Assignment -3 (ALGORITHMS) P. Pradeep chakravarthy

2016114

① If the data is not uniformly distributed in the given range the time complexity will be definitely more than $O(n)$ and in worst case it may lead to $O(n^2)$ when every element is kept in a single bucket.

Memory complexity will be still $O(n)$.

Well we can improve the time complexity by doing mergesort instead of insertion sort in each bucket.

Pseudo Code:

```
Sor(A):
    n = A.len
  For i=0 to A.len -1
       B[i] -> making empty lists.

  For i=0 to n-1
      insert A[i] in B[fun(A[i])]  (Inserting elements into bucket)

  for i=0 to n-1
        sort(B[i]) -> [∵ sort using heapsort or merge sort]

  for i=0 to n-1
      concatenate B[i]
```

Time complexity = $O(n \log n)$

Space complexity = $O(n)$

(2) Algorithm for topological sort :

1. in[v] ⇒ number of vertices having an edge onto v.

first compute indegree of every vertex in graph.

2. After computing if any vertex has indegree as zero put them into ~~stack~~ que.

3. Remove top element of Q and then decrease the indegree of every vertex adjacent to this element. During this loop if any vertex indegree has become zero insert that vertex into Q. Store the removed vertex from Q in an array.

4. Repeat step-3 until the Q becomes empty.

5. The elements in the stored array is topological sort for the above graph (DAG).

Time-Complexity:

Step-1 : O(E)

Step-2 : O(V)

Step-~~3~~ 4 : O(E+V)

Step-5 : O(V)

∴ Time Complexity = O(V+E)

Scanned by CamScanner

For calculating shortest distance from a single source to any point in a graph we use algo called Bellmanford which uses DP and runs in $O(E \cdot V)$, but by applying topological sorting the time complexity becomes $O(V+E)$. ✓

Since if we use topological sorting in any iteration the calculated distance for every point in that iteration will be the shortest path from source.

3. memorize buttom up - DP algorithm :

func (p, n): → array containing costs of lengths.

let A[0, -- n] be new array

A[0] = 0 [∵ we cant cut if it is not there]

for i = 1 to n

temp = −∞

for j = 1 to i

temp = max (temp, ~~...~~ A[i-j] + P[j])

A[i] = temp

return A[n];

T(n)

T(n-1) + T(1)   T(n-2) + T(2)   ...

If T(n-1) is calculated then remaining all terms are already computed.

$\Rightarrow T(n) = T(n-1) + cn \rightarrow$ for loop which runs for n times.

$\Rightarrow T(n) = c\,\frac{n(n+1)}{2} = c\left(\frac{n^2+n}{2}\right)$

$\Rightarrow T(n) = \Theta(n^2)$

3. Given a rod of length n metres, cut the rope in different parts having different lengths such that it minimizes the total cost of buying a rod having length n. This is similar to above problem except we have to keep minimum in the for loop.

- let P[] → array having pi's. flag[n] → if zero → not selected, one → selected, two → removed.
  Mem[n] = {-1} → contains max answer upto that index of fields.

Maxpower (n) :

flag[n] = 1;

if n is taken into sum,

flag[n-1] = 2, flag[n+1] = 2;

~~...~~

for (i = 0; ... ; n)

if flag[i] == 0

flag[i] = 1

Maxpower (i);

~~[illegible]~~

$ans_1 = P[n] + \overset{Mem}{\cancel{max}}(n-3)$

if n is not include

$ans_2 = \overset{Mem}{\cancel{max}}(n-3) + maxpower(i)$

$Mem(n) = max(ans_1, ans_2)$

return Mem(n];

5.

```
for i=2 to N-1
    for j=1 to i-1
        if (A[i-1] == 1)
            k=1;
            l=j-1;
        else
            k=j+1;
            l=i-1;
        for k to l
            dp[i+1][k] += dp[i][j]
        }
    }
```

This can be further reduced to $O(n^2)$ if we slightly modify inner two loops. We can create another array which contains Prefix sums dp[i][j] & it can be used as answer in $O(1)$.