Graph Algorithms

The three basic algorithms that search to find a path in a graph from the given origin to the given destination are the *Breadth First Search* (BFS), the *Depth First Search* (DFS) and the *Shortest Path* (SP) algorithm. All three use the same basic approach and differ only in the manner in which they keep the *To Do* information of nodes to visit next. The BFS keeps the *To Do* information as a queue, the DFS keeps the *To Do* information as a stack, and the (SP) keeps the *To Do* information as a priority queue, selecting at each step to remove the node with the shortest distance to the origin.

For this algorithm to work, we need to represent the graph as a collection of nodes (an *ArrayList* or a *HashMap* can work) where each node can look up its list of neighbors (and for the SP, it also can determine the distance to each neighbor).

When we visit a node *N*, we add all of its neighbors to the *To Do* information together with the information that we came from *N* and, in the case of the SP algorithm, also the distance to each neighbor if we reached it through the node *N*.

In addition, as we go on, we keep track for every visited node how did we get there (from which other node). This can be a simple list or a *HashMap*, or we can add this information to each node of the graph directly. We will call this the *backtrack list*.

Here is a brief description of all three algorithms:

Search Algorithms

1. Start with an empty *To Do* information. Find in the collection of nodes the start node and add it to the *To Do* information, with an empty node as the node we came from and the distance equal to zero.

2. Repeat the steps 3. though 4. until one of the conditions in the next step is satisfied.

3. Remove a node from the the *To Do* information. If one of the condition below holds, stop the loop and take the specified action.

    - The *To Do* information is empty, in which case no path has been found.
    - The node we remove from the *To Do* information is the destination. If this is the case, finish the work with the *Backtracking algorithm*

Otherwise, add the removed node N to the *backtrack list*.

4. Add all neighbors M of the node N to the *To Do* information as follows:

   - Do not add M to the *To Do* information if M has been already visited (it appears in the *backtrack list*).
   - When adding M to the *To Do* information do the following:
     - For the DFS and BFS do not add, if the *To Do* information already contains the node M.
     - For the SP, add if the *To Do* information does not contain the node M. If it already contains the node M check if the new distance is shorter that the one already recorded in the list. If the new distance is shorter, replace the previous entry for M in the *To Do* information with the new one.

Backtracking algorithm

1. Remove the destination from the *backtrack list* and add it to the final path.
2. 2. Repeat: find the node you came from to get to the node added to the path.
3. Add it to the final path.
4. If it is the starting node, stop and print the routing, otherwise return to the step 2.