Austin Row and Jeremy Fischer

Dr. Nayerri

CS 325

October 19, 2016

Group Assignment 2

1

|  |  |
| --- | --- |
| *1* | Jump(P[1…m], Q[1…n], L) |
| *2* | If distance between points P[m] and Q[n] > L |
| *3* | Return False |
| *4* | If m == 1 and n == 1 |
| *5* | Return True |
| *6* |  |
| *7* | If m > 1 and n > 1 |
| *8* | Return True if frogs can jump simultaneously to reach current position  (recursive call with m-1 and n-1) |
| *9* | If m > 1 |
| *10* | Return True if current position can be reached by Pfrog jumping and  Qfrog staying put (recursive call with m-1 and n) |
| *11* | If n > 1 and Cache[m][n] is not already set to true by a previous jump tactic |
| *12* | Return True if current position can be reached by Pfrog staying put  and Qfrog jumping (recursive call with m and n-1) |
| *13* |  |
| *14* | Return False |

**Proof of Correctness**

Claim:

*Jump* correctly returns true if a leash with length L can be used by Pfrog and Qfrog to traverse a pond on points in P[1…m] and Q[1…n] where Pfrog can only travel to points in P[1…m] and Qfrog can only travel to points in Q[1…n].

Base Case 1:

If the distance between the current locations of Pfrog and Qfrog is greater than L, *Jump* correctly returns false.

Base Case 2:

Let m be an arbitrary positive integer that represents the number of points in P[1…m]. Let n be an arbitrary positive integer that represents the number of points in Q[1…n]. If m is equal to one and n is equal to one, and the distance between the current locations of Pfrog and Qfrog is less than or equal to L, *Jump* correctly returns true.

Inductive Hypothesis:

Let M and N be positive integers such that M < m and N < n.

*Jump* correctly returns True for any sub-problem in which P = P[1…M] and/or Q = Q[1…N] and P and Q are traversable using a leash of length L. It correctly returns False for such a sub-problem in which P and Q are not traversable using a leash of length L.

Inductive Step:

If the distance between the current locations of Pfrog and Qfrog (P[m] and Q[n]) is greater than the leash length L, the correct value is returned by Base Case 1. If that is not the case, the distance must be less than or equal to L. If, then, m == 1 and n == 1, the correct value is returned by Base Case 2.

If m > 1 and n > 1 and the sub-problem using P[1…m-1] and Q[1…n-1] returns True (this sub-problem is solvable by the inductive hypothesis) *Jump* correctly returns True.

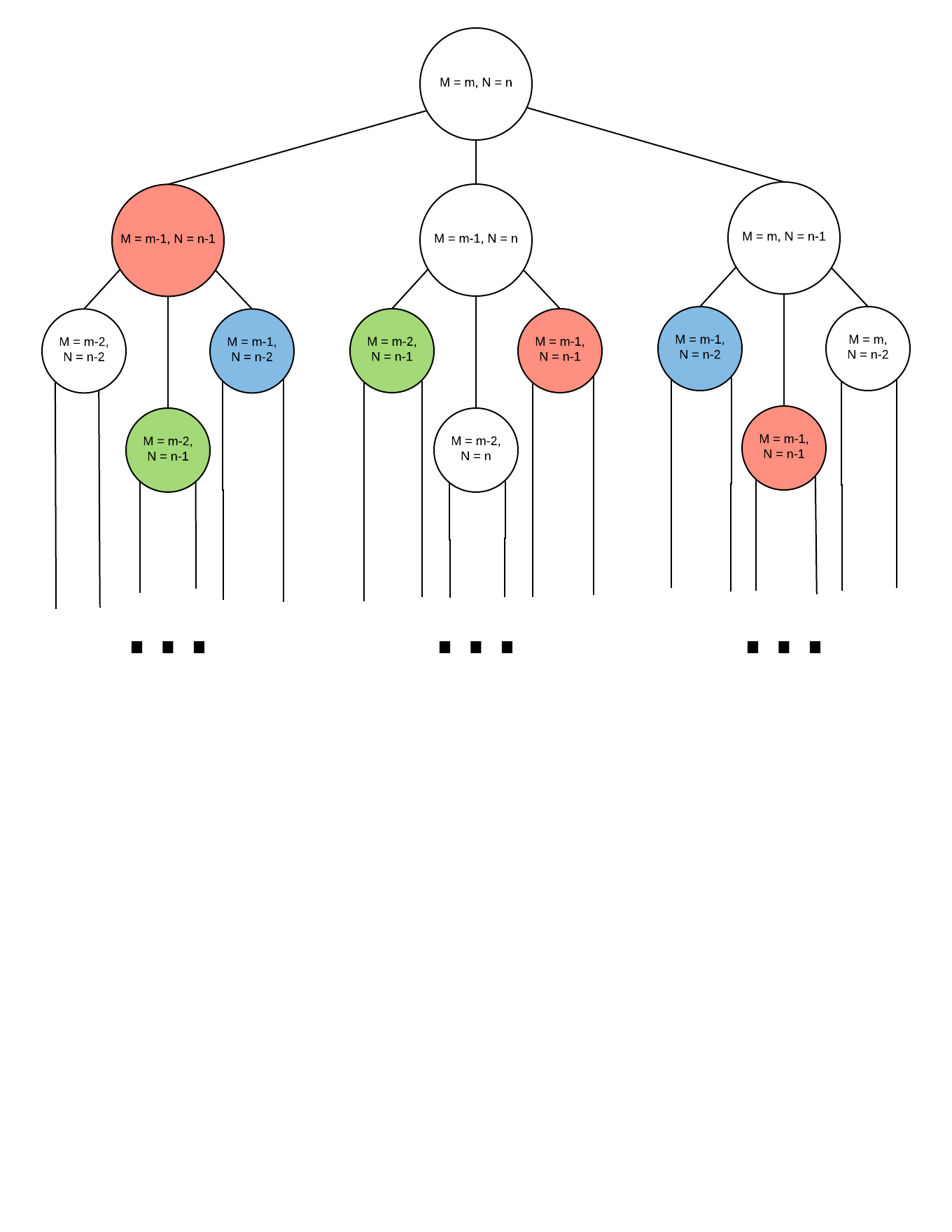
Otherwise if m > 1 and the sub-problem using P[1…m-1] and Q[1…n] returns True (this sub-problem is solvable by the inductive hypothesis) *Jump* correctly returns True.

Otherwise if n > 1 and the sub-problem using P[1…m] and Q[1…n-1] returns True (this sub-problem is solvable by the inductive hypothesis) *Jump* correctly returns True.

Otherwise, there is no way to traverse P[1…m] and Q[1…n] using a leash with length L and thus *Jump* correctly returns False.

2

The algorithm in part (1) is slow because it repeatedly solves redundant sub problems. The following call tree demonstrates how for only the first few levels of recursion, there are multiple instances of overlap where the algorithm has to solve the same problem more than once. In the call tree, ‘M’ refers to the number of points in P[1…M] and ‘N’ to the number of points in Q[1…N] for the current sub-problem, ‘m’ and ‘n’ are the total number of points in P and Q as a whole for the entire problem, and any nodes that share a color other than white represent a redundant sub-problem.



The speed of the algorithm in part one can be substantially improved by recalling the answers to sub-problems that have already been solved rather than solving these sub-problems again in a redundant fashion. The following improved algorithm does this using Memoization.

|  |  |
| --- | --- |
| *1* | Jump(P[1…m], Q[1…n], LeashLength, Cache[1…m][1…n]) |
| *2* | If Cache[m][n] already has the answer to this sub-problem stored |
| *3* | Return Cache[m][n] |
| *4* | If distance between points P[m] and Q[n] > LeashLength |
| *5* | Return False |
| *6* | If m == 1 and n == 1 |
| *7* | Cache[m][n] = True |
| *8* | Return Cache[m][n] |
| *9* |  |
| *10* | If m > 1 and n > 1 |
| *11* | Cache[m][n] = True if frogs can Jump simultaneously to reach current position  (recursive call with m-1 and n-1) |
| *12* | If m > 1 and Cache[m][n] is not already set to true by previous Jump tactic |
| *13* | Cache[m][n] = True if current position can be reached by Pfrog Jumping and  Qfrog staying put (recursive call with m-1 and n) |
| *14* | If n > 1 and Cache[m][n] is not already set to true by a previous Jump tactic |
| *15* | Cache[m][n] = True if current position can be reached by Pfrog staying put  and Qfrog Jumping (recursive call with m and n-1) |
| *16* | Return Cache[m][n] |

3

The memoized algorithm from part (2) can be converted into the following iterative solution:

|  |  |
| --- | --- |
| *1* | Jump(P[1…m], Q[1…n], LeashLength) |
| *2* | Cache = m by n array of booleans where Cache[i][j] for some 1 ≤ i ≤ m and 1 ≤ j ≤ n indicates if P[i] and Q[j] can be reached using the provided leash length |
| *3* |  |
| *4* | Cache[0][0] = True if distance between P[0] and Q[0] <= LeashLength, else False |
| *5* | If Cache[0][0] is False |
| *6* | Return False because the frogs’ starting points can’t even be connected by the  provided leash |
| *7* |  |
| *8* | For i = 2 to m |
| *9* | Cache[i][0] = True if distance between P[i] and Q[0] <= LeashLength and  Cache[i-1][0] is True, else False |
| *10* | For j = 2 to n |
| *11* | Cache[0][j] = True if distance between P[0] and Q[j] <= LeashLength and  Cache[0][j-1] is True, else False |
| *13* | For i = 3 to m |
| *14* | For j = 3 to n |
| *15* | If distance between P[i] and Q[j] <= LeashLength and (Cache[i-1][j] or  Cache[i][j-1] or Cache[i-1][j-1] are True) |
| *16* | Cache[i][j] = True |
| *17* | Else |
| *18* | Cache[i][j] = False |
| *19* |  |
| *20* | Return Cache[m][n] |

**Runtime Analysis**

Line 2 through 6: O(1)

Line 8 through 9: O(m)

Line 10 through 11: O(n)

Line 13 through 18: O(mn)

Line 20: O(1)

T(n,m) = mn + m + n + 1 = **O(mn)**

If the distance between P[1] and Q[1] > LeashLength, the algorithm will return False after having done a constant amount of work, thus T(n,m) = **Ω(1)**

Proof of Upper Bound:

Suppose there exists some constants *n0, m0*,and *c* such that for all n > *n0* and m > *m0*:

T(n,m) ≤ c(mn)

mn + m + n + 1 ≤ c(mn)

1 + 1/n + 1/m + 1/(mn) ≤ c

The values *c* = 4, *m0* = 1, and *n0* = 1 satisfy this inequality, thus T(n,m) = O(mn).

4

|  |  |
| --- | --- |
| *1* | JumpMultipleLengths(P[1…m], Q[1…n], LeashLengths[1…L]) |
| *2* | Sort LeashLengths[1…L] is ascending order |
| *3* |  |
| *4* | For i = 1 to L |
| *5* | If it’s possible to traverse P and Q using LeashLengths[i] (use algorithm from  part(3)) |
| *6* | Return L[i] |
| *7* |  |
| *8* | Return -1 (default value if no usable leash length is found) |