Jeffrey Wan Module 7 - Discussion Prompt Questions

Jot down short answers for each prompt question as you go along. Review and update your answers after finishing and submit before Tuesday midnight. Click on "Module 7 Discussion Prompt Questions" above to submit.

The prompt questions for this module are:

1. What attributes of a problem might you consider when deciding how to represent a graph?

A graph is a pictorial way to represent information and graph is a set of vertices V and a set of edges E, comprising an ordered pair G=(V, E). V is not empty and has size n. E may be empty so a single node is a graph. Graphs can model relationships that have loops or unconnected relationships. Graphs can have direction or be undirected too. Perhaps it could represent the last surviving person after the Coronavirus with no relationships to other people since he/she is the last survivor.

I think the relationships of the data is something to consider when deciding how to represent the graph. The relationships between the nodes can be the edges. So if nodes can be related to more than one node, then you might want to consider a graph with multiple edges. The number of relationships would affect the **degree** of the vertices.

Related question: When would I use a tree?
A tree is a subset of a graph where there are n nodes, n-1 edges, and the graph is connected by acyclic. That's what a tree is.

If the problem has a hierarchical relationship, it's best to use a tree.

But, trees are a kind of graph that can expression hierarchy. Trees have ancestors, children, siblings, cousins… if the problem has this sort of hierarchical relationship, then a tree is a good use case to model this problem.

2. What are some advantages of using a linked structure to represent a graph? Disadvantages?

- One advantage is that it's intuitive… each node has pointer(s) to the next node and has its own data.
- The dynamic allocation is useful because we don't have to set aside memory from the get go and the pointers just point to the next node in the graph.
- A disadvantage is that it's takes up more space because of the pointers? I'm actually not sure why a linked representation is worse here. I guess access time to the last vertices in hte worst case (say the graph is just a line of vertices) takes O(n) access time so that's one disadvantage. Whereas an array based implementation can just exploit the random access of the array to get the leaves.

To the grader: What are other disadvantages?

3. Why might an array be better (or worse) than a linked structure to represent a graph?

The array has a pattern for finding parent. For node at i, the parent is i/2. Children are at 2i, and 2i+1. Easy to find relationships in array based on index. For a node at index 2, the children are at index 4 and 5. For a node at index 11, the parent is at index 5. The relationships can be inferred from the indicies of the array so the structure of the array is more telling than the structure of a linked list. The random access of the array is faster when obtaining a value from the graph. However, if we're adding to the graph, the size of the array would have to be adjusted, and also adding nodes to the middle of the array would cause the other elements to shift which would result in greater time complexity, perhaps on average of O(n/2).

The linked implementation is a multi linked list each node has two child pointers: a left and a right pointer to different nodes. A linked implementation lends well to dynamic allocation and each node simply has a left and right pointer to point to the next vertex instead of preallocating space.

4. Can you name some applications for which a graph representation might be useful?
- Modeling a family tree. The relationships between vertices are modeled well using a tree. A family has children, parents, ancestors, cousins, siblings, and so does a tree.
- Representing a company where the root node is the CEO and each level down represents a lesser degree of power or shares down to the most junior employee which would be leaf. Internal nodes are implicitly managers with children nodes who they manage.

5. What aspects of graphs make them a good (or poor) match for recursive solutions?

I guess at the heart of it all, a larger graph is composed from smaller subgraphs.

- Each node is identical to the next node in the sense that each node has a pointer and data. For example, for inserting, you can just keep calling the insert method until you find a node that that doesn't have a pointer (the base case) in which case you can point the node to the new inserted node. If the node that you pass the method has a pointer, you simply call the method again with the node that the passed node is pointed to.
- Each node is identical and so since the insert method takes a node, you can simply call it with any node. The base case is the when a node is passed without another node that it points to.
- Trees are also built up of other smaller sub trees and so also lend well to recursive definitions. There's a root node, and then under it are left subtrees and right subtrees which are smaller versions of the original problem… that smells like it would lend well to recursion.