DP on DAGs
Strongly Connected Components

- Raghav Goel

Goal

To understand

important Div2 D/E

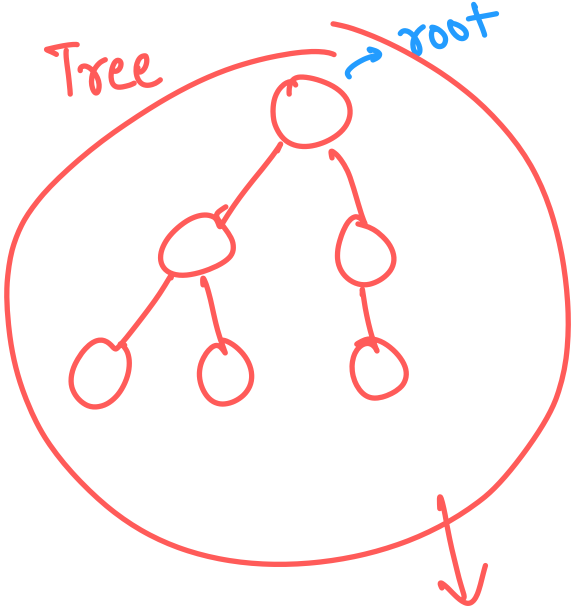
- DP on DAGs ✓
- Strongly Connected Components
- Condensation in Graphs

Div2 E

DP on DAGs

- **Dynamic Programming (DP) on DAGs** is a technique where we compute optimal values for nodes in a **topologically sorted** order.
- DAGs allow **efficient DP state transitions** since there are no cycles.
- **Common Applications:** longest paths, counting paths, and optimization problems.

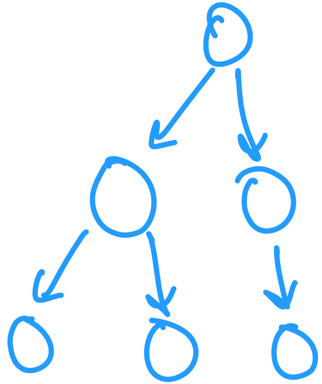
DP on Trees



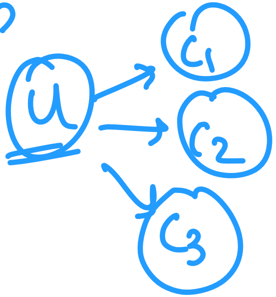
subtree dp

$$dp[u] = \underbrace{dp[c_1]} + \underbrace{dp[c_2]} + \underline{a[u]}$$

DAG



dp



$$dp[u] = dp[c_1] + dp[c_2] + a[u]$$

DAGs

- no root
- we don't know the order inwhich we should calculate the dp states

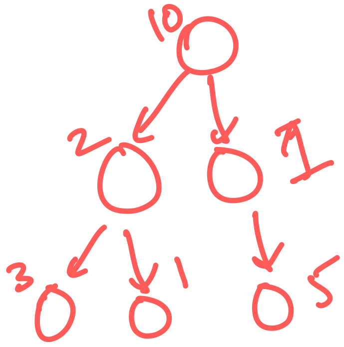
Problem:-

Given a DAG, each node u has some value a[u].

For each node find the maximum score of any path starting from that node

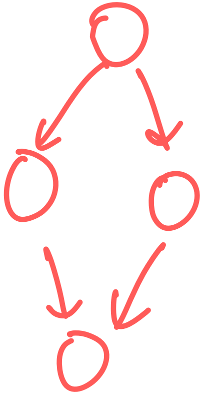
score of path = sum of values of all nodes in the path

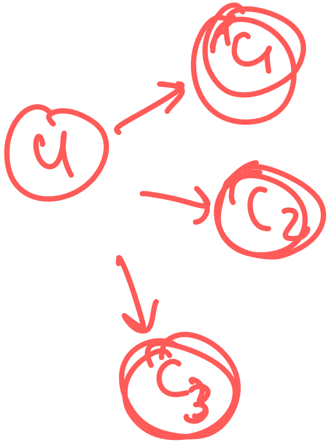
Tree



$dp[u] =$ answer for the node u

$$\boxed{dp[u] = a[u] + \max(dp[c_1], dp[c_2], dp[c_3] \ldots)}$$



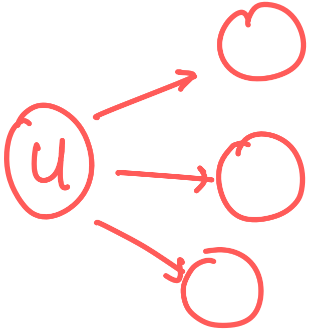


$dfs(u) \to dfs(c_1)$

$\quad\;\; \to dfs(c_2)$

$\quad\;\; \to dfs(c_3)$

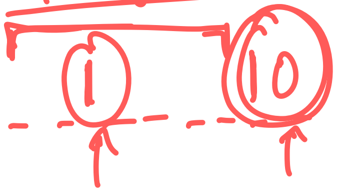
$dp[u] = \ldots\ldots$



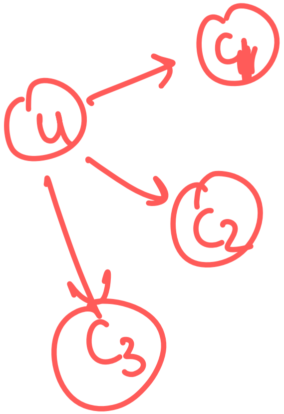
dp[u]

DAGs

Topological sorting

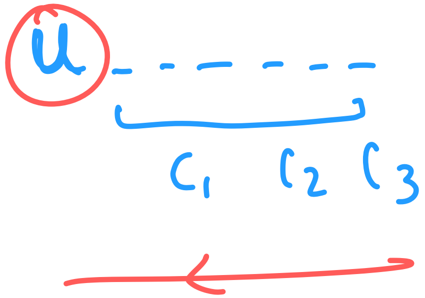


dp[1] ---- dp[10]



dp[c_1] dp[c_2] dp[c_3]

Topological Ordering



DP on DAGs

recursive

topological sorting

faster coz its iterative

Key Steps in DP on DAGs

1. Topological Sorting
2. Define DP State
3. Iterate in Topological Order

dp state

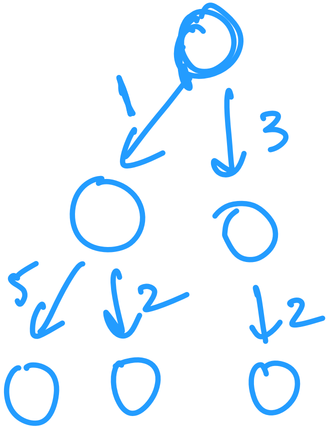
topological ordering

do the transitions

Problem: Longest Path in a DAG

Find the heaviest path from a given source in a weighted DAG.

The weight of a path is defined as the sum of weights of all the edges on the path.



$$dp[u] = \max\big(wt(u, c_1) + dp[c_1],\ wt(u, c_2) + dp[c_2], \ldots\big)$$

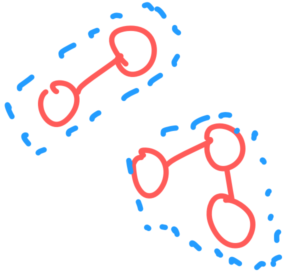
$$\text{for } (i = 1;\ i \le n;\ i{+}{+})\ \{\ \ dfs(i)\ \ \}$$

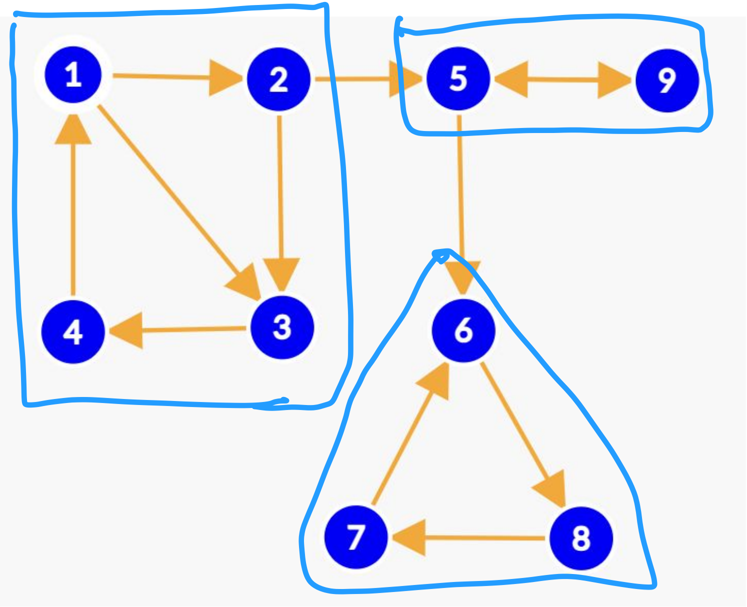
Strongly Connected Components (SCCs)

A strongly connected component in a directed graph is a maximal set of nodes such that there is a path from u to v and v to u for all the nodes in the set.

undirected graph

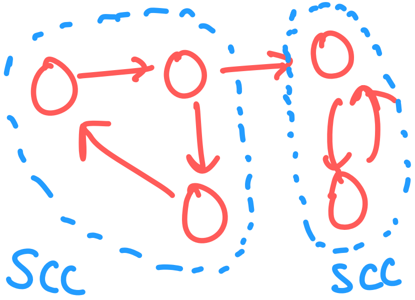
components





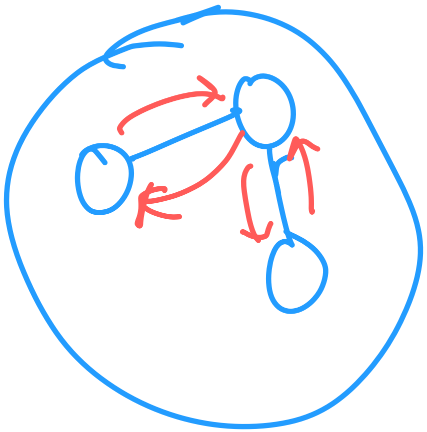
directed graphs

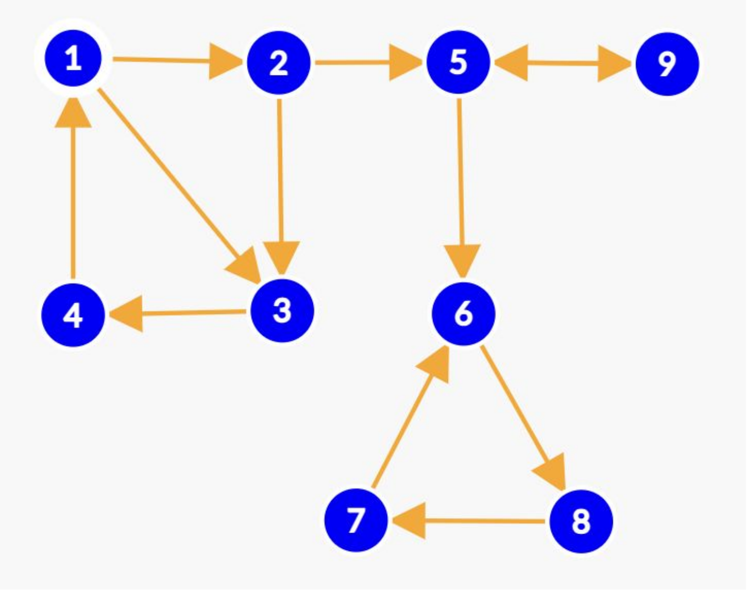
strongly connected components



Strongly Connected Components (SCCs)

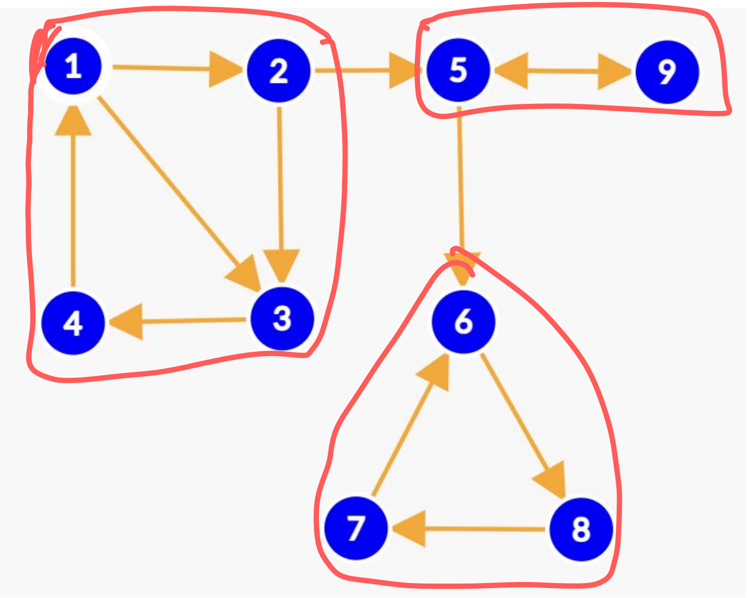
This is not a valid concept for Undirected graphs. There we talk about only connected components and there is no notion of strongly connected.

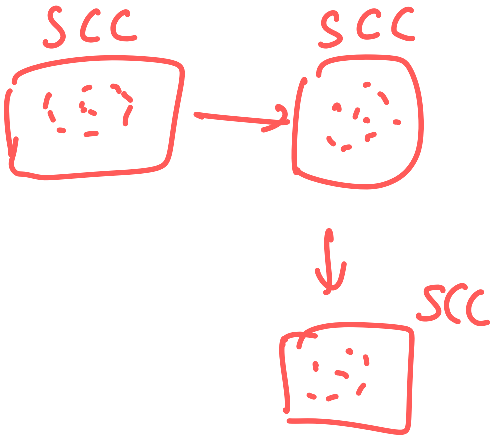


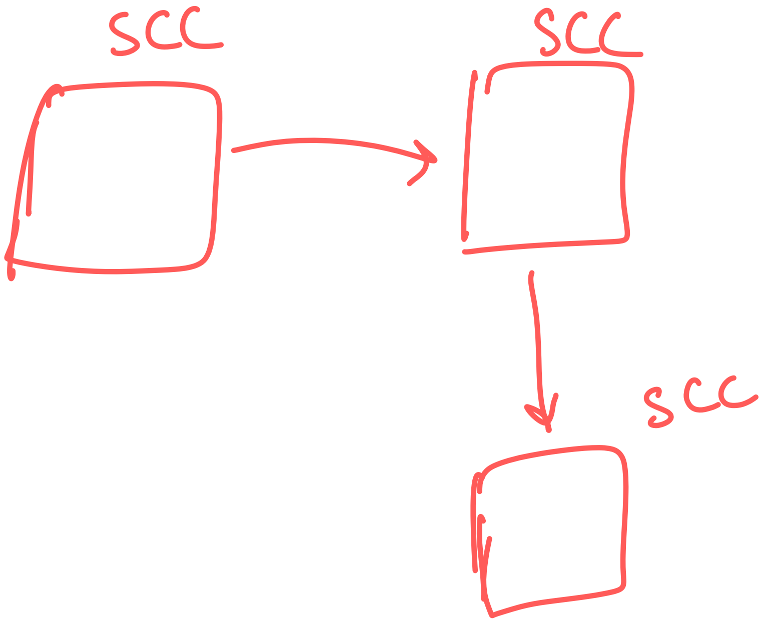


Strongly Connected Components (SCCs)

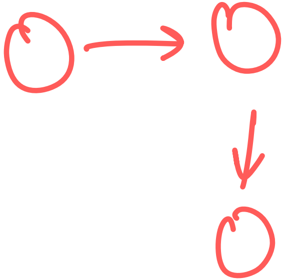
A graph can be decomposed into multiple strongly connected components. After decomposition, the graph becomes acyclic.







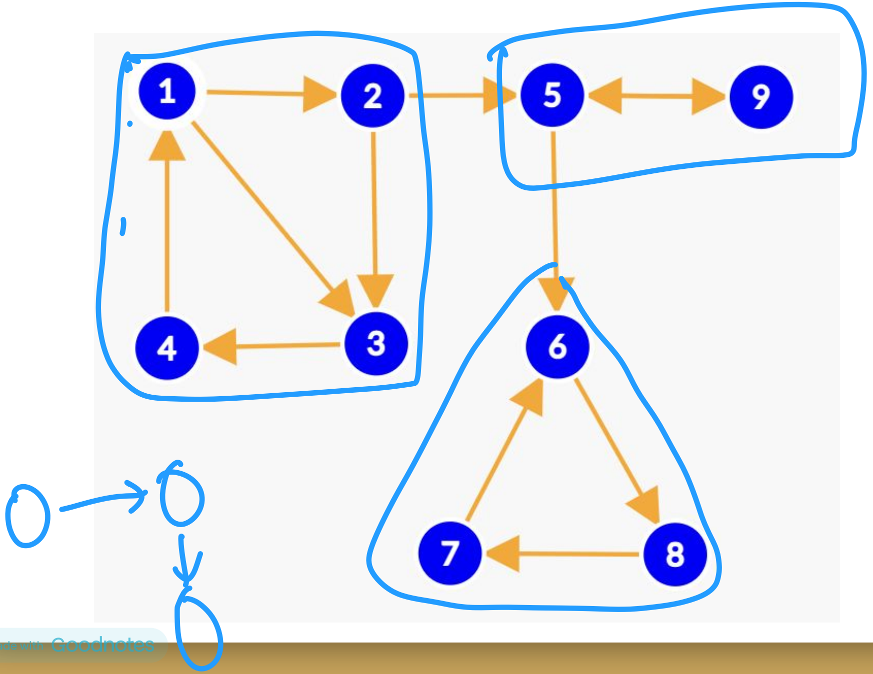
condensation of graph

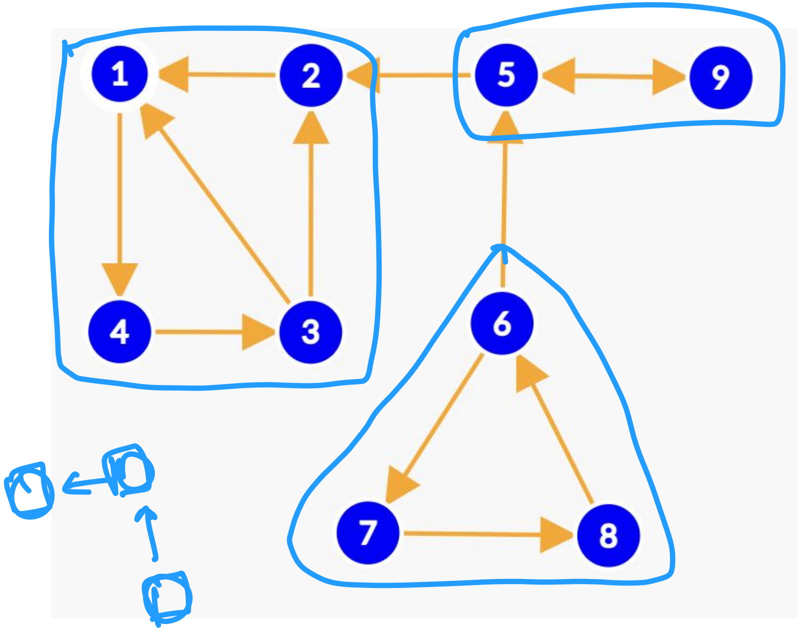


DAG

Strongly Connected Components (SCCs)

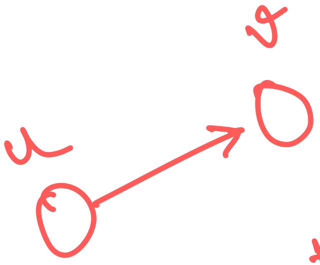
Strongly connected components of a graph don't change if the graph is transposed.



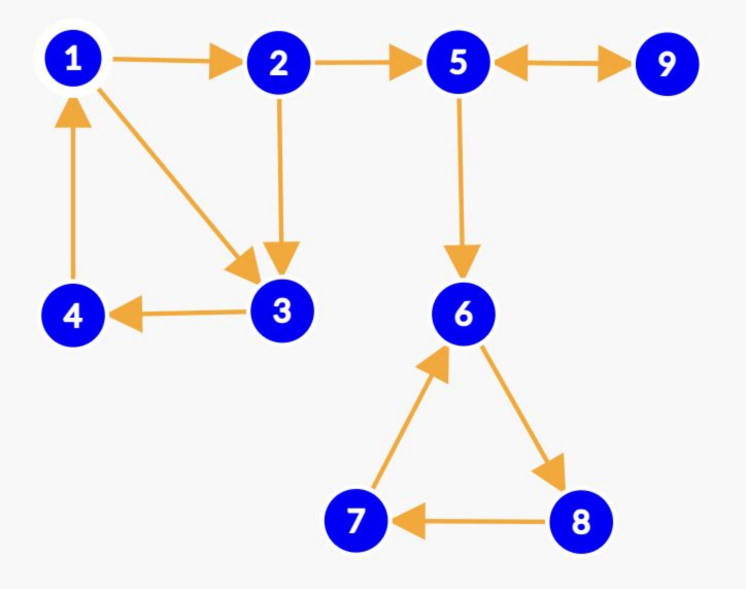


Strongly Connected Components (SCCs)

If there is an edge from SCC_i to SCC_j then $outTime[SCC_i] > outTime[SCC_j]$ irrespective of whether we start DFS from a node in SCC_i or SCC_j



$tout[u] > tout[v]$



How to find all SCCs

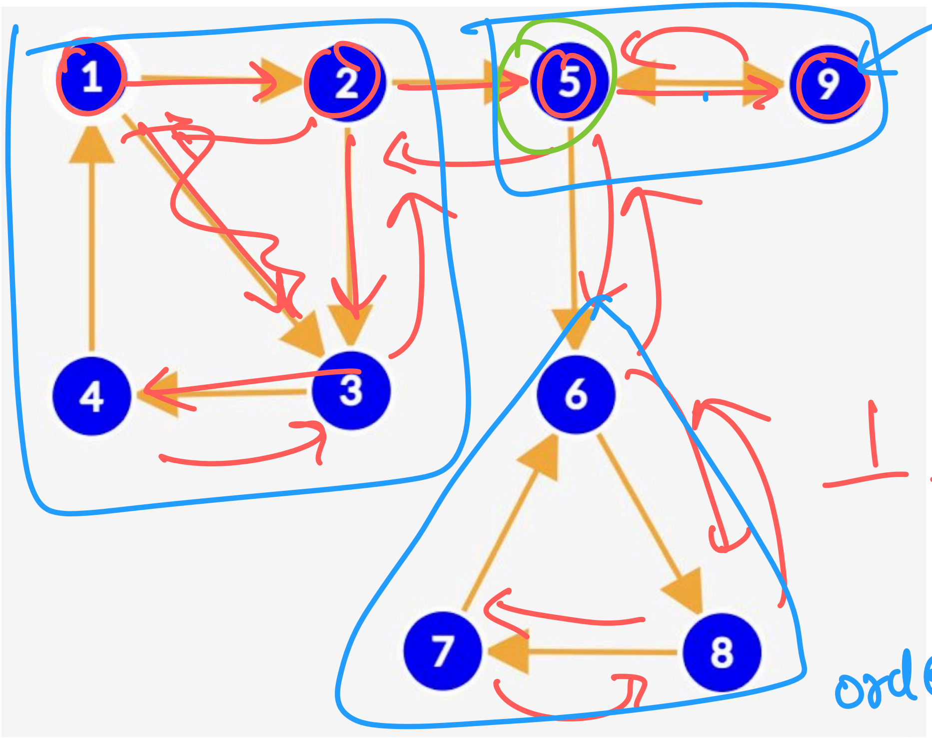
There are 2 famous algorithms:

- Kosaraju's Algorithm ✓
- Tarjan's Algorithm

Kosaraju’s Algorithm

- Perform DFS and store nodes in a stack
  - Run DFS on the original graph.
  - Store nodes in a stack based on their finish time.
- Reverse all edges (Transpose Graph)
  - Reverse every directed edge.
  - Now, SCCs remain SCCs, but edges are flipped.
- Process nodes in stack order on the transposed graph
  - Run DFS using nodes in the stack order.
  - Each DFS call discovers an SCC.

Time Complexity -> O(V + E)



Kosaraju's Algorithm

(10)

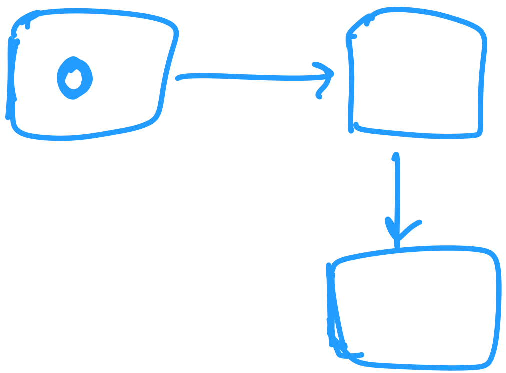
(1.) Similar to topological sorting

1 2 3 4 5 6 8 7 9

order [4]

1, 2, 3, - - - -, n

indegree(scc(4)) = 0

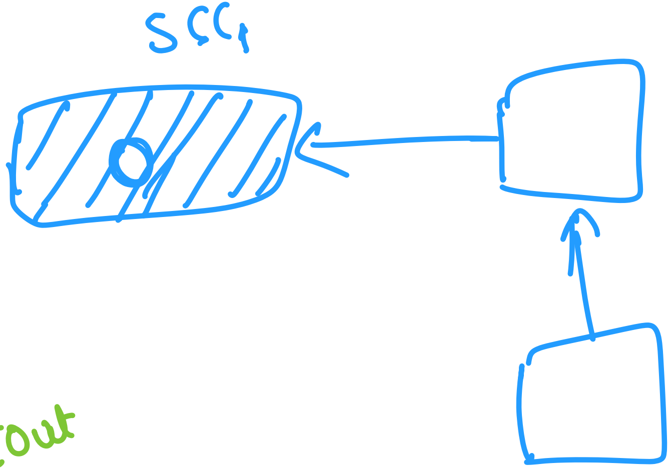


order

SCC indegree = 0

SCC indegree ≥ 1, it can never have the largest tout

- dfs
- find tout for all nodes
- sort all nodes on the basis of decreasing tout time



earlier

SCC₁

had 0 indegree

tout

now SCC₁ has 0 outdegree

yes

confusing but ok

no

Implementation of Kosaraju's Algorithm

```cpp
void kosaraju(vector<vector<int>> &adj, vector<vector<int>> &components,
              vector<vector<int>> &adj_cond) {
  int n = adj.size();

  // Getting the order of vertices in decreasing order of their finishing
  vector<int> vis(n, 0), order;
  auto dfs = [&](auto &&dfs, int u) -> void {
    vis[u] = 1;
    for (int v : adj[u])
      if (!vis[v]) dfs(dfs, v);
    order.push_back(u);
  };
  for (int u = 0; u < n; u++)
    if (!vis[u]) dfs(dfs, u);

  fill(vis.begin(), vis.end(), 0);
  reverse(order.begin(), order.end());

  // Getting the transpose of the graph.
  vector<vector<int>> adj_rev(n);
  for (int u = 0; u < n; u++)
    for (int v : adj[u]) adj_rev[v].push_back(u);

  auto dfs_rev = [&](auto &&dfs_rev, int u) -> void {
    vis[u] = 1;
    components.back().push_back(u);
    for (int v : adj_rev[u])
      if (!vis[v]) dfs_rev(dfs_rev, v);
  };

  vector<int> roots(n);

  // Getting the strongly connected components.
  for (int u : order) {
    if (vis[u]) continue;
    components.push_back({});
    dfs_rev(dfs_rev, u);
    vector<int> &component = components.back();
    int root = *min_element(component.begin(), component.end());
    for (int v : component) roots[v] = root;
  }

  // Getting the condensed graph.
  adj_cond.resize(n);
  for (int u = 0; u < n; u++)
    for (int v : adj[u])
      if (roots[u] != roots[v]) adj_cond[roots[u]].push_back(roots[v]);
}
```

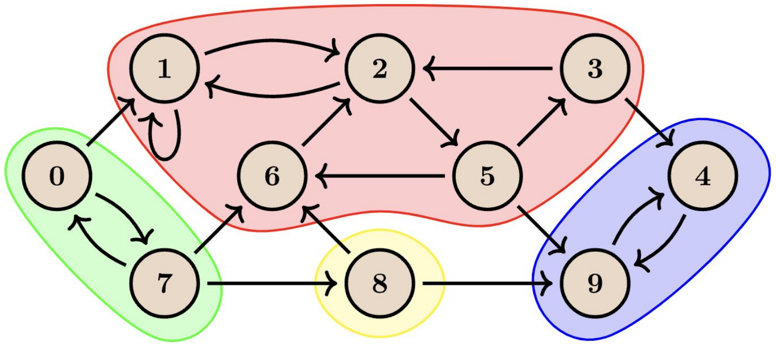
similar to topo order

order

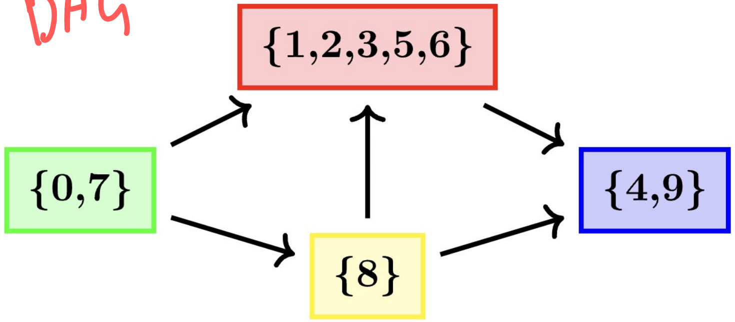
SCC

Condensation of Directed Graphs

Condensation of a directed graph refers to compressing Strongly Connected Components (SCCs) into single nodes, resulting in a Directed Acyclic Graph (DAG).



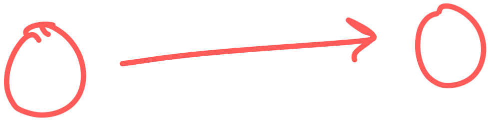
DAG



Problem: Coin Collector

A game has **n** ($1 <= n <= 10^5$) rooms and **m** ($1 <= m <= 2 * 10^5$) tunnels between them. Each room has a certain number of coins. What is the maximum number of coins you can collect while moving through the tunnels when you can freely choose your starting and ending room?

Problem Link - https://cses.fi/problemset/task/1686



n rooms

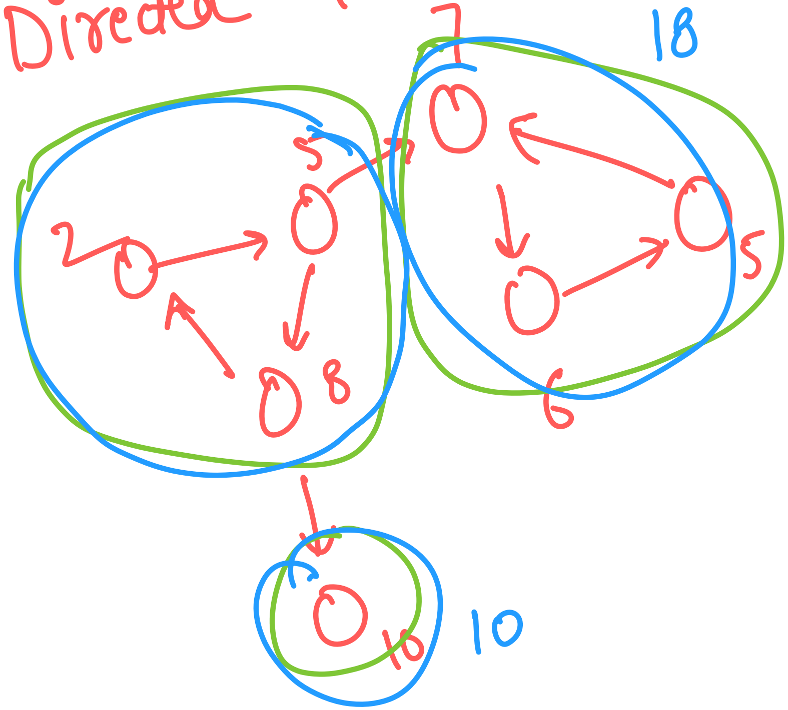
m tunnels (unidirectional)

Directed Graph

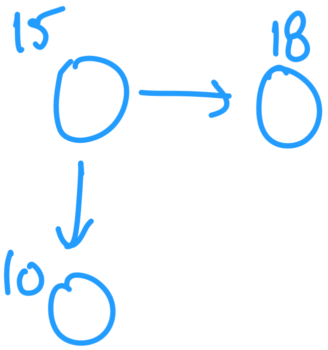
SCC

node

15



DAG



1. condense the graph $\Rightarrow$ DAG

2. DP on DAG

$$\boxed{dp[u] = a[u] + \max_{c \in children}(dp[c])}$$

Resources

- https://cp-algorithms.com/graph/strongly-connected-components.html