COMPSCI330 Design and Analysis of Algorithms
Assignment 4

Due Date: Wednesday, March 4, 2020

Guidelines

- **Describing Algorithms** If you are asked to provide an algorithm, you should clearly define each step of the procedure, establish its correctness, and then analyze its overall running time. There is no need to write pseudo-code; an unambiguous description of your algorithm in plain text will suffice. If the running time of your algorithm is worse than the suggested running time, you might receive partial credits.
- **Typesetting and Submission** Please submit the problems to GradeScope. You will be asked to **label your solution for individual problems**. Failing to label your solution can cost you 5% of the total points (3 points out of 60 for this homework).
- LaTeX is preferred, but answers typed with other software and converted to pdf is also accepted. Please make sure you submit to the correct problem, and your file can be opened by standard pdf reader. **Handwritten answers or pdf files that cannot be opened will not be graded.**
- **Timing** Please start early. The problems are difficult and they can take hours to solve. The time you spend on finding the proof can be much longer than the time to write. If you need more time for your homework please use this form and submit a STINF.
- **Collaboration Policy** Please check this page for the collaboration policy. You are **not** allowed to discuss homework problems in groups of more than 3 students. **Failure to adhere to these guidelines will be promptly reported to the relevant authority without exception.**

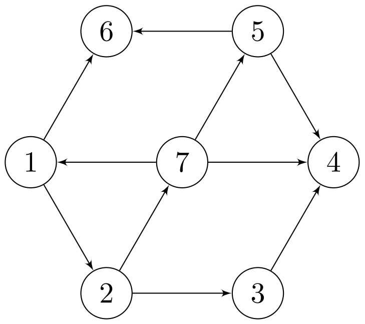
Problem 1 (DFS/BFS)**.** (15 points) Given the following graph, you are going to perform a Depth First Search.

When you have the option of choosing multiple vertices (in the main loop or when choosing the next vertex inside the recursive call), always choose the vertices with smaller indices first. That is, think of the main loop as

```
FOR i = 1 to 7
    DFS_visit(i)
```

For enumerating the edges, think of

```
For v = 1 to 7
    IF (u,v) is an edge and v is not visited THEN
        DFS_visit(v)
```

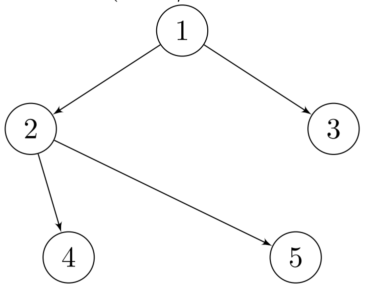


(a) (5 points) List the edges of DFS tree.

(b) (5 points) List the pre-order and post-order.

(c) (5 points) For all the non-tree edges, classify them into forward edges, back edges and cross edges.

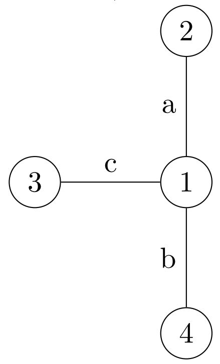
Problem 2 (Examples)**.** (15 points) Consider the following figure. The graph has is an directed graph with one edge between every pair of vertices. In the figure we have only drawn 4 edges that form a tree. The edges between (2, 3), (3, 4), (3, 5), (4, 5) were not drawn (and we don't know which directions they have). Complete the graph so that the current tree is both a DFS tree and a BFS tree (DFS/BFS starts at vertex 1, you are allowed to break ties arbitrarily).



(Extra: If the graph is undirected, connected and has at least n edges, is it possible for there to be a tree that is both a DFS tree and a BFS tree? Give an example or prove that such an example cannot exist. Feel free to think about this problem but you don't need to write the solution.)

Problem 3 (Turning right)**.** (30 points) Rong is driving in city X. City X has a very strange rule of driving - at every crossing, you can either go straight or turn right, but you cannot turn left.

The map of city X is given as an undirected graph, where intersections are vertices and roads are edges (all roads are two-way). There are n vertices and $m = \Theta(n)$ edges, where every vertex is adjacent to at most 4 edges. For each vertex (intersection), we also know the ordering of the edges in clockwise order. For example (see figure below), if vertex 1 is adjacent to 3 roads a, b, c, and the ordering is given as $a, \emptyset, b, c$, then that means road a is to the north of 1; there are no roads to the east of 1; b is to the south and c is to the west. If Rong enters vertex 1 from road a, he can go out using roads b or c (b means he goes straight and c means he turns right). If Rong enters vertex 1 from road b, he can only go out using road a (because turning right will be a dead-end $\emptyset$).



(a) (10 points) Given the map of X, a starting vertex s and an ending vertex t, design an algorithm to find a valid path to go from s to t (the specific starting/ending roads are not important, as long as they are connected to s and t respectively). Analyze the running time of your algorithm. Your algorithm should run in $O(n)$ time.

(b) (10 points) Now for every edge i, we have a weight w_i which represents the amount of time that it takes to drive from one end to the other. City X has synchronized traffic lights with period T, at time 0 to T (including 0 but not including T, in general, $[2kT, (2k+1)T)$ for $k = 0, 1, ...$), all the north/south directions have green lights; at time T to $2T$(including T but not $2T$, in general, $[(2k+1)T, (2k+2)T)$ for $k = 0, 1, ...$), all the east/west directions have green lights. The traffic rule is similar: at every intersection, Rong needs to wait for a green light if he decides to go straight; Rong does not need to wait if he decides to turn right. Design an algorithm to find the path that takes least time from s to t (again the specific starting/ending roads are not important, as long as they are connected to s and t respectively, we also don't need to consider the traffic lights at the two ends of the path). Analyze the running time of your algorithm. Your algorithm should run in $O(n \log n)$ time. (Hint: You can modify Dijkstra's algorithm.)

(c) (10 points) Prove the correctness for the algorithm you designed in (b). (Hint: If you modified Dijkstra for (b), you only need to explain what changes in the main step of Dijkstra's proof.)