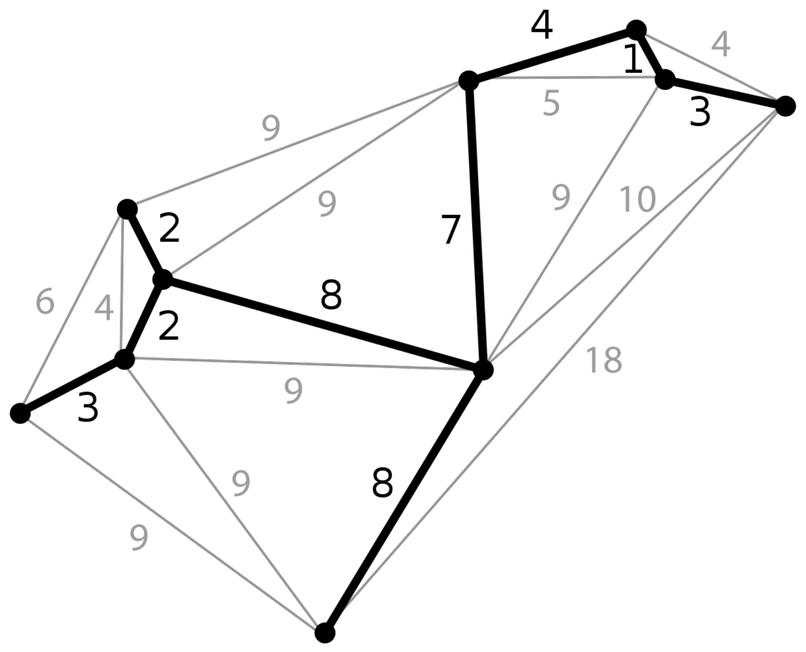
Data Structures and Algorithms (DSA) for AI

Katja Tuma



Single-source shortest path with Dijkstra

Problem formulation

Input: directed $G = (V, E), s \in V$, non-negative length l_e for each edge $e \in E$
Output: $dist(s, v)$, for all $v \in V$

Single-source shortest path with Dijkstra

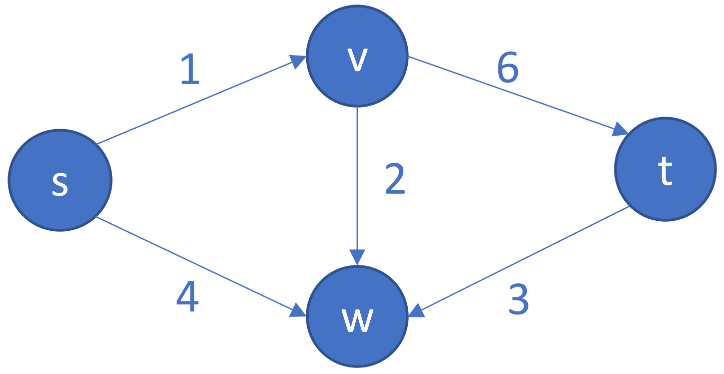
Problem formulation

Input: <u>directed</u> $G = (V, E), s \in V$, <u>non-negative</u> length l_e for each edge $e \in E$
Output: $dist(s, v)$, for all $v \in V$

<u>Dijkstra assumptions:</u> 1) directed, 2) non-negative edge length
<u>Additional assumption:</u> there exist paths from s to every vertex v in V.
<u>Main idea:</u> Explore the cheapest next edge (according to the Dijkstra score)

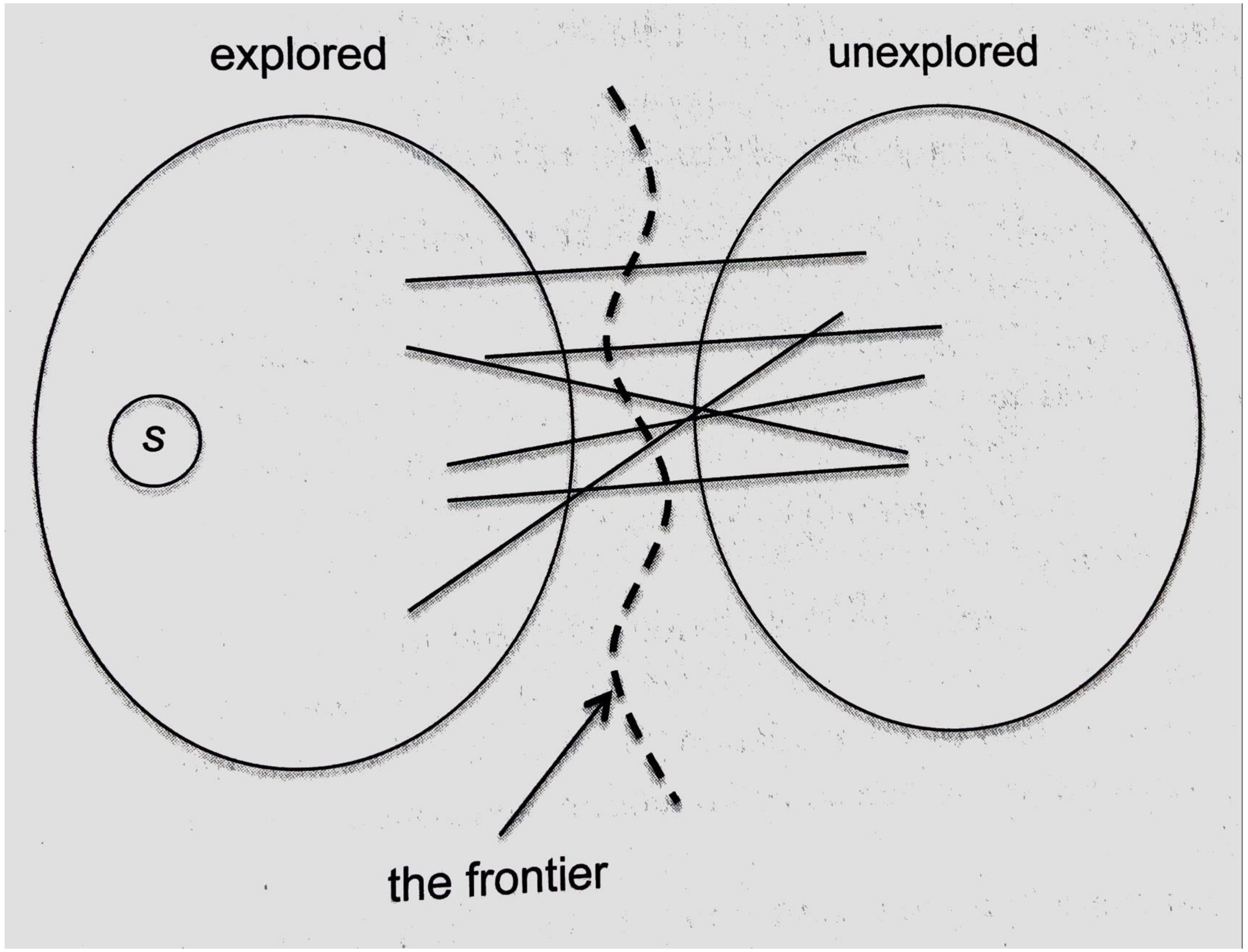
Single-source shortest path with Dijkstra

- X *explored*
- $d_{score} = len(v) + l_{vw}$
- $V - X$ *unexplored*
- *starting vertex = s*



How to choose an edge?

$$d_{score} = len(v) + l_{vw}$$



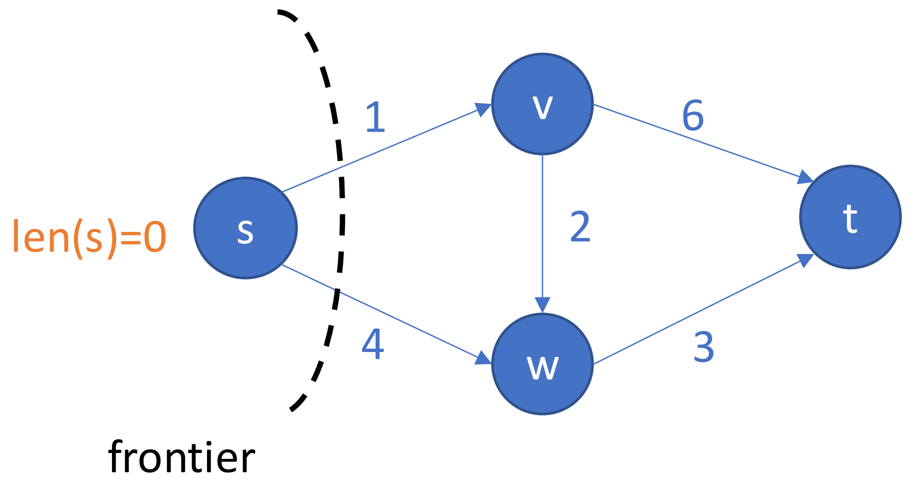
Dijkstra - pseudocode

Input: directed $G = (V, E)\ in\ adjacency\ list\ representation, s \in V$, non-negative length l_e for each edge $e \in E$
Postcondition: for every $v \in V$ the value $\mathrm{len}(v)$ is the true shortest path distance dist(s,v).

```
$X = \{s\}$                                              //Initialization
$len(s) = 0$,  $len(v) = +\infty$ for all $v \neq s$     //Initialization
//main loop
while there is an edge $(v, w)$, $v \in X$ and $w \notin X$ do:
    $(a, b) =$ such an edge minimizing $len(v) + l_{vw}$
    add $b$ to $X$
    $len(b) = len(a) + l_{ab}$
```

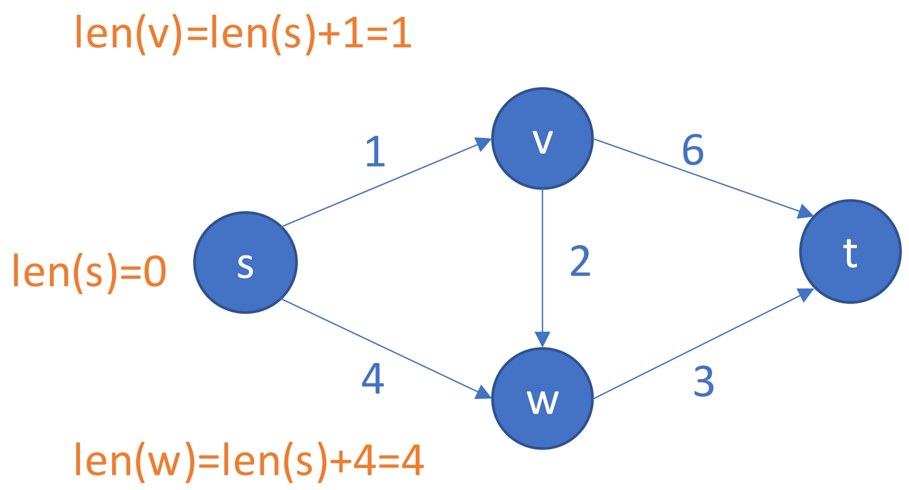
Single-source shortest path with Dijkstra

- $d_{score} = len(v) + l_{vw}$



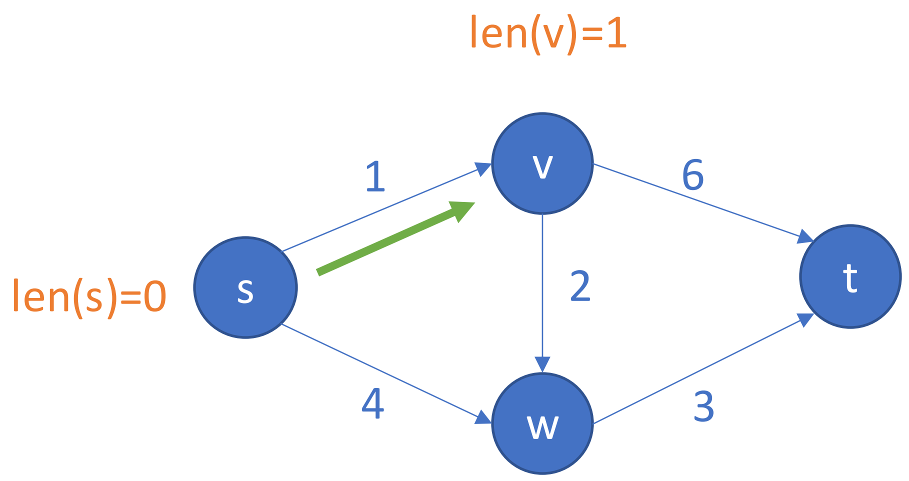
Single-source shortest path with Dijkstra

- $d_{score} = len(v) + l_{vw}$



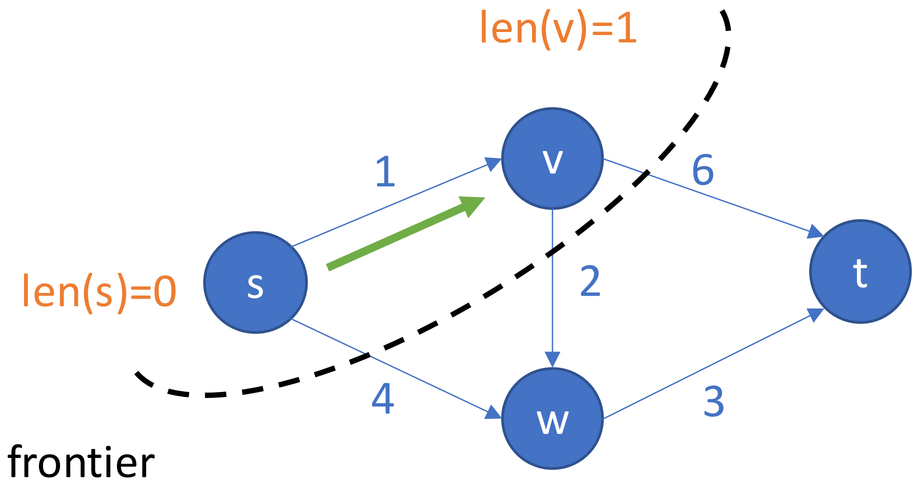
Single-source shortest path with Dijkstra

- $d_{score} = len(v) + l_{vw}$



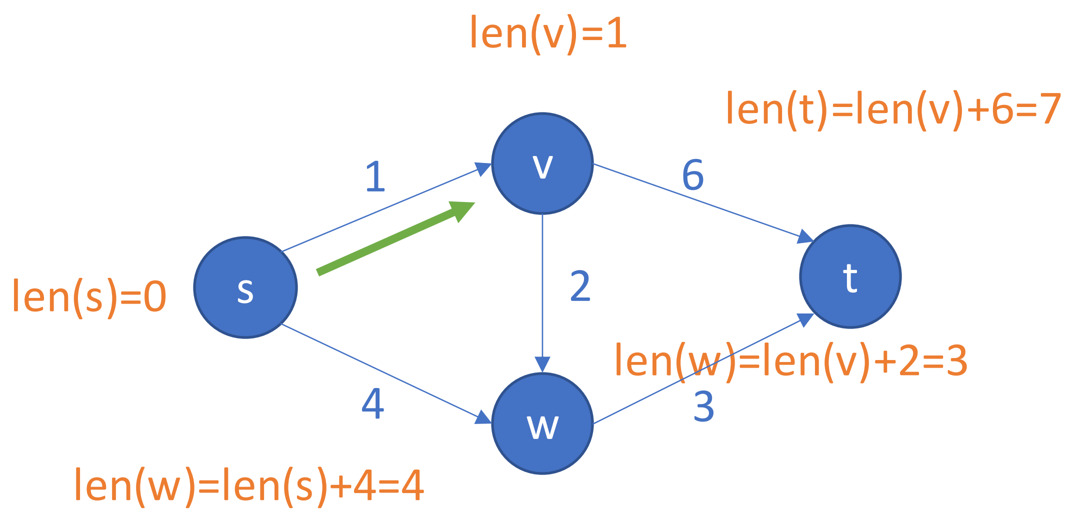
Single-source shortest path with Dijkstra

- $d_{score} = len(v) + l_{vw}$



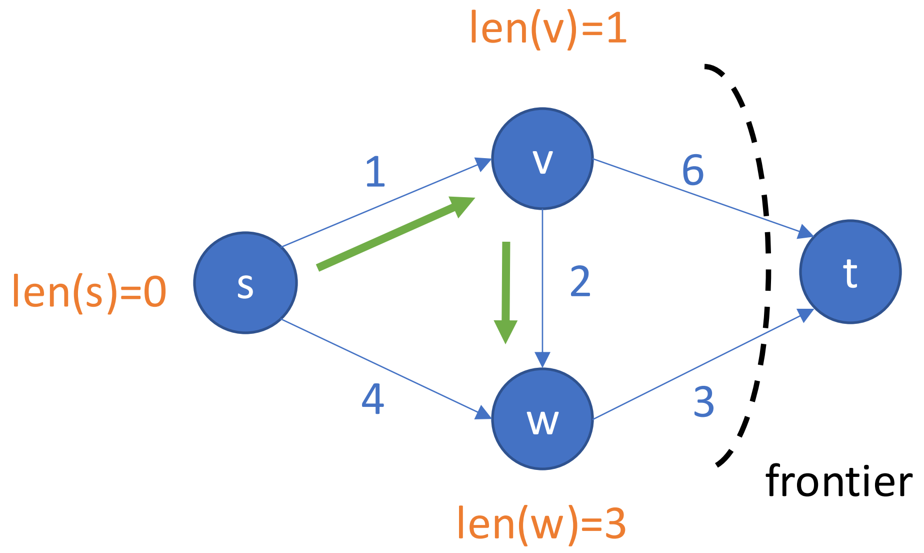
Single-source shortest path with Dijkstra

- $d_{score} = len(v) + l_{vw}$



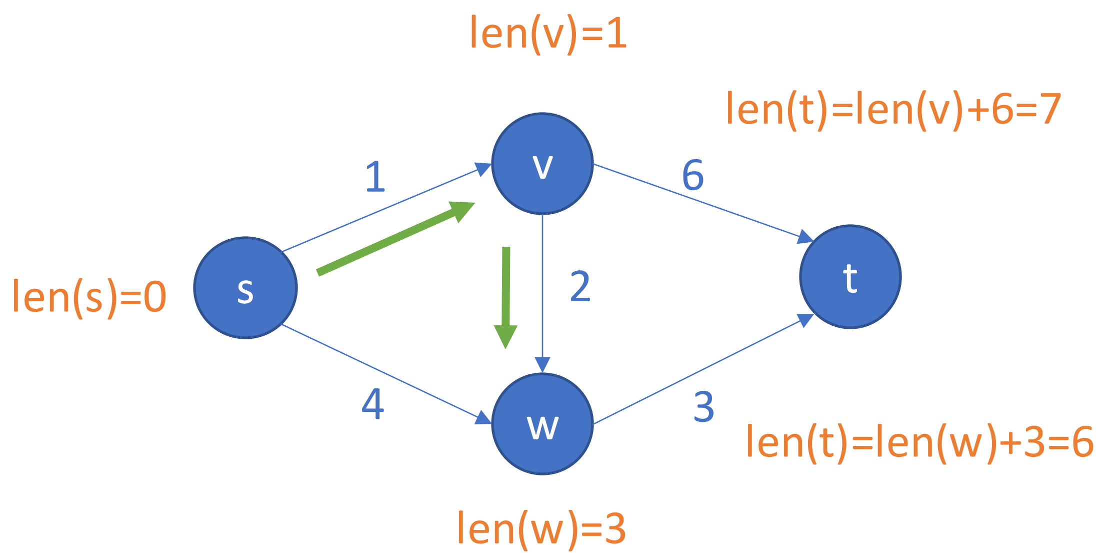
Single-source shortest path with Dijkstra

- $d_{score} = len(v) + l_{vw}$



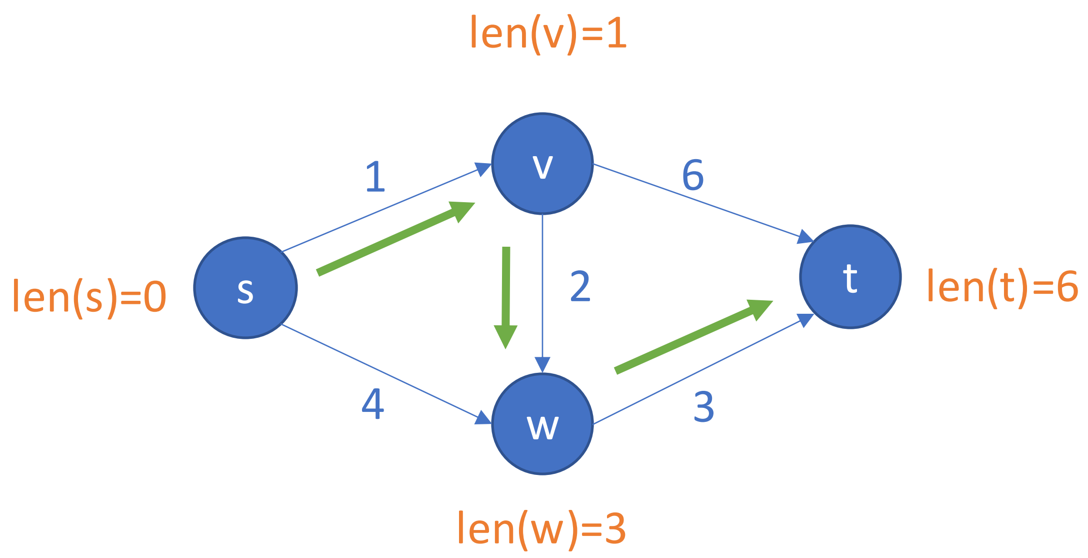
Single-source shortest path with Dijkstra

- $d_{score} = len(v) + l_{vw}$



Single-source shortest path with Dijkstra

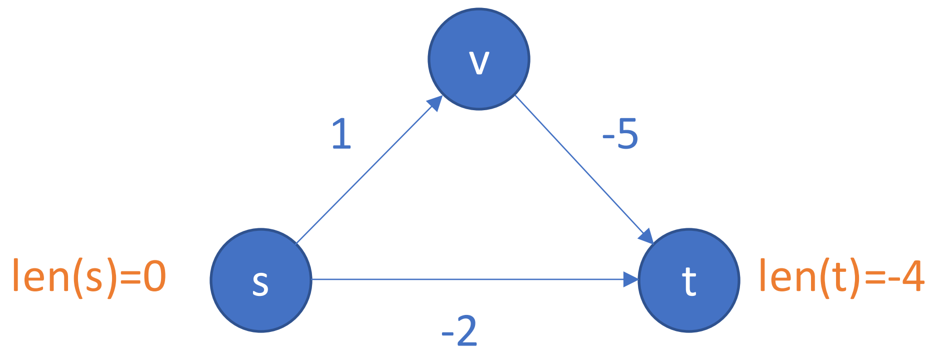
- $d_{score} = len(v) + l_{vw}$



Why not negative weights?

A bad example for Dijkstra:

- Two possible paths s→t:
  - s-v-t with cost 0+1+(-5)=-4
  - s-t with cost 0+(-2)=-2
- Since -4<-2, the true shortest path is -4

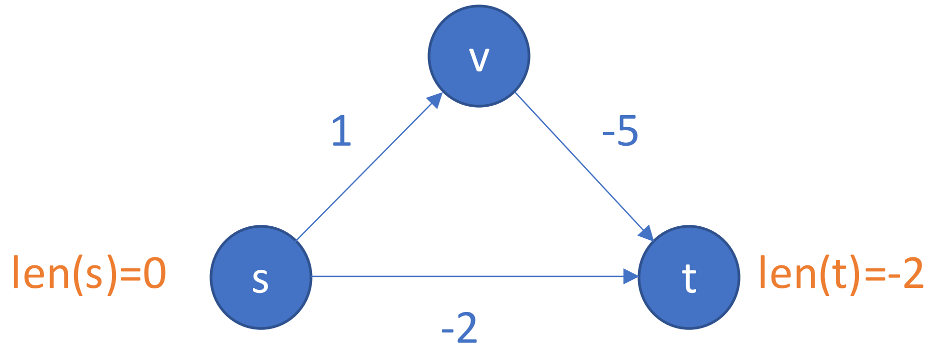


- What happens if we receive this graphs as input?

Why not negative weights?

What happens if we receive this graphs as input?

- X={s} and len(s)=0
- First iteration:
  - Dijkstra Scores:
    - (s,v) = 0+1 = 1
    - (s,t) = 0+(-2) = -2
  - Concludes that t is the cheapest edge.
  - Add t in X
- Returns the shortest path is s-t.
- Incorrect



Why not negative weights?

Can we reduce a problem to something we already know how to solve?

Why not simply add a large enough positive number to all edges?

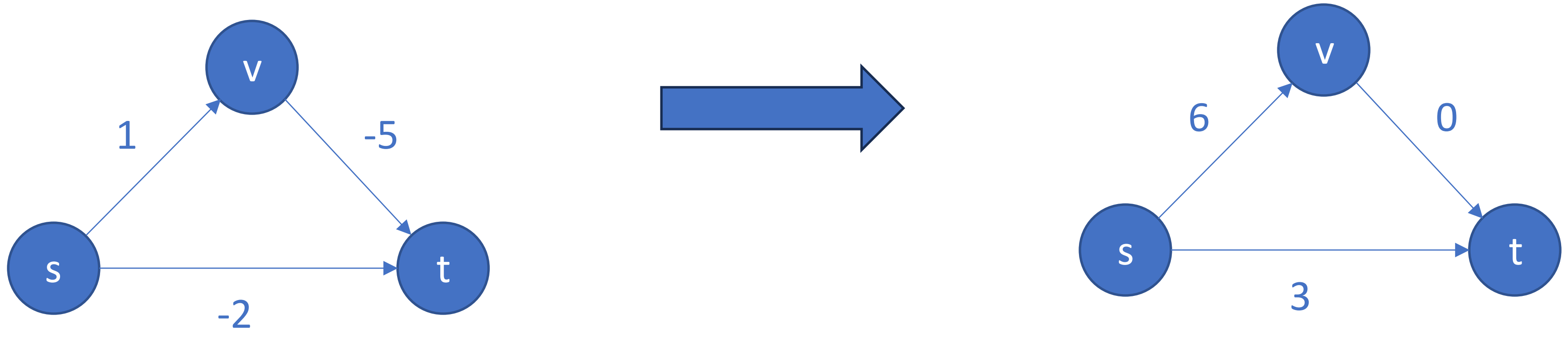
Eg +5

Why not negative weights?

Can we reduce a problem to something we already know how to solve?

Why not simply add a large enough positive number to all edges?

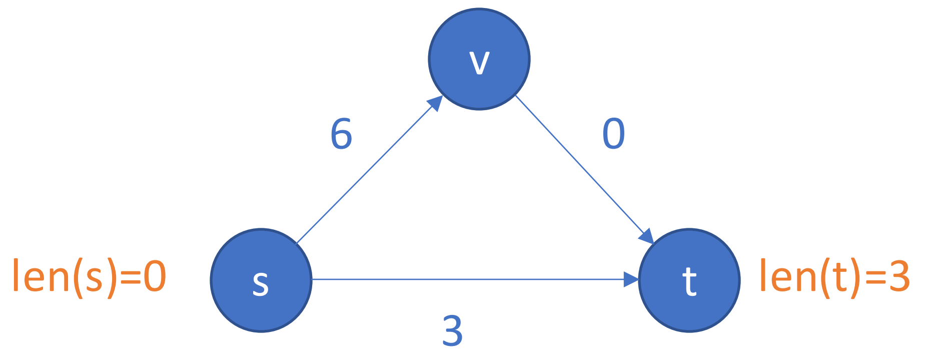
Eg +5



Why not negative weights?

What happens if we receive this graphs as input?

- X={s} and len(s)=0
- First iteration:
  - Dijkstra Scores:
    - (s,v) = 0+6 = 6
    - (s,t) = 0+3 = 3
  - Concludes that t is the cheapest edge.
  - Add t in X
- Returns the shortest path is s-t.
- Incorrect



Correctness of Dijkstra

- Proof by induction in the book

P(1) …shortest path to s=0 (base case)

P(k) set the inductive hypothesis and assume P(k) is true

P(k+1) use the IH and the assumption about non-negative edge lengths to reason that the Dijkstra computes true shortest paths for any vertex in the graph

- Online: http://www.algorithmsilluminated.org

Dijkstra - pseudocode

Input: directed $G = (V, E)$ *in adjacency list representation*, $s \in V$, non-negative length l_e for each edge $e \in E$
Postcondition: for every $v \in V$ the value $\mathrm{len}(v)$ is the true shortest path distance dist(s,v).

```
$X = \{s\}$
$\mathrm{len}(s) = 0,\ \ len(v) = +\infty$ for all $v \neq s$
//main loop
while there is an edge $(v, w)$, $v \in X$ and $w \notin X$ do:
    $(a, b) =$ such an edge minimizing $len(v) + l_{vw}$
    add $b$ to $X$
    $len(b) = len(a) + l_{ab}$
```

Can we do better?

How to improve?

- Minimum heap can help

Invariant

The key of a vertex $w \in (V - X)$ is the minimum Dijkstra score of an edge with tail $v \in X$ and head w OR $+\infty$ if no such edge exists.

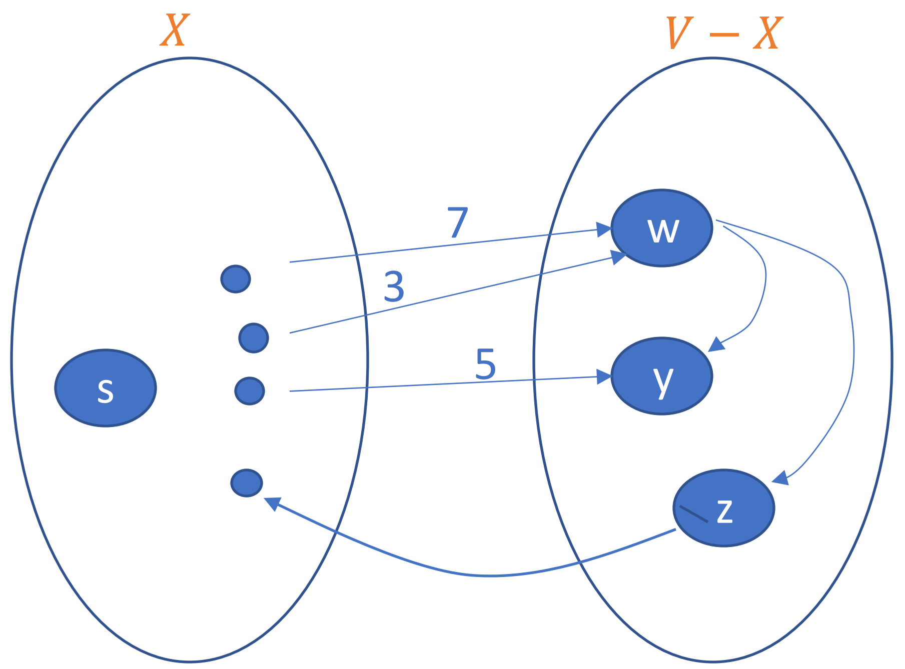
How choosing the next edge works?

- $key(w) = \min_{\substack{(v,w)\in E,\\ v\in X}} (len(v) \ + l_{vw})$

- Two-round knockout tournament
    1. 'local': 3 vs 7: 3 wins
    2. 'global': 3 vs 5: 3 wins

→key(w)=3

→Extract min from heap & add the next vertex (w) to X (frontier changes)



Dijkstra with heap – pseudocode

```
$X = \ empty\ set$; $H = empty\ set$
$key(s) = 0$
for every $v \ \neq s$ do
    $key(v) = +\infty$
//use heapify (insert all other vertices into H)
while H is not empty do
    $w \ = \ extractmin(H)$
    add $w$ to $X$
    $len(w) = key(w)$
    //UPDATE HEAP…pay the price
```

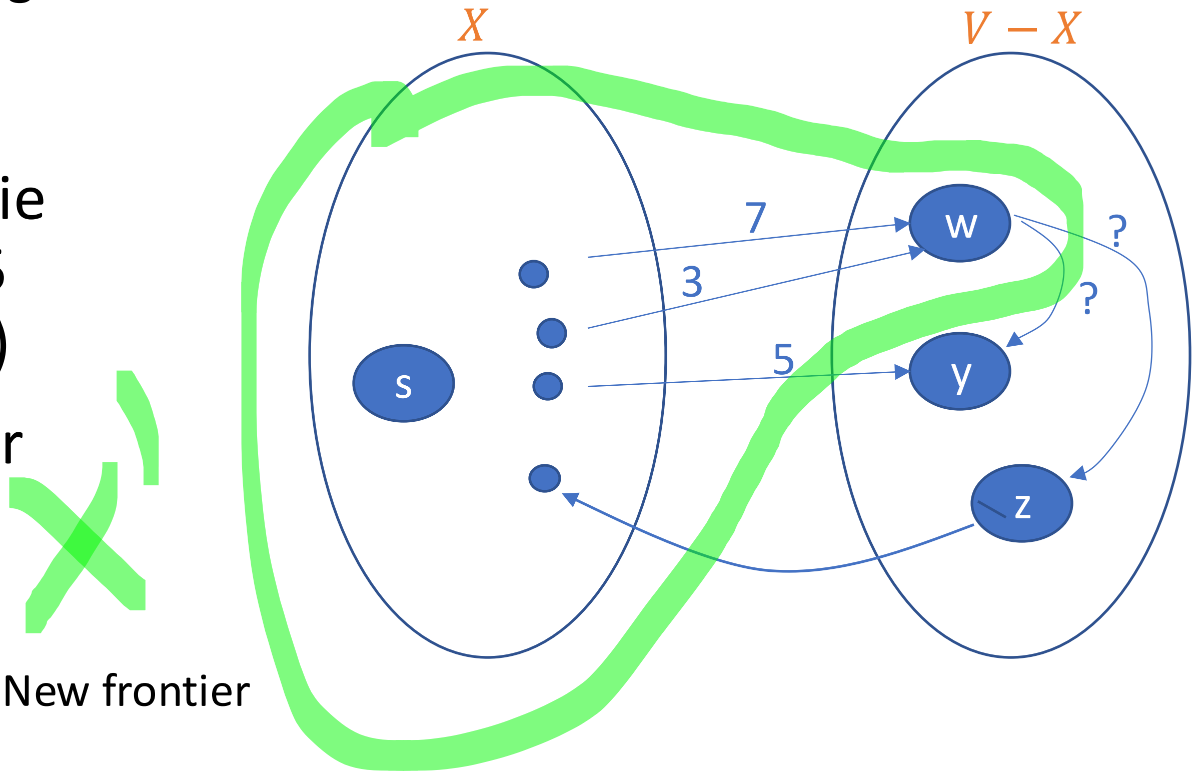
Maintaining the invariant

INVARIANT: The key of a vertex $w \in (V - X)$ is the minimum Dijkstra score of an edge with tail $v \in X$ and head w OR $+\infty$ if no such edge exists.

→We have to maintain the invariant, ie fix the heap (recalculate '?' key(y)=5 and key(z)=+inf need to be updated)

→There are new edges on the frontier

→They need to get their key recalculated in the heap



Dijkstra with heap – pseudocode

```
$X = empty\ set;\ H = empty\ set$
$key(s) = 0$
for every $v \neq s$ do
    $key(v) = +\infty$
//use heapify (insert all other vertices into H)
while H is not empty do
    $w = extractmin(H)$
    add $w$ to $X$
    $len(w) = key(w)$
    //UPDATE HEAP…pay the price
```

```
//UPDATE HEAP
foreach edge $(w, y)$ with $y \in V - X$ do:
    DELETE $y$ from $H$
    $key(y) = \min(key(y), len(w) + l_{vw})$
    INSERT $y$ into $H$
```

Running time analysis

You check: the running time is dominated by the heap operations! (HW)

What work is done for heap ops?

- (n-1) Extract mins (which triggers the heap update = delete+insert)

How many delete/insert?

- a vertex can have as many as n-1 outgoing edges (scary! That would mean nˆ2 heap operations). True for DENSE graphs

  →i.e., many "local tournaments"

- in general, much better. Remember we **only update the key if the tail vertex has been sucked into X**.
- **each edge only triggers at most one** Delete / Insert combo (if v added to X first)

<u>So:</u> # of heap operations is O((n-1)+m)=O(n+m). Since we assumed that there exist all paths (from s to any v), ie the graph is weakly connected, we know that m dominates n. So, we can simplify O(m).

*<u>So:</u> running time = O((m+n)logn) OR, simplified under the assumption O(m*log(n)) .*