DS 320 Algorithms for Data Science	**Lecture #5**
Spring 2024	Prof. Kira Goldner

From DFS: Backedges, Postorder, and Cycles

Claim 1. If $(u, v) \in E$ then $postorder(u) < postorder(v) \iff (u, v)$ is a back edge.

Proof. If $(u, v) \in E$, then before u is popped off of the stack, we *could* have pushed v onto the stack via (u, v).

($\Leftarrow$) But if (u, v) is a backedge and not a tree edge, it must already be on the stack underneath u, and thus will pop after u.

($\Rightarrow$) Or, if $postorder(u) < postorder(v)$, (u, v) cannot be a tree edge or we'd push and pop v after pushing u, and thus we'd have $postorder(v) < postorder(u)$, a contradiction. Then (u, v) must be a backedge. □

Claim 2. $G = (V, E)$ has a cycle $\iff$ the DFS tree of G yields a back edge.

Proof. ($\Leftarrow$) If (u, v) is a back edge, then (u, v) together with the path from v to u in the DFS forest form a cycle.

($\Rightarrow$) Conversely, for any cycle in $G = (V, E)$, consider the vertex assigned the smallest postorder number. Then the edge leaving this vertex in the cycle must be a back edge by Claim 1, since it goes from a lower postorder number to a higher postorder number. □

Application: Topological Sort

Definition 1. A *topological ordering* on the vertices is a total ordering assigning them numbers $1, \ldots, n$ such that only edges $(i, j) \in E$ where $i < j$ in the ordering.

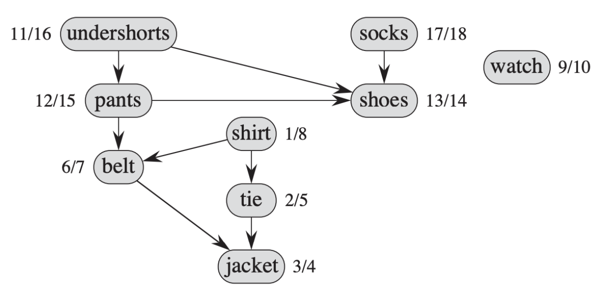

Figure 1: Top sort example graph from CLRS.

Theorem 1. *G has a topological order $\iff$ G is a DAG.*

Proof. ($\Rightarrow$) Topological ordering means only edges $(i, j) \in E$ where $i < j$. Consider the smallest i in the cycle. There exists an edge (j, i) in the cycle for $j > i$. Contradiction.

($\Leftarrow$) If there's a DAG, this implies that there exists a node with no incoming edges. Otherwise, one could backtrack, and after n steps, would find a cycle. □

Topological Sort Algorithm:

- Naive algorithm: Recursively remove a node with no incoming edges. $T(n) = O(n^2)$.
- Or, run DFS to assign postorder times, and then sort the DFS forest by decreasing postorder. $T(n) = O(n + m)$. (We can avoid the sorting runtime using a priority heap data structure.)

Theorem 2. *If the tasks are scheduled by decreasing postorder number, then all precedence constraints are satisfied.*

Proof. If G is acyclic then the DFS tree of G produces no back edges by Claim 2. Therefore by Claim 1, $(u, v) \in G$ implies $postorder(u) > postorder(v)$. So, if we process the tasks in decreasing order by postorder number, when task v is processed, all tasks with precedence constraints into v (and therefore higher postorder numbers) must already have been processed. □

Breadth-First Search

Algorithm 1 BFS(G, s)

```
Input: Graph G = (V, E) and starting vertex s.
initialize: (1) array dist of length n, (2) queue q, (3) linked list L of sets, (4) tree T = ({s}, ∅)
dist[s] = 0
L[0] = {s}
enqueue s to q
mark s as discovered and all other v as undiscovered
while size(q) > 0 do
    v = dequeue(q)
    for (v, w) ∈ E do
        if w is undiscovered then
            enqueue w in q
            mark w as discovered
            dist(w) = dist(v) + 1
            add w to L[dist(w)]
            add (v, w) to T
        end if
    end for
end while
return T, L
```

What happens when we run BFS(G, 1) where G is the graph below?

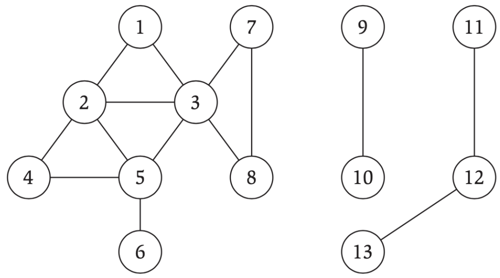

Figure 2: Example graph G. From Kleinberg Tardos.

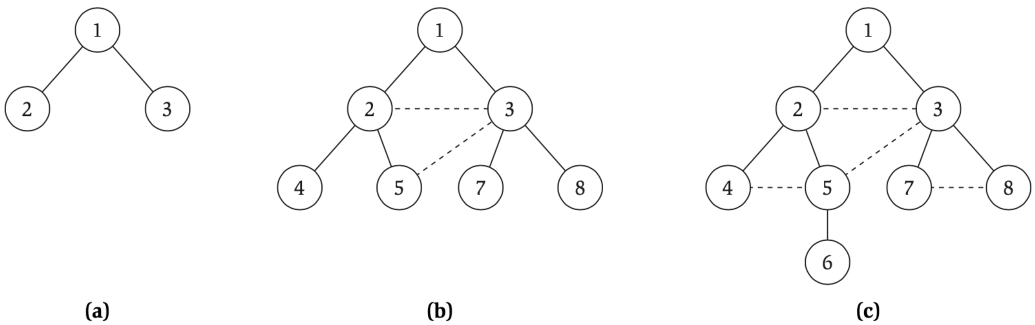

Figure 3: The BFS tree is shown in solid edges as constructed in stages (layers) by the above algorithm. The dashed edges are the edges that do not belong to the BFS tree but do belong to G. From Kleinberg Tardos.

What is BFS doing? BFS labels each vertex with the distance from s, or the number of edges in the shortest path from s to the vertex. (**Exercise:** Prove this!)

Runtime: $O(|E|)$ (assuming that $|E| \geq |V|$). The reason is that BFS visits each edge exactly once, and does a constant amount of work per edge.

Claim 3. Let T be a breadth-first search tree, let x and y be nodes in T belonging to layers L_i and L_j respectively, and let (x, y) be an edge of G. Then i and j differ by at most 1.

Proof. Suppose by way of contradiction that i and j differed by more than 1; in particular, suppose $i < j - 1$. Now consider the point in the BFS algorithm when the edges incident to x were being examined. Since x belongs to layer L_i, the only nodes discovered from x belong to layers $L_i + 1$ and earlier; hence, if y is a neighbor of x, then it should have been discovered by this point at the latest and hence should belong to layer $L_i + 1$ or earlier. $\square$