Data Structures and Algorithms

Lecture 7: Weighted graphs

Heikki Peura

h.peura@imperial.ac.uk

Previously

Graphs:

- Representing a network as a graph
- Breadth-first search

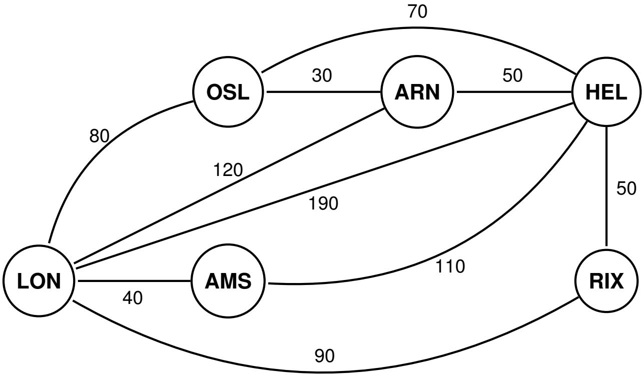
Plan for today:

- 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?

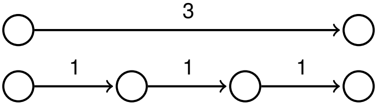
- Assuming that all edges have length 1...

Why not just use BFS?

BFS already allows us to find the shortest paths?

- Assuming that all edges have length 1...

Write each edge as a series of length one edges?

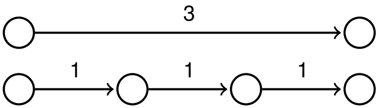


Why not just use BFS?

BFS already allows us to find the shortest paths?

- Assuming that all edges have length 1...

Write each edge as a series of length one edges?



- Graph becomes **impractical**...

Need a new algorithm: **Dijkstra's shortest-paths algorithm**

We need a new algorithm

Dijkstra's algorithm: shortest paths from node s

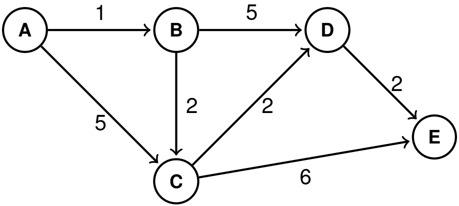
1. Initially mark s "explored"
2. Find "costs" of all edges from explored nodes to unexplored nodes (details to follow)
3. Pick the "cheapest" edge, mark its end node explored
4. Repeat until every node in the graph has been explored
5. Check the final paths (details to follow)

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

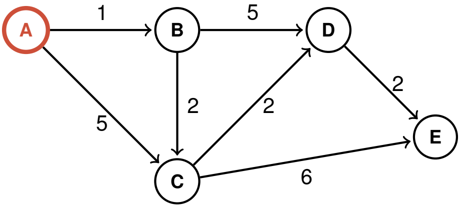
Nodes explored:

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

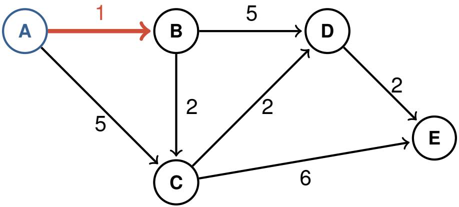
Nodes explored: A

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

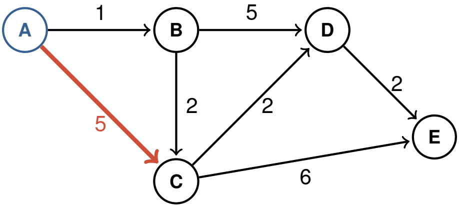
Nodes explored: A

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

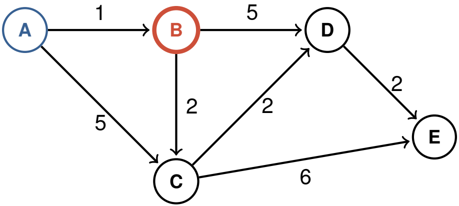
Nodes explored: A

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



Edges from explored to unexplored

AC
5

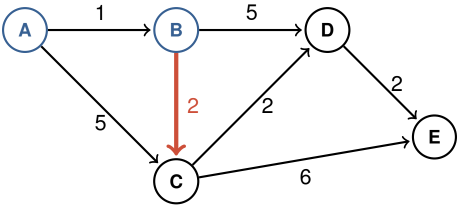
Nodes explored: A, B

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

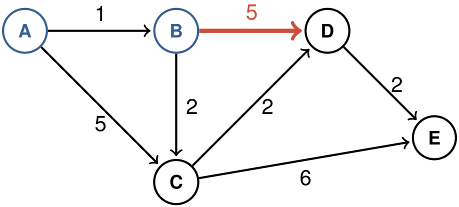
Nodes explored: A, B

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

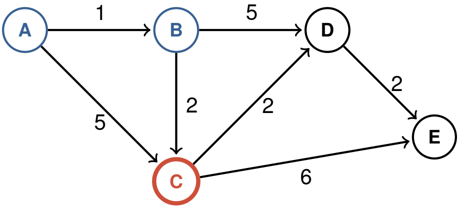
Nodes explored: A, B

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

BD
6

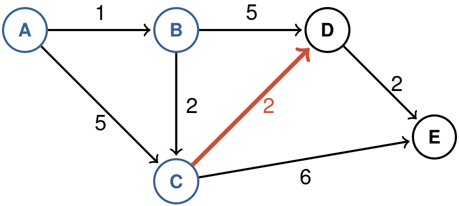
Nodes explored: A, B, C

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

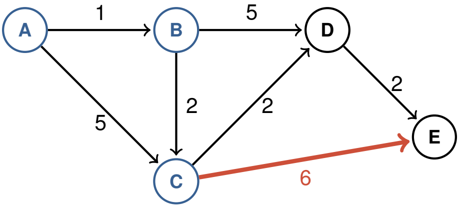
Nodes explored: A, B, C

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

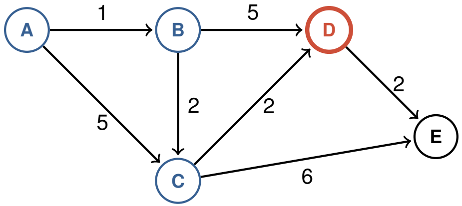
Nodes explored: A, B, C, D

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

CE
9

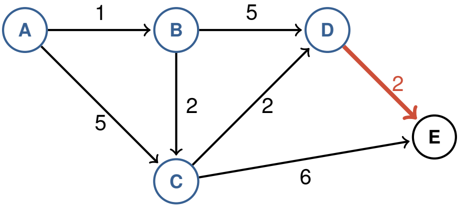
Nodes explored: A, B, C, D

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

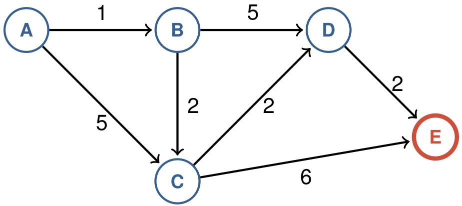
Nodes explored: A, B, C, D

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

CE	DE
9	7

Nodes explored: A, B, C, D, E

Costs of explored nodes

A	B	C	D	E
0	1	3	5	7

The gist

- We start with no information: “infinite” costs (zero for start node)

The gist

- We start with no information: "infinite" costs (zero 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

The gist

- We start with no information: “infinite” costs (zero 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

The gist

- We start with no information: “infinite” costs (zero 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

The gist

- We start with no information: “infinite” costs (zero 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...

The gist

- We start with no information: “infinite” costs (zero 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

The gist

- We start with no information: “infinite” costs (zero 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) graph $G = (V, E)$

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

More formally

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

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

Output: for each vertex v in V:

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

More formally

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

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

Output: for each vertex v in V:

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

Assume:

- That such paths exist (connected graph)
- $c_e \geq 0$ (important!!)

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

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 $V - X$)

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 $X \neq V$:

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 $V - X$

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^*)

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**: $n - 1$

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**: $O(m)$

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'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** $O(mn)$
- **Can we do better?**

How could we improve the algorithm?

Are we doing too much work?

Dijkstra example redux

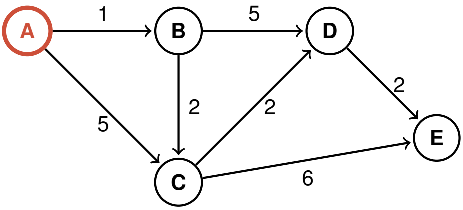
Main loop: While $X \neq V$:

- 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 redux

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 $A[w^*] = A[v^*] + c_{v^* w^*}$

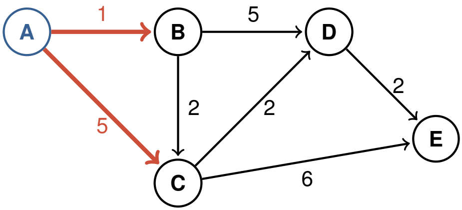


$A = [0, \infty, \infty, \infty, \infty]$
$X = \{A\}$
$D = [(B, \infty), (C, \infty), (D, \infty), (E, \infty)]$ (store Dijkstra scores for unexplored)

Dijkstra example redux

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 $A[w^*] = A[v^*] + c_{v^* w^*}$

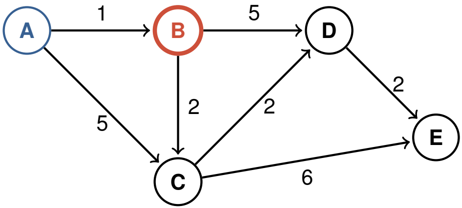


$A = [0, \infty, \infty, \infty, \infty]$
$X = \{A\}$
$D = [(B, 1), (C, 5), (D, \infty), (E, \infty)]$ (store Dijkstra scores for unexplored)

Dijkstra example redux

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 $A[w^*] = A[v^*] + c_{v^* w^*}$

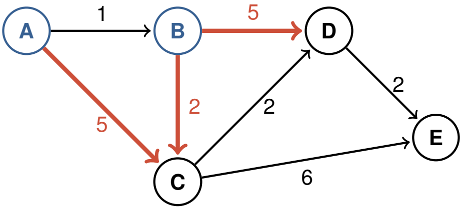


$A = [0, 1, \infty, \infty, \infty]$
$X = \{A, B\}$
$D = [(C, 5), (D, \infty), (E, \infty)]$ (store Dijkstra scores for unexplored)

Dijkstra example redux

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 $A[w^*] = A[v^*] + c_{v^* w^*}$

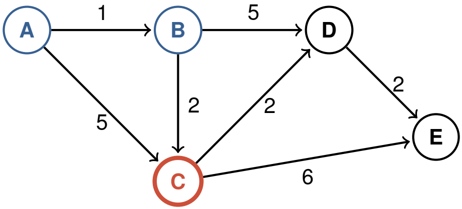


$A = [0, 1, \infty, \infty, \infty]$
$X = \{A, B\}$
$D = [(C, 3), (D, 6), (E, \infty)]$ (store Dijkstra scores for unexplored)

Dijkstra example redux

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 $A[w^*] = A[v^*] + c_{v^* w^*}$

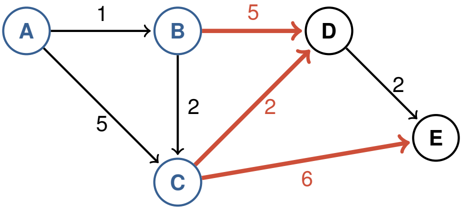


$A = [0, 1, 3, \infty, \infty]$
$X = \{A, B, C\}$
$D = [(D, 6), (E, \infty)]$ (store Dijkstra scores for unexplored)

Dijkstra example redux

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 $A[w^*] = A[v^*] + c_{v^* w^*}$

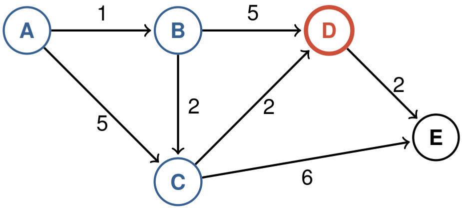


$A = [0, 1, 3, \infty, \infty]$
$X = \{A, B, C\}$
$D = [(D, 5), (E, 9)]$ (store Dijkstra scores for unexplored)

Dijkstra example redux

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 $A[w^*] = A[v^*] + c_{v^* w^*}$

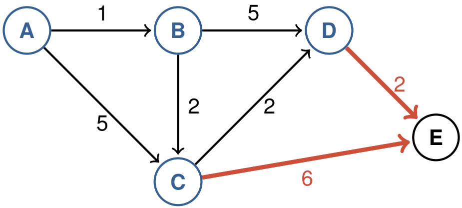


$A = [0, 1, 3, 5, \infty]$
$X = \{A, B, C, D\}$
$D = [(E, 9)]$ (store Dijkstra scores for unexplored)

Dijkstra example redux

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 $A[w^*] = A[v^*] + c_{v^* w^*}$

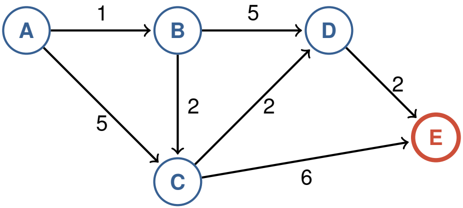


$A = [0, 1, 3, 5, \infty]$
$X = \{A, B, C, D\}$
$D = [(E, 7)]$ (store Dijkstra scores for unexplored)

Dijkstra example redux

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 $A[w^*] = A[v^*] + c_{v^* w^*}$



$A = [0, 1, 3, 5, 7]$
$X = \{A, B, C, D, E\}$
$D = []$ (store Dijkstra scores for unexplored)

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

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

If only there was a data structure that performed these operations in $O(\log n)$ time...

- Then Dijkstra would run in $O((m + n) \log n)$ time

As it happens...

Heap:

- Extract minimum, insert, delete in $O(\log n)$ time

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

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

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

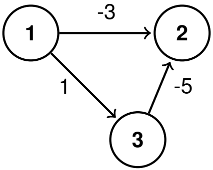
- An optional problem asks you to use a built-in data structure to speed up Dijkstra
- Conceptually, a heap is a type of a **tree**, where the highest-priority item is on top, and links to lower-priority items in branches below.
- **Difficulty**: keeping the order when adding items with different priorities

Remember our assumption?

We assumed **non-negative edge lengths**

Remember our assumption?

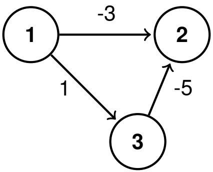
We assumed **non-negative edge lengths**



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

Workshop after the break

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

Workshop

Workshop zip file on the Hub

- HTML instructions
- At some point, you'll need the `.py`-file with skeleton code (open in Spyder)