Problem 11-1 (Fast Route of a Knight) 5 Points

Consider an \( n \times n \) chessboard. In one move, a knight can go from position \((i, j)\) to \((k, \ell)\) 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.

Problem 11-2 (Dividing the Class into Sections) 8 points

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.)

Problem 11-3 (Party Propaganda) 18 Points

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 \( v \not\in \{r, d\} \) are initially “undecided”. For every \( i = 1, 2, 3, \ldots \), the following 2-stage “conversion” process is performed at time \( t = i \). 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.

(b) (8 points) Show how to 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.

(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?

Problem 11-4 (DFS in place of BFS) 12 points

Assume you want to compute shortest distances $\delta(v)$ from a source $s$ to all nodes $v$ of a graph $G$. Normally, you would run BFS($s$) and have $d(s) = \delta(s)$ at the end. Imagine that instead you run the following modification of DFS-Visit($v$):

- You initialize $d(s) = 0$ and $d(v) = \infty$ for all $v \neq s$.
- For every tree edge $(u, v)$ encountered by DFS-Visit, you set $d(v) = d(u) + 1$ (just like BFS).

(a) (1 point) What is the answer produced by this algorithm on a complete graph $K_n$, where $(u, v) \in E$ for all $(u, v)$?

(b) (2 points) Give an example of a graph $G$ with at most $2n$ edges, and an ordering of its edges in the adjacency list, where the algorithm above would still produce the same (bogus) answer as in part (a)?

(c) (2 points) Give an example of a graph $G$ with $\Omega(n^2)$ edges and $\delta(v) < \infty$ for all $v$, and an ordering of its edges in the adjacency list, where the algorithm above would not only produce a correct answer for all $n$ vertices $v$, but even produce exactly the same BFS tree as the BFS itself.

(d) (4 points) Recall, a forward edge $(u, v)$ connects $u$ to some (not immediate) descendant $v$ of $u$ in the DFS forest. For each of the following assertions, either prove it is correct, or give a counter-example.

- If the algorithm above is correct on all nodes $v$, then DFS-Visit($s$) did not encounter any forward edges.
- If the algorithm above is incorrect on at least one node $v$, then DFS-Visit($s$) encountered at least one forward edge.
(e) (3 points) Assume now we modify our procedure as follows: instead of setting \( d(v) = d(u) + 1 \) only for tree edges \((u, v)\) (color WHITE), we right away set \( d(v) = \min(d(v), d(u) + 1) \) for every discovered edge \((u, v)\) irrespective of the color of \(v\). Give an example of a graph \(G\), and an ordering of its edges in the adjacency list, where this algorithm is still wrong.