Data Structures and Algorithms

(CO 322)

Lab 2

Name : Jayathilaka H.A.D.T.T.

E.Number : E/16/156

Date: 23/07/2020

You're required to do the following:

(a) Using recursion (or otherwise) implement the minCost function.

(b) What is the runtime complexity of your implementation.

(c) Argue that dynamic programming can be used to improve the runtime.

(d) Use dynamic programming to improve the runtime.

(e) Calculate the runtime of your implementation in part 4 above. Assume, hashing is O(1).

(b) What is the runtime complexity of your implementation?

From (Station no.)	To (Station no.)	Number of inputs (n)	Complexity
0	1	1	1
0	2	2	3
0	3	3	9
0	4	4	27
0	5	5	81
0	6	6	243
0	7	7	729

By looking at the above results, we can get the run time complexity. Every result is a power of three. So, the result should be power of three. Here, **n** defines the number of inputs used in the algorithm. But, the number of inputs affects the result so much. In the code, the destination station is the input, which we change to see different runtime complexities. So the runtime complexity should be $3^{(n-1)}$.

$$\mathbf{T(n)= 3^{(n-1)}}$$

From big O notation complexity will be $\mathbf{O(3^n)}$

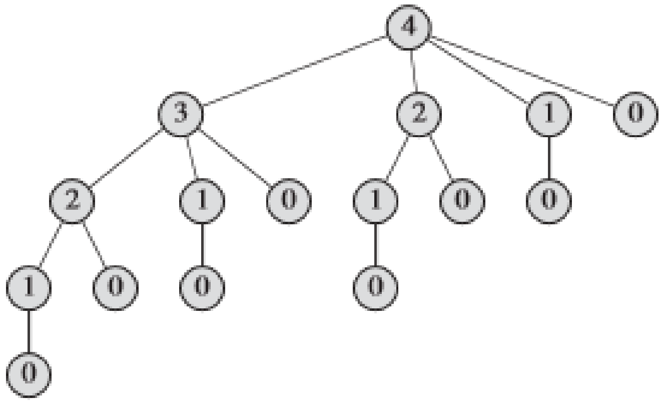
(c) Argue that dynamic programming can be used to improve the runtime.

Dynamic programming is a really useful general technique for solving problems that involves breaking down problems into smaller overlapping subproblems, storing the results computed from the subproblems and reusing those results on larger chunks of the problem.

Overlapping subproblem method has been used for the dynamic programming in task4.java. Having overlapping subproblems means we are computing the same problem more than once.

If we don’t have overlapping subproblems, nothing can stop us from caching values. It won’t actually improve our runtime at all. All it will do is create more work for us.

Here is the tree of all recursive calls required to get the minimum cost to go to the fourth station.



We can see station 1 to 2 is repeated twice, and 0 to 1 is repeated four times. Each of those repeats is an overlapping subproblem. We don’t need to compute those subproblems multiple times because the value won’t change. If we cache it, we can save a lot of run time. By that, we can say that dynamic programming can be used to improve the runtime.

Here are the results; we get without using dynamic programming.

From (Station no.)	To (Station no.)	Number of inputs (n)	Complexity
0	1	1	1
0	2	2	3
0	3	3	9
0	4	4	27
0	5	5	81
0	6	6	243
0	7	7	729

Here are the results, we get using dynamic programming.

From (Station no.)	To (Station no.)	Number of inputs (n)	Complexity
0	1	1	1
0	2	2	3
0	3	3	7
0	4	4	13
0	5	5	21
0	6	6	31
0	7	7	43

When we compare above two result tables we can clearly see that there is a huge difference between those complexities. So we can assume that second code which has been built using dynamic programming, has the minimum complexity.

By that we can prove, dynamic programming can be used to improve the runtime.

(e) Calculate the runtime of your implementation in part 4 above. Assume, hashing is O(1).

Let us look at the below table to get a better idea.

From (Station no.)	To (Station no.)	Number of inputs (n)	Complexity
0	1	1	1
0	2	2	3
0	3	3	7
0	4	4	13
0	5	5	21
0	6	6	31
0	7	7	43

So, complexity can be obtain by above data.

Complexity = $n^2 - n + 1$

So the time complexity of minCostDynamicPro function is **$O(n^2)$** .