# Theory Assignment-4: ADA Winter-2024

Vivaswan Nawani (2021217) Animesh Pareek (2021131)

## 1 Preliminaries

- We are given a Graph $G$ and two vertices $s$ and $t$.
- We need to identify all the $(s, t)$ cut vertices in $G$. Here $G$ is an acyclic directed graph.
- A vertex $v \in V(G)$, where $v \neq s, t$, is $(s, t)$ cut if all paths from $s$ to $t$ pass through $v$.

## 2 Pre-Processing

- Construct a Graph $G^*$, which is the transpose Graph of $G$. In a transpose graph, all the edges of the original Graph are reversed.

## 3 Subproblem Definition

- We need only one subproblem definition.
- Paths(x, y, G) calculates the number of unique paths from vertex $x$ to vertex $y$.
- This subproblem is supposed to work in a manner such that it calculates and stores for each vertex $v \in V(G)$, the number of unique paths from Vertex $v$ to vertex $y$ if vertex $v$ is reachable via vertex $x$.
- Values are stored in an array of the form $(p_1, p_2, p_3, \ldots, p_v)$, where $v$ is the number of vertices in Graph and here, $p_i$ represents the number of unique paths that vertex $v_i$ was part of in all paths from $x$ to $y$.
- We perform a Depth First Search operation starting at $x$ to reach $y$ in this subproblem.

## 4 The specific subproblem(s) that solve the actual problem

- We need two specific subproblems to solve the actual problem.
- Subproblem 1: Paths($s$, $t$, $G$)
- Subproblem 2: Paths($t$, $s$, $G^*$)

## 5 Recurrence Relation

$$\text{Paths}(x, y, G) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq y \text{ and } y \text{ has no adjacent nodes} \\ \sum_{u \in \text{Adj}(x)} \text{Paths}(u, y, G) & \text{otherwise} \end{cases}$$

# 6 Algorithm Description

- Input: vertex $s$, vertex $t$ and Graph $G$.
- Build Graph $G^*$, which is the transpose of $G$ and has all of the edges of $G$ reversed.
- Define Function Path as given below:
- Path($x$, $y$, $G$, $DP$)
  - if($x == y$) return $DP[x] = 1$
  - else if($x \neq y$ and size($G.Adj(x)$) == 0) return $DP[x] = 0$
  - else if($DP[x] != -1$) return $DP[x]$
  - else
  - $sum = 0$
  - for($i \leftarrow 0$ to size($G.Adj(x)$) - 1):
  - $sum+ =$ Path($G.Adj(x)[i]$, $y$, $G$, $DP$)
  - return $DP[x] = sum$
- Initialize arrays $DP1$ and $DP2$ of size $V(G)$ with all values set to $-1$
- $total1 = \text{Path}(s, t, G, DP1)$
- $total2 = \text{Path}(t, s, G^*, DP2)$
- Initialize $DP$ of size $V(G)$ with all values set to 0
- for($i \leftarrow 0$ to $|V(G)| - 1$):
- if($DP1[i] == -1 || DP2[i] == -1$) $DP[i] = 0$
- else $DP[i] = DP1[i] \times DP2[i]$
- Initialize an empty set $Res$ to store the $(s, t)$ cut vertices of Graph $G$
- for($i \leftarrow 0$ to $|V(G)| - 1$):
- if($i \neq s$ and $i \neq t$ and $DP[i] == total1$) $Res$.add($i$)
- Output: $Res$

# 7 Complexity Analysis

- Building Graph $G^*$ takes $O(V + E)$ time as we need to access all the vertices and edges of the graphs given in the form of adjacency lists.
- Path($x$, $y$, $G$, $DP$) is called twice for $G$ and $G^*$, this function will take $O(V + E)$ time as well, as we are solving DFS style dynamic programming problem, where every node and every edge is accessed only once.
- Multiplication of $DP1$ and $DP2$ takes $O(V)$ time
- Finding the $(s, t)$ cut vertices in $DP$ also takes $O(V)$ time
- Therefore, overall the time complexity is $O(V + E) + 2 \times O(V + E) + 2 \times O(V)$ which is equal to $O(V + E)$
- Now, Since $|V(G)|$ = n and $|E(G)|$ = m. Therefore, overall the time complexity is equal to $O(n + m)$

# 8 Proof of Correctness

- Based on the Preliminaries, we can conclude that if, somehow, we can identify all paths from $s$ to $t$ and know the number of paths that each vertex is part of, then we can easily identify the vertices that are $(s, t)$ cut as they would be in the same number of paths as $s$ and $t$.
- Another observation would be that, for any vertex $v$ and Graph $G$,

  $$\text{Number of Paths from } s \text{ to } t \text{ including } v = \text{Number of Paths from } s \text{ to } v \times \text{Number of Paths from } v \text{ to } t.$$

- Therefore, our problem breaks down to finding the number of paths from $s$ to each vertex and finding the number of paths from each vertex to $t$.
- Let $G^*$ be the reversed graph of $G$, where every edge is reversed. Then,

  $$\text{Number of paths from } x \text{ to } y \text{ in } G = \text{Number of paths from } y \text{ to } x \text{ in } G^*,$$

  where $x, y \in V(G)$ and $V(G) = V(G^*)$.
- Therefore,

  $$\begin{aligned}&\text{Number of Paths from } s \text{ to } t \text{ including } v \text{ in } G\\ &= \text{Number of Paths from } v \text{ to } s \text{ in } G^* \times \text{Number of Paths from } v \text{ to } t \text{ in } G.\end{aligned}$$

- It's important to note that the number of paths to go from a vertex $x$ to a vertex $x$ is 1, and our algorithm description assigns the number of paths as $-1$ if there are no available paths between two vertices.
- If for any vertex, the number of paths from $v$ to $s$ in $G^*$ or Number of Paths from $v$ to $t$ in $G$ is $-1$, we assign the total number of paths as 0.
- Once we have the total number of paths for each vertex, we just need to store the vertices which have the same number of paths as the vertex $s$ and $t$.
- Hence Proved.