CS2223, HW2: Graphs and Graph Search[*]

Course website: `http://web.cs.wpi.edu/~cs2223/b12/`. Hand in your answer by midnight, 7 Nov., so that we can discuss some problems in class Thursday. Use Turnin at `https://turnin.cs.wpi.edu/`.

Some *CLRS* problems and exercises are included in this week's list.

Working in groups and talking about the problems is strongly encouraged. More enjoyable and more educational. You can also discuss them with Fei, Linglong, Xianjing, and with me. Our office hours are listed at `http://web.cs.wpi.edu/~cs2223/b12/#personnel`.

A. Run DFS on an Example. Execute DFS on this example graph G_1, showing the start and finish times for each node. Assume DFS always starts from the alphabetically first node, and then—when it has processed everything accessible from there—restarts at the alphabetically next unvisited node, etc.

When it has several edges out from a single node, assume that it always uses them in alphabetical order (by their destinations). Put the start and finish time (separated by a comma) next to the node.

$$\begin{tikzcd}
G_1 & & b \circ \arrow[rr] \arrow[dr] & & c \circ & \\
& a \circ \arrow[ur] \arrow[dr] & & e \circ \arrow[rr] & & f \circ \\
& & d \circ \arrow[ur] & & g \circ &
\end{tikzcd}$$

Check that the parenthesis property holds (see *CLRS*, Corollary 22.8, p. 608).

a: 1, 12, b: 2, 9, c: 3, 4, d: 10, 11, e: 5, 8 f: 6, 7, g: 13, 14.

[*]Due: Wednesday night, 7 Nov.

B. Topological Sort. Suppose G is an *directed acyclic graph*, and we would like to order its nodes in a sequence S such that every edge in G points from an earlier node to a later node: If there is an edge from u to v in G, then u comes before v in the sequence S.

Finding a linear sequence of this kind is called (oddly) *topologically sorting* G. You could also call it "linearizing" G.

Depth first search is one way to sort G topologically. DFS finds the discovery time and the finish time for each node in G. Each time you finish a node, place it in an array S, but fill the array starting from the back, at S[`g.length`], working forward, so that the last node to finish is in $S[1]$.

B.1. Extract the Topological Sort. Write the topologically ordered sequence of nodes S generated by the run of DFS on G_1 from Part A. g, a, d, b, e, f, c

B.2. Other Topological Sorts. There are generally a lot of sequences that are possible topological sorts of the same graph. Some of them could be generated by a DFS that visits the nodes in a different order. But some of them cannot be generated by DFS.

1. Write out a sequence that *could* be generated by DFS on G_1, if it sometimes selects unvisited nodes out of alphabetical order, or sometimes chooses edges out of a single node out of alphabetical order.

   a, d, b, e, f, c, g

2. Write out a sequence that is a topological sort of G_1, but could *never* be generated by DFS, no matter what order it selects its starting points and its edges.

   Ahem. What I was thinking when I wrote these words was, no matter what order it selects its starting points after a. Or something like that. Though, this isn't what the words actually *say*.

   If I had asked that question, an answer would be: a, d, g, b, e, f, c. Others are possible.

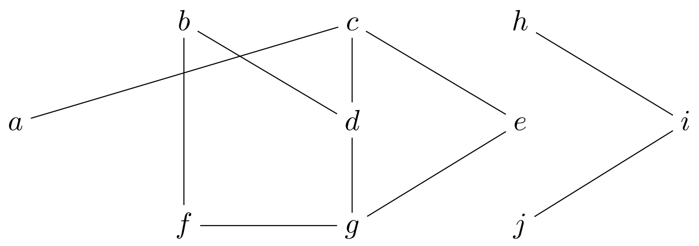
B.3. Prove the algorithm is right. How do we know that using DFS this way always gives a topological sorting? Use the key properties of DFS to show that on an acyclic graph, this will give a topological sort.

Suppose that $f(i) < f(j)$, meaning that j comes before i in the output S. We must show that there is no edge from i to j.

Use the parenthesis property. Then either $d(j) < d(i) < f(i) < f(j)$ or else $d(i) < f(i) < d(j) < f(j)$. In the first case, when j was discovered, there was an undiscovered path from j to i. Then there is certainly no edge (i, j) by acyclicity.

Otherwise, there is no edge (i, j) because j is still undiscovered when i is discovered, but DFS does not explore it till after i finishes. If there was an edge, i could not finish until this edge is explored.

C. Run BFS on an Example. Execute BFS on this example undirected graph G_2. Start at node a. Next to each node, write $d =$ its distance from node a, counting by edges, and $\pi =$ the parent it was reached from. When the first run completes, restart with the alphabetically earliest node you have not yet seen.



Check that if u, v are any two vertices with an edge between them, then their distances are separated by at most 1.

$a : 0 \quad c : 1, a \quad d, e : 2, c \quad b, g : 3, d \quad f : 4, b$

$h : 0 \quad i : 1, h \quad j : 2, i$

D. Bipartite Graphs. Using your BFS traversal of G_2 as an example, say:

1. Is G_2 bipartite? Yes.
2. List the nodes in each of the two classes that traversal identifies, if G_2 is bipartite.

   I. a, d, e, f, h, j. II. b, c, g, i. Could also have h, j in II and i in I.
3. Show an edge between two nodes in the same layer, otherwise.

E. Some exercises from *CLRS*. Do these exercises from the textbook. Discussion is warmly encouraged.

Representing graphs, 22.1. On pp. 592–3, do exercises 22.1-1, 22.1-2, 22.1-5, and, if it looks fun, 22.1-6.

BFS, 22.2. On pp. 601–2, do exercises 22.2-1, 22.2-2, 22.2-5, 22.2-7.

DFS, 22.3. On pp. 610–11, do exercises 22.3-2, 22.3-3, 22.3-5, 22.3-7.

Topological sort, 22.4. On pp. 614–15, do exercises 22.4-1, 22.4-3.