 000_2, а c равно 100_2 или 101_2).
- Если они отличаются в предпоследнем разряде, но не в старшем, напишем 2 (в нашем примере $000_2 \leqslant r \leqslant 011_2$ и $100_2 \leqslant c \leqslant 111_2$).
- Если какая-то координата отличается в старшем разряде, напишем 3.

Почему это работает? Например, квадраты 4×4 устроены так: каждая координата лежит в пределах либо от 000_2 до 011_2, либо от 100_2 до 111_2. Мы видим, что внутри квадрата меняются все двоичные разряды числа, кроме старшего. Аналогичные утверждения можно сформулировать для квадратов 2×2.

Решение 3: разбор случаев. Если совсем ничего не придумывается, можно разбирать случаи вручную и получать за них баллы...

Задача D: Капитал

Попробуем совершить все возможные действия в первые несколько дней. Легко заметить, что оптимальный путь к каждому числу строится однозначно.



Решение 1: жадность с конца. Если $n = 1$, ответ равен 0. Иначе, если число нечётное — очевидно, в предыдущий день монет было на одну больше.

Если же число чётное — в предыдущий день их было в два раза меньше. Действительно, пусть в какой-то день у Буратино $2k$ монет. Если накануне он был в харчевне, то вчера утром у него была $2k + 1$ монета — нечётное число. Значит, в харчевне Буратино провёл как минимум два дня подряд. Пусть перед этим он удвоил монеты (иначе рассмотрим предыдущие два дня, потом идущие перед ними, и так далее, пока не дойдём до удвоения). Тогда в последние три дня количество монет менялось так:

$$k + 1 \longrightarrow 2k + 2 \longrightarrow 2k + 1 \longrightarrow 2k$$

Но того же результата можно достичь на один день быстрее:

$$k + 1 \longrightarrow k \longrightarrow 2k$$

Значит, при поиске решения имеет смысл рассматривать только такие, в которых чётное число получалось умножением на два. Получается, что количество монет в предыдущий день мы восстанавливаем однозначно.

Решение 2: числовое. Рассмотрим двоичную запись числа $n-1$: например, если $n = 23$, то $n - 1 = 22 = 10110_2$. Если мы будем восстанавливать ответ с конца, как в предыдущем решении, то в двоичной записи числа $n-1$ ноль в конце заменяется на единицу (харчевня), а единица в конце стирается (Поле Чудес). Значит, количество дней равно количеству цифр плюс количеству нулей в двоичной записи $n - 1$.

Задача E: Путь

Нужно аккуратно реализовать то, что требуется в условии:

- Для каждой пары точек вычислим расстояние между ними.
- Заведём список из пар точек и отсортируем его по расстоянию.
- Заведём пустой список для построенных отрезков.
- Будем рассматривать пары из списка в порядке сортировки.
- Для очередной пары пересечём отрезок между точками этой пары со всеми уже построенными отрезками. Если пересечений нет, добавим новый отрезок в список построенных.
- Теперь заведём граф — проще всего будет хранить его в виде матрицы расстояний `d`.
- Заведём также матрицу `r` с ответами: следующая вершина на каждом пути.
- Добавим в граф и в ответы рёбра — построенные отрезки.
- Отметим, что расстояние от каждой вершины до самой себя равно нулю, и в матрице ответов тоже стоит ноль.
- Найдём все пары кратчайших путей — хорошо подойдёт алгоритм Флойда.
- В момент, когда расстояние в алгоритме меняется (`d[i][j] = d[i][k] + d[k][j]`), будем менять и следующую вершину (`r[i][j] = r[i][k]`).

Чтобы написание решения увенчалось успехом, помогает разбить его на этапы. После каждого этапа помогает проверять, что написанное работает так, как задумывалось, прежде чем переходить к следующему этапу.

Как узнать, пересекаются ли отрезки AB и CD? Если какие-то из концов совпадают, считаем, что пересечения нет. Иначе проверим, что прямая AB пересекает отрезок CD, а прямая CD пересекает отрезок AB.

Как проверить пересечение прямой AB и отрезка CD? Проверим, что точки C и D лежат по разные стороны от прямой. Для этого посчитаем два псевдоскалярных произведения: $\overline{AB} \wedge \overline{AC}$ и $\overline{AB} \wedge \overline{AD}$. У точек по одну сторону от прямой знак этой величины одинаковый, у точек по разные стороны от прямой — разный.

Косое произведение векторов (x_1, y_1) и (x_2, y_2) равно $x_1 \cdot y_2 - x_2 \cdot y_1$.

Задача F: Буквы

Если просто построить строку до нужного символа и вывести подстроку, можно получить около половины баллов за задачу. Но как написать полное решение?

Заметим, что при построении бесконечной строки каждая следующая строка получается из предыдущих. Можно переписать правила построения так:

- $s_1 =$ `a`
- $s_2 =$ `aaa...aab`
- $s_k = s_{k-1}s_{k-1}\ldots s_{k-1}s_{k-2}$ для $k > 2$

В равенстве для s_k мы записываем n копий s_{k-1}, а после них — одну копию s_{k-2}. Чтобы доказать это равенство, можно посмотреть на s_2 и подумать, во что превратится каждая буква в ней через 0, 1, …, $k-2$ шагов построения.

Например, для $n = 2$:

- $s_1 =$ `a`
- $s_2 =$ `aab`
- $s_3 =$ `aab`|`aab`|`a`
- $s_4 =$ `aabaaba`|`aabaaba`|`aab`
- $s_5 =$ `aabaabaaabaabaaab`|`aabaabaaabaabaaab`|`aabaaba`

Итак, мы видим, что буквы в строке s_k повторяются с периодом, равным длине s_{k-1}. Другими словами, в строке s_k любая буква с номером i (считая с нуля) равна букве с номером $i \bmod |s_{k-1}|$. А эта позиция уже есть в строке s_{k-1}, что позволяет нам перейти к подзадаче.

Из этого наблюдения следует короткое решение нашей задачи — как найти i-ю букву бесконечной строки:

- Вычтем из i единицу, чтобы перейти к нумерации с нуля.
- Вычислим длины строк $s_1, s_2, \ldots$ — до первой длины s_r, которая больше нашего i.
- Длины таковы: $f_1 = 1$, $f_2 = n + 1$, $f_k = n \cdot f_{k-1} + f_{k-2}$.
- Теперь выполним присваивания:
- $i \leftarrow i \bmod f_r$,
- $i \leftarrow i \bmod f_{r-1}$,
- …,
- $i \leftarrow i \bmod f_2$.
- Мы оказались в подзадаче со строкой s_2. В ней ответ — буква «`b`», если $i = n$, и буква «`a`» в остальных случаях.

Автор задач:

Иван Казменко (gassa@mail.ru)