CSCI-GA.1170-001/002 Fundamental Algorithms November 18, 2014

Solutions to Problem 1 of Homework 9 (18 Points)

Name: Keeyon Ebrahimi *Due: Wednesday, November 19*

You have an undirected graph $G = (V, E)$ and two special nodes $r, d \in V$. At time 0, node r is republican, node d is democratic, while all the other nodes $v \notin \{r, d\}$ are initially "undecided". For every $i = 1, 2, 3, \ldots$, the following 2-stage "conversion" process is performed at time time i. At the first stage, all republicans at time $(i-1)$ look at all their neighboring nodes v which are still undecided, and convert those undecided nodes to become republican. Similarly, at the second stage, all democratic nodes at time $(i-1)$ look at all their neighboring nodes v which are still undecided by the end of the first stage above, and convert those undecided nodes to become democratic. The process is repeated until no new conversions can be made. For example, if G is a 5-cycle $1, 2, 3, 4, 5$ where $r = 1$, $d = 5$, after time 1 node 2 becomes republican and node 4 becomes democratic, and after time 2 the last remaining node 3 becomes republican (as republicans move first). On the other hand, if the initial democratic node was $d = 3$ instead, then already after step 1 nodes 2 and 5 become republican, and node 4 becomes democratic, and no step 2 is needed.

Assume each node v have a field $v.color$, where *red* means republican, *blue* means democratic, and *white* means undecided, so that, at time 0, $r.color = red$, $d.color = blue$, and all other nodes v have $v.color = white$.

(a) (5 points) Using two BFS calls, show how to properly fill the final color of each node.

Solution:
The way to solve with two BFS calls is you first run a BFS with the republican node as your source and for each node save the distance as *Blue Distance*. You then run a BFS with the democrat node as your source, and have each node save the distance labeled *Red Distance*.

After that you compare every nodes *Red Distance* with their *Blue Distance*. If the *Red Distance* $<$ *Blue Distance*, you then change the node's color to Red. If the *Blue Distance* $\leq$ *Red Distance*, then you change the node's color to Blue.
The difference of having the republican color change when the distance is $\leq$ and the democrat color only change when the distance is $<$ is there because the republicans are being analyzed first as stated in the problem □

(b) (8 points) Show how speed up your procedure in part (a) by a factor of 2 (or more, depending on your implementation) by directly modifying the BFS procedure given in the book. Namely, instead of computing distances from the root node, you are computing the final colors of each node, by essentially performing a *single*, appropriately modified BFS traversal of G. Please write pseudocode, as it is *very* similar to the standard BFS pseudocode, and is much easier to grade. But briefly explain your code.

Solution:
First a brief explanation, then pseudo code. We first set the distance of the republican node

the democrat node to 0. We will run a modified BFS that will Enqueue all neighbors that are not the same color as itself. We will call the current node CN and the adjacent node AN. When comparing all of its adjacent nodes, if the adjacent color is white, progress as normal. If $AJ.color \neq White$, we run these checks

If

$$CN.Color == Blue \; and \; AN.Color == Red$$

$$and$$

$$CN.Distance + 1 \leq AN.Distance$$

then we change $AN.Color$ to blue, and we enqueue AN.

If

$$CN.Color == Red \; and \; AN.Color == Blue$$

$$and$$

$$CN.Distance + 1 < AN.Distance$$

then we change $AN.Color$ to red, and we enqueue AN.

The reason the change from blue to red happens when distance is $\leq$ and the red to blue only happens with distance is $<$ is because red is analyzed first. If the problem was changed to analyze blue first, we would change this.

This will give us a the correct color scheme with only one BFS search. Here is the Pseudo Code.

```
// r = Initial Red
// b = Initial Blue
BFS(G, r, b)
{
    for each vertex u in G.V - {r, b}
    {
        u.color = white
        u.d = infinity
        u.pi = NIL
    }

    r.color = Red
    r.d = 0
```

```
    r.pi = NIL

    b.color = Red
    b.d = 0
    b.pi = NIL

    Q = EmptySet
    ENQUEUE(Q, r)
}
ENQUEUE(Q, s)
{
    while Q != EmptySet
    {
        u = DEQUEUE(Q)
        for each v in G.adjacent[u]
        {
            if v.color == white
            {
                v.color = u.color
                v.d = u.d + 1
                v.pi = u
                ENQUEUE(Q, v)
            }

            if (u.color == Blue) and (v.color == Red)
            {
                if(u.d + 1 <= v.d)
                {
                    v.color = u.color
                    v.d = u.d + 1
                    v.pi = u
                    ENQUEUE(Q, v)
                }
            }

            if (u.color == Red) and (v.color == Blue)
            {
                if(u.d + 1 < v.d)
                {
                    v.color = u.color
                    v.d = u.d + 1
                    v.pi = u
                    ENQUEUE(Q, v)
                }
            }
```

```
        }
    }
}
```

□

(c) (5 points) Now assume that at time 0 more than one node could be republican or democratic. Namely, you are given as inputs some disjoint subsets R and D of V, where nodes in R are initially republican and nodes in D are initially democratic, but otherwise the conversion process is the same. For concreteness, assume $|R| = |D| = t$ for some $t \geq 1$ (so that parts (a) and (b) correspond to $t = 1$). Show how to generalize your solutions in parts (a) and (b) to this more general setting. Given parts (a) and (b) took time $O(|V| + |E|)$ (with different constants), how long would their modifications take as a function of t, $|V|$, $|E|$? Which procedure gives a faster solution?

Solution:
To generalize part(a), we would have to run a BFS on each node in set R and each node in set D. Then we would have to assign each nodes $Red.distance$ and $Blue.distance$ the the resulting minimum value that any of the BFS's found. Then we would compare each of the node's $Min(Red.distance)$ with $Min(Blue.distance)$ and determine color that way.

The running time of this algorithm would be $O(t(|V| + |E|))$ This is because we are running a BFS t times

To generalize part (b), the only difference is instead of only setting the initial red and blue node's color and distance. We will instead, initially go through the set D and set all distances to 0 and colors to B. We will then Go through set R and set all colors to red and distances to 0.

We then start the BFS giving it any single node from set R or set D and part (b) will solve this issue. This will not change the running time of part (b), so this algorithm will give us a running time of $O(|V| + |E|)$

Part (b) gives a much faster solution because part (b) only runs through one BFS while part (a) has to run a BFS for each item in set D and set R. □

CSCI-GA.1170-001/002 Fundamental Algorithms November 18, 2014

Solutions to Problem 2 of Homework 9 (5 Points)

Name: Keeyon Ebrahimi *Due: Wednesday, November 19*

Consider an $n \times n$ chessboard. In one move, a knight can go from position (i, j) to (k, ℓ) for $1 \leq i, j, k, \ell \leq n$ if either $|k - i| = 1$ and $|j - \ell| = 2$ or $|k - i| = 2$ and $|j - \ell| = 1$. However, a knight is not allowed to go to a square that is already occupied by a piece of the same color. You are given a starting position (s_x, s_y) and a desired final position (f_x, f_y) of a black knight and an array $B[1 \ldots n][1 \ldots n]$ such that $B[i][j] = 1$ if (i, j) is occupied by a black piece, and 0, otherwise. Give an $O(n^2)$ algorithm to find the smallest number of moves needed for the knight to reach from the starting position to the final position.

Solution:
To solve this, we will be running a modified BFS to find the shortest distance to (f_x, f_y). As for the modifications, each node will now have a (x, y) position. Then our BFS will be modified this way.

1. In our modified chess BFS, when we check adjacent nodes, we will instead be checking all nodes that pass $|k - i| = 1$ and $|j - \ell| = 2$ or $|k - i| = 2$ and $|j - \ell| = 1$. These are the only nodes that we can check as adjacent nodes.

2. In our modified chess BFS, when we would originally check if the node color is white, instead, we just check if that nodes position of $B[x][y] == 1$. If it is not equal to one, we then know that it is a possible spot for our knight to land, and we ENQUEUE this node

This will have the desired $O(n^2)$ running time. We know that the running time of a $BFS = O(|V| + |E|)$. In this chess example, we know that we have a total of n^2 V's, and each V can have at max 8 E's (Max amount of squares that pass $|k - i| = 1$ and $|j - \ell| = 2$ or $|k - i| = 2$ and $|j - \ell| = 1$ for an individual square). Now if we ignore constants and replace the $O(|V| + |E|)$ with our newly discovered n values, we will see our running time of $O(n^2)$. □

CSCI-GA.1170-001/002 Fundamental Algorithms November 18, 2014

Solutions to Problem 3 of Homework 9 (6 points)

Name: Keeyon Ebrahimi *Due: Wednesday, November 19*

An undirected graph is said to be conncected if there is a path between any two vertices in the graph. Given a connected undirected graph $G = (V, E)$, where $V = \{1, \ldots, n\}$, give an algorithm that runs in time $O(|V| + |E|)$ and finds a permutation $\pi : [n] \mapsto [n]$ such that the subgraph of G induced by the vertices $\{\pi(1), \ldots, \pi(i)\}$ is connected for any $i \leq n$. Which of BFS or DFS gives a better algorithm for this problem?

Solution:
This can be solved using a BFS or a DFS.
Subnote: For each below method, we just pick any random node in V as our source for each different type of search. This works because we are dealing with a connected undirected graph

DFS : We run a DFS using any node in the connected graph as our source. Whenever we visit a node and change its color to gray, which is the first step of $DFS - Visit$, we then append that node onto our permutation subgraph. This will make our permutation order nodes by their time when initially visited. By definition, all nodes at $t + 1$ can be visited from nodes at time t, hence that is how the DFS assigned those value, so we know our permutation is correct.

BFS : Very similar to the previous algorithm, we will run a BFS using any node in the connected graph as our source. When we visit a node and change it's color to gray, we will then append that value to our permutation order. This will then order our permutation by its distance to $\pi(1)$, which is the source. Because of how BFS works, we will know that later nodes in the permutation will have a route to earlier nodes in the permutation, because at worst case, they can traverse back to the source and then take the source's path to any other node in the permutation.

They both give permutations that fit the given restrictions needed, but the DFS gives more pertinent information as well. The DFS approach will closer mimic the actual traversal from one node to the next. In the DFS approach, elements right next to each other in the permutation will actually be close to each other in the graph, and we cannot say the same for the BFS approach. In the BFS permutation, $\pi(i)$ and $\pi(i+1)$ can have the same distance to the source node, but be on complete opposite sides of G, so even though they are neighbors in the permutation, they are not actual neighbors by any means in G. The DFS approach though will have permutation neighbors that will be visited at one time step after itself, making us know that they are actually close in G as well. □

CSCI-GA.1170-001/002 Fundamental Algorithms November 18, 2014

Solutions to Problem 4 of Homework 9 (8 points)

Name: Keeyon Ebrahimi *Due: Wednesday, November 19*

The class teacher of a kindergarten class wishes to divide the class of n children into two sections. She knows that some students pairs of students are friends with each other, and she wants to try to split the two sections in such a way that in each section all students are friends of each other. Can you help her find an efficient algorithm to form the two sections given as input n, and m statements of the form 'i and j are friends with each other'. What is the running time of your algorithm?

(**Hint**: Assume that the first student goes into the first section. Which section should the students who are friends of the first student go to? Which section should those that are not his friends go to? Try to carefully form a graph and use BFS to solve this problem.)

Solution: ********** INSERT PROBLEM 4 SOLUTION HERE ********** □

CSCI-GA.1170-001/002 Fundamental Algorithms November 18, 2014

Solutions to Problem 5 of Homework 9 (8 points)

Name: Keeyon Ebrahimi *Due: Wednesday, November 19*

(a) (4 points) Explain how a vertex u of a directed graph can end up in a depth-first tree containing only u, although u has both incoming and outgoing edges.

Solution: ********** INSERT PROBLEM 5a SOLUTION HERE **********
□

(b) (4 points) Assume u is part of some directed cycle in G. Can u still end up all by itself in the depth-first forest of G? Justify your answer.
(**Hint**: Recall the White Path Theorem.)

Solution: ********** INSERT PROBLEM 5b SOLUTION HERE **********
□