

2016/11/4/5

- 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:

for (A):

$n = A.length$

for $p = 0$ to $A.length - 1$

• $B[p] \rightarrow$ making empty list.

for $p = 0$ to $n - 1$

insert $A[i]$ in $B[\text{fun}(A[i])]$ (inserting elements into bucket)

for $p = 0$ to $n - 1$

sort($B[p]$) \rightarrow [sort using heapsort or mergesort]

for $p = 0$ to $n - 1$

concatenate $B[p]$

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

Space complexity = $O(n)$

② Algorithm for topological sort :

1. $\text{in}[v] \Rightarrow$ 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: $O(E + V)$

Step-5: $O(V)$

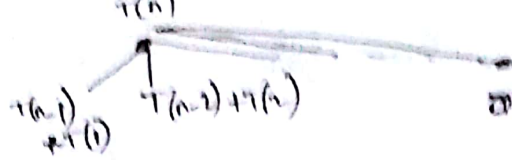
\therefore Time complexity = $O(V + E)$

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 both from source.

3. memorize bottom up-DP algorithm:
func (P, n):

let $A[0, \dots, n]$ be new array
 $A[0] = 0$ [∵ we cont out if it is not there]
for $i = 1$ to n
 $temp = -\infty$
 for $j = 1$ to i
 $temp = \max(temp, A[i-j] + P[j])$
 $A[i] = temp$
return $A[n]$;



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 $P[i] \rightarrow$ array having pp's. $flag[i] \rightarrow$ if zero \rightarrow not selected, one \rightarrow selected, two \rightarrow removed.
 $Hem[n] \rightarrow$ contains max ans upto that index of fibs.
 Maxpower(n):

$$flag[n] = 1;$$

if n is taken into sum,

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

~~max = P[n] + Hem[n-1]~~

for ($i=0; \dots; n$)

if $flag[i] == 0$

$$flag[i] = 1$$

$$Hem[n] = P[i]$$

~~return P[n]~~

$$\text{ans}_1 = P[n] + \text{Mem}(n-3)$$

if n is not include

$$\text{ans}_2 = \text{Mem}(n-3) + \text{manpower}(i)$$

$$\text{Mem}(n) = \max(\text{ans}_1, \text{ans}_2)$$

return Mem[n];

5. for $i=2$ to $N-1$
for $j=1$ to $i-1$
if $(A[i-1] == 1)$

$k=1;$

$l=i-1;$

else

$k=j+1;$

$l=i-1;$

for k to l

$dp[i+1][k] += dp[i][l]$
}

4

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