Algorithms and Data Structures

Graphs – Part III – Prim & Dijkstra

Jacob Trier Frederiksen & Anders Kalhauge

Spring 2019

Program

- Group presentation of Assignment #3, Airport Queueing. Moderated by *'NSquared'* and *'NoGroup'*.
- Excerpt material from Chapters 4.3 and 4.4:
  - Minimum Spanning Trees; **Prim's Algorithm (Lazy)**
  - Shortest Paths; **Dijkstra's Algorithm**
- Pop-up quizzes.

Assignment #3, Airport Queueing

Assignment 3 (mandatory)
Airport Queueing

Jacob Trier Frederiksen (instructor)
& Anders Kalhauge (AKA, assisting)

Algorithms & Data Structures, Spring 2019

In groups:

Implement a priotitized queueing system for an airport. You can use any priority queue algorithm, but you must be able to argue that the time complexity is no worse than $\mathcal{O}(\text{Log}N)$ for enqueue and dequeue respectively.

You should implement the priority queue in a setup that simulates passengers arriving to an airport, and passengers passing security. So you must think about the frequency of passengers in an airport, and how many are rejected (or delayed) in security – and things like that, to simulate the queues populating. So; like; random, time-of-day, ...

Passenger priority is assumed to be derived from properties like passenger category and arrival time (you decide the priority...):

1. Monkey
2. Late to flight
3. Disabled
4. Business class
5. Family

A template for such a setup can be found here:

Airport Queue Template

You may use the template, but you are strongly encouraged to think it out for yourself first, then – perhaps – seek inspiration in the provided code template.

Create a priority queue *instead* of the `NotPrioritisingPassengerArrayQueue` used in the template. Experiment with other values for producer and consumer. Try to add more than one consumer. Upload the solution, which should be

- Your own implementation, in a `PriorityQueue` class.
- A text file describing:
  - your argumentation and choice for a solution
  - your results, demonstrating that you stay $\mathcal{O}(\text{Log}N)$ for de- and en-queueing.

NB: do not upload the rest of the template, it will just confuse the reviewing process, and make extra work.

Upload to the Peergrade website, no later than Tuesday March 12th, 08:30.

Please ask if you are in doubt about any of this.

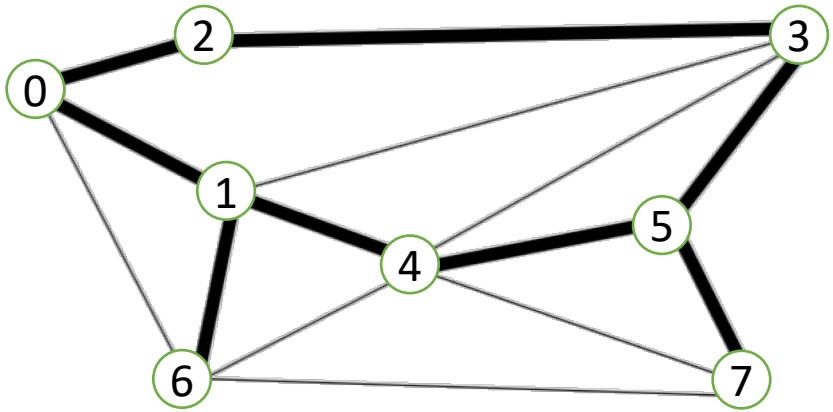
Minimum Spanning Trees

1. A ‘Greedy’ Heuristic Algorithm
2. Prim’s Algorithm (lazy version)

Spanning Tree Definition

- Un-directed weighted graph, G
  - Vertices, V
  - Edges, E
- ***Spanning Tree*** (ST) is then:
  - Connected
  - Acyclic
  - Includes all vertices, V

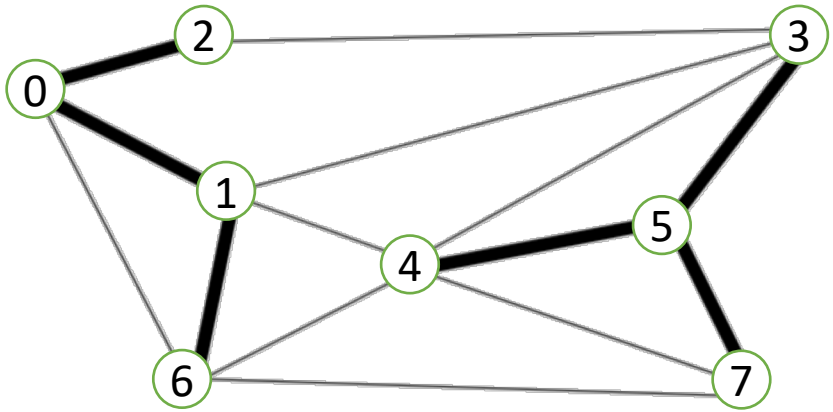
A Spanning Tree ?



Spanning Tree Definition

- Un-directed weighted graph, G
  - Vertices, V
  - Edges, E
- ***Spanning Tree*** (ST) is then:
  - Connected
  - Acyclic
  - Includes all vertices, V

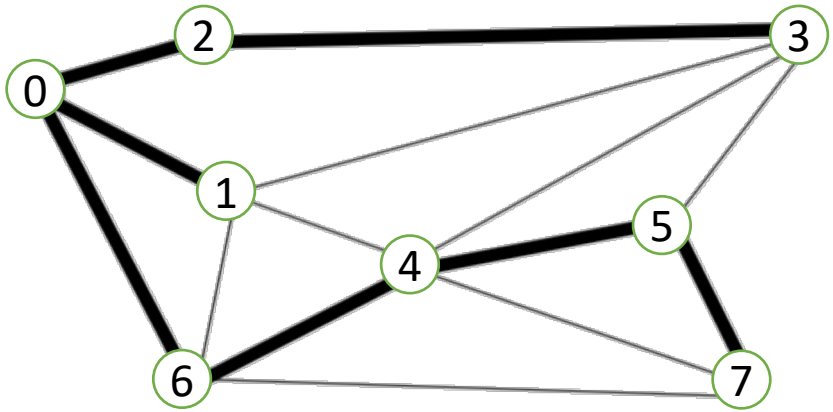
A Spanning Tree ?



Spanning Tree Definition

- Un-directed weighted graph, G
  - Vertices, V
  - Edges, E
- ***Spanning Tree*** (ST) is then:
  - Connected
  - Acyclic
  - Includes all vertices, V

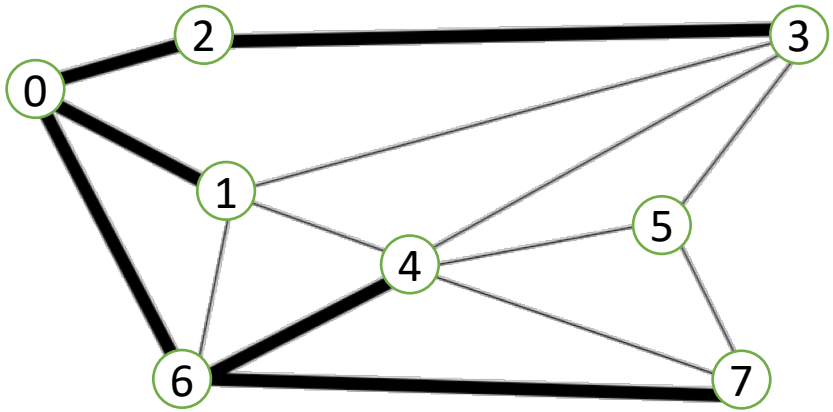
A Spanning Tree ?



Spanning Tree Definition

- Un-directed weighted graph, G
  - Vertices, V
  - Edges, E
- ***Spanning Tree*** (ST) is then:
  - Connected
  - Acyclic
  - Includes all vertices, V

A Spanning Tree ?



Pop Q

Consider a graph ‘G’, with ‘V’ vertices and ‘E’ edges.

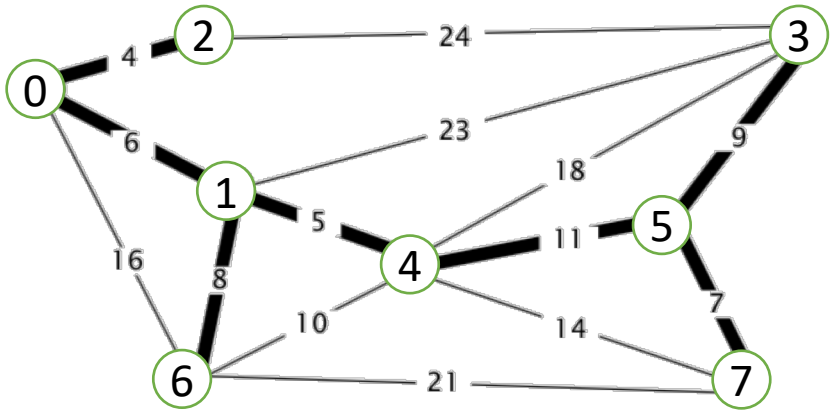
How many edges are in a spanning tree of ‘G’?

Minimum Spanning Tree Definition

- Un-directed weighted graph, G
  - Vertices, V
  - Edges, E
  - Weights, W

- ***Minimum Spanning Tree*** (MST) is then:
  - Connected
  - Acyclic
  - Includes all vertices, V

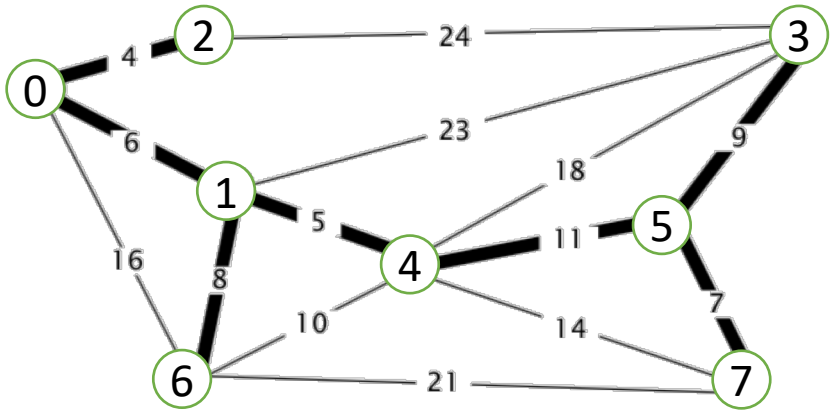
A Minimum Spanning Tree.



Minimum Spanning Tree
Simplifying assumptions

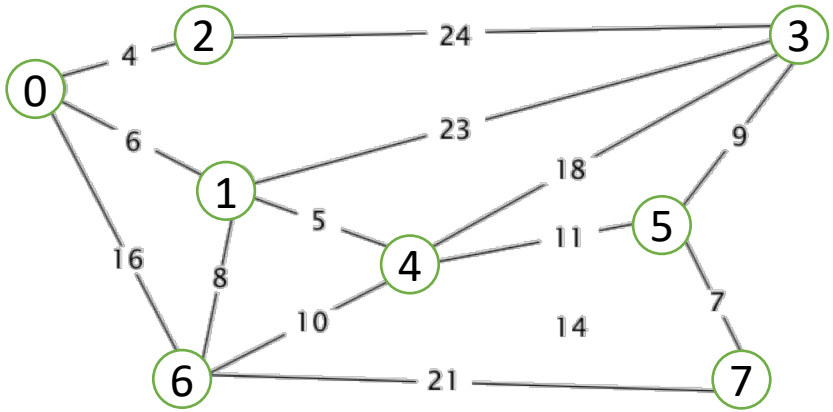
- Graph is connected
  - if not connected, more MSTs of connected components
- Weights are not proportional to distance
- Weights are not 0 or negative, that is V belongs to either $\mathbb{N}, \mathbb{I}$ *or* $\mathbb{R}$
  - it still works – though – but we make it simple here.
- Weights are unique, so $w_i \neq w_j$, for all i and j,
  - if not unique, there may be more than on MST.

A Minimum Spanning Tree.



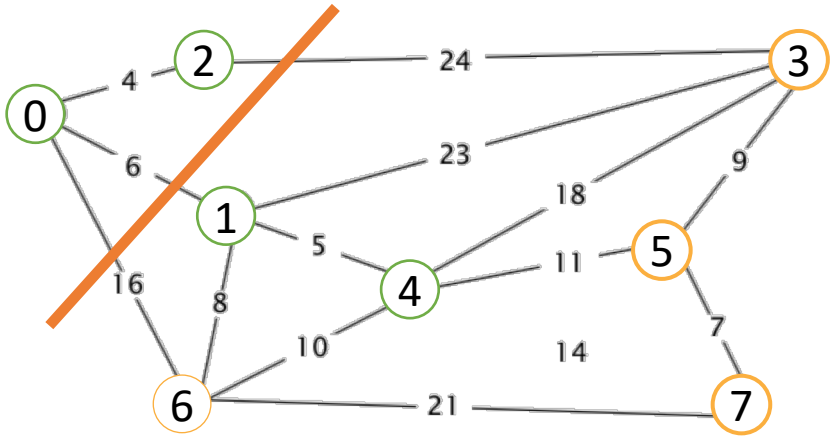
Finding the Minimum Spanning Tree: a 'greedy' algorithm

- A cut through a graph:
- Separates graph in two
- **Minimum path weight <u>crossing</u> the cut is part of MST!**



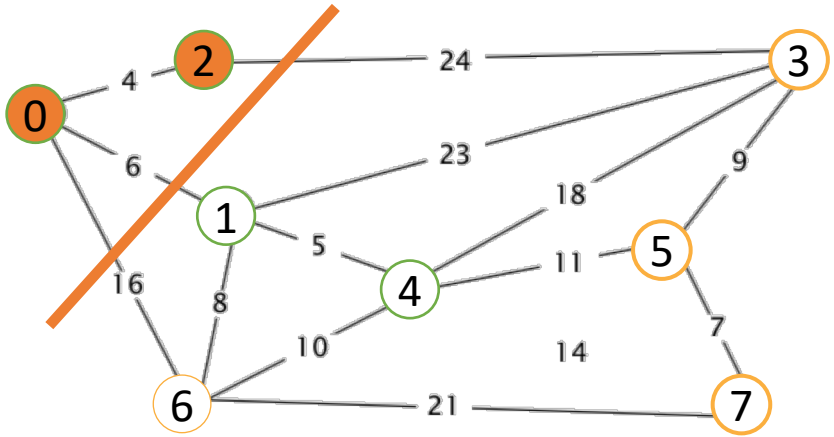
Finding the Minimum Spanning Tree: a 'greedy' algorithm

- A cut through a graph:
- Separates graph in two
- **Minimum path weight <u>crossing</u> the cut is part of MST!**



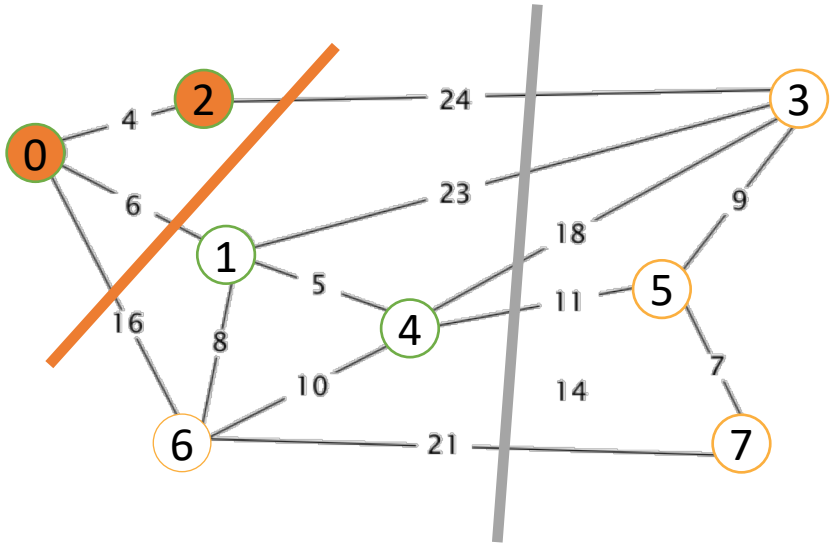
Finding the Minimum Spanning Tree: a 'greedy' algorithm

- A cut through a graph:
- Separates graph in two
- **Minimum path weight <u>crossing</u> the cut is part of MST!**



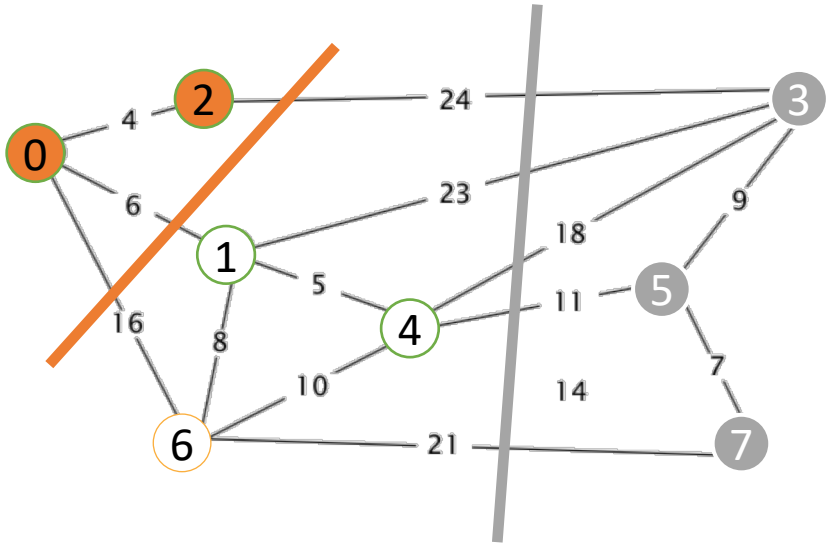
Finding the Minimum Spanning Tree: a 'greedy' algorithm

- A cut through a graph:
- Separates graph in two
- **Minimum path weight <u>crossing</u> the cut is part of MST!**



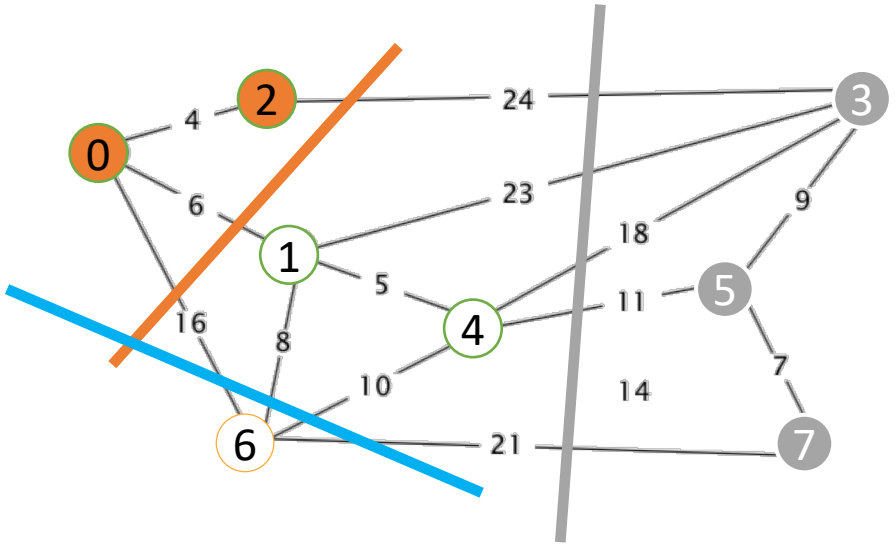
Finding the Minimum Spanning Tree: a 'greedy' algorithm

- A cut through a graph:
- Separates graph in two
- **Minimum path weight <u>crossing</u> the cut is part of MST!**



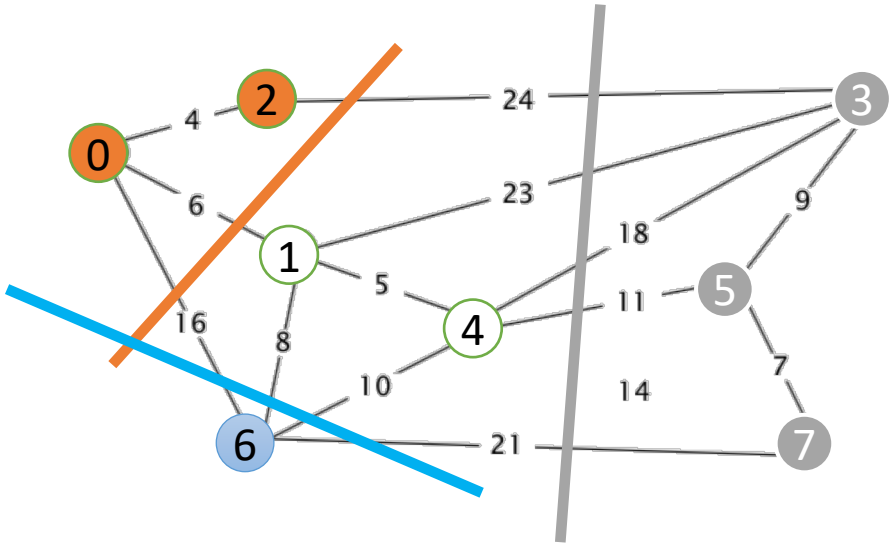
Finding the Minimum Spanning Tree: a 'greedy' algorithm

- A cut through a graph:
- Separates graph in two
- **Minimum path weight <u>crossing</u> the cut is part of MST!**



Finding the Minimum Spanning Tree: a 'greedy' algorithm

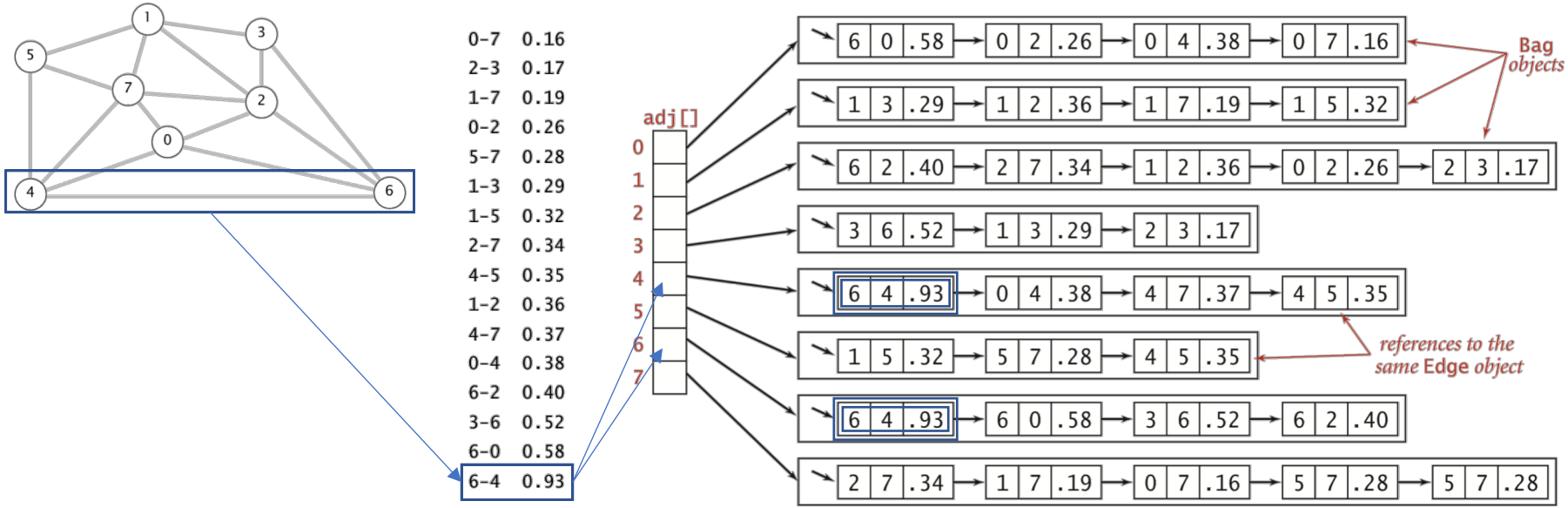
- A cut through a graph:
- Separates graph in two
- **Minimum path weight <u>crossing</u> the cut is part of MST!**



What Data Structure for edge-weighted graph?...

Pop Q.

What Data Structure for edge-weighted graph.



Exercise 1

Use the Book's MST API to test client running time for the five example graphs

- `tinyEWG.txt` (V=8, E=16)
- `mediumEWG.txt` (V=250, E=)
- `1000EWG.txt` (V=1000, E=8433)
- `10000EWG.txt` (V=10000, E=61731)
- `largeEWG.txt` (V=1000000, E=7586063)

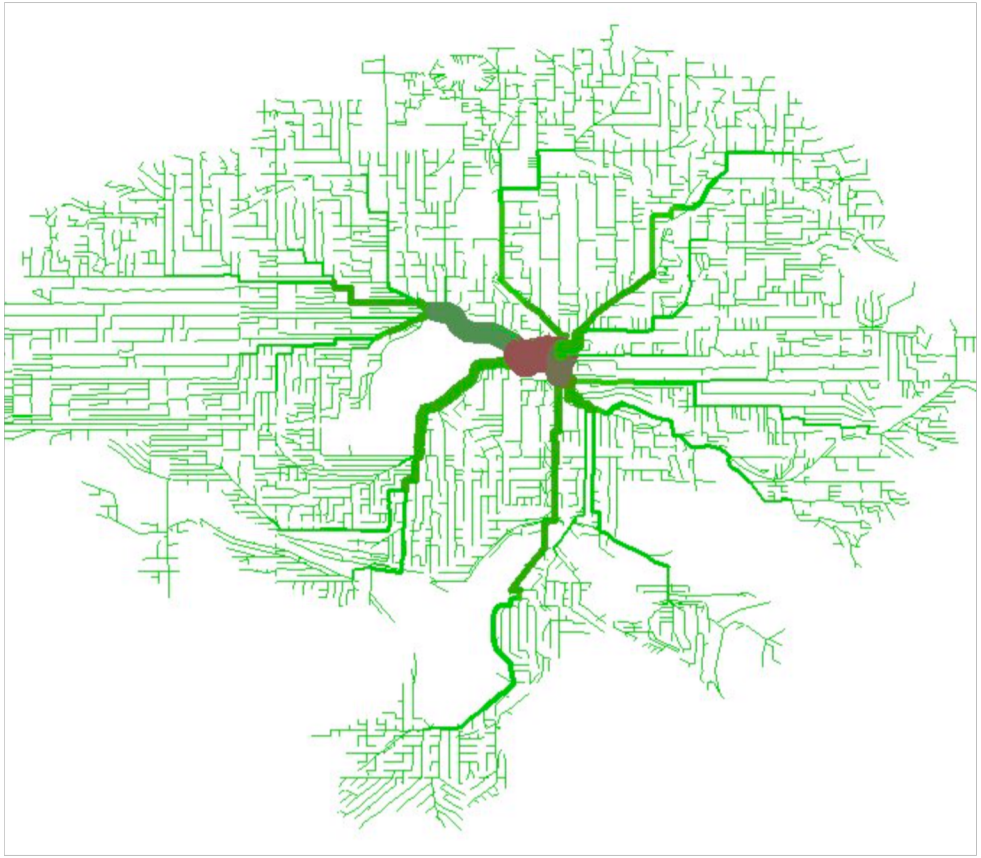
Time the computation, you can use fx the Book's `StopWatch()` class for this purpose. Or a system time diff, or you pick...

Bike Paths in Seattle

Looks almost like a brain to me....

How to find MSTs in graphs that are HUGE!!

=> Need somthing more efficient, enter Prim.

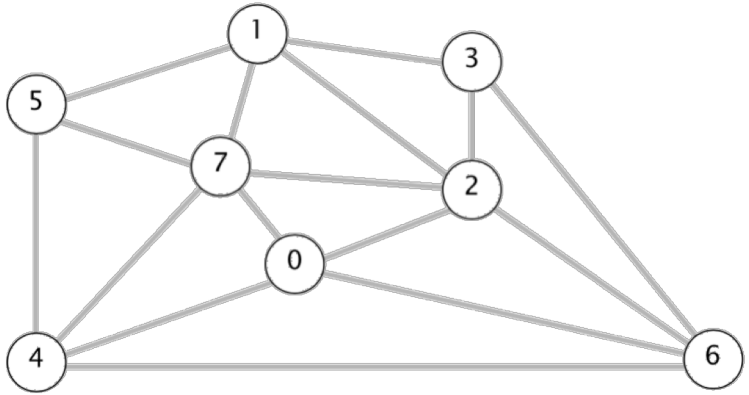


Minimum Spanning Trees

1. A ‘Greedy’ Heuristic Algorithm
2. Prim’s Algorithm (lazy version)

Prim's Algorithm

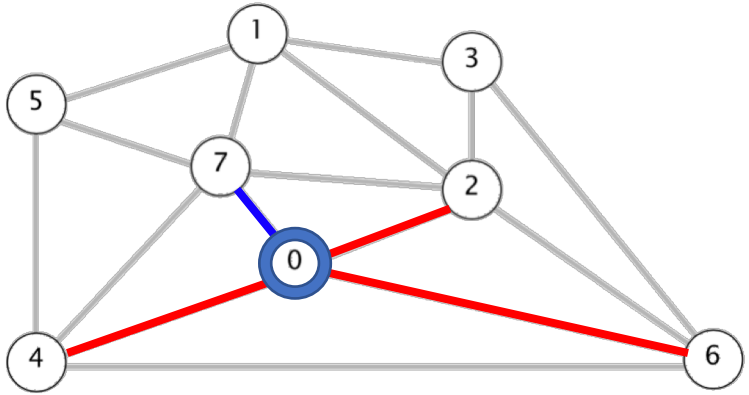
1. Start at vertex 0.
2. Greedily grow the tree, 'T'.
3. Add to '*T*' the minimum weight edge which has *exactly* one endpoint in '*T*'.
4. Repeat until we have *V* - 1 edges.




```
0-7  0.16
2-3  0.17
1-7  0.19
0-2  0.26
5-7  0.28
1-3  0.29
1-5  0.32
2-7  0.34
4-5  0.35
1-2  0.36
4-7  0.37
0-4  0.38
6-2  0.40
3-6  0.52
6-0  0.58
6-4  0.93
```

Prim's Algorithm

1. Start at vertex 0.
2. Greedily grow the tree, 'T'.
3. Add to '*T*' the minimum weight edge which has *exactly* one endpoint in '*T*'.
4. Repeat until we have *V* - 1 edges.

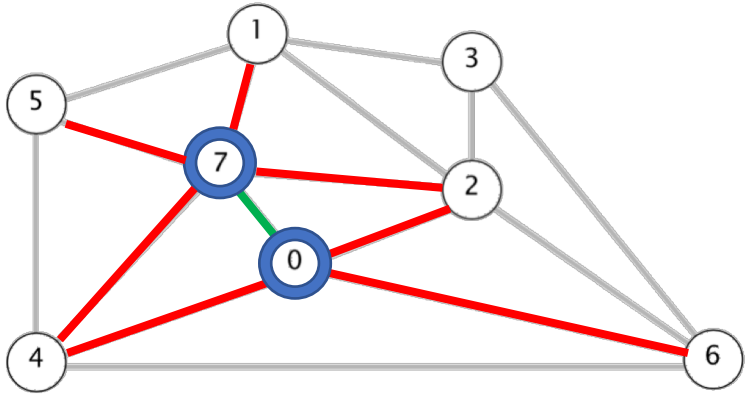


0–7	0.16
2–3	0.17
1–7	0.19
0–2	0.26
5–7	0.28
1–3	0.29
1–5	0.32
2–7	0.34
4–5	0.35
1–2	0.36
4–7	0.37
0–4	0.38
6–2	0.40
3–6	0.52
6–0	0.58
6–4	0.93

MST = $E_{0\text{-}7}$

Prim's Algorithm

1. Start at vertex 0.
2. Greedily grow the tree, 'T'.
3. Add to '*T*' the minimum weight edge which has *exactly* one endpoint in '*T*'.
4. Repeat until we have *V* - 1 edges.

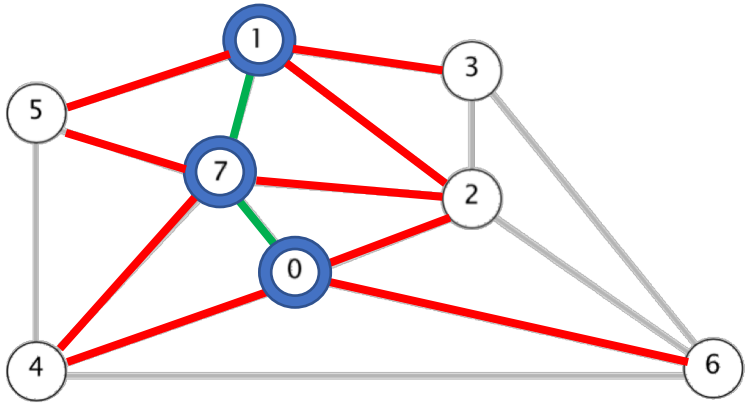


2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

MST = $E_{0\text{-}7}$, $E_{1\text{-}7}$,

Prim's Algorithm

1. Start at vertex 0.
2. Greedily grow the tree, 'T'.
3. Add to '*T*' the minimum weight edge which has *exactly* one endpoint in '*T*'.
4. Repeat until we have *V* - 1 edges.

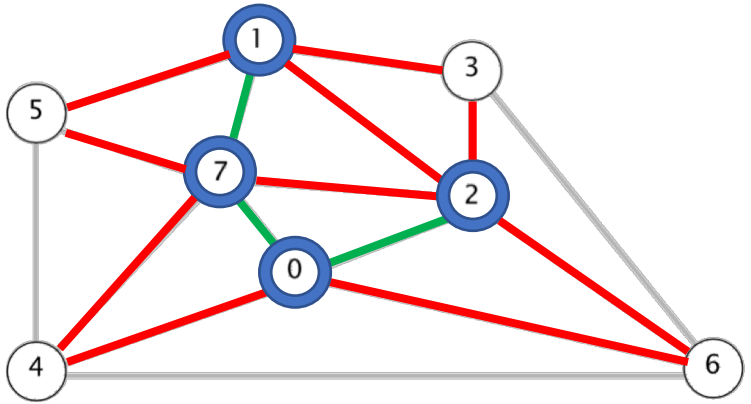


2-3	0.17
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

MST = $E_{0\text{-}7}$, $E_{1\text{-}7}$,

Prim's Algorithm

1. Start at vertex 0.
2. Greedily grow the tree, 'T'.
3. Add to '*T*' the minimum weight edge which has *exactly* one endpoint in '*T*'.
4. Repeat until we have *V* - 1 edges.

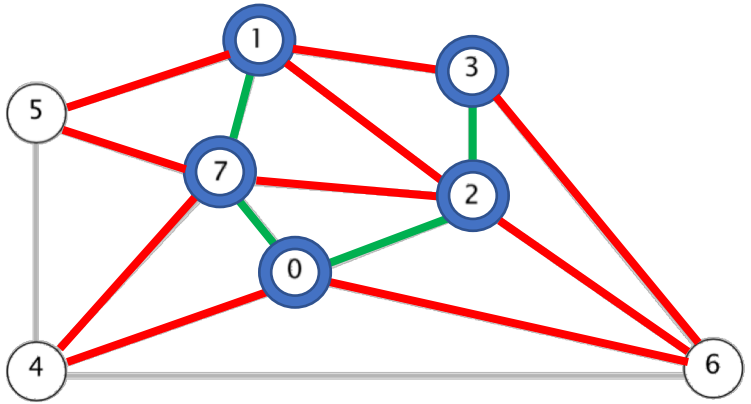


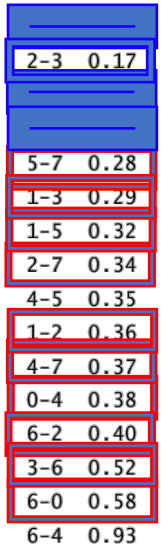
2-3	0.17
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

MST = $E_{0\text{-}7}$, $E_{1\text{-}7}$, $E_{0\text{-}2}$,

Prim's Algorithm

1. Start at vertex 0.
2. Greedily grow the tree, 'T'.
3. Add to '*T*' the minimum weight edge which has *exactly* one endpoint in '*T*'.
4. Repeat until we have *V* - 1 edges.

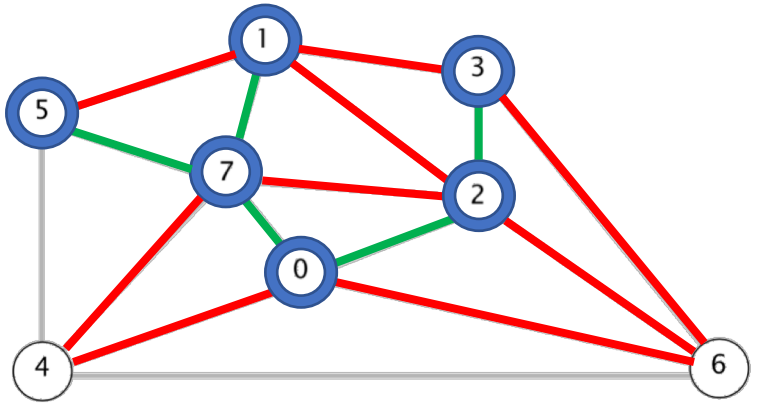


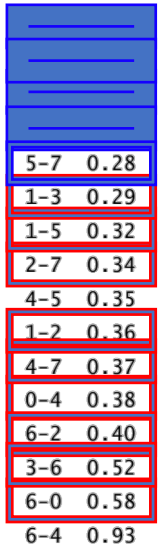


MST = $E_{0\text{-}7}$, $E_{1\text{-}7}$, $E_{0\text{-}2}$, $E_{2\text{-}3}$,

Prim's Algorithm

1. Start at vertex 0.
2. Greedily grow the tree, 'T'.
3. Add to '*T*' the minimum weight edge which has *exactly* one endpoint in '*T*'.
4. Repeat until we have *V* - 1 edges.

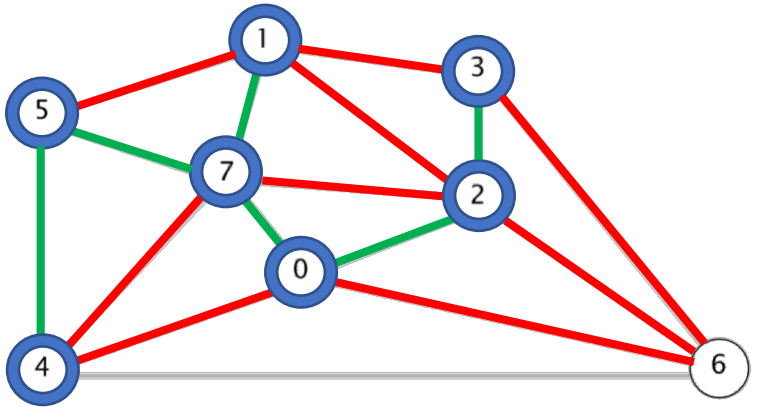


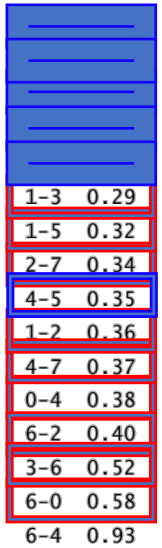


MST = $E_{0\text{-}7}$, $E_{1\text{-}7}$, $E_{0\text{-}2}$, $E_{2\text{-}3}$, $E_{5\text{-}7}$,

Prim's Algorithm

1. Start at vertex 0.
2. Greedily grow the tree, 'T'.
3. Add to '*T*' the minimum weight edge which has *exactly* one endpoint in '*T*'.
4. Repeat until we have *V* - 1 edges.

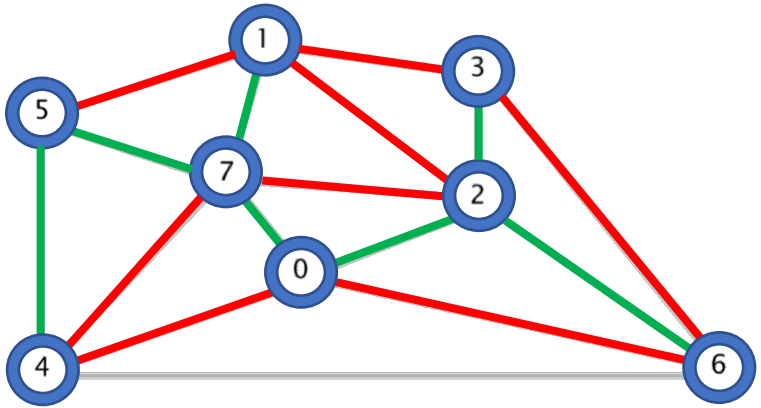


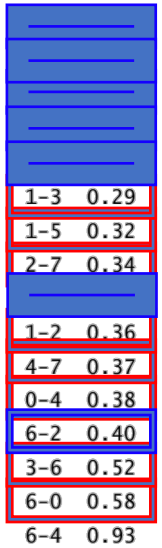


MST = $E_{0\text{-}7}$, $E_{1\text{-}7}$, $E_{0\text{-}2}$, $E_{2\text{-}3}$, $E_{5\text{-}7}$, $E_{4\text{-}5}$,

Prim's Algorithm

1. Start at vertex 0.
2. Greedily grow the tree, 'T'.
3. Add to '*T*' the minimum weight edge which has *exactly* one endpoint in '*T*'.
4. Repeat until we have *V* - 1 edges.

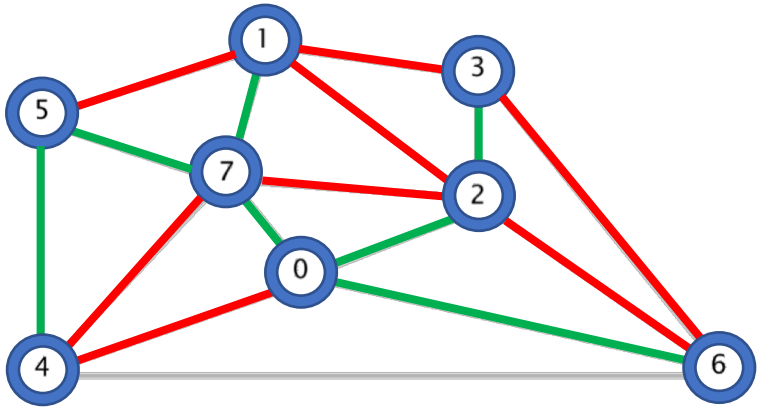


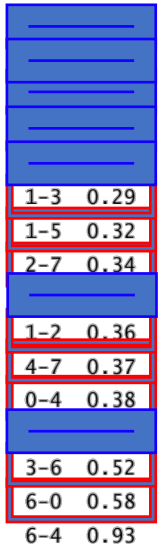


MST = $E_{0\text{-}7}$, $E_{1\text{-}7}$, $E_{0\text{-}2}$, $E_{2\text{-}3}$, $E_{5\text{-}7}$, $E_{4\text{-}5}$, $E_{6\text{-}2}$

Prim's Algorithm

1. Start at vertex 0.
2. Greedily grow the tree, 'T'.
3. Add to '*T*' the minimum weight edge which has *exactly* one endpoint in '*T*'.
4. Repeat until we have *V* - 1 edges.

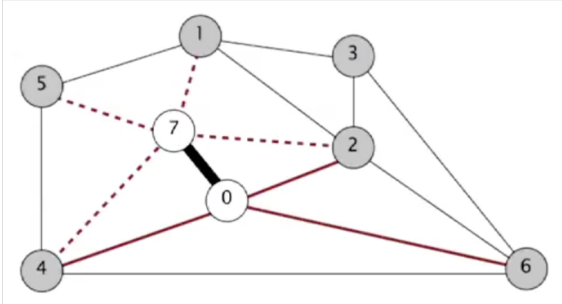




MST = $E_{0\text{-}7}$, $E_{1\text{-}7}$, $E_{0\text{-}2}$, $E_{2\text{-}3}$, $E_{5\text{-}7}$, $E_{4\text{-}5}$, $E_{6\text{-}2}$

Lazy Prim's Algorithm

1. Start at vertex 0.
2. Greedily grow the tree, 'T'.
3. Remove newest vertex from minPQ.
4. Add new possible edges to a minPQ.
5. Add to '*T*' the minimum weight edge which has *exactly* one endpoint in '*T*'.
6. Discard possible obsolete edges.
7. Repeat until we have *V* - 1 edges and discard remaining obsolete edges.



edges on PQ (sorted by weight)		
*	1–7	0.19
	0–2	0.26
*	5–7	0.28
*	2–7	0.34
*	4–7	0.37
	0–4	0.38
	6–0	0.58

0–7	0.16
2–3	0.17
1–7	0.19
0–2	0.26
5–7	0.28
1–3	0.29
1–5	0.32
2–7	0.34
4–5	0.35
1–2	0.36
4–7	0.37
0–4	0.38
6–2	0.40
3–6	0.52
6–0	0.58
6–4	0.93

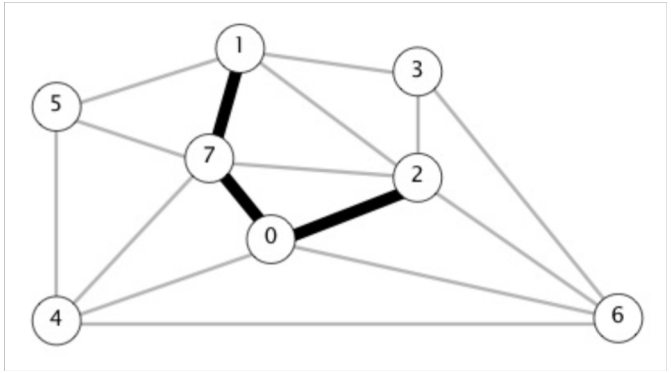
MST = $E_{0\text{-}7}$, $E_{1\text{-}7}$, $E_{0\text{-}2}$, $E_{2\text{-}3}$, $E_{5\text{-}7}$, $E_{4\text{-}5}$, $E_{6\text{-}2}$

Exercise 2

The *LAZY* version of Prim's algorithm has added edges E_{0-7}, E_{1-7}, and E_{0-2}, to the MST so far.

Q: What are the keys on the priority queue immediately after E_{0-2} is added?

- 0.17, 0.26, 0.28, 0.29, 0.38, 0.40
- 0.17, 0.28, 0.29, 0.38, 0.40
- 0.17, 0.28, 0.29, 0.32, 0.37, 0.38, 0.58
- 0.17, 0.28, 0.29, 0.32, 0.34, 0.36, 0.37, 0.38, 0.40, 0.58




```
0-7  0.16
2-3  0.17
1-7  0.19
0-2  0.26
5-7  0.28
1-3  0.29
1-5  0.32
2-7  0.34
4-5  0.35
1-2  0.36
4-7  0.37
0-4  0.38
6-2  0.40
3-6  0.52
6-0  0.58
6-4  0.93
```

Exercise 3

Use the Book's Lazy Prim API to test client running time for the five example graphs

- `tinyEWG.txt` (V=8, E=16)
- `mediumEWG.txt` (V=250, E=)
- `1000EWG.txt` (V=1000, E=8433)
- `10000EWG.txt` (V=10000, E=61731)
- `largeEWG.txt` (V=1000000, E=7586063)

Time the computation, you can use fx the Book's `StopWatch()` class for this purpose. Or a system time diff, or you pick...

Compare timing results with the Greedy MST algorithm.

Prim's MST Algorithm – Complexity

- 'Lazy' Prim uses E extra space, and time LogE, worst case.

- **Q: Which PQ to use?**
  - dense, sparse, size, time??

operation	frequency	binary heap
delete min	E	$\log E$
insert	E	$\log E$

PQ implementation	insert	delete-min	decrease-key	total
unordered array	1	V	1	V^2
binary heap	$\log V$	$\log V$	$\log V$	$E \log V$
d-way heap	$\log_d V$	$d \log_d V$	$\log_d V$	$E \log_{E/V} V$
Fibonacci heap	1 †	$\log V$ †	1 †	$E + V \log V$

MST Complexity; ”All” Algorithms.

Algorithm	Space	Time
Lazy Prim	E	E Log E
Eager Prim	*V*	*E Log V*
Kruskal	*E*	*E Log E*
Fredman-Tarjan	*V*	*E + V Log V*
Impossible	*V*	*E?*

Acknowledgement

- Some material has been taken from the Princeton course ‘Algorithms 4th Edition’, by Sedgewick & Wayne.