Strongly Connected Components

Yian Xu

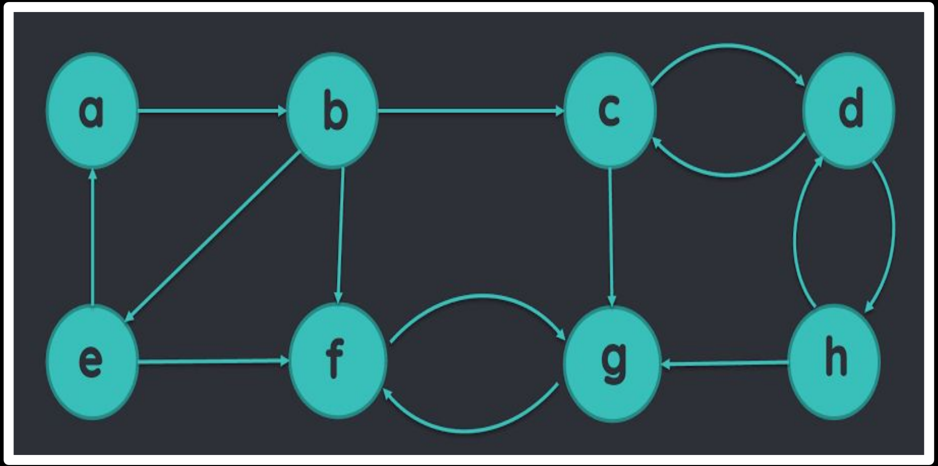
CSES: Planets and Kingdoms

- A game has n planets, connected by m teleporters. Two planets a and b belong to the same kingdom exactly when there is a route from a to b and from b to a. Your task is to determine for each planet its kingdom.

What it does:

A strongly connected component in a subgraph of a directed graph where every node can be reached from another.

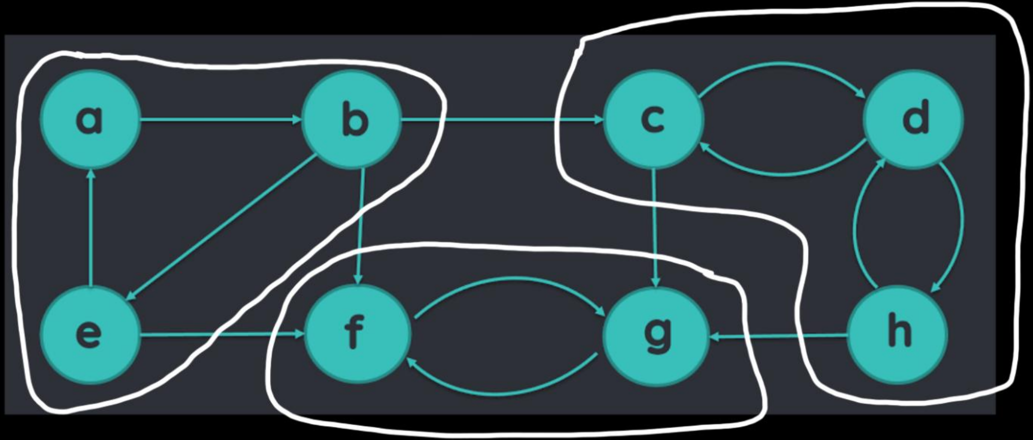
Finding strongly connected components can be helpful in simplifying the structure of the graph by presenting a SCC as a unit/ single node.



What it does:

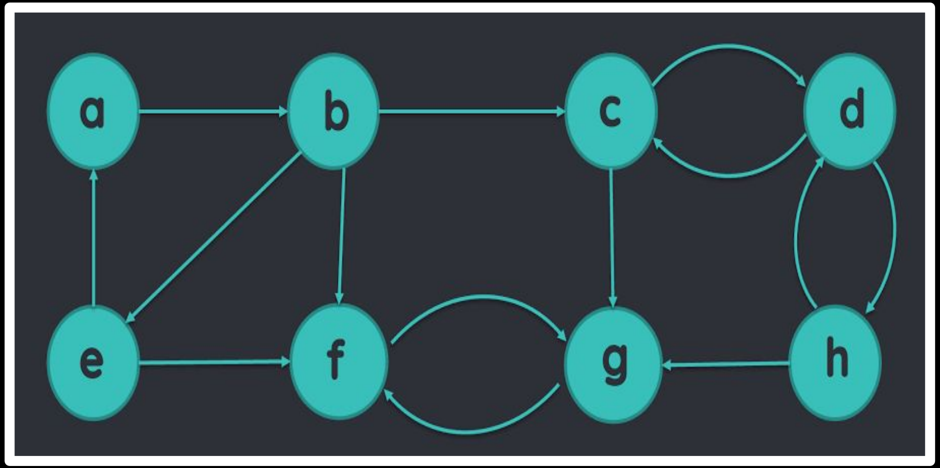
A strongly connected component in a subgraph of a directed graph where every node can be reached from another.

Finding strongly connected components can be helpful in simplifying the structure of the graph by presenting a SCC as a unit/ single node.



Brute force solution

- loop through all pairs of nodes
- for each pair check if they are reachable from eachother
- if they are, mark them as being in the same SCC



The time complexity of this is O(n) * O(n + e) = O(n^2 + en)

Efficient Algorithns

Kosaraju

Tarjan

Trajan’s Algorithm

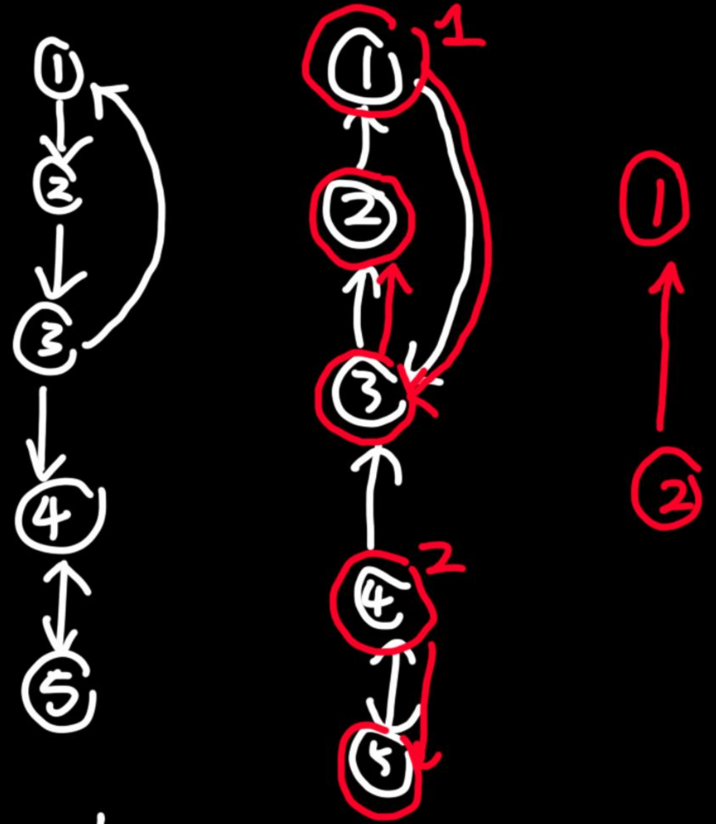
- Do a dfs
- for each node store its dist (distance from the root)
- store also back (the highest node that can be reached from it or a node in its subtree)
- the subtrees of nodes with dist = back with form strongly connected components
- O(V + E)

Kosaraju’s Algorithm

- Euler tour for ‘end time’ using dfs // or just top sort
- Reverse the edges in the graph
- dfs the reverse graph in the order descending time of finished processing
- each tree found of the reversed graph forest will now be a SCC in the origional graph
- O(V + E)

Kosaraju's Algorithm

Example on Planets and Kingdoms



top sort order:

1 2 3 4 5

Why does Kosaraju’s work?

Kosaraju’s Algorithm

finish(u) - the time that u is finished being processed by euler tour

G - the graph

G’ - the reversed graph

finish(S) - max of finish of elements of S for any graph S

1. Lemma

For given SCCs C and C’ in G if there exists an edge from a vertex in C to a vertex in C’ then finish(C) > finish(C’)

In the reverse graph G’ if there exists a vertex from a vertex in C to a vertex in C’ then finish(C) < finish(C’)

Case i) process C before C’

case ii) process C’ before C

2. Induction on subtrees

Assume the graph has been reversed and k subtrees have been process and you are processing the k + 1 th tree, call it T, starting at vertex u, call the SCC U is part of C

i) if v in C then v in T

ii) if v in T then v in C

2. Induction on subtrees

i) if v in C then v in T

- v is reachable by u
- v cannot have been previously visited

2. Induction on subtrees

ii) if v in T then v in C

- v cannot be in any SCCs that were processed before u
- if v were part of any later processed SCC C‘ then there would exist an edge from C to C’ implying that finish(C) < finish(C’)
- this contradicts the fact that the vertexes were processed in descending finish values

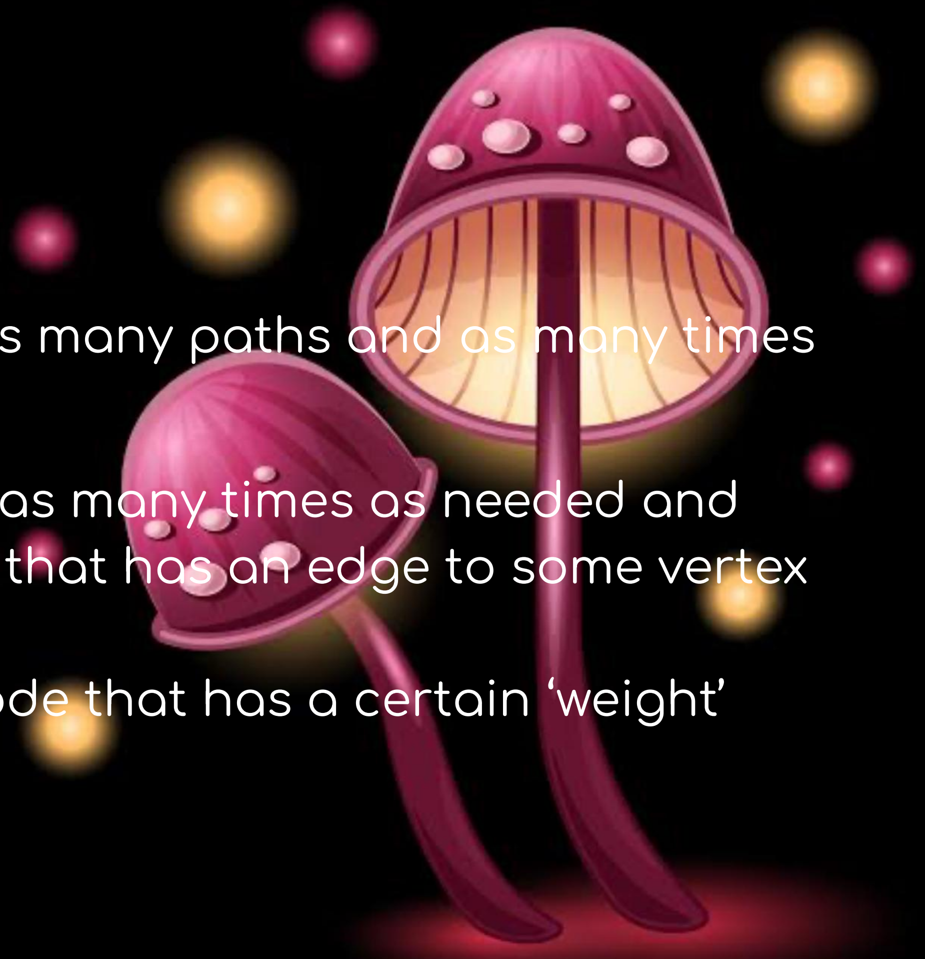
Codeforces: Ralph and Mushrooms

Ralph is going to collect mushrooms in the Mushroom Forest.

There are m directed paths connecting n trees in the Mushroom Forest. On each path grow some mushrooms. When Ralph passes a path, he collects all the mushrooms on the path. The Mushroom Forest has a magical fertile ground where mushrooms grow at a fantastic speed. New mushrooms regrow as soon as Ralph finishes mushroom collection on a path. More specifically, after Ralph passes a path the i-th time, there regrow i mushrooms less than there was before this pass. That is, if there is initially x mushrooms on a path, then Ralph will collect x mushrooms for the first time, x - 1 mushrooms the second time, x - 1 - 2 mushrooms the third time, and so on. However, the number of mushrooms can never be less than 0.

For example, let there be 9 mushrooms on a path initially. The number of mushrooms that can be collected from the path is 9, 8, 6 and 3 when Ralph passes by from first to fourth time. From the fifth time and later Ralph can't collect any mushrooms from the path (but still can pass it).

Ralph decided to start from the tree s. How many mushrooms can he collect using only described paths?



Solution:

Ideally we want to visit each as many paths and as many times as possible

- in a SCC we can loop around as many times as needed and enter/exit to any other vertex that has an edge to some vertex in the SCC
- replace each SCC with one node that has a certain ‘weight’ that it adds

Solution:

Now we are working with a DAG

- just need to find the longest path from s where the length of an edge is its initial mushroom count
- top sort and then dfs
- some other details left as an exercise to the reader :)