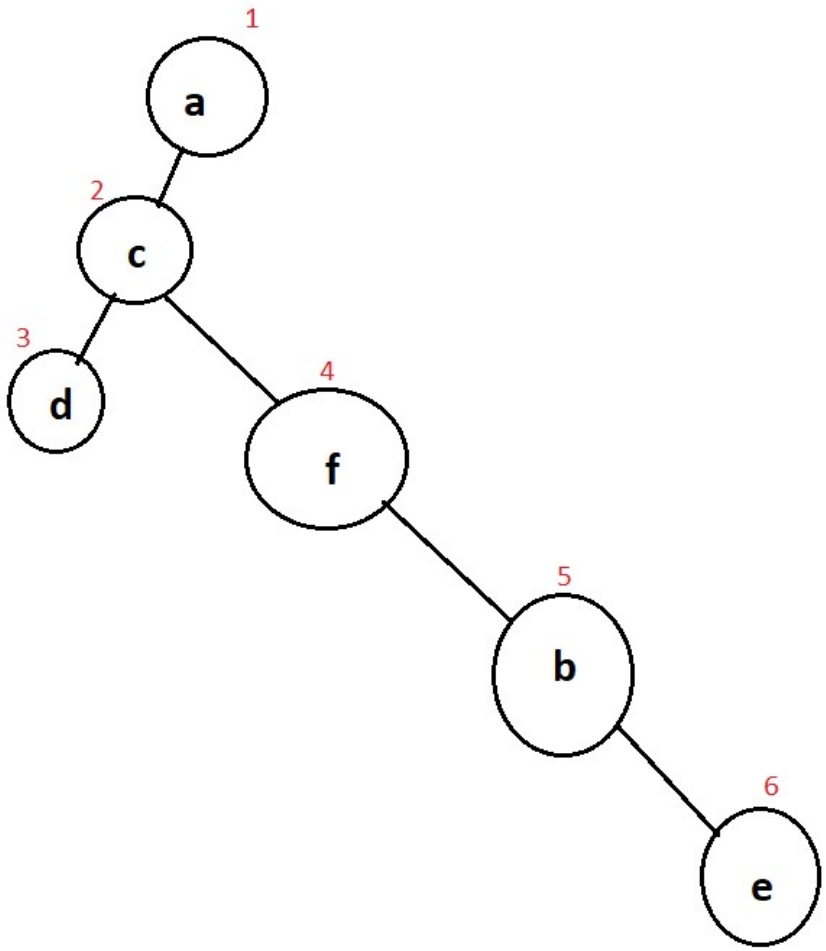
Part 2:

Question 1. a -> c -> d -> f -> b

Question 2. a -> c -> d - > f -> b -> e

Question 3.



Part 3:

Question 1. There is an infinite number of states available for the robot. This is because since each state is represented by an (x,y) coordinate in the plane, and that there are an infinite number of (x,y) coordinates, there can be an infinite amount of potential locations for the robot to be Most notably, the robot start and end positions can be anywhere in the (x,y) plane. Similarly, there can be up to an infinite amount of paths because of the fact that the robot can travel to any 2 points in the plane (given there are no intersections).

Question 2. If you can restrict the robots starting/ending positions to only the vertices of the polygon, or to only one specific coordinate instead of any coordinate on the (x,y) plane, you can restrict the state space from infinity to the number of vertices + 2 (or just number of vertices if the robot is only allowed to start/end on a vertex). This is because restricting the robot's positions to the fixed vertices removes the continuous, infinite nature of the (x,y) plane and means that the positions of the robot are a set of discrete values versus the continuous range of the whole 2d plane.

Question 3.

```
Function aStar (Node start, Node target)
  // openList is storted by f(h) of all elements in it
  openList := start // start off with the starting node
  closedList := null // start off empty
  while openList != empty
    currentNode := openList.pop // get node with lowest f(h) in openList
    if currentNode == target
      return solution // we are done
    for every checkNode in allNodeList // check ALL nodes because…
      // …neighbor nodes are considered any node that we can path to
      // without intersecting any line segments created by the polygons
      // (If intersections are made, this node is invalid to path to)
      if checkNode != currentNode && check_no_intersect(currentNode, checkNode)
        if checkNode is not in openList and checkNode is not in closedList
          set checkNode parent to currentNode
          // parent refers to node currentNode will path to in the final path
          calculate g(n) and f(n), save it to checkNode, and add it to openList
          // also need to sort openList after adding new node to it
        else
          if f(n) from last iteration is better than g(n) from this iteration:
            set checkNode parent to currentNode
            // parent refers to node currentNode will path to in the final path
            calculate g(n) and f(n), save it to checkNode
            if checkNode is in closedList
              add checkNode to openList, remove from closedList, sort openList
    remove currentNode from openList and closedList // we are done with this node
// if we finish the while loop and don't return a solution, no path was found
```

Question 4.

```
Node id: 0 Cords: (115, 655) Polygon ID: -1
Node id: 1 Cords: (380, 560) Polygon ID: -1
Node id: 2 Cords: (220, 616) Polygon ID: 1
Node id: 3 Cords: (220, 666) Polygon ID: 1
Node id: 4 Cords: (251, 670) Polygon ID: 1
Node id: 5 Cords: (272, 647) Polygon ID: 1
Node id: 6 Cords: (341, 655) Polygon ID: 2
Node id: 7 Cords: (359, 667) Polygon ID: 2
Node id: 8 Cords: (374, 651) Polygon ID: 2
Node id: 9 Cords: (366, 577) Polygon ID: 2
Node id: 10 Cords: (311, 530) Polygon ID: 3
Node id: 11 Cords: (311, 559) Polygon ID: 3
Node id: 12 Cords: (339, 578) Polygon ID: 3
Node id: 13 Cords: (361, 560) Polygon ID: 3
Node id: 14 Cords: (361, 528) Polygon ID: 3
Node id: 15 Cords: (336, 516) Polygon ID: 3
Node id: 16 Cords: (105, 628) Polygon ID: 4
Node id: 17 Cords: (151, 670) Polygon ID: 4
Node id: 18 Cords: (180, 629) Polygon ID: 4
Node id: 19 Cords: (156, 577) Polygon ID: 4
Node id: 20 Cords: (113, 587) Polygon ID: 4
Node id: 21 Cords: (118, 517) Polygon ID: 5
Node id: 22 Cords: (245, 517) Polygon ID: 5
Node id: 23 Cords: (245, 577) Polygon ID: 5
Node id: 24 Cords: (118, 577) Polygon ID: 5
Node id: 25 Cords: (280, 583) Polygon ID: 6
Node id: 26 Cords: (333, 583) Polygon ID: 6
Node id: 27 Cords: (333, 665) Polygon ID: 6
Node id: 28 Cords: (280, 665) Polygon ID: 6
Node id: 29 Cords: (252, 594) Polygon ID: 7
Node id: 30 Cords: (290, 562) Polygon ID: 7
Node id: 31 Cords: (264, 538) Polygon ID: 7
Node id: 32 Cords: (198, 635) Polygon ID: 8
Node id: 33 Cords: (217, 574) Polygon ID: 8
Node id: 34 Cords: (182, 574) Polygon ID: 8
0 17 32 2 25 12 1
```

Start: (115, 655) -> (151, 670) -> (198, 635) -> (220, 616) -> (280, 583) -> (339, 578) -> (380, 560)

Question 5. Yes, a BFS/DFS is possible. This is because there still exist “neighbor” nodes for a BFS/DFS to search through, it just happens that the neighbor set is all existing nodes in the set minus nodes not available due to intersections. BFS will have the start node iterate through every node it can connect to before proceeding to the next node and repeating. DFS would consist of

long chains of searches as each node has at least two children, potentially iterating through the whole set of nodes in a single “chain”.