

Name: Yinsheng Dong
Student Number: 11148648
NSID: yid164
Assignment 1

1. Kleinberg and Tardos p. 109 #8

Proof by contradiction:

Assume to the contrary that there exists a positive natural number c so that for all connected graphs G , it is the case that $\text{diam}(G)/\text{apd}(G) > c$

We have that $\text{diam}(G) = \max\{\text{dist}(u, v) : u, v \in V\}$ and $\text{apd}(G) \leq \max\{\text{dist}(u, v) : u, v \in V\} = \text{diam}(G)$.

Since we've known that $\text{diam}(G) \neq \text{apd}(G)$, $\text{diam}(G)/\text{apd}(G) \neq 1$.

We got that $(\text{diam}(G))/(\text{apd}(G)) > 1$ which $c = 1$. Which contradict the problem given which is $\text{diam}(G)/\text{apd}(G) \leq c$.

We now proved that the statement is false.

2. Kleinberg and Tardos p. 110 #10

a. Solution:

To solve this problem, we need to compute the number of shortest paths from v to w .

First, we can perform a BFS from v , the algorithm designed:

```
BFS(u)
  Set Discovered[u] = true and Discovered[v] = false for the node we need to find
  Set the number of shortest path n = 0
  Initialize L[0] to consist of the single element u
  Set the layer counter i = 0;

  While L[i] is not empty
    for each node u is in L[i]
      if Discovered[v] = false then
        i+1
      else if Discovered[v] = true
        n = count L[i-1]
      endif
    end for
    i++
  end while
  return n
```

From his algorithm, we got that for each node v in L_1 , the complexity is 1, and for the at most the degree of L_n , we have the sum of the degrees in the $O(m)$, so the overall running time complexity is $O(n+m)$.

b. For all $n = 3q+1$ where q is a positive integer, there exists a graph with n vertices and two named vertices, u and v , that has $2^{(n-1)/3}$ shortest paths from u to v .

Direct Prove:

If we want to have multiple shortest paths from u to v in a graph G , G at least has 4 vertices which contains u and v , and it has at least 4 edges, and the shortest paths has 2.

We assume p is the number of shortest paths, so we have known $n = 3q+1$, $p = 2^{n-1/3}$ and q is the positive integer. We got $n \geq 4$ and $p \geq 2$.

Then using induction to prove.

Basic case: $q = 1$, then $n = 4$, $p = 2$, that shows the statement is true.

Inductive step: $q_1 = q+1 = 2$. Then $n_1 = n+3 = 7$, and $p_1 = 4$ which is also true.

To conclude, we got the statement is true.

3. In a directed graph $G = (V, E)$, a vertex v is called middle if and only if for every vertex x in V either there exists a directed path from v to x , or there exists a directed path from x to v .
 - a. Given a Directed acyclic graph G and a vertex u in V , provide an $O(|V|+|E|)$ time algorithm for determining whether or not vertex u is middle in G .

```
DFS(u) Algorithm to determine for (a)
  Mark u as "Explored" and add u to array visited[]
  for each edge (u, v) incident to u
    if v is not marked "Explored" and u == visited[0]
      mark v is Explored
      add v to visited[]
      call DFS(v)
    endif

    if v == visited[0]
      add u to visited[]
      call DFS[u]
    endif
  endfor

  if visited[] contains all element of graph g
    return true
  else
    return false
  endif
end Algorithm
```

Explanations:

I used an array to store the vertices that visited to check if it contains all vertices in the graph, so I used DFS to start from the given vertex on the graph, and store all the elements that it relates, then I compare all the graph items and the array items, to make sure if it is a mid-vertex or not, since it runs a time of all vertices and edges at most, so the complexity is $O(|V|+|E|)$

- b. Given an acyclic directed graph $G = (V, E)$ provide an $O(|V|+|E|)$ time algorithm for computing all the middle vertices of G .

```
Algorithm FindMidPoint for (b)
  i = 0
  Explored[] = false
  while (vi in g)
    if vi is not marked "Explored"
      mark vi Explored
      DFS(vi)
      i++
    endif
    if explored [vi] == true && DFS(vi) == false
      return -1
    else
      return vi
  endwhile
end Algorithm
```

This algorithm is based on the algorithm that I implement from problem (a), it firstly checks if the vertex has been explored or not, and it goes to BFS to check the vertex. Since each stop's complexity is $O(|V|+|E|)$, so the totally complexity is $O(V+E)$

- c. Given a general directed graph $G = (V, E)$ provide an $O(|V|+|E|)$ time algorithm for computing all the middle vertices of G .

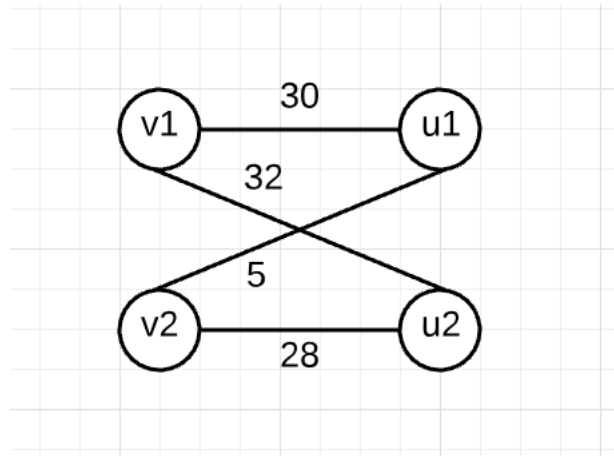
```

Alg. FindMidPoint(graph g) (for c)
  i = 0
  Explored[] = false
  while (vi in g)
    if explored[vi] == false
      DFS(vi)
      i++
    if explored[vi] == true && DFS(vi) == false
      return -1
    else
      return vi
    endif
  end while

```

This algorithm is similar like (b), and we need to avoid the circle graph, so when it does the recursive call of DFS, we need to think about the (u,v) and (v,u), when it comes, we consider that u is not the middle point or both u, v are midpoint. Since the algorithm is similar like b, and it also stops at a time of $O(|V|+|E|)$, so the complexity is also $O(|V|+|E|)$.

4. Let $G = (V \cup U, E)$ be a bipartite graph such that each edge $e \in E$ has an associated weight $w(e)$. A matching for G is a subset $M \subseteq E$ such that no two edges in M share a common vertex. The weight of M is $w(M) = \sum_{e \in M} w(e)$.
A greedy algorithm for bipartite matching could start with an empty matching M , and then repeatedly add the largest weight edge that does not share a vertex with an edge already included in M
 - a. Given an example edge weighted bipartite graph for which the above greedy algorithm will fail to find the maximum weight matching



Explain: $w(v1, u1) = 30$, $w(v1, u2) = 32$, $w(v2, u1) = 5$, $w(v2, u2) = 28$

In the situation above will make the greedy algorithm fail to find the maximum weight matching

When the greedy algorithm starts at $w(v1, u2) = 32$, then $w(v2, u1) = 5$,
We got the maximum weight from greedy = 37.

However, the best match should be $w(v1, u1) = 30$, $w(v2, u2) = 28$.

The max weight = $30 + 28 = 58$.

- b. For bipartite graphs in which all the edge weights are distance and each is power of w , prove that the above greedy algorithm always produces the maximum weight match.

Inductive Proof : Since the edge weights are distinct and each is a power of 2, and the greedy algorithm repeatedly add the largest weight edge.

Basic case: We assume there has 4 vertices in the graph, the largest weight edge is 2^n , and all other weight edge power $i < n$.

We know that $2^n + 2^1 > 2^{(n-1)} + 2^{(n-2)}$.

Inductive step: we add one more vertex and the largest weight edge is $2^{(n+1)}$, then we found that $2^{(n+1)} + 2^1 + 2^2 > 2^n + 2^{(n-1)} + 2^{(n-2)}$.

To conclude, we can get that the greedy algorithm picked the largest one.