Pre-tutorial questions

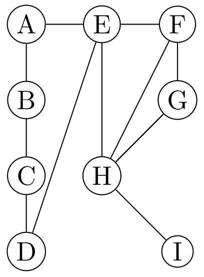
Do you know the basic concepts of this week's lecture content? These questions are only to test yourself. They will not be explicitly discussed in the tutorial, and no solutions will be given to them.

1. Graph terminologi
   (a) What is a graph $G(V, E)$?
   (b) Graphs are usually represented either as an adjacency matrix or as an adjacency list. Can you explain the two representations?
   (c) What are the advantages/disadvantages between the two different representations?
   (d) What is a simple path in a graph?
   (e) What is a cycle in a graph?
   (f) What is a tree? If a tree has n vertices, how many edges does it have?
   (g) What's the difference between a rooted tree and an unrooted tree?
   (h) What is a bipartite graph?
2. Graph traversals
   (a) What's the difference between BFS and DFS?
   (b) Explain the two search algorithms BFS and DFS.

Tutorial

Problem 1

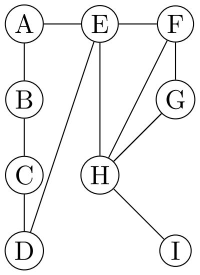
Run a Breadth First Traversal for graph G starting at A. Write the order the nodes are explored. Assuming breaking ties lexicographically (i.e., if more than one child pick the one that comes first in lexicographic order).



Solution:
A
B,E
C,D,F,H
G, I

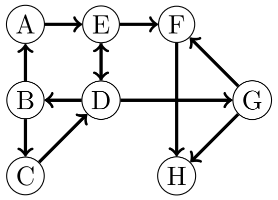
Problem 2

Run a Depth First Traversal for graph G starting at A. Write the order the nodes are explored. Assuming breaking ties lexicographically.



Solution:
A
B
C
D
E
F
G
H
I

Problem 3
Consider the following directed graph G



1. Give the adjacency matrix and the adjacency list representation of this graph
2. Is this graph strongly connected?
3. This graph has a few directed cycles. Find the minimum number of edges that you need to delete from this graph in order to make it a directed acyclic graph (DAG).
4. Write down the transitive closure of the graph.
5. [**Advanced**] Write down a topological ordering of the resulting DAG.

Solution:

1. adjacency matrix:

```
  A B C D E F G H
A 0 0 0 0 1 0 0 0
B 1 0 1 0 0 0 0 0
C 0 0 0 1 0 0 0 0
D 0 1 0 0 1 0 1 0
E 0 0 0 1 0 1 0 0
F 0 0 0 0 0 0 0 1
G 0 0 0 0 0 1 0 1
H 0 0 0 0 0 0 0 0
```

   adjacency list:

```
A: E
B: A,C
C: D
D: B,E,G
E: D,F
F: H
G: F,H
H:
```

2. No, H is a sink. There is no path from H to any other node. Note that you do not need to check all possible ordered pairs of vertices to see if the graph is connected. Just pick one vertex (any will do) and check if that particular vertex has a path to every other vertex and every other vertex has a path back to it.

3. delete edges (D,B) and (D,E) to get a DAG.

4. A has outgoing edges to all other vertices

   B has outgoing edges to all other vertices

   C has outgoing edges to all other vertices

   D has outgoing edges to all other vertices

   E has outgoing edges to all other vertices

   F has one outgoing edge to H

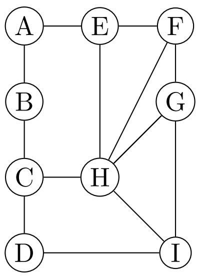
   G has outgoing edges to F, H

   H has no outgoing edges.

5. In the resulting graph, B is a source. Add to the sequence as the first node and delete it from the graph along with all edges incident to it. Now A is a source (as well as C, we go with the lexicographically first- but any choice is fine in breaking symmetry/ties), then take C, followed by E, D, G, F, H. So the topological order is B,A,C,D,E,G,F,H. You can try the same by starting with sinks.

Problem 4

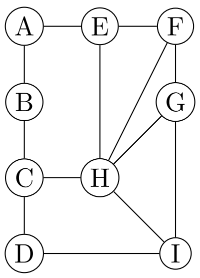
Run a Breadth First Search for graph G starting at A and searching for I. Write out the path that is explored. Assuming breaking ties lexicographically.



Solution:
A
B,E
C,F,H
D,G,I

Problem 5

Run a Depth First Search for graph G starting at A and searching for I. Write out the path that is explored. Assuming breaking ties lexicographically.



Solution:
A
B
C
D
I

Problem 6
An undirected graph $G = (V, E)$ is said to be bipartite if its vertex set V can be partition into two sets A and B such that $E \subseteq A \times B$. Design an $O(n+m)$ algorithm to test if a given input graph is bipartite using the following guide:

1. Suppose we run BFS from some vertex $s \in V$ and obtain layers $L_1, \ldots, L_k$. Let (u, v) be some edge in E. Show that if $u \in L_i$ and $v \in L_j$ then $|i - j| \leq 1$.

2. Suppose we run BFS on G. Show that if there is an edge (u, v) such that u and v belong to the same layer then the graph is not bipartite

3. Suppose G is connected and we run BFS. Show that if there are no intra-layer edges then the graph is bipartite

4. Put together all the above to design an $O(n + m)$ time algorithm for testing bipartiness.

Solution:

- If $i = j$ then we are done. Otherwise, assume without loss of generality that u is discovered by BFS before v; in other words, assume $i < j$. When processing u, BFS scans the neighborhood of u. Since v ends up in layer L_j and $j > i$, this means that v had not been discovered yet at the time we started processing u. But then that means that v will be placed in the next layer; in other words, $j = i + 1$, and the property follows.

- Suppose $u, v \in L_i$ and $(u, v) \in E$. Then we can form an odd cycle by taking the shortest path from s to u, the edge (u, v) and the shortest path from v to s; this cycle has length $2i + 1$. Now if the graph was bipartite the vertices in the cycle should alternate between the A set and the B set, but that is not possible if the cycle has odd length.

- Let A be the even layers $L_0, L_2, \ldots$, and let B be the odd layers $L_1, L_3, \ldots$. Because the graph is connected every vertex belongs to some layer, so (A, B) is a partition of V. Because there are no intra-layer edges, we have $E \subseteq A \times B$.

- Given an input graph G, find its connected components and for each component we run BFS, check if there are intra-layer edges; if not, partition the vertices into odd and even layer vertices.

The correctness of the algorithm follows readily from the previous tasks.
The running time is dominated by the BFS computation, which takes $O(n + m)$ time.

Problem 7
Give an $O(n)$ time algorithm to detect whether a given undirected graph contains a cycle. If the answer is yes, the algorithm should produce a cycle. (Assume adjacency list representation.)

Solution: We run DFS with a minor modification. Every time we scan the neighborhood of a vertex u we check if the neighbor v has been discovered before and whether it is different than u's parent. If we can find such a vertex the we have our cycle: $v, u, \text{parent}[u], \text{parent}[\text{parent}[u]], \ldots, v$.
We only need to argue that this algorithm runs in $O(n)$ time. Consider the execution of the algorithm up the point when we discovered the cycle. After the $O(n)$ time spent initializing the arrays needed to run DFS, each call to DFS-VISIT takes time that is proportional to the edges discovered. However, up until the time we find the cycle we have only discovered tree edges. So the total number of edges is upper bounded by $n - 1$. Thus, the overall running time in $O(n)$.

Problem 8
[Advanced] In class we sketched a proof that a Directed Acyclic Graph always has a topological ordering. Write out a formal proof using induction.

Solution: The base case when $n = 1$ is trivial. The induction hypothesis is that for any DAG with k vertices, there is a topological ordering. The induction step is to show that the claim therefore must hold for a graph with $k + 1$ vertices. So consider a DAG with $n = k + 1$ vertices. As argued in class, at least one of these $k + 1$ vertices has no incoming edges. So take one such vertex (there may be many) and call it v_n. Define $G = (V, E)$ where $V = V - v_n$ and E is the set of edges in E with the edges incident to v_n removed. But G is obviously a DAG too, and it has k vertices. So by the induction hypothesis, it has a topological ordering, namely $(v_1, \ldots, v_{n-1})$. This gives a topological ordering for the original graph G, namely $(v_n, v_1, \ldots, v_{n-1})$