Coin Path

Given an array `A` (index starts at `1`) consisting of N integers: $A_1, A_2, ..., A_N$ and an integer `B`. The integer `B` denotes that from any place (suppose the index is `i`) in the array `A`, you can jump to any one of the place in the array `A` indexed `i+1`, `i+2`, …, `i+B` if this place can be jumped to. Also, if you step on the index `i`, you have to pay A_i coins. If A_i is -1, it means you can't jump to the place indexed `i` in the array.

Now, you start from the place indexed `1` in the array `A`, and your aim is to reach the place indexed `N` using the minimum coins. You need to return the path of indexes (starting from 1 to N) in the array you should take to get to the place indexed `N` using minimum coins.

If there are multiple paths with the same cost, return the lexicographically smallest such path.

If it's not possible to reach the place indexed N then you need to return an empty array.

Example 1:

```
Input: [1,2,4,-1,2], 2
Output: [1,3,5]
```

Example 2:

```
Input: [1,2,4,-1,2], 1
Output: []
```

Note:

1. Path $Pa_1, Pa_2, ..., Pa_n$ is lexicographically smaller than $Pb_1, Pb_2, ..., Pb_m$, if and only if at the first `i` where Pa_i and Pb_i differ, Pa_i i; when no such `i` exists, then `n` m.
2. A_1 >= 0. $A_2, ..., A_N$ (if exist) will in the range of [-1, 100].
3. Length of A is in the range of [1, 1000].
4. B is in the range of [1, 100].

Solution 1

The following solution is based on that:

If there are two path to reach `n`, and they have the same optimal cost, then the longer path is lexicographically smaller.

Proof by contradiction:
Assume path `P` and `Q` have the same cost, and `P` is strictly shorter and `P` is lexicographically smaller.
Since `P` is lexicographically smaller, `P` and `Q` must start to differ at some point.
In other words, there must be `i` in `P` and `j` in `Q` such that `i < j` and
`len([1...i]) == len([1...j])`
`P = [1...i...n]`
`Q = [1...j...n]`
Since `i` is further away from `n` there need to be no less steps taken to jump from `i` to `n` **unless `j` to `n` is not optimal**
So `len([i...n]) >= len([j...n])`
So `len(P) >= len(Q)` which contradicts the assumption that `P` is strictly shorter.

For example:
Input: `[1, 4, 2, 2, 0], 2`
Path `P = [1, 2, 5]`
Path `Q = [1, 3, 4, 5]`
They both have the same cost `4` to reach `n`
They differ at `i = 2` in `P` and `j = 3` in `Q`
Here `Q` is longer but not lexicographically smaller.
Why? Because `j = 3` to `n = 5` is not optimal.
The optimal path should be `[1, 3, 5]` where the cost is only `2`

```java
    public List<Integer> cheapestJump(int[] A, int B) {
        int n = A.length;
        int[] c = new int[n]; // cost
        int[] p = new int[n]; // previous index
        int[] l = new int[n]; // length
        Arrays.fill(c, Integer.MAX_VALUE);
        Arrays.fill(p, -1);
        c[0] = 0;
        for (int i = 0; i < n; i++) {
            if (A[i] == -1) continue;
            for (int j = Math.max(0, i - B); j < i; j++) {
                if (A[j] == -1) continue;
                int alt = c[j] + A[i];
                if (alt < c[i] || alt == c[i] && l[i] < l[j] + 1) {
                    c[i] = alt;
                    p[i] = j;
                    l[i] = l[j] + 1;
                }
            }
        }
        List<Integer> path = new ArrayList<>();
        for (int cur = n - 1; cur >= 0; cur = p[cur]) path.add(0, cur + 1);
        return path.get(0) != 1 ? Collections.emptyList() : path;
    }
```

written by yuxiangmusic original link here

Solution 2

This is a classic DP problem. dp[k] (starting from k = 0) is the minimum coins from A_{k+1} to A_n, and pos[k] is the next place to jump from A_{k+1}.

If working backward from dp[n-1] to dp[0], and considering smaller index first, i.e. i+1 to i+B, there is no need to worry about lexicographical order. I argue pos[k] always holds the lexicographically smallest path from k to n-1, i.e. from A_{k+1} to A_n. The prove is as below.

Clearly, when k = n-1, it is true because there is only 1 possible path, which is [n]. When k = i and i < n-1, we search for an index j, which has smallest cost or smallest j if the same cost. If there are >= 2 paths having the same minimum cost, for example,
P = [k+1, j+1, ..., n]
Q = [k+1, m+1, ..., n] (m > j)
The path P with smaller index j is always the lexicographically smaller path.
So the argument is true by induction.

```
class Solution {
public:
    vector<int> cheapestJump(vector<int>& A, int B) {
        vector<int> ans;
        if (A.empty() || A.back() == -1) return ans;
        int n = A.size();
        vector<int> dp(n, INT_MAX), pos(n, -1);
        dp[n-1] = A[n-1];
        // working backward
        for (int i = n-2; i >= 0; i--) {
            if (A[i] == -1) continue;
            for (int j = i+1; j <= min(i+B, n-1); j++) {
                if (dp[j] == INT_MAX) continue;
                if (A[i]+dp[j] < dp[i]) {
                    dp[i] = A[i]+dp[j];
                    pos[i] = j;
                }
            }
        }
        // cannot jump to An
        if (dp[0] == INT_MAX) return ans;
        int k = 0;
        while (k != -1) {
            ans.push_back(k+1);
            k = pos[k];
        }
        return ans;
    }
};
```

written by zestypanda original link here

Solution 3

```java
public List<Integer> cheapestJump(int[] A, int B) {

    int n = A.length;
    int[] dp = new int[n], decisions = new int[n];
    dp[n-1] = A[n-1];
    decisions[n-1] = A[n-1] == -1 ? -2 : -1;
    for (int i=n-2;i>=0;i--) {
        int minHopValue = Integer.MAX_VALUE, minHopIndex = -2;
        for (int j=i+1;j<Math.min(n, i+B+1);j++) {
            if (dp[j] < minHopValue && decisions[j] != -2) {
                minHopValue = dp[j];
                minHopIndex = j;
            }
        }
        dp[i] = A[i] + minHopValue;
        decisions[i] = A[i] == -1 ? -2 : minHopIndex;
    }

    // Construct Path
    List<Integer> res = new LinkedList<>();
    if (decisions[0] == -2) return res;
    int k = 0;
    while (k != -1) {
        res.add(k+1);
        k = decisions[k];
    }

    return res;

}
```

written by compton_scatter original link here