Planets Queries I

You are playing a game consisting of n planets.
Each planet has a teleporter to another planet (or the planet itself).

Your task is to process q queries of the form:
when you begin on planet x and travel through k teleporters,

which planet will you reach?

Input

The first input line has two integers n and q: the number of planets and queries.
The planets are numbered 1,2,...,n.

The second line has n integers t_1,t_2,...,t_n: for each planet, the destination of the teleporter. It is possible that t_i=i.

Finally, there are q lines describing the queries.
Each line has two integers x and k: you start on planet x and travel through k teleporters.

Output

Print the answer to each query.

Constraints

1 <= n, q <= 2 . 10^5
1 <= t_i <= n
1 <= x <= n
0 <= k <= 10^9

Example

Input:

```
4 3
2 1 1 4
1 2
3 4
4 1
```

Output:

```
1
2
4
```

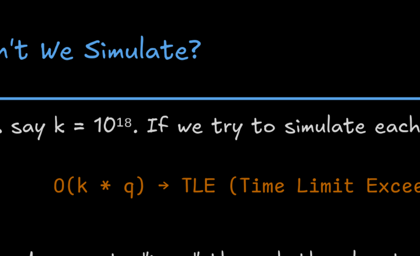
Problem Breakdown

Lets understand the problem first

You're given n planets. Each planet has exactly one portal that leads to another planet (possibly itself).

You're also given q queries, where each query asks:

If I start at planet a, where will I be after k portal jumps?



Why Can't We Simulate?

Let's say $k = 10^{18}$. If we try to simulate each of the k steps naively, the time complexity will be:

O(k * q) → TLE (Time Limit Exceeded)

We need a way to "jump" through the planets faster.

Let's look at a similar problem (k-th ancestor)

nodes

kth step

	1	2	3	4
1	1	1	1	3
2	1	1	1	1
3	1	1	1	1

there's only one outgoing edge from every node so we can just use parent[] array to construct first row

Now for the second row instead of going all the way we will just ask our parent who is their (k-1)th ancestor and so on

Like if we want to know the 3rd ancestor of node 4 instead of doing this what we will do is since 3 is 4's parent then 4 will just ask 3 (its parent) what's your 2nd(k-1) ancestor and so on

But we fail here as k = 10^9 which is just too much and we will get TLE so we need a better approach

Binary Lifting

It's a technique that lets you jump in powers of two instead of jumping one step at a time.

Why is this useful?

Because any number k can be written as a sum of powers of 2.
For example:

$k = 13 = 8 + 4 + 1 = 2^3 + 2^2 + 2^0$

So, instead of 13 individual steps, we do just 3 jumps:

Jump 8 steps

Then 4 steps

Then 1 step

Each of these jumps can be done in constant time using preprocessed data.

nodes

kth step

	1	2	3	4
1	1	1	1	3
2	1	1	1	1
3	1	1	1	1

2^3 = 8

in rows now we will store 2^i answers

Algorithm

```cpp
#include <bits/stdc++.h>
using namespace std;

const int LOG = 30; // Enough for k up to 1e9 (2^30 > 1e9)
vector<vector<int>> up; // up[i][j] = planet after 2^i jumps from j

void build(const vector<int>& parents) {
    int n = parents.size();
    up.assign(LOG, vector<int>(n));

    // Base case: 2^0 = 1 jump
    for (int j = 0; j < n; j++) {
        up[0][j] = parents[j];
    }

    // Fill the binary lifting table
    for (int i = 1; i < LOG; i++) {
        for (int j = 0; j < n; j++) {
            up[i][j] = up[i-1][up[i-1][j]];
        }
    }
}
```

```
up[0][i] = p[i]; // Where do you land after 2⁰ = 1 jump from i

up[i][j] = up[i-1][up[i-1][j]];

This means:

    To find the 2^i-th jump from node j:
    → First jump 2^(i-1) from j, then jump 2^(i-1) again from that location.

Thus we compute:

    up[1][i] = 2 jumps from i

    up[2][i] = 4 jumps from i

    ...

    up[30][i] = 2³⁰ jumps from i
```

```cpp
int query(int start, int jumps) {
    int current = start;
    for (int i = 0; i < LOG; i++) {
        if (jumps & (1 << i)) { // If the i-th bit is set
            current = up[i][current];
        }
    }
    return current;
}
```

check if ith bit is set
then jump up

```cpp
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n, q;
    cin >> n >> q;

    vector<int> parents(n);
    for (int i = 0; i < n; i++) {
        cin >> parents[i];
        parents[i]--; // Convert to 0-based
    }
```

Reading input and creating parent array

```cpp
    build(parents);

    while (q--) {
        int a, k;
        cin >> a >> k;
        a--; // Convert to 0-based
        cout << query(a, k) + 1 << '\n'; // Convert back to 1-based for output
    }
```

Answering Queires

```cpp
    return 0;
}
```

Time Complexity

Preprocessing (build)	O(n * log k)
Query (query)	O(log k)
Total for q queries	O((n + q) * log k)

Space Complexity

O(n * log k)

Common Questions

Why is MAX = 30?

Because 2^{30} is ~10^9. Even 2^{60} is sufficient for 10^{18}.

Can this be used in tree problems too?

YES! Binary Lifting is commonly used in Lowest Common Ancestor (LCA) problems on trees.

Is this similar to matrix exponentiation?

Similar concept: breaking down large operations using powers of 2.

Real-World Applications

Jump queries in linked structures or parent chains

LCA in trees

Range lifting in sparse structures

Time travel simulations in functional graphs