Questions on jump game

01 jump game

Eg 1: arr[] = [2 3 1 0 4] → Yes

Eg 2: arr[] = [1 2 3 1 1 0 2 5] → No not possible.

- array jut has the number then always possible, jumps possible.

Eg: arr[] = [3 2 1 0 4] → No

Eg. [1 2 4 1 1 0 2 5] → Yes

0 1 2 3 4 5 6 7

maxInd = ~~0~~ ~~1~~ ~~3~~ ~~3~~ 3

done ans

```
bool canJump(vector<int>& nums){
    int maxInd = 0;
    for(int i=0; i<nums.size(); i++){
        if(i > maxInd){
            return false;
        }
        maxInd = max(maxInd, i+nums[i]);
    }
    return true;
}
```

02 jump game II

arr[] = [2 3 1 1 4] (indexes 0 1 2 3 4)

min no. of more required to reach at last.

(index, no. of jumps)

F(0,0)

+1 → F(1,1)

+2 → F(2,2)

F(1,1): +1 → F(2,2); +2 → F(3,2); +3 → F(4,2)

F(4,2) → min no. jumps = 2

```
f(ind, jump){
    if(ind >= n-1)
        return jumps;
    mini = INT_MAX;
    for(i=1; i<=arr[ind]; ){
        mini = min(mini, f(ind+i, jump+1));
    }
    return mini;
}
```

- memoization will take

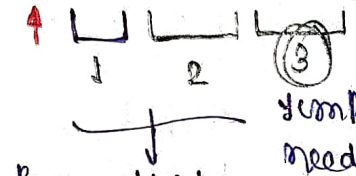
TC = O(N²)

SC = O(N²).

TC = N^N

SC = O(N).

Eg. arr = [2 3 1 4 1 1 1 2]



jumps = 0 1 2 (3) Ans

```
Func ( arr) {
    jumps = 0, l = 0, r = 0;
    while ( r < n-1) {
        Farthest = 0
        For ( ind = l → r) {
            Farthest = max (farthest,
                arr [ind] + i) ;
        }
        l = r + 1;
        r = Farthest ;
        jumps = jumps + 1 ;
    }
    return jumps ;
```

TC = O(N)

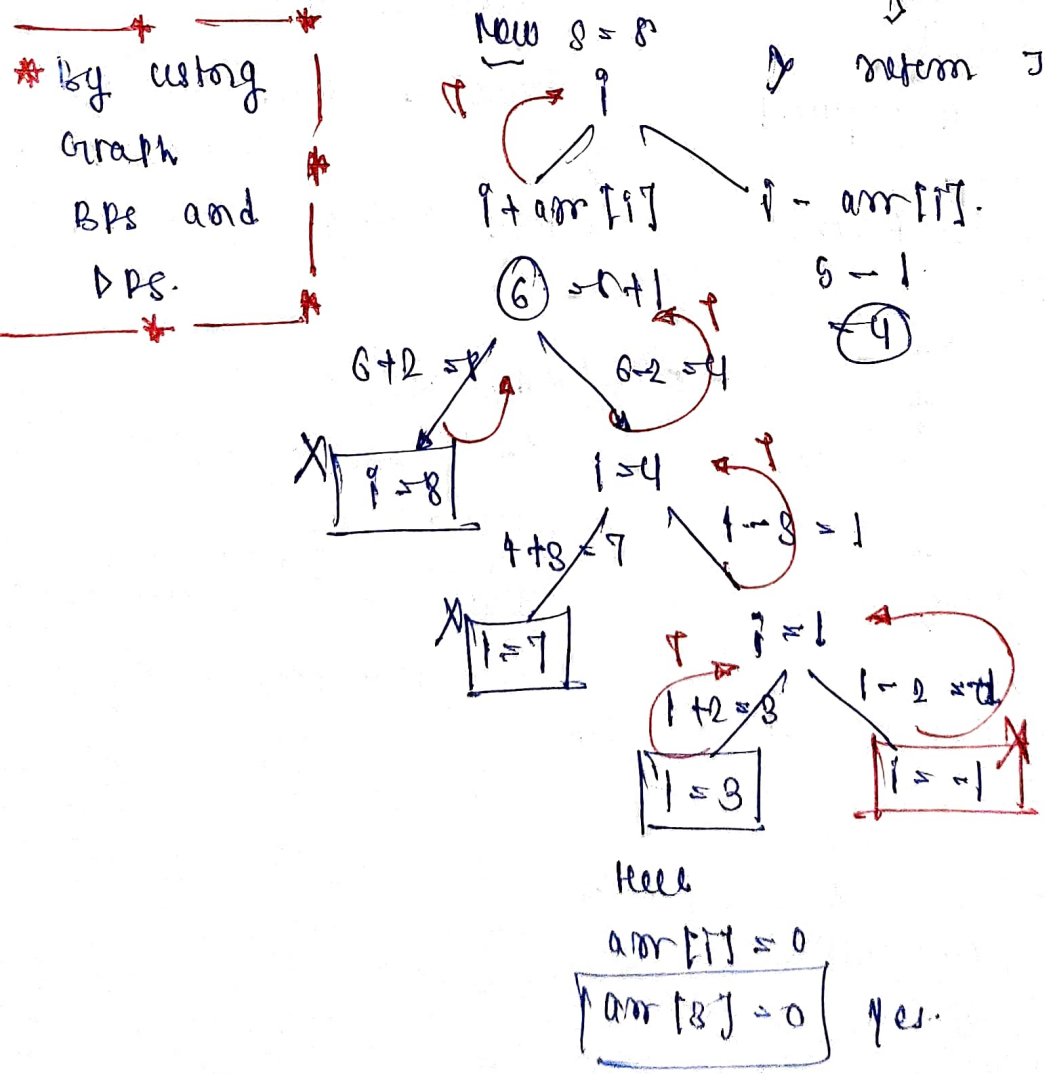
SC = O(1).

03 Jump game III

	0	1	2	3	4	5	6
Eg	4	2	3	0	3	1	2

New S = 5

* By using Graph BFS and DFS.



Here

arr[3] = 0

arr[8] = 0 Yes.

04 Jump game IV

Eg. [100 −23 −23 404 100 23 23 23 3 404]
0 1 2 3 4 5 6 7 8 9

**

0 → 4 → 8 → 9
1 2 3
100 100 404 404.

min steps = 3.

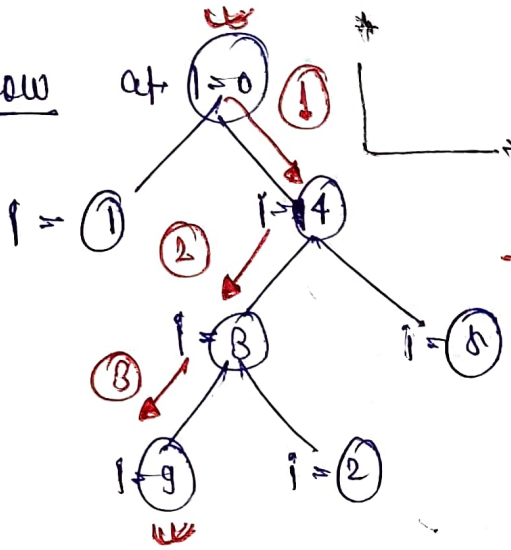
Each step (i)

(arr[i] == arr[j])

Hash map:

- 100 → 0, 4
- −23 → 1, 2
- 404 → 8, 9
- 23 → 5, 6, 7
- 3 → 3

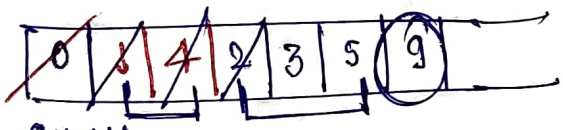
now at i=0 (1)



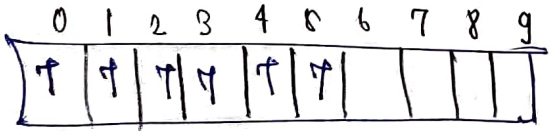
→ Here indexes act as nodes and steps will be edges

→ We have to find shortest path.

We BFS.



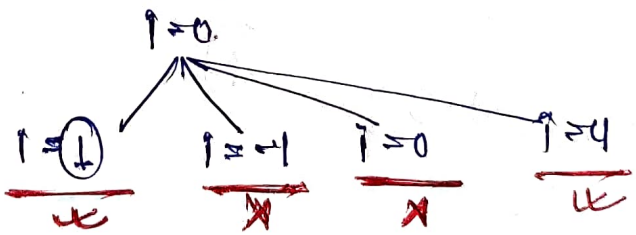
queue



visited

now

node i=0.

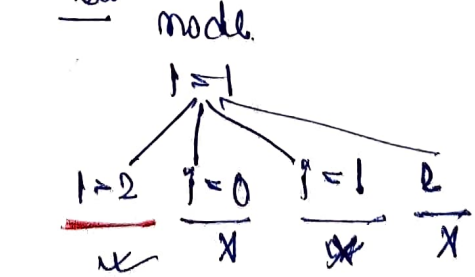


steps = ~~0~~ ~~1~~ ~~2~~ 3

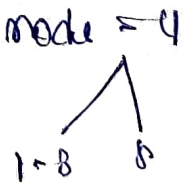
And will be given by BFS.

now

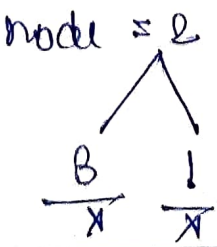
node i=1



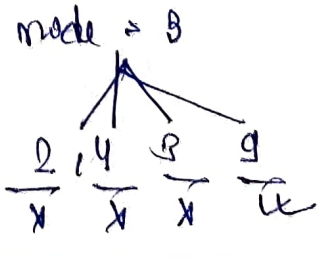
node = 4



node = 2

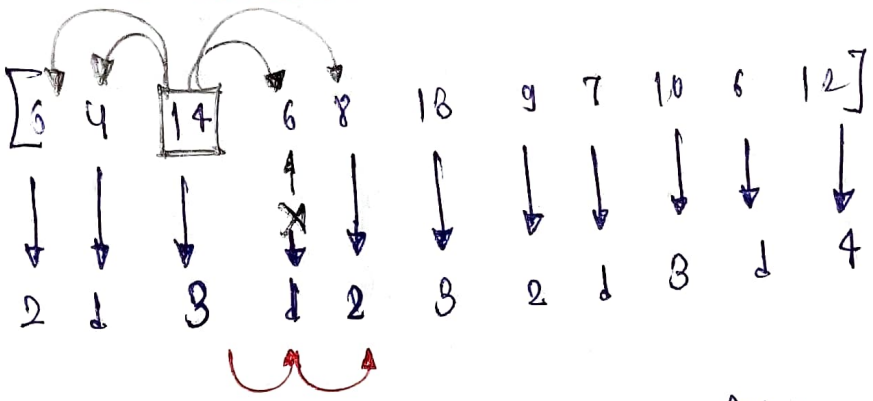


node = 8



TC = O(n)
SC = O(n).

05 jump game - 6



So ans will be 4

jumps

- 14 → 6 → 4 (calculated)
  total = 3.
- 14 → 4
  total = 2
- 14 → 6
  total = 2
- 14 → 8
  total = 3

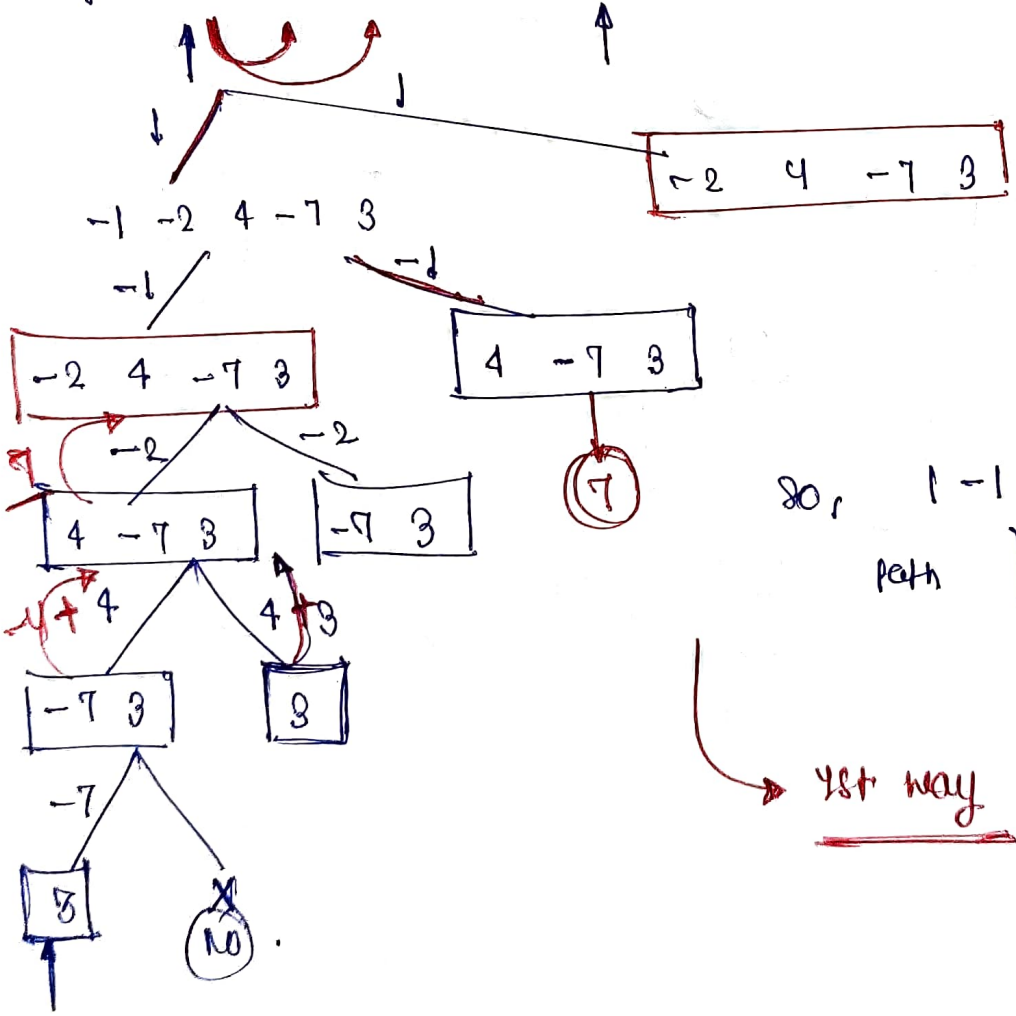
max = 3

new recursion

TC = O(n*k)

06 jump game - 6.

Eg. 1 -1 -2 4 -7 3.



memoization

dp

So, 1 -1 +7 = 0 +7 = 7

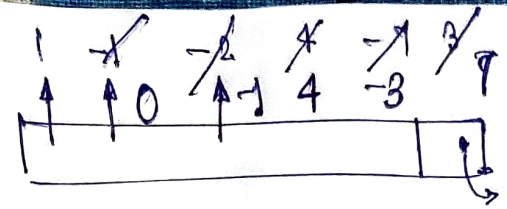
path 1 -1 4 3

1st way

1 -1 +(-2) +4 +(-7) +3

= -2

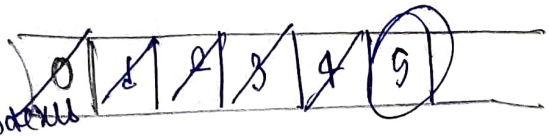
2nd way



last index pe aane ka max score leya hoga.

use deque type

[sliding window maximum].



index deque

$-1+1=0$

$-2+1=-1$

$4+0=4$

$-7+4=-3$

$8+4=1$

- agar arr change hua or usska value zada hai pehle queue wale index se to usko remove kr denge queue se.

07 Jump game 7

Eg. $s = 0\,|\,|\,0\,|\,|\,|\,0$

minJ = 2

maxJ = 3

true.

Eg $s = 0\,|\,|\,0\,|\,|\,|\,0$

minJ = 2

maxJ = 3

false

```
bool canReach(string s, int minJump, int maxJump) {
    int n = s.size();
    vector<bool> dp(n, false);
    dp[0] = true;
    int prev = 0;
    for (int i = 1; i < n; i++) {
        if (i >= minJump) prev = prev + dp[i - minJump];
        if (i > maxJump) prev = prev - dp[i - maxJump - 1];
        dp[i] = (s[i] == '0') && (prev > 0);
    }
    return dp[n-1];
}
```

JUMP GAME PATTERN QUESTIONS

QUESTIONS ON JUMP GAME

01.) JUMP GAME (LC-55)

```cpp
class Solution {
public:
    bool canJump(vector<int>& nums) {
        int maxInd=0;
        for(int i=0;i<nums.size();i++){
            if(i>maxInd)
            return false;
            maxInd=max(maxInd, i+nums[i]);
        }
        return true;
    }
};
```

JUMP GAME PATTERN QUESTIONS

02.) JUMP GAME II (LC-45)

RECURSION (TLE)

```
class Solution {
public:
    int func(int ind, int jump, vector<int> &nums){
        int n=nums.size();
        if(ind>=(n-1))
        return jump;

        int mini=INT_MAX;
        for(int i=1;i<=nums[ind];i++){
            mini=min(mini, func(ind+i, jump+1, nums));
        }
        return mini;
    }
    int jump(vector<int>& nums) {
        return func(0, 0, nums);
    }
};
```

MEMOIZATION (TLE)

```
class Solution {
public:
    int func(int ind, int jump, vector<int> &nums, vector<vector<int>> &dp){
        int n=nums.size();
        if(ind>=(n-1))
        return jump;
        if(dp[ind][jump]!=-1)
        return dp[ind][jump];

        int mini=INT_MAX;
        for(int i=1;i<=nums[ind];i++){
            mini=min(mini, func(ind+i, jump+1, nums, dp));
        }
        return dp[ind][jump]=mini;
    }
    int jump(vector<int>& nums) {
        int n=nums.size();
        vector<vector<int>> dp(n, vector<int> (n, -1));
        return func(0, 0, nums, dp);
    }
};
```

JUMP GAME PATTERN QUESTIONS

MEMOIZATION (GOOD)

```cpp
class Solution {
public:
    int jump(vector<int>& nums) {
        vector<int> memo(nums.size(), -1);
        return jumpFromPosition(0, nums, memo);
    }
private:
    int jumpFromPosition(int position, vector<int>& nums, vector<int>& memo) {
        if (position >= nums.size() - 1) {
            return 0;
        }

        if (memo[position] != -1) {
            return memo[position];
        }

        int furthestJump = min(position + nums[position],
                               static_cast<int>(nums.size() - 1));
        int minJumps = INT_MAX;

        for (int nextPosition = position + 1; nextPosition <= furthestJump;
             ++nextPosition) {
            int jumps = jumpFromPosition(nextPosition, nums, memo);
            if (jumps != INT_MAX) {
                minJumps = min(minJumps, jumps + 1);
            }
        }

        memo[position] = minJumps;
        return memo[position];
    }
};
```

JUMP GAME PATTERN QUESTIONS

TABULATION (GOOD)

```cpp
class Solution {
public:
    int jump(vector<int>& nums) {
        int n = nums.size();
        if (n == 1) return 0;

        vector<int> dp(n, INT_MAX);
        dp[0] = 0;

        for (int i = 0; i < n; ++i) {
            for (int j = i + 1; j <= i + nums[i] && j < n; ++j) {
                dp[j] = min(dp[j], dp[i] + 1);
            }
        }

        return dp[n - 1];
    }
};
```

GREEDY APPROACH

```cpp
class Solution {
public:
    int jump(vector<int>& nums) {
        int n=nums.size();
        int jumps=0;
        int l=0;
        int r=0;

        while(r<n-1){
            int farthest=0;
            for(int i=l;i<=r;i++){
                farthest=max(farthest, nums[i]+i);
            }
            l=r+1;
            r=farthest;
            jumps=jumps+1;
        }
        return jumps;
    }
};
```

JUMP GAME PATTERN QUESTIONS

03.) JUMP GAME III (LC-1306)

DFS APPROACH

```
class Solution {
public:
    bool dfs(vector<int> &arr, int index){
        if(index<0 || index>=arr.size() ||arr[index]<0)
        return false;

        if(arr[index]==0)
        return true;

        int jump=arr[index];
        arr[index]=-arr[index];

        bool left=dfs(arr, index-jump);
        bool right=dfs(arr, index+jump);

        return left || right;
    }
    bool canReach(vector<int>& arr, int start) {
        return dfs(arr, start);
    }
};
```

JUMP GAME PATTERN QUESTIONS

BFS APPROACH

```cpp
class Solution {
public:
    bool canReach(vector<int>& arr, int start) {
        int n = arr.size();
        vector<bool> visited(n, false);
        queue<int> q;

        q.push(start);
        visited[start] = true;

        while (!q.empty()) {
            int index = q.front();
            q.pop();

            if (arr[index] == 0) {
                return true;
            }

            int left = index - arr[index];
            int right = index + arr[index];

            if (left >= 0 && !visited[left]) {
                q.push(left);
                visited[left] = true;
            }

            if (right < n && !visited[right]) {
                q.push(right);
                visited[right] = true;
            }
        }

        return false;
    }
};
```

JUMP GAME PATTERN QUESTIONS

04.) JUMP GAME IV (LC-1345)

BFS APPROACH

```cpp
class Solution {
public:
    int minJumps(vector<int>& arr) {
        int n = arr.size();
        if (n == 1) return 0;

        unordered_map<int, vector<int>> graph;
        for (int i = 0; i < n; ++i) {
            graph[arr[i]].push_back(i);
        }

        queue<int> q;
        q.push(0);

        vector<bool> visited(n, false);
        visited[0] = true;

        int steps = 0;

        while (!q.empty()) {
            int size = q.size();
            while (size--) {
                int index = q.front();
                q.pop();

                if (index == n - 1) return steps;

                vector<int>& nextIndices = graph[arr[index]];
                nextIndices.push_back(index - 1);
                nextIndices.push_back(index + 1);

                for (int nextIndex : nextIndices) {
                    if (nextIndex >= 0 && nextIndex < n &&
                        !visited[nextIndex]) {
                        q.push(nextIndex);
                        visited[nextIndex] = true;
                    }
                }
                nextIndices.clear();
            }
            steps++;
        }
        return -1;
    }
};
```

JUMP GAME PATTERN QUESTIONS

05.) JUMP GAME V (LC-1340)

```cpp
class Solution {
public:
    int maxJumps(vector<int>& arr, int d) {
        int n = arr.size();
        vector<int> memo(n, -1);
        int result = 0;

        for (int i = 0; i < n; ++i) {
            result = max(result, dfs(arr, d, i, memo));
        }

        return result;
    }

private:
    int dfs(vector<int>& arr, int d, int i, vector<int>& memo) {
        if (memo[i] != -1) return memo[i];

        int max_jumps = 1;

        for (int j = i + 1; j <= min(i + d, (int)arr.size() - 1)
             && arr[i] > arr[j]; ++j) {
            max_jumps = max(max_jumps, 1 + dfs(arr, d, j, memo));
        }

        for (int j = i - 1; j >= max(i - d, 0) && arr[i] > arr[j]; --j) {
            max_jumps = max(max_jumps, 1 + dfs(arr, d, j, memo));
        }

        memo[i] = max_jumps;
        return max_jumps;
    }
};
```

JUMP GAME PATTERN QUESTIONS

06.) JUMP GAME VI (LC-1696)

RECUSION + MEMOIZATION (TLE)

```
class Solution {
public:
    int maxResult(vector<int>& nums, int k) {
        int n = nums.size();
        vector<int> memo(n, INT_MIN);
        return dfs(nums, k, 0, memo);
    }

private:
    int dfs(vector<int>& nums, int k, int index, vector<int>& memo) {
        if (index == nums.size() - 1) {
            return nums[index];
        }

        if (memo[index] != INT_MIN) {
            return memo[index];
        }

        int maxScore = INT_MIN;
        for (int i = 1; i <= k && index + i < nums.size(); ++i) {
            maxScore = max(maxScore, nums[index] +
                           dfs(nums, k, index + i, memo));
        }

        memo[index] = maxScore;
        return maxScore;
    }
};
```

JUMP GAME PATTERN QUESTIONS

VIA DEQUE (SLIDING WINDOW MAXIMUM TECHNIQUE)

```cpp
class Solution {
public:
    int maxResult(vector<int>& nums, int k) {
        int n = nums.size();
        vector<int> dp(n, INT_MIN);
        dp[0] = nums[0];

        deque<int> dq;
        dq.push_back(0);

        for (int i = 1; i < n; ++i) {
            if (dq.front() < i - k) {
                dq.pop_front();
            }

            dp[i] = dp[dq.front()] + nums[i];

            while (!dq.empty() && dp[dq.back()] <= dp[i]) {
                dq.pop_back();
            }

            dq.push_back(i);
        }

        return dp[n - 1];
    }
};
```

JUMP GAME PATTERN QUESTIONS

07.) JUMP GAME VII (LC-1871)

```cpp
class Solution {
public:
    bool canReach(string s, int minJump, int maxJump) {
        int n=s.size();
        vector<bool> dp(n, false);

        dp[0]=true;
        int prev=0;

        for(int i=1;i<n;i++){
            if(i>=minJump)
            prev=prev+dp[i-minJump];

            if(i>maxJump)
            prev=prev-dp[i-maxJump-1];

            dp[i]=(s[i]=='0') && (prev>0);
        }
        return dp[n-1];
    }
};
```

THANK YOU !