# Homework 10

Tushar Jain, N12753339
CSCI-GA 1170-003 - Fundamental Algorithms

November 10, 2017

**Problem 1.** The diameter of an undirected tree $T = (V, E)$ on $n$ vertices $V$ (and $(n-1)$ edges E) is the largest of all shortest paths distances in the tree: $D = max_{x,y \in V} \delta(x, y)$. You will design an $O(n)$ algorithm to compute D and will prove its correctness as follows.

**Problem 1.a.** Let $r$ be the root of $T$. Let $b$ is the furthest node from $r$ in $T$. Show that the diameter path in $T$ either ends or starts at $b$.

*Solution.* Let's us assume there is longer path between 2 vertices $u$ and $v$, neither of which is $b$. We observe that on the unique path between u and v, there must be some highest (closest to the root) vertex h. There are two possibilities:
1. the $u - v$ path intersects the path from the root to $b$ (at some vertex $x$, not necessarily at the u-v path's highest point h), and
2. it doesn't.

We show by method of contradiction that in both cases, the $u - v$ path can be made at least as long by replacing some path segment in it with a path to $b$.

For case 1: We know that $\delta(r, b) \geq \delta(r, u)$ as $b$ is the farthest most point from root, $r$. Since the highest point of the intersection of path $u - v$ and $r - b$ would be closer to the root than both $b$ and $u$. Therefore, $\delta(x, b) \geq \delta(x, u)$. Thus, replacing the $u - x$ part of the $u - v$ with $b - x$ would lead to longer path and would include $b$ as either starting or ending node.

For case 2: Let's assume $u$ to be further away from root, $r$, among the two nodes $u$ and $v$. But still $\delta(r, b) \geq \delta(r, u)$ as $b$ is the farthest most point from root, $r$. And thus, $\delta(r, b) \geq \delta(r, v)$ obviously. Since, the $u - v$ path and root to $b$ path do not intersect. We can find a node $x$(which would be the root itself in the extreme case) closer to the root which would be on both the $v - r$ and $b - r$ paths. Now, the distance $v - x$ and distance $x - b$ would be additional to the distance of $u - v$ if we consider the path $u - b$. Here again, $b$ is either at the end or the beginning of the diameter.

Thus, for all the cases, the condition that b is either at the end or the beginning of the diameter is maintained.

□

**Problem 1.b.** Assuming part (a), irrespective of whether or not you solved it, design an O(n) algorithm to compute $D$. For partial credit, give a slower algorithm.

*Solution.*

We can achieve this by adding an additional attribute, length, to every vertex which keeps track of length from current node/vertex to the farthest vertex 'b'. Then using DFS-VISIT from the $b$, we get diameter, i.e., the maximum shortest path between 2 vertices in the graph as we proved that diameter either starts/ends on the farthest vertex $b$.

**Pseudo-Code:**

```
def dfs_dia(G):
    color = ['white']*(len(G.Vertices))
    p = [None]*(len(G.Vertices))

    time = 0
    for u in range(len(G.V)):
        if color[u] == 'white':
            dfs_visit_dia(u,G, 0)

def dfs_visit_dia(u, G, l):
    time += 1
    d[u] = time
    length[u] = l
    color[u] = 'grey'
    for v in G[u]:
        if color[v] == 'white':
            p[v] = u
            dfs_visit_dia(v, G, l+1)
    color[u] = 'black'
    time +=1
    f[u] = time

def diameter(G):
    dfs_dia(G)
    v = argmax(length)
    dfs_visit_dia(G, v, 0)
    D = max(length)

    return D
```

*Running Time:* Total running time is $O(V+E) = O(n)$ as DFS-Visit takes $O(V+E)$ time and finding max, argmax takes $O(V)$.

□

**Problem 2.** An undirected graph is said to be connected if there is a path between any two vertices in the graph. Given a connected undirected graph G = (V, E), where V = 1, . . . , n, give an algorithm that runs in time $O(|V|+|E|)$ and finds a permutation $\pi : [n] \to [n]$ such that the subgraph of G induced by the vertices (1), . . . , (i) is connected for any $i \leq n$. Justify briefly the correctness and running time of your algorithm.

***Solution:***

This can be achieved by including nodes only after including their ancestors. A topological sorting is one way to solve. Here however, we're given an undirected connected graph and we can achieve the same by just accumulating new encountered nodes using a single DFS-visit

**Pseudo-Code:**

```
def dfs_visit_dia(u, G, A):
    time += 1
    d[u] = time
    A.append(u)
    color[u] = 'grey'
    for v in G[u]:
```

```
            if color[v] == 'white':
                p[v] = u
                dfs_visit_dia(v, G, l+1)
        color[u] = 'black'
        time +=1
        f[u] = time

        return A
```

$\square$

**Problem 3a.** Explain how a vertex $u$ of a directed graph can end up in a depth-first tree containing only $u$, although $u$ has both incoming and outgoing edges.

***Solution:***

This can happen because of the order of how we pick nodes to be part of the DFS forest. For instance:

*Example:* Consider a directed Graph
Vertices: a,b,c
Edges: [(a,b),(b,c)]
Order of visit: c, b, a

$\square$

**Problem 3b.** Assume $u$ is part of some directed cycle in $G$. Can $u$ still end up all by itself in the depth-first forest of $G$? Justify your answer.
**Hint:** Recall the White Path Theorem.

***Solution:***

Using the white Path Theorem, we know whenever a vertex in a directed cycle $u$ is discovered there will be a white-path to all other vertices in the cycle during a DFS traversal. Thus no vertex in the directed cycle can be in a DFS-tree containing only itself. This is because each vertex in the cycle will be in the same DFS-tree as they are descendants of some other vertex in the loop. This can however happen if a vertex in the graph has a self-loop and no other incoming or outgoing edges. The self-loop can be considered as a directed cycle. As this vertex doesnt have any other incoming or outgoing edges, it will form a DFS tree containing only itself. $\square$

**Problem 4.** Give a counterexample to the conjecture that if a directed graph G contains a path from $u$ to $v$, then any depth-first search must result in $v.d \leq u.f$.

***Solution:***

*Counter-example:*
Vertices: a,b,c
Edges: [(a,b),(a,c),(b,a)]

Let's assume we start our DFS at 'a' and then look at 'b' before 'c', here the $b.f = 3$ and the $c.d = 4$. In this case we do have a path from 'b' to 'c' however, $b.f \leq c.d$ which is a contradiction of the provided conjecture.

$\square$

**Problem 5.** Show that we can use a depth-first search of an undirected graph G to identify the connected components of G, and that the depth-first forest contains as many trees as G has connected components. More precisely, show how to modify depth-first search so that it assigns to each vertex v an integer label $v.cc$

between 1 and $k$, where $k$ is the number of connected components of G, such that $u.cc = v.cc$ if and only if u and v are in the same connected component. Your solution should also run in time $O(|V| + |E|)$. Briefly justify correctness and running time.

***Solution:***

Every time we encounter a white node in the original DFS, we update $cc$ count by 1 as we have found a new root of a tree in the forest. Also, in the recursive calls to $DFS - VISIT$, we simply label the current node being visited with the current $cc$ value. **Pseudo-Code:**

```
def dfs_cc(G):
    color = ['white']*(len(G.Vertices))
    p = [None]*(len(G.Vertices))

    time = 0
    cc = 0
    for u in range(len(G.V)):
        if color[u] == 'white':
            cc += 1
            dfs_visit_cc(u,G, cc)

def dfs_visit_cc(u, G, cc):
    u.cc = cc
    ...
    ...
    ... Same as normal DFS-VISIT
```

*Running Time:*
The total running time is the same as that of a normal DFS, i.e., $O(V + E)$ running time as only 2 lines which takes constant time have been added into the algorithm
*Correctness:* In, an undirected graph, all nodes of a connected components can be reached using only DFS-visit. Thus, whenever we start DFS-visit from a new node, we know it's going to be a a tree in DFS forest and thus, we increase the connected component account and keep the count constant for all descendants. □

**Problem 6.** Professor Jeff conjectures the following: let G = (V, E) be a connected, undirected graph with a real-valued weight function $w$ defined on $E$. Let $A$ be a subset of $E$ that is included in some minimum spanning tree for G, let $(S, V\backslash S)$ be any cut of G that respects $A$, and let $(u, v)$ be a safe edge for $A$ crossing $(S, V\backslash S)$. Then, $(u, v)$ is a light edge for the cut. Show that the professors conjecture is incorrect by giving a counterexample. Recall that we say a cut $(S, V\backslash S)$ respects a set of edges $A$ if no edge $(u, v) \in A$ has one node in S and one node in $V\backslash S$.

***Solution:***
*Counter-example:*
Vertices: a,b,c,d
Edges: [(a,b),(a,c),(c,d)]
Weights: [5,1,3]

Let A be the set $(a, c)$. Let $S = A$. S clearly respects A. Also, since G is a tree, its minimum spanning tree is itself, so A is trivially a subset of a minimum spanning tree. Also, every edge is safe. In particular, $(a, b)$ is safe but not a light edge for the cut. Therefore Professor Jeffs conjecture is false. □

**Problem 7.** Show that if an edge $(u, v)$ is contained in some minimum spanning tree, then it is a light edge crossing some cut of the graph.

***Solution:***
*Method of Contradiction:*

If edge $(u, v)$ is a bridge of a minimum spanning tree, and if we make a split at that edge, we would get 2 trees.

If we suppose there's an edge that has weight less than that of $(u, v)$ in this cut, it would imply that we can construct a minimum spanning tree of the whole graph using that edge which would create an MST with weight less than the original. This a contradiction as we assumed the original selection to be a Minimum Spanning Tree on the given graph. $\square$

**Problem 8.** Give a simple example of a connected graph such that the set of edges $(u, v)|$ there exists a cut $(S, V \backslash S)$ such that $(u, v)$ is a light edge crossing $(S, V \backslash S)$ does not form a minimum spanning tree.

***Solution:***
*Example:*
Vertices: a,b,c
Edges: [(a,b),(b,c),(c,a)] Weights: [1,1,1]

As the weights of all the edges are the same, each edge is a light edge for the a cut which it spans. But if we take all edges, we would get a cycle. $\square$