Introducation to Graph Theory

Definition and Representation

In mathematical and computer science terms, a graph is a collection of values (known as nodes or vertices) and a collection of connections, known as edges. This is frequently represented by endpoints and line segments connecting them. Mathematically a graph is a tuple of two sets, $G(E, V)$, an edge set and a vertex set. Graph functions as a template to abstract many problems, it represents relationships between similar constructs.

For the purposes of complexity analysis, the runtime is often expressed in terms of the size of the E and V sets, similar to how complexities of operations with 1D data structures is often in terms of the length. In most cases, we can make the assumption of $E \approx V^2$. This is because in a fully connected graph, the size of the edge set is equal to $V(V-1)$, and in contests it is often best to assume the worst unless there's a good reason for acting otherwise.

There are many types of graphs, and each of them are used to represent different relationships between values, and their difference is mostly on what the edges can be. To start off there's directed and undirected graphs, in directed graphs the edges only go in one direction whereas in undirected graphs the edges go in both. There are further restrictions on whether the edge length is specified, as well as the allowed range of the length, for example, some graphs call for negative edge lengths while others restrict the edge lengths to only positive. For the purposes of this lecture, we are going to deal with directed graphs with positive edge lengths.

I know that we usually don't go into implementation in this class. But the implementation of a graph can also help us visualize information. There are two major ways of representing graph:

- Adjacency Matrix: A 2D array of booleans (or edge lengths) where the indices are the nodes connecting the edge.
- Edge List: This involves a slight modification of the node data structure, which is usually just a 1D array, but this modification involves adding an additional list of connected vertices.

The advantage of using adjacency matrix is that it's much simpler to code up, the disadvantage is that it uses a lot of space. But I digress, back to graph algorithms.

Depth First Search and Applications

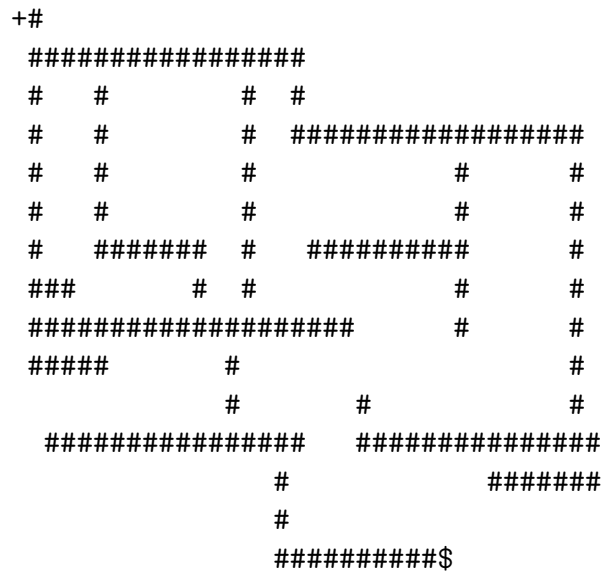
The most basic algorithm we will study today is the depth first search, otherwise known as depth first transversal. The point of this algorithm is to find a path from one node to another. Let's look through an example together.

We are going to solve a maze. A maze can be represented as a graph where each grid point represent a vertex and if two points are "adjacent," i.e. If you can reach one square from another, then they are connected by an edge in our representation.

The algorithm goes as follows, we keep a list of booleans corresponding to the list of vertices. This list marks if a square has been visited or not. We start by the beginning of the maze.

At each crossroads (aka each vertex with more than one edge coming out of it). We will recursively search along each of the paths until it reaches either an exit, in which case we're done, or a dead end, or a visited node. In either of the later cases we go one the next branch, hence earning the algorithm its name **depth first search (DFS)**, because it always going the full depth along each path.

This algorithm has the complexity of $O(E + V)$, as in the worst case we are guaranteed to visit every vertex, and in doing so we have to traverse every edge. Using the earlier substitution, the worse case scenario can be estimated to be about $O(V^2)$.



This algorithm is actually very versatile and has profound implications, for example, it can be used to detect if a graph is connected or have several disconnected pieces. This can be done by trying to traverse the graph with no endpoint, only seeking to mark all nodes as visited. If the call returns and there still exists unvisited nodes, those nodes are disconnected from the starting point, allowing us to subdivide an unconnected graph into connected pieces.

Another modification of this algorithm is cycle detection. Recall the few terminating conditions, another one we haven't used yet is reaching a previously visited node. Think about what this means. If we hit a previously visited node, that means there is a cycle in our graph starting at that node. This will also come in handy in problems.

Breath First Search and Meet In The Middle

An alternative method of transversal is known as breath first search or transversal. As its name suggests, unlike depth first search this method takes one step in all possible directions. This is a different algorithm but the basic tenement is the same. As a transversal algorithm, it can do everything depth first search can, including connectedness and cycle detection. This approach has two advantages. The first being that if we are looking for a path between two nodes, breath first search guarantees an optimal solution. The second is that it has a much better average time complexity than depth first search. Consider we are trying to get from point A to point B, but A branches into B and C, and in DFS, if we stumble into C, we will reach the worst case before we get to B, but with breath first search we are guaranteed to reach B in one step.

In breath first search, if the average degree of the graph is p and the true distance between the starting and ending positions is d, then $O(p^d)$ nodes will be traversed, whereas DFS is much more erratic and "luck based." However, the choice of which to use is highly context dependent. While BFS offers the potential for higher speed, it uses up a lot of memory, making it potentially unfeasible. It may also be the case that there are many desirable results and they are all far in the graph, in which case DFS will be preferable.

Meet in the Middle

Let's talk about an interview question. Given the name of two people on Facebook, how does one find their degrees of separation.

This is one of those cases where DFS would not solve the problem for rather obvious reasons. But let's talk about BFS, starting from friend A, if we were to find the distance to friend B via BFS, we will have to traverse $O(p^d)$ people as previously stated.

Sociology theory says that d is bounded by 6, and let's put a lower bound on p as, say, 150. This means that we will need to traverse about $150^6 = 11390625000000$ people. This is not good enough.

Consider, however, that we concurrently conduct BFS from both A and B. Each with their own unique visited-A and visited-B markings. Then if they encounter the visited marking of the other person, we are done. This effectively reduced the complexity of $O(p^d)$ to $O(2 \times p^{d/2})$, which further reduces to $O(p^{d/2})$, a drastic decrease. In our example it reduces the number of people traversed to $2 * 150^3 = 6750000$, which is much, much less.

Although we introduced this technique in the context of graph theory, this technique can be used in a variety of settings. However, this technique is usually known as the "smart brute-force" method, and you should only try to do it when there is no other algorithms.

Let's see an example outside the context of graphs, and yet another popular interview question, the four number problem!

The problem is as follows, you are given a list of integers with length n, find if there exists 4 number a, b, c, d s.t. $a + b + c + d = 0$.

The brute force solution is to try all combination of 4 numbers, which is $O(n^4)$. For those interview oldies, you probably know that "a hashtable makes everything better," so a slightly better solution is to hash all possible $-(a + b + c)$ and check that against the original array.

That last approach is incredibly close to the correct one, but as the name suggests, meet in the **middle** requires using symmetry to our maximum advantage. The insight is that $-(a + b) = (c + d)$. We still use a hashset, and store all possible $-(a + b)$'s, then we iterate through all possible $(c + d)$'s and check them against the subset. This has complexity of $O(n^2)$, which is better than either of the previous ones.