CH-231-A

# Algorithms and Data Structures

ADS

## Lecture 37

Dr. Kinga Lipskoch

Spring 2020

## Correctness (i)

Lemma:

- Initializing $d[s] = 0$ and $d[v] = \infty$ for all $v \in V \setminus \{s\}$ establishes $d[v] \geq \delta(s, v)$ for all $v \in V$.
- This invariant is maintained over any sequence of relaxation steps.

Proof:

Suppose the Lemma is not true, then let $v$ be the first vertex for which $d[v] < \delta(s, v)$ and let $u$ be the vertex that caused $d[v]$ to change by $d[v] = d[u] + w(u, v)$. Then,

$$\begin{aligned}
d[v] &< \delta(s, v) && \text{supposition} \\
&\leq \delta(s, u) + \delta(u, v) && \text{triangle inequality} \\
&\leq \delta(s,u) + w(u, v) && \text{sh. path} \leq \text{specific path} \\
&\leq d[u] + w(u, v) && v \text{ is first violation}
\end{aligned}$$

Contradiction.

# Correctness (ii)

Lemma:

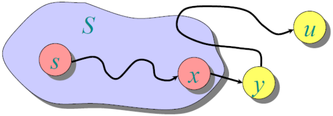
- Let $u$ be $v$'s predecessor on a shortest path from $s$ to $v$.
- Then, if $d[u] = \delta(s, u)$, we have $d[v] = \delta(s, v)$ after the relaxation of edge $(u, v)$.

Proof:

- Observe that $\delta(s, v) = \delta(s, u) + w(u, v)$.
- Suppose that $d[v] > \delta(s, v)$ before relaxation (else: done).
- Then, $d[v] > \delta(s, v) = \delta(s, u) + w(u, v) = d[u] + w(u, v)$ (if clause in the algorithm).
- Thus, the algorithm sets $d[v] = d[u] + w(u, v) = \delta(s, v)$.

# Correctness (iii)

Theorem:
Dijkstra's algorithm terminates with $d[v] = \delta(s, v)$ for all $v \in V$.




Proof:

- It suffices to show that $d[v] = \delta(s, v)$ for every $v \in V$ when $v$ is added to $S$.
- Suppose $u$ is the first vertex added to $S$ with $d[u] > \delta(s, u)$.
- Let $y$ be the first vertex in $V \setminus S$ along the shortest path from $s$ to $u$, and let $x$ be its predecessor.
- Then, $d[x] = \delta(s, x)$ and $d[y] = \delta(s, y) \leq \delta(s, u) < d[u]$.
- But we chose $u$ such that $d[u] \leq d[y]$. Contradiction.

## Complexity Analysis

$$
\left.\begin{array}{l}
\textbf{while } Q \neq \emptyset \\
\quad \textbf{do } u \leftarrow \text{EXTRACT-MIN}(Q) \\
\qquad S \leftarrow S \cup \{u\} \\
\qquad \left.\begin{array}{l}
\textbf{for each } v \in Adj[u] \\
\quad \textbf{do if } d[v] > d[u] + w(u, v) \\
\qquad \textbf{then } d[v] \leftarrow d[u] + w(u, v)
\end{array}\right\} \begin{array}{c} degree(u) \\ \text{times} \end{array}
\end{array}\right\} \begin{array}{c} |V| \\ \text{times} \end{array}
$$

- Similar to Prim's minimum spanning tree algorithm, we get the computation time

$$\Theta(V \cdot T_{\text{EXTRACT-MIN}} + E \cdot T_{\text{DECREASE-KEY}})$$

- Hence, depending on what data structure we use, we get the same computation times as for Prim's algorithm.

# Unweighted Graphs

- Suppose that we have an unweighted graph, i.e., the weights $w(u, v) = 1$ for all $(u, v) \in E$.
- Can we improve the performance of Dijkstra's algorithm?
- Observation: The vertices in our data structure $Q$ are processed following the FIFO principle.
- Hence, we can replace the min-priority queue with a queue.
- This leads to a breadth-first search.

# BFS Algorithm

```
d[s] := 0
for each v ∈ V\{s}
  d[v] := infinity
Enqueue (Q,s)
while Q != Ø
  u := Dequeue(Q)
  for each v ∈ Adj[u]
      if d[v] = infinity
      then d[v] := d[u] + 1
           pi[v] :=u
           Enqueue(Q,v)
```

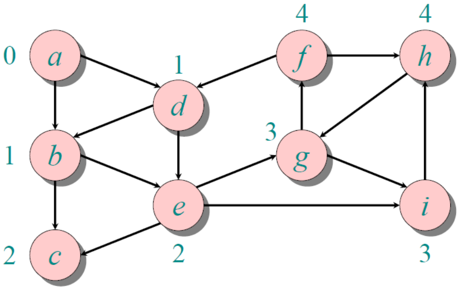
# Analysis: BFS Algorithm

Correctness:

- The FIFO queue $Q$ mimics the min-priority queue in Dijkstra's algorithm.
- Invariant:
  If $v$ follows $u$ in $Q$, then $d[v] = d[u]$ or $d[v] = d[u] + 1$.
- Hence, we always dequeue the vertex with smallest $d$.

Time complexity:
$O(|V| T_{Dequeue} + |E| T_{Enqueue}) = O(|V| + |E|)$

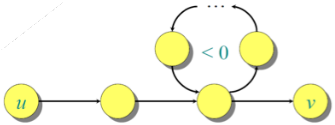
# Example: BFS Algorithm




$Q$: $a$ $b$ $d$ $c$ $e$ $g$ $i$ $f$ $h$

# Negative Weights

- We had postulated that all weights are nonnegative.
- How can we extend the algorithm to also handle negative entries?
- The problems are caused by negative weight cycles.




- Goal: Find shortest-path lengths from a source vertex $s \in V$ to all vertices $v \in V$ or determine the existence of a negative-weight cycle.

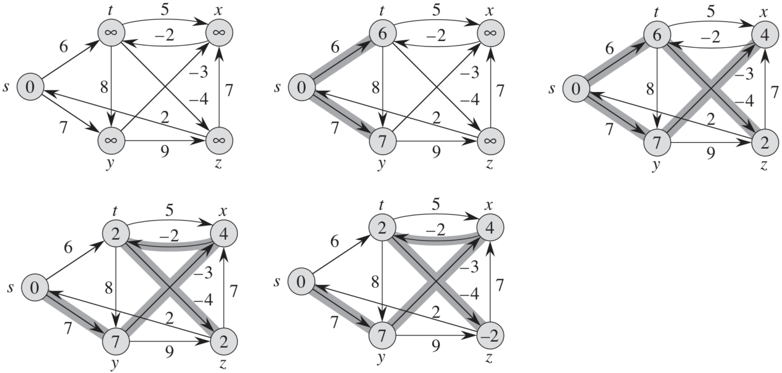
# Bellmann-Ford Algorithm

```
d[s] := 0
for each v ∊ V\{s}
  d[v] := infinity
for i:=1 to |V|-1
    for each (u,v) ∊ E
        if d[v] > d[u] + w(u,v)
        then  d[v] := d[u] + w(u,v)
              pi[v] :=u

for each (u,v) ∊ E
  if d[v] > d[u] + w(u,v)
    report existence of negative-weight cycle
```

Time complexity: $O(|V| \cdot |E|)$

# Example: Bellman-Ford Algorithm

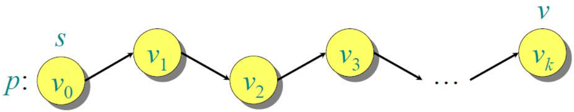



```
for i:=1 to |V|-1
    for each (u,v) ∈ E
        if d[v] > d[u] + w(u,v)
        then  d[v] := d[u] + w(u,v)
              pi[v] :=u
```

## Bellmann-Ford Algorithm: Correctness (1)

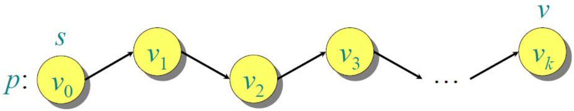
Theorem:
If $G = (V, E)$ contains no negative-weight cycles, then the Bellman-Ford algorithm terminates with $d[v] = \delta(s, v)$ for all $v \in V$.

Proof:
Let $v \in V$ be any vertex.
Consider a shortest path $p = (v_0, ..., v_k)$ from $s$ to $v$.
Then, $\delta(s, v_i) = \delta(s, v_{i-1}) + w(v_{i-1}, v_i)$ for $i = 1, ..., k$.



# Bellmann-Ford Algorithm: Correctness (2)

Initially, $d[v_0] = 0 = \delta(s, v_0)$.
According to our Lemma from Dijkstra's algorithm we have $d[v] \geq \delta(s, v)$, i.e., $d[v_0]$ is not changed.
After the 1st pass, we have $d[v_1] = \delta(s, v_1)$.
After the 2nd pass, we have $d[v_2] = \delta(s, v_2)$.
...
After the $k^{\text{th}}$ pass, we have $d[v_k] = \delta(s, v_k)$.
Since $G$ has no negative-weight cycles, $p$ is a simple path, i.e., it has $\leq |V| - 1$ edges.



# Detecting Negative-Weight Cycles

Corollary:
If a value $d[v]$ fails to converge after $|V| - 1$ passes, there exists a negative-weight cycle in $G$ reachable from $s$.