Thoughts

Frog crossing – More on dynamic programming – 3

FEBRUARY 15, 2011 LEAVE A COMMENT (HTTPS://TKRAMESH.WORDPRESS.COM/2011/02/15/FROG-CROSSING-MORE-ON-DYNAMIC-PROGRAMMING-3/#RESPOND)

Question. There is a river n meters wide. A frog wants to cross the river. There were originally n-1 stones across the river at a distance of 1 meter from one another. However, some of these stones were removed recently.

The frog has a peculiar jumping rule. If it jumps x meters on one jump, the next jump has to lie in the range {x-1, x, x+1}. Further, the 1st jump that the frog makes is of exactly 1 meter. (We assume that the stone at a distance of 1 meter from the frog's end was not removed.)

Given which of the stones have been removed, how can you tell whether or not it is possible for the frog to reach the other end.

Solution.

Let us define an array, Stone [1 .. n] as:

Stone[i] = true, iff there is a stone at a distance of i meters from the frog's end.

Note that Stone[1] and Stone[n] are already given to be true.

——

We define another quantity, can_reach[d,s] as:

can_reach [d,s] = true,

if all of the following conditions hold true:

1. Stone[d] = true.

2. Stone[s] = true.

3. Given the previous jumps of the frog, the frog can reach the stone at distance d from the bank, by directly jumping from the stone at distance s from the bank.

——

Our aim is to figure out if can_reach[n,s] is true for any value of s i.e. for any of s = {1, …, n-1}.

—–

We make some observations about can-reach [d,s]:

1. can_reach [1,0] = true.

2. can_reach [d,0] = false, if d > 1.

3. can_reach [d,s] = false, if s >= d.

4. We also have the following recurrence:

If d > 1, can_reach [d,s] =

Stone[d] AND Stone[s] AND [can_reach [s, (d-s)-1] OR can_reach[s,d-s] OR can_reach[s,(d-s)+1]].

——-

Using the above equations, we can compute a table of can_reach values as follows:

——-

can_reach [1,0] = true;

For (d going from 2 to n)

can_reach [d,0] = false;

For (d going from 1 to n)

For (s going from d+1 to n)

can_reach[d,s] = false;

For (d going from 2 to n)

For (s going from 1 to d-1)

can_reach[d,s] is as given by equation (4) above.

//answer

For (s going from 1 to n-1)

If (can_reach[n,s] == true)

Output: "Can reach the other bank";

Output: "Cannot reach the other bank";

————

As can be seen, the algorithm runs in O(n^2) time.

FILED UNDER THEORETICAL COMPUTER SCIENCE TAGGED WITH DYNAMIC PROGRAMMING