Name: Krishna Biswabarma

Roll: 24CS60R71

Tutorial 2: Algorithm and Design Analysis

Date: 05/09/2024.

① Algorithm:

```
countPaths(graph, s, g){ // s = Source Node, g = Sink Node.
    topOrder = topologicalSort(graph);
    dp[V] = {0} // Initialize dp vector to 0
    dp[g] = 1 // Trivial case of path from 'g' to 'g'.
    for each node in reverse(topOrder):
        for each neighbour u of v:
            dp[v] += dp[u]

    return dp[s]
}
```

Time Complexity: $TC = O(V+E) + O(V+E) \Rightarrow O(V+E)$

$O(V+E)$ ↓ Topological sort

$O(V+E)$ ↓ DP Calculation

•> **Working** -

$s \rightarrow 1 \rightarrow 3 \rightarrow g$

$s \rightarrow 2 \rightarrow 4 \rightarrow g$

After topological sort assume, s, 1, 2, 3, 4, g.

Starting from g and process in reverse topological order.

$dp[4] = dp[g] = 1$ // Edge from $4 \rightarrow g$

$dp[3] = dp[g] = 1$ // " " $3 \rightarrow g$

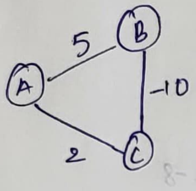
$dp[2] = dp[4] = 1$ // " " $2 \rightarrow 4$

$dp[1] = dp[3] = 1$ // " " $1 \rightarrow 3$.

$dp[s] = dp[1] + dp[2] = 1+1 \Rightarrow 2$ // There are edges from $s \rightarrow 1$ and s-

∴ No. of paths from $s \rightarrow g \Rightarrow dp[s] = 2$.

(2) Assume graph:



Dijkstra Working:

step1: A = 0, B = ∞, C = ∞

step2: A = 0, B = 5, C = 2

step3: B = min(5, 2 + (-10)) ⇒ -8

Now, since it is a cycle we will again visit B and relax the vertex and it will go on. So, it fails for negative edge weight.

• Bellman-Ford Algorithm –

Time Complexity: O(V*E).

```
bellman_ford(graph, s) {
    // Initialize all nodes distance as ∞.
    for each vertex V in graph {
        distance[v] ← ∞
        predecessor[v] ← null
    }
    distance[s] ← 0.
    // Relax edges V-1 times
    for (i=1 → V-1) {
        for each edges (u,v) in graph {
            if (distance[u] + weight(u,v) < distance[v]) {
                distance[v] ← distance[u] + weight(u,v)
                predecessor[v] ← u
            }
        }
    }
    // Check for -ve weight cycle.
    for each edge (u,v) in graph:
        if (distance[u] + weight(u,v) < distance[v]:
            return "-ve weight";

    return {distance, predecessor};
}
```

• Working:

$A = 0$, $B = \infty$, $C = \infty$

Iteration 1:

Relax A – B:

$B = \min(\infty, 0+5) = 5$

$A = \min(0, 5+5) = 0$

Relax A – C:

$C = \min(\infty, 0+2) = 2$

$A = \min(0, 2+2) = 0$

Relax B – C:

$C = \min(2, 5+(-10)) = -5$

$B = \min(5, -5+(-10)) = -8$

Distance: $A = 0$, $B = -8$, $C = -5$

Iteration 2:

Relax A – B:

$B = \min(-8, 0+5) = -8$

Relax A – C:

$C = \min(-5, 0+2) = -5$

Relax B – C:

$C = \min(-5, -8 + -10) = -5$

Distance:

$A = 0$, $B = -8$, $C = -5$

Hence bellman ford works for -ve weight edge cycle.

③ Algorithm:

Here, we can use two pointer approach:

```
pairwithsum (arr, sum){
    sort(arr);
    // Initializing pointers
    i=0, j= length(arr)-1;
    while (i<j) {
        current_sum = arr[i] + arr[j];
        if (current_sum == sum)
            return {arr[i], arr[j]};
        else if (current_sum < sum):
            i = i+1
        else
            j = j-1
    }
    return { } // No pair found.
}.
```

Time Complexity:

$O(n\log n) + O(n)$

$O(n\log n)$ → Sorting; $O(n)$ → 2-pointer approach.

$\therefore O(n\log n)$.

•) Working-

arr = {10, 15, 3, 7} , sum = 17.

// sort: {3, 7, 10, 15}.

step 1: i=0, j=3 current_sum = 3+15 ⇒ 18

step 2: i=0, j=2 current_sum = 3+10 ⇒ 13

step 3: i=1, j=2 current_sum = 7+10 ⇒ 17 → Return {7, 10}.

(4) Quick sort Algorithm:

```
quicksort (arr, low, high):
        if (low < high):
            pi = partition (arr, low, high)
            quick sort (arr, low, pi-1);
            quick sort (arr, pi+1, high);
        }
}

partition (arr, low, high){.
    pivot = arr[high]
    i = low - 1
    for (j=low to high-1){
        if arr[j] < pivot {
            i = i+1;
            swap (arr[i], arr[j]);
        }
    }
    swap (arr[i+1], arr[high])
    return i+1;
}
```

• Proving that Quick sort sorts the element correctly is:

We will use Mathematical Induction on the size of the array:

Assume QS(Quicksort) always work for all array of size < n i.e;

$1 \leq K \leq n$.

Now,

i) Partioning correctly arranges the element such that all elements less than pivot is on left and others are on right and pivot is in its perfect place.

ii) Now since the size of array is less than 'n' for the both ends.

Now, by our inductive hypothesis these are sorted correctly.

$\therefore$ Quicksorts sorts the array correctly.

TC: $O(n \log n)$