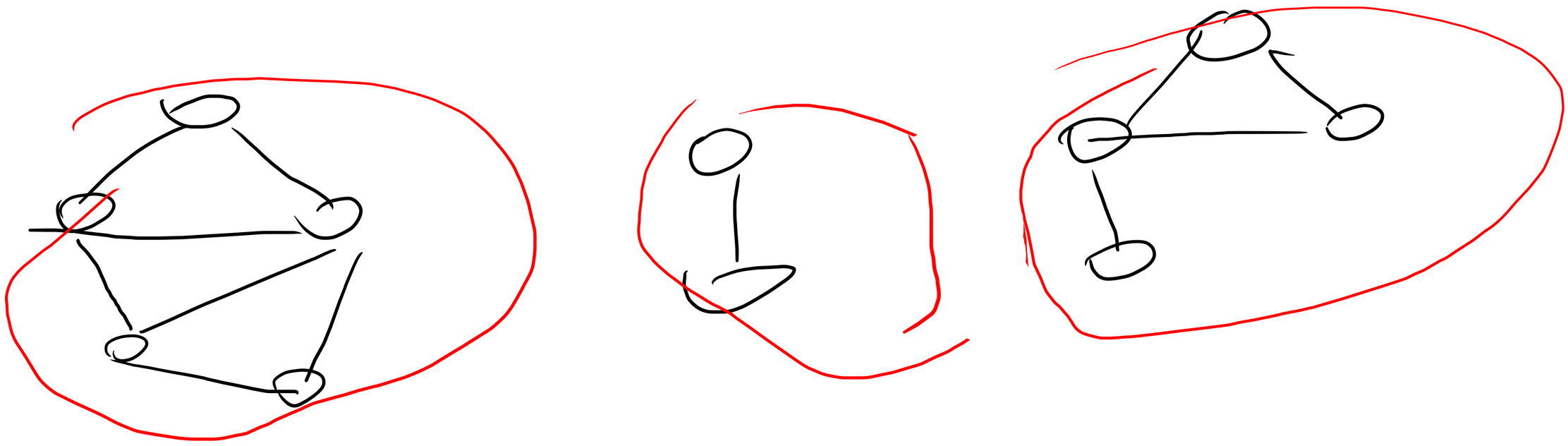
Fundamental Graph Problems

Finding connected Components in undirected Graphs

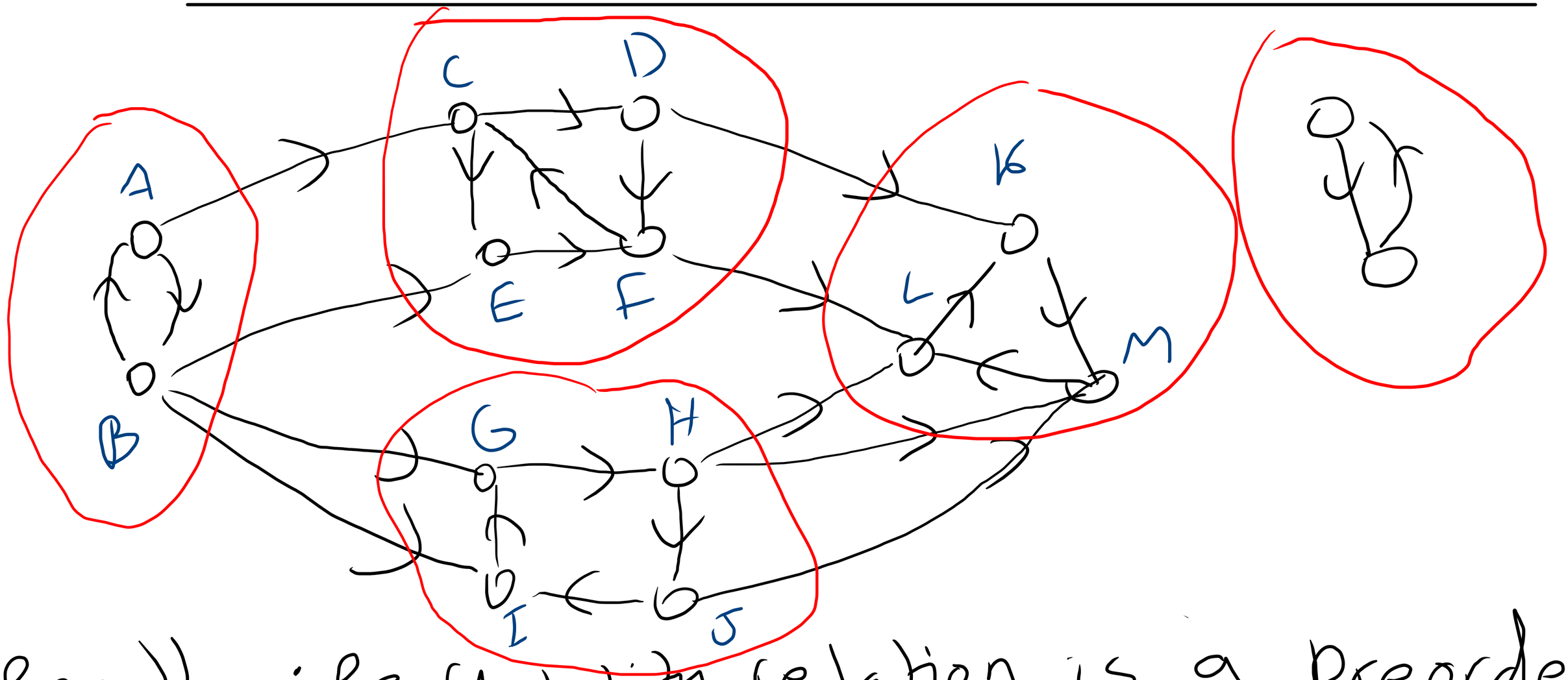


Algorithm:

- while $\exists v \in V$ which has not been visited
  - visit all nodes reachable from v using BFS(v) or DFS(v). Record them as a component.

Runtime: $O(m+n)$

Finding strongly connected components in directed Graphs



Recall : Reachability relation is a preorder

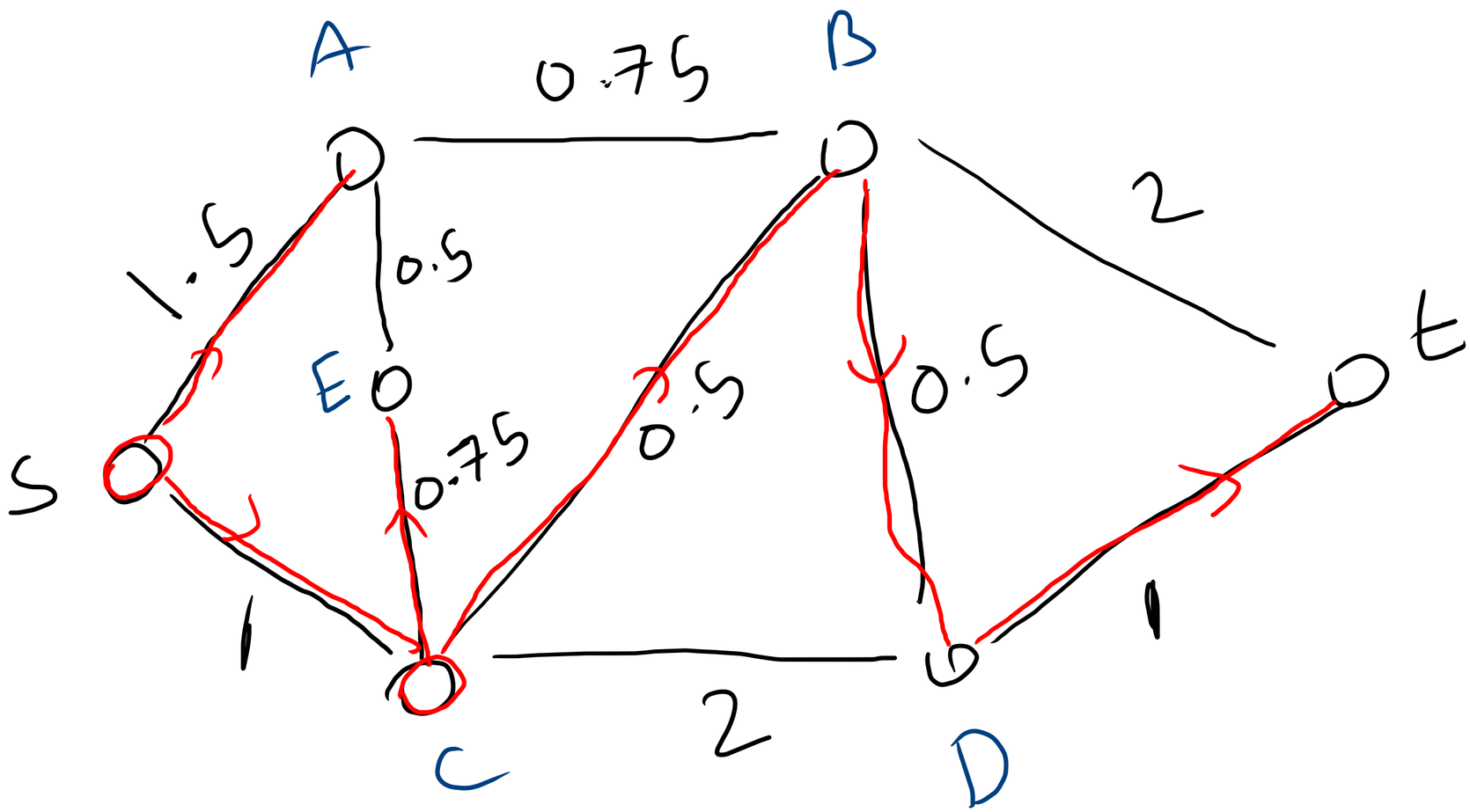
- Equivalence Classes of this preorder are called Strongly Connected Components (SCC)

- In an SCC every node can reach every other (in both directions)
- If u and v are in different SCC's either u can't reach v or v can't reach u or both.

- How do we find the SCCs?
  - Tarjan's Algorithm modifies DFS to find SCC's in O(m+n)

Shortest paths in weighted graphs

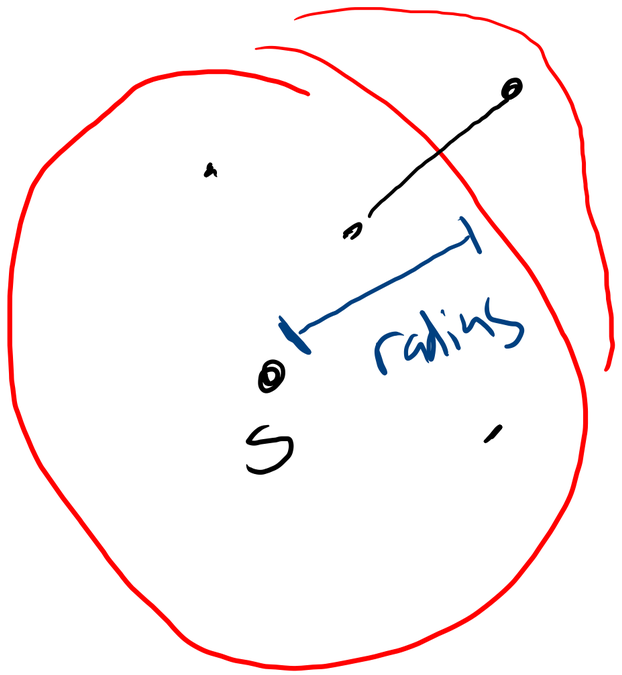
Given graph $G=(V,E)$ (directed or undirected), edge lengths (a.k.a. distances) d_e for $e \in E$, source $s \in V$, target $t \in V$. Goal is to find shortest path from s to t.



Shortest path: SC B D t, of length 3

Dijkstra's Algorithm

- start from s, visit nodes increasing order of distance from s, until ~~you reach~~ t.



Actually finds the shortest path tree rooted at s, which includes shortest path from s to everyone else.

(Single-source shortest paths Problem)

- using the right datastructures, can be implemented in $O(m + n \log n)$ time.

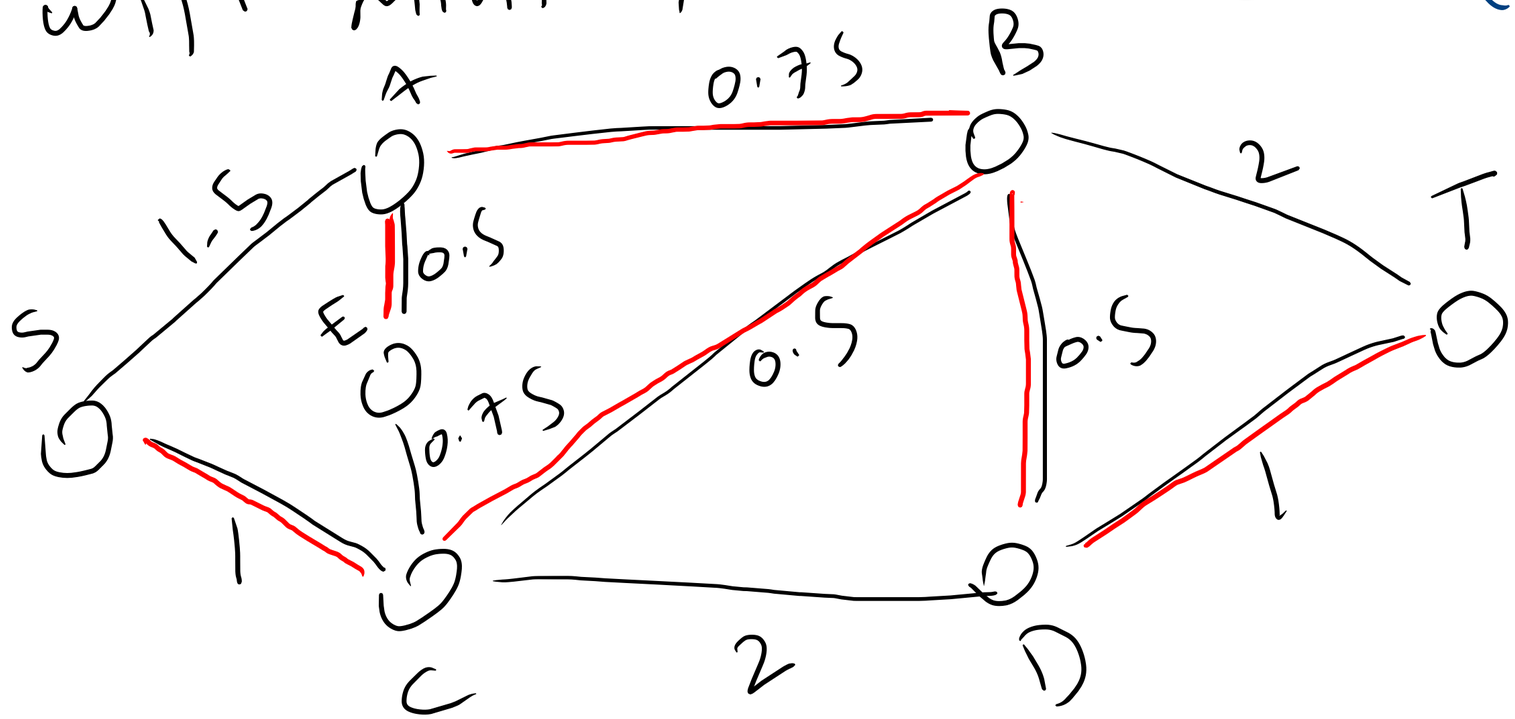
In 270 you will see $O((m+n)\log n)$

- Only works for nonnegative distances.
  More advanced algorithms (e.g. Bellman-ford) work for general (+ve or -ve) distances.
- Only solves single source shortest paths.
  More advanced algorithms (e.g. Floyd-Warshall) solve All pairs shortest paths.

Minimum Spanning Tree

connect undirected ↑

- Given graph $G=(V,E)$ with weights w_e on edges $e \in E$.
- Goal: find tree (spanning tree ↗) connecting all nodes, with minimum total weight. (MST)



observation:

MST ≠ SPT

Kruskal's Algorithm:

Repeatedly add cheapest edge which doesn't create cycle with what you already have

with the right datastructures: $O(m \log n)$