Foundations of Algorithms, Spring 2021: Homework 5

Due: Wednesday, April 14, 11:59pm

Problem 1 (6 points)

You are given the following bungled pseudo-code for the Breadth-First-Search algorithm.

```
BFS( G=(V,E), s )
    initialize distance array to infinity for every vertex
    initialize a queue and place starting node s in queue
    set distance[s] = 0
    while (queue not empty) do
        curr = dequeue from queue
        for every nbr of curr    # (curr, nbr) contained in E
            set distance[nbr] = distance[curr] + 1
            enqueue nbr in queue
```

Consider graphs G with 50 nodes, for which the designated starting node s can reach all other nodes in the graph.

1. When used on a first input graph with these characteristics, the code never terminates. Explain why.
2. When used on a second input graph with these characteristics, the code successfully completes.
   - Explain what must be true about the graph for this to happen.
   - Additionally, consider the values contained in the distance array. What do the values in the array represent? Explain.

Problem 2 (15 points: 10 for implementation / 5 for writeup)

Given are n cards, where n is an even, positive integer. Each card has two numbers written on it. Each number is an integer in the range $[1, n]$. Every number appears exactly twice in total.

Give an $O(n)$ algorithm to determine if it is possible to select $\frac{n}{2}$ cards such that each number appears exactly once on those cards.

For example, given $n = 6$ and cards: $(1, 5), (1, 4), (2, 4), (6, 3), (3, 6), (5, 2)$. Then the answer is yes, because the cards $(1, 5), (2, 4), (6, 3)$ can be selected such that all numbers appear exactly once.

On the other hand, given $n = 6$, and cards: $(1, 5), (2, 6), (1, 4), (4, 5), (3, 6), (3, 2)$. Then the answer is no, as there is no subset of 3 cards such that every number appears exactly once.

Problem 3 (20 points: 14 for implementation / 6 for writeup)

Given is a directed graph $G = (V, E)$, with $|V| = n$ and $|E| = m$. It is known that G is not strongly connected. Design an $O(m + n)$ algorithm that determines if it is possible to add just a single edge to the graph such that it becomes strongly connected.

Problem 4 (20 points: 14 for implementation / 6 for writeup)

Given is a connected, undirected graph $G = (V, E)$. Each edge $e_i \in E$ has an associated cost, c_i. The costs may not be distinct, however, it is guaranteed that for any cost, c, there are never more than two edges that have cost c. Design an $O(m \log n)$ algorithm that determines how many minimum spanning trees G has.

- *Connected* is usually associated with undirected graphs (two way edges): there is a **path** between every two nodes.
- *Strongly connected* is usually associated with directed graphs (one way edges): there is a **route** between every two nodes.
- *Complete graphs* are undirected graphs where there is an **edge** between every pair of nodes.