

COMP2521 26T1

Graphs (VII)

Minimum Spanning Trees

Kevin Luxa

`cs2521@cse.unsw.edu.au`

minimum spanning trees
kruskal's algorithm
prim's algorithm

A **spanning tree** of an undirected graph G is a subgraph of G that contains all vertices of G , that is connected and contains no cycles

A **minimum spanning tree** of an undirected weighted graph G is a spanning tree of G that has minimum total edge weight among all spanning trees of G

Applications:

Electrical grids, networks

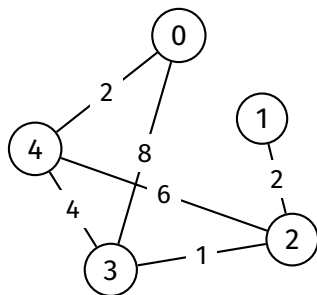
Any situation where we want to connect nodes as cheaply as possible

Minimum
Spanning
TreesKruskal's
AlgorithmPrim's
Algorithm

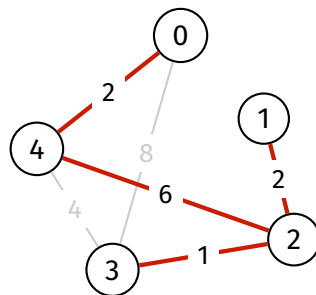
Comparison

Other
Algorithms

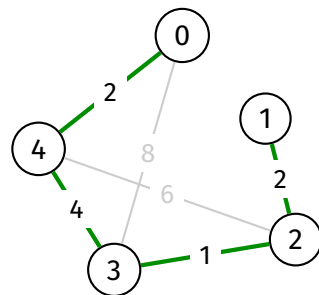
Appendix



Original graph



Spanning tree



Minimum spanning tree

Basic minimum spanning tree algorithms:

- Kruskal's algorithm
- Prim's algorithm

Minimum
Spanning
Trees

Kruskal's
Algorithm

Example
Pseudocode
Analysis

Prim's
Algorithm

Comparison

Other
Algorithms

Appendix

Invented by
American mathematician, statistician, computer scientist
Joseph Kruskal in 1956



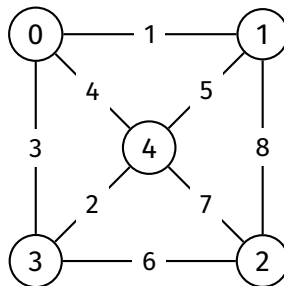
Algorithm:

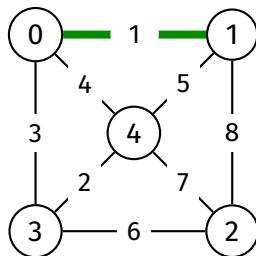
- ① Start with an empty graph
 - With same vertices as original graph
- ② Consider edges in increasing weight order
 - Add edge if it does not form a cycle in the MST
- ③ Repeat until $V - 1$ edges have been added

Critical operations:

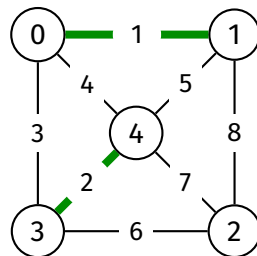
- Iterating over edges in weight order
- Checking if adding an edge would form a cycle

Run Kruskal's algorithm on this graph:

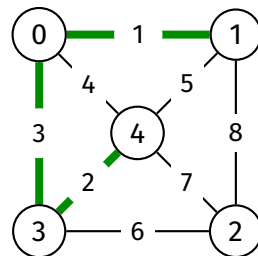




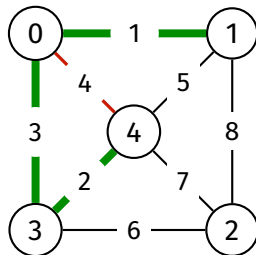
Add 0-1



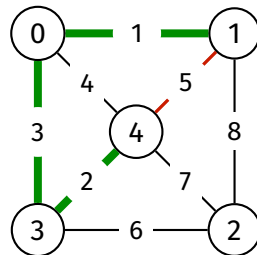
Add 3-4



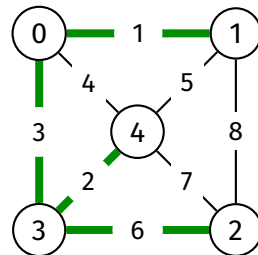
Add 0-3



Don't add 0-4



Don't add 1-4



Add 2-3

Minimum
Spanning
Trees

Kruskal's
Algorithm

Example

Pseudocode

Analysis

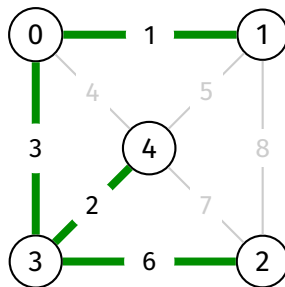
Prim's
Algorithm

Comparison

Other
Algorithms

Appendix

MST:



```
kruskalMst( $G$ ):
```

```
    Input: graph  $G$  with  $V$  vertices
```

```
    Output: minimum spanning tree of  $G$ 
```

```
    mst = empty graph with  $V$  vertices
```

```
    sortedEdges = sort edges of  $G$  by weight
```

```
    for each edge  $e$  in sortedEdges:
```

```
        add  $e$  to mst
```

```
        if mst has a cycle:
```

```
            remove  $e$  from mst
```

```
    if mst has  $V - 1$  edges:
```

```
        return mst
```

```
kruskalMst( $G$ ):
```

```
    Input: graph  $G$  with  $V$  vertices
```

```
    Output: minimum spanning tree of  $G$ 
```

```
    mst = empty graph with  $V$  vertices
```

```
    sortedEdges = sort edges of  $G$  by weight
```

```
    for each edge  $(v, w, \text{weight})$  in sortedEdges:
```

```
        if there is no path between  $v$  and  $w$  in mst:  
            add edge  $(v, w, \text{weight})$  to mst
```

```
    if mst has  $V - 1$  edges:
```

```
        return mst
```

Proof by exchange argument.

Idea:

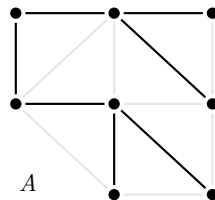
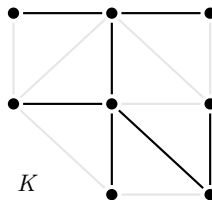
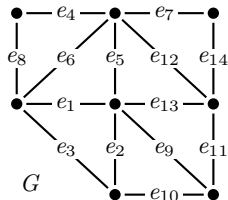
- Suppose there exists another algorithm A which makes a different set of choices
 - In this case, chooses a different set of edges for the MST
- Identify *one* choice made by A which is not made by our algorithm
- Show that by exchanging that choice with one of the choices made by our algorithm, the solution does not become worse or less optimal
 - In this case, the “solution” is the spanning tree produced
 - In this case, an “optimal” solution is a spanning tree that costs as little as possible

Sort the edges of G in increasing order.

Let K be the set of edges selected by Kruskal's algorithm.

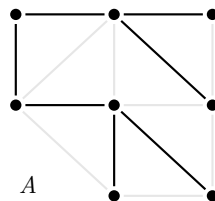
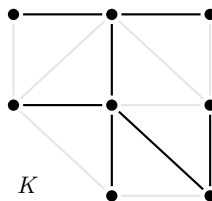
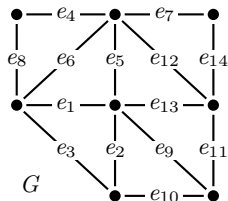
Let A be the set of edges selected by a different algorithm.

edges of G	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	$\dots$
edges of K	e_1	e_2		e_4	e_5		e_7		e_9	$\dots$
edges of A	e_1	e_2		e_4			e_7	e_8	e_9	$\dots$



Consider the first edge that is chosen by K but not by A .

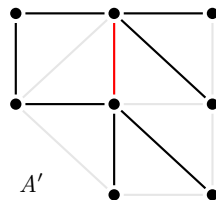
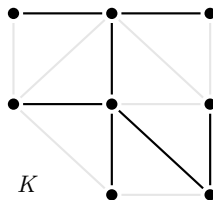
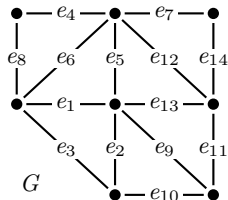
edges of G	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	$\dots$
edges of K	e_1	e_2		e_4	e_5		e_7		e_9	$\dots$
edges of A	e_1	e_2		e_4			e_7	e_8	e_9	$\dots$



Consider the first edge that is chosen by K but *not* by A .

Add this edge to a copy of A (call it A'). This forms a cycle in A' .

edges of G	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	$\dots$
edges of K	e_1	e_2		e_4	e_5		e_7		e_9	$\dots$
edges of A'	e_1	e_2		e_4	e_5		e_7	e_8	e_9	$\dots$

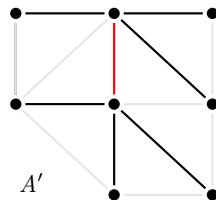
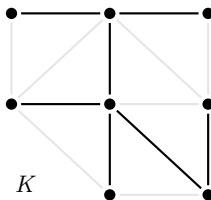
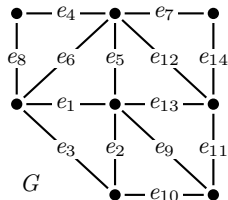


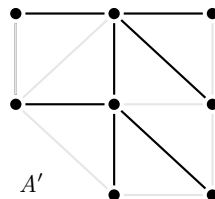
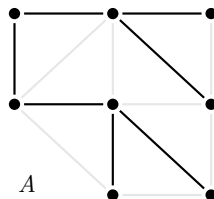
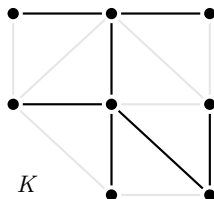
Consider the first edge that is chosen by K but *not* by A .

Add this edge to a copy of A (call it A'). This forms a cycle in A' .

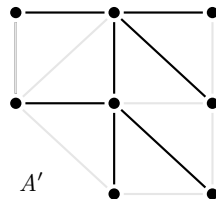
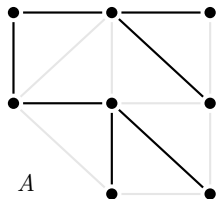
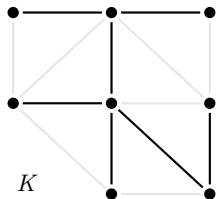
Now find the highest-weight edge in this cycle and *remove* it from A' .

edges of G	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	$\dots$
edges of K	e_1	e_2		e_4	e_5		e_7		e_9	$\dots$
edges of A'	e_1	e_2		e_4	e_5		e_7	e_8	e_9	$\dots$





Now A' is once again a spanning tree,
but it is more similar to K than A and it costs no more than A .



Now A' is once again a spanning tree,
but it is more similar to K than A and it costs no more than A .

Repeat until A' is identical to K . Each time we perform an exchange, the spanning tree does not increase in cost.

Therefore, K is an optimal spanning tree (MST).

Analysis:

- Sorting edges is $O(E \cdot \log E)$
- Main loop has at most E iterations
- Different ways to check if adding an edge would form a cycle
 - Cycle/path checking is $O(V)$ in the worst case (adjacency list)
 $\Rightarrow$ overall cost = $O(E \cdot \log E + E \cdot V) = O(E \cdot V)$
 - Using union-find data structure is close to $O(1)$ in the worst case
 $\Rightarrow$ overall cost = $O(E \cdot \log E + E) = O(E \cdot \log E) = O(E \cdot \log V)$

Minimum
Spanning
Trees

Kruskal's
Algorithm

**Prim's
Algorithm**

Example

Pseudocode
Analysis

Comparison

Other
Algorithms

Appendix

Developed by Vojtěch Jarník in 1930
and rediscovered by Robert C. Prim in 1957



Vojtěch Jarník



Robert C. Prim

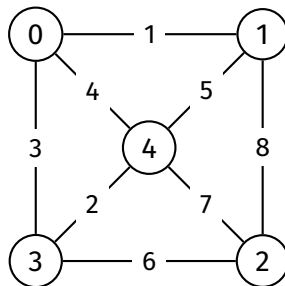
Algorithm:

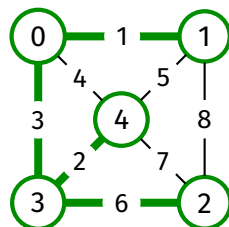
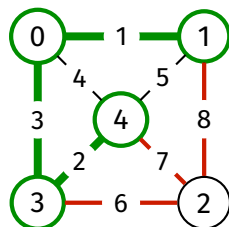
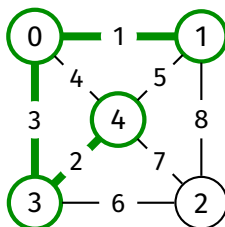
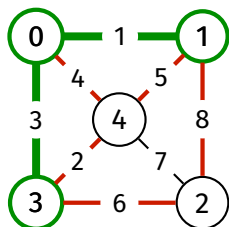
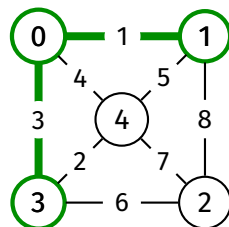
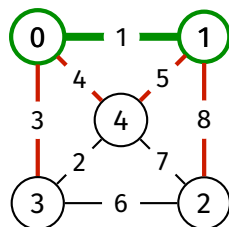
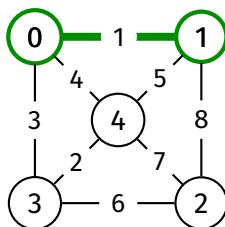
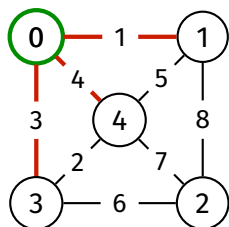
- ① Start with an empty graph
- ② Start from any vertex, add it to the MST
- ③ Choose cheapest edge $s-t$ such that:
 - s has been added to the MST, and
 - t has not been added to the MSTand add this edge and the vertex t to the MST
- ④ Repeat previous step until $V - 1$ edges have been added
 - Or until all vertices have been added

Critical operations:

- Finding the cheapest edge $s-t$ such that s has been added to the MST and t has not been added to the MST

Run Prim's algorithm on this graph (starting at 0):





Minimum
Spanning
TreesKruskal's
AlgorithmPrim's
Algorithm

Example

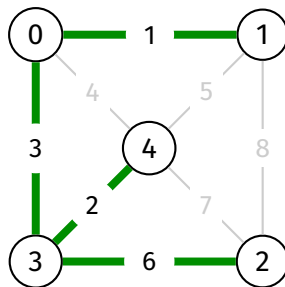
Pseudocode
Analysis

Comparison

Other
Algorithms

Appendix

MST:



primMst(G):

Input: graph G with V vertices

Output: minimum spanning tree of G

mst = empty graph with V vertices

usedV = {0}

unusedE = edges of G

while $|\text{usedV}| < V$:

 find cheapest edge e (s, t, weight) in unusedE such that
 $s \in \text{usedV}$ **and** $t \notin \text{usedV}$

 add e to mst

 add t to usedV

 remove e from unusedE

return mst

Analysis:

- Algorithm considers at most E edges $\Rightarrow O(E)$
- Loop has V iterations
- In each iteration, finding the minimum-weighted edge:
 - With set of edges is $O(E)$
 $\Rightarrow$ overall cost = $O(E + V \cdot E) = O(V \cdot E)$
 - With Fibonacci heap is $O(\log E) = O(\log V)$
 $\Rightarrow$ overall cost = $O(E + V \cdot \log V)$

Kruskal's algorithm...

- is $O(E \cdot \log V)$
- uses array-based data structures
- performs better on sparse graphs

Prim's algorithm...

- is $O(E + V \cdot \log V)$
- uses complex linked data structures
 - in its most efficient implementation (Fibonacci heap)
- performs better on dense graphs

- Boruvka's algorithm
 - Oldest MST algorithm
 - Start with V separate components
 - Join components using min cost links
 - Continue until only a single component
 - Worst-case time complexity: $O(E \cdot \log V)$
- Karger, Klein and Tarjan
 - Based on Boruvka's algorithm, but non-deterministic
 - Randomly selects subset of edges to consider
 - Time complexity: $O(E)$ on average

Minimum
Spanning
Trees

Kruskal's
Algorithm

Prim's
Algorithm

Comparison

Other
Algorithms

Appendix

<https://forms.office.com/r/ucsJ4r99Ag>



Minimum
Spanning
Trees

Kruskal's
Algorithm

Prim's
Algorithm

Comparison

Other
Algorithms

Appendix

Kruskal's Algorithm
Example

Prim's Algorithm
Example

Appendix

Minimum
Spanning
TreesKruskal's
AlgorithmPrim's
Algorithm

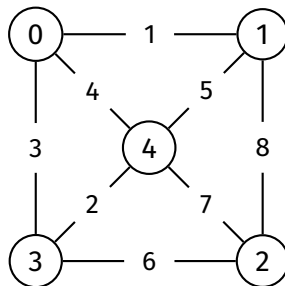
Comparison

Other
Algorithms

Appendix

Kruskal's Algorithm
ExamplePrim's Algorithm
Example

Original graph



Minimum
Spanning
TreesKruskal's
AlgorithmPrim's
Algorithm

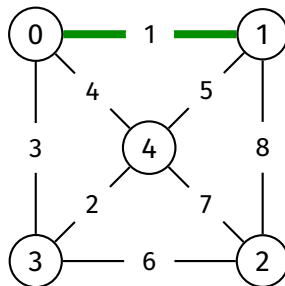
Comparison

Other
Algorithms

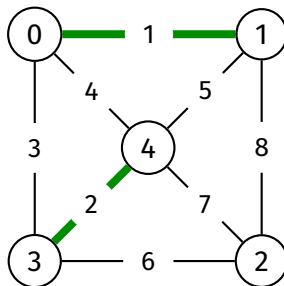
Appendix

Kruskal's Algorithm
ExamplePrim's Algorithm
Example

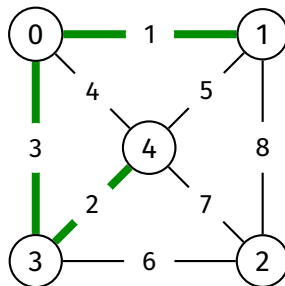
Adding 0-1 would not create a cycle



Adding 3-4 would not create a cycle



Adding 0-3 would not create a cycle



Minimum
Spanning
TreesKruskal's
AlgorithmPrim's
Algorithm

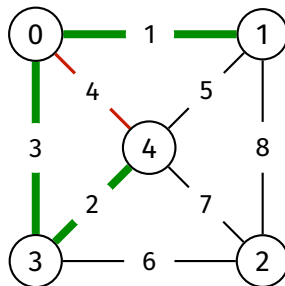
Comparison

Other
Algorithms

Appendix

Kruskal's Algorithm
ExamplePrim's Algorithm
Example

Adding 0-4 would create a cycle



Minimum
Spanning
TreesKruskal's
AlgorithmPrim's
Algorithm

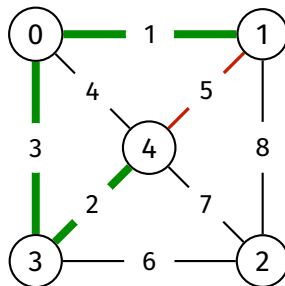
Comparison

Other
Algorithms

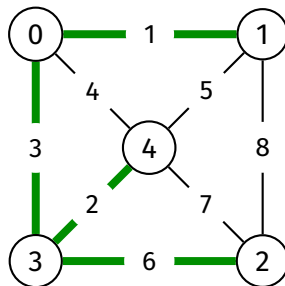
Appendix

Kruskal's Algorithm
ExamplePrim's Algorithm
Example

Adding 1-4 would create a cycle



Adding 2-3 would not create a cycle



Minimum
Spanning
TreesKruskal's
AlgorithmPrim's
Algorithm

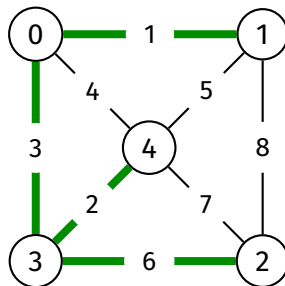
Comparison

Other
Algorithms

Appendix

Kruskal's Algorithm
ExamplePrim's Algorithm
Example

Done - MST has 4 edges



Minimum
Spanning
Trees

Kruskal's
Algorithm

Prim's
Algorithm

Comparison

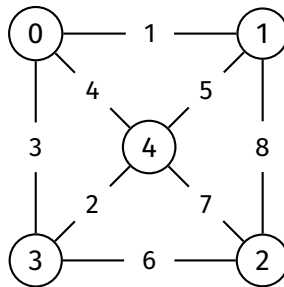
Other
Algorithms

Appendix

Kruskal's Algorithm
Example

Prim's Algorithm
Example

Original graph



Minimum
Spanning
TreesKruskal's
AlgorithmPrim's
Algorithm

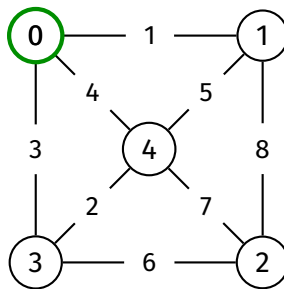
Comparison

Other
Algorithms

Appendix

Kruskal's Algorithm
ExamplePrim's Algorithm
Example

Start at vertex 0



Minimum
Spanning
TreesKruskal's
AlgorithmPrim's
Algorithm

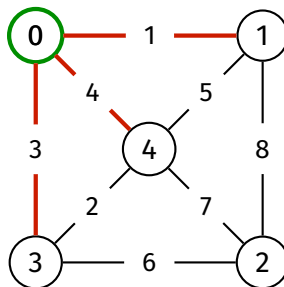
Comparison

Other
Algorithms

Appendix

Kruskal's Algorithm
ExamplePrim's Algorithm
Example

Choose cheapest edge out of these (in red)



Minimum
Spanning
TreesKruskal's
AlgorithmPrim's
Algorithm

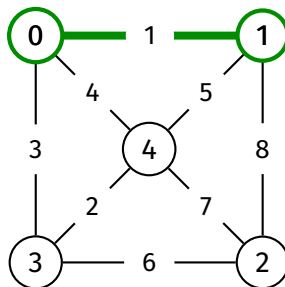
Comparison

Other
Algorithms

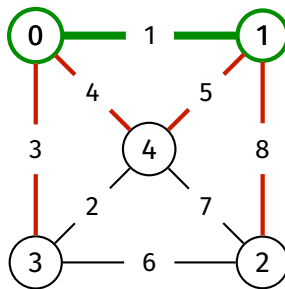
Appendix

Kruskal's Algorithm
ExamplePrim's Algorithm
Example

Add 0-1 to MST



Choose cheapest edge out of these (in red)



Minimum
Spanning
TreesKruskal's
AlgorithmPrim's
Algorithm

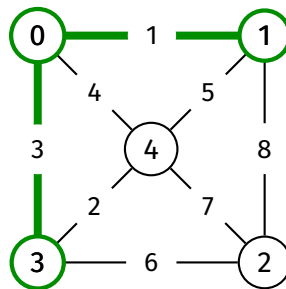
Comparison

Other
Algorithms

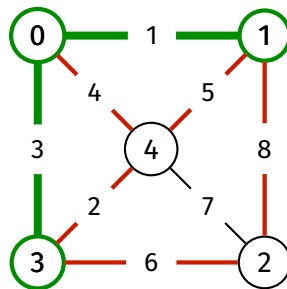
Appendix

Kruskal's Algorithm
ExamplePrim's Algorithm
Example

Add 0-3 to MST



Choose cheapest edge out of these (in red)



Minimum
Spanning
TreesKruskal's
AlgorithmPrim's
Algorithm

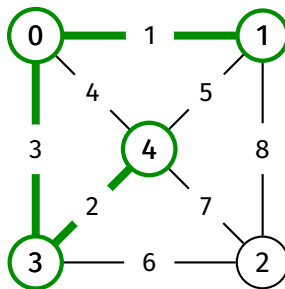
Comparison

Other
Algorithms

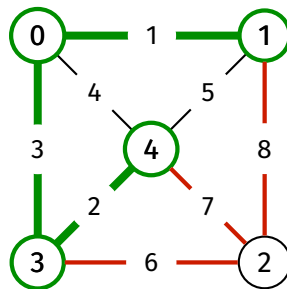
Appendix

Kruskal's Algorithm
ExamplePrim's Algorithm
Example

Add 3-4 to MST



Choose cheapest edge out of these (in red)



Minimum
Spanning
TreesKruskal's
AlgorithmPrim's
Algorithm

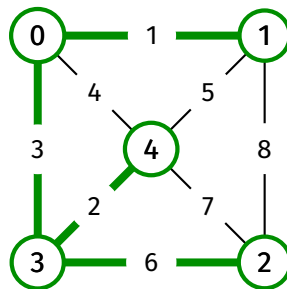
Comparison

Other
Algorithms

Appendix

Kruskal's Algorithm
ExamplePrim's Algorithm
Example

Add 3-2 to MST



Minimum
Spanning
TreesKruskal's
AlgorithmPrim's
Algorithm

Comparison

Other
Algorithms

Appendix

Kruskal's Algorithm
ExamplePrim's Algorithm
Example

Done - MST has 4 edges

