Daniel Willard

CS 315 W 2023

Lei Jiao

Problem 1)

1)->2->3

2)->1->4->5

3)->1->6->7

4)->2

5)->2

6)->3

7)->3

(I,J)	1)	2)	3)	4)	5)	6)	7)
1)	0	1	1	0	0	0	0
2)	1	0	0	1	1	0	0
3)	1	0	0	0	0	1	1
4)	0	1	0	0	0	0	0
5)	0	1	0	0	0	0	0
6)	0	0	1	0	0	0	0
7)	0	0	1	0	0	0	0

Problem 2)

BFS

Queue = Letter

Dequeue = ~~Letter~~

Steps are as fallows:

Q

~~Q~~, S, T, W

~~Q, S~~, T, W, V

~~Q, S, T~~, W, V, X, Y

~~Q, S, T, W~~, V, X, Y

~~Q, S, T, W, V~~, X, Y

~~Q, S, T, W, V, X~~, Y, Z

~~Q, S, T, W, V, X, Y~~, Z

~~Q, S, T, W, V, X, Y, Z~~

2nd iteration due to not all node visited.

R

~~R~~, U

~~R, U~~

Final Output = Q, S, T, W, V, X, Y, Z, R, U

DFS

Push: Letter

Pop: ~~Letter~~

Steps as fallows

Q

Q, S, V, W

Q, S, V, ~~W~~

Q, S, ~~V, W~~

Q~~, S, V, W~~

Q, ~~S, V, W,~~ T, X, Z

Q, ~~S, V, W~~, T, X, ~~Z~~

Q, ~~S, V, W~~, T, ~~X, Z~~

Q, ~~S, V, W~~, T, ~~X, Z,~~ Y

Q, ~~S, V, W~~, T, ~~X, Z, Y~~

Q, ~~S, V, W, T, X, Z, Y~~

~~Q, S, V, W, T, X, Z, Y~~

2ND iteration due to not all nodes visited.

R

R, U

R, ~~U~~

~~R, U~~

Final Output: W, V, S, Z, X, Y, T, Q, U, R

Or Q, S, V, W, T, X, Z, Y, R, U

Problem 3)

My proposed algorithm would be to use a graph data structure to represent the relationship between the butterflies.

I would start by initializing n vertices, each representing a butterfly Li.

Then I would iterate through the list of determinations and for each determination (Li, Lj)

I would add an edge between the vertices representing Li and Lj.

If at any point adding an edge results in the creation of a cycle, then the determinations are not consistent and the algorithm can return false.

If no cycle is detected after iterating through all the determinations, I would check if the graph has exactly two connected components, since there should be two different species. If it does, then the determinations are consistent and the algorithm can return true.

The time complexity of this algorithm would be O(n + r) because iterating through the list of determinations takes O(r) time and there are n butterflies.

Problem 4)

CS Curriculum of N courses all mandatory.

Graph = (N # or course, v number of edges (prerequisites to a course))

Linear run time.

Assume that students can take infinite number of courses in parallel during the term.
a set of courses can be taken in one semester if and only if there is no direct and indirect “prerequisite”
relationship among any of the courses in this set. “Direct” means a directed edge; “indirect” means
a path (following the directions of the edges). Besides, a course can be taken if and only if all of
its prerequisite courses were taken in a previous semester.

Answer)

My preposed algorithm to solve this problem is to use topological sorting. The basic idea is to repeatedly remove nodes from the graph that have no incoming edges.

1) Given the graph arrange the graph in a layered approach based off it dependances listed above.

2) Then we will use a topological sort where:

3) we will record all the nodes that have no incoming dependencies for term 1.

4) then once you have recorded the class remove the nodes and increase the term counter.

5) Then check again and see if there are classes to be taken

6) If so, take all the classes where there is no incoming dependencies and record the classes for that term.

7) repeat steps 4-5 till there are no classes left.

Finally) Print out term for the minimum number of terms to take all the CS courses.

O(courses + prerequisites) = O(n+m) so linear time to iterate over the nodes and edges once.