CS 420
Section 001
Winter 2019
Mazen Alotaibi *(alotaima)* HW 1

Problem 1. The **square** of a directed graph $G = (V, E)$ is the graph $G^2 = (V, E^2)$, such that $(u, v) \in E^2$ if and only if G contains a path with two edges between u and v. Describe efficient algorithms for computing.

(a) the adjacency-list for G^2 from the adjacency-list of G.

(b) the adjacency-matrix of G^2 from the adjacency matrix of G.

Analyze the running times of your algorithms.

Solution:

(a) Please check algorithm 1 at the end of the document. Time Complexity:

$$O(V \times adj(V)^2) = O(V^3)$$

(b) Please check algorithm 2 at the end of the document. Time Complexity:

$$O(V^2 + V^3) = O(V^3)$$

□

Problem 2. A graph (V, E) is **bipartite** if the vertices V can be partitioned into two subsets L and R, such that every edge has one vertex in L and the other in R.

(a) Prove that every tree is a bipartite graph.

(b) Describe an efficient algorithm that determines whether a given undirected graph is bipartite.

(c) Time Complexity:

$$O((V + E) \times V) = O(V^2)$$

Solution:

(a) A graph is bipartite if and only if it doesn't contain an odd cycle. A tree doesn't contain any cycle. Therefore, a tree doesn't contain an odd cycle too. Thus, a tree is bipartite.

(b) Please check algorithm 3 at the end of the document.

(c) Time Complexity:

$$O(V + E)$$

□

Problem 3. In the **bottleneck-path problem**, you are given a graph G with edge weights, two vertices s and t and a particular weight W; your goal is to find a path from s to t in which every edge has at least weight W.

(a) Describe an efficient algorithm to solve this problem.

(b) What is the running time of your algorithm.

Solution:

(a) Breadth-First Search (BFS) with ignoring any weights that are less than W.

(b) Time Complexity:
$$O(V + E)$$

□

Problem 4. Longest Path in a DAG (LP-DAG)

(a) Describe an algorithm to find the longest path (measured in number of edges) in an unweighted DAG.

(b) What is the running time of the algorithm?

Solution:

(a) Please check algorithm 4 at the end of the document.

(b) Time Complexity:
$$O(V \times ((V + E) + V)) = O(V \times E)$$

□

Problem 5. Implement the algorithm you described in (4) in C, C++ or Python, name the programs lp-dag. Your program should read input from a file called graph.txt where the first line in the file is the number of vertices n (with $n \leq 100$) in the graph and the second line is the number of edges. Assume the vertices are numbered $1, 2, .., n$. The following lines each contains an edge. The program should output to the terminal the length of the longest path and the path itself.

Solution: Please check README.md to run lp-dag.py □

Algorithm 1 Problem 1.a

procedure SQUARELIST
    $G^2_{list} = G_{list}$
    for $v_i \ \in \ G_{list}$ **do**
        for $v_j \ \in \ adj(v_i)$ **do**
            for $v_k \ \in \ adj(v_j)$ **do**
                $G^2_{list}[adj(v_i)].append(v_k)$

Algorithm 2 Problem 1.b

procedure SQUAREMATRIX
    for $i \ \leq \ G_{matrix}.length$ **do**
        for $j \ \leq \ G_{matrix}.length$ **do**
            $G^2_{matrix}[i][j] = G_{matrix}[i][j]$
    for $i \ \leq \ G_{matrix}.length$ **do**
        for $j \ \leq \ G_{matrix}.length$ **do**
            for $k \ \leq \ G_{matrix}.length$ **do**
                if $G_{matrix}[i][j] == 1 \&\& G_{matrix}[i][j] == 1$ **then**
                    $G^2_{matrix}[i][j] = 1$

Algorithm 3 Problem 2.b

```
procedure isBipartite
    if G[0] ∉ visited then
        v = G[0]
        isBipartite(G, v)
    visited.append(v)
    L.append(v)
    v_s.value = 1
    for v_i ∈ adjList(v) do
        if G[0] ∉ visited then
            if v_i ∈ L then
                return False
            visited.append(v_i)
            R.append(v_i)
            v_i.value = 0
    v = R[0]
    for v_i ∈ adjList(v) do
        if G[0] ∉ visited then
            if v_i ∈ R then
                return False
            isBipartite(G, v_i)
    if checkIfAllVisited(G) then
        return True
```

Algorithm 4 Problem 4.a

```
procedure LPDAG
    all_paths_global = []
    max_len_global = 0
    max_paths_global = 0
    for i ∈ V do
        all_path = DFS(E_adjList, i)
        all_paths_global.append(paths)
        if all_paths not empty then
            max_len = 0
            for path ∈ all_paths do
                if path.length > max_len then
                    max_paths = path
                    max_len = path.length
            if max_len > max_len_global then
                max_len_global = max_len
                max_paths_global = max_paths
    return all_paths_global, max_len_global, max_paths_global
```