Председатель жюри: Кориненко Матвей
Состав жюри: Базилевич Григорий
Коробко Семен
Девятериков Иван
Поспелова Ольга

Спонсоры олимпиады: XSOLLA
PARMA Technologies Group
Knoema

Благодарность тем, кто протестировал задачи: Поспелов Сергей
Субботин Игорь
Перескокова Марина
Пупков Кирилл

Отдельная благодарность: Шульгина Галина Михайловна

Задача A. Собери костюм!

Автор и разработчик: Григорий Базилевич

Для решения второй подзадачи можно было написать переборное решение.

Для решения первой подзадачи нужно понять, какая ячейка будет e-й по счету. Для этого перейдем в индексацию с 0, тогда координата искомой ячейки в 0-индексации: (e // m // p, e // m % p, e % m), где % означает операцию остатка от деления, а // — операцию целочисленного деления. Перейдем в 1-индексацию: (e // m // $p+1$, e // m % $p+1$, e % $m+1$)

Для перехода к полному решению заметим, что мы можем свести задачу к ранее решенной. Нам необходимо найти $((e-1) \times \sum_i c_i) + \sum_i^{s-1} c_i + 1$ ячейку (в 1-индексации), потому что нам необходимо пропустить $(e-1)$ элемент костюма и еще $(s-1)$ размер искомого. Обозначим $x = (e-1) \times \sum_i c_i) + \sum_i^{s-1} c_i$, тогда искомая координата: (x // m // $p+1$, x // m % $p+1$, x % $m+1$).

Задача B. Ваня поступает в университет

Автор и разработчик: Григорий Базилевич

Для хранения информации о степенях дипломов воспользуемся структурой map. По ключу, в качестве которого возьмем название олимпиады, будем хранить список с баллами за степень диплома (например, в vector). Для каждого диплома мы сможем за $O(1)$ (или $O(\log n)$ в зависимости от реализации) находить его баллы.

Для решения первой подзадачи нам необходимо сравнить баллы единственного диплома с баллами на БВИ и ДВИ.

Для решения второй подзадачи нам необходимо выбрать диплом с наибольшими баллами и сравнить его с баллами на БВИ и ДВИ.

Для полного решения задачи сохраним баллы всех в рамках одного запроса дипломов в массив и отсортируем его по убыванию. Тогда баллы абитуриента — сумма баллов первых k элементов массива. Сравним их с баллами на БВИ и ДВИ, выведем ответ.

Задача C. Поход в магазин

Автор и разработчик: Ольга Поспелова

Построим граф, где дома — это вершины, мост — ребро с весом 1, телепорт — ребро с весом 0. Тогда в задаче требуется посчитать минимальное расстояние от вершины (X_h, Y_h) до вершины (X_s, Y_s). Это можно сделать с помощью обхода в ширину (01-bfs). В bfs давайте поддерживать дек (deque) вместо очереди (queue). Чтобы обработать ребро весом 1, мы добавим его в конец нашего дека. Чтобы обработать ребро весом 0, мы добавим его в начало нашего дека. Запустив bfs из

вершины (X_h, Y_h), мы сможем посчитать минимальное расстояние до вершины (X_s, Y_s). Так как каждую вершину мы добавляем не более одного раза и обрабатываем все ребра (мосты и телепорты), то решение работает за $O(nm+n(m-1)+m(n-1)+k)$. Эту задачу также можно решить с помощью алгоритма Дейкстры или другим алгоритмом поиска кратчайших путей.

Задача D. Урбанизация Лайнландии

Автор и разработчик: Матвей Кориненко

Подзадача 1 Для решения этой подзадачи можно было для запросов 1-го типа «в лоб» добавлять ребра в граф городов. То бишь если нам задан отрезок $[a, b]$, для всех городов отрезка $[a+1, b]$ можно добавить ребра из города a в города этого отрезка и обратные им.

Для проверки доступности города a из города b достаточно запускать обход в глубину по нашему графу из вершины a. Если вершина b будет посещена в конце обхода, значит, города достижимы друг из друга.

Обозначим за n длину всего отрезка Лайнландии.

Оба запроса будут выполняться за $O(n)$, тогда итоговая сложность решения $O(q \cdot n)$.

Подзадача 2 Давайте заметим, что если до операции первого типа пара городов была достижима друг из друга, то и после нее они будут достижимы. Соответственно, нам не требуется хранить полную информацию о всех ребрах. Достаточно уметь объединять города в множества.

Давайте в качестве идентификатора множества будем использовать номер вершины, которая стоит в конце отрезка (заметим, что если вершины a и b достижимы друг из друга, то и другие вершины между ними достижимы из них). Изначально будет $r - l + 1$ идентификаторов, то бишь $r - l + 1$ отрезков длины 1. Для хранения этих идентификаторов используем структуру данных *set*.

Операция первого типа для отрезка $[a, b]$ будет равносильна удалению из *set* всех элементов на отрезке $[a, b-1]$. Заметим, что они уже никогда не станут идентификатором любого из отрезков.

Тогда операцию второго типа можно интерпретировать как проверку равенства идентификаторов множеств для городов a, b. Это можно проверять, используя метод *lower_bound*.

Обозначим за n длину всего отрезка Лайнландии.

Суммарно операций удаления элементов из *set* будет не более n, каждая из которых будет выполняться за $\log(n)$.

Операция *lower_bound* будет выполняться не более q раз, каждый раз работая с временной сложностью $\log(n)$.

Соответственно суммарная сложность решения будет $O((n + q) \cdot log(n))$.

Задача E. Петя и web-сервис

Автор и разработчик: Коробко Семён

Для решения первых двух позадач будет достаточно простого ДП: пусть dp_{ij} — сколько существует способов загадать i чисел и оказаться в вершине j. Тогда переход:

$$dp_{(i+1)j} = \sum_k dp_{ik}$$

где k — вершина, из которой существует портал в j, причём $dp_{01} = 1$. Тогда ответ:

$$\sum_{i=1}^{K} dp_{La_i}$$

.

Асимптотика такого решения $O(L(N + M))$, так как для каждого числа от 1 до L мы пробегаем весь граф.

Для решения на полный балл, нужно знать следущий факт, который часто применяется для оптимизаций ДП: пусть у нас есть невзвешанный граф и его матрица смежности A. Пусть $B = A^K$.

Тогда b_{ij} — количество способов добраться из i в j за K шагов. Этот факт можно доказать по индукции, посмотрев на формулу умножения двух матриц.

Тогда составим матрицу смежности для нашего графа. Не забудем, что

$$a_{ii} = E - \sum_{i \neq j} a_{ij}$$

Тогда ответ:

$$\sum_{i=1}^{N} b_{1i}$$

где $B = A^L$.

Такое решение будет работать за $O(n^3 \log L)$ (если вы не умеете перемножать матрицы быстрее чем за $O(n^3)$).

Задача F. Перемножь последовательность!

Автор и разработчик: Иван Девятериков

Подзадача 1 (общая идея) Для решения первой подзадачи нужно было на каждый запрос типа ? итерироваться циклом, пока произведение не превзойдёт x. Для запроса типа = нужно обновить значение в массиве. Асимптотика решения: $O(n)$ на запрос типа ?, и $O(1)$ на запрос типа =. В худшем случае, если у нас все запросы вида ?, мы получаем асимптотику $O(q \times n)$.

Подзадача 1 Переполнение не могло случиться. Если в какой-то момент произведение становилось больше, чем 10^{18}, можно сразу прерывать цикл и отвечать на запрос. Для решения на языке c++ надо было использовать тип данных long long.

C++ «безопасное перемножение» Для решения всех подзадач на языке c++ (кроме первой) нужно было реализовать функцию для безопасного умножения, так как умножение двух больших чисел могло не влезать в нужный тип. В C++ тип long long или даже int128_t спокойно переполнялся при умножении двух больших чисел.

Данная функция перемножает два числа, и если результат будет больше чем MX, то возвращает -1. Для простоты будем считать MX равной LLONG_MAX или в условиях данной задачи 10^{18}

```
ll mult(ll a, ll b) {
    if (a <= MX / b) {
        return a * b;
    } else {
        return -1;
    }
}
```

Подзадача 2 (множество ответов) В данной подзадаче только запросы вида ?.

Будем решать задачу оффлайн — не будем отвечать на запросы сразу. Создадим массив *querys* запросов, для каждой пары (i, x), запишем в $querys[i]$ пару из x и номера запроса.

```
for (int id = 0; id < q; id++) {
    char t;
    int i;
    ll x;
    cin >> t >> i >> x;
    querys[i - 1].push_back({x, id});
}
```

Будем поддерживать все возможные различные произведения, которые не превышают диапазон MX и начинаются в позиции i. Размер такого множества всего $log_2 MX$, округлённый вверх, из-за того, что новый (неравный старому) результат умножения будет минимум в 2 раза больше. Такое множество — это просто массив пар, сначала идёт значение произведения, а потом длина отрезка, на котором всё ещё остаётся такое же произведение. Это нужно для случая, когда у нас будет блок из подряд идущих единиц.

Тогда для i, используя посчитанное множество, нужно для каждого запроса из $querys[i]$ проитерироваться по множеству ответов и запросто посчитать ответ (для решения можно было использовать даже бинарный поиск по данному множеству, но размер множества и так очень мал, и это никак не давало преимуществ в скорости).

Для вычисления данного множества не за $O(n)$ для каждого i попробуем использовать посчитанные данные для $i-1$. Действительно, при переходе с $i-1$-го на i-е множество произведений меняет свои значения с x на $\frac{x}{a_{i-1}}$, также иногда удаляется первый элемент из начала, когда $x = 1$, а также надо попробовать расширить наше множество. Для этого надо было ещё поддерживать указатель на последний элемент, добавленный в множество. Для удаления из начала надо было использовать структуру данных очередь.

Асимптотика решения $O(n + q \log_2 10^{18})$ или при использовании бинарного поиска $O(n + q \log_2(\log_2 10^{18}))$.

Подзадача 2 (быстрые прыжки, приближаемся к полному решению) В данной подзадаче только запросы вида ?.

Простая итерация на запрос вида ? i x, как в подзадаче номер один, была, в худшем случае, за $O(n)$, а не за $O(\log_2 x)$, так как только на тестах с единицами получаются блоки, в которых произведение никак не меняется. Чтобы быстро пропускать такие блоки с единицами, нужно заранее подсчитать массив nxt, где $nxt[i]$ - следующий индекс j, такой, что $i \leqslant j$ и $a_j \neq 1$. Посчитать такой массив можно за одну итерацию в самом начале до обработки запросов, достаточно итерироваться с конца.

```
cin >> t >> i >> x;
i--;
ll res = 1;
bool pr = false;
int j = i;
while (j < n) {
    j = nxt[j];
    if (j == -1) break;
    res = mult(res, a[j]);
    if (res == -1 || res > x) {
        cout << j + 1 << '\n';
        pr = true;
        break;
    }
    j++;
}
if (!pr) {
    cout << -1 << '\n';
}
```

Подзадача 1+2 Достаточно было считать все запросы, и, в зависимости от того, есть ли запросы вида =, запустить соответствующее решение.

Подзадача 3 Воспользуемся идеей корневой декомпозиции на массиве. Разобьём массив на блоки длиной $\sqrt{n}$ и посчитаем их произведение. Подробнее про эту замечательную технику можно

прочитать тут: https://ru.algorithmica.org/cs/decomposition/sqrt-heuristics/.

Подзадача 4 (быстрые прыжки с изменениями) Модернизируем наш алгоритм из подзадачи номер 2 (с массивом nxt), чтобы работали для запросов вида =.

Заведём структуру данных set, в сете будем хранить все индексы элементов, которые не равны 1. Тогда для получения следующего за i элемента, который не равен 1, достаточно сделать upper_bound.

Запрос вида = просто меняет или не меняет значения в сете с индексом i.

Асимптотика (в худшем случае, если у нас только запросы вида ?) $O(q \times \log_2 x \log_2 n)$ — на деле данное решение работает гораздо быстрее.

Подзадача 4 (спуск по Дереву Отрзков) Можно сделать бинарный поиск по ответу (по индексу j), а произведение получать с помощью запроса в Дерево Отрезков на операцию произведения, но это очень неэффективно ($O(log_2^2 n)$ на запрос и большая временная константа. Чтобы заменить неэффективный $log_2^2 n$ на $log_2 n$ надо написать обычный спуск по Дереву Отрезков.

```
pair<ll, int> query(int v, int tl, int tr, int pos, ll x, ll cur = 1) {
    if (pos >= tr) return {1, -1};
    ll res = mult(cur, t[v]);
    if (tl + 1 == tr) return {res, (res > x || res == -1 ? tl : -1)};
    if (pos <= tl && res != -1 && res <= x) return {res, -1};
    int tm = (tl + tr) / 2;
    auto resL = query(2 * v, tl, tm, pos, x, cur);
    if (resL.second != -1) return resL;
    return query(2 * v + 1, tm, tr, pos, x, resL.first);
}
```

Каждый спуск работает за $\log_2 n$, потому что этот код эквивалентен следующему подходу:

1. Разобьём отрезок запроса на вершины.
2. В полученном массиве вершин размера $O(\log_2 n)$ найдём за линию префикс, на котором произведение не меньше требуемого.
3. В граничной вершине запустили обычный спуск за O(глубина).

Асимптотика $O(q \times \log_2 n)$.

Заключение Решение с сетом достаточно производительное и на некоторых тестах даже быстрее решения с ДО. Все возможные реализации решения можно посмотреть в архиве к данной задаче.

Например, совершенно другая задача с похожей идеей про сет.

https://codeforces.com/problemset/problem/920/F

Для быстрой обработки суммы будем использовать ДО (ДО — сокращение от слова Дерево Отрезков). А используя тот факт, что при операции замены a_i на $D(a_i)$ число a_i будет значительно уменьшаться, а запросов на другое изменение массива у нас нет, после 6-ти шагов наше число превратится в 1 или в 2, и далее значение не будет меняться. Для запросов обновления создадим сет индексов, у которых значение не равно 1 или 2. При запросе обновления отрезка проитерируемся по индексам между границами отрезка и обновим значение с помощью обычного обновления в ДО.

Также эту задачу можно решить без сета, только с помощью ДО. Для этого надо в вершине сохранить дополнительную информацию: полностью ли отрезок, за который отвечает вершина, состоит из значений 1 или 2.