Региональная студенческая (межвузовская) олимпиада по программированию
20 апреля 2019 г.

Разбор задач

Автор презентации: Игорь Андрианов

Вологодский государственный университет

Вологда
2019

Задача A - «Числа на окружности»

Ответ можно построить так:

1). Сначала пишем все нечётные числа по возрастанию
2). Затем все чётные по убыванию

Работает и для чётных, и для нечётных *n*. Примеры:

n = 6: 1 3 5 6 4 2
n = 7: 1 3 5 7 6 4 2

Задача B - «Проблема 2019»

Чисел всего 5, поэтому можно решить «в лоб» - полным перебором:

1. Перебираем 5!=120 перестановок. В языке C++ это можно сделать стандартными средствами:
   - вначале сортируем массив функцией std::sort
   - затем перебираем перестановки функцией std::next_permutation

2. Для каждой перестановки перебираем 3^4=81 способ расстановки знаков операций.

3. Для получившегося выражения считаем ответ. Так как скобок нет, его можно легко вычислить за два прохода: вначале считаем произведения, затем – суммы и разности.

Задача C - «Чёт-нечет»

Заметим, что искомые числа имеют вид:
1 3 5 7 19 39 59 79 199 399 599 799 …
(первая цифра 1, 3, 5 или 7, далее идут девятки).

Теперь несложно их сосчитать: перебираем первую цифру (1, 3, 5, 7) и пробуем приписывать справа девятки, пока не превысим n.

Задача D - «Карточки»

Изобразим схематично ответ для каких-то *a*, *b*, *n*. Точки - числа, не входящие в сумму, звёздочки – входящие:

..*...**...***....

Основная идея: пока существуют две звёздочки, такие что у одной есть точка слева, а у другой – точка справа, первую звёздочку сдвинем на шаг влево, а вторую – на шаг вправо. Сумма от этого не изменится. В итоге имеем:

***..........*..**

То есть, решение можно искать в виде: сколько-то чисел подряд с начала интервала, сколько-то с конца, и, возможно, одно из середины. Используя метод двух указателей, несложно реализовать за *O(b-a)*.

Задача E - «Порталы»

Решение «в лоб», не проходящее по времени: перебираем пары вершин с порталами, для каждой пары перебираем остальные планеты и смотрим, куда дешевле.

Ускорим это решение, используя битовый параллелизм. Для каждой станции *v* построим два битовых множества *a1[v]* и *a3[v]*, где:

- *a1[v][i] =1*, если есть ребро от *i* к *v* стоимостью 1
- *a3[v][i] =1*, если есть ребро от *i* к *v* стоимостью 3

Задача E - продолжение

Пусть мы пробуем поставить порталы на планетах *u* и *v*.

- *a1[u]* ∪ *a1[v]* даст планеты, которые можно связать с одним из порталов ребром стоимостью 1. Пусть C_1 – размер этого множества.
- *a3[u]* ⋂ *a3[v]* даст планеты, у которых рёбра к обоим порталам имеют стоимость 3. Пусть C_2 – размер данного множества.
- для остальных планет стоимость будет равна 2, а их количество равно $n - 2 - C_1 - C_2$.

*Вычислительная сложность: о*бъединение, пересечение и вычисление размера битового множества требует около n/32 или n/64 операции в зависимости от разрядности.

В C++ удобно использовать готовый класс std::bitset.

Задача F - «Пробелы»

При *n* до 100 потребуется не более 5 операций замены.
Возможные последовательности замен, которые работает при наличии всех групп от 1 до 100 (а значит, и для любого их подмножества): 28 7 3 2 2, 21 6 3 2 2 и другие.
Такую последовательность несложно подобрать заранее.

Осталось проверить, нельзя ли обойтись четырьмя или менее заменами – это можно сделать перебором.

Оценка сложности. Переберём три первых числа и найдём итоговые размеры групп. Последняя (четвертая) замена возможна лишь, если остались группы единственного размера (кроме 1).
Итого порядка $100^4=10^8$ действий.

Задача G - «Генеалогическое древо»

Вначале за $O(n^2)$ выполним предобработку входного графа. Затем на каждый запрос ответим за $O(1)$.

Предобработка:

Этап 1. Для каждой вершины найдём и запомним длину кратчайшего пути до каждого её потомка. Можно сделать поиском в глубину: для каждой вершины вначале получаем ответы для её детей, а из них получаем и для текущей вершины (как наименьшие ответы из детей + 1).

Так как рёбер не больше $2 \cdot n$, то сложность будет $O(n^2)$. Памяти потребуется $n \cdot n \cdot 2$ (если использовать 2-байтные числа), то есть не более $2 \cdot 10^8$ байт.

Задача G - продолжение

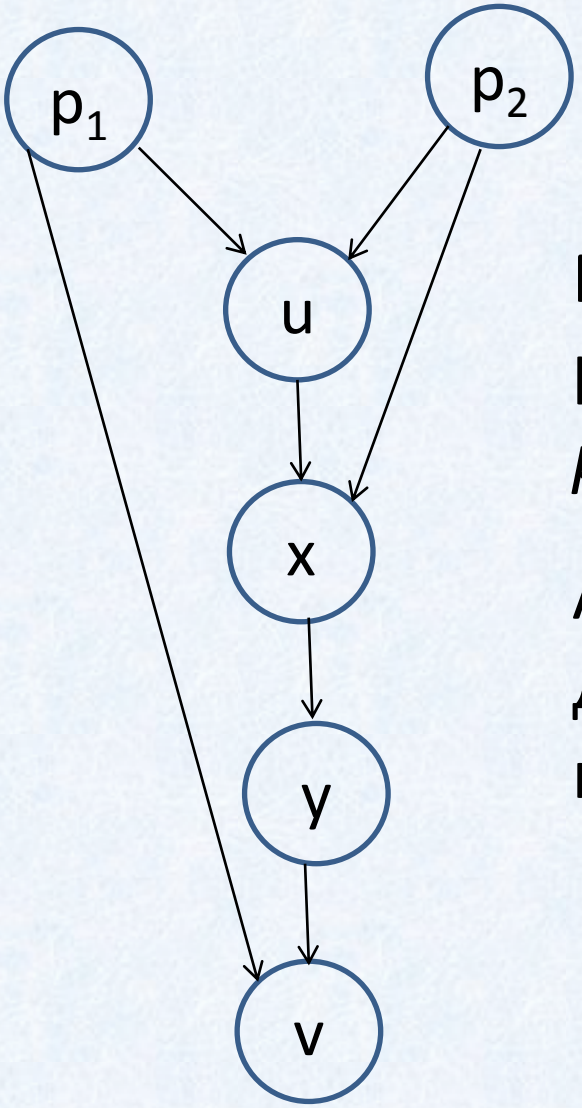
Этап 2. Теперь для каждой вершины вычислим искомые расстояния между ней и всеми остальными вершинами. Это можно также сделать поиском в глубину, только идти от детей к родителям (то есть развернув рёбра графа).

Пусть мы рассматриваем вершину u, её родители – p_1 и p_2
Для каждой вершины v минимальная дистанция между u и v – это минимум из:

- длина кратчайшего пути от u до v как до потомка *(*если v – потомок u*)*
- 1 + минимальная дистанция между p_1 и v
- 1 + минимальная дистанция между p_2 и v

Задача G - продолжение

Пример:



Минимальная дистанция от *u* до *v* равна минимальной дистанции от p_1 до *v* плюс 1.

А, например, минимальная дистанция от *u* до *y* равна длине кратчайшего пути до этого потомка

Задача H - «Красивые числа»

Пусть число в десятичной системе выглядит как $\overline{xk}$, то есть оно равно $x \cdot 10^{len(k)} + k$ (где *len(k)* – число цифр в *k*).
В *k*-й системе число кончается на 1 – это значит, что $x \cdot 10^{len(k)} + k - 1$ кратно *k*, отсюда: $x \cdot 10^{len(k)} - 1$ кратно *k*.

Пришли к задаче: найти минимальное положительное *x*, такое что $x \cdot 10^{len(k)} \equiv 1 \pmod{k}$.

Сравнение вида $ax \equiv b \pmod{k}$ решается известным методом. Пусть $d = \gcd(a, k) = \gcd(10^{len(k)}, k)$.
Сравнение не имеет решения, если *b* не делится на *d*. Так как *b*=1, то для наличия решения должно быть *d*=1.
А для этого нужно, чтобы k не делилось на 2 и 5.

Итак, если k делится на 2 или на 5, то решений нет.

Задача H - продолжение

При $d = 1$ сравнение имеет единственное решение – это обратный элемент к числу $a = 10^{len(k)}$ по модулю k.

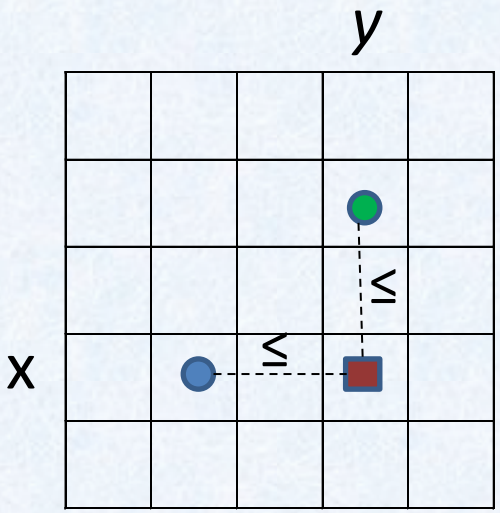
Обратный элемент к числу a по модулю k можно найти с помощью расширенного алгоритма Евклида:

для чисел a и k расширенный алгоритм Евклида находит два таких числа x и y, что $a \cdot x + k \cdot y = gcd(a, k)$.
Обратным элементом к a будет как раз число x. Точнее, $(x+k) \bmod k$ (так как x могло получиться отрицательным).

Задача I - «Седловая точка»

Максимин по строкам равен минимаксу по столбцам тогда и только тогда, когда в матрице существует элемент, который одновременно является минимальным в своей строке и максимальным в своём столбце.

Схема доказательства:



В прямую сторону – понятно из рисунка (синий кружок – максимин, зеленый – минимакс, квадратик – полученный седловой элемент).

В обратную сторону: если элемент – min в своей строке *x* и max в своём столбце y, то не может быть строк, в которых минимум будет больше (ведь каждая строка пересекает столбец y). Аналогично, нет столбцов, где максимум меньше.

Задача I - продолжение

Отсюда решение: перебираем элементы матрицы и проверяем, можно ли каждый элемент сделать седловым.

- Если элемент – минимум в своей строке, то его можно увеличивать вплоть до второго минимума. А если нет, то нужно заменить на минимум.

- Аналогично, если элемент – максимум в своём столбце, то его можно уменьшать вплоть до второго максимума. А если нет, то нужно заменить на максимум.

Если полученные интервалы пересекаются, седловая точка найдена.

Для эффективной реализации заранее преподсчитаем первый и второй минимум по строкам, а также первый и второй максимум по столбцам. Сложность – $O(n \cdot m)$.

Задача J - «Copy - Paste»

Варианты решения:

- суффиксное дерево
- суффиксный автомат

Пусть уже введён текст из *i* символов. Суффиксное дерево или автомат хранит все суффиксы этого текста и позволяет эффективно найти самую длинную подстроку, совпадающую со вставляемым далее текстом.

Затем дерево (или автомат) достраивается до той позиции, где закончился добавленный текст, и процесс повторяется.

Вычислительная сложность решения – O(n).