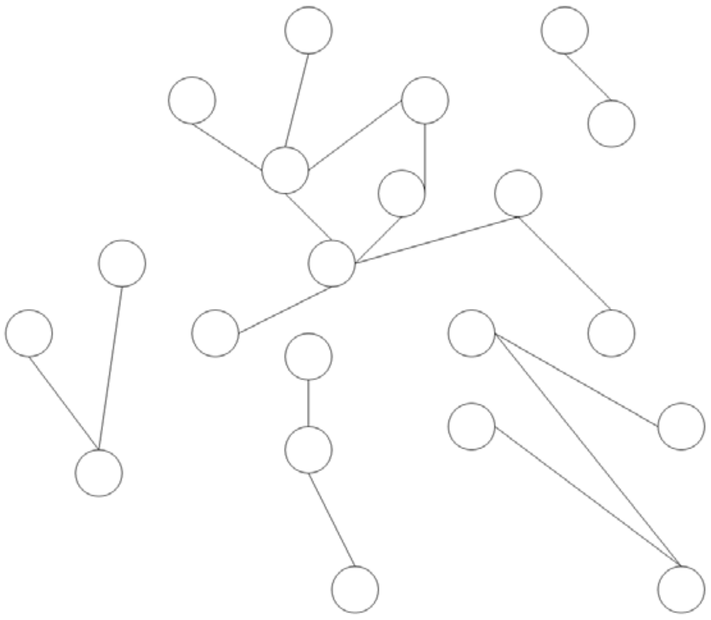
CS1102: Data Structure and Algorithms

Tutorial 11 - Graphs

Week of 12 April 2010

1. **Connected components**



a. A connected component is a set of vertices where there exists a path between all vertices in the component. How many connected components do you see in the undirected graph shown above? How many vertices belong to the largest component?

b. The pseudocode

```
//main loop
for (each vertex v in the graph)
        if (v has not yet been processed)
                recursiveMethod(v);
…
recursiveMethod(v)
{
        mark v as processed;
        for (each v’ adjacent to v)
                if (v’ has not yet been processed)
                        recursiveMethod(v’);
}
```

works on every vertex in a graph with **n** vertices and **e** edges. We assume the line “mark **v** as processed;” inside the recursiveMethod() where we process each **v** takes O(1) time to finish.

Suppose that the graph is represented as an array of adjacency lists. In "big-Oh" terms involving **n** and **e**, give the best estimate for the running time of the algorithm just given.

c. Now suppose that the graph is represented as an adjacency matrix. Give the best estimate for the running time of the algorithm given in 1.b.

d. How do you determine how many connected components are there in a graph? How do you determine how many vertices are in the largest connected component? You may want to modify the pseudocode in 1.b.

2. Is This Graph a Tree?

Definition: Tree T is a special graph G with *either one* the following properties hold:

A. G is connected and |E| = |V| - 1
B. G is acyclic and |E| = |V| - 1
C. There exists a unique path between every pair of vertices in G

Create a simple pseudo code to determine whether a given graph G (a set of vertices and edges) is actually a tree using property A (the simplest), i.e. “bool IsTree(Graph G)”.

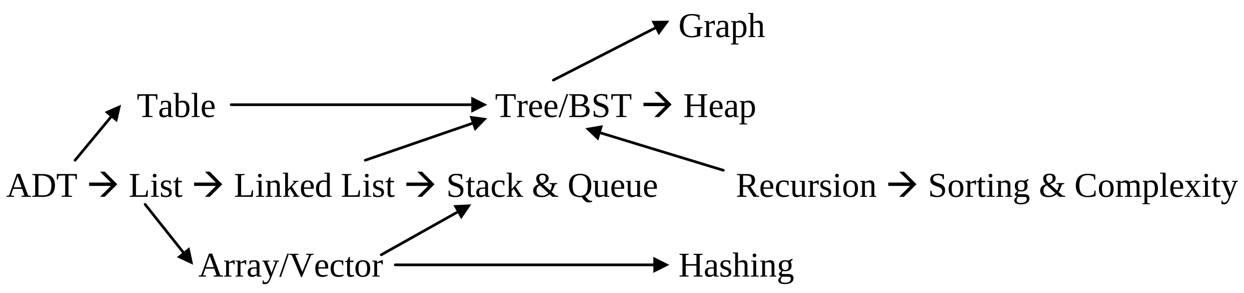
You can try property B or C on your own. Assume that the graph is stored as Adjacency List and the size of |V| and |E| are given.

Hint: What are the nodes reachable from a single call of DFS (starting vertex v)?

CS1102: Data Structure and Algorithms

3. Topological Sort

The graph below contains information about (some) CS1102 materials and their dependencies, i.e. the materials must be taught "in certain order" so that students can understand the materials. For example, the concept of linked list must be taught before tree data structure, graph must be taught after tree, etc. *Note that this sample graph does not model course materials perfectly! Answer the questions with respect to THIS graph!*


Graph
Table
Tree/BST → Heap
ADT → List → Linked List → Stack & Queue
Recursion → Sorting & Complexity
Array/Vector
Hashing

Suppose your lecturer/TA have taught this course for many years and wants to have a fresh challenge by reordering the syllabus. However, given the dependency constraints like the one shown in the graph above, he cannot do much.

a. Use **topological sort algorithm** taught in class to come up with one valid topological order. Topological order is a linear ordering of these CS1102 materials that satisfy the dependency constraint. If when there are two possible choices (ties), we choose the one with **lower lexicographic order**, e.g. after processing Tree/BST, then "Graph" and "Heap" become available, enqueue "Graph" first as "G"raph is lexicographically smaller than "H"eap.

b. The result of part a), if done correctly, will be quite different from the current CS1102 syllabus. Please suggest few edge constraint(s) in the dependency graph to force all "List" data structures discussed before "Table" data structures.
   - "List" data structures include "ADT List", "Array/Vector", "Linked List", and "Stack & Queue" as we implement them using array/vector or linked list.
   - "Table" data structures include "ADT Table", "Tree/BST", and "Hashing".

c. Please list down <u>two</u> additional *logical* orderings of these CS1102 materials according to the given graph plus your modifications in part b) above. Note that you do NOT need to use the topological sort algorithm taught in lecture.