Graph Searching

Yongkoo Kang

October 18, 2013

1 Introduction

Before we start, what's a graph? Graphs are a series of nodes connected by edges. Nodes and edges may have their own values. There are many ways to traverse a graph; however, the most basic ones are the Breadth First Search (BFS) and the Depth First Search (DFS).

2 Breadth First Search (BFS)

The BFS goes through a graph layer by layer, essentially visiting every node within a certain distance before moving onto nodes that are farther away from the starting node. Implementation of the BFS generally consists of a queue because of the first in first out processed nature of the search. I'm actually a bit too comfortable with Dijkstra so when I implement BFS I just write a Dijkstra and change the priority queue to a regular queue.

2.1 Pseudocode

The following pseudocode is a very crudely written version of BFS which finds and returns a node.

```
def BFS(Graph g, node start, node SEARCH)
    Queue q;
    boolean[] visited;
    boolean found = false;
    while(!q.isEmpty())
        node temp = q.pop();
        if(temp.equals(SEARCH))
            return SEARCH
        if(!visited[temp))
            for(node n : temp.edges)
                if(!visited[n])
                    q.put(n);
```

3 Depth First Search (DFS)

On the other hand, DFS follows chains all the way to the bottom, a node which does not connect to any other nodes except for its parent, and unravels the rest in a stack-based fashion. As the last node to be found is the first one to be used, the implementation of a DFS is based on a stack or, more commonly, recursion.

3.1 Pseudocode

The following pseudocode is a crudely written DFS that finds and returns a node. This is incredibly crude in that it fails to implement visited arrays and also does not fully return up the ladder, but this is just for the sake of showing the general recursive implementation of the DFS.

```
def DFS(node loc, node SEARCH)
    if(loc.equals(SEARCH))
        return SEARCH;
    for(node n : loc.edges)
        DFS(n, SEARCH);
```

4 Comparison

DFS and BFS are basically the same algorithm with different data structures. Both BFS and DFS have a runtime complexity of $O(E+V)$ where E is the number of edges and V is the number of vertices. However, the difference between the two lies in the space complexity. Initialization of both requires space of $O(E+V)$ for storage of edges and vertices. However, although both have a worst case of V during runtime, there will usually be more nodes in the queue for BFS, which will require a higher amount of memory, because most graphs are wider than they are tall. On the other hand, due to the nature of visiting the closest nodes first, BFS will, on an unweighted graph, find the nodes in terms of distance from the root allowing for knowledge of shortest path information and quicker searches for closer queries. Conversely, DFS will find nodes farther away faster, though the definition of a farther node is blurred in terms of a DFS due to the possibility of existence of shorter paths to such nodes, and also requires far less runtime memory.

5 Other Searches

Other graph searches include but are not limited to:

1. Bidirectional BFS: Shortest path on unweighted graphs which approaches from both sides to decrease memory usage
2. Iterative Deepening DFS: Solves issue of inability of DFS to find shortest path with a distance limit at each step
3. Dijkstra : Shortest path on weighted graphs
4. Bellman-Ford: Shortest path on weighted graph with negative weights
5. A*: Similar to Dijkstra but with improvements with heuristics

6 Problems

1. Erratic Ants (ICPC Qual October 5, Problem E) Max, an ant, is searching for food. He can walk in the 4 cardinal directions and will eventually reach a food source; however, this will likely not be the best way. As such, given that the ants can only walk in charted territory, namely along any path that Max has already walked, determine what the length of the shortest path to the food is. The length of Max's initial path is at max 60.
2. Clocks (IOI '94 - Day 2) There are 9 clocks labeled A-I each set to a time in the set 3,6,9,12. There are 9 different moves possible, where each letter represents moving forward the time on the specified clock by 3 hours: ABDE, ABC, BCEF, ADG, BDEFH, CFI, DEGH, GHI, and EFHI. Given a certain arrangement of clocks, find the shortest sequence of moves which returns all the clocks to 12, and when there is more than one way to do it print the one that is lowest numerically.