Assignment 4

Aryan Choudhary - 170152
Abhinav Sharma - 180017

Pseudo Code

Dfs(u,t,adj,PathCount)

```
if PathCount[u] has a value already assigned to it
    return
end if
for loop on all v in adj[u]
    DFS(v,t,adj,PathCount)
end for
if u equals to t
    PathCount[u] ← 1
else
    PathCount[u] ← 0
end if
for loop on all v in adj[u]
    w[u,v] ← PathCount[u]
    PathCount[u] ← PathCount[u] + PathCount[v]
end for
```

AssignWeights(s,t,G)

```
PathCount ← Empty array with unassigned values for each vertex in V.
adj ← Adjacency list of G
DFS(s,adj,PathCount)
Assign 0 weight to all edges with unassigned weights.
```

Definitions of various variable used.
G = Given DAG.
V or G.V = Set of vertices of G.
E or G.E = Set of edges of G.
w[u,v] denotes weight of $u \rightarrow v$ edge.
s = Start vertex. (As defined in question.)
t = End vertex. (As defined in question.)
PathCount[u] = No of paths from u to t.

Proof of Correctness

Let A be the set of all vertices reachable from s in the given DAG. The DFS function called in step 4 (in Assign-Weights function) starts from vertex s and reaches all the vertices reachable from s, (i.e. vertices in set A) and no other vertex.

1. **For all vertices i ∈ A, the order in which PathCount[i] is computed forms a reverse topological order.**
   For any vertex i ∈ A, PathCount[i] is computed before Dfs(i,t,adj,PathCount) returns (step 8-11). Consider any pair of vertices u,v such that (u → v) is an edge in the given DAG and when Dfs function reaches u. We need to show that PathCount[v] is computed before PathCount[u], or equivalently Dfs(v,t,adj,PathCount) returns before PathCount[u] is computed. For loop in step 4-6 iterates over all the vertices v for which (u→v) is an edge in G and step 5 completes when Dfs(v,t,adj,PathCount) exits. This is true for all v in adj[u]. After this, the for loop in step 12-14 computes PathCount[u]. Therefore, for any edge (u→v) , where u,v ∈ A, PathCount[u] is computed after PathCount[v].

2. **Value of PathCount[i] is computed for all vertices i ∈ A.**
   All the vertices in set A are defined to be the exhaustive set of vertices reachable from s. Also, vertices in A form a connected subgraph of the given graph G. In general, DFS algorithm starting from a vertex reaches all the vertices reachable from this starting vertex. Therefore we can say that DFS run starting from s reaches all the vertices in A. Equivalently, PathCount[i] is computed ∀ i ∈ A.

3. **For all i ∈ A, PathCount[i] contains the number of path from i to t after Dfs(i,t,adj,PathCount) returns.**
   We will prove this using induction. Consider the order of vertices in which their PathCount value is calculated. From lemma 1 and 2, we know that this order is one of the reverse topological order of the vertices in A. Let this reverse topological order be $(a_1, a_2, ..., a_{|A|})$.

   Applying induction on index of a vertex in the above formed reverse topological order-
   Base Case -
   Consider the first vertex (a_1) in this reverse topological order. This vertex must be the last vertex of some topological order of vertices ∈ A. Therefore , there does not exist any v in A, such that (a_1 →v) is an edge in G. If a_1 equals to t , then PathCount[a_1]=1 (step 8) else it is 0 (step 10). It is trivial that there can exist either only one path (if a_1 is t) or 0 path from a_1 to t, because there is no outgoing edges from a_1.

   Induction -
   Induction hypothesis - PathCount[a_k] contains the number of paths from a_k to t ∀ k < i.
   Case 1 → a_i = t :
   In this case ,we initialize PathCount[a_i] = 1 (step 8), since there is a path from t to t consisting of just one vertex in the path. For any vertex v such that (t→v) is an edge in G, PathCount[v] must be 0. We can use contradiction to prove this. Suppose PathCount[v] ≠ 0, this means there exist atleast one path from v to t in G. Also, since (t→v) is an edge in G , there also exist a path from t to v in G. Therefore there exist a cycle (t → v → ... → t) in G. This conradicts the fact that G is a DAG. Therefore, PathCount[v] must be 0. Hence, after this induction step, PathCount[t] remains 1, which is also obvious because there exist exactly one path from t to t.

   Case 2 → a_i ≠ t :
   In this case any path from a_i to t will contain more than one vertex in the path. Let this path be (a_i → v → ... → t), where v is a vertex such that (a_i → v) ∈ E. i.e. the second vertex in any path from a_i to t will always be v such that v∈ adj[a_i]. Index of v in reverse topological order shown above must be less than i, because there exist edge (a_i → v) ∈ E. Therefore, according to our induction hypothesis PathCount[u] is already computed ∀ u ∈ adj[a_i].
   PathCount[a_i] = $\sum_{v \in \text{adj}[a_i]}$number of path from v to t

or, PathCount[a_i] = $\sum_{v \in \text{adj}[a_i]}$ PathCount[v]. (Induction Hypothesis)

This computation is done in the step 14 of our algorithm.
This completes the proof that PathCount[i] contains the number of paths from i to t in G.

4. **For all i $\in$ A, all the paths from i to t have unique pathids from 0 to PathCount[i]-1 after Dfs(i,t,adj,PathCount) returns.**
We will use similar technique of induction (as used in lemma 3) to prove this lemma. Applying induction on index of a vertex in the above formed reverse topological order-

Base Case -
If a_1 = t, then PathCount[a_1] = 1 (from lemma 3) and since there is no edge in this path (path consists of only one vertex) pathid of this path is 0, i.e path with pathid equals to PathCount[a_1]-1. If $a_i \neq$ t, there does not exist any path from a_1 to t (using lemma 3). Hence it satisfies for all paths from a_1 to t.

Induction -
Induction hypothesis - For all k < i , the paths from a_k to t have unique pathids from 0 to PathCount[a_k]-1.

let adj[a_i] = (u_1, u_2,....,u_x) such that x is the outdegree of a_i. Since there exist an edge ($a_i \rightarrow u_j$)$\in$ E $\forall$ (1 $\leq$ j $\leq$ x), therefore $\forall$ (1 $\leq$ j $\leq$ x), index of u_j is less than i in the above reverse topological order. According to the induction hypothesis, all the paths from u_j to t ($\forall$ $1 \leq j \leq x$) have unique pathids in the range 0 to PathCount[u_j]-1. As per our proposed algorithm -
w[a_i,u_j] = $\sum_{l<j}$PathCount[u_l] $\forall$ j $\in$ [1 : x]
Pathid of any path from a_k to t passing through the edge ($a_k \rightarrow u_j$) can be written as (w[a_i,u_j] + pathid of corresponding path from u_j to t). Therefore, Paths from a_k to t passing through the edge ($a_k \rightarrow u_j$) have unique pathids in the range [w[a_i,u_j], w[a_i,u_j] + PathCount[u_j]-1].Let this Interval be I_j.
I_j = [$\sum_{l<j}$PathCount[u_l], $\sum_{l<j}$PathCount[u_l] + PathCount[u_j]-1].

Consider two intervals I_j and I_{j+1} for any j<x. Union of these two intervals give-

$$I_j + I_{j+1} = [\textstyle\sum_{l<j}\text{PathCount}[u_l], \sum_{l<j}\text{PathCount}[u_l] + \text{PathCount}[u_j] - 1] \cup [\sum_{l<j+1}\text{PathCount}[u_l], \sum_{l<j+1}\text{PathCount}[u_l] + \text{PathCount}[u_{j+1}] - 1]$$

$$I_j + I_{j+1} = [\textstyle\sum_{l<j}\text{PathCount}[u_l], \sum_{l<j}\text{PathCount}[u_l] + \text{PathCount}[u_j] - 1] \cup [\sum_{l<j}\text{PathCount}[u_l] + \text{PathCount}[u_{j+1}] - 1 + 1, \sum_{l<j}\text{PathCount}[u_l] + \text{PathCount}[u_{j+1}]-1]$$

$$I_j + I_{j+1} = [\textstyle\sum_{l<j}\text{PathCount}[u_l], \sum_{l<j+1}\text{PathCount}[u_l] + \text{PathCount}[u_{j+1}]-1]$$

In general, combining all the intervals I_j $\forall$ 1 $\leq$ j $\leq$ x, we get the interval I, such that-

I = $\cup_{r=1}^{x}$[$\sum_{l<r}$PathCount[u_l] , $\sum_{l<r}$PathCount[u_l] + PathCount[u_r] - 1]
I = [$\sum_{l<1}$PathCount[u_l] , $\sum_{l<x}$PathCount[u_l] + PathCount[u_x] - 1]
I = [0, $\sum_{l\leq x}$PathCount[u_l] - 1]
I = [0, PathCount[a_i]-1] - using proof in lemma 3

I is an interval of length PathCount[a_i]. The number of paths from a_i to t is also equal to PathCount[a_i]. Therefore, each path from a_i to t gets an unique pathid from the interval I i.e. from 0 to PathCount[a_i]-1.

Complexity Analysis

DFS across all calls takes $O(N+M)$ time, where N is no of vertices reachable from s and M is no of edges in subgraph of these N vertices. Since this sub graph is connected $\implies N \leq M$. Hence we can say DFS across all calls take $O(M)$. Since its a subgraph of given graph $M \leq |E|$. Hence, we can say DFS across all calls takes $O(|E|)$

Since we are supposed to assign weights to all edges. $O(|E|)$ is the lowerbound for any algorithm possible.

In AssignWeights function array creation on step 1 takes $O(|V|)$. Assignment in step 2 takes $O(1)$. Adjacency list in step 3 takes $O(|V|+|E|)$. Step 4 for dfs across all calls is already analysed before. Step 5 will take $O(|E|)$.

Overall this takes $O(|E|+|V|)$.