Data Structures and Algorithms

Live Class 9

Heikki Peura

h.peura@imperial.ac.uk

Announcements

- **This week's deadlines moved to Thursday**
- Last office hours on Thursday (1pm-3pm)

Survey feedback

I have shared the results with the programme team.

Common themes:

1. There's not quite enough time to absorb the module content.
2. Some people would like to go deeper — others feel like the workload is very high.
3. Maths and Stats is more work than the other two classes.
4. We could do better in coordinating deadlines and modes of communication between modules.
5. We could do better in encouraging more interactions and discussion.

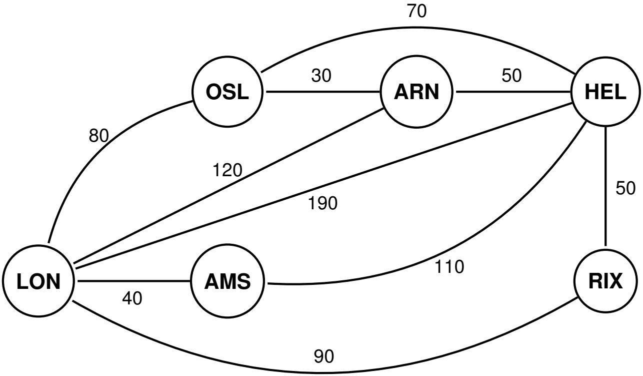
Today

- Weighted graphs
- Dijkstra's shortest-path algorithm

Shortest paths

Suppose you're travelling around Europe

- Know flights and their prices
- Cheapest price?



What is the shortest path from LON to HEL using BFS?

Why not just use BFS?

BFS already allows us to find the shortest paths?

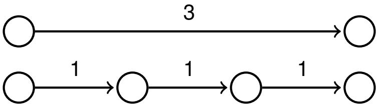
- We assumed that all edges have equal lengths

Why not just use BFS?

BFS already allows us to find the shortest paths?

- We assumed that all edges have equal lengths

Write each edge as a series of length one edges?

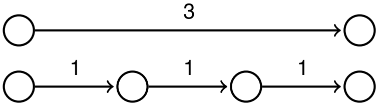


Why not just use BFS?

BFS already allows us to find the shortest paths?

- We assumed that all edges have equal lengths

Write each edge as a series of length one edges?



- Graph becomes **impractical**

Dijkstra's algorithm

Dijkstra's algorithm

Dijkstra (sketch): shortest paths from node s

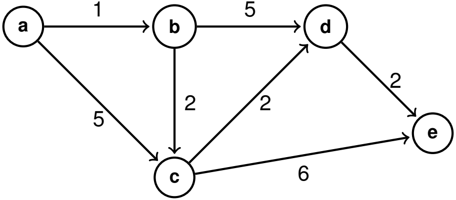
1. Initially mark s "explored"
2. Find "costs" of all edges from explored nodes to unexplored nodes
3. Pick the "cheapest" edge, mark its end node explored
4. Repeat until every node in the graph has been explored
5. Check the final paths

Dijkstra example

1. Initially mark *s* “explored”
2. Find **costs** of all **edges from explored** nodes **to unexplored** nodes **as cost of explored node plus edge cost**
3. Pick the **cheapest edge**, mark its end node explored
4. Repeat until every node in the graph has been explored

Dijkstra example

1. Initially mark *s* “explored”
2. Find **costs** of all **edges from explored** nodes **to unexplored** nodes **as cost of explored node plus edge cost**
3. Pick the **cheapest edge**, mark its end node explored
4. Repeat until every node in the graph has been explored



Edges from explored to unexplored

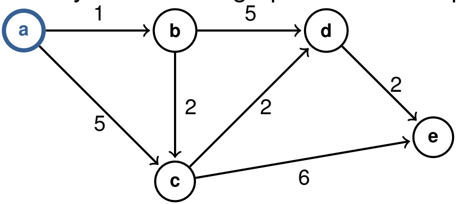
Nodes explored:

Costs of explored nodes

a	b	c	d	e
∞	∞	∞	∞	∞

Dijkstra example

1. Initially mark *s* “explored”
2. Find **costs** of all **edges from explored** nodes **to unexplored** nodes **as cost of explored node plus edge cost**
3. Pick the **cheapest edge**, mark its end node explored
4. Repeat until every node in the graph has been explored



Edges from explored to unexplored

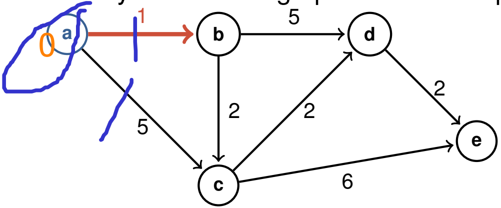
Nodes explored: a

Costs of explored nodes

a	b	c	d	e
0	∞	∞	∞	∞

Dijkstra example

1. Initially mark *s* “explored”
2. Find **costs** of all **edges from explored** nodes **to unexplored** nodes **as cost of explored node plus edge cost**
3. Pick the **cheapest edge**, mark its end node explored
4. Repeat until every node in the graph has been explored



Edges from explored to unexplored

ab
1

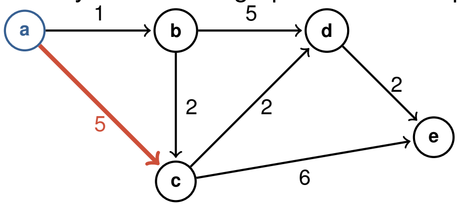
Nodes explored: a

Costs of explored nodes

a	b	c	d	e
0	∞	∞	∞	∞

Dijkstra example

1. Initially mark *s* “explored”
2. Find **costs** of all **edges from explored** nodes **to unexplored** nodes **as cost of explored node plus edge cost**
3. Pick the **cheapest edge**, mark its end node explored
4. Repeat until every node in the graph has been explored



Edges from explored to unexplored

ab	ac
1	5

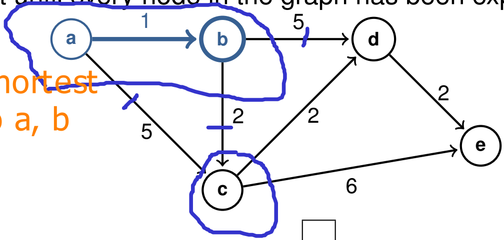
Nodes explored: a

Costs of explored nodes

a	b	c	d	e
0	∞	∞	∞	∞

Dijkstra example

1. Initially mark *s* "explored"
2. Find **costs** of all **edges from explored** nodes **to unexplored** nodes **as cost of explored node plus edge cost**
3. Pick the **cheapest edge**, mark its end node explored
4. Repeat until every node in the graph has been explored



We know shortest distances to a, b

cross to c

Edges from explored to unexplored

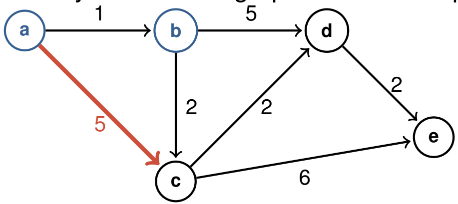
Nodes explored: a, b

Costs of explored nodes

a	b	c	d	e
0	1	∞	∞	∞

Dijkstra example

1. Initially mark *s* “explored”
2. Find **costs** of all **edges from explored** nodes **to unexplored** nodes **as cost of explored node plus edge cost**
3. Pick the **cheapest edge**, mark its end node explored
4. Repeat until every node in the graph has been explored



Edges from explored to unexplored

ac
5

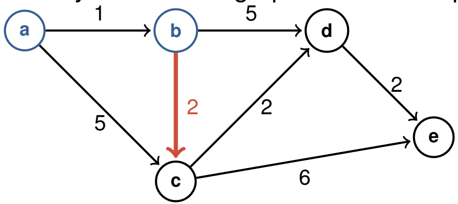
Nodes explored: a, b

Costs of explored nodes

a	b	c	d	e
0	1	∞	∞	∞

Dijkstra example

1. Initially mark *s* “explored”
2. Find **costs** of all **edges from explored** nodes **to unexplored** nodes **as cost of explored node plus edge cost**
3. Pick the **cheapest edge**, mark its end node explored
4. Repeat until every node in the graph has been explored



Edges from explored to unexplored

ac	bc
5	3

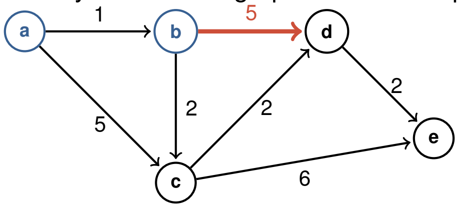
Nodes explored: a, b

Costs of explored nodes

a	b	c	d	e
0	1	∞	∞	∞

Dijkstra example

1. Initially mark *s* “explored”
2. Find **costs** of all **edges from explored** nodes **to unexplored** nodes **as cost of explored node plus edge cost**
3. Pick the **cheapest edge**, mark its end node explored
4. Repeat until every node in the graph has been explored



Edges from explored to unexplored

ac	bc	bd
5	3	6

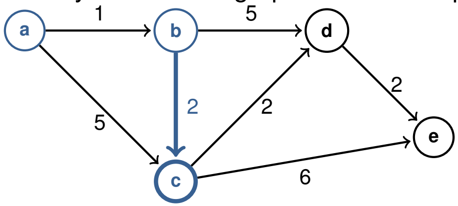
Nodes explored: a, b

Costs of explored nodes

a	b	c	d	e
0	1	∞	∞	∞

Dijkstra example

1. Initially mark *s* “explored”
2. Find **costs** of all **edges from explored** nodes **to unexplored** nodes **as cost of explored node plus edge cost**
3. Pick the **cheapest edge**, mark its end node explored
4. Repeat until every node in the graph has been explored



Edges from explored to unexplored

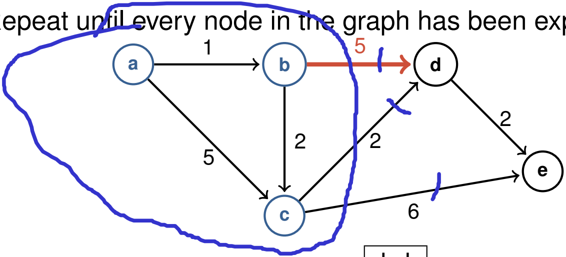
Nodes explored: a, b, c

Costs of explored nodes

a	b	c	d	e
0	1	3	∞	∞

Dijkstra example

1. Initially mark *s* “explored”
2. Find **costs** of all **edges from explored** nodes **to unexplored** nodes **as cost of explored node plus edge cost**
3. Pick the **cheapest edge**, mark its end node explored
4. Repeat until every node in the graph has been explored



Edges from explored to unexplored

bd
6

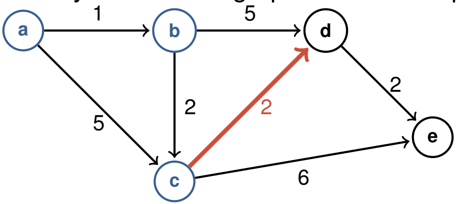
Nodes explored: a, b, c

Costs of explored nodes

a	b	c	d	e
0	1	3	∞	∞

Dijkstra example

1. Initially mark *s* “explored”
2. Find **costs** of all **edges from explored** nodes **to unexplored** nodes **as cost of explored node plus edge cost**
3. Pick the **cheapest edge**, mark its end node explored
4. Repeat until every node in the graph has been explored



Edges from explored to unexplored

bd	cd
6	5

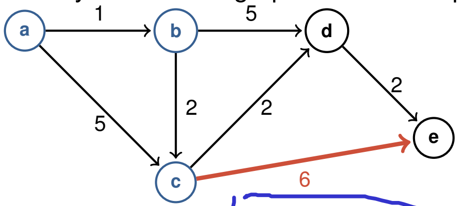
Nodes explored: a, b, c

Costs of explored nodes

a	b	c	d	e
0	1	3	∞	∞

Dijkstra example

1. Initially mark *s* "explored"
2. Find **costs** of all **edges from explored** nodes **to unexplored** nodes **as cost of explored node plus edge cost**
3. Pick the **cheapest edge**, mark its end node explored
4. Repeat until every node in the graph has been explored



Edges from explored to unexplored

bd	cd	ce
6	5	9

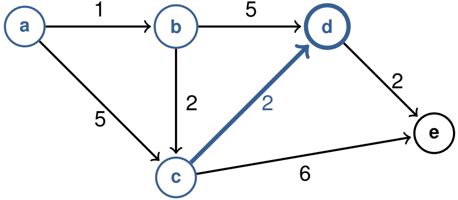
Nodes explored: a, b, c

Costs of explored nodes

a	b	c	d	e
0	1	3	∞	∞

Dijkstra example

1. Initially mark *s* “explored”
2. Find **costs** of all **edges from explored** nodes **to unexplored** nodes **as cost of explored node plus edge cost**
3. Pick the **cheapest edge**, mark its end node explored
4. Repeat until every node in the graph has been explored



Edges from explored to unexplored

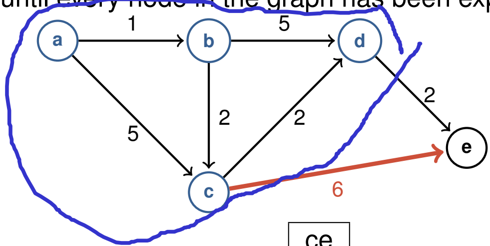
Nodes explored: a, b, c, d

Costs of explored nodes

a	b	c	d	e
0	1	3	5	∞

Dijkstra example

1. Initially mark *s* “explored”
2. Find **costs** of all **edges from explored** nodes **to unexplored** nodes **as cost of explored node plus edge cost**
3. Pick the **cheapest edge**, mark its end node explored
4. Repeat until every node in the graph has been explored



Edges from explored to unexplored

ce
9

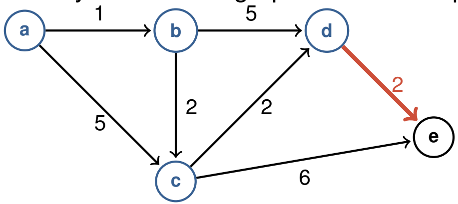
Nodes explored: a, b, c, d

Costs of explored nodes

a	b	c	d	e
0	1	3	5	∞

Dijkstra example

1. Initially mark *s* "explored"
2. Find **costs** of all **edges from explored** nodes **to unexplored** nodes **as cost of explored node plus edge cost**
3. Pick the **cheapest edge**, mark its end node explored
4. Repeat until every node in the graph has been explored



Edges from explored to unexplored

ce	de
9	7

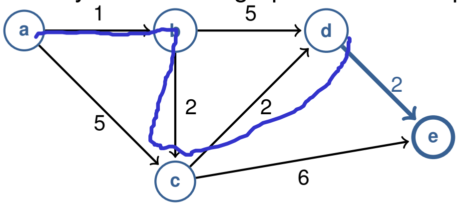
Nodes explored: a, b, c, d

Costs of explored nodes

a	b	c	d	e
0	1	3	5	∞

Dijkstra example

1. Initially mark *s* “explored”
2. Find **costs** of all **edges from explored** nodes **to unexplored** nodes **as cost of explored node plus edge cost**
3. Pick the **cheapest edge**, mark its end node explored
4. Repeat until every node in the graph has been explored



Edges from explored to unexplored

Nodes explored: a, b, c, d, e

Costs of explored nodes

a	b	c	d	e
0	1	3	5	7

How it works

- We start with no information: only zero cost for start node

How it works

- We start with no information: only zero cost for start node
- First iteration: the lowest-cost edge from explored (start node) to unexplored (any other node) must be the shortest distance to that node

How it works

- We start with no information: only zero cost for start node
- First iteration: the lowest-cost edge from explored (start node) to unexplored (any other node) must be the shortest distance to that node
  - Any other path would need to take a costlier route

How it works

- We start with no information: only zero cost for start node
- First iteration: the lowest-cost edge from explored (start node) to unexplored (any other node) must be the shortest distance to that node
  - Any other path would need to take a costlier route
- We then know the shortest distance to the two explored nodes

How it works

- We start with no information: only zero cost for start node
- First iteration: the lowest-cost edge from explored (start node) to unexplored (any other node) must be the shortest distance to that node
  - Any other path would need to take a costlier route
- We then know the shortest distance to the two explored nodes
- Second iteration: the lowest-cost edge from explored to unexplored must be shortest distance to the that node...

How it works

- We start with no information: only zero cost for start node
- First iteration: the lowest-cost edge from explored (start node) to unexplored (any other node) must be the shortest distance to that node
  - Any other path would need to take a costlier route
- We then know the shortest distance to the two explored nodes
- Second iteration: the lowest-cost edge from explored to unexplored must be shortest distance to the that node...
  - Any other path would need to take a costlier route

How it works

- We start with no information: only zero cost for start node
- First iteration: the lowest-cost edge from explored (start node) to unexplored (any other node) must be the shortest distance to that node
  - Any other path would need to take a costlier route
- We then know the shortest distance to the two explored nodes
- Second iteration: the lowest-cost edge from explored to unexplored must be shortest distance to the that node...
  - Any other path would need to take a costlier route
- Whenever we add a “cheapest” node to “explored” nodes, we have the shortest distance to that node

Thinking about data structures

What data structure would you use for:

- The nodes we have explored?
- The distances to explored nodes?

More formally

Input: (directed, connected) graph $G = (V, E)$

- V: set of n vertices, E: set of m edges
- Each edge e has length (cost) $c_e \geq 0$ (important!)
- Start from vertex s

Output: for each vertex v in V:

- length of shortest $s - v$ path in G

Dijkstra's algorithm

For a graph $G = (V, E)$: V: n vertices, E: m edges, edge (v, w) has length c_{vw}

Initialize: starting vertex s

- Set of vertices gone through so far $X = \{s\}$ (not gone through $V - X$)
- Shortest distances to all vertices A: $A[s] = 0$

Dijkstra's algorithm

For a graph $G = (V, E)$: V: n vertices, E: m edges, edge (v, w) has length c_{vw}

Initialize: starting vertex s

- Set of vertices gone through so far $X = \{s\}$ (not gone through $V - X$)
- Shortest distances to all vertices A: $A[s] = 0$

Main loop: While $X \neq V$:

- Go through all edges (v, w) starting in X and ending in $V - X$
- Pick the edge that minimizes $A[v] + c_{vw}$, call it (v^*, w^*)
- Add vertex w^* to X and set $A[w^*] = A[v^*] + c_{v^* w^*}$

Dijkstra's running time?

Input: graph $G = (V, E)$ of m edges, n vertices

Loop: While $X \neq V$:

- Look at all edges (v, w) starting in X and ending in $V - X$
- Pick the one that minimizes $A[v] + c_{vw}$, call it (v^*, w^*)
- Add vertex w^* to X and set $A[w^*] = A[v^*] + c_{v^* w^*}$

Dijkstra's running time?

Input: graph $G = (V, E)$ of m edges, n vertices

Loop: While $X \neq V$:

- Look at all edges (v, w) starting in X and ending in $V - X$
- Pick the one that minimizes $A[v] + c_{vw}$, call it (v^*, w^*)
- Add vertex w^* to X and set $A[w^*] = A[v^*] + c_{v^* w^*}$

- **Iterations of while loop**: $n - 1$
- **Work per iteration**: $O(m)$
- **Total complexity** $O(mn)$

Dijkstra example, repeated

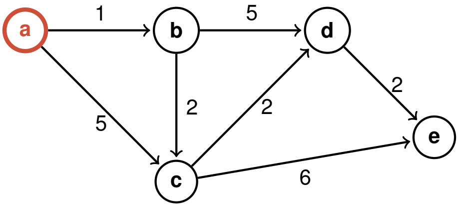
Main loop: While $X \neq V$:

- Go through all edges (v, w) starting in X and ending in $V - X$
- **Pick the edge that minimizes $A[v] + c_{vw}$, call it (v^*, w^*)**
- Add vertex w^* to X and set $A[w^*] = A[v^*] + c_{v^* w^*}$

Dijkstra example, repeated

Main loop: While $X \neq V$:

- Go through all edges (v, w) starting in X and ending in $V - X$
- **Pick the edge that minimizes $A[v] + c_{vw}$, call it (v^*, w^*)**
- Add vertex w^* to X and set $A[w^*] = A[v^*] + c_{v^* w^*}$

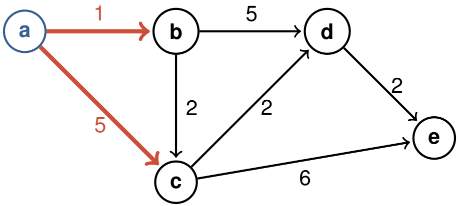


$A = [a : 0, b : \infty, c : \infty, d : \infty, e : \infty]$
$X = \{a\}$

Dijkstra example, repeated

Main loop: While $X \neq V$:

- Go through all edges (v, w) starting in X and ending in $V - X$
- **Pick the edge that minimizes $A[v] + c_{vw}$, call it (v^*, w^*)**
- Add vertex w^* to X and set $A[w^*] = A[v^*] + c_{v^* w^*}$

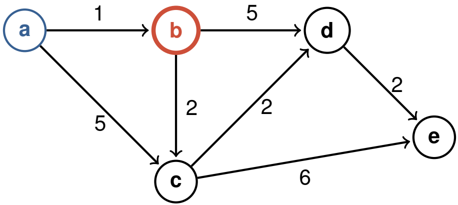


$A = [a : 0, b : \infty, c : \infty, d : \infty, e : \infty]$
$X = \{a\}$

Dijkstra example, repeated

Main loop: While $X \neq V$:

- Go through all edges (v, w) starting in X and ending in $V - X$
- **Pick the edge that minimizes $A[v] + c_{vw}$, call it (v^*, w^*)**
- Add vertex w^* to X and set $A[w^*] = A[v^*] + c_{v^* w^*}$

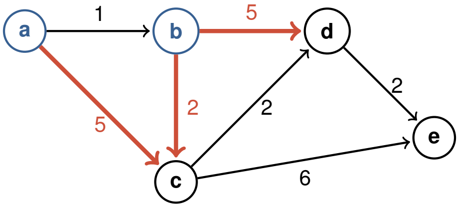


$A = [a : 0, b : 1, c : \infty, d : \infty, e : \infty]$
$X = \{a, b\}$

Dijkstra example, repeated

Main loop: While $X \neq V$:

- Go through all edges (v, w) starting in X and ending in $V - X$
- **Pick the edge that minimizes $A[v] + c_{vw}$, call it (v^*, w^*)**
- Add vertex w^* to X and set $A[w^*] = A[v^*] + c_{v^* w^*}$

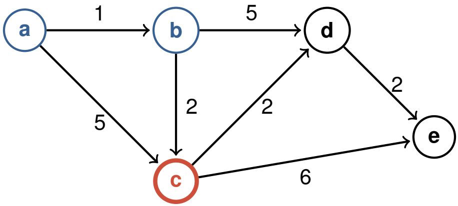


$A = [a : 0, b : 1, c : \infty, d : \infty, e : \infty]$
$X = \{a, b\}$

Dijkstra example, repeated

Main loop: While $X \neq V$:

- Go through all edges (v, w) starting in X and ending in $V - X$
- **Pick the edge that minimizes $A[v] + c_{vw}$, call it (v^*, w^*)**
- Add vertex w^* to X and set $A[w^*] = A[v^*] + c_{v^* w^*}$

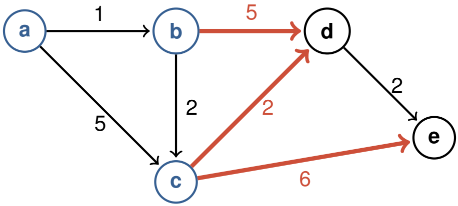


$A = [a : 0, b : 1, c : 3, d : \infty, e : \infty]$
$X = \{a, b, c\}$

Dijkstra example, repeated

Main loop: While $X \neq V$:

- Go through all edges (v, w) starting in X and ending in $V - X$
- **Pick the edge that minimizes $A[v] + c_{vw}$, call it (v^*, w^*)**
- Add vertex w^* to X and set $A[w^*] = A[v^*] + c_{v^* w^*}$

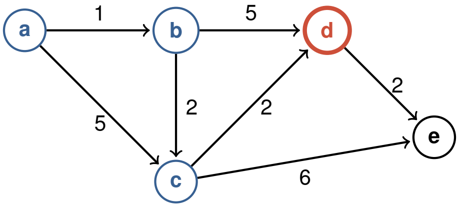


$A = [a : 0, b : 1, c : 3, d : \infty, e : \infty]$
$X = \{a, b, c\}$

Dijkstra example, repeated

Main loop: While $X \neq V$:

- Go through all edges (v, w) starting in X and ending in $V - X$
- **Pick the edge that minimizes $A[v] + c_{vw}$, call it (v^*, w^*)**
- Add vertex w^* to X and set $A[w^*] = A[v^*] + c_{v^*w^*}$

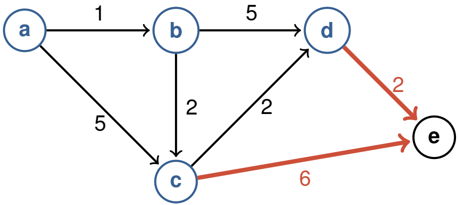


$A = [a : 0, b : 1, c : 3, d : 5, e : \infty]$
$X = \{a, b, c, d\}$

Dijkstra example, repeated

Main loop: While $X \neq V$:

- Go through all edges (v, w) starting in X and ending in $V - X$
- **Pick the edge that minimizes $A[v] + c_{vw}$, call it (v^*, w^*)**
- Add vertex w^* to X and set $A[w^*] = A[v^*] + c_{v^* w^*}$

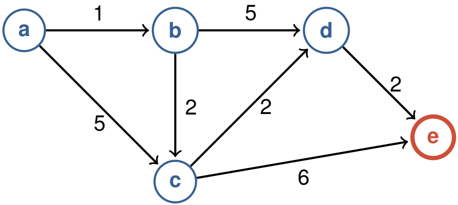


$A = [a : 0, b : 1, c : 3, d : 5, e : \infty]$
$X = \{a, b, c, d\}$

Dijkstra example, repeated

Main loop: While $X \neq V$:

- Go through all edges (v, w) starting in X and ending in $V - X$
- **Pick the edge that minimizes $A[v] + c_{vw}$, call it (v^*, w^*)**
- Add vertex w^* to X and set $A[w^*] = A[v^*] + c_{v^* w^*}$



$A = [a : 0, b : 1, c : 3, d : 5, e : 7]$
$X = \{a, b, c, d, e\}$

Dijkstra with mystery data structure D

Use D to store best Dijkstra scores so far for each unexplored node

Main loop: While $X \neq V$:

- Go through all edges (v, w) starting from last explored node v and ending in $V - X$: update Dijkstra scores in D
- **Pop the node with the lowest score from D, call it w^***
- Add w^* to X: it is now the last explored node

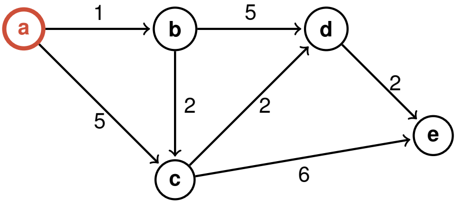
Dijkstra with mystery data structure D

Use D to store best Dijkstra scores so far for each unexplored node

Main loop: While $X \neq V$:

not all nodes in X

- Go through all edges (v, w) starting from last explored node v and ending in $V - X$: update Dijkstra scores in D
- **Pop the node with the lowest score from D, call it w^***
- Add w^* to X: it is now the last explored node



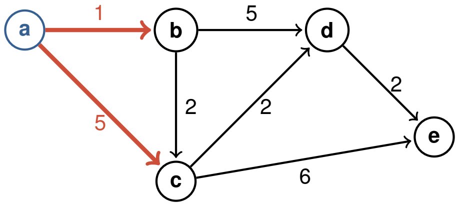
$A = [a : 0, b : \infty, c : \infty, d : \infty, e : \infty]$
$X = \{a\}$
$D = [(b, \infty), (c, \infty), (d, \infty), (e, \infty)]$

Dijkstra with mystery data structure D

Use D to store best Dijkstra scores so far for each unexplored node

Main loop: While $X \neq V$:

- Go through all edges (v, w) starting from last explored node v and ending in $V - X$: update Dijkstra scores in D
- **Pop the node with the lowest score from D, call it w^***
- Add w^* to X: it is now the last explored node



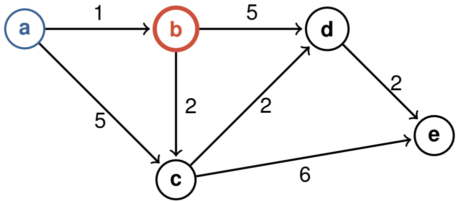
$A = [a : 0, b : \infty, c : \infty, d : \infty, e : \infty]$
$X = \{a\}$
$D = [(b, 1), (c, 5), (d, \infty), (e, \infty)]$

Dijkstra with mystery data structure D

Use D to store best Dijkstra scores so far for each unexplored node

Main loop: While $X \neq V$:

- Go through all edges (v, w) starting from last explored node v and ending in $V - X$: update Dijkstra scores in D
- **Pop the node with the lowest score from D, call it w^***
- Add w^* to X: it is now the last explored node



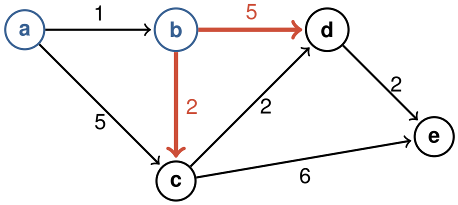
$A = [a : 0, b : 1, c : \infty, d : \infty, e : \infty]$
$X = \{a, b\}$
$D = [(c, 5), (d, \infty), (e, \infty)]$

Dijkstra with mystery data structure D

Use D to store best Dijkstra scores so far for each unexplored node

Main loop: While $X \neq V$:

- Go through all edges (v, w) starting from last explored node v and ending in $V - X$: update Dijkstra scores in D
- **Pop the node with the lowest score from D, call it w^***
- Add w^* to X: it is now the last explored node



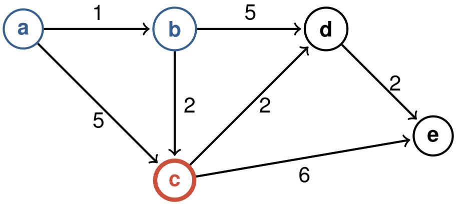
$A = [a : 0, b : 1, c : \infty, d : \infty, e : \infty]$
$X = \{a, b\}$
$D = [(c, 3), (d, 6), (e, \infty)]$

Dijkstra with mystery data structure D

Use D to store best Dijkstra scores so far for each unexplored node

Main loop: While $X \neq V$:

- Go through all edges (v, w) starting from last explored node v and ending in $V - X$: update Dijkstra scores in D
- **Pop the node with the lowest score from D, call it w^***
- Add w^* to X: it is now the last explored node



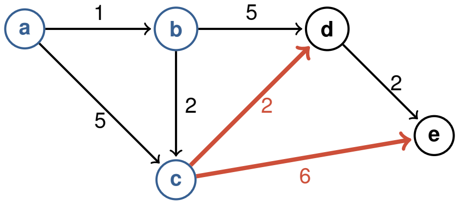
$A = [a : 0, b : 1, c : 3, d : \infty, e : \infty]$
$X = \{a, b, c\}$
$D = [(d, 6), (e, \infty)]$

Dijkstra with mystery data structure D

Use D to store best Dijkstra scores so far for each unexplored node

Main loop: While $X \neq V$:

- Go through all edges (v, w) starting from last explored node v and ending in $V - X$: update Dijkstra scores in D
- **Pop the node with the lowest score from D, call it w^***
- Add w^* to X: it is now the last explored node



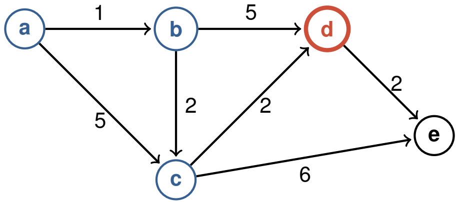
$A = [a : 0, b : 1, c : 3, d : \infty, e : \infty]$
$X = \{a, b, c\}$
$D = [(d, 5), (e, 9)]$

Dijkstra with mystery data structure D

Use D to store best Dijkstra scores so far for each unexplored node

Main loop: While $X \neq V$:

- Go through all edges (v, w) starting from last explored node v and ending in $V - X$: update Dijkstra scores in D
- **Pop the node with the lowest score from D, call it w^***
- Add w^* to X: it is now the last explored node



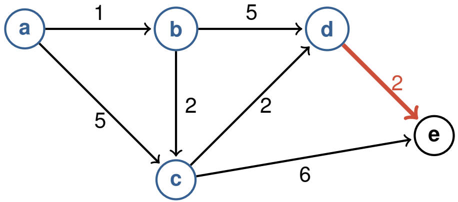
$A = [a : 0, b : 1, c : 3, d : 5, e : \infty]$
$X = \{a, b, c, d\}$
$D = [(e, 9)]$

Dijkstra with mystery data structure D

Use D to store best Dijkstra scores so far for each unexplored node

Main loop: While $X \neq V$:

- Go through all edges (v, w) starting from last explored node v and ending in $V - X$: update Dijkstra scores in D
- **Pop the node with the lowest score from D, call it w^***
- Add w^* to X: it is now the last explored node



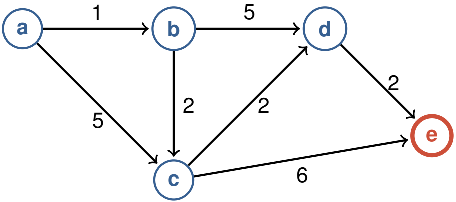
$A = [a : 0, b : 1, c : 3, d : 5, e : \infty]$
$X = \{a, b, c, d\}$
$D = [(e, 7)]$

Dijkstra with mystery data structure D

Use D to store best Dijkstra scores so far for each unexplored node

Main loop: While $X \neq V$:

- Go through all edges (v, w) starting from last explored node v and ending in $V - X$: update Dijkstra scores in D
- **Pop the node with the lowest score from D, call it w^***
- Add w^* to X: it is now the last explored node



$A = [a : 0, b : 1, c : 3, d : 5, e : 7]$
$X = \{a, b, c, d, e\}$
$D = []$

What do we want from D?

We store Dijkstra scores for unprocessed nodes in D

- In each iteration, we need to find the minimum score
- We also recalculate Dijkstra scores for edges starting from each node that we process (replace a Dikstra score or remove the old score and add a new score)

What do we want from D?

We store Dijkstra scores for unprocessed nodes in D

- In each iteration, we need to find the minimum score
- We also recalculate Dijkstra scores for edges starting from each node that we process (replace a Dikstra score or remove the old score and add a new score)

How many times do we do these operations?

- Find minimum in D: once per each iteration ie $n - 1$ times
- Recalculate score (replace ie remove/add score): once per each edge: m times
- Total $O(m + n)$ data structure operations on D

What do we want from D?

We store Dijkstra scores for unprocessed nodes in D

- In each iteration, we need to find the minimum score
- We also recalculate Dijkstra scores for edges starting from each node that we process (replace a Dikstra score or remove the old score and add a new score)

How many times do we do these operations?

- Find minimum in D: once per each iteration ie $n - 1$ times
- Recalculate score (replace ie remove/add score): once per each edge: m times
- Total $O(m + n)$ data structure operations on D

If D performed these operations in $O(\log n)$ time, ...

- ... then Dijkstra would run in $O((m + n) \log n)$ time (instead of $O(mn)$)

As it happens...

Heap:

- Extract minimum, insert, delete in $O(\log n)$ time
- Essentially a **priority queue** – not FIFO, but instead the node with the highest priority comes out first
- Our priority ordering is the Dijkstra score

As it happens...

Heap:

- Extract minimum, insert, delete in $O(\log n)$ time
- Essentially a **priority queue** – not FIFO, but instead the node with the highest priority comes out first
- Our priority ordering is the Dijkstra score

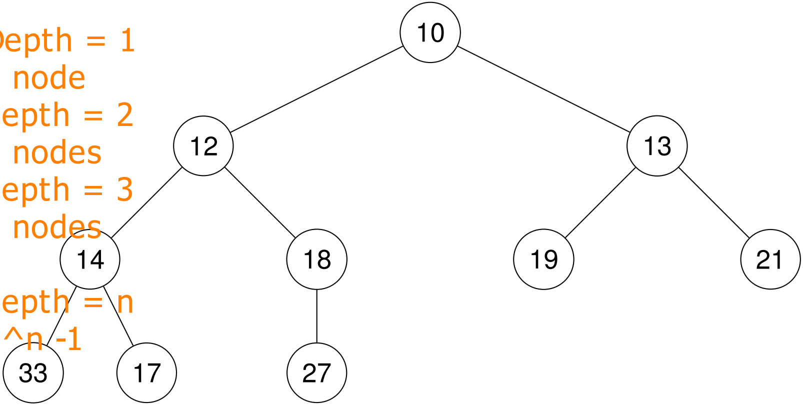
Implementation is a great exercise for you to try

- An optional problem asks you to use a built-in data structure to speed up Dijkstra
- Conceptually, a heap is a type of a **tree**, where the highest-priority item is on top, with links to lower-priority items in branches below
- **Difficulty**: keeping the order when adding items with different priorities

Heap

Depth = 1
1 node
depth = 2
3 nodes
depth = 3
7 nodes

depth = n
2^n -1

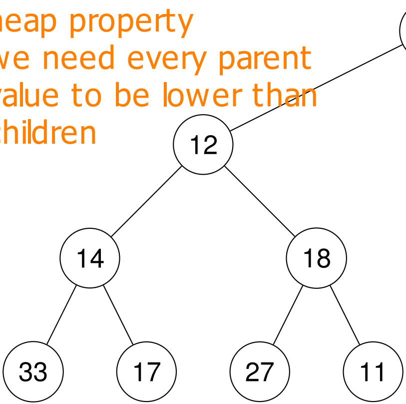



```
L = [10, 12, 13, 14, 18, 19, 21, 33, 17, 27]
```

depth is logarithmic in the number of items

Add item

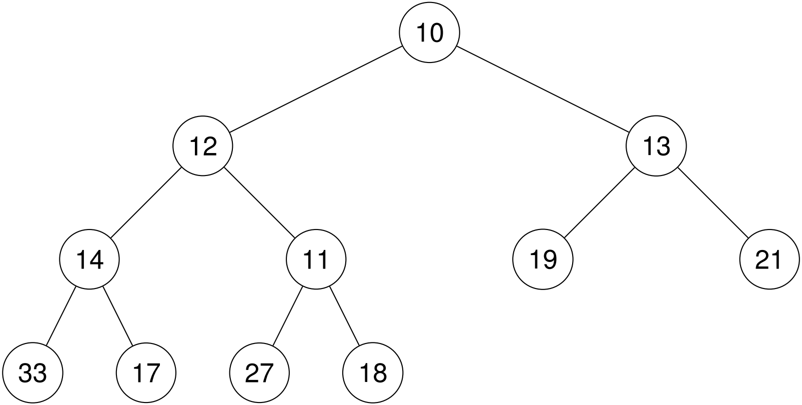
heap property
we need every parent
value to be lower than
children



no more than logarithmic
number of swaps

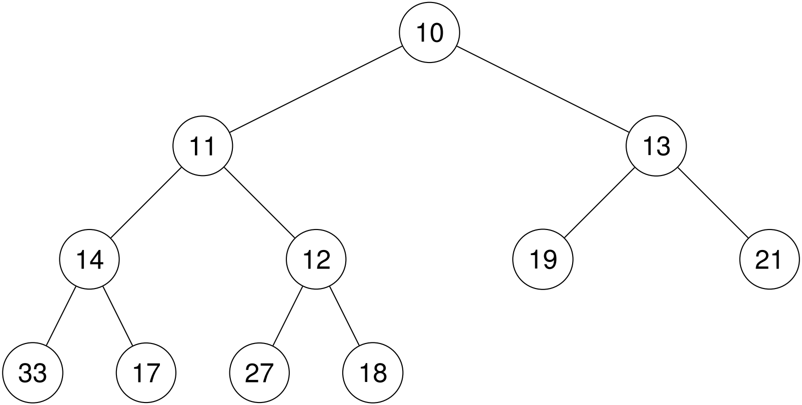
```
L = [10, 12, 13, 14, 18, 19, 21, 33, 17, 27, 11]
```

Add item



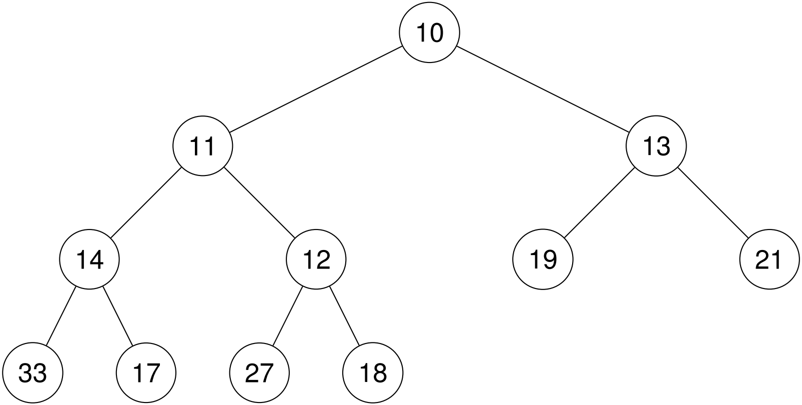

```
L = [10, 12, 13, 14, 11, 19, 21, 33, 17, 27, 18]
```

Add item



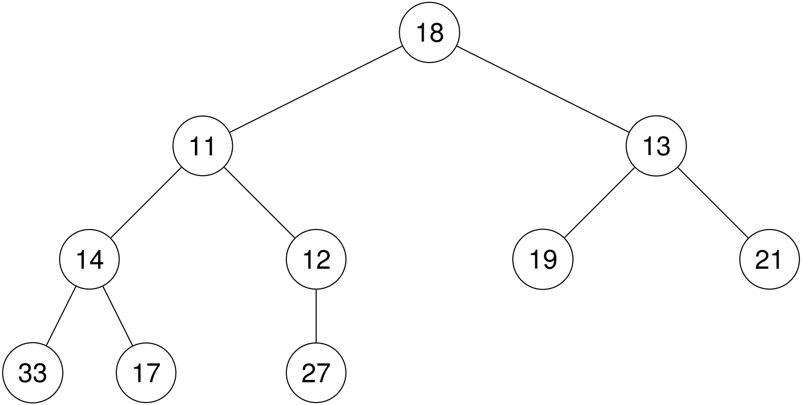

```
L = [10, 11, 13, 14, 12, 19, 21, 33, 17, 27, 18]
```

Extract minimum




```
L = [10, 11, 13, 14, 12, 19, 21, 33, 17, 27, 18]
```

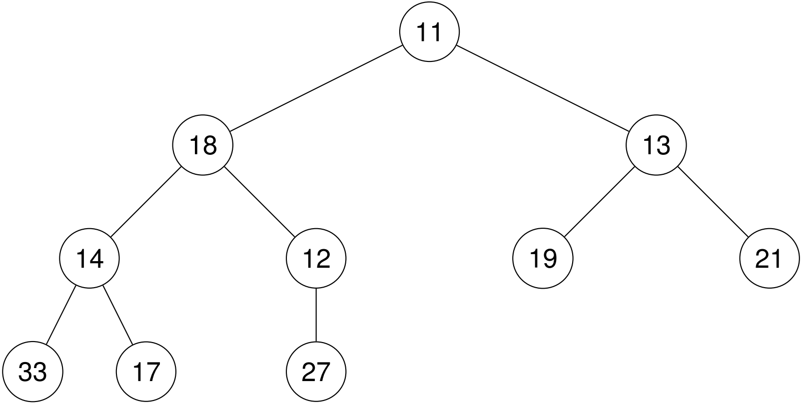
Extract minimum




```
L = [18, 11, 13, 14, 12, 19, 21, 33, 17, 27]
```

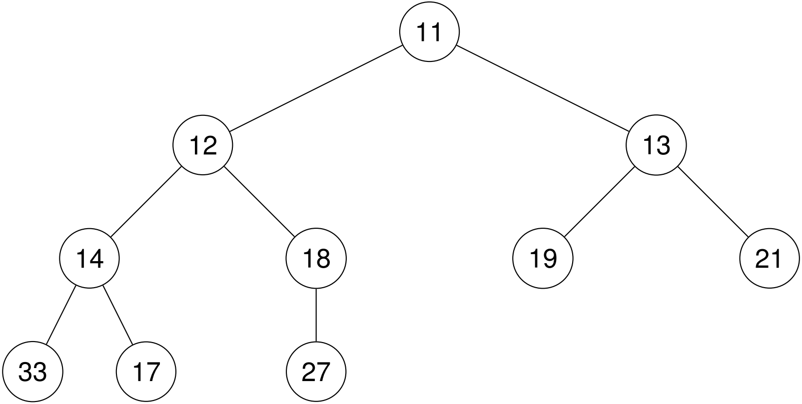
at most logarithmic number of swaps to reach bottom - each swap is constant time

Extract minimum




```
L = [11, 18, 13, 14, 12, 19, 21, 33, 17, 27]
```

Extract minimum



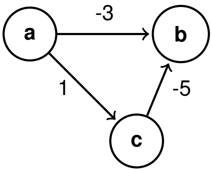
`L = [11, 12, 13, 14, 18, 19, 21, 33, 17, 27]`

Remember our assumption?

We assumed **non-negative edge lengths**

Remember our assumption?

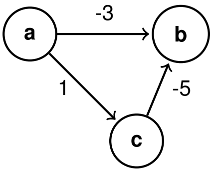
We assumed **non-negative edge lengths**



Here Dijkstra will fail...

Remember our assumption?

We assumed **non-negative edge lengths**



Here Dijkstra will fail...

With negative edge weights, **we need another algorithm**:

- **Bellman-Ford** — for you to explore

Review

Shortest paths

- Dijkstra’s algorithm
- Data structure selection matters!
- In Dijkstra’s case: heap

Review exercises

- Try Dijkstra in Python
- Recover shortest paths too...
- Shortest paths on the London Underground