

CO 322 Data Structures and Algorithms

Lab 02 - Dynamic Programming

E/16/388

Answers

4. Task

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

```
static int minCost(int i, int j)
{
    comp ++; //1 step
    if(j == i+1) return cost[i][j]; //2 steps
    int min = cost[i][j]; //1 step
    for(int k = i+1; k < j; k++) //iterate (j - 1) iterations
    {
        int tmp = minCost(i, k) + minCost(k, j);
        if(min > tmp) min = tmp;
    }
    return min; //1 step
}
```

For recursive approach consider $f(n)$ is the number of operations that the function needs to perform to solve $\text{minCost}(\text{int } i, \text{int } j)$ where $i = 1$ and $j = n$
 $\text{minCost}(0, 1)$ only has 3 steps.(base case consider the code above)

therefore $f(1) = c$ (c is a constant)

considering $\text{minCost}(\text{int } i, \text{int } j)$ general case,

$$T(n) = c + \sum_{k=0}^{n-1} (f(n-k) + f(k) + c)$$

$$T(n) = c + (T(n-1) + T(1) + c_1) + T(n-2) + T(2) + c_2 + T(n-3) + T(3) + c_3 + \dots \\ + (T(1) + T(n-1) + c_{n-1}) \text{-----}(A)$$

$$= c + 2(T(n-1) + \dots T(3) + T(2) + T(1)) + c_T \quad (c_T = c_1 + c_2 + \dots + c_{n-1}.)$$

Substituting $n = n - 1$, to $T(n)$ we can obtain $T(n - 1)$

$$T(n-1) = c + 2(T(n-2) + T(n-3) + \dots T(1)) + c_T'$$

By $T(n) - T(n-1)$

$$T(n) - T(n-1) = 2(T(n-1)) + c_T - c_T'$$

$$T(n) = 3(T(n-1)) + c_T - c_T'$$

Also by using similar procedure to above steps, $n = n - 2$

$$T(n-2) = 3(T(n-3)) + c'$$

By substituting these values for (A),

$$T(n) = 3^3 T(n-3) + c''$$

-

-

-

$$T(n) = 3^{n-1} T(1) + c'''$$

$$T(n) = \frac{1}{3} 3^n + c''' \quad (\text{All variables using } c \text{ are constants})$$

Hence, using $O()$ notation,

Time complexity is $O(3^n)$ for recursive approach. This is an exponential growth.

By observing the results below,

Complexity= 1 , Destination: 1 , Cost: 3

Complexity= 3 , Destination: 2 , Cost: 5

Complexity= 9 , Destination: 3 , Cost: 7

Complexity= 27 , Destination: 4 , Cost: 8

Complexity= 81 , Destination: 5 , Cost: 6

Complexity= 243 , Destination: 6 , Cost: 7

We can confirm its time complexity as $O(3^n)$.

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

Consider $\text{minCost}(\text{int } i, \text{int } j)$ as $f(i, j)$; i = start, j = destination station

Consider example, $f(0,4)$

Considering loop, for 1st iteration

$$F(0, 4) = f(0,1) + f(1,4)$$

$$F(1,4) = f(1,2) + f(2,4)$$

$$F(2,4) = f(2,3) + f(3,4)$$

We can directly get $f(i, j)$ when $j = i + 1$

Considering loop, for 2nd iteration

$$F(0, 4) = f(0,2) + f(2,4)$$

$$F(0,2) = f(0,1) + f(1,2) \qquad f(2,4) = f(2,3) + f(3,4)$$

In here we recalculated the same values that we calculated in the 1st iteration ($f(0,1)$, $f(1,2)$, $f(2,4)$, $f(2,3)$, $f(3,4)$)

But if we stored the values we calculated for each unique $f(i, j)$ in a data structure with a constant time complexity for data access, retrieve and other required operations(such as hashmap), when we need to calculate then again we can simply access those values from the data structure. Only if the value is not already in the data structure we can calculate that $f(i, j)$ and store it in the data structure for future use. Therefore directly accessing previously calculated values can reduce runtime complexity of the algorithm.

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

In dynamic program approach we store unique start and destination costs in a hashmap. Therefore we only need to calculate the values not currently in the hashmap. Therefore,

Runtime = number of unique subproblems \times time for each sub problem

For minCostDynamicPro(int i, int j) ; $0 \leq i < j$

Number of sub problems = n_{C_2}

$$\begin{aligned} &= \frac{n!}{2!(n-2)!} \\ &= \frac{n^2}{2} - \frac{n}{2} \end{aligned}$$

Hashing has $O(1)$ time complexity therefore assume time taken for each subproblem is constant.

Therefore $T(n)$

Therefore time complexity is $O(n^2)$.

Complexity= 1 , Destination: 1 , Cost: 3

Complexity= 3 , Destination: 2 , Cost: 5

Complexity= 7 , Destination: 3 , Cost: 7

Complexity= 13 , Destination: 4 , Cost: 8

Complexity= 21 , Destination: 5 , Cost: 6

Complexity= 31 , Destination: 6 , Cost: 7

Also according to this data we can see above time complexity behaviour of

$$T(n) = \frac{n^2}{2} - \frac{n}{2}$$