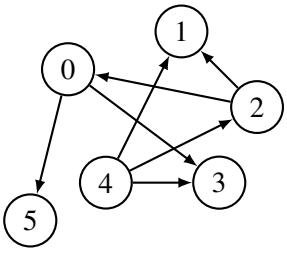
Ravi Daryabhai

Problem Session 6

Problem 6-1. Topological Training

Please answer the following questions about the unweighted directed graph G below, whose vertex set is $\{0, 1, 2, 3, 4, 5\}$.



(a) State a topological ordering of G. Then state and **justify** the number of distinct topological orderings of G.

(b) State a single directed edge that could be added to G to construct a simple[1] graph with no topological ordering. Then state and **justify** the number of distinct single edges that could be added to G to construct a simple graph with no topological ordering.

Problem 6-2. Never Return

Bimsa is a young lioness whose evil uncle has just banished her from the land of Honor Stone, proclaiming: "Run away, Bimsa. Run away, and never return." Bimsa has knowledge of all landmarks and trails in and around Honor Stone. For each landmark, she knows its x and y coordinates and whether it is inside or outside of Honor Stone. For each trail, she knows its positive integer length and the two landmarks it directly connects. Each landmark connects to at most five trails, each trail may be traversed in either direction, and every landmark can be reached along trails from the **gorge**, the landmark where Bimsa is now. Bimsa wants to leave Honor Stone quickly, while only traversing trails in a way that **never returns**: traversing a trail from landmark a to landmark b never returns if the distance[2] from a to the gorge is strictly smaller than the distance from b to the gorge. If there are n landmarks in Honor Stone, describe an $O(n)$-time algorithm to determine a shortest route that never returns, from the gorge to any landmark outside of Honor Stone.

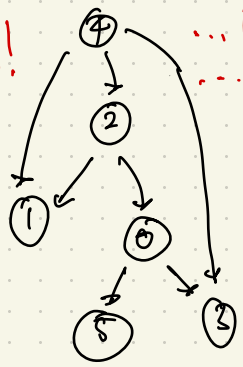
Problem 6-3. DigBuild

Software company Jomang is developing **DigBuild**, a new 3D voxel-based exploration game. The game world features n collectable block types, each identified with a unique integer from 1 to n. Some of these block types can be converted into other block types: a conversion (d_1, b_1, d_2, b_2) allows d_1 blocks of type b_1 to be converted into d_2 blocks of type b_2; each conversion is animated in the game by repeatedly dividing blocks in half, then recombining them, so d_1 and d_2 are always constrained to be integer powers of two. Jomang wants to randomly generate allowable game conversions to increase replayability, but wants to disallow sets of game conversions through which players can generate infinite numbers of blocks via conversion. Describe an $O(n^3)$-time algorithm to determine whether a given a set of $\lfloor \frac{1}{5} n^2 \rfloor$ conversions should be disallowed. Assume that a starting world contains D blocks of each type, where D is the product of all d_i appearing in any conversion. Given positive integer x, assume that its bit-length, $\log_2 x$, can be computed in $O(1)$ time.

[1] A simple graph has no self-loops (i.e., each edge connects two different vertices) and has no multi-edges (i.e., there can be at most one directed edge from vertex a to vertex b, though a directed edge from b to a may exist).

[2] We mean Euclidean distance between landmark (x_1, y_1) and landmark (x_2, y_2), i.e., $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

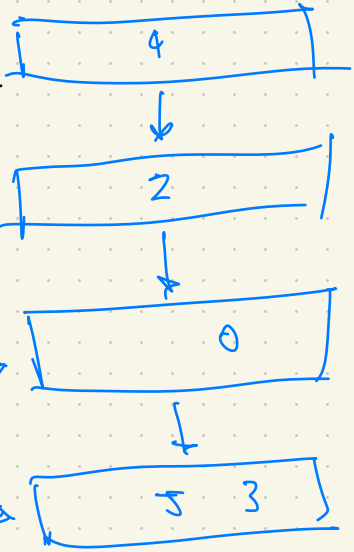
① ~~[4,2,1,0,5,3]~~ Didn't account for duplicates!



...153
...135
are counted twice!

There are $2! + 2! + 3!$ total topological orderings since ① can belong to either the anti-chain with ⓪ or the anti-chain with ⑤ & ③. If the former, there are $2! \cdot 2!$ permutations that yield a valid top. sort; if the latter, the last anti-chain can be permuted $3!$ ways.

[3] Any added edge that results in a cycle would preclude a top-ordering: e.g. if (1,4) were added, the graph would no longer be a DAG.



For any single edge to create a cycle it needs to connect a node from a "lower" anti-chain to a node in that must be in a "higher" anti-chain:

node	incident nodes
5	0, 2, 4
3	0, 2, 4
1	2, 4
0	2, 4
2	4

(because a simple graph has no self-loops, those are ignored in this tabulation)

TOTAL: 11

② The "never returns" condition allows for the construction of a DAG from the graph given by landmarks (nodes) and trails (edges). This DAG can be constructed in $O(n)$ time since the degree of all landmarks is $O(1)$. Running DAG relaxation on this DAG takes $O(n)$ time (again, trails are linear in the number of landmarks). Finally, finding the shortest path to any landmark outside of the Honor Stone can be done in $O(n)$ time, too.

③ This problem is an example of using Bellman-Ford to detect the existence of a negative-weight cycle on a digraph: nodes are types, edges are conversions whose weights are the bit-length difference such that if $d_2 > d_1$, the bit-length difference is signed negative (for any conversion). Constructing this graph representation takes $O(n + L\frac{1}{5}n^2 J) = O(n^2)$ time. Running Bellman-Ford on this graph takes $O(n \cdot L\frac{1}{5}n^2 J) = O(n^3)$ time, for an overall runtime of $O(n^3)$.

④ (SEE ANSWER KEY FOR EXPLANATION)

Had right idea to use Bellman-Ford but didn't come up with graph transformation to perform algorithm with drinking policy.

Problem 6-4. Tipsy Tannister

Lyrion Tannister is a greedy, drunken nobleman residing in Prince's Pier, the capital of Easteros, who wants to travel to the northern town of Summerrise. Lyrion has a map depicting all towns and roads in Easteros, where each road allows direct travel between a pair of towns, and each town is connected to at most seven roads. Each road is marked with the non-negative number of gold pieces he will be able to collect in taxes by traveling along that road in either direction (this number may be zero). Every town has a tavern, and Lyrion has marked each town with the positive number of gold pieces he would spend if he were to drink there. If Lyrion ever runs out of money on his trip, he will just leave a debt marker[3] indicating the number of gold pieces he owes. After leaving the tavern in Prince's Pier with no gold and zero debt, Lyrion's policy will be to drink at the tavern of every third town he visits along his route until reaching Summerrise. Given Lyrion's map depicting the n towns in Easteros, describe an $O(n^2)$-time algorithm to compute the maximum amount of gold he can have (minus debts incurred) upon arriving in Summerrise, traveling from Prince's Pier along a route that follows his drinking policy.

Problem 6-5. Cloud Computing

Azrosoft Micure is a cloud computing company where users can upload **computing jobs** to be executed remotely. The company has a large number of identical cloud computers available to run code, many more than the number of pieces of code in any single job. Any piece of code may be run on any available computer at any time. Each computing job consists of a code list and a dependency list.

A **code list** C is an array of code pairs $(f, t) \in C$, where string f is the file name of a piece of code, and t is the positive integer number of microseconds needed for that code to complete when assigned to a cloud computer. Assume file names are short and can be read in $O(1)$ time.

A **dependency list** D is an array of dependency pairs $(f_1, f_2) \in D$, where f_1 and f_2 are distinct file names that appear in C. A dependency pair (f_1, f_2) indicates that the piece of code named f_1 must be completed before the piece of code named f_2 can begin. Assume that every file name exists in some dependency pair.

(a) A job (C, D) can be **completed** if every piece of code in C can be completed while respecting the dependencies in D. Given job (C, D), describe an $O(|D|)$-time algorithm to decide whether the job can be completed.

(b) Azrosoft Micure wants to know how fast they can complete a given job. Given a job (C, D), describe an $O(|D|)$-time algorithm to determine the minimum number of microseconds that would be needed to complete the job (or return that the job cannot be completed).

(c) Write a Python function `min_time(C, D)` that implements your algorithm from (b).

(skip for time)

[3] The marker will be readily accepted because Tannisters always pay their debts.

(6) [A] This reduces to cycle detection on a digraph given by the edge list D. Running a topological sort algorithm (e.g. reverse DFS) in $\Theta(|D|)$ time will either produce a topological sort $\Rightarrow$ job can complete OR report a cycle $\Rightarrow$ job cannot complete

[B] To find the fastest completion time for a job that can complete boils down to finding the longest chain in the DAG, where for any edge (f_i, f_j), its weight is given by t in $(f_i, t) \in C$. We can negate all t and then proceed to find the shortest path via DAG relaxation (which includes topologically sorting the files).

Topological sorting takes $O(|D|)$ time (from [A]), and DAG relaxation on all edges is also $\Theta(|D|)$ in its runtime.

6.006 Introduction to Algorithms

Spring 2020