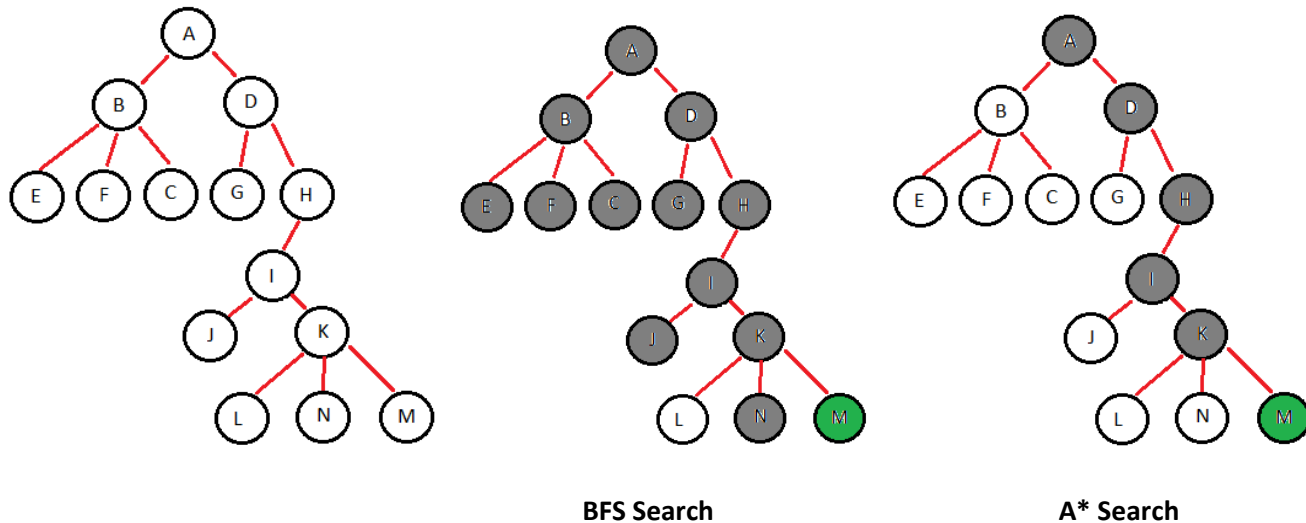


Project 3: Maze Navigation

In this project we demonstrated the power of informed search specifically in the context of maze navigation. A path through a maze can be portrayed as a tree of interconnected nodes (or a graph). We see that an informed search algorithm, particularly A*, cuts the number of operations by a significant amount. A*, which uses a heuristic to determine its next best path, proves to be an ideal algorithm for pathfinding both because it is *admissible* and *consistent*.

The source code for everything below can be found [here](#).

Sketch the search tree representing the maze:



What kind of search is implemented during the left (right) side traverse?

Given that the robot has found all possible nodes (A through M), it has probably used a breadth-first search. Visually, BFS expands radially outwards and thus explores all possible states. This is true only if it is not looking for a goal but merely searching.

Describe and illustrate a greedy algorithm for finding the exit:

I chose breadth-first search for my greedy algorithm. We can see that it visit nearly every node in the maze before finding M. Although complete, it is inefficient.

```
public void BreadthFirstSearch(Node start, Node goal)
{
    Queue<Node> queue = new Queue<Node>();
    Node current = start;

    queue.Enqueue(current);

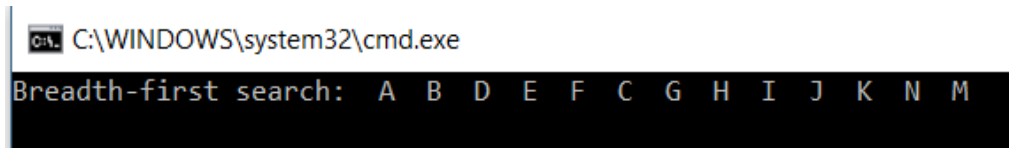
    Console.WriteLine("Breadth-first search: ");

    while(current != goal)
    {
        current = queue.Dequeue();

        Console.Write(current.Value + " ");

        foreach (Node neighbor in current.Children)
        {
            queue.Enqueue(neighbor);
        }
    }

    Console.WriteLine();
}
```



```
C:\WINDOWS\system32\cmd.exe
Breadth-first search: A B D E F C G H I J K N M
```

As demonstrated and drawn above, we see that BFS is guaranteed to find a solution if there is one. However, we might visit every possibility before doing so.

Describe and Illustrate an A* algorithm for finding the exit:

```
public List<Node> AStarSearch(Node start, Node goal)
{
    Dictionary<Node, Node> parentMap = new Dictionary<Node, Node>();
    PriorityQueue<Node> queue = new PriorityQueue<Node>();

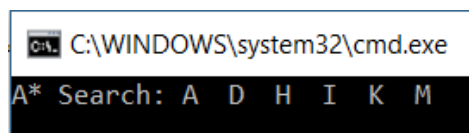
    queue.Enqueue(start, 0);
    Node current;
    Console.WriteLine("A* Search: ");

    while(queue.Count > 0)
    {
        current = queue.Dequeue();

        if (!current.IsVisited)
        {
            current.IsVisited = true;

            if (current == goal)
            {
                break;
            }

            foreach (Node neighbor in current.Children)
            {
                parentMap.Add(neighbor, current);
                double priority = Heuristic(neighbor, goal);
                queue.Enqueue(neighbor, priority);
            }
        }
    }
    List<Node> path = ReconstructPath(parentMap, start, goal);
    return path;
}
```



This is a fairly typical implementation of A*: a parent hash map keeping track of the path progression, a priority queue to explore the nodes with the best path first, and obviously a heuristic which is the Euclidean distance between nodes:

```
public double Heuristic(Node nodeA, Node nodeB)
{
    return nodeA.EuclidianDist(nodeB);
}

public double EuclidianDist(Node node)
{
    double xComponent = Math.Pow((Location.X - node.Location.X), 2);
    double yComponent = Math.Pow((Location.Y - node.Location.Y), 2);

    return Math.Sqrt(xComponent + yComponent);
}
```