DS 320 Algorithms for Data Science **Lecture #5**
Spring 2026 Prof. Kira Goldner and Jeffrey Considine

Application: Topological Sort

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

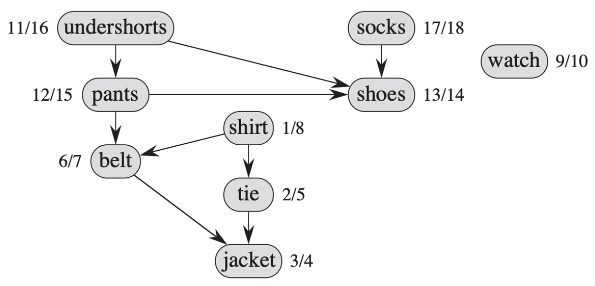

Figure 1: Top sort example graph from CLRS.

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 **??**. Therefore by Claim **??**, $(u, v) \in G$ 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. □

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 $T = (\{s\}, \emptyset)$
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 $(v, w) \in E$ 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 $T, L$
```

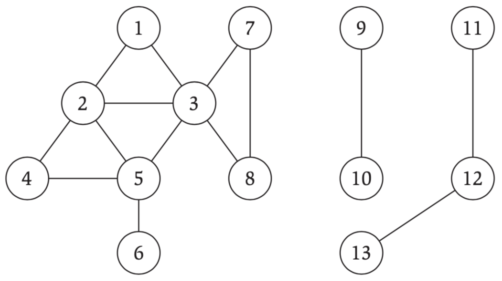

Figure 2: Example graph G. From Kleinberg Tardos.

Breadth-First Search

What happens when we run BFS(G, 1) where G is the graph below?
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! Use the next claim.)

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

Claim 1. 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

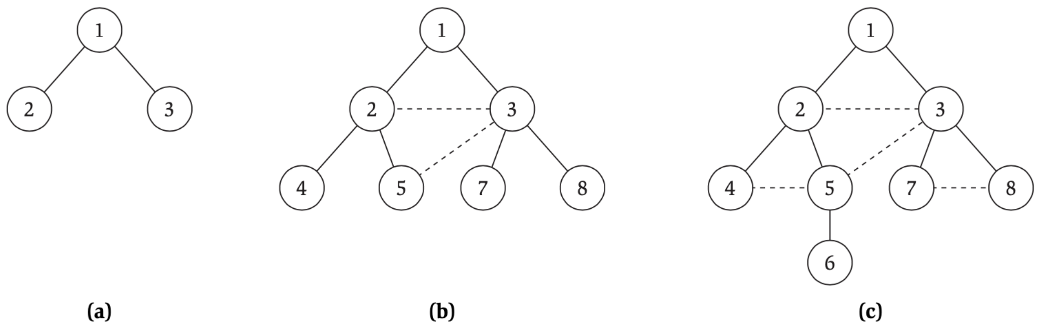

Figure 3: The BFS tree is shown in solid edges as constructed in stages (layers) by the above algorithm. The dashed edges are the edges that do not belong to the BFS tree but do belong to G. From Kleinberg Tardos.

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. $\square$