DS 320 Algorithms for Data Science **Lecture #6 Worksheet**
Spring 2023 Prof. Kira Goldner

Preorder, Postorder, and Backedges

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

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

Application: Topological Sort

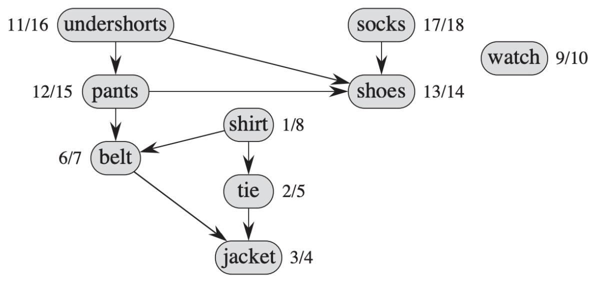

Figure 1: Top sort example graph from CLRS.

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

Topological Sort Algorithm:

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

Introduction to Greedy: Shortest Path

Say we want to find the shortest path from home to any other point in the city. How might we do that?

But, what if there's congestion on the roads? That is, what if the graph is weighted?

Definition 1. Let w_e (or w_{uv}) denote the *weight* of edge e (or (u, v)).

We can think of this as the length of the edge, or the time (or cost) to traverse it.

Question: Given a graph G, how can we find the shortest (least-weight) path from s to any other vertex v?

Algorithm 1 Dijkstra's Algorithm(G, w)

```
Input: Graph $G = (V, E)$ and weights $w$.
Let $S$ be the set of explored nodes
for each $u \in S$, store a distance $d(u)$
Initially $S = \{s\}$ and $d(s) = 0$
while $S \neq V$ do
    let $d'(v) = \min_{(u,v) \in E, u \in S} d(u) + w_{uv}$
    select $v \in \text{argmin}_{v \notin S} d'(v)$
    add $v$ to $S$ and set $d(v) = d'(v)$
end while
```

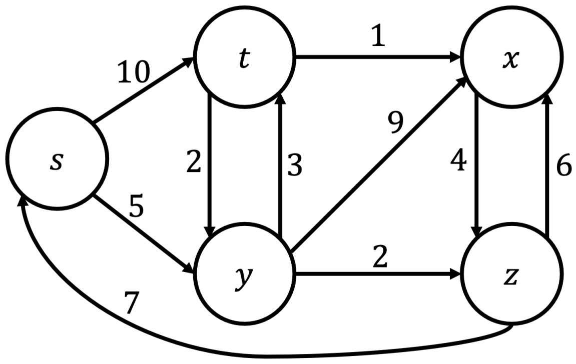

Figure 2: A weighted graph G.

Let P_v denote the shortest path to v from s—that is, $P_v = P_u \cup (u, v)$ when $v \in \text{argmin}_{v \notin S} d'(v)$ and $(u, v) \in \text{argmin}_{(u,v) \in E, u \in S} d(u) + w_{u,v}$.

Greedy Stays Ahead

There are four main steps for a greedy stays ahead proof.

Step 1: Define your solutions. Describe the form your greedy solution takes, and what form some other solution takes (possibly the optimal solution). For example, let A be the solution constructed by the greedy algorithm, and let O be a (possibly optimal) solution.

Step 2: Find a measure. Find a *measure* by which greedy stays ahead of the other solution you chose to compare with. Let $a_1, \ldots, a_k$ be the first k measures of the greedy algorithm, and let $o_1, \ldots, o_m$ be the first m measures of the other solution ($m = k$ sometimes).

Step 3: Prove greedy stays ahead. Show that the partial solutions constructed by greedy are always just as good as the initial segments of your other solution, based on the measure you selected.

- For all indices $r \leq \min(k, m)$, prove (often by induction) that $a_r \geq o_r$ or that $a_r \leq o_r$, whichever the case may be. Don't forget to use your algorithm to help you argue the inductive step.

Step 4: Prove optimality. Prove that since greedy stays ahead of the other solution with respect to the measure you selected, then it is optimal.

Proof of Dijkstra

Then we prove the following via a "greedy stays ahead"-style induction.

Lemma 1. *Consider the set S at any point in the algorithm's execution. For each $u \in S$, P_u is a shortest $s - u$ path.*

(*Proof of Correctness:*

)

Proof. By induction on $|S|$.

Base Case:

Inductive Hypothesis:

Inductive Step:

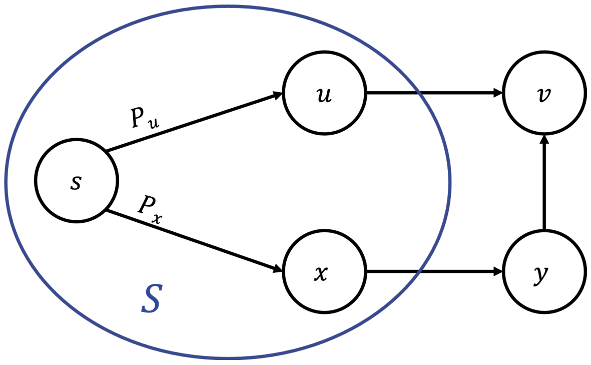

Figure 3: Illustration of the argument in the inductive step.