CS 4349.003 Homework 6

Due Wednesday October 16th on eLearning

October 9, 2019

Please answer the following **2** questions. The normal late penalties apply for this assignment.

1. A ***tournament*** is a directed graph with exactly one directed edge between any pair of vertices. That is, for any vertices v and w, a tournament contains either an edge $v \to w$ or an edge $w \to v$, but not both. A ***Hamiltonian path*** in a directed graph G is a directed path that visits every vertex of G exactly once.

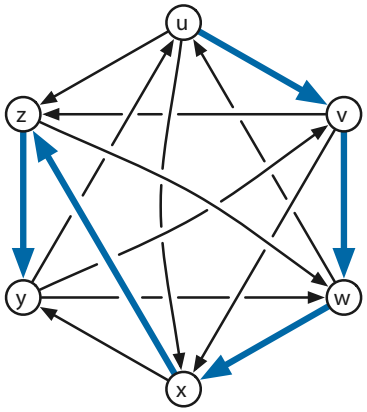

Figure 1. A tournament with a Hamiltonian path $u \to v \to w \to x \to z \to y$.

   (a) Prove that every tournament $G = (V, E)$ contains a Hamiltonian path. *[Hint: Pick an arbitrary vertex $v \in V$. Let V^- be the set of predecessors of v and let V^+ be the set of successors. What does induction guarantee about those two sets of vertices?]*

   (b) Describe and analyze an algorithm to compute a Hamiltonian path in a tournament G. Any output that reasonably describes the path or the order of the vertices along the path is fine. *[Hint: Your proof from part (a) probably describes an algorithm. What is its running time?]*

2. Let's use graph algorithms to solve a simple puzzle. Instances of this puzzle consist of an $n \times n$ grid of squares, where each square is labeled with a positive integer, and two tokens, one red and one blue. The tokens always lie on distinct squares of the grid. The tokens start in the top left and bottom right corners of the grid; the goal of the puzzle is to swap the tokens.

   In a single turn, you may move either token up, right, down, or left *by a distance determined by the* **other** *token*. For example, if the red token is on a square labeled 3, then you may move the blue token 3 steps up, 3 steps left, 3 steps right, or 3 steps down. However, you may not move either token off the grid, and at the end of a move the tokens cannot lie on the same square.

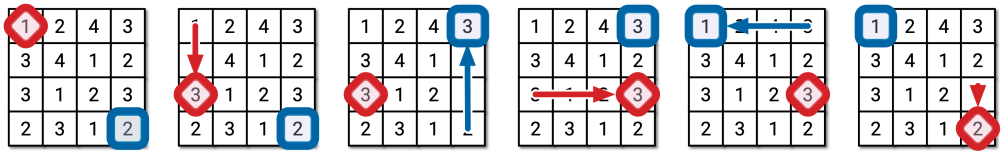

Figure 2. A five-move solution for a 4×4 instance of the puzzle.

Describe and analyze an efficient algorithm that correctly determines if the puzzle has a solution. If you reduce to a graph algorithm (which we highly recommend), then you should 1) clearly describe the graph you'll use, including the direction of its edge if directed; 2) specify the graph algorithm you'll use including any additional parameter it uses other than the graph itself; 3) specify how you'll use the output or results of the graph algorithm; and 4) express your running time in terms of n, the original input size.

[Hint: The description of configuration graphs in Erickson 5.3 may provide some inspiration.]