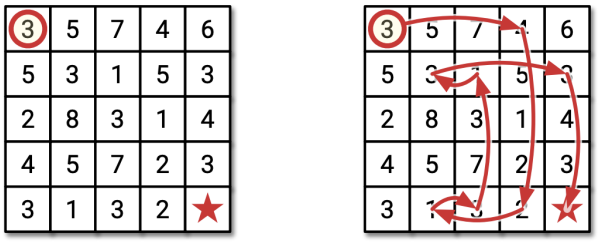
CSE 40113: Algorithms
Homework 5

You may complete this homework in groups of 3 or less students. Note that the integrity policy applies: your group should write up your own work, although you're welcome to work on the problems in a larger group. If you have any questions, please re-read both the homework guidelines and the academic integrity policy carefully, and then come discuss any questions or concerns with me.

Required Problems

1. A number maze is an $n \times n$ grid of positive integers. A token starts in the upper left corner; your goal is to move the token to the lower-right corner. On each turn, you are allowed to move the token up, down, left, or right; the distance you may move the token is determined by the number on its current square. For example, if the token is on a square labeled 3, then you may move the token three steps up, three steps down, three steps left, or three steps right. However, you are never allowed to move the token off the edge of the board.

   Describe and analyze an efficient algorithm that either returns the minimum number of moves required to solve a given number maze, or correctly reports that the maze has no solution. For example, given the number maze below, your algorithm should return the integer 8.



2. There's a natural (and largely correct!) intuition that two vertices which are far apart in a graph somehow have a more tenuous connection compared to two nodes that are close together. In fact, there are a bunch of algorithmic results that try to make this intuition precise in a variety of ways. In this problem, I want you to consider one of them: namely, how many vertex removals could disconnect the two vertices, so that they cannot reach each other. (This has some obvious implications for reliability of network communication robustness, if you stop to think about it.)

   Suppose that we have a graph $G = (V, E)$, and fix two nodes u and v. Furthermore, suppose that the distance between u and v is strictly greater than $V/2$. Prove that there must exist some other node x (which is not equal to u or to v) whose deletion destroys all u to v paths in the graph. Then, give an algorithm to find such a node (as quickly as possible).

3. Suppose we are given a directed acyclic graph G whose nodes represent jobs and whose edges represent precedence constraints; that is, each edge uv indicates the job u must be completed before job v begins. Each node v also has a weight $T(v)$ indicating the time required to execute job v. Note that in this problem, jobs can run in parallel.

   (a) Describe an algorithm to determine the shortest interval of time in which all jobs in G can be executed.

   (b) Suppose the first job starts at time 0. Describe an algorithm to determine, for each vertex v, the earliest time when job v can begin.

   (c) Now describe an algorithm to determine, for each vertex v, the latest time when job v can begin without violating the precedence constraints or increasing the overall completion time (computed in part (a)), assuming that every job except v starts at its earliest start time (computed in part (b)).

4. Sample Solved problem:

   Some friends of yours are working on techniques for coordinating groups of mobile robots. Each robot has a radio transmitter that it uses to communicate with a base station, and your friends find that if the robots get too close to one another, then there are problems with interference among the transmitters. So a natural problem arises: how to plan the motion of the robots in such a way that each robot gets to its intended destination, but in the process the robots don't come close enough together to cause interference problems.

   We can model this problem abstractly as follows. Suppose that we have an undirected graph $G = (V, E)$, representing the floor plan of a building, and there are two robots initially located at nodes a and b in the graph. The robot at node a wants to travel to node c along a path in G, and the robot at node b wants to travel to node d. This is accomplished by means of a schedule: at each time step, the schedule specifies that one of the robots moves across a single edge, from one node to a neighboring node; at the end of the schedule, the robot from node a should be sitting on c, and the robot from b should be sitting on d.

   A schedule is interference-free if there is no point at which the two.robots occupy nodes that are at a distance $< r$ from one another in the graph, for a given parameter r. We'll assume that the two starting nodes a and b are at a distance greater than r, and so are the two ending nodes c and d.

   Give a polynomial-time algorithm that decides whether there exists an interference-free schedule by which each robot can get to its destination.

   Solution: This is a problem of the following general flavor. We have a set of possible configurations for the robots, where we define a configuration to be a choice of location for each one. We are trying to get from a given starting configuration (a, b) to a given ending configuration (c, d), subject to constraints on how we can move between configurations (we can only change one robot's location to a neighboring node), and also subject to constraints on which configurations are "legal."

   This problem can be tricky to think about if we view things at the level of the underlying graph G: for a given configuration of the robots–that is, the current location of each one–it's not clear what rule we should be using to decide how to move one of the robots next. So instead we apply an idea that can be very useful for situations in which we're trying to perform this type of search. We observe that our problem looks a lot like a path-finding problem, not in the original graph G but in the space of all possible configurations.

   So, we will define the following (larger) graph H: The node set of H is the set of all possible configurations of the robots; that is, H consists of all possible pairs of nodes in G. We join two nodes of H by an edge if they represent configurations that could be consecutive in a schedule; that is, (u, v) and (u', v') will be joined by an edge in H if one of the pairs u, u' or v, v' are equal, and the other pair corresponds to an edge in G.

   We can already observe that paths in H from (a, b) to (c, d) correspond to schedules for the robots: such a path consists precisely of a sequence of configurations in which, at each step, one robot crosses a single edge in G. However, we have not yet encoded the notion that the schedule should be interference-free.

   To do this, we simply delete from H all nodes that correspond to configurations in which there would be interference. Thus we define H' to be the graph obtained from H by deleting all nodes (u, v) for which the distance between u and v in G is at most r.

The full algorithm is then as follows. We construct the graph H', and then run the DFS/BFS algorithm from the text to determine whether there is a path from (a, b) to (c, d). The correctness of the algorithm follows from the fact that paths in H' correspond to schedules, and the nodes in H' correspond precisely to the configurations in which there is no interference.

Finally, we need to consider the running time. Let V denote the number of nodes in G, and E denote the number of edges in G. We'll analyze the running time by doing three things: (1) bounding the size of H' (which will in general be larger than G), (2) bounding the time it takes to construct H', and (3) bounding the time it takes to search for a path from (a, b) to (c, d) in H'.

First, then, let's consider the size of H'. H' has at most V^2 nodes, since its nodes correspond to pairs of nodes in G. Now, how many edges does H' have? A node (u, v) will have edges to (u', v) for each neighbor u' of u in G, and to (u, v') for each neighbor v' of v in G. A simple upper bound says that there can be at most V choices for (u', u), and at most V choices for (u, v'), so there are at most $2V$ edges incident to each node of H'. Summing over the (at most) V^2 nodes of H', this would give $O(V^3)$ edges. However, we can also note that u has exactly $d(u)$ neighbors, and v has exactly $d(v)$. Using the degree sum formula, we can slightly improve our total number of edges to $O(VE)$ total.

Second, we bound the time needed to construct H'. We first build H by enumerating all pairs of nodes in G in time $O(V^2)$, and constructing edges using the definition above in time $O(V)$ per node, for a total of $O(V^3)$. Next, we must delete nodes from H so as to produce H'. We can do this as follows: For each node $u \in G$, we run a breadth-first search from u and identify all nodes v within distance r of u. We list all these pairs (u, v) and delete them from H. Each breadth-first search in G takes time $O(V + E)$, and we're doing one from each node, so the total time for this part is $O(V(V + E))$.

Finally, we have H', and so we just need to decide whether there is a path from (a, b) to (c, d). This can be done using the connectivity algorithm from the text in time that is linear in the number of nodes and edges of H'. Since H' has $O(V^2)$ nodes and $O(VE)$ edges, this final step takes $O(V^2 + VE)$.