**Question 1** (Airport placement, 35 points). The nation of Graphia, has a number of cities, some pairs of which are connected by roads (all of which are two-way). Unfortunately, due to various hazardous terrain, not all cities are reachable from all other cities. There is a plan to fix this problem by building airports in some number of the cities. The requirement is that it must be possible to reach at least one airport from any starting city. Furthermore, each city has an airport cost which is a positive integer denoting the cost of building an airport in that city.

(a) Find a linear time algorithm that given the airport costs and the connections made by the roads, returns the cheapest collection of cities to build airports in so that every city will be reachable from at least one airport. Show that your algorithm is correct and runs in an appropriate amount of time. [30 points]

(b) What goes wrong with this algorithm if some of the roads are one-way? [5 points]

**Solution 1.** (a) It is straightforward to convert the given problem into a graph problem: Each city is a vertex and and each road is an undirected edge. Now we need to build airports at some of the vertices such that the graph becomes connected and the cost is minimized.

**Algorithm:**

- If the graph is already connected
  - No need to build any airports
- Else
  - Using DFS, decompose the graph into connected components
  - For each connected component, build an airport at the vertex with the least cost

**Time Complexity:** Finding connected components takes $O(|V| + |E|)$ and finding the least cost vertex in all connected components takes $O(|V|)$. Thus, the overall time complexity is $O(|V| + |E|)$.

**Correctness:**

**Property:** In an undirected graph, the only vertices reachable from a given vertex are the ones in its connected component. Thus, each connected component needs exactly one airport since all vertices in that connected component can reach that airport. Since we make an independent choice for each connected component for an airport, we choose the minimum cost city.

(b) The property mentioned above does not hold for directed graphs.
Question 2 (Puzzle Solving, 35 points). John is trying to solve a puzzle. In this puzzle he has an $n \times n$ board. Some of the squares are marked with obstacles, and one of which is a designated target square. The board also has two pawns on it. As a move, John makes take either pawn and move it as far as it goes in one of the four cardinal directions until it bumps into an obstacle, the edge of the board, or the other pawn. His objective is to make some sequence of moves that ends with one of the pawns at the target square. For example, in the figure below, the moves shown allow him to bring pawn A to the target square in 6 moves. Find an algorithm that given $n$, and the locations of the obstacles, pawns, and target runs in time polynomial in $n$ and determines whether or not it is possible for John to solve this puzzle. What is the runtime of this algorithm? 

Hint: Relate this question to one of reachability in some appropriate graph.

Solution 2. First, let's consider the simplified version of the problem where there is only one pawn while all other conditions remain the same. This problem can be converted to a graph problem as follows:

Vertices: The set of all possible pawn positions $(i, j)$. This is the set of all cells without obstacles on them.

Edges: For any given cell $C_{ij}$, there can be at most 4 valid moves that can be performed when the pawn is present on that cell. Since the pawn can only move horizontally or vertically, let the cells which can be reached from $C_{ij}$ in one move be: $C_{ij1}, C_{ij2}, C_{i1j}, C_{i2j}$. We add directed edges from $C_{ij}$ to all the 4 cells.

Note that our graph needs to be directed since it may not be possible to reach a cell back. For example, look at the edge labeled 4 in the figure. Now given the starting position of the pawn, we want to find whether the target can be reached. This translates to finding whether a path exists from the source (starting position) to the target in the graph.

Now we tackle the original problem which has two pawns, both of which can be moved. Imagine what would happen if we used the same graph as the simplified problem. While adding the edges for a given pawn position, we do not know the position of the other pawn. Note that this information is necessary since the other pawn acts as an obstacle when the current pawn is being moved. Thus, we need to consider the position of both the pawns.

We now construct the graph for the problem with two pawns:

Vertices: The set of all possible pair of pawn positions $(i, j, k, l)$. $(i, j)$ denotes the position for pawn 1 while $(k, l)$ denotes the position for pawn 2.

Edges: For a given vertex $(i, j, k, l)$, there are at most four moves for each of the two pawns. Since only one pawn can move at a time, there are 8 possible moves from a given pair of pawn positions. We add edges for these 8 moves.
Again, we translate this to a graph problem. The source is pair of starting positions for the two pawns. The target is slightly tricky: there are multiple target vertices. Any vertex in which one of the pawns is in the target square is a target vertex.

**Algorithm**

Input: Puzzle board, pawn starting positions, target square  
Output: Can one of the pawns reach target square: Yes/No  
Let V be the set of vertices  
Let E be the set of the edges  
For each square (i,j):  
    For each square (k,l):  
        If (i,j) != (k,l):  
            Add (i,j,k,l) to V  

For each (i,j,k,l) in V:  
    For all (i',j') reachable from (i,j) and all (k',l') reachable from (k,l) in one move  
        Add ((i,j,k,l) -> (i',j',k,l)) to E  
        Add ((i,j,k,l) -> (i,j,k',l')) to E  

Use DFS from starting pawn positions to find all reachable vertices  

**Correctness:**  
The number of vertices is the number of pair of squares which is $O(n^4)$. The number of edges is at most 8 times the number of vertices and hence is $O(n^4)$. Finding each edge can take $O(n)$ time. Finally, DFS takes $O(|V| + |E|)$. Thus, the total time complexity is $O(n^5)$.  

Question 3 (Largest Previsit Numbers, 30 points). Suppose that $G$ is a connected, undirected graph on $n > 1$ vertices.

(a) Show that when doing a depth first search on $G$ that the largest previsit number of any vertex is at most $2n - 2$. [5 points]

(b) Give an example of a connected graph on 5 vertices and a depth first search traversal of this graph so that some vertex has previsit number exactly 8. [10 points]

(c) Show that when doing a depth first search on $G$ that the largest previsit number is at least $n$. [5 points]

(d) Give an example of a connected graph on 5 vertices and a depth first search traversal of this graph so that the largest previsit number is exactly 5. [10 points]

Solution 3. (a) The total number of pre/post numbers are $2n$. The largest post number would be taken by the root of the DFS tree which has a pre number of 1. So the largest pre number must belong to some other vertex $v$. Since the post number of $v$ must be larger than its pre number, the largest possible pre number is at most $2n - 2$. Note that $n > 1$, otherwise the argument won’t hold.

(b) Example:

(c) The number of vertices is $n$ and each vertex must have a previsit number. Since previsit numbers are distinct, the largest must be at least $n$.

(d) Example: