Graph Theory

zyBooks Chapters: 11, 12.6

Logistics

*** Complete the Course Evaluation Survey ASAP ***

HW10 – Due: Friday, July 24

Lab 2 – Due: Sunday, July 26 (http://lin-res09.csc.ncsu.edu:1998/)

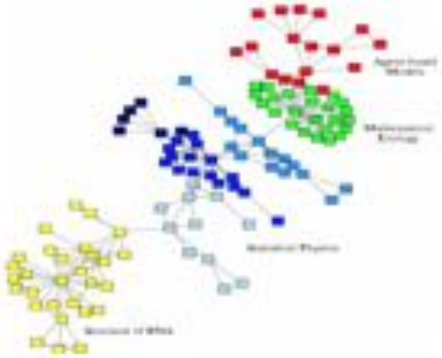
Q&A Sessions

– Tuesday, 07/21: Lecture 0 – 7 (Midterm 1)

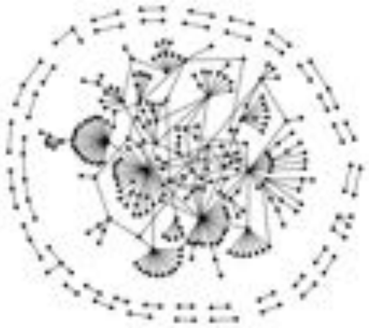
– Wednesday, 07/22: Lecture 8 – 16 (Midterm 2)

– Thursday, 07/23: Lecture 17 – 22 (Midterm 3)

Social networks



Economic networks



Biomedical networks

Information networks:
Web & citations

Internet

Networks of neurons

Why Graphs?

Graph

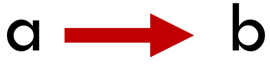
$G = (V, E)$

– a set of vertices V

– a set of edges E

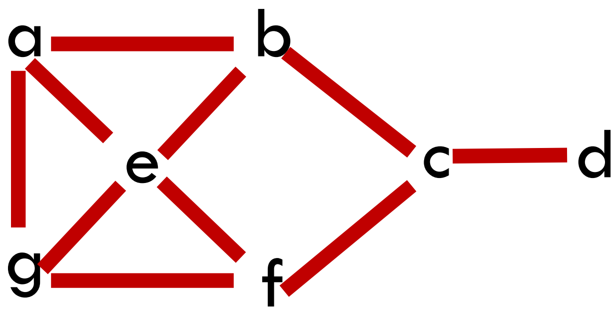
Undirected

Directed

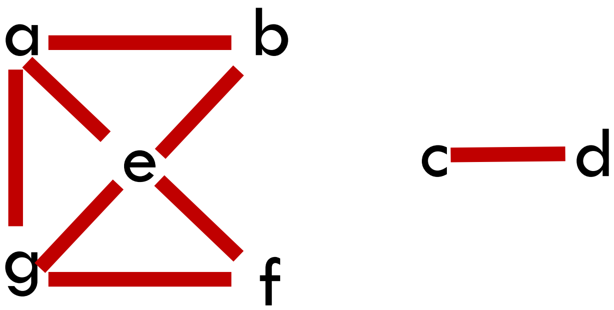


Weighted

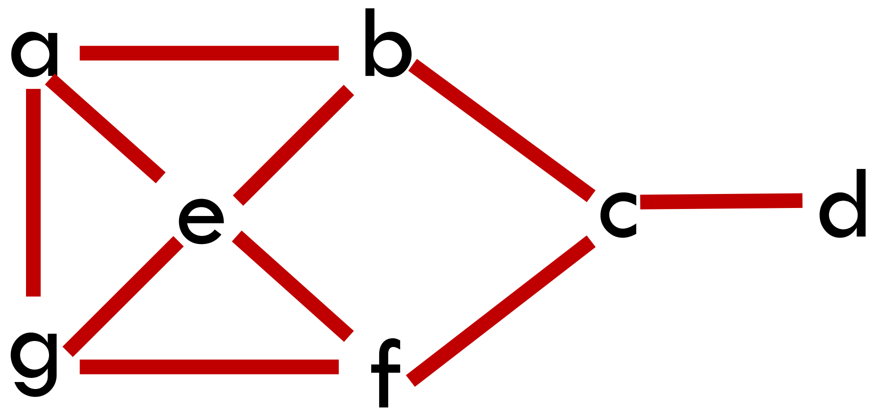
Connected



Disconnected



Graph



Vertices

- 7 vertices

Edges

- 10 edges

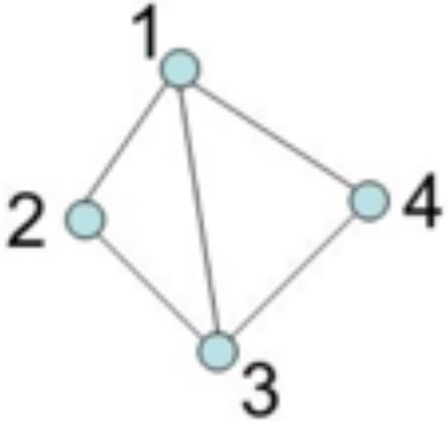
Degree

- Vertex a is degree 3
- Vertex e is degree 4

Adjacent

- Vertex a is adjacent to vertices g, e, b
- Vertex b is NOT adjacent to vertices g, f, d

Graph Representation (unweighted)



	1	2	3	4
1	0	1	1	1
2	1	0	1	0
3	1	1	0	1
4	1	0	1	0

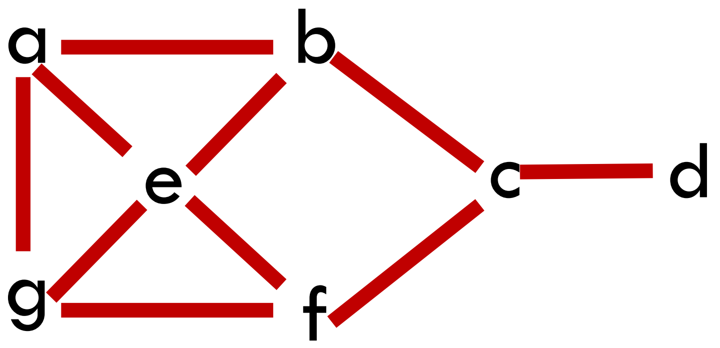
Adjacency matrix

1 →	2	3	4
2 →	1	3	
3 →	1	2	4
4 →	1	3	

Adjacency list

Terminologies

Walk	A sequence of vertices connected by edges
Trail	A walk with no repeated edges
	e.g., { (a, e) (e, f) (f, g) (g, e) (e, b) }
Path	A trail with no repeated vertices
	e.g., { (a, g) (g, f) (f, c) (c, d) }
Circuit	A walk that begins and ends with the same vertex
	e.g., { (a, e) (e, g) (g, f) (f, e) (e, b) (b, a) }
Cycle	A circuit with no repeated vertices
	e.g., { (a, g) (g, f) (f, c) (c, b) (b, a) }



Classic Graphs

Euler Trail/Circuit

Hamiltonian Path/Cycle

Minimum Spanning Tree – Weighted

Euler Trail/Circuit

Euler trail – a trail that uses every edge of graph G exactly once

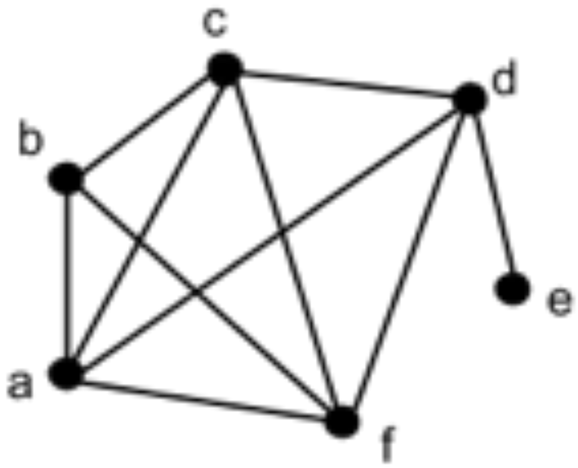
Euler circuit – an Euler trail which starts and stops at the same vertex

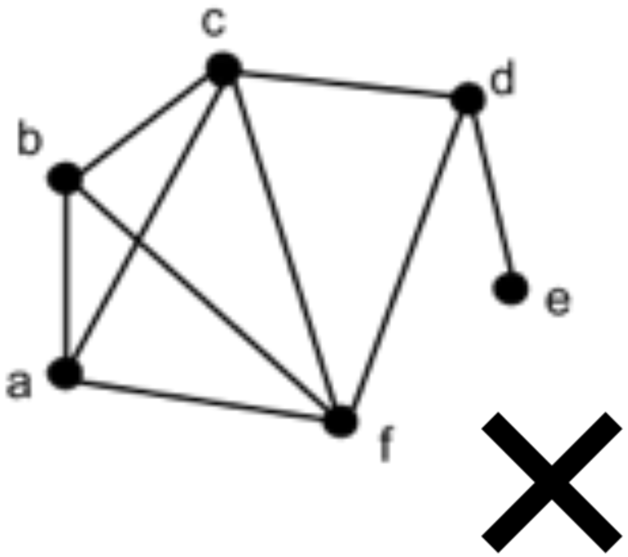
Euler Trail/Circuit – General Rules

A graph G has an Euler trail if and only if there are exactly 2 vertices with odd degree.

A graph G has an Euler circuit if and only if all vertices have even degree.

Euler Trail/Circuit





Euler trail: ⟨ e, d, f, a, d, c, a, b, f, c, b⟩.

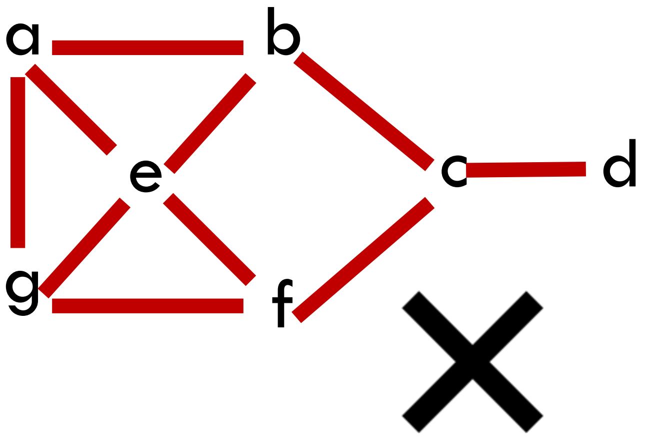
Starting from vertex with odd degree,
if none, pick a random one

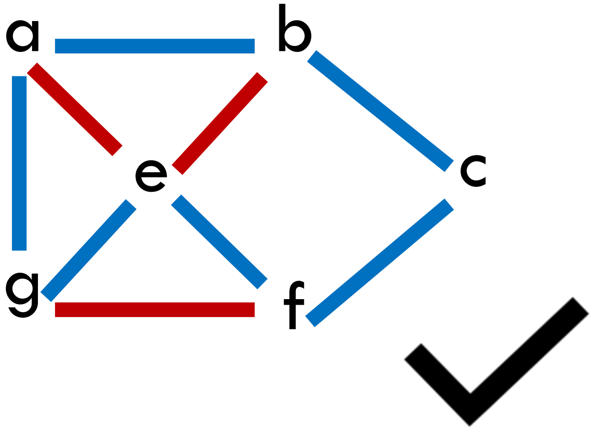
4 vertices with odd degree: a, b, d, e.

Hamiltonian Path/Cycle

Hamiltonian Path – a path that that visits all vertices of graph G exactly once

Hamiltonian Cycle – A cycle that visits all vertices of graph G exactly once





Hamiltonian Cycle – General Rules

If graph G has vertices with degree 1, then there exists no Hamiltonian Cycle in G

If graph G has all vertices of degree $\geq \frac{|V|}{2}$, then there exists an Hamiltonian Cycle in G

Spanning Tree

Tree – A connected acyclic graph

Spanning Tree – A subset of edges of graph G that form a tree and contains all vertices of G

A complete undirected graph can have maximum n^{n-2} number of spanning trees, where **n** is the number of nodes.

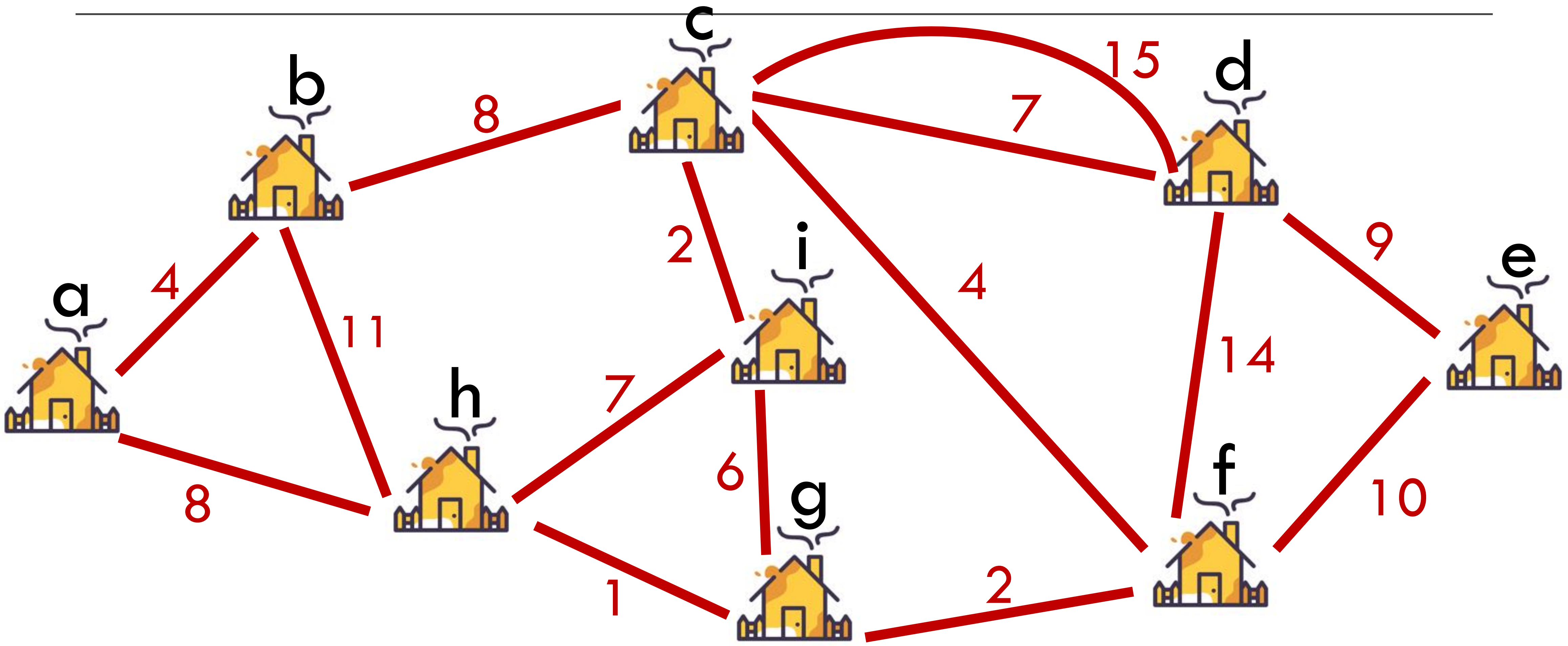
Minimum Spanning Tree

Minimum? For What?

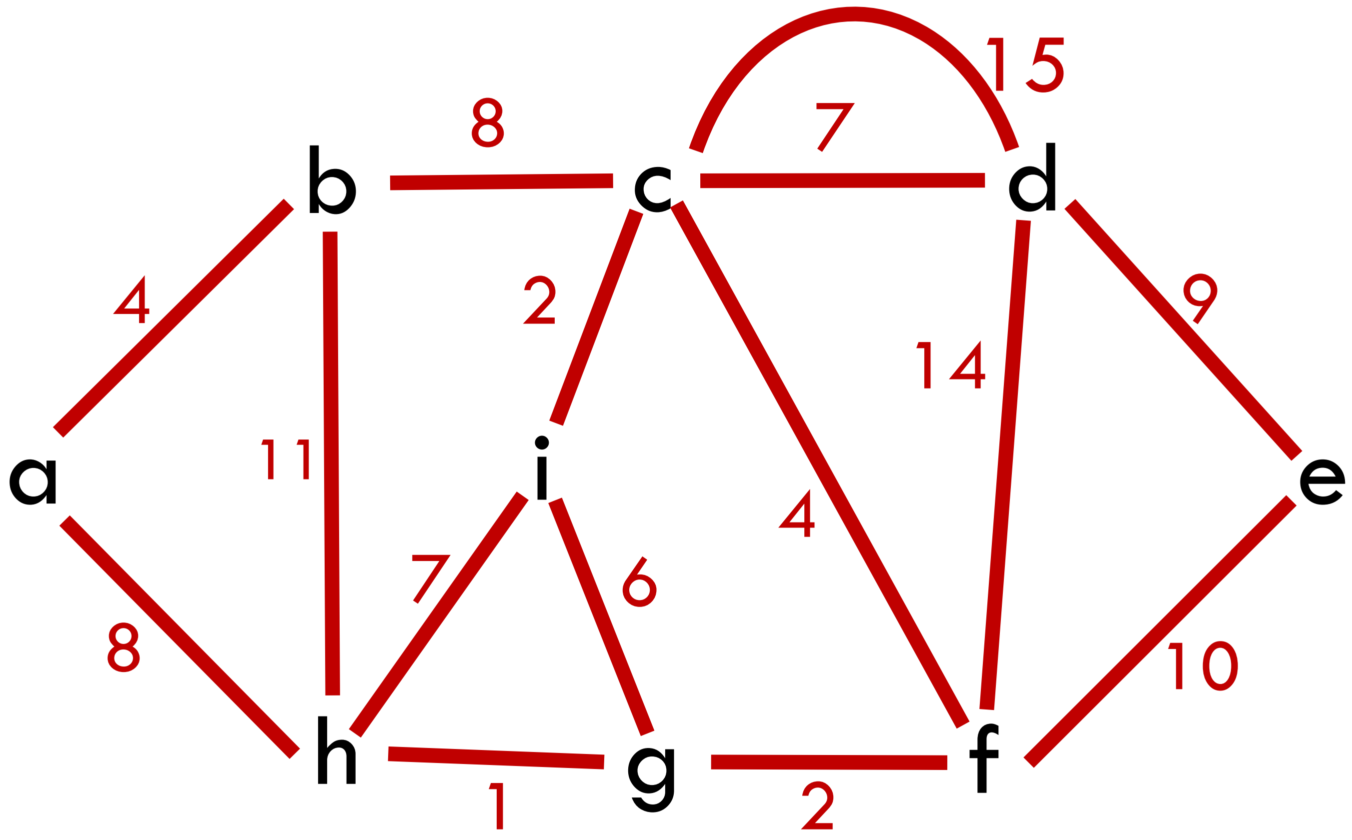
Weight (e.g., cost, distance, congestion…)

A minimum spanning tree is a spanning tree that has minimum weight than all other spanning trees of the same graph.

Road Renovation Challenge



Edge-weighted Graph



Kruskal’s Algorithm

Kruskal's algorithm is a greedy algorithm that takes a graph G as input and finds the subset of the edges of G which

1. form a tree that includes every vertex
2. has the minimum sum of weights among all the trees that can be formed from the graph

Kruskal’s Algorithm

Step 1 – Arrange all edges in increasing order of their weights

Step 2 – Pick the least weighted edge (u,v).

Check if it forms a cycle within the spanning tree formed so far.

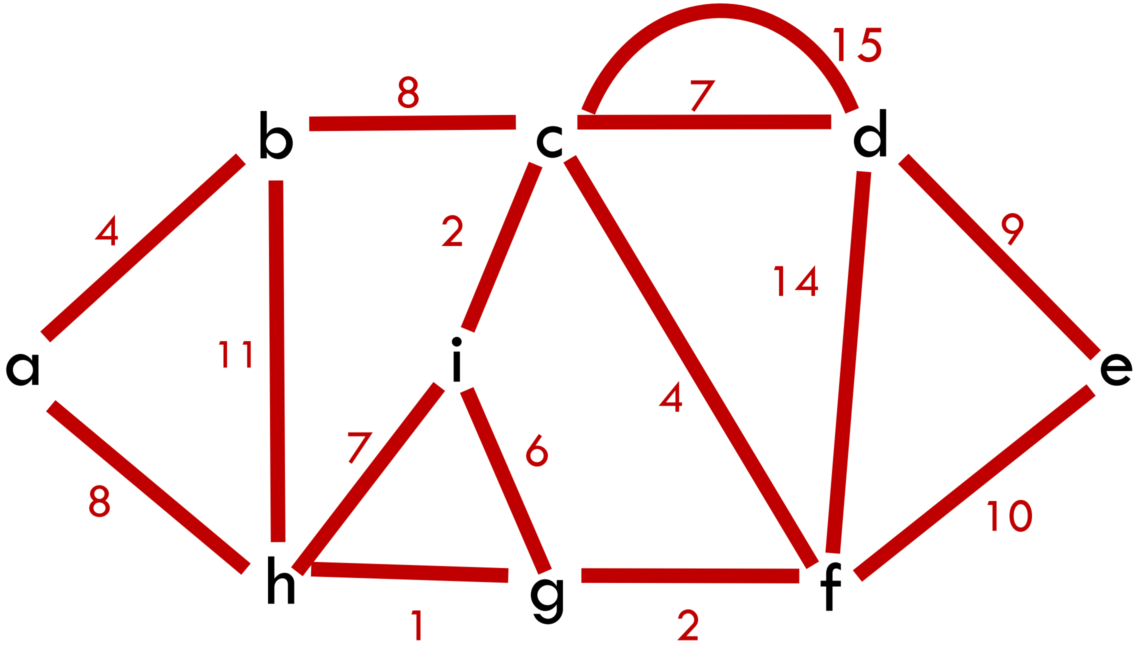
If cycle is not formed, include this edge.

Else, discard it.

Step 3 – Repeat Step 2 until there are no more valid edges.

Kruskal’s Algorithm

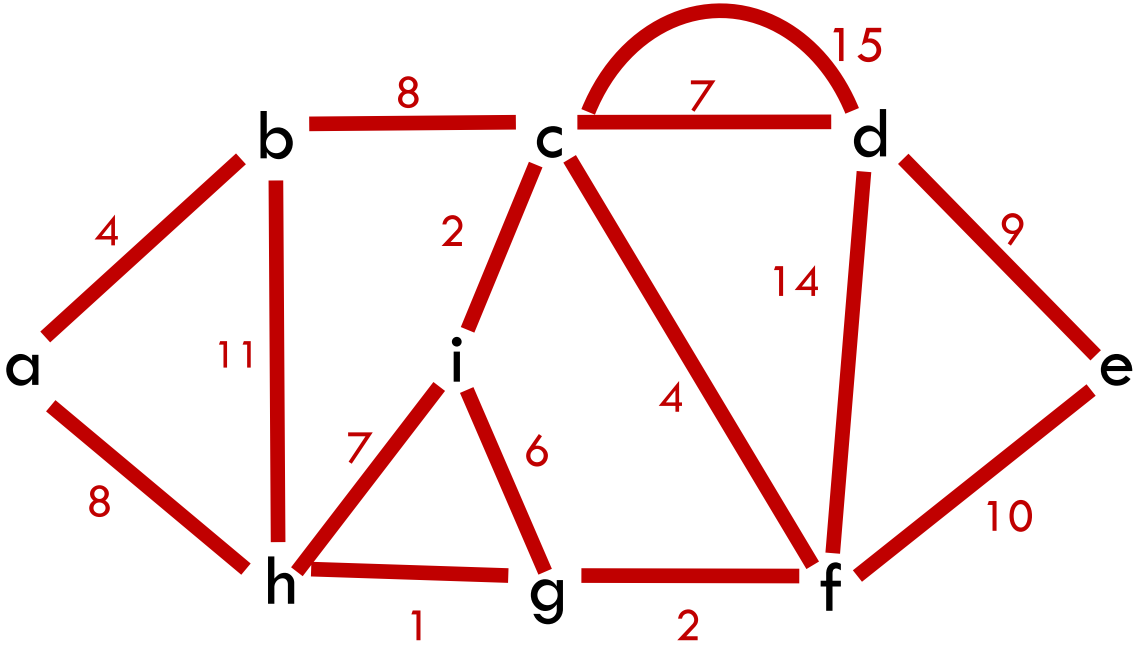
Step 1 – Arrange all edges in increasing order of their weight



(g,h)
1

Kruskal’s Algorithm

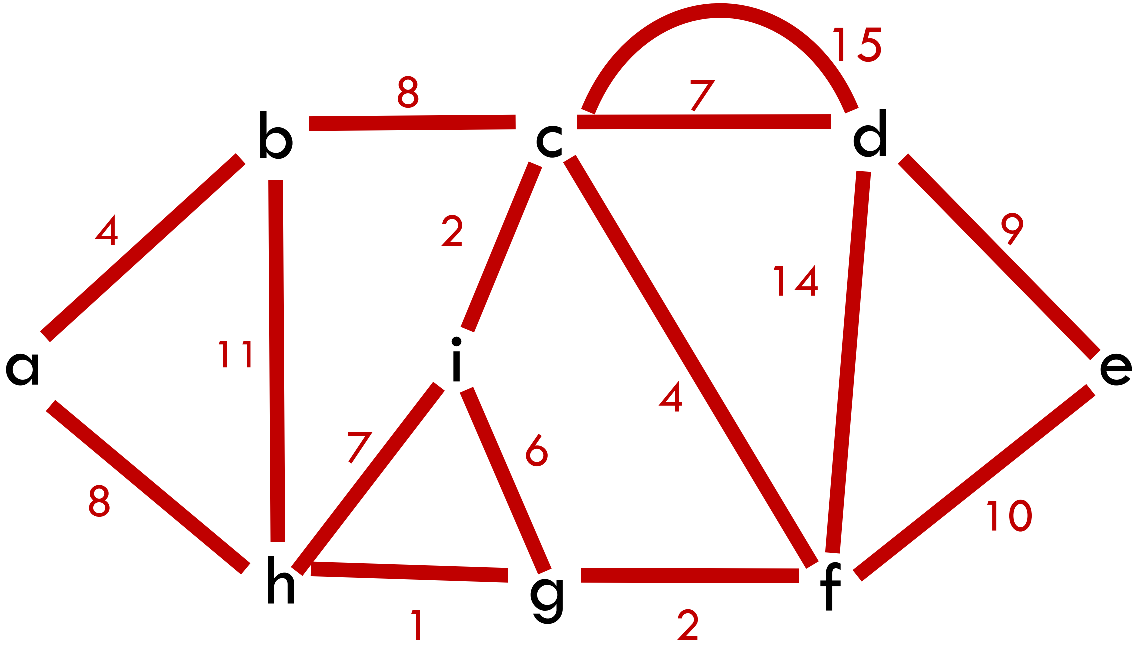
Step 1 – Arrange all edges in increasing order of their weight



(g,h)	(c,i)	(f,g)
1	2	2

Kruskal's Algorithm

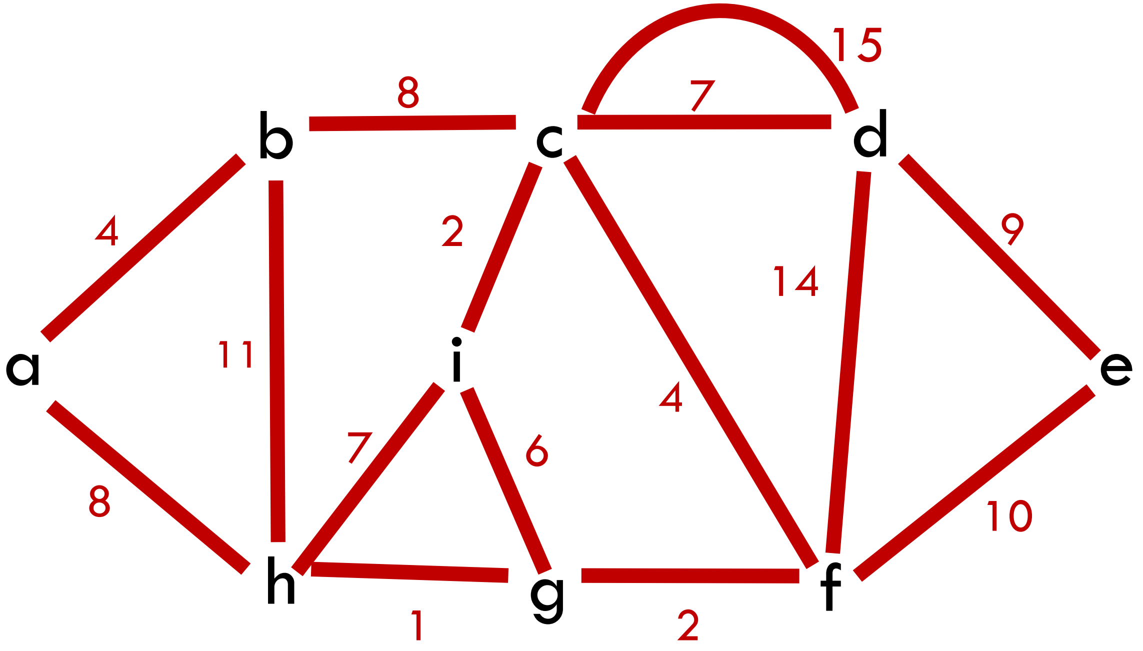
Step 1 – Arrange all edges in increasing order of their weight



(g,h)	(c,i)	(f,g)	(a,b)	(c,f)
1	2	2	4	4

Kruskal's Algorithm

Step 1 – Arrange all edges in increasing order of their weight



(g,h)	(c,i)	(f,g)	(a,b)	(c,f)	(i,g)	(c,d)	(i,h)	(a,h)	(b,c)	(d,e)	(e,f)	(b,h)	(d,f)	(c,d)
1	2	2	4	4	6	7	7	8	8	9	10	11	14	15

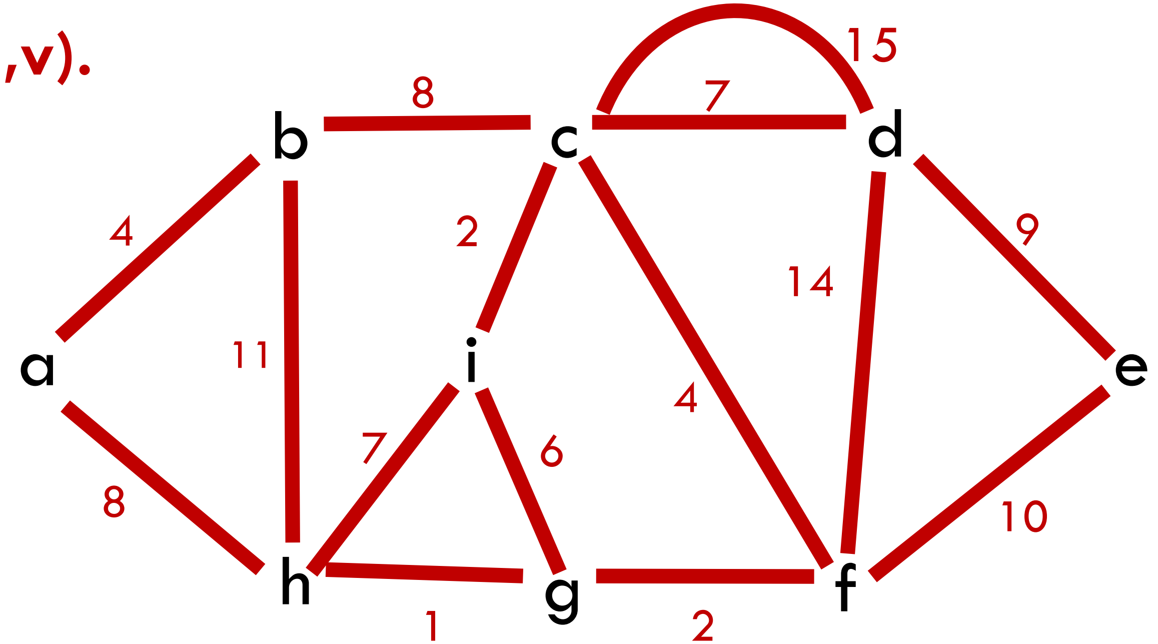
Kruskal's Algorithm

Step 2 – Pick the least weighted edge (u,v).

Forms a cycle if being added?

No → add (u,v)

Yes → discard



(g,h)	(c,i)	(f,g)	(a,b)	(c,f)	(i,g)	(c,d)	(i,h)	(a,h)	(b,c)	(d,e)	(e,f)	(b,h)	(d,f)	(c,d)
1	2	2	4	4	6	7	7	8	8	9	10	11	14	15

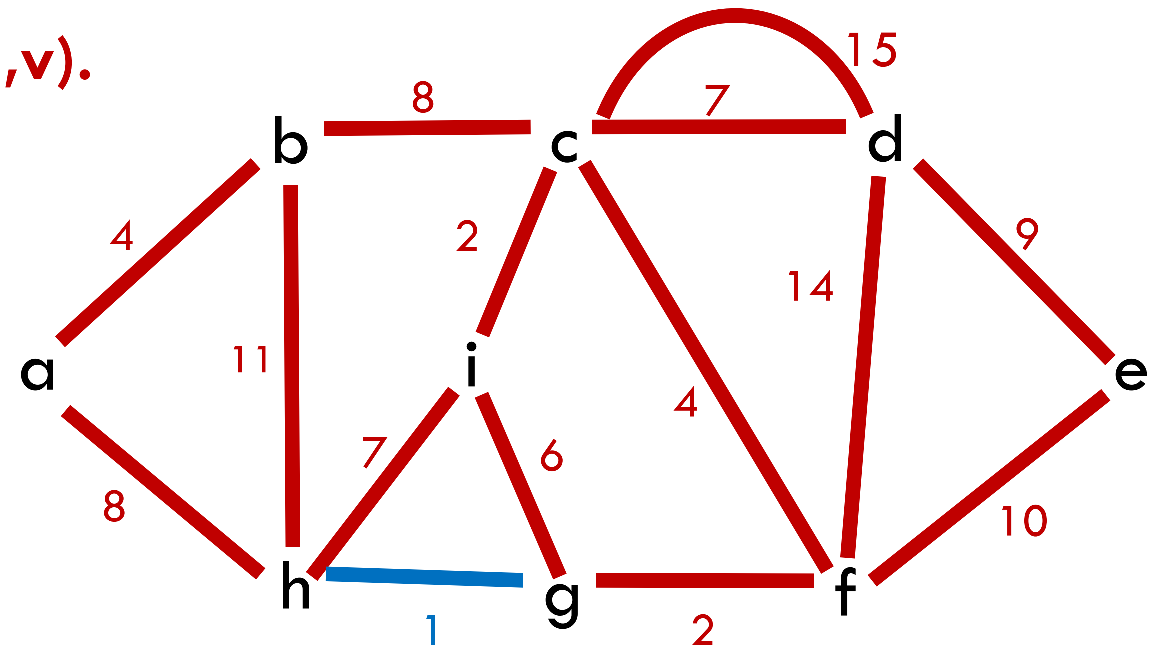
Kruskal's Algorithm

Step 2 – Pick the least weighted edge (u,v).

Forms a cycle if being added?

No → add (u,v)

Yes → discard

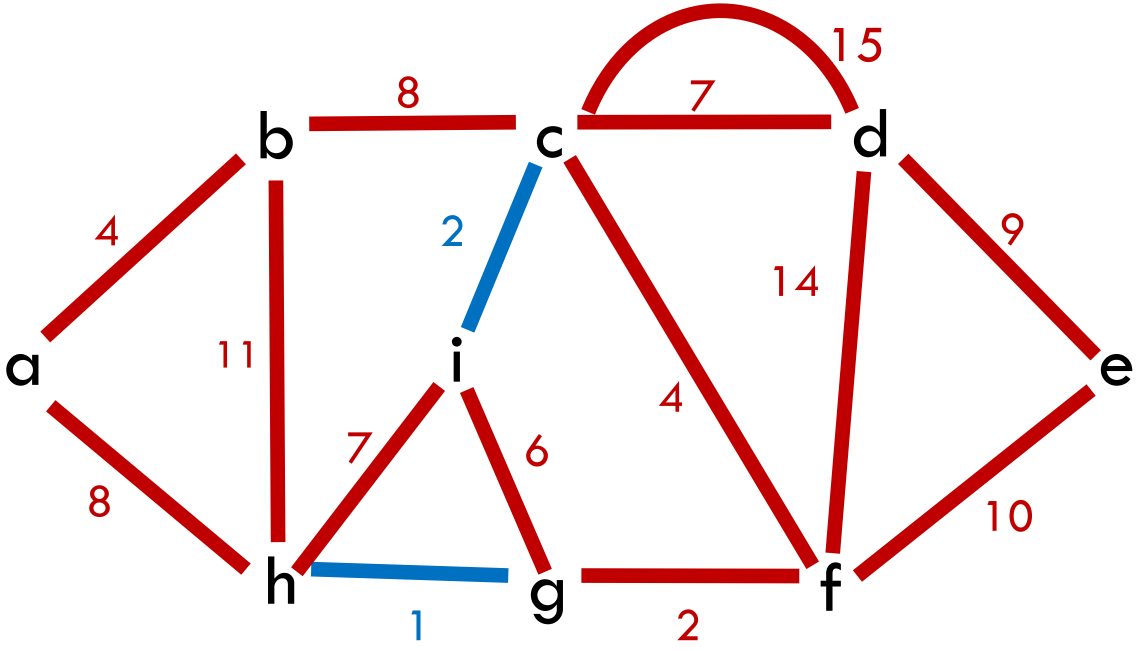


(g,h)	(c,i)	(f,g)	(a,b)	(c,f)	(i,g)	(c,d)	(i,h)	(a,h)	(b,c)	(d,e)	(e,f)	(b,h)	(d,f)	(c,d)
1	2	2	4	4	6	7	7	8	8	9	10	11	14	15

Kruskal’s Algorithm

Step 3 – Repeat Step 2

Pick → Check → Add/Discard

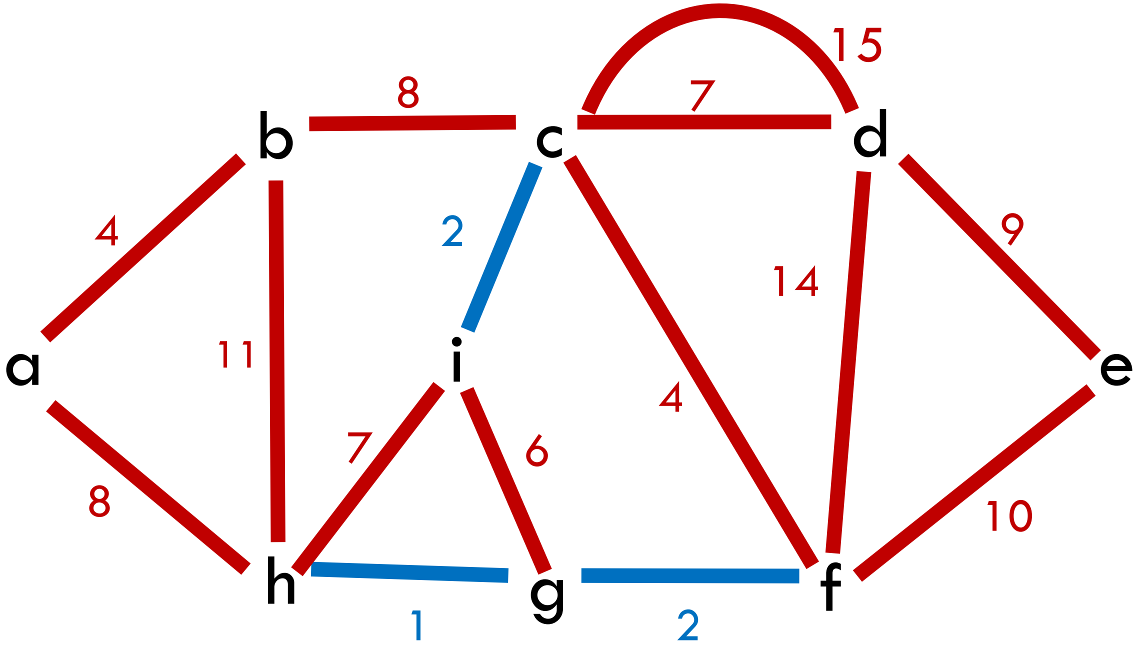


(g,h)	(c,i)	(f,g)	(a,b)	(c,f)	(i,g)	(c,d)	(i,h)	(a,h)	(b,c)	(d,e)	(e,f)	(b,h)	(d,f)	(c,d)
1	2	2	4	4	6	7	7	8	8	9	10	11	14	15

Kruskal’s Algorithm

Step 3 – Repeat Step 2

Pick → Check → Add/Discard

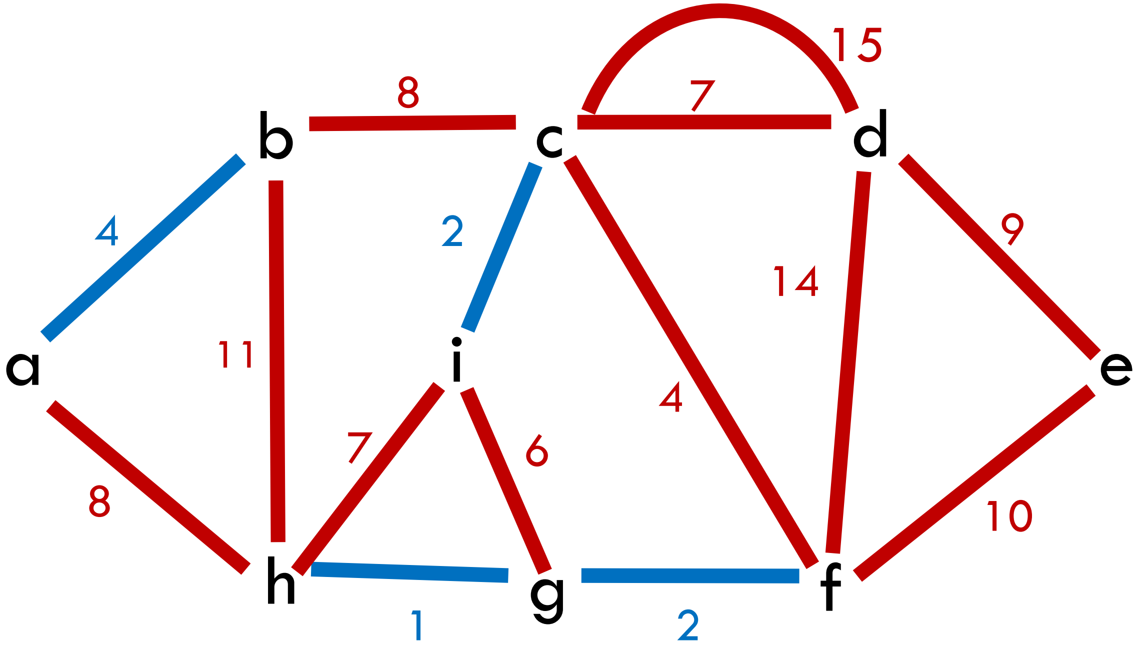


(g,h)	(c,i)	(f,g)	(a,b)	(c,f)	(i,g)	(c,d)	(i,h)	(a,h)	(b,c)	(d,e)	(e,f)	(b,h)	(d,f)	(c,d)
1	2	2	4	4	6	7	7	8	8	9	10	11	14	15

Kruskal’s Algorithm

Step 3 – Repeat Step 2

Pick → Check → Add/Discard

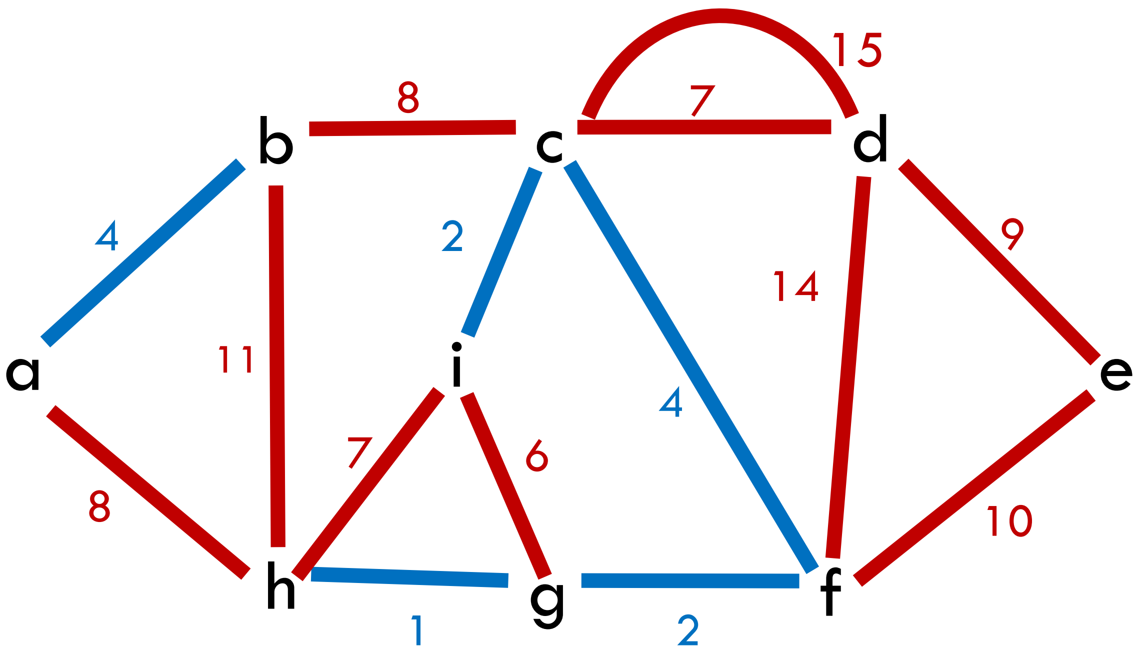


(g,h)	(c,i)	(f,g)	(a,b)	(c,f)	(i,g)	(c,d)	(i,h)	(a,h)	(b,c)	(d,e)	(e,f)	(b,h)	(d,f)	(c,d)
1	2	2	4	4	6	7	7	8	8	9	10	11	14	15

Kruskal’s Algorithm

Step 3 – Repeat Step 2

Pick → Check → Add/Discard



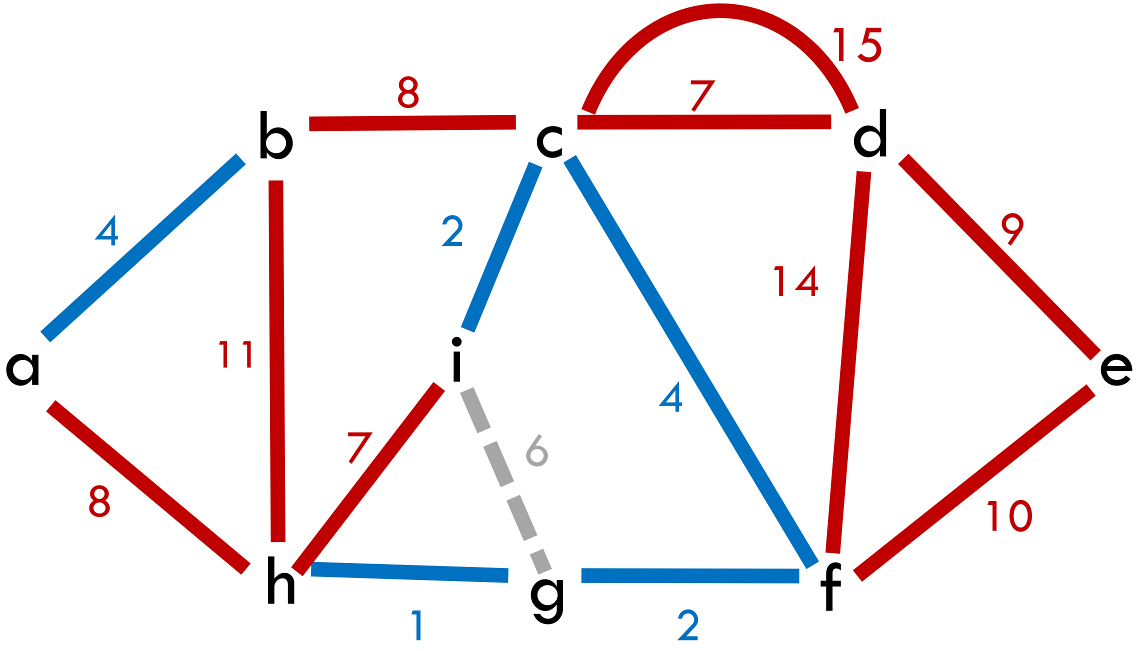
(g,h)	(c,i)	(f,g)	(a,b)	(c,f)	(i,g)	(c,d)	(i,h)	(a,h)	(b,c)	(d,e)	(e,f)	(b,h)	(d,f)	(c,d)
1	2	2	4	4	6	7	7	8	8	9	10	11	14	15

Kruskal's Algorithm

Step 3 – Repeat Step 2

Pick → Check → Add/Discard

Adding this edge will introduce a cycle

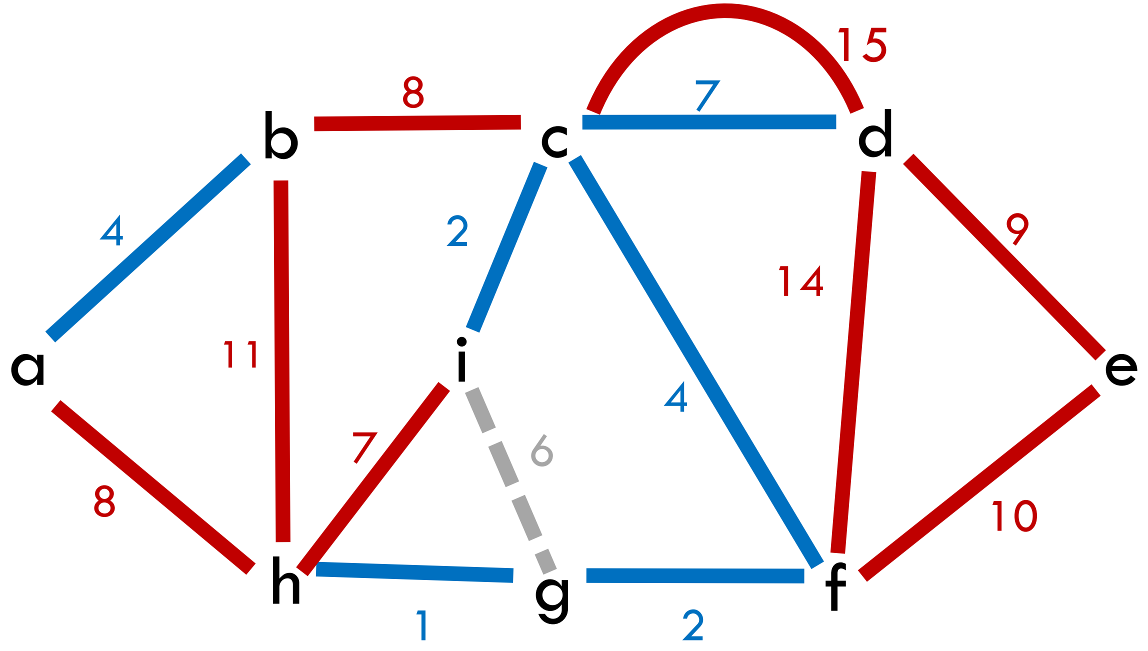


(g,h)	(c,i)	(f,g)	(a,b)	(c,f)	(i,g)	(c,d)	(i,h)	(a,h)	(b,c)	(d,e)	(e,f)	(b,h)	(d,f)	(c,d)
1	2	2	4	4	6	7	7	8	8	9	10	11	14	15

Kruskal's Algorithm

Step 3 – Repeat Step 2

Pick → Check → Add/Discard

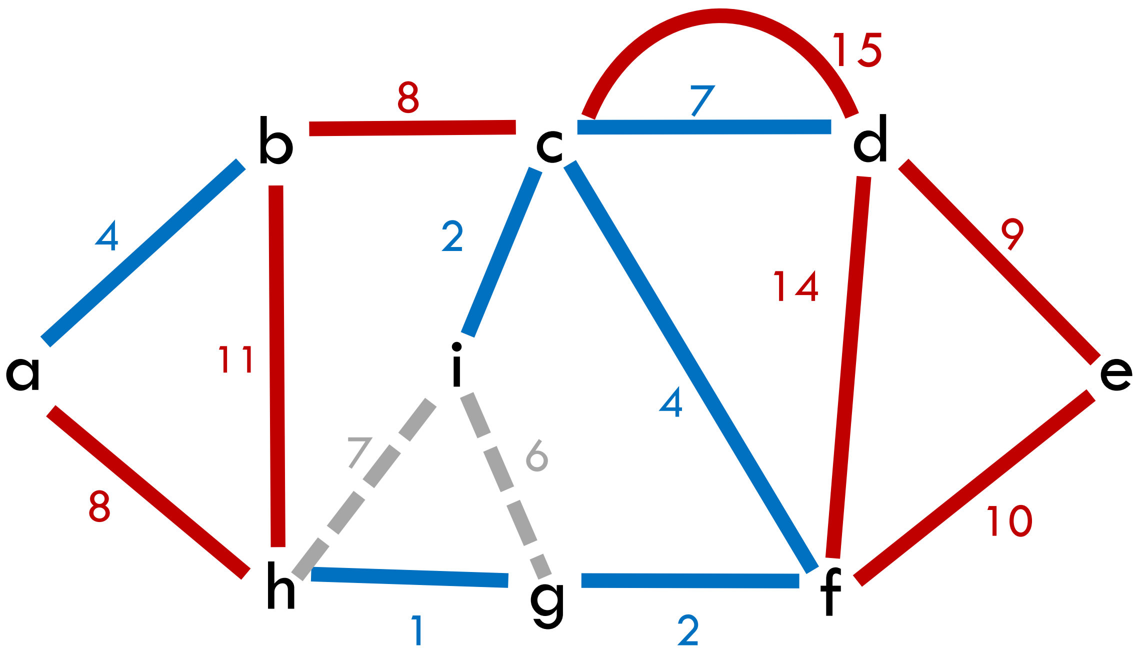


(g,h)	(c,i)	(f,g)	(a,b)	(c,f)	(i,g)	(c,d)	(i,h)	(a,h)	(b,c)	(d,e)	(e,f)	(b,h)	(d,f)	(c,d)
1	2	2	4	4	6	7	7	8	8	9	10	11	14	15

Kruskal's Algorithm

Step 3 – Repeat Step 2

Pick → Check → Add/Discard

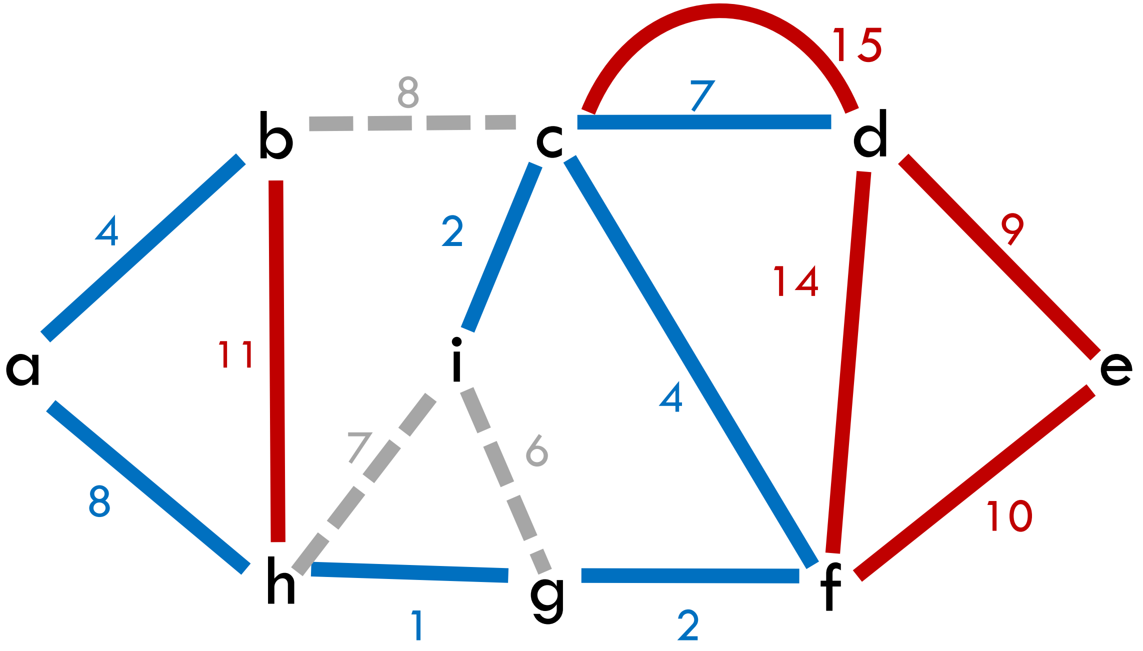


(g,h)	(c,i)	(f,g)	(a,b)	(c,f)	(i,g)	(c,d)	(i,h)	(a,h)	(b,c)	(d,e)	(e,f)	(b,h)	(d,f)	(c,d)
1	2	2	4	4	6	7	7	8	8	9	10	11	14	15

Kruskal's Algorithm

Step 3 – Repeat Step 2

Pick → Check → Add/Discard

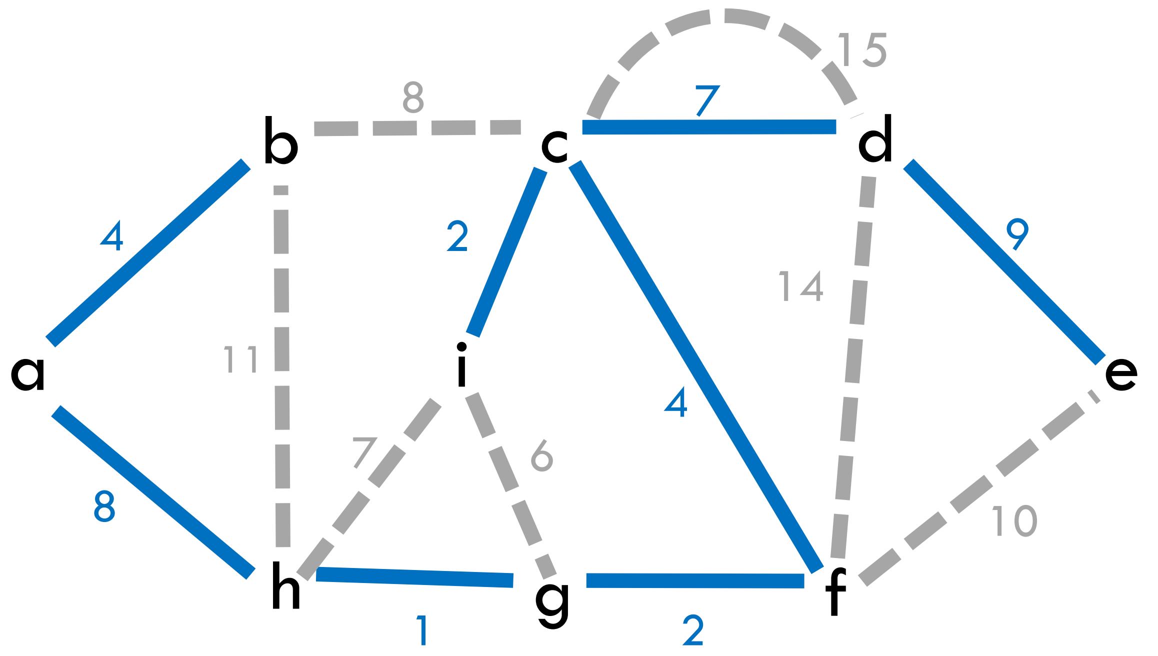


(g,h)	(c,i)	(f,g)	(a,b)	(c,f)	(i,g)	(c,d)	(i,h)	(a,h)	(b,c)	(d,e)	(e,f)	(b,h)	(d,f)	(c,d)
1	2	2	4	4	6	7	7	8	8	9	10	11	14	15

Kruskal’s Algorithm

Step 3 – Repeat Step 2

Pick → Check → Add/Discard



(g,h)	(c,i)	(f,g)	(a,b)	(c,f)	(i,g)	(c,d)	(i,h)	(a,h)	(b,c)	(d,e)	(e,f)	(b,h)	(d,f)	(c,d)
1	2	2	4	4	6	7	7	8	8	9	10	11	14	15

Kruskal's Algorithm – MST