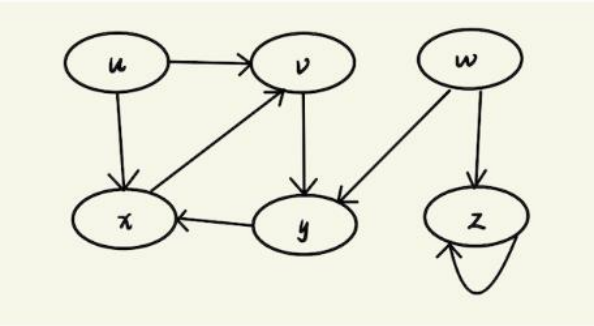
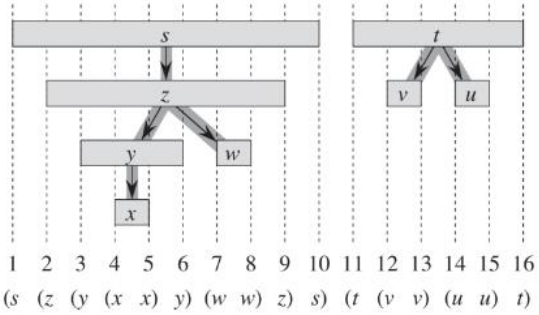
EL9343 Homework 8

Due: Nov. 10th 11:00 a.m.

1. What is the running time of DFS if the graph ($G = (V, E)$, where there are $|V|$ nodes and $|E|$ edges) is given as an adjacency list and adjacency matrix? Justify your running time.
   Solution:
   For adjacency list: $O(|V| + |E|)$; for adjacency matrix: $O(|V|^2)$.
   In DFS, we will visit every node once and visit those nodes are adjacent to them. In both case, we first need $O(|V|)$ time to visit all nodes.
   If we use adjacency list, we only need to visit every edge in each list, which is $O(|E|)$. So, the total running time is $O(|V|) + O(|E|) = O(|V| + |E|)$.
   If we use adjacency matrix, although there are only $|E|$ edges, once we visit a node (i.e. the i-th node), we will visit the element in the i-th row of the adjacency matrix. In total, we will visit every element in the adjacency matrix (no matter it is 0 or 1), which is $|V|^2$. So the total running time will be $O(|V|^2) + O(|V|) = O(|V|^2)$.

2. Draw the parenthesis structure of the DFS of bottom left figure (start from u, assume that DFS considers vertices in alphabetical order) and see the example parenthesis structure as is shown in the bottom right.





   Solution:
   (u (v (y (x x) y) v) u) (w (z z) w)

3. **Bipartiteness**. Given a undirected graph $G = (V, E)$, it is *bipartite* if there exist U and W such that $U \cup W = V$, $U \cap W = \emptyset$, and every edge has one endpoint each in U and W.
   (a) Prove: G is bipartite only if G has no odd cycle. (Hint: proof by contradiction)
   (b) In fact, G is bipartite if and only if G has no odd cycle. Suppose this is given, consider the Algorithm 1 and briefly describe why it is correct.
   (c) Analyze the time complexity (worst-case, big-O) of the algorithm, in terms of $|V|$ and $|E|$. (Hint: can we check that there is no edge inside any layer in $O(|E|)$? Why?)

Algorithm 1 Testing bipartiteness of graphs

Do BFS starting at any node u
Let $L_0, L_1, L_2, \ldots, L_k$ be layers in the resulting breadth-first tree ($L_0 = \{u\}$, $L_i, i = 1, 2, \ldots, k$ contains the vertices at distance i from u)
if There is no edge inside any layer L_i **then**
  Declare G to be bipartite, and $U = L_0 \cup L_2 \cup L_4 \cup \ldots, W = L_1 \cup L_3 \cup L_5 \cup \ldots$ are the bipartition
else
  Declare G to be non-bipartite
end if

Solution:

(a) Proof by contradiction, we assume G is bipartite when there is an odd cycle in G.
Let this cycle be $u_1 \to u_2 \to u_3 \to \cdots \to u_{2k+1} \to u_1$. Since G is bipartite, let the bipartition be U and W, and we can assign $u_1, \ldots, u_{2k+1}$ to either U or W.
Without loss of generality, we let $u_1 \in U$. There is an edge between u_1 and u_2. Because of bipartiteness, u_2 must be in W. Similarly, $u_3 \in U, u_4 \in W, \ldots$.

Finally, $u_1, u_3, \ldots, u_{2k+1} \in U, u_2, u_4, \ldots, u2k \in W$, and there is an edge between u_{2k+1} and u_1. However, u_{2k+1} and u_1 are all in set U, which contradicts to the assumption that G is bipartite (every edge has one endpoint each in U and W).
Thus, G is bipartite only if G has no odd cycle.

(b) In BFS, it is impossible that there is an edge between a node in L_i and a node in L_j, where $|i-j| \geq 2$. If there is no edge inside any layer L_i, consider the partition $U = L_0 \cup L_2 \cup L_4 \cup \ldots, W = L_1 \cup L_3 \cup L_5 \cup \ldots$: there will be no edges inside U or W, and every edge $e \in E$ will have one endpoint each in U and W. Therefore, the graph G is bipartite.
If there is an edge inside a layer, let's say this edge $e = (u_1, u_2), u_1, u_2 \in L_k$, and consider their lowest common ancestor in the breadth-first tree $s \in L_i, i < k$, $u_1 \to s \to u_2 \to u_1$ will be a odd cycle, meaning that the graph G is not bipartite.
Note: To get full mark, one needs to briefly show **both** no edge inside any layer $\to$ bipartite, **and** edge exists inside a layer $\to$ non-bipartite. It's also safe to show the first part this way: no edge inside any layer $\to$ all cycles must be even cycles $\to$ no odd cycle $\to$ bipartite. Detailed proof of correctness is not necessary but welcomed (since you may need it later in dynamic programming and greedy algorithms).
Consider a cycle in this case, choose two endpoints u_1, u_2, and suppose $u_1 \in L_i, u_2 \in L_j, i \leq j$ in the breadth-first tree. There must exist two different paths (two arcs) from u_1 to u_2, call them l_1 and l_2. Since there are only edges going from L_k to L_{k-1} or L_{k+1}, the length of l_1 and l_2 could only be $2t_1 + j - i$ and $2t_2 + j - i$, meaning that the length of the cycle is $2t_1 + 2t_2 + 2(j - i)$, is even.

(c) The time complexity is $O(|V| + |E|)$. Doing the BFS is $O(|V| + |E|)$, checking that there is no edge inside any layer is $O(|E|)$, and anything else is $O(1)$. Therefore, it is $O(|V| + |E|)$ in total.
We can check each edge in set E on the two layer index of its two endpoints (for each node, its layer index is i if the node is in layer L_i), which is $O(|E|)$. If every pair of the two endpoints are in different layers, we are done. This is faster than checking if an edge exists in every pair of nodes inside a layer, which in worst case is $O(|V|^2)$.

4. You're helping a group of ethnographers analyze some oral history data they've collected by interviewing members of a village to learn about the lives of people who've lived there over the past two hundred years. From these interviews, they've learned about a set of n people (all of them now deceased), whom we'll denote $P_1, P_2, \ldots, P_n$. They've also collected facts about when these people lived relative to one another. Each fact has one of the following two forms:

   (a) For some i and j, person P_i died before person P_j was born; or

   (b) for some i and j, the life spans of P_i and P_j overlapped at least partially.

   Naturally, they're not sure that all these facts are correct; memories are not so good, and a lot of this was passed down by word of mouth. So what they'd like you to determine is whether the data they've collected is at least internally consistent, in the sense that there could have existed a set of people for which all the facts they've learned simultaneously hold.
   The original problem asks you to give an efficient algorithm to either produce proposed dates of birth and death for each of the n people so that all the facts hold true, or report (correctly) that no such dates can exist — that is, the facts collected by the ethnographers are not internally consistent. Since topological sort will be taught next week, now let's do make it simple. **You only need to give an efficient algorithm to determine whether the data is internally consistent.**
   Hints: Let's say there are totally m recorded facts, and your algorithm should run in $O(m+n)$. It should be useful to turn the data into a graph. If each node represents an event (birth or death of someone), what is the total number of nodes? How to define the edges, directed or undirected? What makes the data inconsistent? What is the key structure to check in the graph?
   Solution:
   Create a directed graph which has 2 vertices associated with each person, b_i and d_i, denoting birth date and death date. For each person, add a directed edge from b_i to d_i, implying chronological order. For each fact of the form i died before j was born, add an edge from d_i to b_j. For each fact of the form lives of i and j overlapped, add edges from b_i to d_j and from b_j to d_i. (Their lives overlap if and only if each of them was born before the other died.) Now each edge from A to B represents the partial order of event A "happens before" B. For the causality reason, there should not be any cycle inside the directed graph. Check if there is a cycle, if so, return "inconsistent". Otherwise, return "consistent". To get an

order of births and deaths, simply run a topological sort.

Notes:

(a) You must clearly describe how to construct edges. A common mistake is to forget the edges from b_i to d_i. Without these edges, a problematic collection of records may also be acyclic.

(b) Just marking visited or not is not enough for cycle detection using DFS. The only indication for the existence of cycles is a back edge. Please refer to the slides and book if you are not clear on this. BFS cannot detect cycles efficiently in a directed graph. Running multiple times of DFS is not efficient (unless you could prove the complexity bound).

(c) If we don't ask for codes, you don't have to write the algorithm in codes. Just describe your algorithm clearly in words. If you want to practice on code writing in python or in c++, you can first run your codes with some simple examples generated on computer. Don't write your code half in python and the other half in c++. Points will be deducted if you are writing codes incorrectly.