Problem Set 4

This problem set is divided into two parts: Part A problems are theory questions, and Part B problems are programming tasks.

Part A questions are due on Matt's Birthday, **Tuesday, April 6th** at **11:59PM**.

Part B questions are due **Thursday, April 8th** at **11:59PM**.

Solutions should be turned in through the course website in PDF form using LaTeX or scanned handwritten solutions.

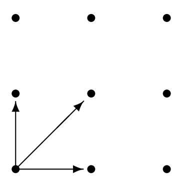
A template for writing up solutions in LaTeX is available on the course website.

Remember, your goal is to communicate. Full credit will be given only to the correct solution which is described clearly. Convoluted and obtuse descriptions might receive low marks, even when they are correct. Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

Part A: Due Tuesday, April 6th

1. **(20 points)** Reasoning about Search

   For each of the following statements, prove the statement or give a *small* counter example to show that it is false. You may use LaTeX to draw example graphs if necessary (the solution template contains a drawing of the following graph to get you started).



   (a) **(6 points)** A cyclic directed graph G has exactly one back edge produced by DFS. It is possible to remove one edge from G to make G acyclic.

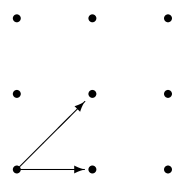
   Solution:

   True. As stated in 22.11 in CLRS, a graph is cyclic if and only if DFS yields at least one back edge. Since the back edge is not in the DFS tree, if we remove it, we will still have the same DFS tree, which will have no back edges. Thus, the new graph will be acyclic.

(b) **(6 points)** If a **directed** graph G has some vertices u and v in the same DFS tree and the DFS finishing time for v is after that of u then v is an ancestor of u in the DFS tree.

Solution:

False. Two children of the same parent must have one node with a DFS finishing time earlier than the other, but neither is an ancestor of the other. A counterexample is shown below.



(c) **(8 points)** A graph G with n vertices can have DFS produce a tree with depth $n-1$ while BFS on G will have depth of only 1. A root node is defined as having depth 0. (Note: if an example is difficult to draw with LaTeX vectors you may use text to describe it)

Solution:

True. Consider a "wheel" model with a central hub node with edges going to all nodes along the rim. Each rim node has an edge to the next node circularly. BFS starting with the central node will find all rim nodes with depth 1, while DFS from that central node will find all rim nodes sequentially with depth $n-1$.

2. **(14 points)** Bipartite graphs

An undirected graph is called *bipartite* if its nodes can each be assigned a color, either red or blue, such that no red node is adjacent to another red node, and no blue node is adjacent to another blue node. Give an efficient algorithm to determine if a graph is bipartite. What is its running time?

Hint: You may want to relate bipartiteness of a graph to the presence/absence of odd length cycles in it.

Solution: Use breadth-first search repeatedly (in case the graph is not connected). Assign the starting point to be red, those at odd depth from the starting point to be blue, and those at even depth to be red. When visiting a node and a neighbor has already been visited, check that they do not have the same color. If they do, return "graph is not bipartite". There can be no other configuration of the graph coloring which allows for bipartiteness because given only two colors, the colors on opposite ends of an edge must be different so coloring starting from a single source will always yield the same coloring. Starting at a different node will yield the same coloring as well because the graph is undirected. If all nodes are successfully colored, return "graph is bipartite". This runs in $O(V+E)$, the running time of BFS.

Argument using odd length cycles: In an odd length cycle, coloring with two alternating colors from some start node will end with the last node having the same color as the start node. Thus, any graph containing an odd cycle cannot be bipartite. The above algorithm uses BFS, so any node colored blue is an odd distance away from the starting point while any node colored red is an even distance away from the starting point. If an attempt is made to color a node both colors, this means it is both an odd and even distance from the source node, for an odd total number of nodes in the cycle passing through it, thus a bipartite coloring is impossible.

3. **(16 points)** Flight Planning

You are traveling an island nation with N cities. You start at city 1 at time 0 and need to get to city N no later than time T. There are M flights each of which flies from some city to some other city. All flights leave at an integer hour and arrive at an integer hour. That is, each flight i leaves some city a_i at an integer time t_{il} and arrives at another city b_i at integer time $t_{ia} > t_{il}$. Assume that if you arrive in a city at time t and there is a flight leaving at time t then you can make the transfer.

Given the times of the M flights, propose an algorithm based on BFS for determining whether there is a way to get from city 1 to city N in no more than time T. Your algorithm should run in $O(N \cdot T + M)$ time.

Solution:

Construct a graph $G = (V, E)$, where vertices $v = (a, t)$ represent a tuple denoting a particular city a and some time $t < T$. For each city and each time (hour) we have a corresponding vertex in the graph. Therefore in total we have $N \cdot T$ number of vertices. For every flight that leaves city a_i at time t_{il} and reaches city b_i at time t_{ia}, we add an edge from vertex (a_i, t_{il}) to (b_i, t_{ia}). There are M such edges. Also, for staying in a city when there are no flights available we add an edge from vertex (a, t) to vertex $(a, t + 1)$ for all cities a and time t. There are $N \cdot T$ such edges. Therefore graph G has $|V| = N \cdot T$ and $|E| = N \cdot T + M$. Now we can perform a BFS from start vertex $(1, 0)$ and find out if the node (N, T) is reachable. This runs in $O(V + E)$ time, i.e. $O(N \cdot T + M)$ time.