COMP20230: Data Structures & Algorithms
Lecture 18: Graphs (3)

Dr Andrew Hines

Office: E3.13 Science East
School of Computer Science
University College Dublin

andrew.hines@ucd.ie

Outline

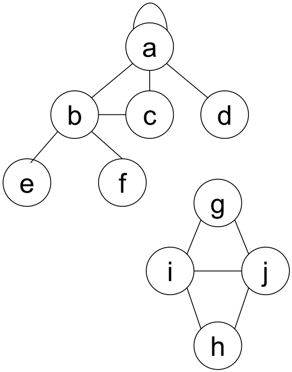
Today

- Minimum Spanning Tree
- Prim's Algorithm
- Kruskal's Algorithm

Take home message

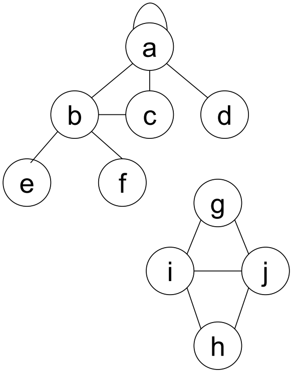
Finding the connected set of edges connecting all vertices can be done using two algorithms: Prim's and Kruskal's

Unweighted, Undirected Graph: Adjacency List



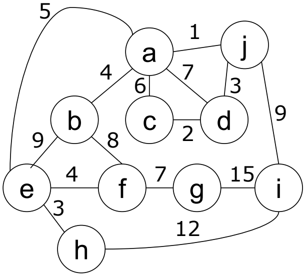
a	a, b, c, d
b	a, c, e, f
c	a, b
d	a
e	b
f	b
g	i, j
h	i, j
i	g, h, j
j	g, h, i

Unweighted, Undirected Graph: Adjacency Matrix



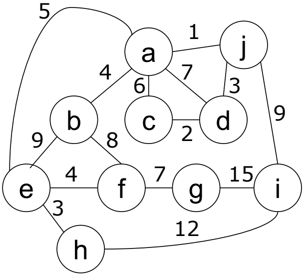
	a	b	c	d	e	f	g	h	i	j
a	2	1	1	1						
b	1		1		1	1				
c	1	1								
d	1									
e		1								
f		1								
g									1	1
h									1	1
i							1	1		1
j							1	1	1	

Weighted, Undirected Graph: Adjacency List



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

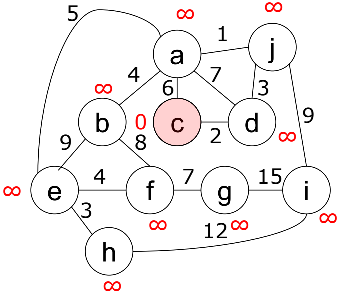
Weighted, Undirected Graph: Adjacency Matrix



	a	b	c	d	e	f	g	h	i	j
a		4	6	7	5					1
b	4				9	8				
c	6			2						
d	7		2							3
e	5	9				4		3		
f		8			4		7			
g						7			15	
h					3				12	
i							15	12		9
j	1			3					9	

Dijkstra’s Algorithm

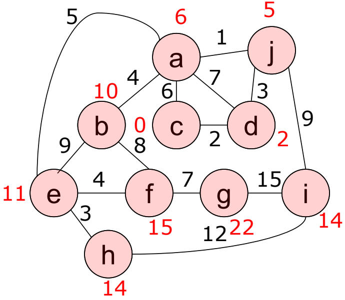
Visited = {c}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Dijkstra's Algorithm

Visited = {c, d, j, a, b, e, i, h, f, g}

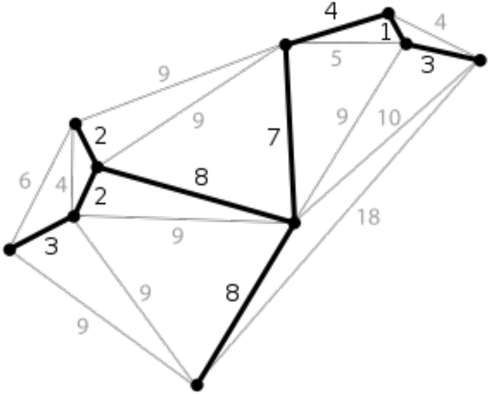


a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

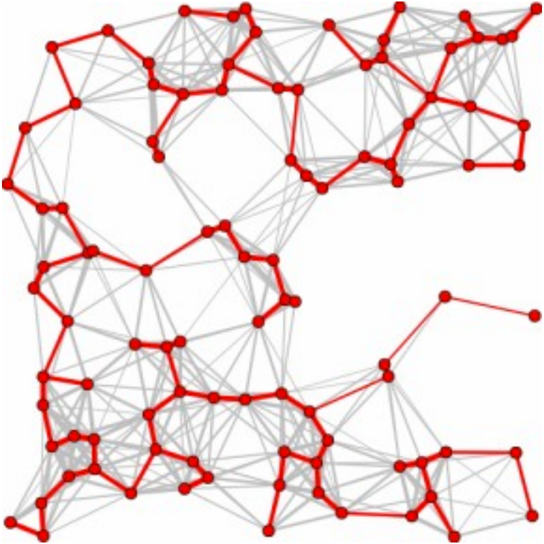
Minimum Spanning Tree (MST)

Minimum Spanning Tree

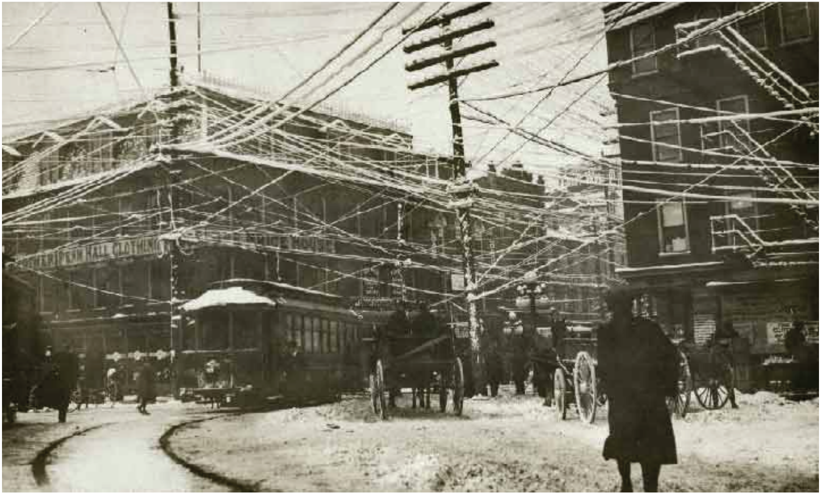
Tree spanning a connected, undirected graph. It connects all the vertices together with the **minimal total weighting** for its edges.



MST: Complex

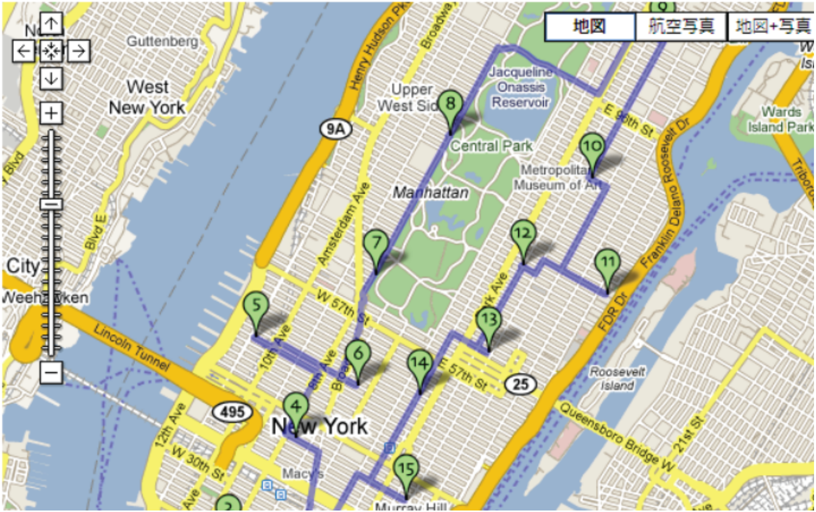


MST Applications

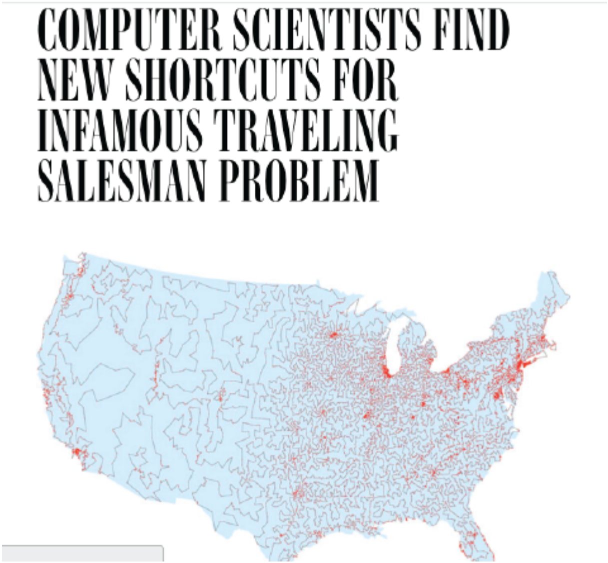


MST Applications

MST Applications



MST Applications

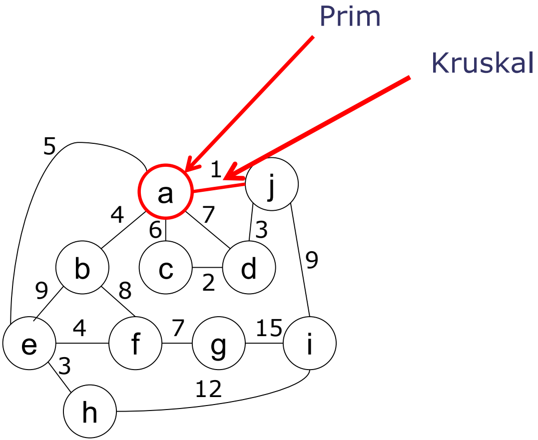


Finding the Minimum Spanning Tree

Algorithms approach same problem in different ways

Prim: Start with a random node

Kruskal: Start with (the smallest) edge



Prim's Algorithm

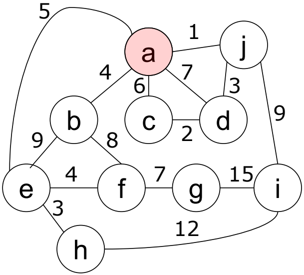
Inputs and outputs

We have a graph: a weighted, undirected graph.
We want to create a tree: A minimum spanning tree (but it is a tree).

1. Pick a random vertex from the graph as the tree root node
2. From the edges connecting to neighbours (excluding ones already in the tree), find the minimum-weighted edge, and add it into the tree
3. Move along the edge to the vertex and mark it as visited
4. Repeat from step 2 until all vertices are represented in the tree

Prim's Algorithm

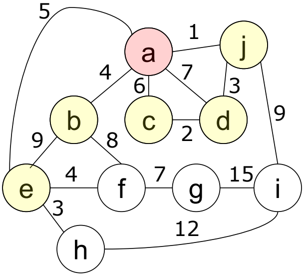
Visited = {a}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Prim's Algorithm

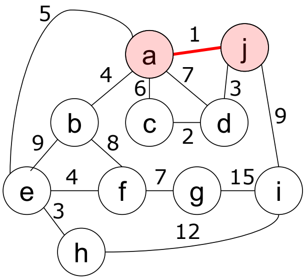
Visited = {a}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Prim's Algorithm

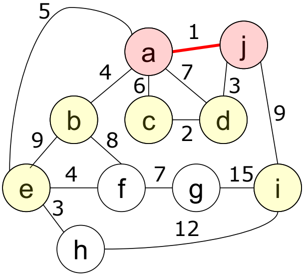
Visited = {a, j}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Prim's Algorithm

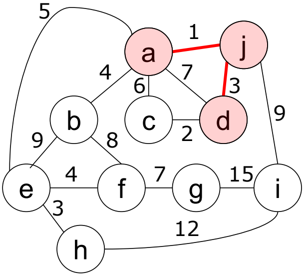
Visited = {a, j}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

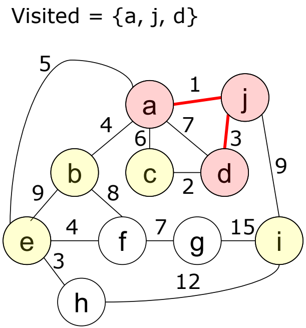
Prim's Algorithm

Visited = {a, j, d}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

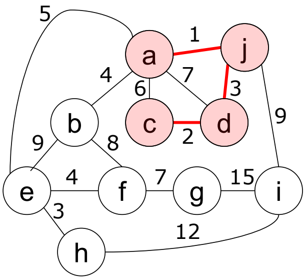
Prim's Algorithm



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Prim's Algorithm

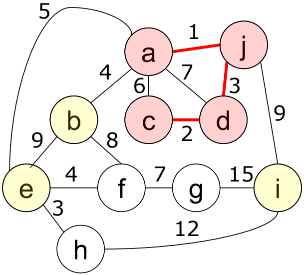
Visited = {a, j, d, c}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Prim's Algorithm

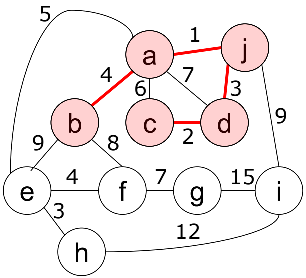
Visited = {a, j, d, c}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Prim's Algorithm

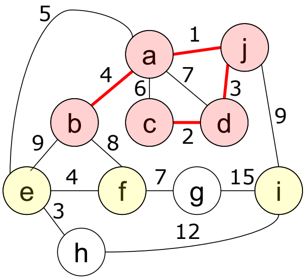
Visited = {a, j, d, c, b}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Prim's Algorithm

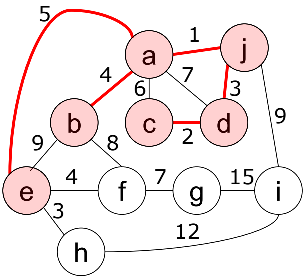
Visited = {a, j, d, c, b}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Prim's Algorithm

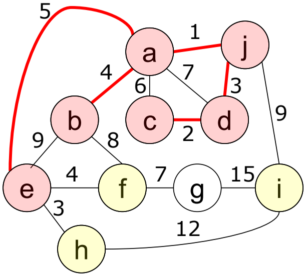
Visited = {a, j, d, c, b, e}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Prim's Algorithm

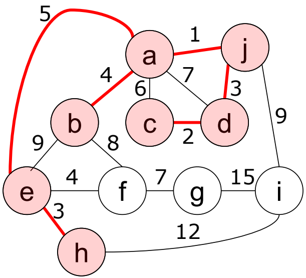
Visited = {a, j, d, c, b, e}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

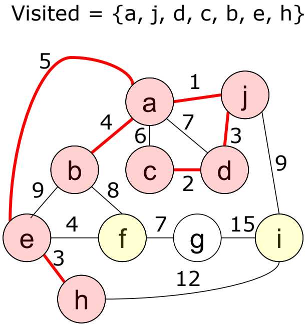
Prim's Algorithm

Visited = {a, j, d, c, b, e, h}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

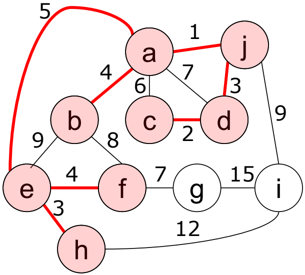
Prim's Algorithm



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Prim's Algorithm

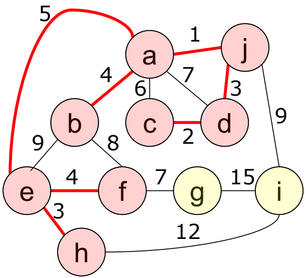
Visited = {a, j, d, c, b, e, h, f}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Prim's Algorithm

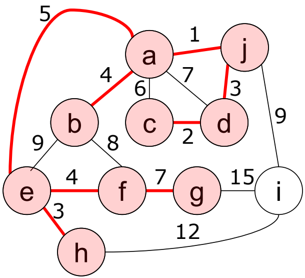
Visited = {a, j, d, c, b, e, h, f}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Prim's Algorithm

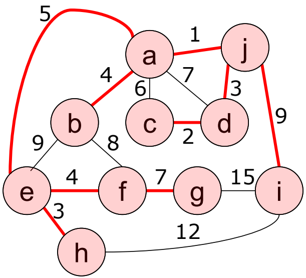
Visited = {a, j, d, c, b, e, h, f, g}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Prim's Algorithm

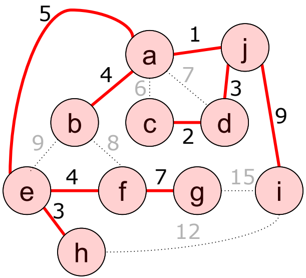
Visited = {a, j, d, c, b, e, h, f, g, i}



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Prim's Algorithm

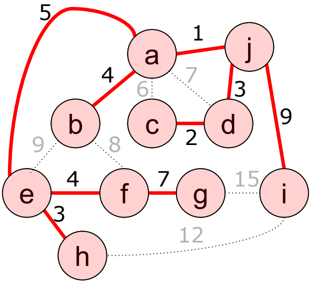
Visited = {a, j, d, c, b, e, h, f, g, i}



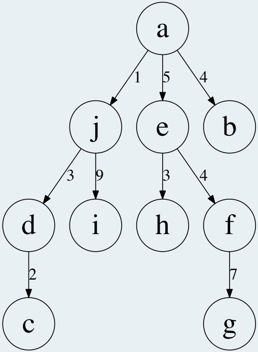
a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Prim: Minimum Spanning Tree

Visited = {a, j, d, c, b, e, h, f, g, i}



MST from Prim



Prim's Algorithm Pseudocode

```
function Prim:
Input:  a connected undirected weighted graph G
Output:  T a minimum spanning tree based on G
T ← tree with all nodes from G but no edge
visited ← {random node}
while visited does not contain all nodes from G do
    minimum ← random edge (m,n) with m in visited and n not in visited
                                                     and no cycle
    for all node n not in visited reachable from a node m in visited do
        if weight of edge (m, n) < weight of minimum and no cycle
            minimum ← edge (m, n)
        endif
    endfor
    add n in minimum to visited
    add edge minimum to T
endwhile
return T
```

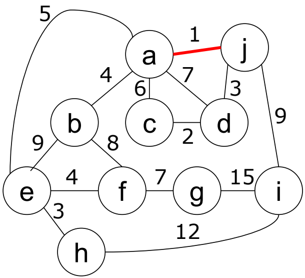
Kruskal's Algorithm

Inputs and outputs (same as Prim)

We have a graph: a weighted, undirected graph.
We want to create a tree: A minimum spanning tree (but it is a tree).

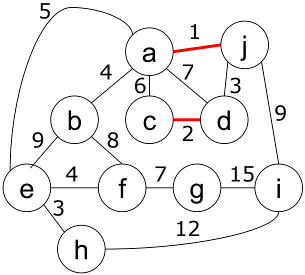
1. Create a set containing all the edges in the graph
2. Remove edge with minimum weight from the set
3. If the edge connects two different trees (remember a tree can be a single node) then add it to span, but not if it connects back to the its own tree (cycling)
4. Repeat from 2 until all all edges are removed and MST is not formed

Kruskal's Algorithm



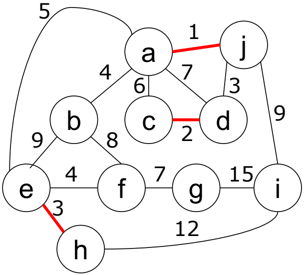
a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Kruskal's Algorithm



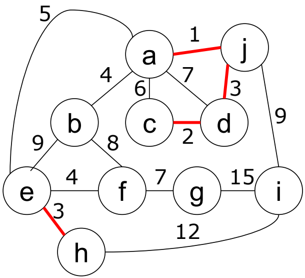
a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Kruskal's Algorithm



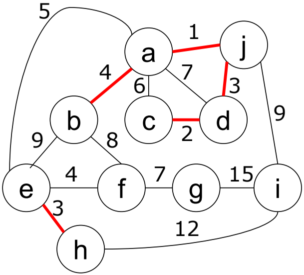
a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Kruskal's Algorithm



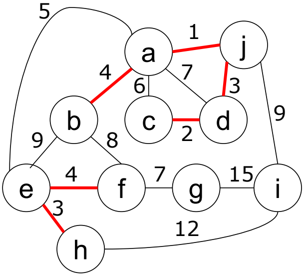
a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Kruskal's Algorithm



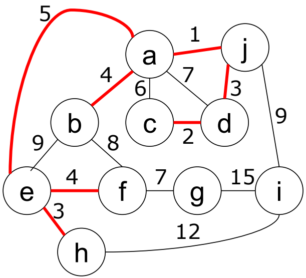
a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Kruskal's Algorithm



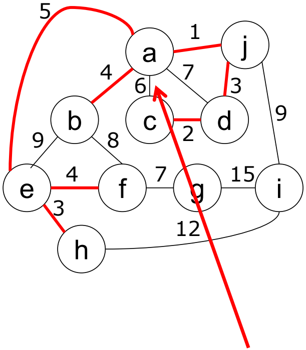
a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Kruskal's Algorithm



a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

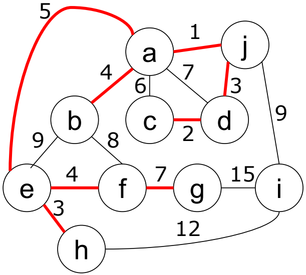
Kruskal's Algorithm



Don't take:
cycle

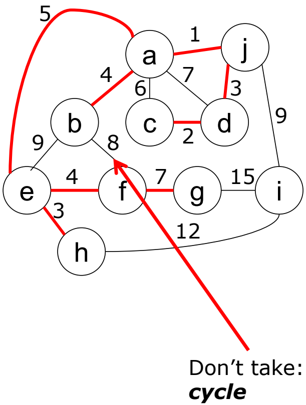
a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Kruskal's Algorithm



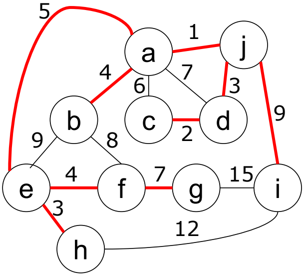
a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Kruskal's Algorithm



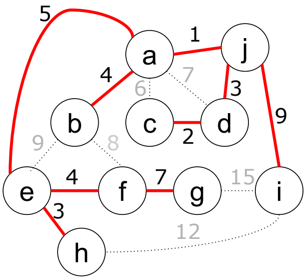
a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Kruskal's Algorithm



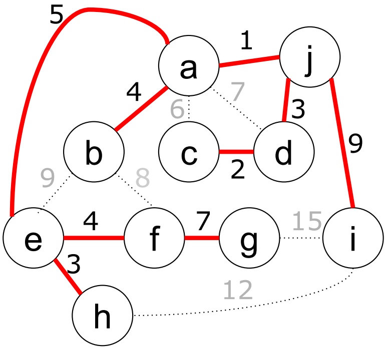
a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Kruskal's Algorithm

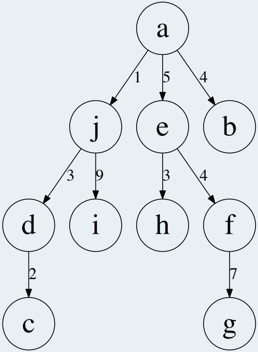


a	b(4), c(6), d(7), e(5), j(1)
b	a(4), e(9), f(8)
c	a(6), d(2)
d	a(7), c(2), j(3)
e	a(5), b(9), f(4), h(3)
f	b(8), e(4), g(7)
g	f(7), i(15)
h	e(3), i(12)
i	g(15), h(12), j(9)
j	a(1), d(3), i(9)

Kruskal: Minimum Spanning Tree



MST from Kruskal



Different Method Same Result

Same MST as Prim

Kruskal's Algorithm Pseudocode

```
function Kruskal:
Input:  a connected undirected weighted graph G
Output:  T a minimum spanning tree based on G
T ← tree with all nodes from G but no edge
while T is not connected do
    chosen ← random edge from G not in T and not creating cycle
    for each edge e in G do
        if e is not in T and does not create a cycle in T
                                and weight e < weight chosen then
            chosen ← e
        endif
endfor
    add chosen to T
endwhile
return T
```

Summary

Complexity

Search Algorithm	Basic	Optimised
Dijkstra	$\mathcal{O}(\lvert V\rvert^2)$	$\mathcal{O}(\lvert E\rvert + \lvert V\rvert \log \lvert V\rvert)$
Prim	$\mathcal{O}(\lvert V\rvert^2)$	$\mathcal{O}(\lvert E\rvert + \lvert V\rvert log \lvert V\rvert)$ or $\mathcal{O}(\lvert E\rvert log \lvert V\rvert)$
Kruskal	$\mathcal{O}(\lvert E\rvert log \lvert E\rvert)$	$\mathcal{O}(\lvert E\rvert a(\lvert V\rvert))^*$

Weighted, undirected graphs

Can better represent and model some scenarios.
Dijkstra's algorithm is a shortest path algorithm to traverse weighted edges.
Prim and Kruskal are two algorithms to find the minimum spanning tree for a graph

* You don't need to know this but if you are interested: In computability theory, the Ackermann function, named after Wilhelm Ackermann, is one of the simplest and earliest-discovered examples of a total computable function that is not primitive recursive. All primitive recursive functions are total and computable, but the Ackermann function illustrates that not all total computable functions are primitive recursive. `https://en.wikipedia.org/wiki/Ackermann_function`

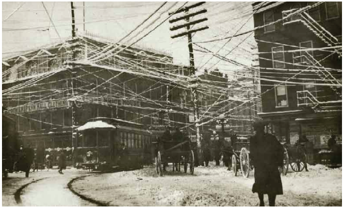
MSTs in the real world

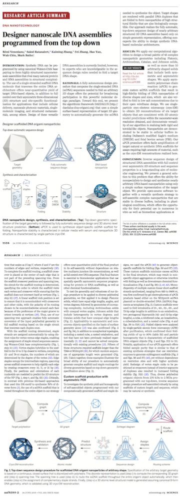
Real Minimum Spanning Trees

Electrical networks / telephone networks – minimise wiring

Tour operations – visiting the sites of a city

Nanoscale DNA assembly





RESEARCH ARTICLE SUMMARY

DNA NANOTECHNOLOGY

Designer nanoscale DNA assemblies programmed from the top down

ScienceMag, 2016: http://dx.doi.org/10.1126/science.aaf4388