Homework 5

Problem 1.

Construct Binary Tree from its preorder and inorder traversals.

buildTree (preorder, inorder)

IF length of preorder = 0 THEN
    Return NULL
ENDIF

Root.id = preorder[0];

1 = location of root in inorder

Root.left = buildTree(preorder[ 1 ... l ], inorder[ 0 ... l-1]);

Root.right = buildTree(preorder[ l+1 ... end ], inorder[ l+1 ... end]);

Return Root

END

If the first index is greater than the last index, the array is empty.

Problem 2.

a, Removing an edge from a tree creates two trees i.e. one more connected component is generated. Thus we have p+1 connected components after removing p edges from a tree.

b, We cannot compute the number of connected components because we don’t know the number of removed edges. This number depends on the degrees of removed vertices and whether they are adjacent or not.

Problem 3.

2^h

Problem 4.

Breadth-first Search and Depth-first Search:

BFS and DFS are exhaustive search algorithms. They are simple to implement. And it can be applied to any search problem. Comparing BFS to depth-first search algorithm, BFS does not
suffer from any potential infinite loop problem, which may cause the computer to crash. One disadvantage of BFS and DFS is that they are ‘blind’ search, when the search space is large the search performance will be poor compared to other heuristic searches.

BFS will perform well if the search space is small. It performs best if the goal state lies in upper left-hand side of the tree. But it will perform relatively poorly relative to the depth-first search algorithm if the goal state lies in the bottom of the tree. BFS will always find the shortest path if the weight on the links are uniform.

**A* algorithm**

Heuristics search makes use of the fact that most problem spaces provide some information that distinguishes among states in terms of their likelihood of leading to a goal. It’s more complicated than BFS and DFS. It tends to reduce the number of searched nodes. If the heuristics function is good, the amount of searched nodes is reduced significantly.

The main shortcoming of A*, and any best-first search, is its memory requirement. Because the entire open pathway list must be saved, A* is space-limited in practice and is no more practical than breadth first search. For large search spaces, A* will run out of memory.

**Problem 5.**

Notice that we only know points’ locations. First, compute the weight between any pair of nodes then use Prim or Kruskal algorithm to find Minimum Spanning Tree. This algorithm has time complexity of $O(dn^2)$ for computing the weight of edges.

To reduce the time to compute the distances between distinguish points, we can first compute Delaunay triangulation of input points. Then we use Prim, Kruskal algorithm on this subgraph of n points. It’s proved that all edges of MST belong to this subgraph. Especially with $d = 2$, time complexity is $O(n \log n)$ which is much better than $O(n^2)$.