CMPSC 465: LECTURE XV

Dijkstra's Algorithm

Ke Chen

October 03, 2025

Shortest path on weighted graphs

In many applications, having weights on edges is useful.

The edge weights could represent distances, cost, time, etc.

Second scenario Find shortest paths on weighted graphs.

Idea Suppose all the weights are positive integers, we can add dummy nodes to represent edge weights.

Shortest path on weighted graphs

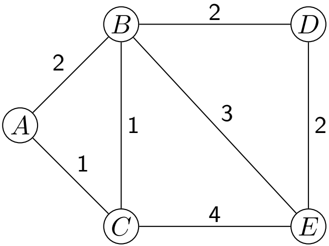
In many applications, having weights on edges is useful.

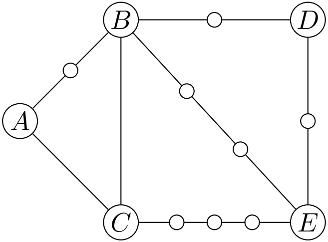
The edge weights could represent distances, cost, time, etc.

Second scenario Find shortest paths on weighted graphs.

Idea Suppose all the weights are positive integers, we can add dummy nodes to represent edge weights.

Example





Don't watch the pot, just take a nap

Problem with this approach?

Don't watch the pot, just take a nap

Problem with this approach?

- What about non-integer weights? Negative weights?

Don't watch the pot, just take a nap

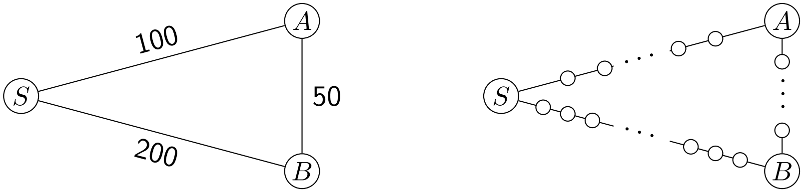
Problem with this approach?

- What about non-integer weights? Negative weights?
- The graph size can increase exponentially.

Don't watch the pot, just take a nap

Problem with this approach?

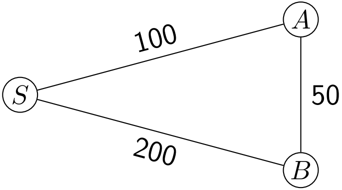
- What about non-integer weights? Negative weights?
- The graph size can increase exponentially.

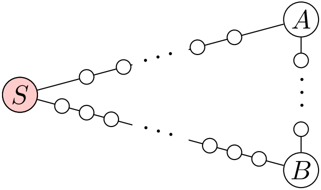


Don't watch the pot, just take a nap

Problem with this approach?

- What about non-integer weights? Negative weights?
- The graph size can increase exponentially.



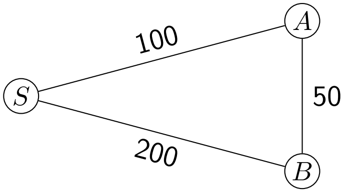


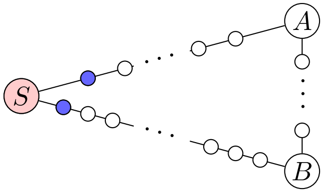
- Suppose that every second we discover one new layer.

Don't watch the pot, just take a nap

Problem with this approach?

- What about non-integer weights? Negative weights?
- The graph size can increase exponentially.



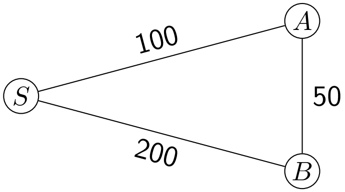


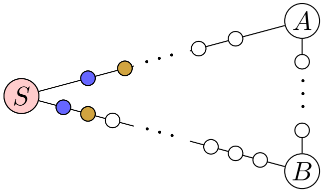
- Suppose that every second we discover one new layer.

Don't watch the pot, just take a nap

Problem with this approach?

- What about non-integer weights? Negative weights?
- The graph size can increase exponentially.



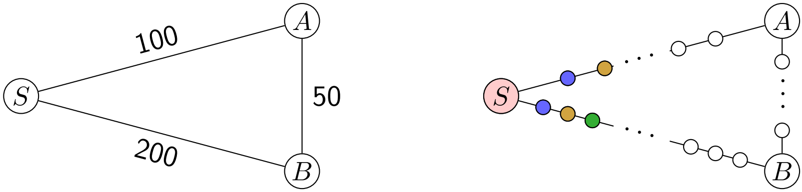


- Suppose that every second we discover one new layer.

Don't watch the pot, just take a nap

Problem with this approach?

- What about non-integer weights? Negative weights?
- The graph size can increase exponentially.

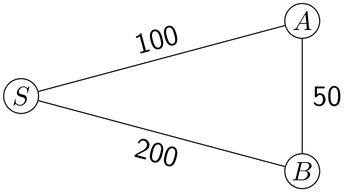


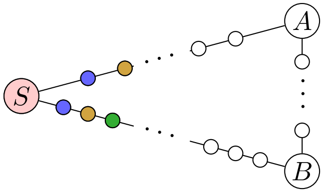
- Suppose that every second we discover one new layer.

Don't watch the pot, just take a nap

Problem with this approach?

- What about non-integer weights? Negative weights?
- The graph size can increase exponentially.



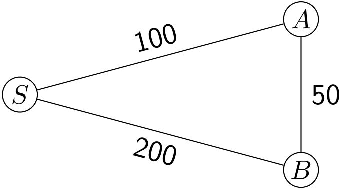


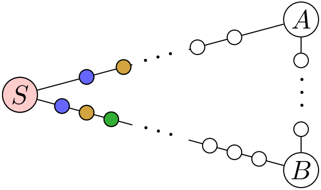
- Suppose that every second we discover one new layer.
- For the first 99 seconds, we are waiting for the dummy nodes we don't care about.

Don't watch the pot, just take a nap

Problem with this approach?

- What about non-integer weights? Negative weights?
- The graph size can increase exponentially.



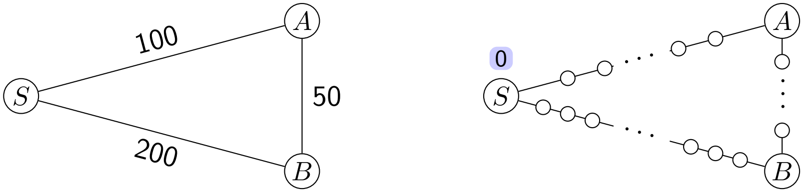


- Suppose that every second we discover one new layer.
- For the first 99 seconds, we are waiting for the dummy nodes we don't care about.
- We could set alarm clocks and go take a nap.

Don't watch the pot, just take a nap

Problem with this approach?

- What about non-integer weights? Negative weights?
- The graph size can increase exponentially.

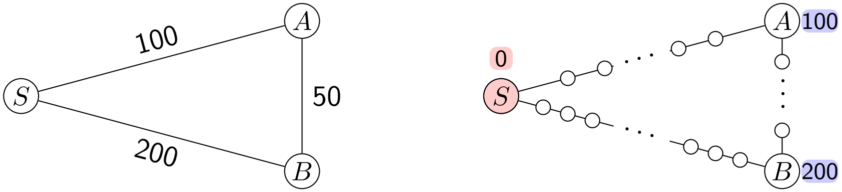


- Suppose that every second we discover one new layer.
- For the first 99 seconds, we are waiting for the dummy nodes we don't care about.
- We could set alarm clocks and go take a nap.

Don't watch the pot, just take a nap

Problem with this approach?

- What about non-integer weights? Negative weights?
- The graph size can increase exponentially.

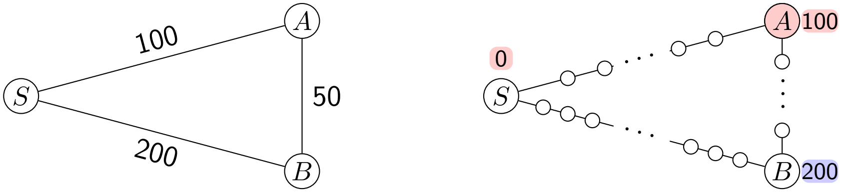


- Suppose that every second we discover one new layer.
- For the first 99 seconds, we are waiting for the dummy nodes we don't care about.
- We could set alarm clocks and go take a nap.

Don't watch the pot, just take a nap

Problem with this approach?

- What about non-integer weights? Negative weights?
- The graph size can increase exponentially.

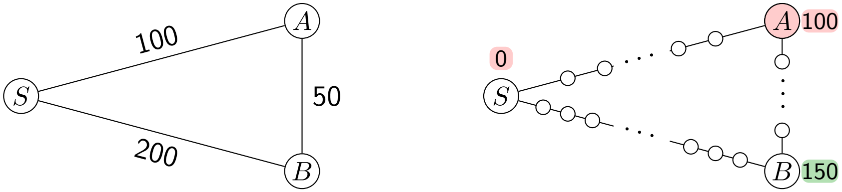


- Suppose that every second we discover one new layer.
- For the first 99 seconds, we are waiting for the dummy nodes we don't care about.
- We could set alarm clocks and go take a nap.

Don't watch the pot, just take a nap

Problem with this approach?

- What about non-integer weights? Negative weights?
- The graph size can increase exponentially.

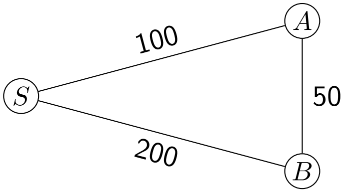


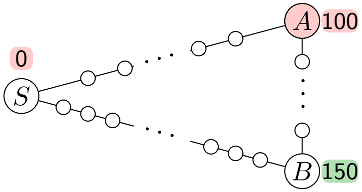
- Suppose that every second we discover one new layer.
- For the first 99 seconds, we are waiting for the dummy nodes we don't care about.
- We could set alarm clocks and go take a nap.

Don't watch the pot, just take a nap

Problem with this approach?

- What about non-integer weights? Negative weights?
- The graph size can increase exponentially.

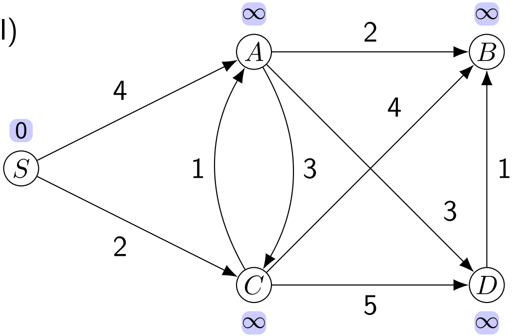




- Suppose that every second we discover one new layer.
- For the first 99 seconds, we are waiting for the dummy nodes we don't care about.
- We could set alarm clocks and go take a nap.
- These are the estimated upper bounds for arrival time.

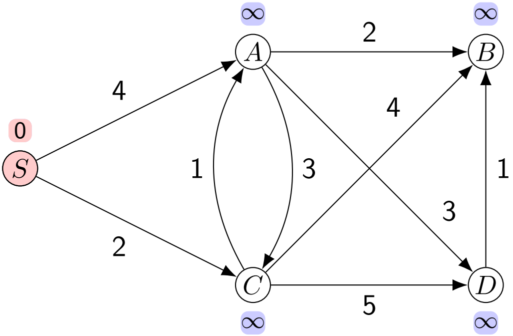
Another example

Time: (initial)



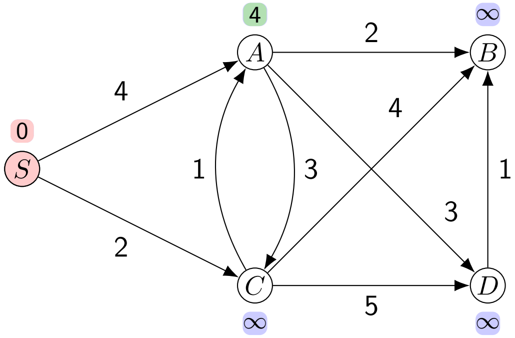
Another example

Time: 0



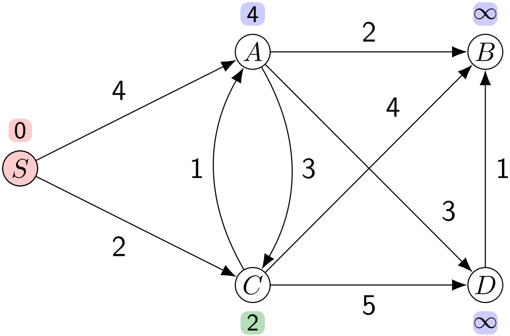
Another example

Time: 0



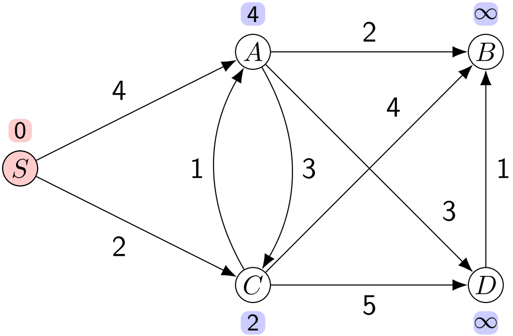
Another example

Time: 0



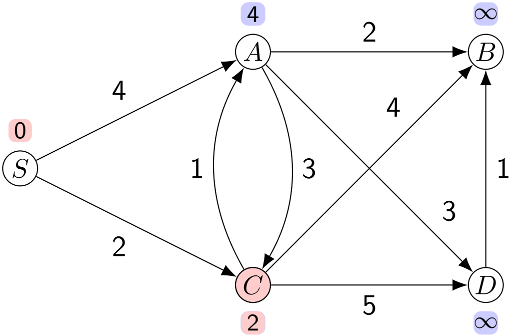
Another example

Time: 0



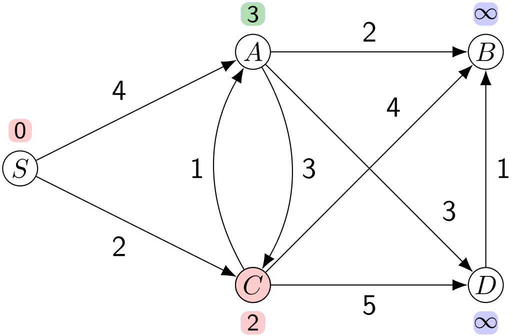
Another example

Time: 2



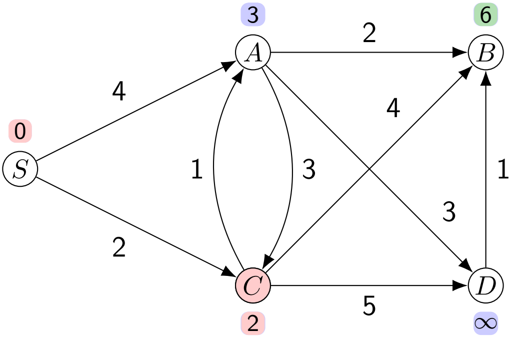
Another example

Time: 2



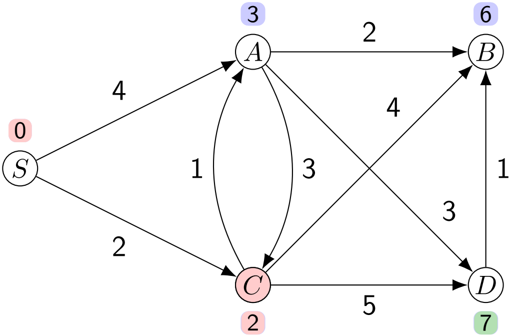
Another example

Time: 2



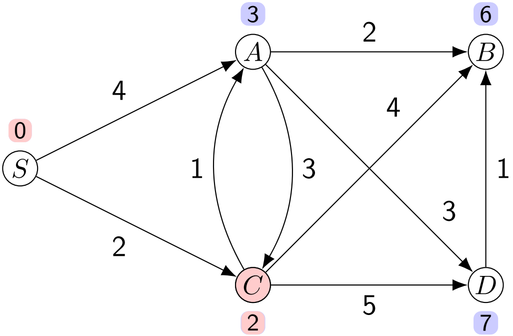
Another example

Time: 2



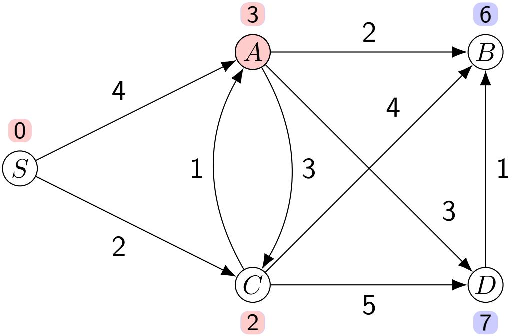
Another example

Time: 2



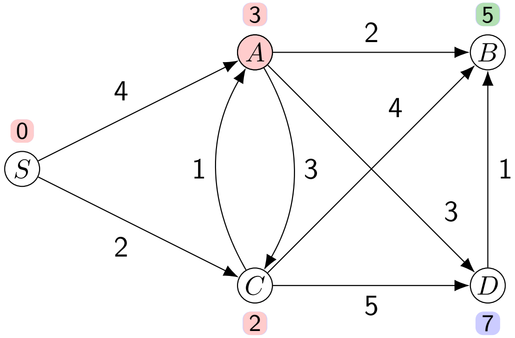
Another example

Time: 3



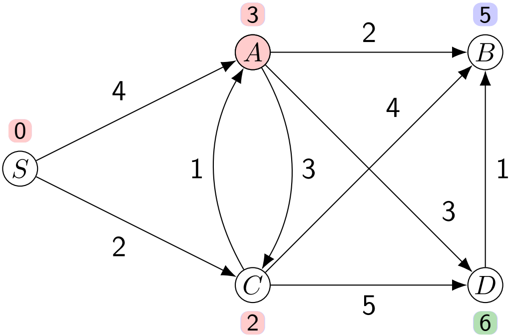
Another example

Time: 3



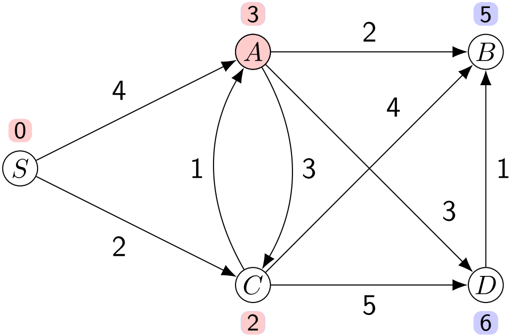
Another example

Time: 3



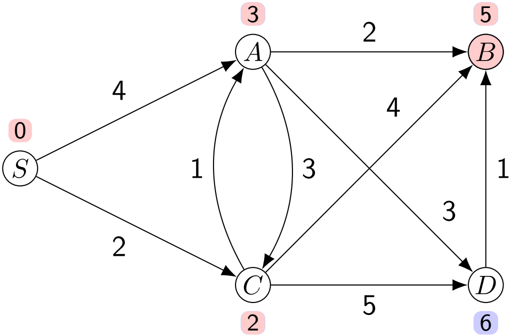
Another example

Time: 3



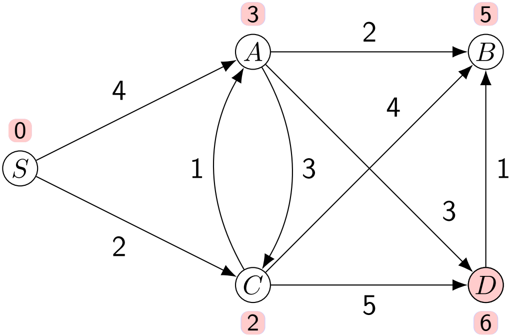
Another example

Time: 5



Another example

Time: 6



Dijkstra's algorithm

Input: A graph $G = (V, E, \ell)$ where $\ell : E \to \mathbb{N}$ maps edges to weights, a starting vertex s

Output: Shortest paths from s to any other vertex

Dijkstra(G, s)

```
// dist has length |V|, for the alarm clocks
foreach v ∈ V do
    dist[v] = ∞
dist[s] = 0

repeat |V| times do
    Find the vertex v with the smallest time among those
      whose alarms have not rung yet

    foreach (v, w) ∈ E do
        if dist[w] > dist[v] + ℓ(v, w) then
            dist[w] = dist[v] + ℓ(v, w)
```

Dijkstra's algorithm

Input: A graph $G = (V, E, \ell)$ where $\ell : E \to \mathbb{N}$ maps edges to weights, a starting vertex s

Output: Shortest paths from s to any other vertex

Dijkstra(G, s)

```
// dist has length |V|, for the alarm clocks
foreach v ∈ V do
    dist[v] = ∞
dist[s] = 0

repeat |V| times do
    Find the vertex v with the smallest time among those
     whose alarms have not rung yet

    foreach (v, w) ∈ E do
        if dist[w] > dist[v] + ℓ(v, w) then
            dist[w] = dist[v] + ℓ(v, w)
```

$O(|V|^2)$ with naive implementation, we'll see how to improve later.

Correctness of Dijkstra's algorithm

Lemma If at time T the alarm clock for vertex v rings for the first time, the shortest path from s to v has weight T.

Correctness of Dijkstra's algorithm

Lemma If at time T the alarm clock for vertex v rings for the first time, the shortest path from s to v has weight T.

Proof. Let R be the set of vertices whose alarm have already gone off. We aim to prove by induction that the claim holds for each vertex $v \in R$.

Correctness of Dijkstra's algorithm

Lemma If at time T the alarm clock for vertex v rings for the first time, the shortest path from s to v has weight T.

Proof. Let R be the set of vertices whose alarm have already gone off. We aim to prove by induction that the claim holds for each vertex $v \in R$.

Base case: At time $T = 0$, $R = \{s\}$ and $dist[s] = 0$.

Correctness of Dijkstra's algorithm

Lemma If at time T the alarm clock for vertex v rings for the first time, the shortest path from s to v has weight T.

Proof. Let R be the set of vertices whose alarm have already gone off. We aim to prove by induction that the claim holds for each vertex $v \in R$.

Base case: At time $T = 0$, $R = \{s\}$ and $dist[s] = 0$.

Suppose the alarm of v goes off first at time T. This means:

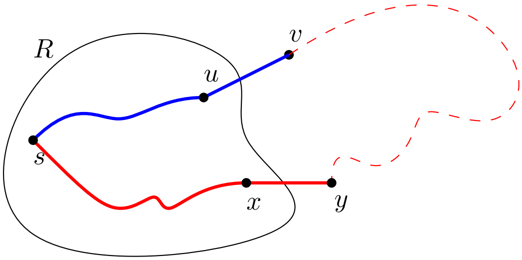
- $T = dist[v] = \min_{w \notin R} dist[w]$
- v will now be added to R
- $dist[v] = dist[u] + \ell(u, v)$ for some $u \in R$.

Correctness of Dijkstra's algorithm

Suppose the alarm of v goes off first at time T. This means:

- $T = dist[v] = \min_{w \notin R} dist[w]$
- v will now be added to R
- $dist[v] = dist[u] + \ell(u, v)$ for some $u \in R$.

Assume, for contradiction, that there is a shorter path from s to v:

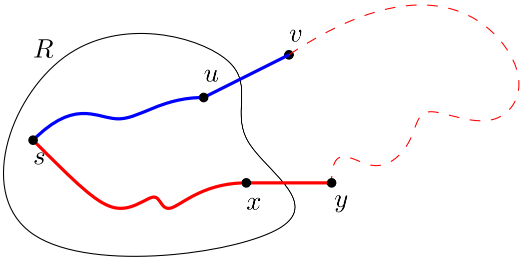


Correctness of Dijkstra's algorithm

Suppose the alarm of v goes off first at time T. This means:

- $T = dist[v] = \min_{w \notin R} dist[w]$
- v will now be added to R
- $dist[v] = dist[u] + \ell(u, v)$ for some $u \in R$.

Assume, for contradiction, that there is a shorter path from s to v:



$len(\text{red path}) \geq dist[x] + \ell(x, y) \geq dist[y] \geq dist[v] = len(\text{blue path})$.

Speed up our implementation of Dijkstra

We need to find, in each iteration of the outer loop, the vertex with the minimum alarm time.

Speed up our implementation of Dijkstra

We need to find, in each iteration of the outer loop, the vertex with the minimum alarm time.

This can be done using a min-heap:

- Store key-value pairs (alarm time, vertex) in the heap.

Speed up our implementation of Dijkstra

We need to find, in each iteration of the outer loop, the vertex with the minimum alarm time.

This can be done using a min-heap:

- Store key-value pairs (alarm time, vertex) in the heap.
- Call DecreaseKey if an alarm needs update.

Speed up our implementation of Dijkstra

We need to find, in each iteration of the outer loop, the vertex with the minimum alarm time.

This can be done using a min-heap:

- Store key-value pairs (alarm time, vertex) in the heap.
- Call DecreaseKey if an alarm needs update.
- Each vertex also needs to know its index in the heap.