Data Structures and Algorithms

(ESO207)

Lecture 38

- **An interesting problem:**

shortest path from a source to destination

SHORTEST PATHS IN A GRAPH

A fundamental problem

Notations and Terminologies

A directed graph $G = (V, E)$

- $\omega: E \to R^+$
- Represented as **Adjacency** lists or **Adjacency** matrix
- $n = |V|$, $m = |E|$

Question: what is a path in G?

Answer: A sequence $v_1, v_2, \ldots, v_k$ such that $(v_i, v_{i+1}) \in E$ for all $1 \leq i < k$.

Length of a path P = $\sum_{e \in P} \omega(e)$

Notations and Terminologies

Definition:

The path from $\boldsymbol{u}$ to $\boldsymbol{v}$ of <u>minimum length</u> is called the **shortest path** from $\boldsymbol{u}$ to $\boldsymbol{v}$

Definition: **Distance** from $\boldsymbol{u}$ to $\boldsymbol{v}$ is the **<u>length</u>** of the shortest path from $\boldsymbol{u}$ to $\boldsymbol{v}$.

Notations:

$\boldsymbol{\delta}(\boldsymbol{u}, \boldsymbol{v})$: distance from $\boldsymbol{u}$ to $\boldsymbol{v}$.

$\boldsymbol{P}(\boldsymbol{u}, \boldsymbol{v})$: The shortest path from $\boldsymbol{u}$ to $\boldsymbol{v}$.

Problem Definition

Input: A directed graph $G = (V, E)$ with $\omega: E \rightarrow R^+$ and a source vertex $s \in V$

Aim:

- Compute $\delta(s, v)$ for all $v \in V \backslash \{s\}$
- Compute $P(s, v)$ for all $v \in V \backslash \{s\}$

What if $\omega(e) = 1$?

Do **BFS** from s
☺

Problem Definition

Input: A directed graph $G = (V, E)$ with $\omega: E \rightarrow R^{+}$ and a source vertex $s \in V$

Aim:

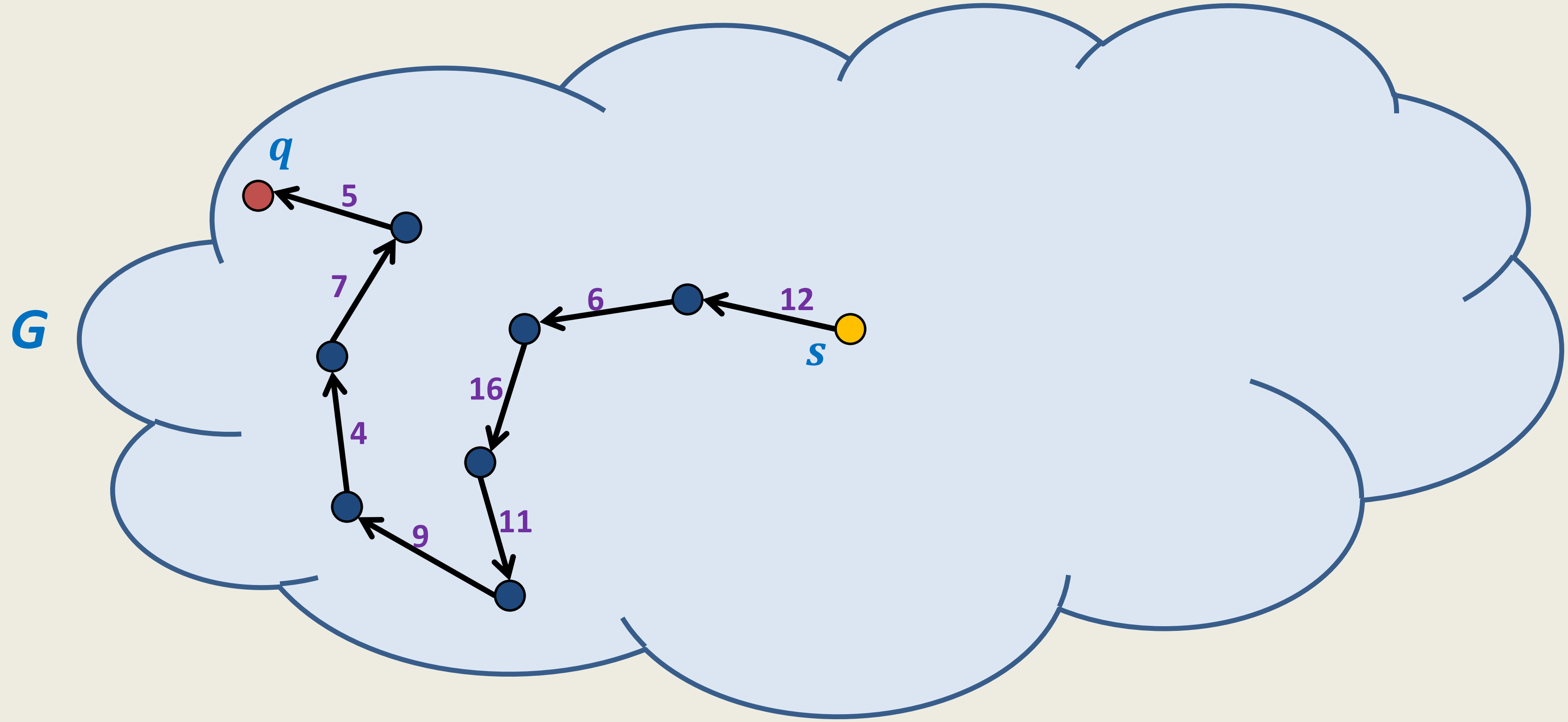
- Compute $\delta(s, v)$ for all $v \in V \backslash \{s\}$
- Compute $P(s, v)$ for all $v \in V \backslash \{s\}$

First algorithm : by **Edsger Dijkstra** in **1956**

And still the best …

I am sure you will be able to re-invent it yourself if you are asked right questions ☺ So get ready !

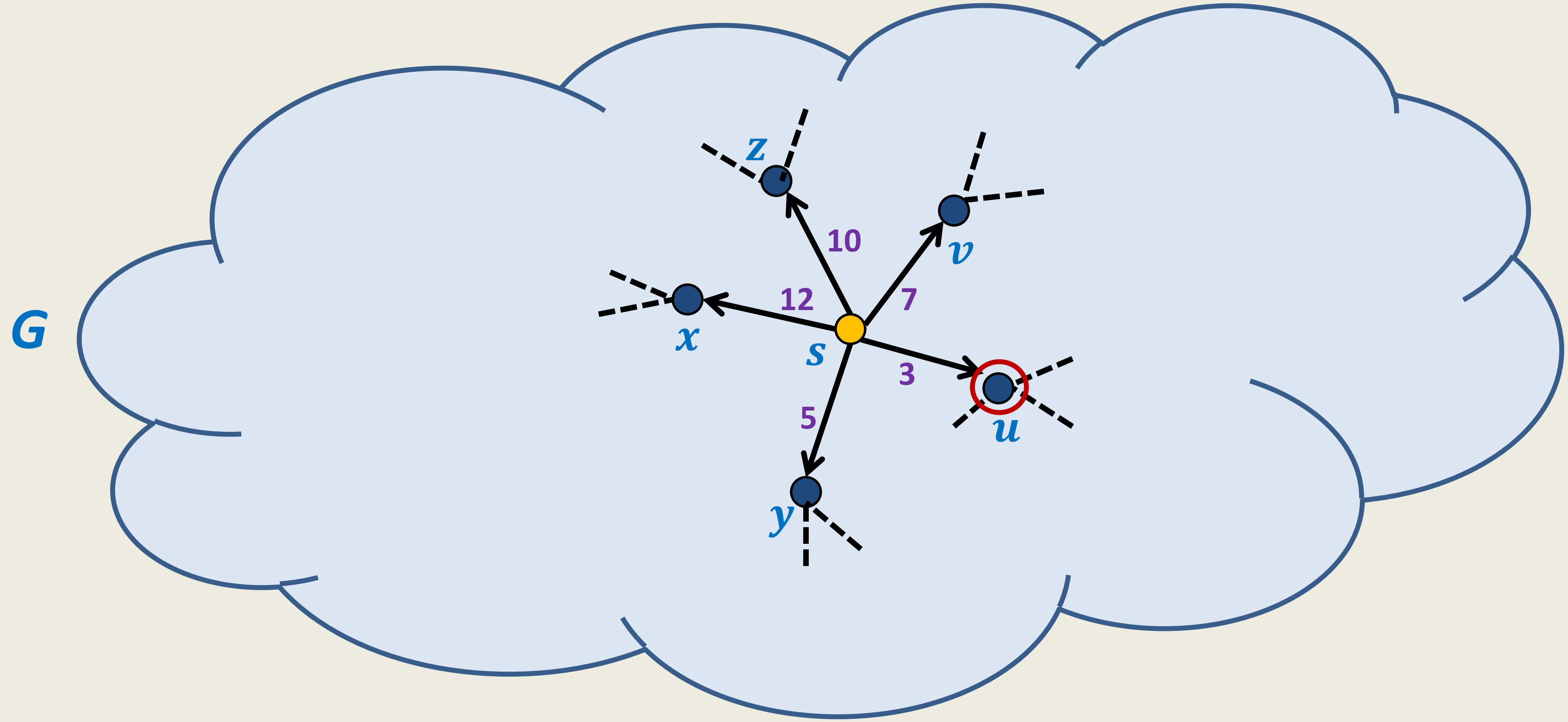
An example to get an insight into this problem



Inference:

The distance to any vertex depends upon global parameters.

An example to get an insight into this problem

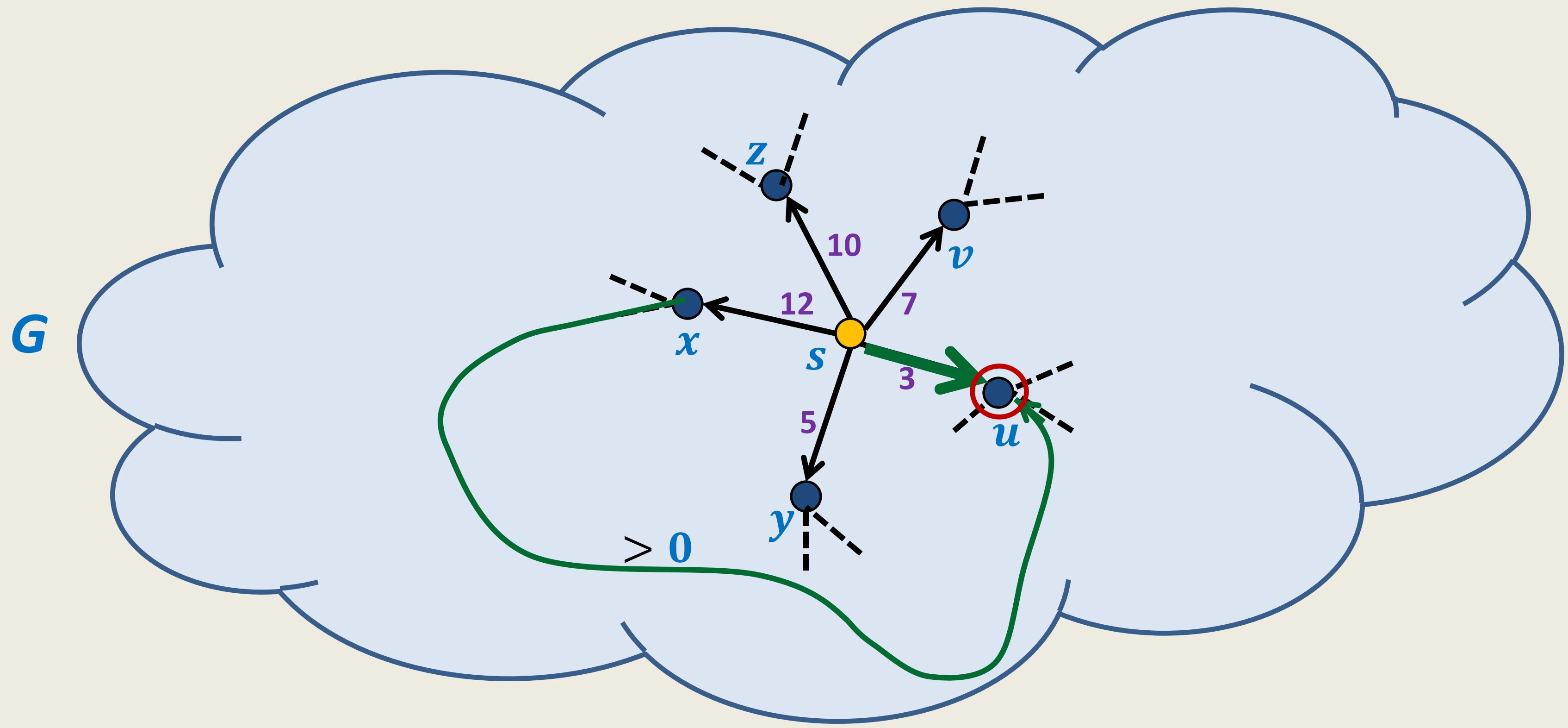


Question: Is there any vertex in this picture for which you are certain about the distance from s ?

Answer: vertex u.

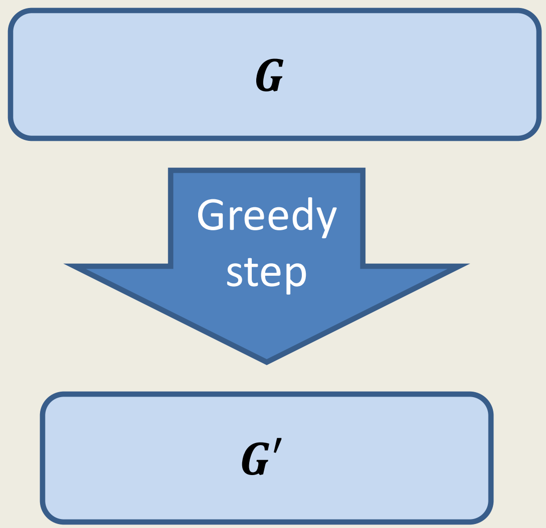
Give reasons.

An example to get an insight into this problem



➔ The shortest path to vertex u is edge (s,u).

Designing a greedy algorithm for shortest paths



instance of size n

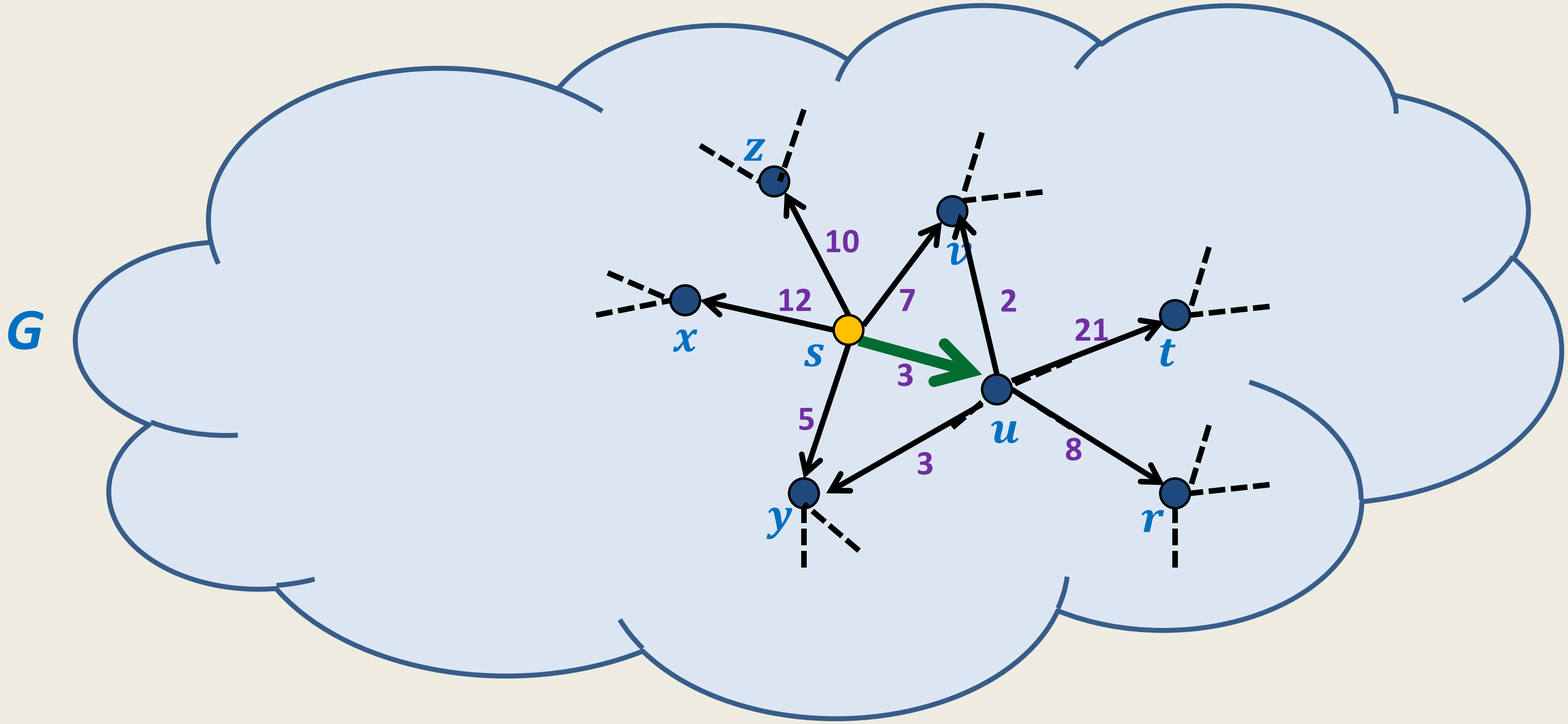
instance of size $< n$

1. **Establish** a relation between

and

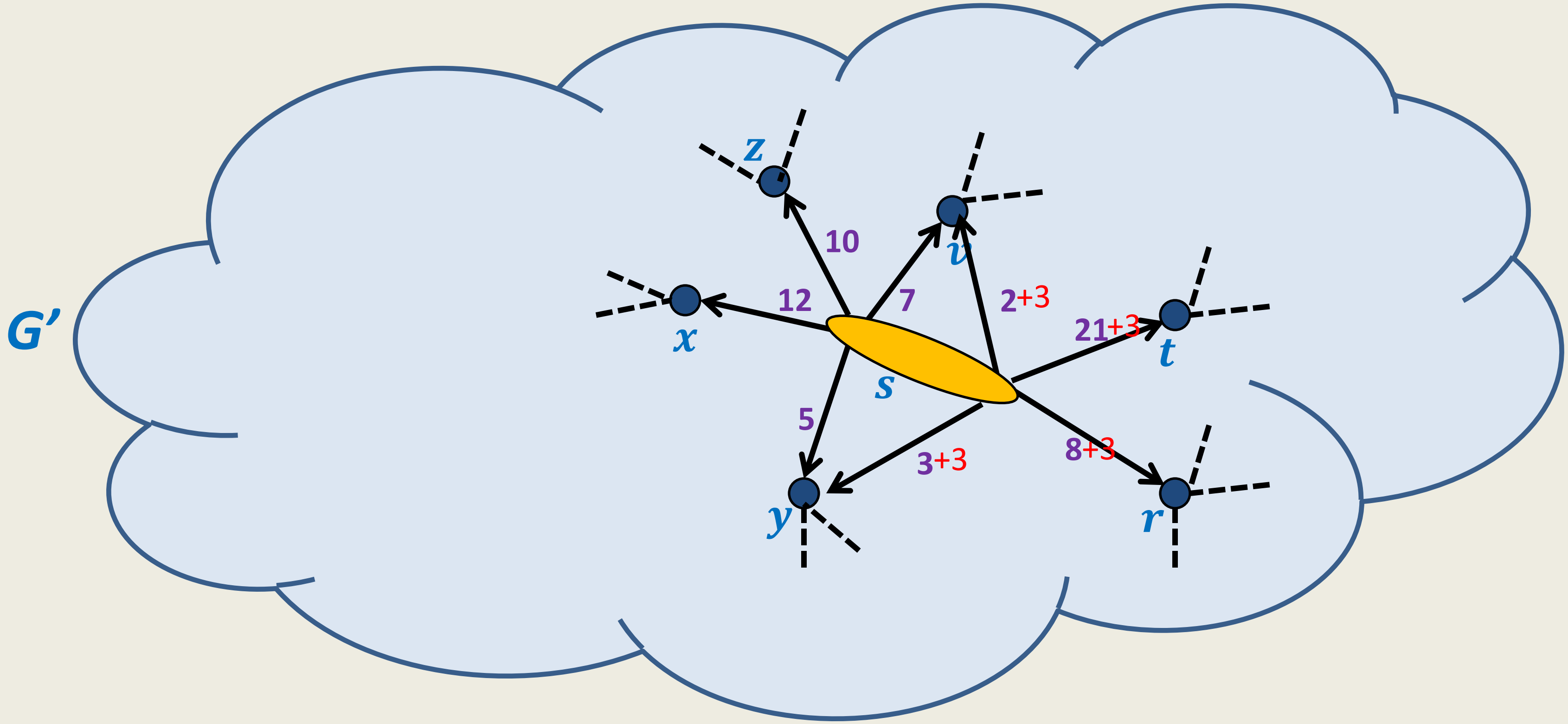
shortest paths in G'

An example to get an insight into this problem

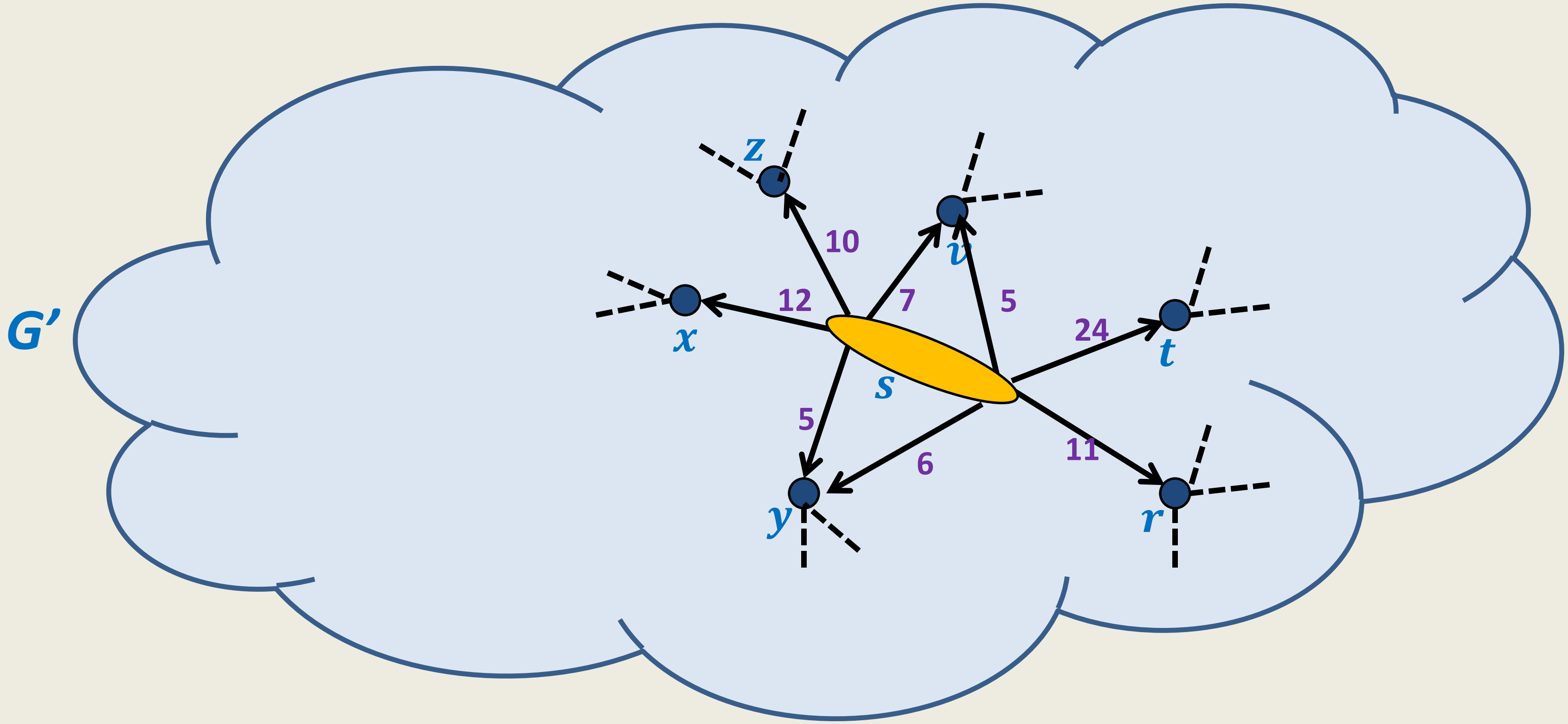


Question: Can you remove vertex u without affecting the distance from s ?

An example to get an insight into this problem



An example to get an insight into this problem



How to compute instance G'

Let (s,u) be the **least weight edge** from s in G=(V, E).

Transform G into G' as follows.

1. For each edge (u,x)ϵ E,

   add edge (s,x);

   ω(s,x)← ω(s,u) + ω(u,x);

2. In case of two edges from s to any vertex x

3. Remove vertex u.

Theorem: For each $v \in V \backslash \{s, u\}$,

$$\delta_G(s, v) = \delta_{G'}(s, v)$$

➔ an algorithm for **distances** from s

Can you see some negative points of this algorithm ?

Shortcomings of the algorithm

- **No insight** into the (beautiful) structure of shortest paths.

- **Just convinces** that we can solve the shortest paths problem in polynomial time.

- **Very few options** to improve the time complexity.

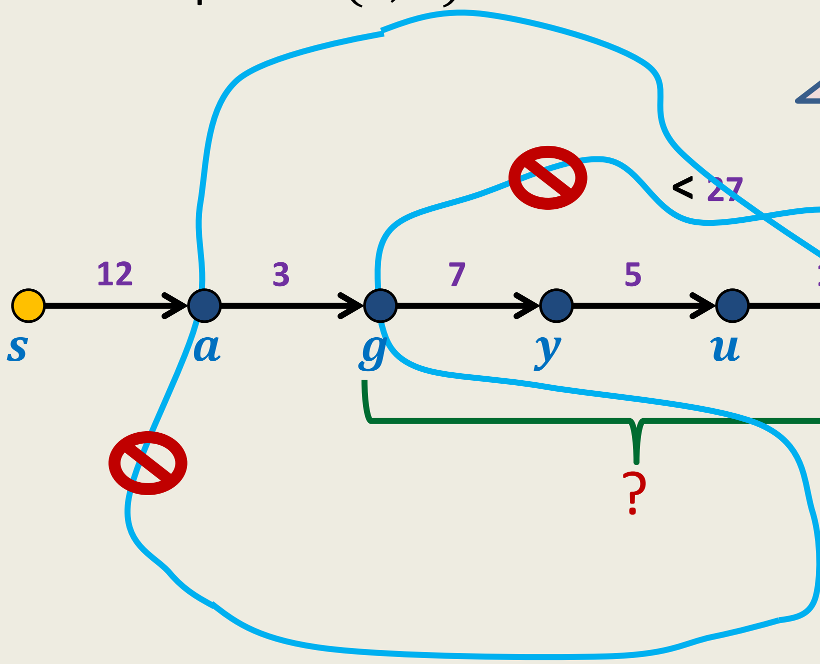
We shall now design a very **insightful** algorithm based on **properties** of shortest paths.

PROPERTY OF A SHORTEST PATH

Optimal subpath property

Consider any shortest path $P(s, v)$.

$\delta(s, v)$= 51



Anyway, give a proof of the lemma

Can you realize an assumption in your proof ?

Lemma 1: Every **subpath** of a shortest path is also a shortest path.

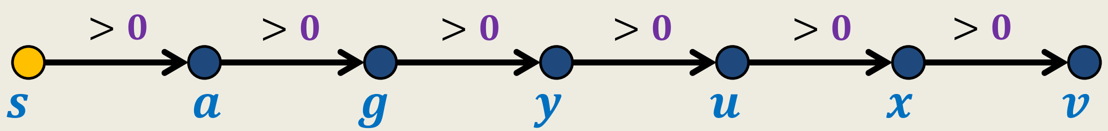
NOTE: Does the lemma use the fact that the edge weights are positive?

If yes, can you locate the exact place where it used it?

Homework: Write a complete and formal proof for **Lemma 1**

Exploiting the positive weight on edges

Consider once again a shortest path $\boldsymbol{P}(\boldsymbol{s},\boldsymbol{v})$.



➔ $\boldsymbol{\delta(s,a) < \delta(s,g) < \delta(s,y) < \delta(s,u) < \delta(s,x) < \delta(s,v)}$ (1)

➔ The first nearest vertex of $\boldsymbol{s}$ must be its **neighbor**.

More insights …

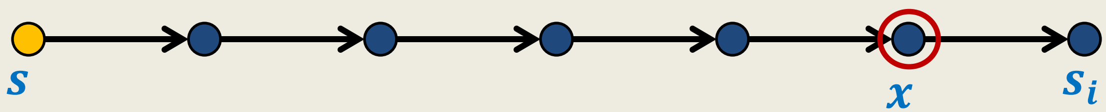
What can we say about x ?

Let s_i :

$s_0 = s$.

Consider the shortest path $P(s, s_i)$.

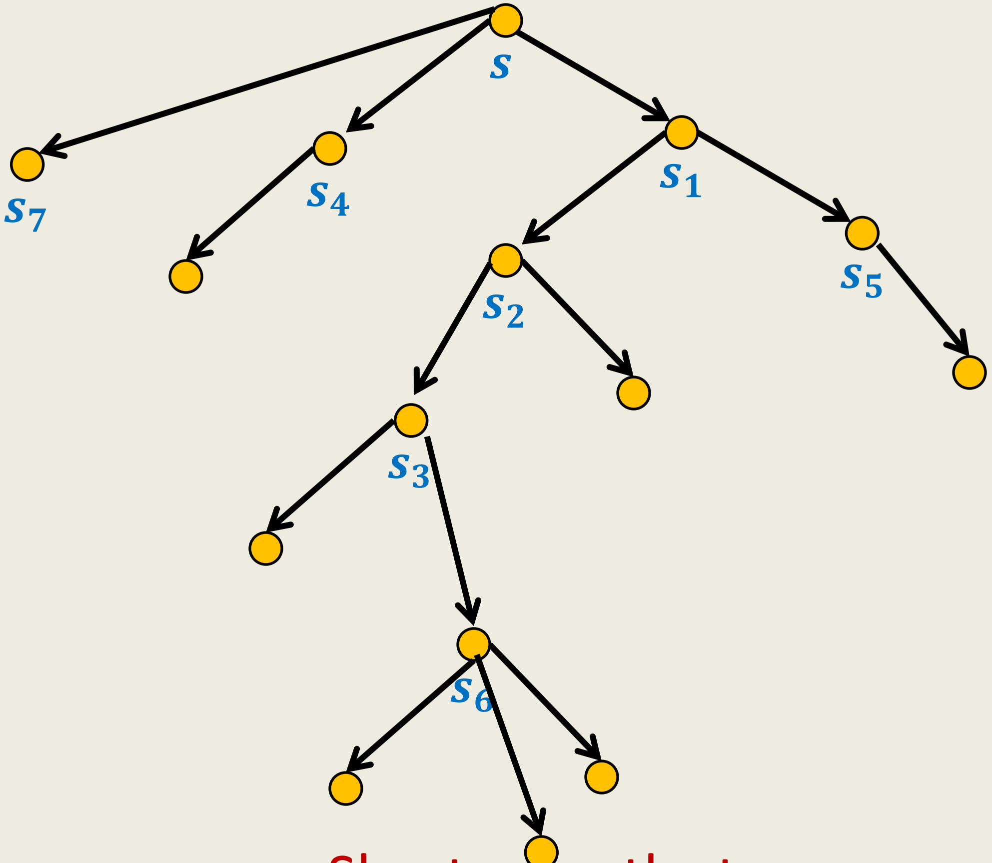
x must be s_j for some $j < i$.



Lemma 2: $P(s, s_i)$ must be of the form

What picture captures all shortest paths from s?

Complete picture of all shortest paths ?



Shortest paths tree

Designing the algorithm …

Lemma 2: $P(s, s_i)$ must be of the form $\underbrace{\boxed{s \rightsquigarrow s_j}}_{\text{shortest}} \rightarrow s_i$ for some $j < i$.

Question: Can we use **Lemma 2** to design an algorithm ?

Incremental way to compute shortest paths.

Ponder over it before going to the next slide ☺

Designing the algorithm …

Lemma 2: $P(s, s_i)$ must be of the form $\boxed{s \rightsquigarrow s_j} \rightarrow s_i$ for some $j < i$.

shortest

Question: Can we use **Lemma 2** to design an algorithm ?

Incremental way to compute shortest paths.

The next slide explains it precisely.

If shortest paths to s_j, $j < i$ is known, we can compute s_i.

But how ?

All we know is that

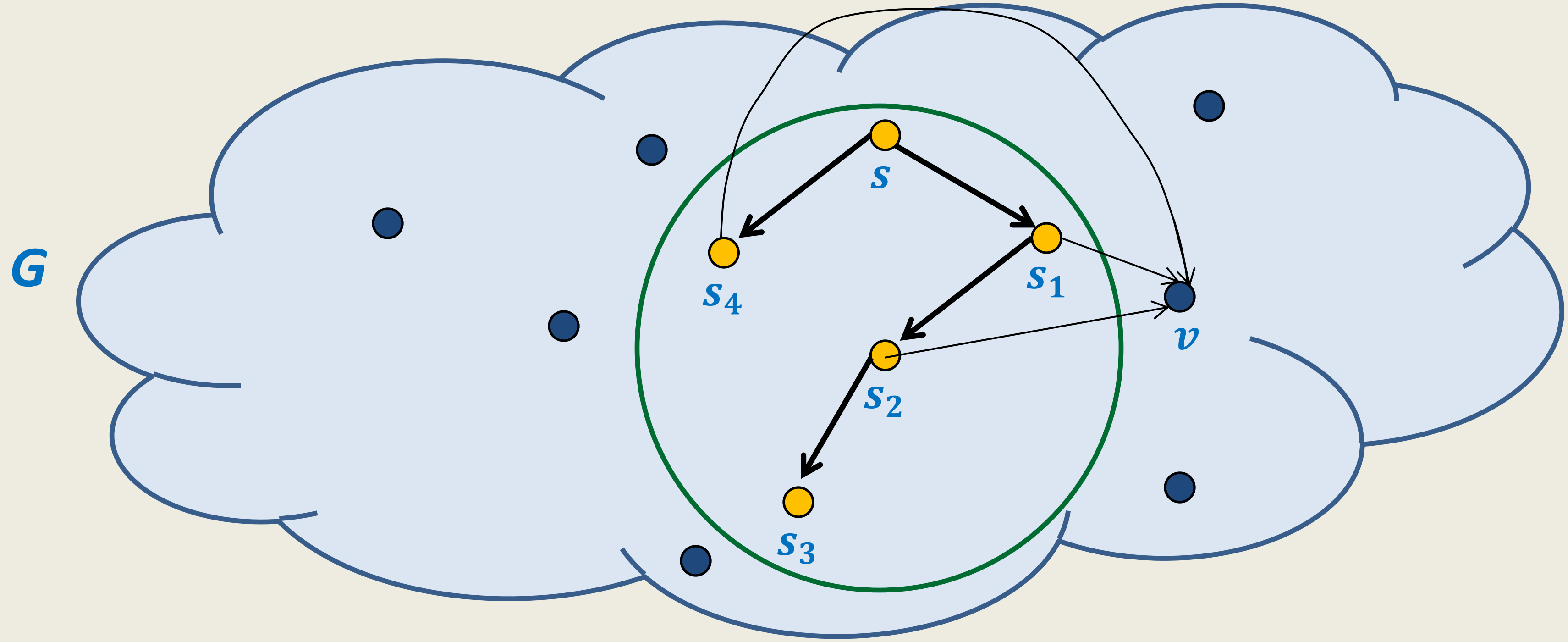
But which neighbor ?

Hint:

For each neighbor, compute some *label* based on **Lemma 2** s.t. s_i is the neighbour with least label.

Suppose we have computed $s_1, s_2, \ldots, s_{i-1}$.

We can compute s_i as follows.



For each $v \in V \setminus \{s_1, s_2, \ldots, s_{i-1}\}$

$$L(v) = \min_{((s_j, v) \in E)} \left(\delta(s, s_j) + \omega(s_j, v)\right)$$

s_i is the vertex with **minimum** value of L.

Dijkstra's algorithm

Dijkstra-algo(s,G)

{ $U \leftarrow V \setminus \{s\}$;

$S \leftarrow \{s\}$;

For i = 1 to $n - 1$ do

{ For each $v \in U$ do

{ $L(v) \leftarrow \infty$;

For each $(x, v) \in E$ with $x \in S$ do

$L(v) \leftarrow$ min($L(v)$, $\delta(s, x) + \omega(x, v)$)

}

$y \leftarrow$ vertex from U with **minimum** value of L;

$\delta(s, y) \leftarrow L(y)$;

move y from U to S;

}

In this algorithm, we first compute $\boldsymbol{L}(\boldsymbol{v})$ for each $\boldsymbol{v} \in \boldsymbol{U}$ and then find the vertex with the least $\boldsymbol{L}$ value.

Try to rearrange its statements so that in the beginning of each iteration, we have $\boldsymbol{L}$ values **computed already**.

This rearrangement will be helpful for improving the running time.

So please try it on your own first before viewing the next slide.

Dijkstra's algorithm

Dijkstra-algo(s,G)

{ $U \leftarrow V$; $L(v) \leftarrow \infty$ for all $v \in U$;

$L(s) \leftarrow 0$;

For i = 0 to $n-1$ do

{ $y \leftarrow$ vertex from U with **minimum** value of L;

$\delta(s,y) \leftarrow L(y)$;

move y from U to S;

For each $v \in U$ do

{ $L(v) \leftarrow \infty$;

For each $(x,v) \in E$ with $x \in S$ do

$L(v) \leftarrow$ **min**($L(v)$, $\delta(s,x) + \omega(x,v)$)

}

}

a lot of **re-computation**

Dijkstra's algorithm

Dijkstra-algo(s,G)

{ $U \leftarrow V$; $L(v) \leftarrow \infty$ for all $v \in U$;

$L(s) \leftarrow 0$;

For i = 0 to $n-1$ do

{ $y \leftarrow$ vertex from U with **minimum** value of L;

$\delta(s,y) \leftarrow L(y)$;

move y from U to S;

For each $v \in U$ do

{

For each $(x,v) \in E$ with $x \in S$ do

$L(v) \leftarrow$ **min**($L(v)$, $\delta(s,x) + \omega(x,v)$)

}

}

Only <u>neighbors</u> of y

What are the vertices whose L value may change in this iteration ?

Dijkstra's algorithm

Dijkstra-algo(s,G)

{ $U \leftarrow V$; $L(v) \leftarrow \infty$ for all $v \in U$;

$L(s) \leftarrow 0$;

For $i = 0$ to $n-1$ do

{ $y \leftarrow$ vertex from U with **minimum** value of L;

$\delta(s,y) \leftarrow L(y)$;

move y from U to S;

For each $(y,v) \in E$ with $v \in U$ do

{

$L(v) \leftarrow$ **min**($L(v)$, $\delta(s,y) + \omega(y,v)$)

}

}

1 **extract-min** operation

deg(y) **Decrease-key** operations

Time complexity of Dijkstra's algorithm

Total number of **extract-min** operation : n

Total **Decrease-key** operations : m

Using **Binary heap** to maintain the set U, the time complexity: O(m log n)

Theorem: Given a directed graph with positive weights on edges, we can compute all shortest paths from a given vertex in O(m log n) time.

Fibonacci heap supports **Decrease-key** in O(1) time and **extract-min** in O(log n).

➔ Total time complexity using Fibonacci heap: O(m + n log n)