Question 1 (Gravity Maze, 30 points). Harry is trying to navigate a three dimensional maze. The maze is constructed using an $n \times n \times n$ grid of cubes where each cube is either open space that can be traversed or a wall which cannot. Harry’s location at any point in time must be one of the open cubes. Harry cannot climb, but he does have the ability to manipulate gravity. At any point gravity must be pointing in one of the six cardinal directions (up, down, north, south, east, or west, these are aligned with the axes of the cube). Harry can do two things to traverse the maze. Firstly, he can take a step in one of the six cardinal directions to an adjacent open cube in the maze, and then fall as many squares in the current direction of gravity as possible until a wall or the end of the maze stops him. His other option is that he can change the direction that gravity points 90 degrees to a different one of the cardinal directions and then fall in the appropriate direction.

Produce an algorithm that given a description of the maze (which squares are open, and which are walls), the location of the exit, Harry’s initial location, and the initial direction of gravity determines whether or not it is possible for Harry to complete the maze. The runtime for this algorithm should be polynomial in $n$.

Solution. The basic idea is to run DFS to determine reachability. Since we cannot figure out the legal moves by just considering the locations, we will need to create a more complicated graph whose vertices represent states which are determined by both the location and the gravity direction.

Graph Construction: Denoting the set of cubes as $C$ and the set of directions as $D$, we define the set of vertices $V$ as $C \times D$. For any two states $(c, d), (c', d') \in V$, $(c', d')$ is reachable by $(c, d)$ means being inside cube $c$ under gravity with direction $d$, Harry will be able to reach cube $c'$ under gravity with direction $d'$. For each $(c, d)$, we find its children by the following steps:

1. If $c$ is a wall, $(c, d)$ has no child.
2. Else if right next to $c$ in the direction $d$ is a open cube $c'$, then $(c', d)$ is the only child of $(c, d)$.
3. Else, for every direction $d' \perp d$, add $(c, d')$ to $(c, d)$’s children and if there is an open cube $c'$ right next to $c$ in the direction $d'$, add $(c', d)$ to $(c, d)$’s children.

Algorithm: After building the graph, explore the starting point $(c_0, d_0)$ where $c_0$ is the starting cube and $d_0$ is the starting gravity direction. Return true if any vertex containing the destination is explored when explore halts, false otherwise.

Correctness: A path that can be taken in the graph corresponds to a path Harry can take in the maze. Any path in the graph must either move to an adjacent square or rotate gravity by 90 degrees and then fall until hitting a floor. These are exactly the things Harry can do. Therefore a path in the graph to the exit corresponds exactly to a way that Harry can get out of the maze. Given the property of DFS, we know explore will return true iff Harry can get.

Complexity: Since there are $n^3$ cubes, each cube generates 6 vertices, and each vertex has at most 8 children, $|V| = O(n^3)$ and $|E| = O(n^3)$. Since building each edge requires $O(1)$ time, building the whole graph costs $O(n^3)$ time. Since running DFS costs $O(|V| + |E|) = O(n^3)$ time, the whole algorithm takes $O(n^3) + O(n^3) = O(n^3)$ time.
Question 2 (Best Neighborhood, 30 points). Murphy is looking to buy a house. One of her major considerations is the ability to walk to places that she might want to go. The city she lives in is described by an undirected graph \( G \) whose vertices are locations and whose edges denote walking paths. Each location is assigned by Murphy a non-negative number of points. Murphy wants to find a location so that the greatest possible total number of points of other locations are reachable on foot from there.

(a) Give a linear time algorithm for solving this problem. [25 points]

(b) Does the above algorithm work if \( G \) were a directed graph instead? Why or why not? [5 points]

Solution.

(a) **Algorithm:** Let’s find all connected components with the help of DFS. Then, let’s find the sum of all points of all nodes for each connected component, i.e. \( s(i) = \sum_{v \in c(v)} a(v) \), where \( i \in \{1, 2, \ldots, k\} \), \( a(v) \) is the number of points for node \( v \), \( c(v) \) is the label of the connected component to which node \( v \) belongs to and \( k \) is the number of connected components. Finally, we choose a connected component that has the largest total number of points, i.e. \( i^* = \arg\max_{1 \leq i \leq k} \{ s(i) \} \). We take any node in this connected component and output it as the answer, i.e. output any node \( v^* \), such that \( c(v^*) = i^* \).

**Correctness:** The greatest possible total number of points of nodes reachable from some \( v \) will be the total number of points in a connected component, which includes \( v \). Indeed, a connected component which includes node \( v \) is a set of nodes in the graph that are reachable from \( v \). As all points are non-negative numbers, we should take all nodes in the connected component. This set is also the largest of its kind, thus there are no other nodes that can be included to it. It means, that we can’t get more points starting from node \( v \) then the sum of all points of all nodes in the connected component which includes \( v \). We can also see that any node in the connected component will give the same total amount of points (which equals the total amount of points in the connected component). From the above, we can conclude that if \( v_{\text{opt}} \) is the optimal node, then:

(a) The total number of points that it gives equals the total number of points of the connected component it belongs to

(b) The total number of points of its connected component is at least as total number of points in any other connected component

(c) Any node from its connected component will give the same amount of total points

Our algorithm finds the connected component with the largest number of points (if there are several of them, then it chooses any of them). From the first two properties, we know that the optimal answer will belong to this connected component. Then the algorithm chooses any node that belongs to this connected component. From the third property we know, that this node will give the optimal answer.

**Time complexity:** We find connected components in \( O(|V| + |E|) \), because each node visited only once by DFS. We can find total points for each connected component in \( O(|V|) \), by iterating through nodes and updating values for total points for connected components, which are initially equal to zero. Finding maximum and the argument (optimal node) can also be done trivially in \( O(|V|) \). Thus, overall time complexity will be \( O(|V| + |E|) \).

(b) For directed graphs our algorithm will find a set of reachable nodes for each node from which we start DFS, but not strongly connected components. Even if we did find strongly connected components first, then the second part of the algorithm would still be incorrect, because for directed graphs we may have edges between strongly connected components. Thus, there may be nodes from one strongly connected component that are reachable from nodes from other strongly connected components. This implies, that the node which gives the optimal answer doesn’t have to be in the strongly connected component with the maximal total points of nodes that it includes. A very simple example: we have three nodes \( \{1, 2, 3\} \) with points \( \{10, 20, 30\} \) respectively. We also have two directed edges: \( \{(1, 2), (2, 3)\} \). Each node is a strongly connected component is this graph. The strongly connected component with the largest total number of points is node 3, while the optimal node is 1.
Question 3 (Preorder Numbers and Graph Structure, 40 points). We are given a graph \( G \) on \( n \) vertices. Depth first search is run on \( G \) and pre- and post-order numbers are computed.

(a) If \( G \) is an undirected graph, and the pre-order numbers are \( 1, 2, 4, 6, \ldots, 2n - 2 \), what can you say about the structure of \( G \)? [10 points]

(b) If \( G \) is a directed graph, and the pre-order numbers are \( 1, 2, 3, \ldots, n \), what can you say about \( G \)? [10 points]

(c) If \( G \) is an undirected graph, and the pre-order numbers are \( 1, 3, 5, 7, \ldots, 2n - 1 \), what can you say about \( G \)? [10 points]

(d) If \( G \) is a directed graph, and the pre-order numbers are \( 1, 3, 5, 7, \ldots, 2n - 1 \), show that \( G \) is a DAG. [10 points]

Solution.

Let \( \text{Pre}(i) \) represents the pre-order number of the \( i \)th vertex, and \( \text{Post}(i) \) represents the post-order number of the \( i \)th vertex of the graph \( G \).

(a) This graph will be a tree of height 1 with \( n - 1 \) leaves. The root of this tree will be the 1st vertex with all other \( n - 1 \) vertices being its direct descendants.

Explanation: Since, \( \text{Pre}(1) = 1 \) and \( \text{Pre}(2) = 2 \), we can say that the second vertex is a descendant of the first vertex, because the second vertex is being explored during the exploration of the first vertex. Next, we see \( \text{Pre}(3) = 4 \) which means that the Post(2) must be equal to 3, implying that the second vertex has been explored completely and, therefore, it can’t have any other outgoing edges. \( \text{Pre}(3) = 4 \) also means that the third vertex is being explored during the exploration of the first vertex. Therefore, we can say that the third vertex is also a descendant of the first vertex. Similarly, we can conclude that all other vertices are the descendants of the first vertex. Since, pre-order labellings are not consecutive after the second vertex, it can also be inferred that there can’t be other edges \((u, v)\) for \( u, v \neq 1 \).

(b) This graph can be represented by vertices arranged in a straight line where there is a directed edge from every \( i \)th vertex to \((i + 1)\)th vertex.

Explanation: Since the pre-order numberings of the graph \( G \) are consecutive, it can be inferred that the \((i + 1)\)th vertex is the descendant of the \( i \)th vertex, and hence, there is a path containing all the vertices once.

(c) This graph contains a set of \( n \) disconnected vertices.

Explanation: As it’s an undirected graph, if we had at least one edge then there would have been at least two vertices with consecutive pre-order labellings. But, as there are no vertices with consecutive pre-order labellings, we conclude that there is no edge between any pair of vertices.

(d) The post-order labellings of the vertices will be \( 2, 4, 6, \ldots, 2n \). It can be noticed that \( \text{Post}(i) = \text{Pre}(i) + 1 \) for all the vertices. This means that there are no edges from the vertex \( i \) to an unexplored vertex. Therefore, we can say that the \( i \)th vertex visited cannot have an edge to the \( j \)th vertex visited for \( j > i \), and this provides a linear ordering. In other words, since for every edge \((u, v)\) we have \( \text{Post}(u) > \text{Post}(v) \), it can be inferred that there are no back edges present in the graph. Therefore we conclude that the graph is a DAG.