CSI - 3105 Design & Analysis of Algorithms
Course 7

Jean-Lou De Carufel

Fall 2019

Cyclic Directed Graphs

How to test if a directed graph is cyclic?

Cyclic Directed Graphs

How to test if a directed graph is cyclic?

Step 1 : Run DFS (including pre/post-numbers)

Step 2 : For each non-tree edge (v, u), test if

$$pre(u) < pre(v) < post(v) < post(u).$$

- If "yes" for at least one non-tree edge, return "cyclic".
- If "no" for all non-tree edges, return "acyclic".

Cyclic Directed Graphs

How to test if a directed graph is cyclic?

Step 1 : Run DFS (including pre/post-numbers)

Step 2 : For each non-tree edge (v, u), test if

$$pre(u) < pre(v) < post(v) < post(u).$$

- If "yes" for at least one non-tree edge, return "cyclic".
- If "no" for all non-tree edges, return "acyclic".

Running time: $O(|V| + |E|)$.

Back to Topological Ordering

Assume that $G = (V, E)$ is a directed acyclic graph.

How do we compute a topological ordering?

Back to Topological Ordering

Assume that $G = (V, E)$ is a directed acyclic graph.

How do we compute a topological ordering?

Step 1 : Run DFS (including pre/post-numbers)

Step 2 : Run Bucket Sort to sort the vertices by postnumber.

Step 3 : Obtain the topological ordering from the reverse sorted order of the postnumbers.

Back to Topological Ordering

Assume that $G = (V, E)$ is a directed acyclic graph.

How do we compute a topological ordering?

- Step 1 : Run DFS (including pre/post-numbers)
- Step 2 : Run Bucket Sort to sort the vertices by postnumber.
- Step 3 : Obtain the topological ordering from the reverse sorted order of the postnumbers.

How much time does it take for Bucket Sort?

Back to Topological Ordering

Assume that $G = (V, E)$ is a directed acyclic graph.

How do we compute a topological ordering?

Step 1 : Run DFS (including pre/post-numbers)

Step 2 : Run Bucket Sort to sort the vertices by postnumber.

Step 3 : Obtain the topological ordering from the reverse sorted order of the postnumbers.

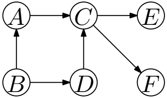
How much time does it take for Bucket Sort?

Total running time: $O(|V| + |E|)$.

Step 1 : Run DFS (including pre/post-numbers)

Step 2 : Run Bucket Sort to sort the vertices by postnumber.

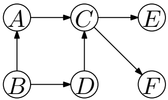
Step 3 : Obtain the topological ordering from the reverse sorted order of the postnumbers.

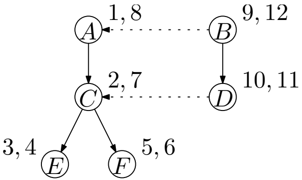


Step 1 : Run DFS (including pre/post-numbers)

Step 2 : Run Bucket Sort to sort the vertices by postnumber.

Step 3 : Obtain the topological ordering from the reverse sorted order of the postnumbers.

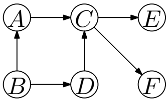


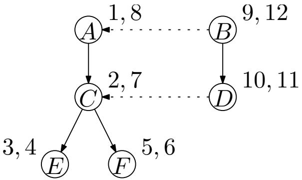


Step 1 : Run DFS (including pre/post-numbers)

Step 2 : Run Bucket Sort to sort the vertices by postnumber.

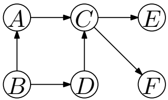
Step 3 : Obtain the topological ordering from the reverse sorted order of the postnumbers.

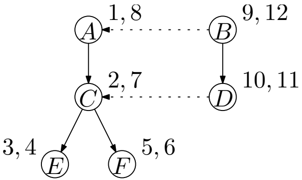




Sort by postnumber: E, F, C, A, D, B

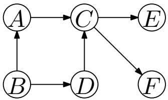
Step 1 : Run DFS (including pre/post-numbers)
Step 2 : Run Bucket Sort to sort the vertices by postnumber.
Step 3 : Obtain the topological ordering from the reverse sorted order of the postnumbers.

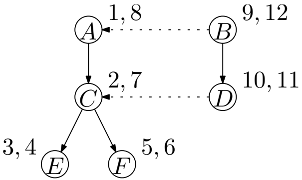




Sort by postnumber: E, F, C, A, D, B
Topologial ordering:
$B \quad D \quad A \quad C \quad F \quad E$

Step 1 : Run DFS (including pre/post-numbers)
Step 2 : Run Bucket Sort to sort the vertices by postnumber.
Step 3 : Obtain the topological ordering from the reverse sorted order of the postnumbers.





Sort by postnumber: E, F, C, A, D, B
Topologial ordering:

	B	D	A	C	F	E
#	1	2	3	4	5	6

Correctness: Let (v, u) be any edge in G.

To show: in topological sorting:
number of v < number of u

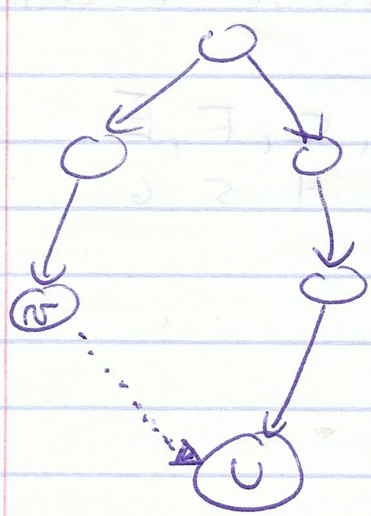
$$\text{post}(v) > \text{post}(u)$$

By contradiction. Assume $\text{post}(v) < \text{post}(u)$

Therefore, (v, u) is not a tree edge and not a forward edge (see page 73).

Since G is acyclic, (v, u) is not a back edge.

Hence, (v, u) is a cross edge.



But since $\text{post}(v) < \text{post}(u)$, (v, u) is not a cross edge, which is a contradiction.

$\square$

Shortest Paths

Input :

- A directed graph $G = (V, E)$, where each edge $(u, v) \in E$ has a weight $wt(u, v) > 0$.
- A vertex $s \in V$ (called the *source*).

Output :

- For each vertex $v \in V$,

$$\delta(s, v) = \text{lenght of a shortest path from } s \text{ to } v.$$

If all weights are equal, this is easy:

Shortest Paths

Input :

- A directed graph $G = (V, E)$, where each edge $(u, v) \in E$ has a weight $wt(u, v) > 0$.
- A vertex $s \in V$ (called the *source*).

Output :

- For each vertex $v \in V$,

$$\delta(s, v) = \text{lenght of a shortest path from } s \text{ to } v.$$

If all weights are equal, this is easy: use Breadth-first search!

Shortest Paths

General Approach

For each vertex $v \in V$, maintain variable

$$d(v) = \text{length of a shortest path from } s \text{ to } v \textit{ found so far}$$

Shortest Paths

General Approach

For each vertex $v \in V$, maintain variable

$$d(v) = \text{length of a shortest path from } s \text{ to } v \textit{ found so far}$$

At start,

$$d(v) = \begin{cases} 0 & \text{if } v = s \\ \infty & \text{if } v \neq s \end{cases}$$

Shortest Paths

General Approach

For each vertex $v \in V$, maintain variable

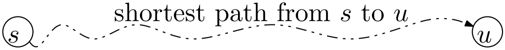
$$d(v) = \text{length of a shortest path from } s \text{ to } v \textit{ found so far}$$

At start,

$$d(v) = \begin{cases} 0 & \text{if } v = s \\ \infty & \text{if } v \neq s \end{cases}$$

Loop: Pick a vertex u for which $d(u) = \delta(s, u)$. For each edge u, v,

$$d(v) = \min\{d(v), d(u) + wt(u, v)\}$$



Shortest Paths

General Approach

For each vertex $v \in V$, maintain variable

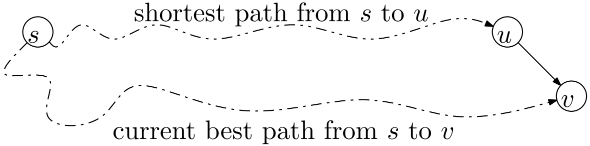
$$d(v) = \text{length of a shortest path from } s \text{ to } v \textit{ found so far}$$

At start,

$$d(v) = \begin{cases} 0 & \text{if } v = s \\ \infty & \text{if } v \neq s \end{cases}$$

Loop: Pick a vertex u for which $d(u) = \delta(s, u)$. For each edge u, v,

$$d(v) = \min\{d(v), d(u) + wt(u, v)\}$$



The hope is that at the end, for all vertices $v \in V$, we have

$$d(v) = \delta(s, v)$$

But how do we choose u? How do we know that $d(u) = \delta(s, u)$?

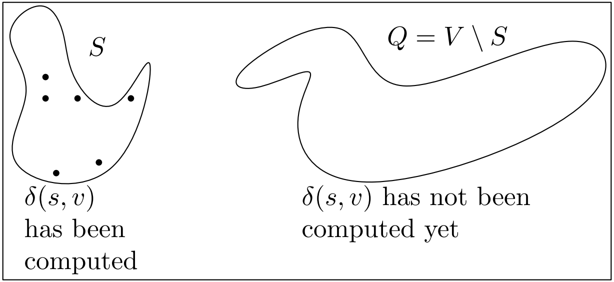
The hope is that at the end, for all vertices $v \in V$, we have

$$d(v) = \delta(s, v)$$

But how do we choose u? How do we know that $d(u) = \delta(s, u)$?

Maintain $S \subset V$ such that for all $v \in S$:

$$d(v) = \delta(s, v), \qquad \text{i.e., we know } \delta(s, v)$$



Start:

- $S = \{ \}$
- $Q = V$
- $d(s) = 0$
- $d(v) = \infty$ for each vertex $v \neq s$

Start:

- $S = \{\ \}$
- $Q = V$
- $d(s) = 0$
- $d(v) = \infty$ for each vertex $v \neq s$

One iteration: grow S by moving one vertex u from Q to S.

Start:

- $S = \{ \}$
- $Q = V$
- $d(s) = 0$
- $d(v) = \infty$ for each vertex $v \neq s$

One iteration: grow S by moving one vertex u from Q to S.

Which vertex u do we move?

Start:

- $S = \{ \}$
- $Q = V$
- $d(s) = 0$
- $d(v) = \infty$ for each vertex $v \neq s$

One iteration: grow S by moving one vertex u from Q to S.

Which vertex u do we move? The vertex $u \in Q$ for which $d(u)$ is minimum.

Start:

- $S = \{\ \}$
- $Q = V$
- $d(s) = 0$
- $d(v) = \infty$ for each vertex $v \neq s$

One iteration: grow S by moving one vertex u from Q to S.

Which vertex u do we move? The vertex $u \in Q$ for which $d(u)$ is minimum.

Later, we will prove that for this vertex u, $d(u) = \delta(s, u)$.

Start:

- $S = \{ \}$
- $Q = V$
- $d(s) = 0$
- $d(v) = \infty$ for each vertex $v \neq s$

One iteration: grow S by moving one vertex u from Q to S.

Which vertex u do we move? The vertex $u \in Q$ for which $d(u)$ is minimum.

Later, we will prove that for this vertex u, $d(u) = \delta(s, u)$.

Then for each edge (u, v),

$$d(v) = \min \{d(v), d(u) + wt(u, v)\}$$

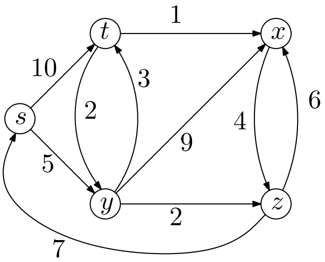
Algorithm *Dijkstra*(G, s)

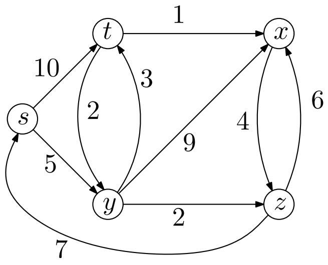
for each vertex $v \in V$ **do**
    $d(v) = \infty$
end for
$d(s) = 0$
$S = \{\,\}$
$Q = V$
while $Q \neq \{\,\}$ **do**
    $u =$ vertex in Q for which $d(u)$ is minimum

    delete u from Q
    insert u into S
    for each edge (u, v) **do**
        $d(v) = \min\{d(v), d(u) + wt(u, v)\}$
    end for
end while

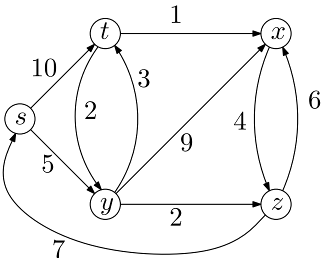
Algorithm *Dijkstra*(G, s)

for each vertex $v \in V$ **do**
    $d(v) = \infty$
end for
$d(s) = 0$
$S = \{\}$
$Q = V$
while $Q \neq \{\}$ **do**
    $u =$ vertex in Q for which $d(u)$ is minimum
    We will prove later that $d(u) = \delta(s, u)$.
    delete u from Q
    insert u into S
    for each edge (u, v) **do**
        $d(v) = \min\{d(v), d(u) + wt(u, v)\}$
    end for
end while


1
t
x
10
3
2
6
s
4
9
5
y
z
2
7

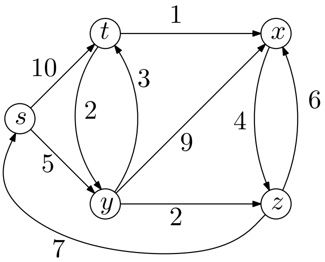


Q	s	t	x	y	z
d	0	∞	∞	∞	∞

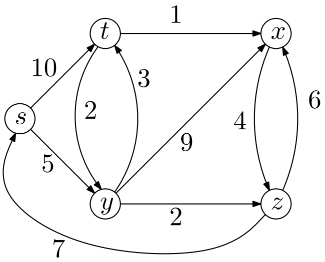


Q	s	t	x	y	z
d	0	∞	∞	∞	∞

- $u = s$
- $\delta(s, s) = d(s) = 0$
- delete s from Q
- update $d(t)$ and $d(y)$

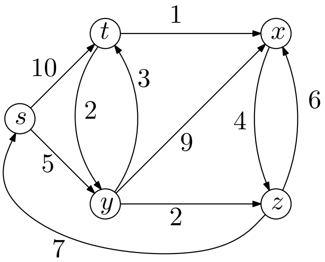


Q	t	x	y	z
d	10	∞	5	∞

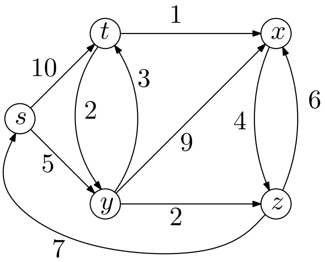


Q	t	x	y	z
d	10	∞	5	∞

- $u = y$
- $\delta(s, y) = d(y) = 5$
- delete y from Q
- update $d(t)$, $d(x)$ and $d(z)$

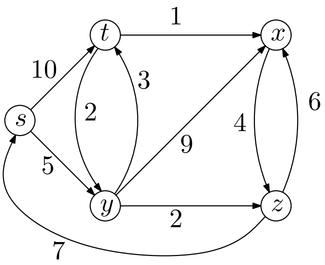


Q	t	x	z
d	8	14	7

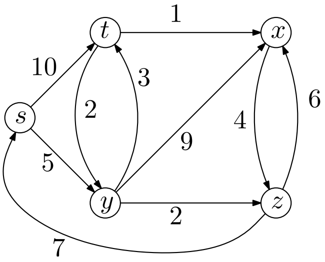


Q	t	x	z
d	8	14	7

- $u = z$
- $\delta(s, z) = d(z) = 7$
- delete z from Q
- update $d(x)$ and $d(s)$

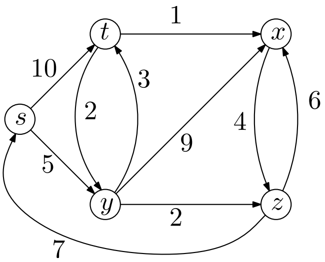


Q	t	x
d	8	13

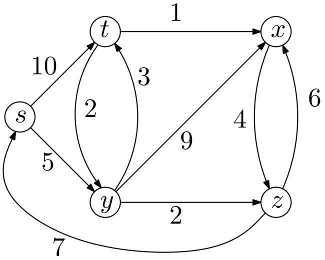


Q	t	x
d	8	13

- $u = t$
- $\delta(s, t) = d(t) = 8$
- delete t from Q
- update $d(x)$ and $d(y)$

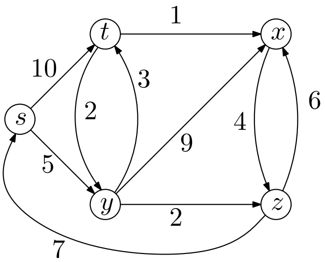


Q	x
d	9



Q	x
d	9

- $u = x$
- $\delta(s, x) = d(x) = 9$
- delete x from Q
- update $d(z)$



Q	x
d	9

- $u = x$
- $\delta(s, x) = d(x) = 9$
- delete x from Q
- update $d(z)$

$Q = \{\,\}$: done!

What is the running time of Dijkstra?

Algorithm *Dijkstra*(G, s)

for each vertex $v \in V$ **do**
    $d(v) = \infty$
end for
$d(s) = 0$
$S = \{\,\}$
$Q = V$
while $Q \neq \{\,\}$ **do**
    $u =$ vertex in Q for which $d(u)$ is minimum
    delete u from Q
    insert u into S
    for each edge (u, v) **do**
        $d(v) = \min\{d(v), d(u) + wt(u, v)\}$
    end for
end while

Let $n = |V|$ and $m = |E|$.

Let $n = |V|$ and $m = |E|$.

Store Q in a min-heap, where the key of each vertex is $d(v)$.

Let $n = |V|$ and $m = |E|$.

Store Q in a min-heap, where the key of each vertex is $d(v)$.

Initialization: $O(n)$ (including the time to build the heap storing $Q = V$).

Let $n = |V|$ and $m = |E|$.

Store Q in a min-heap, where the key of each vertex is $d(v)$.

Initialization: $O(n)$ (including the time to build the heap storing $Q = V$).

One iteration:

Let $n = |V|$ and $m = |E|$.

Store Q in a min-heap, where the key of each vertex is $d(v)$.

Initialization: $O(n)$ (including the time to build the heap storing $Q = V$).

One iteration:

- Find u and delete it from Q.

Let $n = |V|$ and $m = |E|$.

Store Q in a min-heap, where the key of each vertex is $d(v)$.

Initialization: $O(n)$ (including the time to build the heap storing $Q = V$).

One iteration:

- Find u and delete it from Q.

$$extract\_min : O(\log(n)) \text{ time}$$

Let $n = |V|$ and $m = |E|$.

Store Q in a min-heap, where the key of each vertex is $d(v)$.

Initialization: $O(n)$ (including the time to build the heap storing $Q = V$).

One iteration:

- Find u and delete it from Q.

$$\mathit{extract\_min} : O(\log(n)) \text{ time}$$

- For each edge (u, v), we update $d(v)$

Let $n = |V|$ and $m = |E|$.

Store Q in a min-heap, where the key of each vertex is $d(v)$.

Initialization: $O(n)$ (including the time to build the heap storing $Q = V$).

One iteration:

- Find u and delete it from Q.

$$\mathit{extract\_min} : O(\log(n)) \text{ time}$$

- For each edge (u, v), we update $d(v)$

$$\mathit{decrease\_key} : O(\log(n)) \text{ time}$$

Let $n = |V|$ and $m = |E|$.

Store Q in a min-heap, where the key of each vertex is $d(v)$.

Initialization: $O(n)$ (including the time to build the heap storing $Q = V$).

One iteration:

- Find u and delete it from Q.

$$extract\_min : O(\log(n)) \text{ time}$$

- For each edge (u, v), we update $d(v)$

$$decrease\_key : O(\log(n)) \text{ time}$$

Total time for one iteration:

$$O(\log(n)) + O(outdegree(u) \cdot \log(n))$$

Total running time:

$$O(n) + O\left(\sum_{u \in V} (\log(n) + outdegree(u) \cdot \log(n))\right)$$

Total running time:

$$
\begin{aligned}
& O(n) + O\left(\sum_{u \in V} (\log(n) + outdegree(u) \cdot \log(n))\right) \\
= \quad & O(n) + O\left(\log(n) \sum_{u \in V} (1 + outdegree(u))\right)
\end{aligned}
$$

Total running time:

$$
\begin{aligned}
& O(n) + O\left(\sum_{u \in V} (\log(n) + \mathit{outdegree}(u) \cdot \log(n))\right) \\
= \quad & O(n) + O\left(\log(n) \sum_{u \in V} (1 + \mathit{outdegree}(u))\right) \\
= \quad & O(n) + O\left(\log(n)(n + 2m)\right)
\end{aligned}
$$

Total running time:

$$
\begin{aligned}
& O(n) + O\left(\sum_{u \in V} (\log(n) + \mathit{outdegree}(u) \cdot \log(n))\right) \\
= \quad & O(n) + O\left(\log(n) \sum_{u \in V} (1 + \mathit{outdegree}(u))\right) \\
= \quad & O(n) + O\left(\log(n)(n + 2m)\right) \\
= \quad & O\left((m + n) \log(n)\right)
\end{aligned}
$$

Total running time:

$$
\begin{aligned}
& O(n) + O\left(\sum_{u \in V} (\log(n) + \mathit{outdegree}(u) \cdot \log(n))\right) \\
= \; & O(n) + O\left(\log(n) \sum_{u \in V} (1 + \mathit{outdegree}(u))\right) \\
= \; & O(n) + O\left(\log(n)(n + 2m)\right) \\
= \; & O\left((m + n)\log(n)\right)
\end{aligned}
$$

Note: Using a data structure called Fibonacci Heap to store Q, we can do $O(n\log(n) + m)$ time.