Lecture Notes #17

Single-Source Shortest Paths in Directed Acyclic Graphs

There is no cycle in a directed acyclic graph (DAG). Hence, no negative-weight cycle can exists in a DAC, and SPs are well defined.

Single-source shortest paths problem for DAGs can be solved more efficiently by using topological sort.

Topological sort of a DAG $G = (V, E)$ is a linear ordering of all its nodes such that if G has an edge (u, v), then u appears before v in the ordering.

A topological sort of G can be viewed as an ordering of its nodes along a horizontal line so that all directed edges go from left to right.

Topological Sort Algorithm

It is an application of depth-first search (see pp. 540 - 551 of the textbook).

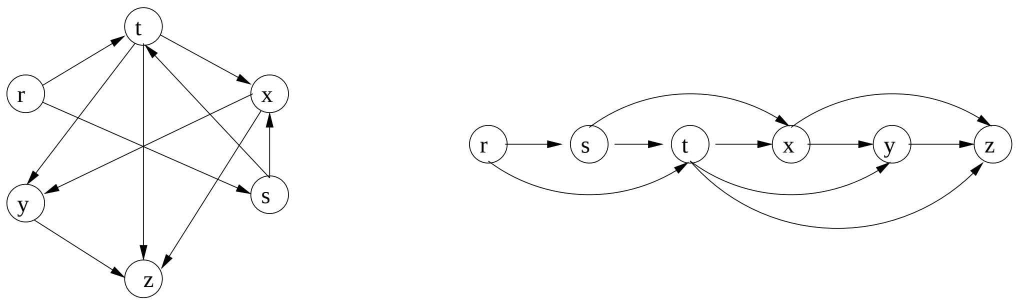
procedure *Topsort*(G)
{
    for all $v \in G$ **do** mark v "unvisited";
    while there exists a node $v \in G$ marked "unvisited" **do** *Sort*(G, v)
}

procedure *Sort*(G, v)
{
    mark v "visited";
    print v;

for each $u \in Adj[v]$ **do if** u is marked "unvisited" **then** $Sort(G, u)$

}

For an example, see Figure 1. The time complexity of *Topsort* is $O(|V| + |E|)$.



(a) (b)

Figure 1: (a) A DAG. (b) The same graph shown topologically sorted.

Shortest Paths for DAGs

The following algorithm solves the single-source shortest paths problem for DAGs.

procedure *DAG-Shortest-Paths*(G, s, w)
{
    Topsort(G);
    Initialize-Single-Source(G, s);
    $S := \emptyset$;
    for each node u, taken in topologically sorted order **do**
    {
        for each node $v \in Adj[u]$ **do** *Relax*(u, v, w)
        $S := S \cup \{u\}$;
    }
}

The subroutines *Initialize-Single-Source* and *Relax*, which are the ones used for the Bellman-Ford algorithm, are repeated below:

```
procedure Initialize-Single-Source(G, s)
{
    for each node v ∈ V of G do
    {
        d[v] := ∞; π[v] := nil
    }
    d[s] := 0;
}
```

```
procedure Relax(u, v, w) /* operation for a relaxation step on edge (u, v) */
{
    if d[v] > d[u] + w(u, v) then
    {
        d[v] := d[u] + w(u, v); π[v] := u
    }
}
```

Example 1 *Figure 2 shows the execution of DAG-Shortest-Paths on a DAG.*

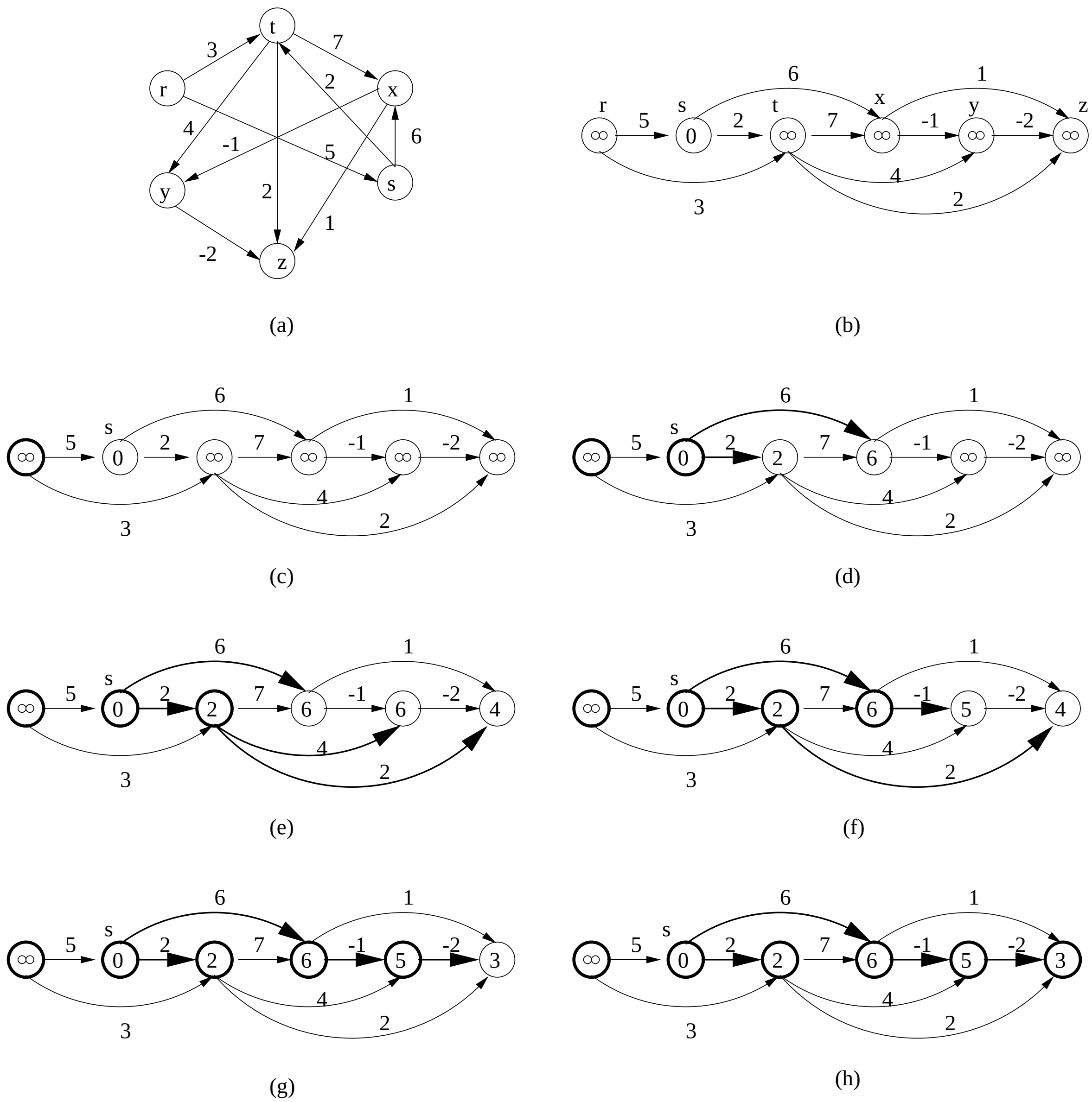


Figure 2: Execution of algorithm *DAG-Shortest-Paths* on a DAG. (a) Given DAG. (b) After topological sorting. (c) - (h) correspond to 6 iterations. A newly darkened circle (node) in each iteration is the node u in the iteration. Values in (h) are final lengths of shortest paths from s.

Correctness of the Algorithm

Let $\delta(s, v)$ be defined as before:

$$\delta(s,v) = \begin{cases} \min\{w(p) | s \overset{p}{\rightsquigarrow} v\}, & \text{if a path from } s \text{ to } v \text{ exi sts} \\ \infty, & \text{otherwise} \end{cases}$$

Let $P = v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow \cdots \rightarrow v_m$ be a path. The nodes in $\{v_1, v_2, \cdots, v_{j-1}\}$ are called the predecessors of v_j in P.

Define the shortest path P from s to v such that all predecessors of v are in S as *the shortest path from s to v with respect to S.*

Lemma 1 *Let $(u_1, u_2, \cdots, u_n)$ be the list of nodes in topologically sorted order with $n = |V|$, and $u_i = s$. Right after j-th iteration of the outer for-loop of DAG-Shortest-Paths, $d[u_k]$, $k > j$, is the weight of the shortest path (SP) from s to u_k with respect to S. Furthermore, $d[u_{j+1}] = \delta(s, u_{j+1})$.*

Proof.

Base: $j = i$. After the i-th iteration of the outer for-loop, $S = \{s\}$ and $d[u_k] = w(s, u_k)$. Clearly, the lemma is true.

Hypothesis: Suppose the lemma is true for $j = m < n - 1$.

Induction: Consider $j = m + 1$. In the $(m + 1)$-th iteration, u_{m+1} is included into S and *Relax*(u_{m+1}, v, w) is called for every $v \in Adj[u_{m+1}]$, which is to the right of u_{m+1}.

If u_{m+1} is not reachable from s, then v is also not reachable from s, and $d[v]$ remains to have value ∞.

If u_{m+1} is reachable from s, then v is also reachable from s. If $d[v] > d[u_{m+1}] + w(u_{m+1}, v)$, i.e. the SP from s to v with respect to $\{u_i, u_{i+1}, \cdots, u_m\}$ is longer than the SP from s to v with u_{m+1} as immediate predecessor, $d[v]$ value is updated as $d[v] := d[u_{m+1}] + w(u_{m+1}, v)$. By the hypothesis, this new $d[v]$ value is the weight of the shortest path (SP) from s to v with respect to new $S = \{u_i, u_{i+1}, \cdots, u_{m+1}\}$.

Furthermore, if $v = u_{m+2}$, then, $d[v] = \delta(s, v)$, because there is no other node in $\{u_{m+3}, u_{m+4}, \cdots, u_n\}$ from which u_{m+2} can be reached (by the topological order of nodes).

Hence, the lemma is true for the $(m+1)$-th iteration. This completes the induction and the proof of the lemma.

□

Theorem 1 *Algorithm DAG-Shortest-Paths, run on a weighted DAG $G = (V, E)$ with source s, terminates with $d[v] = \delta(s, v)$ for all nodes $v \in V$.*

Proof. By Lemma 1, after the j-th iteration, $d[u_{j+1}] = \delta(s, u_{j+1})$ is finalized. After $n-1$ iterations, $d[v] = \delta(s, v)$ for all nodes $v \in V$. The outer for-loop runs for n iterations. The last iteration is actually redundant. □

Time Complexity Analysis

- *Topsort* takes $O(|V| + |E|)$ time.
- *Initialize-Single-Source* takes $O(|V|)$ time.
- Outer for-loop runs $|V|$ iterations. But regardless of the outer for-loop, the inner for-loop runs a total of $|E|$ iterations since each edge is "traversed" from left to right exactly once. Each iteration of the inner loop takes $O(1)$ time. Hence, the nested for-loops take $O(|V| + |E|)$ time.

Hence, total running time is $O(|V| + |E|)$.