United International University
CSI 227: Algorithms, Summer 2017

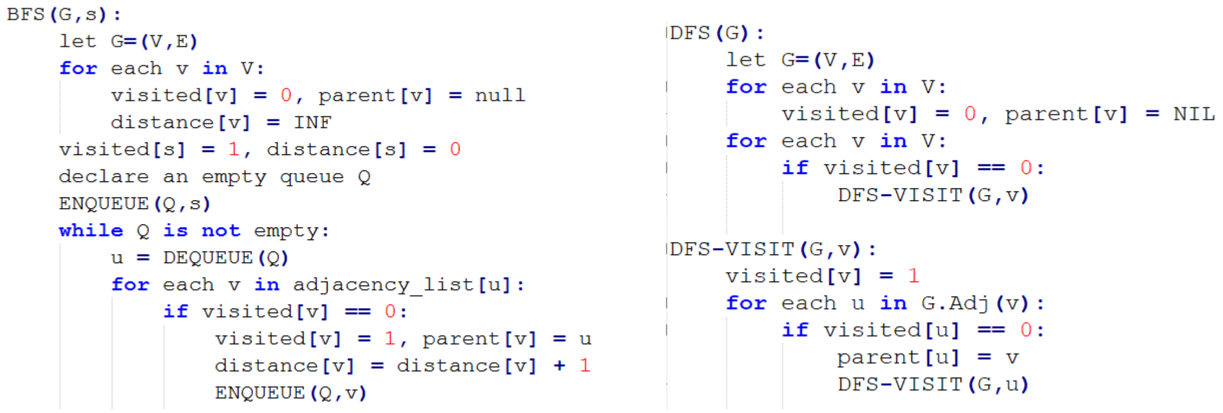
Final Exam

Total Marks: 120, Time: 2 hours

Answer any 6 out of the following 8 questions ($6 \times 20 = 120$)**.**

1. (a) Provide an algorithm that produces an **adjacency list** representation *Adj* from a provided **adjacency matrix** *Adj_Mat*. What is the running time of your algorithm? [8 + 2]

   (b) The following pseudo-code is of the **BFS** algorithm, which uses an adjacency list representation of the corresponding graph. Modify the algorithm such that it uses the **adjacency matrix** of the graph. Then derive the running-time of your modified algorithm, with explanation. [6 + 4]



```
BFS(G,s):
    let G=(V,E)
    for each v in V:
        visited[v] = 0, parent[v] = null
        distance[v] = INF
    visited[s] = 1, distance[s] = 0
    declare an empty queue Q
    ENQUEUE(Q,s)
    while Q is not empty:
        u = DEQUEUE(Q)
        for each v in adjacency_list[u]:
            if visited[v] == 0:
                visited[v] = 1, parent[v] = u
                distance[v] = distance[v] + 1
                ENQUEUE(Q,v)
```

(a) Q. 1(b)

```
DFS(G):
    let G=(V,E)
    for each v in V:
        visited[v] = 0, parent[v] = NIL
    for each v in V:
        if visited[v] == 0:
            DFS-VISIT(G,v)

DFS-VISIT(G,v):
    visited[v] = 1
    for each u in G.Adj(v):
        if visited[u] == 0:
            parent[u] = v
            DFS-VISIT(G,u)
```

(b) Q. 5(c)

Figure 1: The BFS and the DFS algorithms

2. (a) Explain with a clear example and simulation of the **Bellman-Ford algorithm** that how may it be used to detect negative cycles in a directed graph. Use a directed graph with 4 vertices and 5 edges. [10]

   (b) A Single-Source Shortest Paths (SSSP) algorithm in a graph determines the shortest paths from a fixed source vertex s to all the vertices in the graph. Consider a directed graph $G = (V, E)$. Suppose that you have fixed a destination vertex $t \in V$ in this graph. Using **just one execution** of an SSSP algorithm, how can you determine that which vertices in G have the shortest paths **towards** t? [6]

   (c) At the **Dijkstra's algorithm**, how many times do we have to search for a lowest estimated distance vertex? How many times is each edge relaxed? [4]

3. (a) State an advantage and a disadvantage of hashing with **Direct-Address tables**. [4]

   (b) For hash tables with **chained linked-lists**, why is it better to insert a data item at the beginning of a list, rather than anywhere else? [4]

   (c) Assume that the size of a data item and of a pointer is 100 and 4 units respectively. With **direct-address table** hashing with $m = 200$ slots and $n = 50$ data items, what is the total amount of memory required, if data items are stored - i) outside of the table; ii) directly at the table. [4 + 4]

   (d) Why is the load factor $\alpha \leq 1$ for the **open-addressing** hashing scheme? [4]

4. (a) Design a directed weighted graph $G = (V, E)$ where $V = \{s, t, u, v, w\}$ and every vertex has well defined shortest paths (path cost $\neq -\infty$) from s even if there is a negative weight cycle in the graph. [8]

(b) At the **Disjoint-Set Forests** data structure, what are the objectives of using the following heuristics: *union-by-rank* and *path-compression*? What if we do not use them? [7]

(c) Why are all the problems in the P class also in the NP class ($P \subseteq NP$)? Explain briefly. [5]

5. (a) What do you understand by **"collision"** in the context of hashing? At which hashing scheme, no collisions occur at all? [4]

(b) Why is finding a Hamiltonian Cycle in an undirected graph an *NP* problem? Explain briefly. [4]

(c) Modify the **DFS(v)** algorithm provided in Fig. 1(b) such that it can determine the size of the connected component that contains the vertex v. Using this modified algorithm, design an algorithm **MAX-COMPONENT-SIZE(n)** that finds out the maximum component size of the graph. [6+6]

6. (a) Find a **topological sort** for the graph Fig. 2 using **DFS**. You must present each of the recursion-trees generated by the algorithm. [7]

(b) For a hash table with m slots, what may be a problem if we use a hash function that always provides hash values, $h < m - 1$? What if the values are sometimes $\geq m$? Which one is the worse, and why? [2 + 2 + 4]

(c) Derive the running-time of the **Rabin-Karp algorithm** with explanation. [5]

7. (a) Does the problem of **finding a minimum element** from an array belong to the **NP** class? Explain briefly. [3]

(b) Let $G = (V, E)$ be a directed graph. Provide an algorithm to detect existence of a cycle in G using **Disjoint-Set Forests** data structure. Assume that you have other Disjoint-Set operations (MAKE-SET, FIND-SET, UNION) at your disposal. What is the run-time of your algorithm? [5+2]

(c) Working with modulo $q = 13$ and $d = |\Sigma| = 10$, find all valid matches and spurious hits (incorrect matches) in the text T = "41591851" for pattern P = "159", using the **Rabin-Karp algorithm**. [10]

8. (a) Does the **sorting problem** belong to the class **P**? Why or why not? [4]

(b) At the **Rabin-Karp algorithm** for string matching, the converted decimal numbers at various steps may become quite large to store in typical variables. What solution does this algorithm use to this problem? [3]

(c) The optimal substructure property of shortest paths states that – in a graph $G = (V, E)$, if the shortest path from a source vertex v_0 to a destination vertex v_k is $p = (v_0, v_1, ..., v_i, v_{i+1}, ..., v_{j-1}, v_j, ..., v_k)$, then the sub-path $(v_i, v_{i+1}, ..., v_{j-1}, v_j)$ included in p must be a shortest path from v_i to v_j. Why so? [5]

(d) Use the **Disjoint-Set Forests** data structure to find the connected components in the graph $G = (V, E)$ where $V = \{v_1, v_2, \ldots, v_9\}$ and $E = \{(v_1, v_2), (v_2, v_3), (v_3, v_1), (v_4, v_5), (v_5, v_6), (v_5, v_7), (v_6, v_4), (v_7, v_8), (v_5, v_8)\}$, using the *union-by-rank* and the *path-compression* heuristics. [8]