Sample Solution for Problem Set 9

Data Structures and Algorithms, Fall 2022

February 13, 2023

Contents

1 Problem 1

Using adjacency-matrix Let A denotes the adjacency matrix of G. Thus, we have the adjacency matrix of G^2 is A^2. Thus, the time complexity of this algorithm is equal to the time complexity of the matrix multiplication.

Using adjacency-list Let *Adj* denote the adjacency list of G. Thus, we can compute the adjacency list of G^2 as follow:

```
function compute-G2(Adj)
    Initialize an empty adjacency list Adj2
    for (u in V) {
        for (v in Adj[u]) {
            for (w in Adj[v])
                Adj2[u].add(v)
        }
    }
    return Adj2
```

Thus, the time complexity of the above algorithm is equal to $O(|V| \cdot |E|)$. Removing duplicate edges can be done in $O(|V| \cdot |E|)$. Thus, the total time complexity is $O(|V| \cdot |E|)$.

2 Problem 4

(a) Hint: graph induced by $E = \{(a,b),(a,c),(b,d),(c,e),(b,e),(c,d)\}$.

(b) Hint: graph induced by $E = \{(a,b),(b,a),(a,c)\}$.

(c) Hint: graph induced by $E = \{(a,b),(b,a),(b,c),(c,b),(a,c)\}$

3 Problem 6

let l be the minimum number of semesters necessary to complete the curriculum.

Observation 1: $l \geq \max\{\mathrm{dis}(u,v) : u, v \in V$ and u can reach $v\} + 1$, since for a fixed path $u \to v$, we can take at most one vertex each semesters.

Observation 2: $l \leq \max\{\mathrm{dis}(u,v) : u, v \in V$ and u can reach $v\} + 1$. Here is a greedy scheduling to achieve this bound: at each semester, select all courses with no prerequisite course, and delete these courses in G. Note that any longest path of G decreases its length by 1 each semester.

Algorithm 1 : The above process can be implemented with a modified version of BFS-based toposort. The following program provides a classic BFS-based toposort: the key observation here is that a vertex v is pushed into Q iff all vertices precedes v have been processed. However, here we need to process nodes ”layer by layer”. A possible way to implement this is using two queues. In a semester, instead of directly push all ”freed nodes”(nodes whose in-degree becomes 0 this semester) into original queue, we delay this push until next semester using an auxiliary queue. When the current queue is empty, we goto next semester and swap these two queues. Detailed implementation is left to readers.

```
function toposort-BFS(Node now):
    Compute in-degree of each vertex.
    Create a queue Q with all vertices with in-degree 0
    order = []
    while (Q is not empty) {
        Node u = Q.front();
        order.append(u);
        for all nodes v adjacent to u {
            in-degree[v] --;
            if (in-degree[v] == 0)
                Q.push(v);
        }
    }
```

Algorithm 2 : Actually we only need to compute the longest path in a DAG. An easier and classic way uses dynamic programming. Let f_u be the longest path from any vertex to u. For all nodes with in-degree 0, we have $f_u = 0$. For other nodes, $f_u = \max_{(v,u)\in E} f_v + 1$. However, we need to compute f in an appropriate order such that when u is taken into consideration, we have get all f_v for nodes v precedes u. It's easy to check that topological ordering satisfies this condition. We only need to compute it using methods teached in class and compute f one by one in topological ordering. Note that f can be calculated simultaneously when doing topo-sorting. Detailed implementation is left to readers.