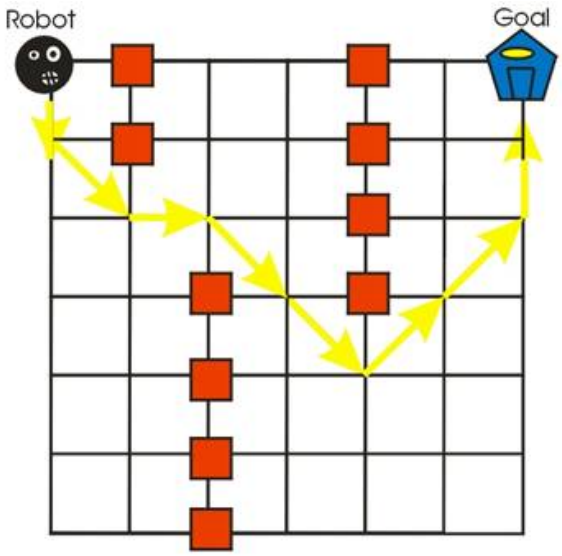
Breadth-First and Depth-First Search for Path Planning

Keywords: breadth, depth, search, path planning



The photo depicts a scenario where a robot is in an environment filled with obstacles and needs to get to its goal location. The robot is allowed to move one north/south, one unit west/east, and one unit diagonal NW/NE/SW/SE at each node (grid intersections). The yellow line represents the path that the robot should know to take. The big question is, how do we get our robot to find a path to the goal around obstacles? Solving this problem is important because autonomous robots require this ability to plan its path. This tutorial shows you how to use Breadth first search and Depth First Search algorithms to generate a path for the robot to take and the tutorial takes approximately 1 hour to complete.

Motivation and Audience

Programming a robot to travel from one location to a goal location is usually more complicated than just moving in a straight line as there are usually obstacles that lie in the straight line path between the starting position and the goal. Obstacle avoidance becomes an important issue if we are concerned about damage of the obstacles by the robot or vice versa. There are many different obstacle avoidance strategies used in path planning from very simple strategies like “change direction when the robot encounters an obstacle” to more advanced strategies such as the use of A* which utilizes information from previous states and estimated future states to generate a path.

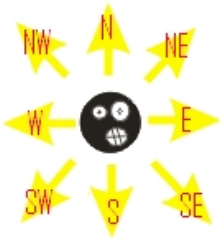
This tutorial's motivation is to illustrate the usefulness of breadth first search and depth first search algorithms for path planning. This tutorial assumes the reader has the following background and interests:

- ***Familiar with Visual Basic 6.0***
- ***Familiar with Graph Trees***
- ***Interest in path planning and artificial intelligence***

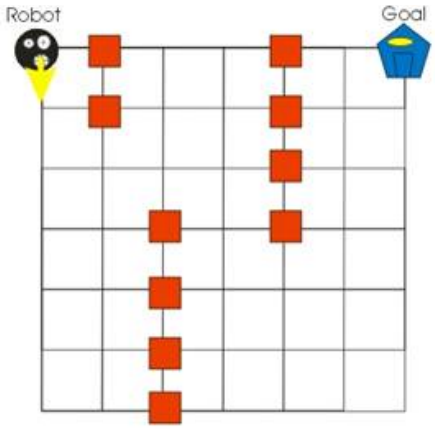
The rest of the tutorial is presented as follows:

Introduction to Breadth First Search (BFS)

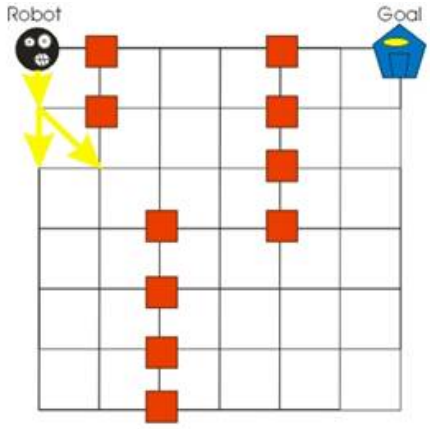
Breadth First search is known as an uninformed search because it does not use any information about how far the robot has traveled or how far the robot is from the goal. BFS begins at the starting position of the robot (root node) and begins looking for the goal by expanding all of the successors of the root node. In the scenario stated at the very start of this tutorial, the successors of a node are all allowable directions that the robot could travel next. Allowable means that directions causing the robot to crash into an obstacle, to move outside of the workspace will not be considered as successors of the node. Nodes that have already been visited by the robot will not be considered successors either. As seen in the figure below, all the directions that the robot could possibly travel are shown in yellow. There are 8 possible directions.



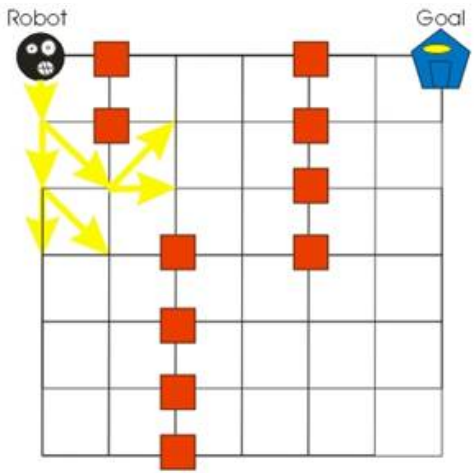
Now if we look at the starting position of the robot, we can see that the only successor of the starting node is S because all other directions cause the robot to either crash into an obstacle or to move out of the workspace.



If the next node is not the goal, then BFS expands all of the successors of the next node. In this next case, the there are two successors, S and SE.



If S and SE are not the goal, then BFS expands each of those nodes.

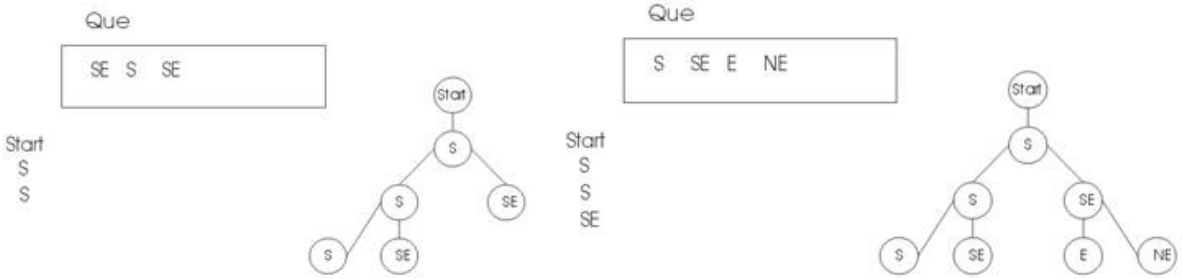


If those successors are not the goal, then BFS expands each of those nodes, and this loop continues until the goal node is reached. It is much easier to visualize BFS if we convert this scenario into a tree and search the tree with a fringe that is FIFO (first in first out) que. The que is an array of expanded nodes from which we use to determine which node is to be expanded next. A FIFO que means that the node that has been sitting on the que for the longest amount of time is the next node to be expanded. The following figures will demonstrate this technique. We begin with the START node and add it to the que.

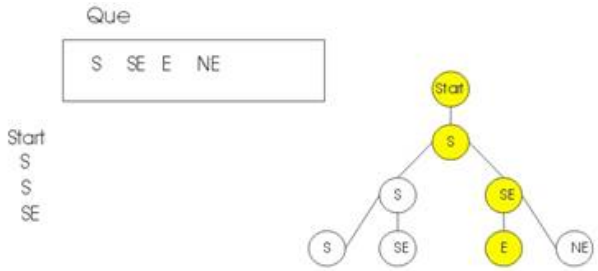
Start is taken out of the que and expanded. South is the only successor so it is added to the que.

South is taken out of the que and expanded. S and SE are the successors and they are added to the que.

S is taken out of the que and expanded adding S and SE to the que. Next SE is expanded adding E and NE to the que.



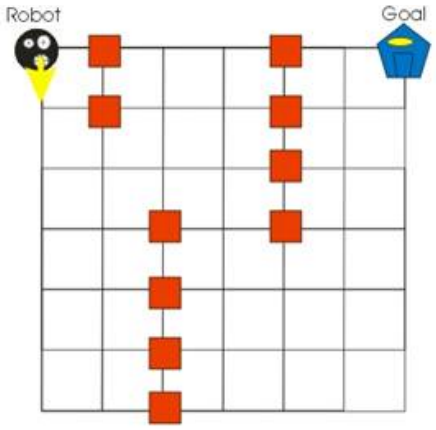
The FIFO que continues until the goal node is found. When found, the path leading to the goal node is traced back up the tree which maps out the directions that the robot must follow to reach the goal. Lets say that the goal was found when the SE was expanded and the goal was found at the E node. Traveling back up the tree, we can see that the robot from the start would have to go south, then south east, then east to reach the goal.

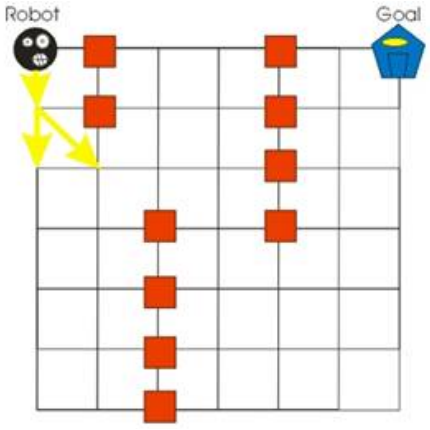


Breadth first search is complete as long as the branching factor is finite. This means that the algorithm will always return a solution if a solution exists. In a path planning sense, breadth first search will always find the path from the starting position to the goal as long as there is an actual path that can be found. Breadth first search is only optimal if the path cost is the same for each direction. In this case, the path cost would be the direction and optimal means the shortest distance path from the start to the goal. Every node generated must remain in memory and the number of nodes generated is at most O(7^(d+1)) where 7 represents the maximum branching factor for each node and d is the depth one must expand to reach the goal. We can see from this that a for very large workspace where the goal is deep within the workspace, the number of nodes could expand exponentially and demand a very large memory requirement.

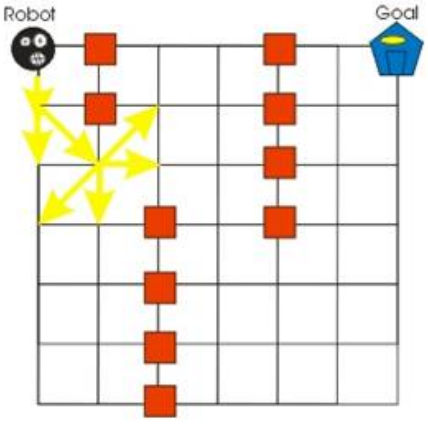
Introduction to Depth First Search (DFS)

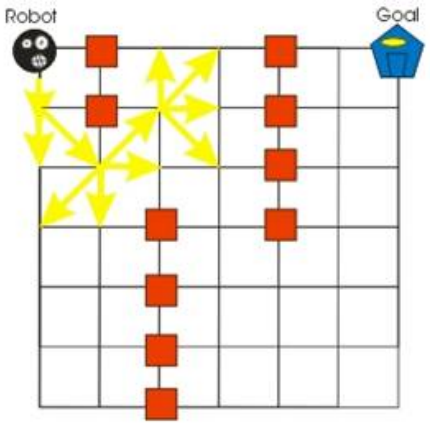
Depth first search is the complement to breadth first search. It is also an uninformed search method. Starting from the root node it expands all the successors of the start node same as BFS. However after the first node, DFS always expands the last successor added. As seen from the figure below, the first node is expanded just like BFS and then the second node is expanded. These two steps look similar to BFS because there was only one successor to the start node.



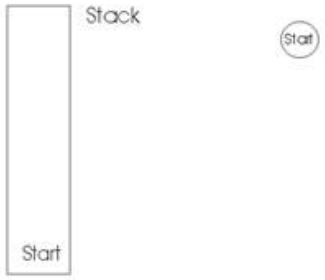


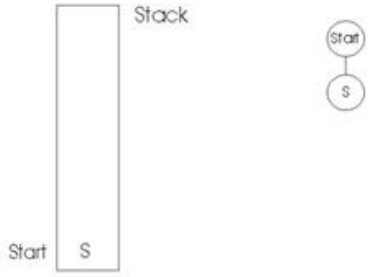
Then next step is different from BFS as the node expanded next is the SW node and then the nodes expanded after that it the NW node from the SW parent node.



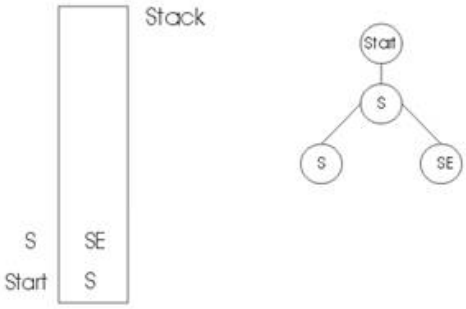


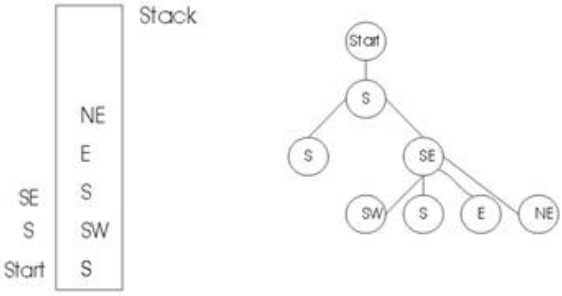
If DFS gets to a point where it can not find any more successors for the node it is currently expanding, it will then move to the newest discovered node that hasn't been expanded. As with BFS, DFS is easier to understand if we look at a tree and use a LIFO (last-in-first-out) stack. The stack contains the list of discovered nodes. The most recent discovered node is put on top of the LIFO stack. The next node to be expanded is then taken from the top of the stack and all of its successors are added to the stack. The following figures illustrate DFS using the path planning scenario stated at the start of this tutorial.
We first start with the START node and add that to the stack. Then START is removed from the stack and its successors (S) are added to the stack.





The S node is removed from the stack and its successors are added to the stack in the order that they are discovered (S, SE). SE is then removed from the stack and its successors are added (SW, S, E, NE) in the order that they are discovered.

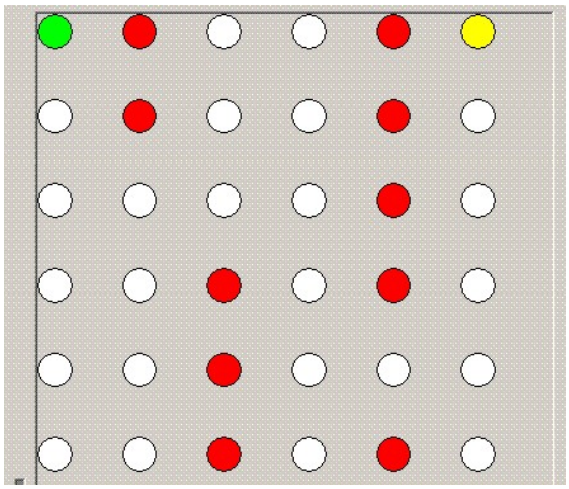




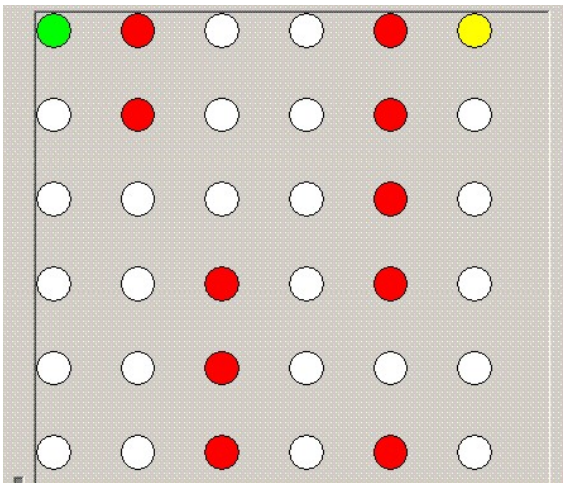
The next node to be expanded would be NE and its successors would be added to the stack and this loop continues until the goal is found. Once the goal is found, you can then trace back through the tree to obtain the path for the robot to follow.
Depth first search usually requires a considerably less amount of memory that BFS. This is mainly because DFS does not always expand out every single node at each depth. However there are some drawbacks to depth first search the BFS will not suffer from. Depth first search is not optimal and it is not complete. The reason it is not optimal is that it returns a path as soon as the goal is reached, however, this path is not always the shortest path but a path generated by the result of going down a long branch. Another problem is the DFS could continue down an unbounded branch forever even if the goal is not located on that branch. There are a couple of techniques used to keep DFS from continuing down an infinite branch and that technique is called Iterative Deepening which sets a limit for the depth that DFS will search down a branch before switching to another node. This approach is the preferred uninformed search method when there is a large search space and the depth of the solution is not known.

Implementation of BFS and DFS (Visual Basic 6)

Both BFS and DFS search algorithms were implemented using visual basic 6. A 6 by 6 grid was created with a starting point (green), a target (yellow) and obstacles (red). The code including detail descriptions of each part of the code can be found in the download section of this tutorial. Two animations of the search results for a given configuration of obstacles, start and goal can be seen below. The left illustrates the breadth first search algorithm and the right illustrates the depth first search algorithm.



BREADTH FIRST SEARCH



DEPTH FIRST SEARCH

Each node that is expanded is given the color blue and its successors are given the color black. Once the goal has been found, the path of travel is traced by green dots.

Final Words and Exercises

The Visual Basic simulations give a very good illustration of the differences between breadth first search and depth first search. Interestingly, both searches produce the same final optimal path for the configuration given in the simulation. It is quite obvious that for the configuration given in the simulation, depth first search produces a much faster result to the target and expands a much smaller number of nodes to reach the target. BFS and DFS are just a few of the algorithms used for path planning in robotics. Remember, DFS and BFS are uninformed searches and are good to use (especially iterative deepening) if the location of the goal is unknown. If the location of the goal is known A* is a more established algorithm for the general searching of optimal paths in path planning. The tutorial for A* can be found here.

Exercise 1) Using the Visual Basic code provided, change the location of the target such that Breadth first search will give a faster result than depth first search (expand less nodes).

Exercise 2) In the case of the simulation shown, the path towards the goal was optimal meaning the shortest possible path between the start and the target was found. Is this always the case? Change the orientation of the obstacles, start, and goal such that the results provide non—optimal solutions.

Downloadable Files from this Tutorial

dfsbfs.zip (Visual Basic 6 Code Author: James Hing)
depthandbreadth.zip (Matlab implementation of DFS an BFS Author: Keith Sevcik)

References

Stout, W. Bryan “Smart Moves: Intelligent Path Finding” Gamasutra.com Feb 12, 1999
S. Russel, P. Norvig, Artificial Intelligence: A Modern Approach Second Edition.

If you have any questions or comments about this tutorial, please email me at jth23@drexel.edu