🏒

Strongly Connected Components

📘 Notes: Strongly Connected Components & Kosaraju’s Algorithm

---

🔷 DFS Discovery & Finishing Times

- **Discovery time** `d(v)`: when a vertex is first visited.
- **Finishing time** `f(v)`: when all its descendants are fully explored and we backtrack.
- Parentheses analogy:
  - Each discovery is an “open parenthesis (”.
  - Each finish is a “close parenthesis)”.
  - DFS on a tree produces properly nested parentheses (e.g., `( ( ) ( ) )`).lecture07_sccs

👉 These times help order nodes for Kosaraju’s algorithm.

🔷 Reachability

- Vertex `v` is *reachable* from `u` if there exists a directed path `u → … → v`.
- **Not symmetric**: If `u` can reach `v`, it doesn’t mean `v` can reach `u` (e.g., one-way street).lecture07_sccs
- BFS or DFS finds **all vertices reachable** from a start node.
- BFS/DFS on the **transpose graph** $G^T = (V, E^T)$ finds all vertices that can **reach** the starting point in the original graph.lecture07_sccs

🔷 Strongly Connected Graphs

- **Strongly connected graph**: every vertex can reach every other vertex.
- **SCC (Strongly Connected Component)**: a maximal subgraph that is strongly connected.
  - Maximal = you can't add any other vertex without breaking the property.
  - SCCs partition the graph.
- Between SCCs, edges form a **DAG** (no cycles between SCCs).lecture07_sccs

🔷 Naïve Solutions

1. **Naïve algorithm**:
   - For each vertex `u`: DFS to find all reachable nodes, then for each `v`, check if `v` can reach `u`.
   - Worst case: `O(V²(V+E))`. Too slow.lecture07_sccs
2. **Less naïve (reachability matrix)**:
   - Run DFS from every vertex → build `V×V` matrix of reachability.
   - SCCs found by checking mutual reachability.
   - Complexity: `O(V(V+E))`. Still expensive for large graphs.lecture07_sccs

🔷 Kosaraju's Algorithm

Process:

1. Run DFS on `G` and record finishing times (postorder).
2. Compute transpose `Gᵀ`.
3. Process vertices in **decreasing finishing time** order:
   - For each unvisited vertex, run DFS on `Gᵀ`.
   - Each DFS tree = one SCC.

Why it works (proof idea):
lecture07_sccs

- If SCC `C'` has an edge to SCC `C`, then `f(C') > f(C)` (C finishes before C').
- In the transpose, no edge goes from later-finished to earlier-finished SCCs.
- So, when we process nodes in decreasing `f(v)`, DFS stays inside one SCC.
- If DFS could spill into another, those SCCs were actually the same.

🔷 Complexity & Correctness

- **Time complexity**:
  - First DFS = `O(V+E)`
  - Transpose graph = `O(V+E)`
  - Second DFS = `O(V+E)`
  - ✅ Total = `O(V+E)`
- **Memory complexity**:
  - Adjacency list storage = `O(V+E)`
  - Stack/visited arrays = `O(V)`
  - ✅ Total = `O(V+E)`

🔷 Big Picture

- **DFS/BFS** = tools to test reachability.
- **Reachability isn't symmetric** in directed graphs.
- **SCCs** = subsets where reachability is symmetric (mutual).
- **Kosaraju's algorithm**:
  - First DFS (forward) orders SCCs.
  - Transpose makes SCCs self-contained.
  - Second DFS (backward, in order) isolates each SCC.

- Conceptual proof: **postorder finishing times ≈ reverse topological order of SCC DAG**, ensuring SCCs are peeled off one by one.

✅ These notes cover: discovery/finishing times, parentheses analogy, reachability, definition of SCCs, naïve/less naïve methods, Kosaraju's algorithm process, its proof, and complexity.