CS3510-A: Design and Analysis of Algorithms, Spring 2021

Homework-3

February 2, 2021

DUE DATE: Tuesday, February 9, 1l:59pm

Note-1: Your homework solutions should be electronically formatted as a single PDF document that you will upload on Gradescope. If you have to include some handwritten parts, please make sure that they are very clearly written and that you include them as high resolution images.
Note-2: Please think twice before you copy a solution from another student or resource (book, web site, etc). It is not worth the risk and embarrassment.
Note-3: You need to **explain/justify** your answers. Do not expect full credit if you just state the correct answer.
Note-4: You will get 2 extra points if you submit electronically typed solutions instead of hand-written.

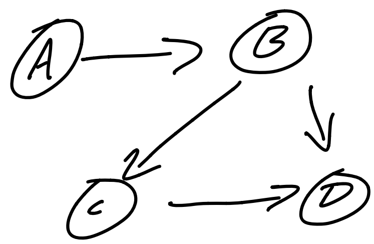
Problem-1 (30 points)

You are given a DAG. How would you check in linear time if the DAG includes a Hamiltonian path, i.e., a (directed) path that traverses every vertex exactly once?

Note 1: Please provide a detailed description of your solution in plain English or in pseudocode.
Note 2: You are also allowed to use BFS and DFS as blackbox algorithms (you do not need to explain how it works if you use it) so long as you detail the inputs and outputs to the algorithms as well as any modifications you make.

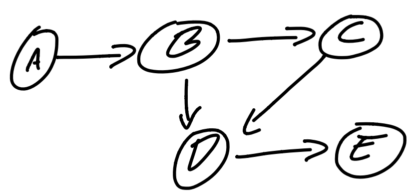
Solution

Topological Sort:



A B C D

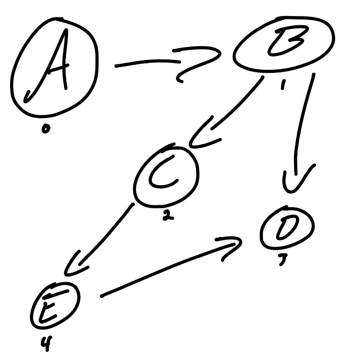
```
out = []
explore(G, n):
    explored(n) = True
    for each u in edge(n):
        if not explored(u):
            explore(u)
    if edge(n) = ∅:
        out = [n] + out
```



A B C D E
0.0 1.0 1.2 1.1 1.11

A B C D E

G =



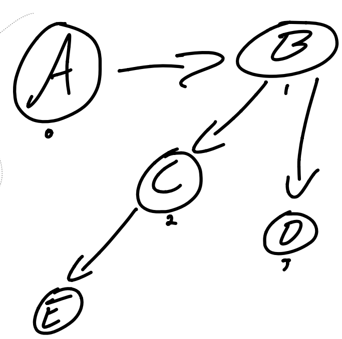
A B C E D
0 1 2 4 3

Code doesn't account for graph where Hamiltonian path doesn't exist

Checks over path that all nodes are connected

Checks longest depth on DFS?
Find root node by inverted DFS?
Find a sink node?

No Ham Path



Code Returns:

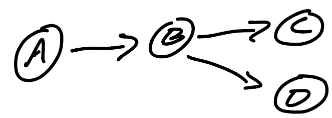
A B C D E
0 1 2 3 4

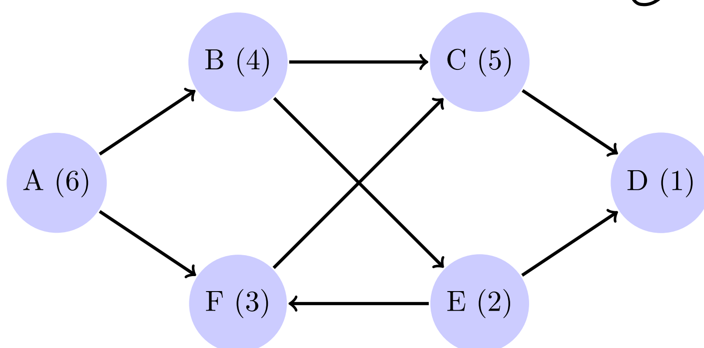
Problem-2 (35 points)

You are given a directed graph in which each node $u \in V$ has an associated capacity c_u which is a positive integer. Define the array *bottleneck* as follows: for each $u \in V$,

$$bottleneck[u] = \text{capacity of the lowest-capacity node reachable from } u \text{ (including } u \text{ itself).}$$

For instance, in the graph below (with capacities shown inside each vertex), the bottleneck values of all nodes are equal to 1. Your goal is to design an algorithm that fills in the entire bottleneck array (i.e., for all vertices).





(a) Design a linear-time algorithm that works for DAGs.

(b) Extend your previous answer to a linear-time algorithm that works for all directed graphs.

Note 1: Please provide a detailed description of your solution in plain English or in pseudocode.

Note 2: You are also allowed to use BFS and DFS as blackbox algorithms (you do not need to explain how it works if you use it) so long as you detail the inputs and outputs to the algorithms as well as any modifications you make.

Solution

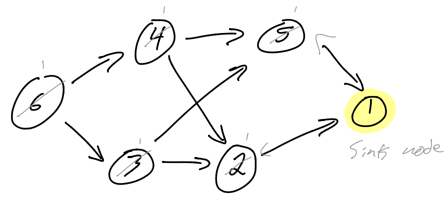
- Find all sink nodes
- propogate sink values backwards to update
  - curr value = min from sinks
  - if node value >= curr value:
    - update node val to current
    - continue on curr node connections
  - else if node value < curr value:
    - curr value = node value

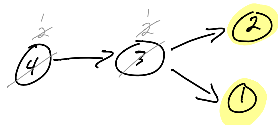
loop over graph

Runtime: T(DFS) · X

X: # of sink nodes in the graph

Curr: 1





2: 2

1: 1 Update

5: 1

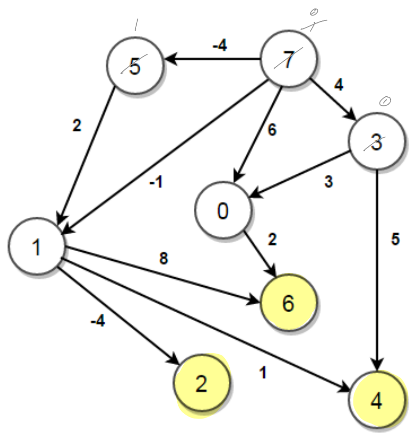
7: 1

6: 6

0: 0 Update

7: 0

4: No moves



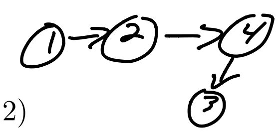
Problem-3 (35 points)

You're helping some security analysts monitor a collection of networked computers, tracking the spread of an online virus. There are n computers in the system, labeled $C_1, C_2, ...C_n$, and as input you're given a collection of trace data indicating the times at which pairs of computers communicated. Thus the data is a sequence of ordered triples (C_i, C_j, t_k); such a triple indicates that C_i and C_j exchanged bits at time t_k. There are m triples total.

We'll assume that the triples are presented to you in sorted order of time. For purposes of simplicity, we'll assume that each pair of computers communicates at most once during the interval you're observing.

The security analysts you're working with would like to be able to answer questions of the following form: If the virus was inserted into computer C_a at time x, could it possibly have infected computer C_b by time y? The mechanics of infection are simple: if an infected computer C_i communicates with an uninfected computer C_j at time t_k (in other words, if one of the triples (C_i, C_j, t_k) or (C_j, C_i, t_k) appears in the trace data), then computer C_j becomes infected as well, starting at time t_k. Infection can thus spread from one machine to another across a sequence of communications, provided that no step in this sequence involves a move backward in time. Thus, for example, if C_i is infected by time t_k, and the trace data contains triples (C_i, C_j, t_k) and (C_j, C_q, t_r), where $t_k < t_r$, then C_q will become infected via C_j. (Note that it is okay for t_k to be equal to t_r; this would mean that C_j had open connections to both C_i and C_q at the same time, and so a virus could move from C_i to C_q.)

For example, suppose $n = 4$, the trace data consists of the triples:



$$(C1, C2, 4), (C2, C4, 8), (C3, C4, 8), (C1, C4, 12)$$

and the virus was inserted into computer C_1 at time 2. Then C_3 would be infected at time 8 by a sequence of three steps: first C_2 becomes infected at time 4, then C_4 gets the virus from C_2 at time 8, and then C_3 gets the virus from C_4 at time 8. On the other hand, if the trace data were:

$$(C2, C3, 8), (C1, C4, 12), (C1, C2, 14)$$

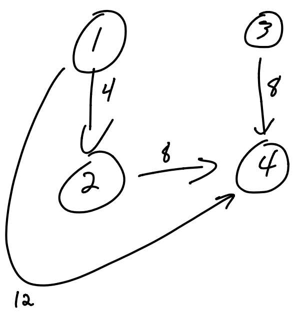
and again the virus was inserted into computer C_1 at time 2, then C_3 would not become infected during the period of observation: although C_2 becomes infected at time 14, we see that C_3 only communicates with C_2 before C_2 was infected. There is no sequence of communications moving forward in time by which the virus could get from C_1 to C_3 in this second example.

Design an algorithm that answers questions of this type: given a collection of trace data, the algorithm should decide whether a virus introduced at computer C_a at time x could have infected computer C_b by time y. The algorithm should run in time $O(m + n)$.

Note 1: Please provide a detailed description of your solution in plain English or in pseudocode. Also, explain the time complexity and correctness of the prescribed algorithm.

Note 2: You are also allowed to use BFS and DFS as blackbox algorithms (you do not need to explain how it works if you use it) so long as you detail the inputs and outputs to the algorithms as well as any modifications you make.

Solution Contact Tracing Algo



C_1 @ 2
got to C_4 @ 10 ?

$x = 2$
$y = 12$

only take paths: p
where: $x < p < y$

DFS from C_a to C_b
Only w/ paths when
$x <$ weight $< y$

- Use communication data to build weighted + directed graph where weights are times of communication (can be done in $O(n+m)$)
- Do a DFS from C_a to C_b, but only explore paths with $x \leq$ weight $\leq y$ (has $O(DFS) = O(n+m)$)
  - If path is not possible, then return false