Minimum Spanning Trees

Dr. Chung-Wen Albert Tsao

Minimum Spanning Trees (MST)

Definitions:

- A subgraph of an undirected graph that spans (includes) all nodes, is connected and has minimum total edge weight.

Prim's Algorithm

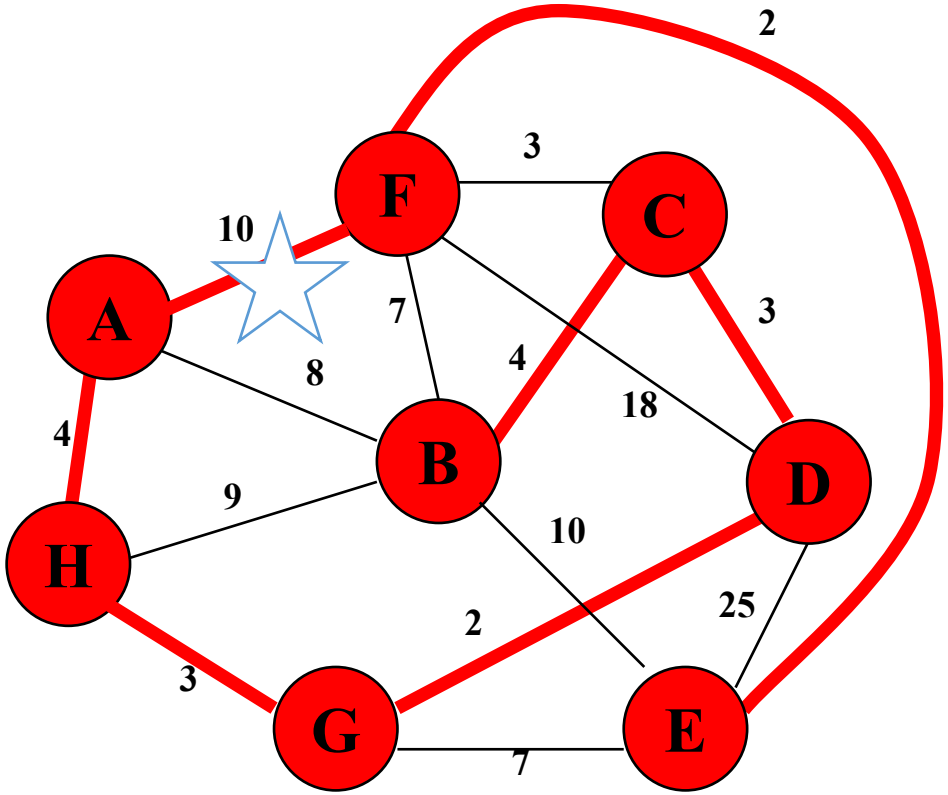
- Start with any vertex s and **greedily** grow a tree T from s.
- At each step, add the **cheapest** edge to T that has exactly one endpoint in T
- Similar to Dijkstra's Algorithm

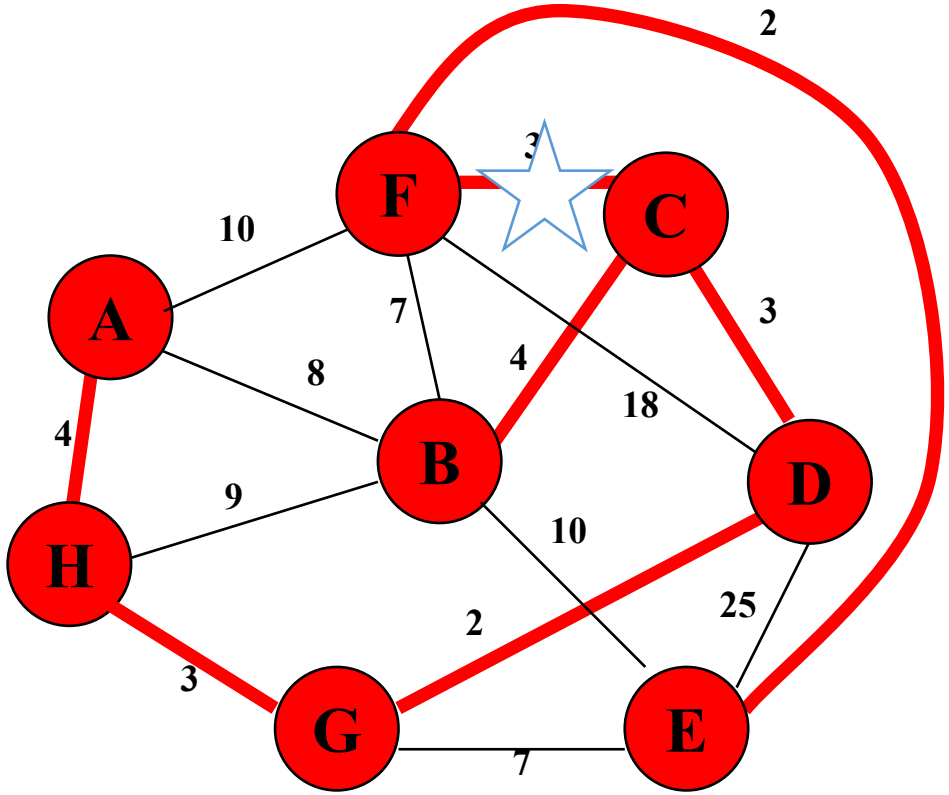
Kruskal's Algorithm

- Sort edges in ascending order of cost.
- Add the next edge to T unless doing so would create a cycle
- Focuses on edges, rather than nodes

Minimum Spanning Trees (MST)

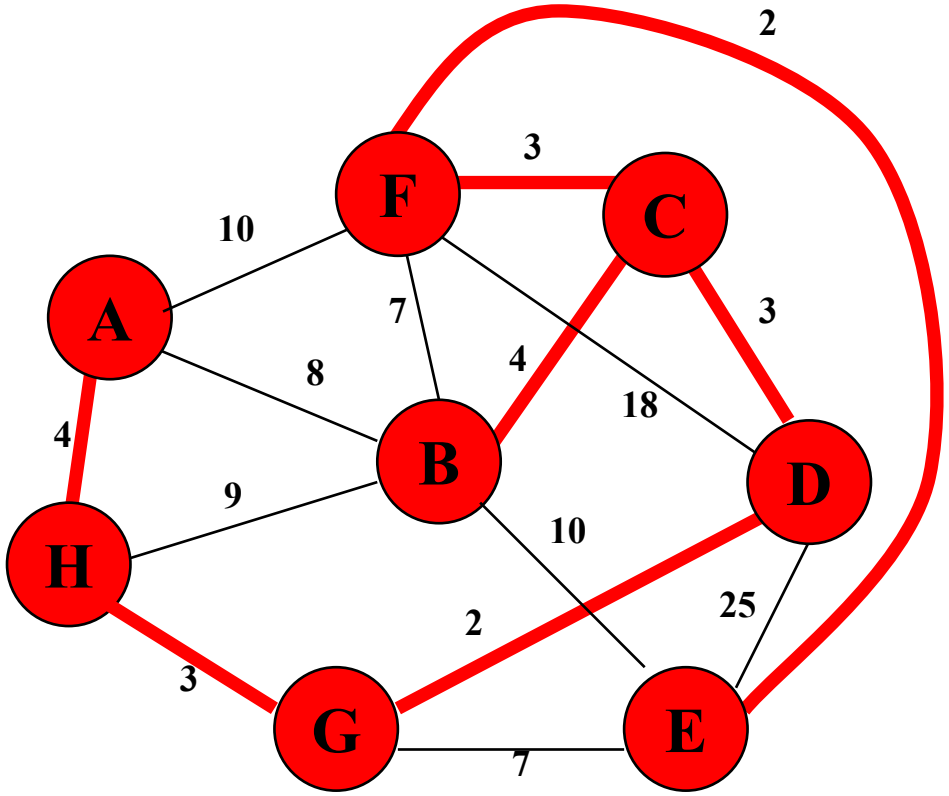
Given an undirected graph G, finding a spanning tree that includes all nodes and has minimum total edge weight.





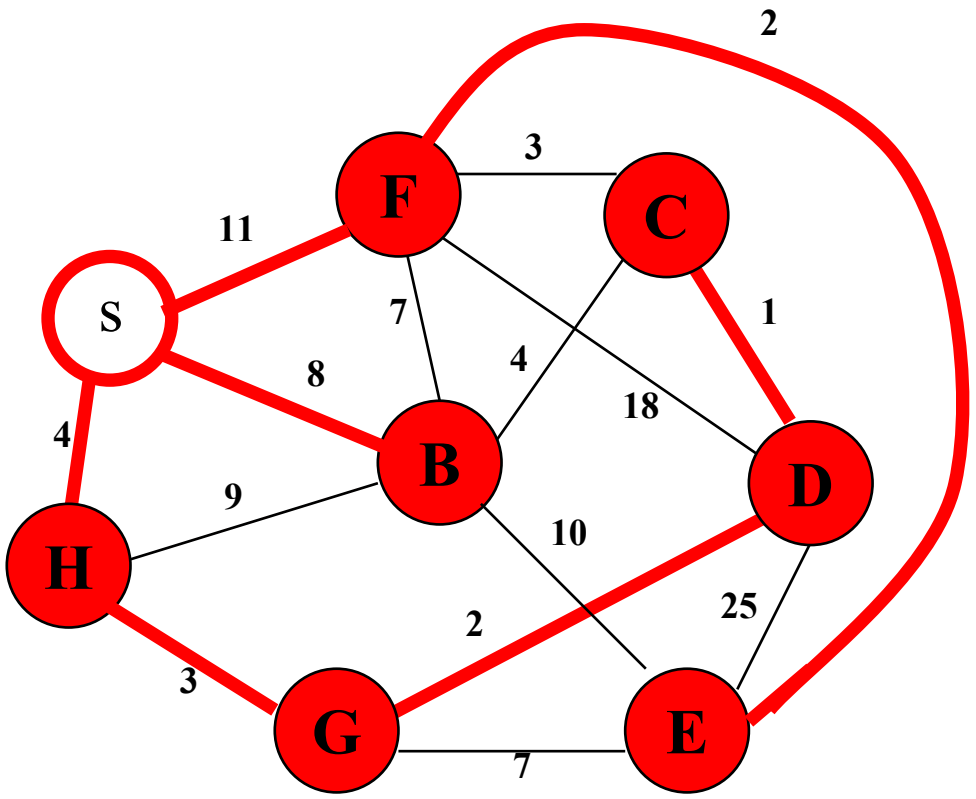
Minimum Spanning Trees (MST)

Given an undirected graph G, finding a spanning tree that includes all nodes and has minimum total edge weight.



Single-Source Shortest Path Problem

Finding shortest paths from a source vertex **s** to all other vertices in the graph.



Algorithm Characteristics

- Prim's Algorithm
  - Start with any vertex s and **greedily** grow a tree T from s.
  - At each step, add the cheapest edge to T that has exactly one endpoint in T.
- Kruskal's Algorithm
  - Sort edges in ascending order of cost.
  - Add the next edge to T unless doing so would create a cycle
  - Focuses on edges, rather than nodes
- Both work with undirected graphs
- Both are **greedy** algorithms that produce optimal solutions

Prim's Algorithm

- Start with any vertex s and greedily grow a tree T from s.
- Repeat the same process until no more nodes outside T
  - Among all nodes outside T, pick a node, say v, with the **cheapest edge to T** (denoted d_v)
  - Connect v to T

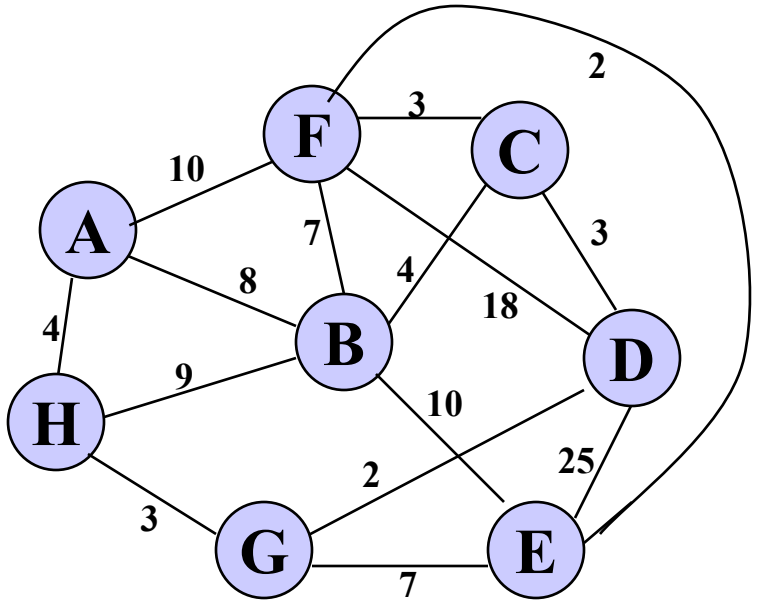
Dijkstra's Algorithm

- Start with any vertex s and greedily grow a tree T from s.
- Repeat the same process until no more nodes outside T
  - Among all nodes outside T, pick a node, say v, with the **shortest path length to s** (denoted d_v)
  - Connect v to T

Similar to Prim's Algorithm except that

- There is a source node s
- d_v records shortest path length to s, not the cheapest edge to T

Prim's Algorithm: Walk-Through



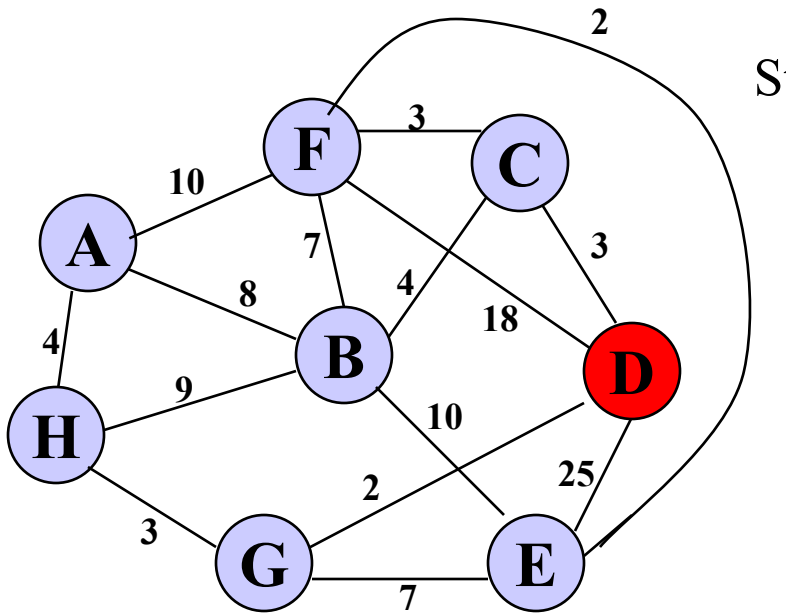
Initialize array

	K	d_v	p_v
A	F	∞	–
B	F	∞	–
C	F	∞	–
D	F	∞	–
E	F	∞	–
F	F	∞	–
G	F	∞	–
H	F	∞	–

d_v = Cheapest edge cost to T

p_v = Node in T to which the cheapest edge is connected

Prim's Algorithm: Walk-Through



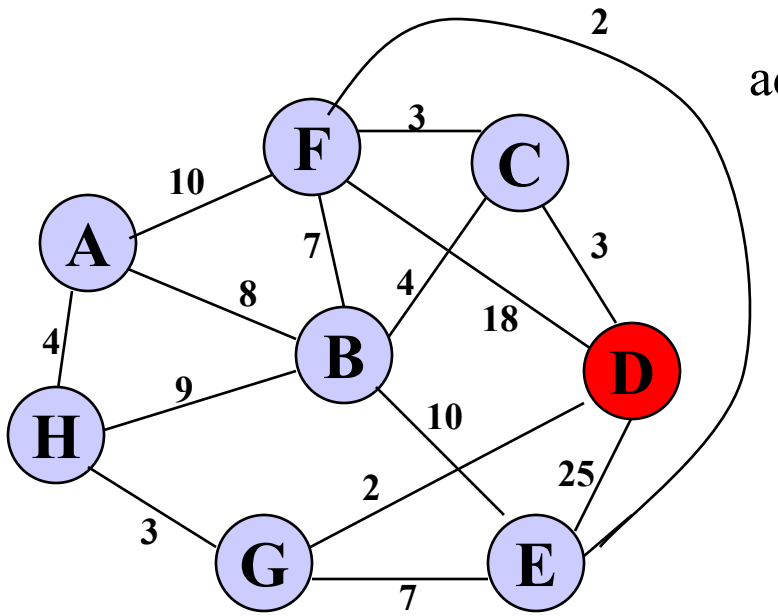
Start with any node, say D

	K	d_v	p_v
A			
B			
C			
D	T	0	–
E			
F			
G			
H			

d_v = Cheapest edge cost to T

p_v = Node in T to which the cheapest edge is connected

Prim's Algorithm: Walk-Through



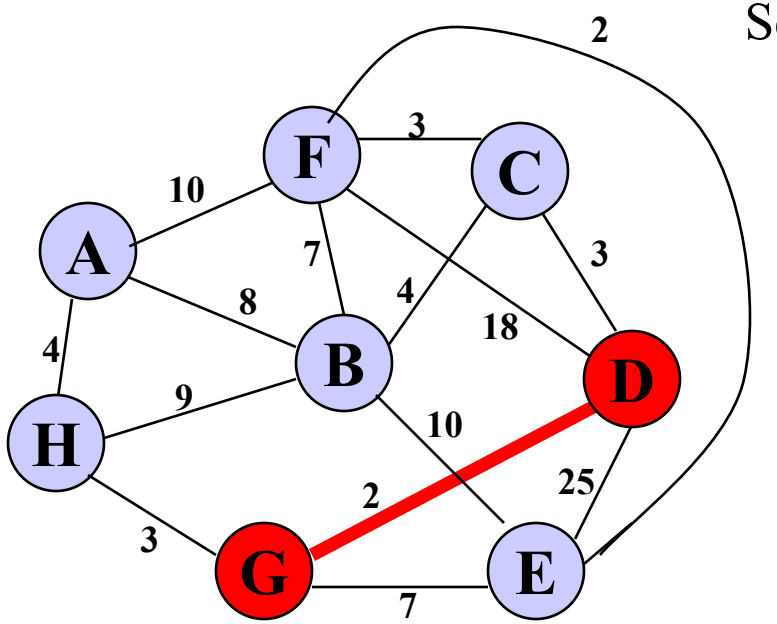
Update distances of adjacent, unselected nodes

	in tree	d_v	p_v
A			
B			
C		3	D
D	T	0	–
E		25	D
F		18	D
G		2	D
H			

d_v = Cheapest edge cost to T

p_v = Node in T to which the cheapest edge is connected

Prim's Algorithm: Walk-Through



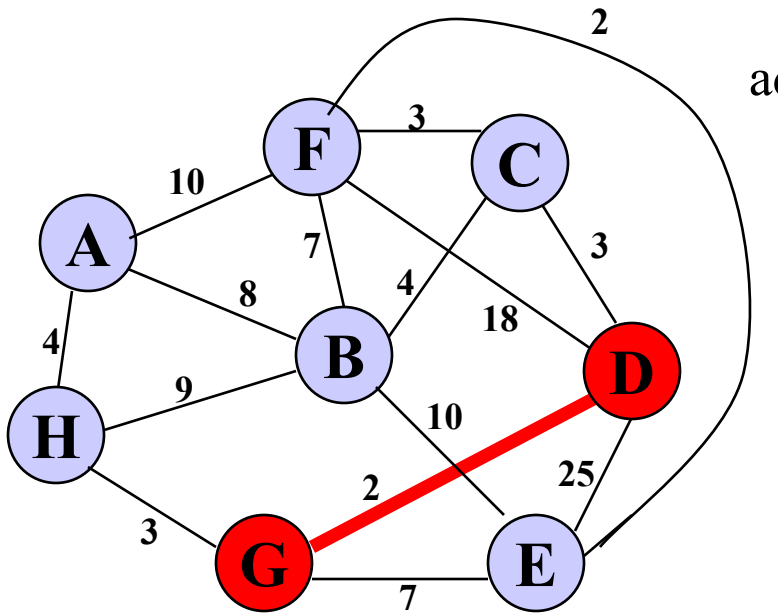
Select node with minimum distance

	K	d_v	p_v
A			
B			
C		3	D
D	T	0	–
E		25	D
F		18	D
G	T	2	D
H			

d_v = Cheapest edge cost to T

p_v = Node in T to which the cheapest edge is connected

Prim's Algorithm: Walk-Through



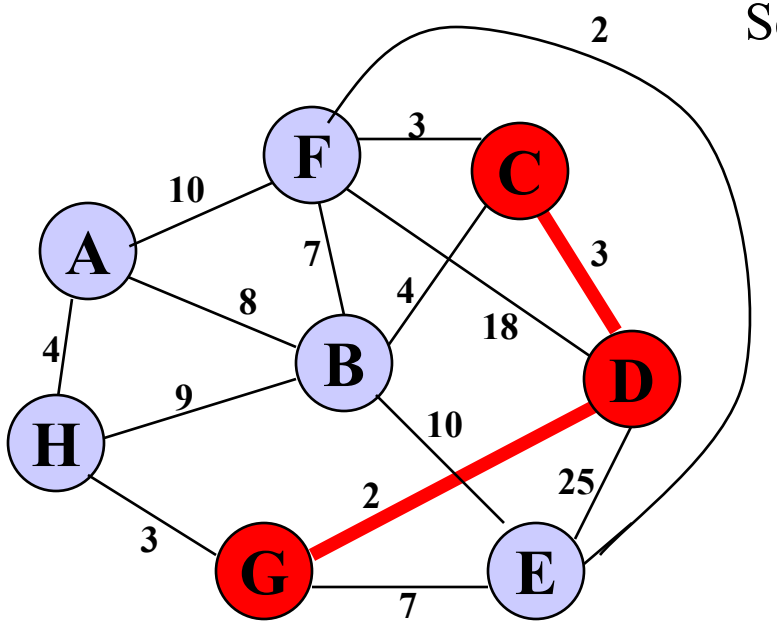
Update distances of adjacent, unselected nodes

	K	d_v	p_v
A			
B			
C		3	D
D	T	0	–
E		7	G
F		18	D
G	T	2	D
H		3	G

d_v = Cheapest edge cost to T

p_v = Node in T to which the cheapest edge is connected

Prim's Algorithm: Walk-Through



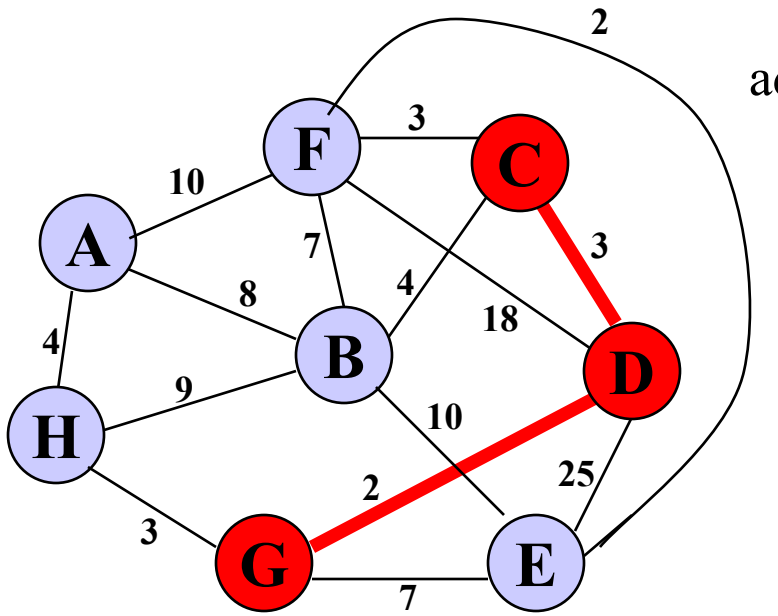
Select node with minimum distance

	K	d_v	p_v
A			
B			
C	T	3	D
D	T	0	–
E		7	G
F		18	D
G	T	2	D
H		3	G

d_v = Cheapest edge cost to T

p_v = Node in T to which the cheapest edge is connected

Prim's Algorithm: Walk-Through



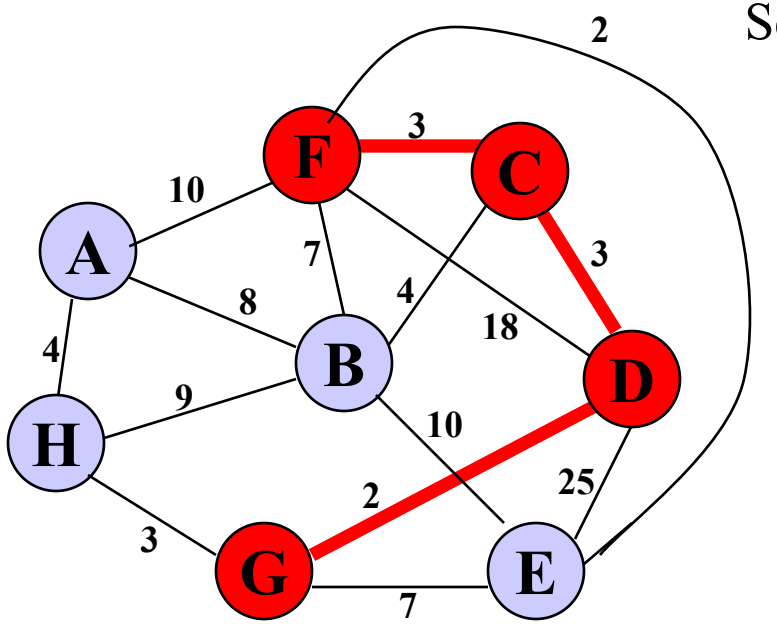
Update distances of adjacent, unselected nodes

	K	d_v	p_v
A			
B		4	C
C	T	3	D
D	T	0	–
E		7	G
F		3	C
G	T	2	D
H		3	G

d_v = Cheapest edge cost to T

p_v = Node in T to which the cheapest edge is connected

Prim's Algorithm: Walk-Through



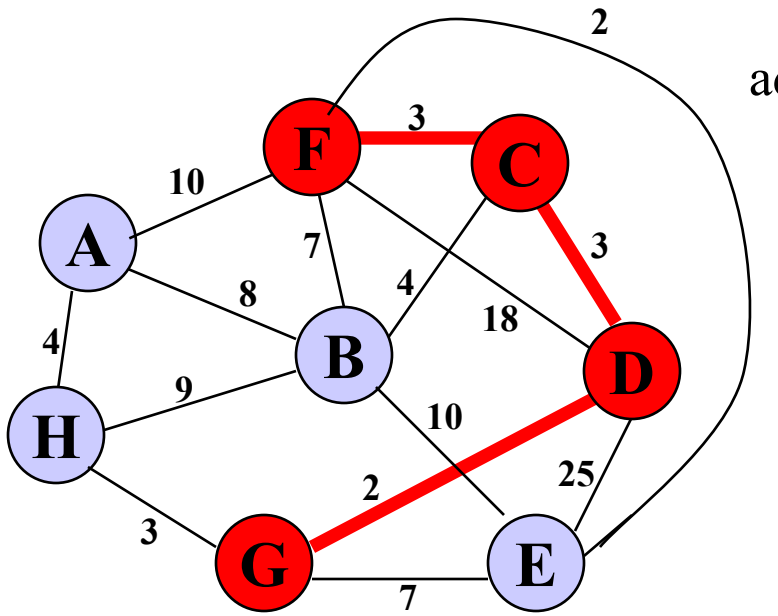
Select node with minimum distance

	K	d_v	p_v
A			
B		4	C
C	T	3	D
D	T	0	–
E		7	G
F	T	3	C
G	T	2	D
H		3	G

d_v = Cheapest edge cost to T

p_v = Node in T to which the cheapest edge is connected

Prim's Algorithm: Walk-Through



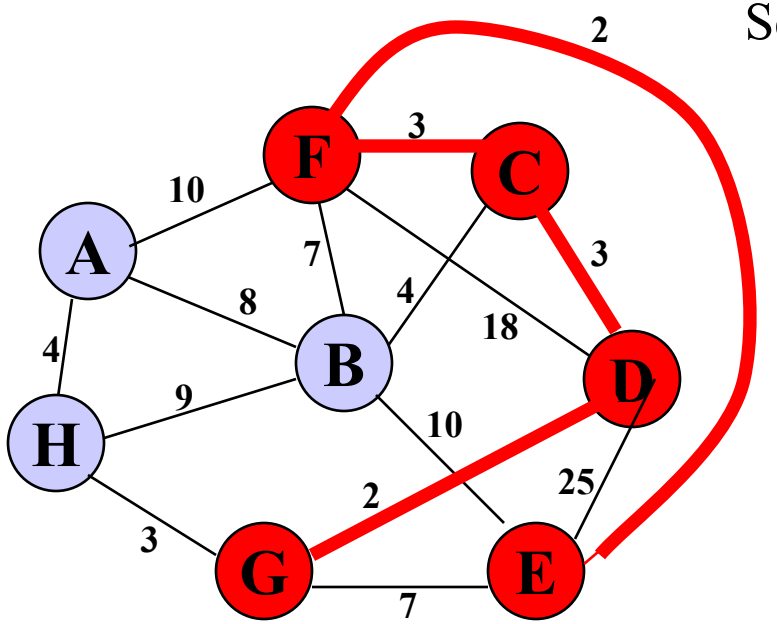
Update distances of adjacent, unselected nodes

	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	–
E		2	F
F	T	3	C
G	T	2	D
H		3	G

d_v = Cheapest edge cost to T

p_v = Node in T to which the cheapest edge is connected

Prim's Algorithm: Walk-Through



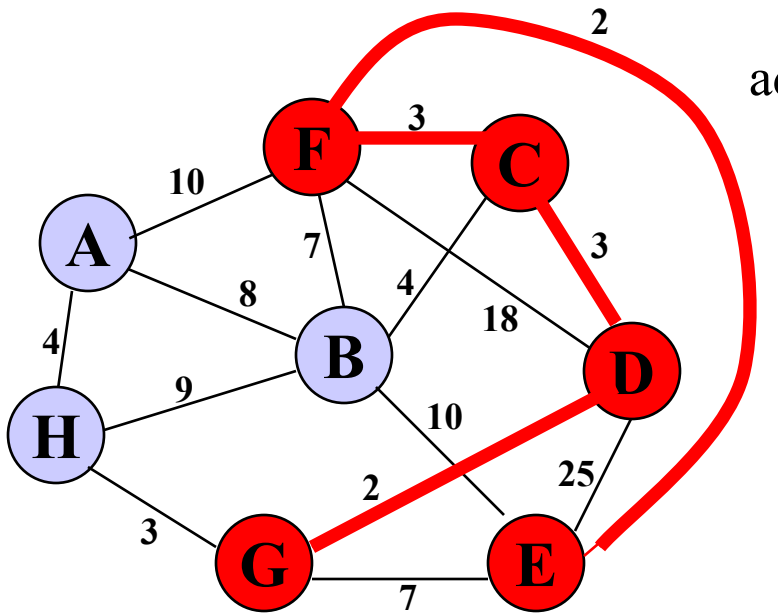
Select node with minimum distance

	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H		3	G

d_v = Cheapest edge cost to T

p_v = Node in T to which the cheapest edge is connected

Prim's Algorithm: Walk-Through



Update distances of adjacent, unselected nodes

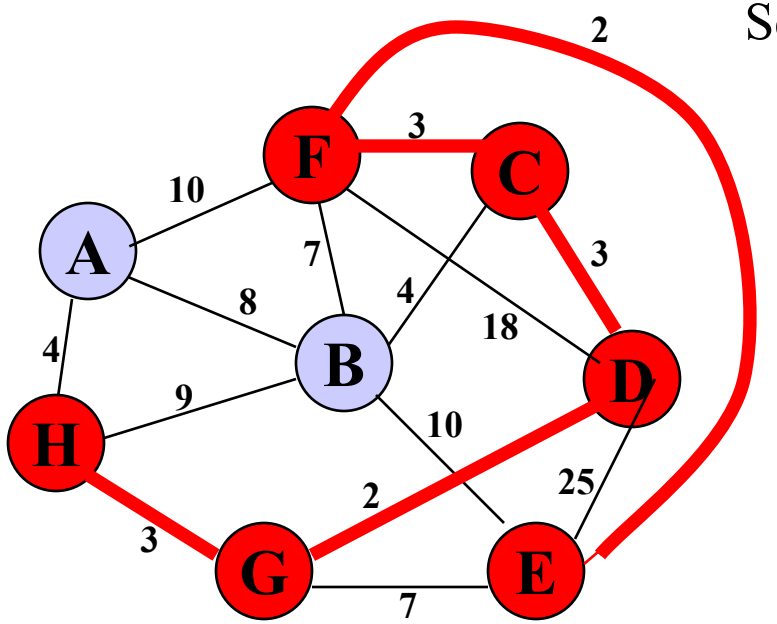
	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H		3	G

Table entries unchanged

d_v = Cheapest edge cost to T

p_v = Node in T to which the cheapest edge is connected

Prim's Algorithm: Walk-Through



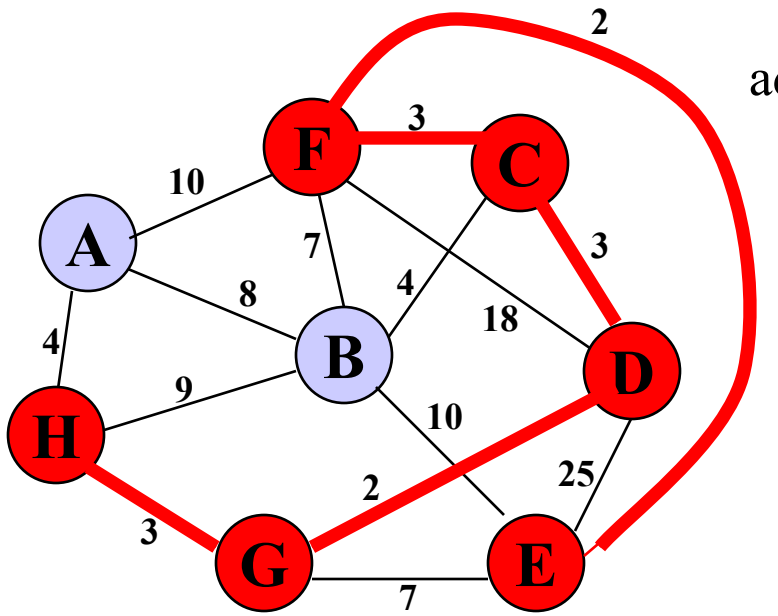
Select node with minimum distance

	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

d_v = Cheapest edge cost to T

p_v = Node in T to which the cheapest edge is connected

Prim's Algorithm: Walk-Through



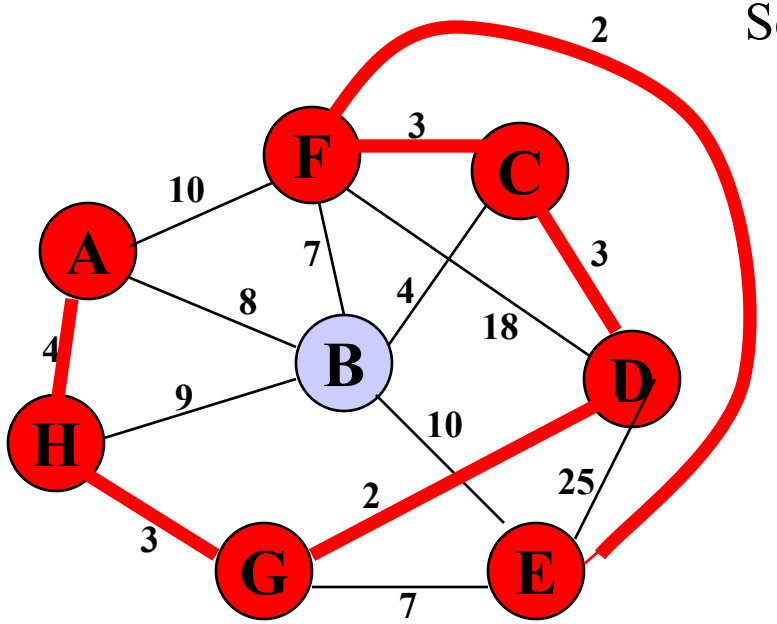
Update distances of adjacent, unselected nodes

	K	d_v	p_v
A		4	H
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

d_v = Cheapest edge cost to T

p_v = Node in T to which the cheapest edge is connected

Prim's Algorithm: Walk-Through



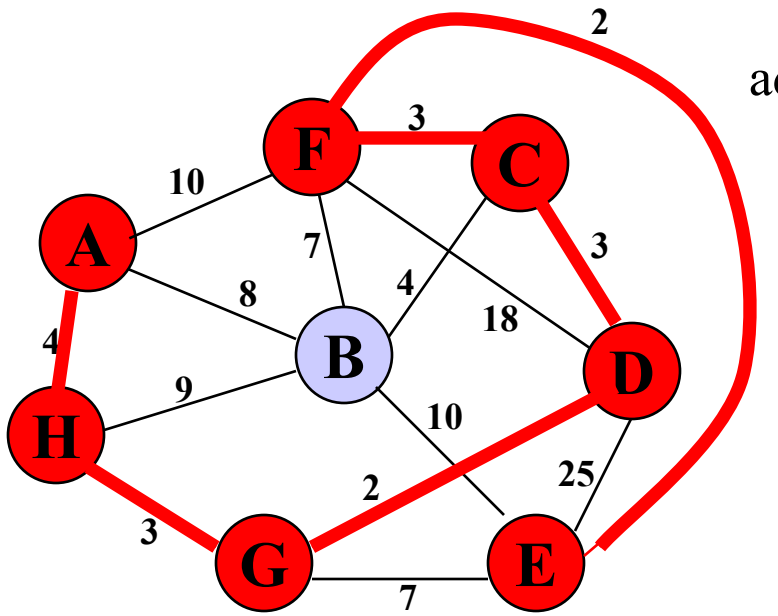
Select node with minimum distance

	K	d_v	p_v
A	T	4	H
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

d_v = Cheapest edge cost to T

p_v = Node in T to which the cheapest edge is connected

Prim's Algorithm: Walk-Through



Update distances of adjacent, unselected nodes

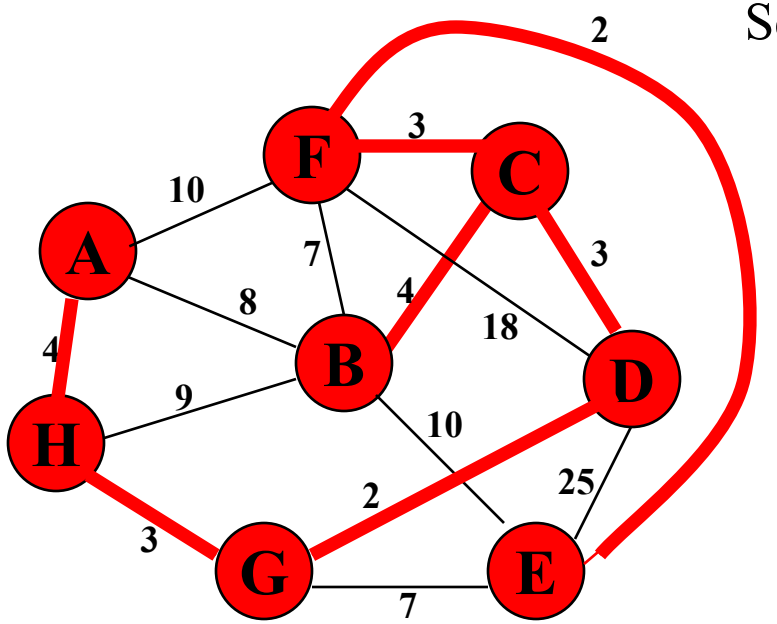
	K	d_v	p_v
A	T	4	H
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

Table entries unchanged

d_v = Cheapest edge cost to T

p_v = Node in T to which the cheapest edge is connected

Prim's Algorithm: Walk-Through



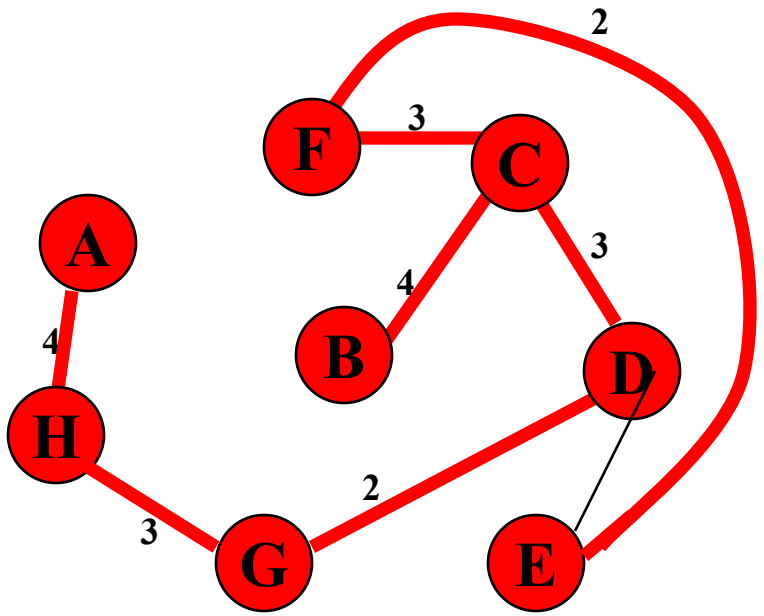
Select node with minimum distance

	K	d_v	p_v
A	T	4	H
B	T	4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

d_v = Cheapest edge cost to T

p_v = Node in T to which the cheapest edge is connected

Prim's Algorithm: Walk-Through



Cost of Minimum Spanning Tree = $\Sigma\, d_v$ = **21**

	K	d_v	p_v
A	T	4	H
B	T	4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C
G	T	2	D
H	T	3	G

d_v = Cheapest edge cost to T

p_v = Node in T to which the cheapest edge is connected

Prim's Algorithm:

```
MST-PRIM(G, w, r)
1  for each u ∈ G.V
2      u.key ← ∞
3      u.π ← NIL
4   r.key ← 0
5   Q ← G.V
6   while Q ≠ Ø
7      u ← EXTRACT-MIN(Q)
8      for each v ∈ G.Adj[u]
9          if v ∈ Q and w(u, v) < v.key
10             v.π ← u
11             v.key ← w(u, v)
```

```
USING A BINARY HEAP: O(VlogV + ElogV)
USING A FIBONACCI HEAP: O(E+VlogV)
```

http://stackoverflow.com/questions/20430740/time-complexity-of-prims-algorithm

Kruskal's Algorithm:

```
KRUSKAL(G):
1 A = ∅
2 foreach v ∈ G.V:
3    MAKE-SET(v)
4 foreach (u, v) ordered by weight(u, v), increasing:
5    if FIND-SET(u) ≠ FIND-SET(v):
6       A = A ∪ {(u, v)}
7       UNION(u, v)
8 return A
```

http://stackoverflow.com/questions/20430740/time-complexity-of-prims-algorithm

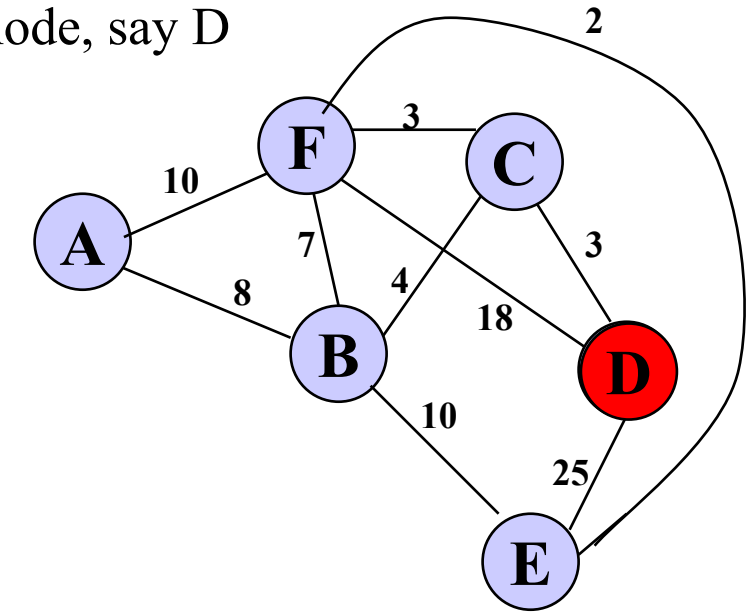
CS502, Design and Analysis of Algorithm, Spring 2016

Student ID: _____ Name: _____
Student ID: _____ Name: _____
Student ID: _____ Name: _____

Prim's Algorithm

	K	d_v	p_v
A			
B			
C			
D	T	0	–
E			
F			

Start with any node, say D



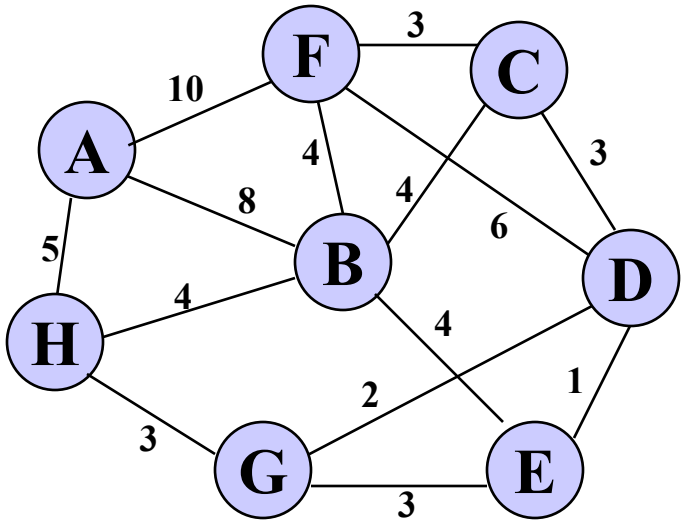
Fill in the above table (5pt)

What is the cost of the minimum spanning tree? (1pt)

Kruskal's Algorithm

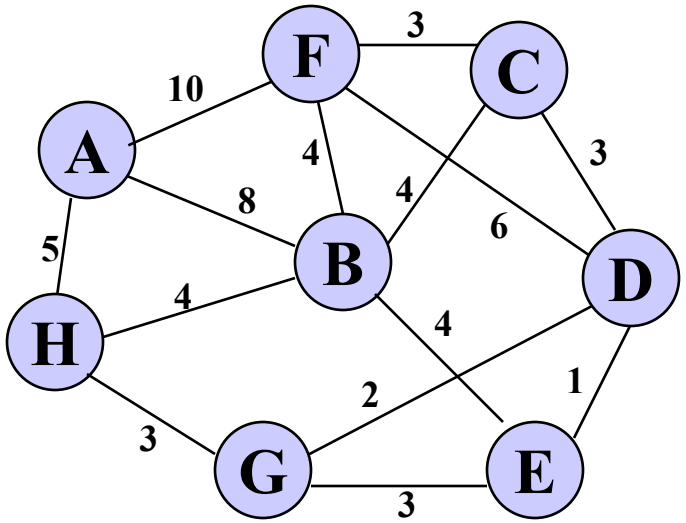
- Work with edges, rather than nodes
- Two steps:
  - Sort edges by increasing edge weight
  - Select the first |V| – 1 edges that do not generate a cycle

Kruskal's Algorithm: Walk-Through



Consider an undirected, weight graph

Kruskal's Algorithm: Walk-Through



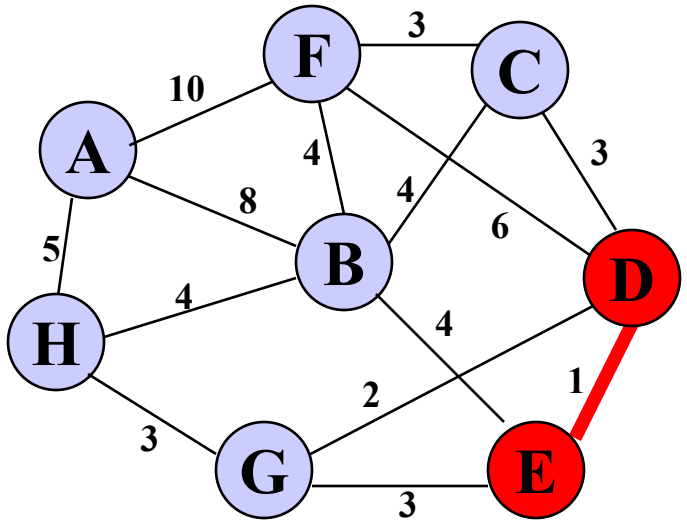
Sort the edges by increasing edge weight

edge	d_v	
(D,E)	1	
(D,G)	2	
(E,G)	3	
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Kruskal's Algorithm: Walk-Through

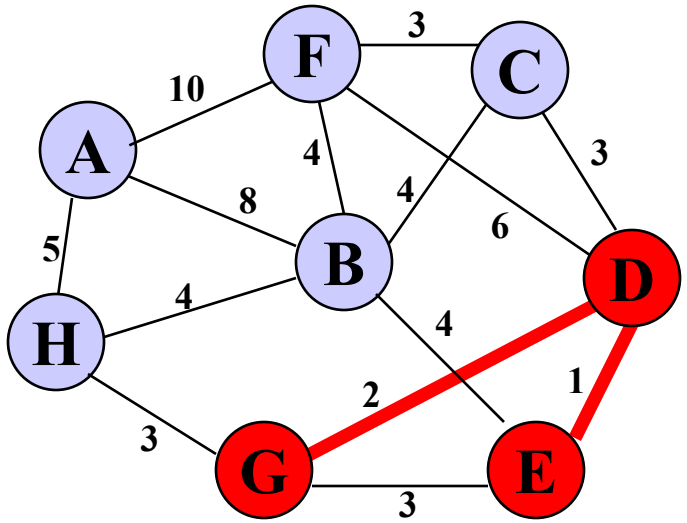
Select first |V|–1 edges which do not generate a cycle



edge	d_v	
(D,E)	1	√
(D,G)	2	
(E,G)	3	
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Kruskal's Algorithm: Walk-Through

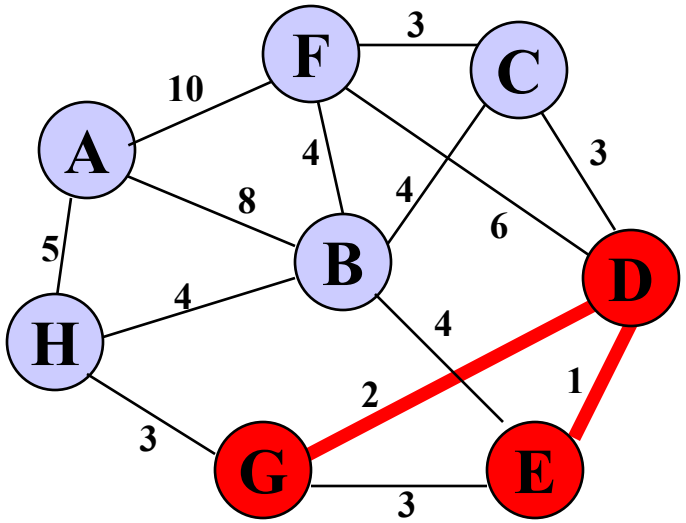


Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Kruskal's Algorithm: Walk-Through



Select first |V|–1 edges which do not generate a cycle

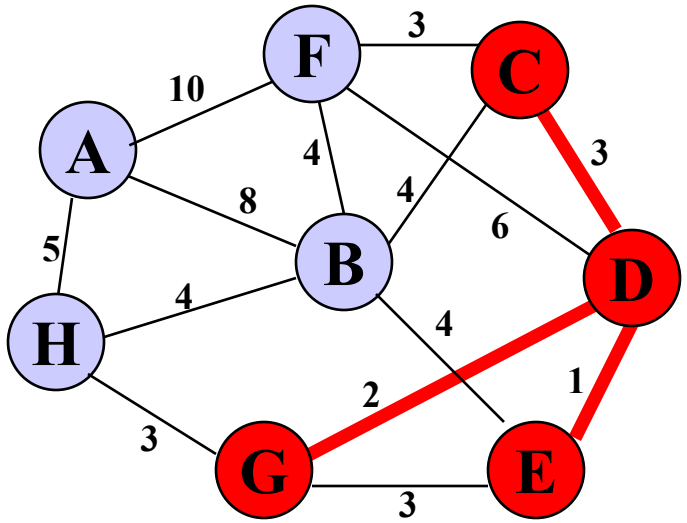
edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	
(G,H)	3	
(C,F)	3	
(B,C)	4	

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Accepting edge (E,G) would create a cycle

Kruskal's Algorithm: Walk-Through

Select first |V|–1 edges which do not generate a cycle

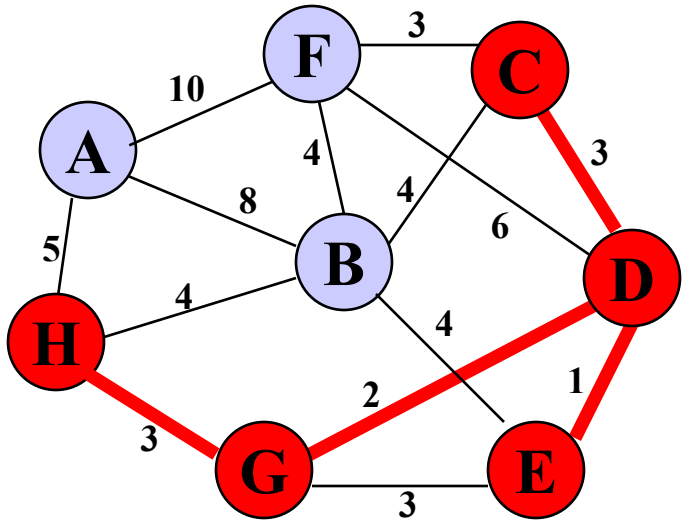


edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	
(C,F)	3	
(B,C)	4	

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Kruskal's Algorithm: Walk-Through

Select first |V|–1 edges which do not generate a cycle

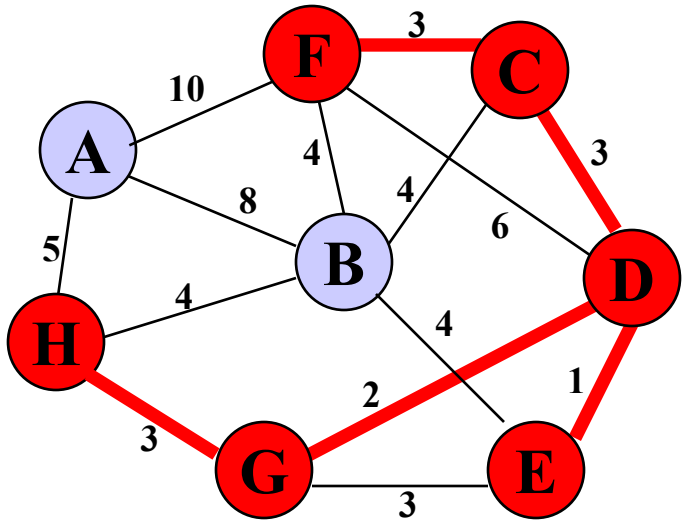


edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	√
(C,F)	3	
(B,C)	4	

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Kruskal's Algorithm: Walk-Through

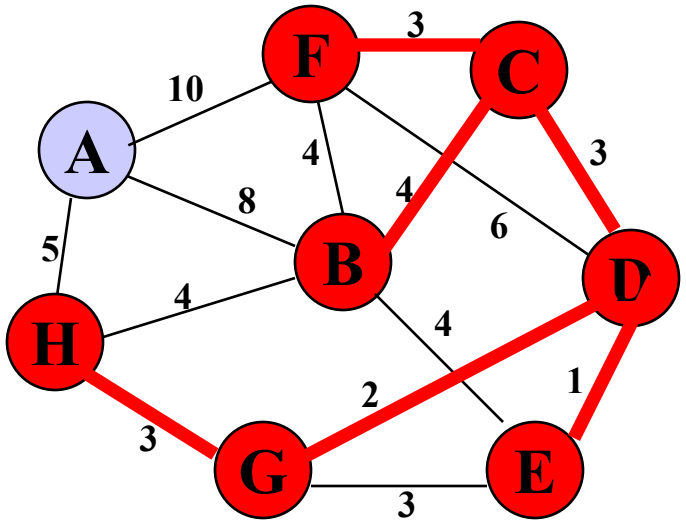
Select first |V|–1 edges which do not generate a cycle



edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Kruskal's Algorithm: Walk-Through



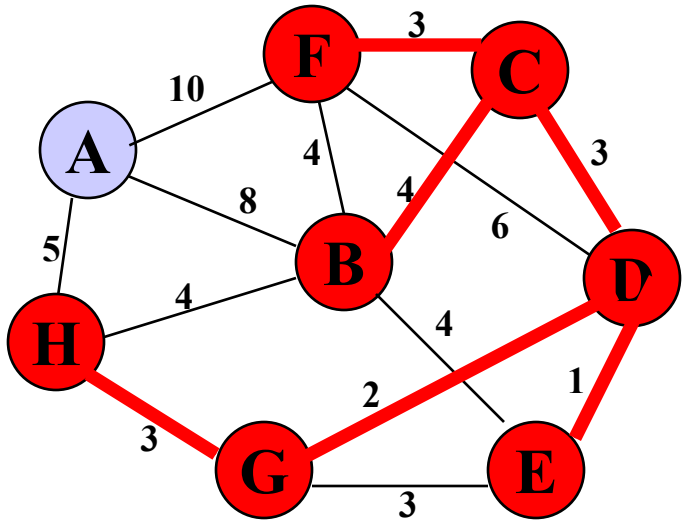
Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	√

edge	d_v	
(B,E)	4	
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Kruskal's Algorithm: Walk-Through

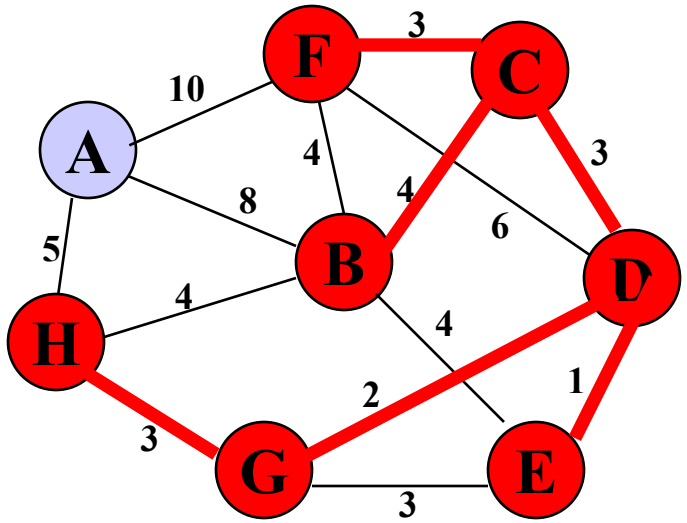
Select first |V|–1 edges which do not generate a cycle



edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	√

edge	d_v	
(B,E)	4	χ
(B,F)	4	
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Kruskal's Algorithm: Walk-Through

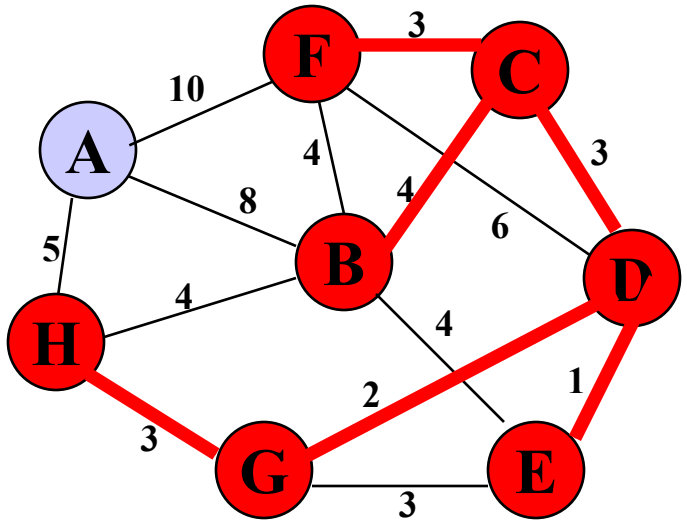


Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	√

edge	d_v	
(B,E)	4	χ
(B,F)	4	χ
(B,H)	4	
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Kruskal's Algorithm: Walk-Through

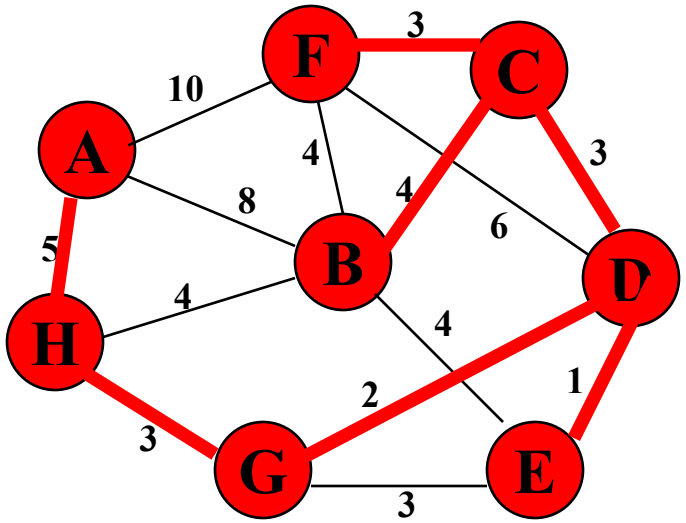


Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	√

edge	d_v	
(B,E)	4	χ
(B,F)	4	χ
(B,H)	4	χ
(A,H)	5	
(D,F)	6	
(A,B)	8	
(A,F)	10	

Kruskal's Algorithm: Walk-Through

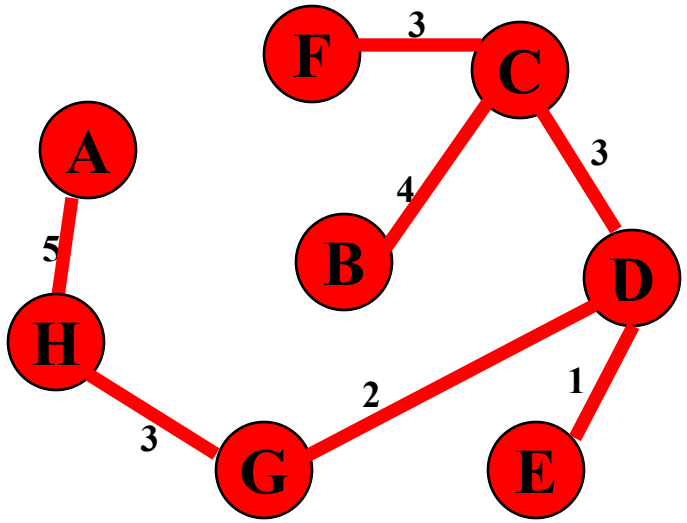


Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	√

edge	d_v	
(B,E)	4	χ
(B,F)	4	χ
(B,H)	4	χ
(A,H)	5	√
(D,F)	6	
(A,B)	8	
(A,F)	10	

Kruskal's Algorithm: Walk-Through



Select first |V|–1 edges which do not generate a cycle

edge	d_v	
(D,E)	1	√
(D,G)	2	√
(E,G)	3	χ
(C,D)	3	√
(G,H)	3	√
(C,F)	3	√
(B,C)	4	√

edge	d_v	
(B,E)	4	χ
(B,F)	4	χ
(B,H)	4	χ
(A,H)	5	√
(D,F)	6	
(A,B)	8	
(A,F)	10	

not considered

Done

Time Complexity:
O(E log E), or equivalently, O(E log V)

Total Cost = $\Sigma\ d_v = 21$

Kruskal's Algorithm:

```
KRUSKAL(G):
1 A = ∅
2 foreach v ∈ G.V:
3    MAKE-SET(v)
4 foreach (u, v) ordered by weight(u, v), increasing:
5    if FIND-SET(u) ≠ FIND-SET(v):
6       A = A ∪ {(u, v)}
7       UNION(u, v)
8 return A
```

Time Complexity:
O(E log E), or equivalently, O(E log V)

http://stackoverflow.com/questions/20430740/time-complexity-of-prims-algorithm

Prim's vs Kruskal's Algorithm: Time Complexity

For a graph with V vertices E edges,

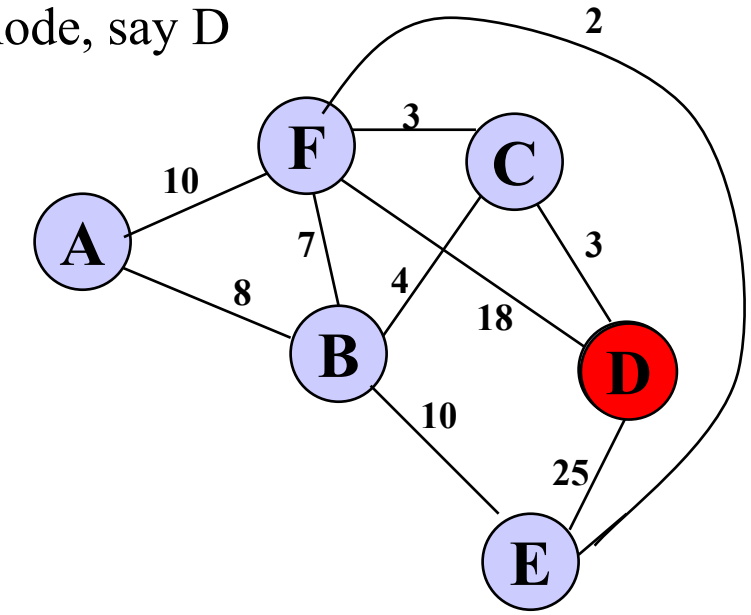
- Kruskal's algorithm runs in O(E log V) time
  - Good for sparse graphs because it uses simpler data structures.
- Prim's algorithm can run in O(E + V log V) amortized time, if you use a Fibonacci Heap.
  - Good for a **dense** graph : $E = V^2$

Student ID: _____ Name: _____
Student ID: _____ Name: _____
Student ID: _____ Name: _____

Prim's Algorithm

	K	d_v	p_v
A			
B			
C			
D	T	0	–
E			
F			

Start with any node, say D



Fill in the above table (5pt)

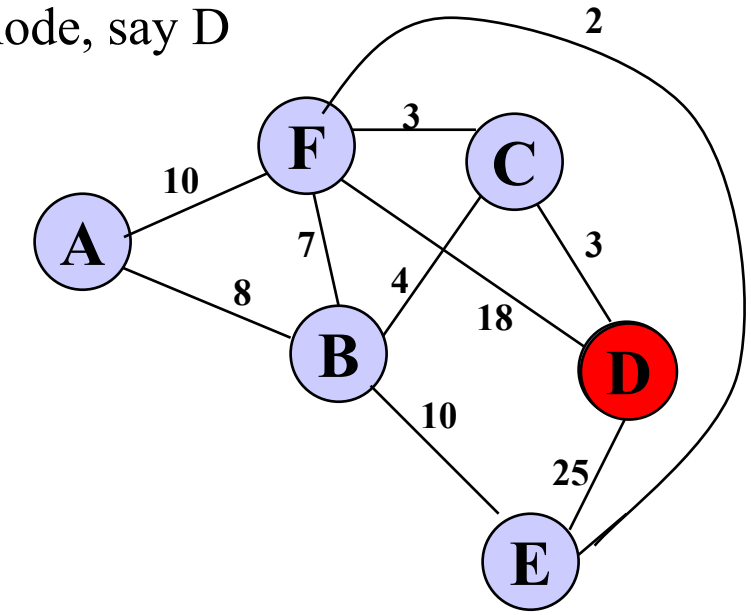
What is the cost of the minimum spanning tree? (1pt)

Student ID: _____ Name: _____
Student ID: _____ Name: _____
Student ID: _____ Name: _____

Prim's Algorithm

	K	d_v	p_v
A			
B			
C		3	D
D	T	0	–
E		25	D
F		18	D

Start with any node, say D



Fill in the above table (5pt)

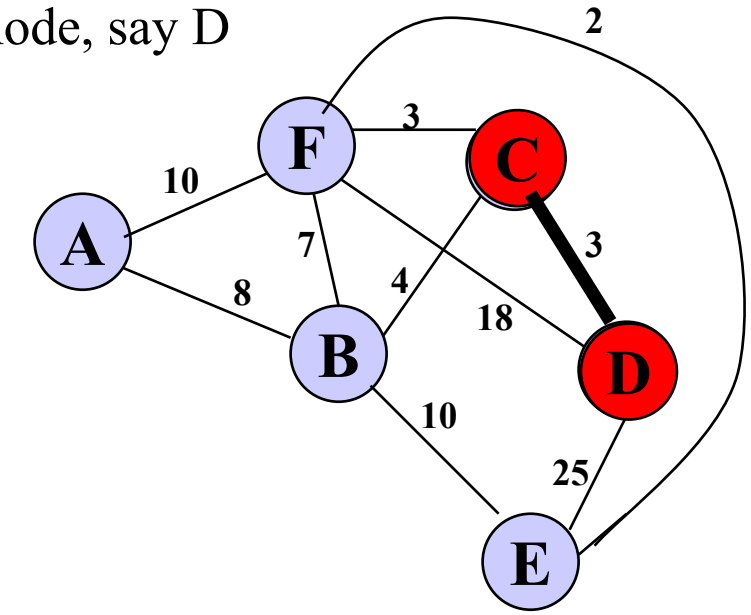
What is the cost of the minimum spanning tree? (1pt)

Student ID: _____ Name: _____
Student ID: _____ Name: _____
Student ID: _____ Name: _____

Prim's Algorithm

	K	d_v	p_v
A			
B		4	C
C	T	3	D
D	T	0	–
E		25	D
F		3	C

Start with any node, say D



Fill in the above table (5pt)

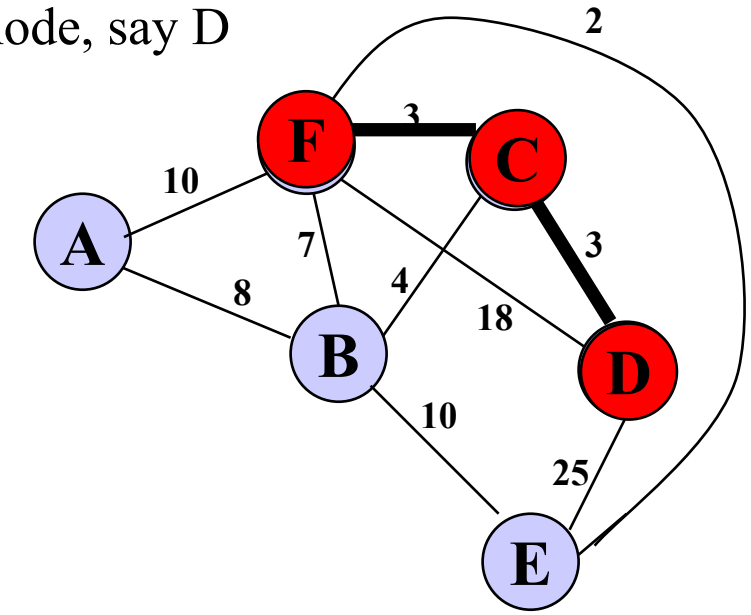
What is the cost of the minimum spanning tree? (1pt)

Student ID: _____ Name: _____
Student ID: _____ Name: _____
Student ID: _____ Name: _____

Prim's Algorithm

	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	–
E		2	F
F	T	3	C

Start with any node, say D



Fill in the above table (5pt)

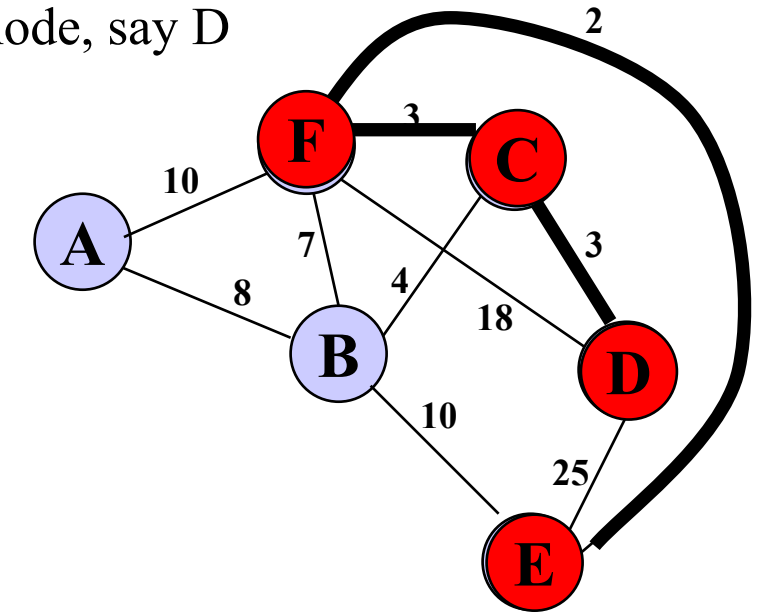
What is the cost of the minimum spanning tree? (1pt)

Student ID: _____ Name: _____
Student ID: _____ Name: _____
Student ID: _____ Name: _____

Prim's Algorithm

	K	d_v	p_v
A		10	F
B		4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C

Start with any node, say D



Fill in the above table (5pt)

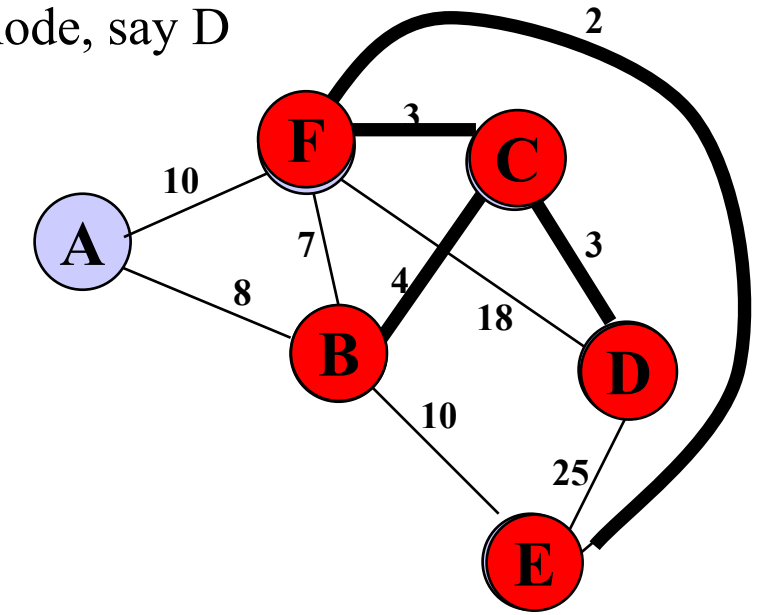
What is the cost of the minimum spanning tree? (1pt)

Student ID: _____ Name: _____
Student ID: _____ Name: _____
Student ID: _____ Name: _____

Prim's Algorithm

	K	d_v	p_v
A		8	F
B	T	4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C

Start with any node, say D



Fill in the above table (5pt)

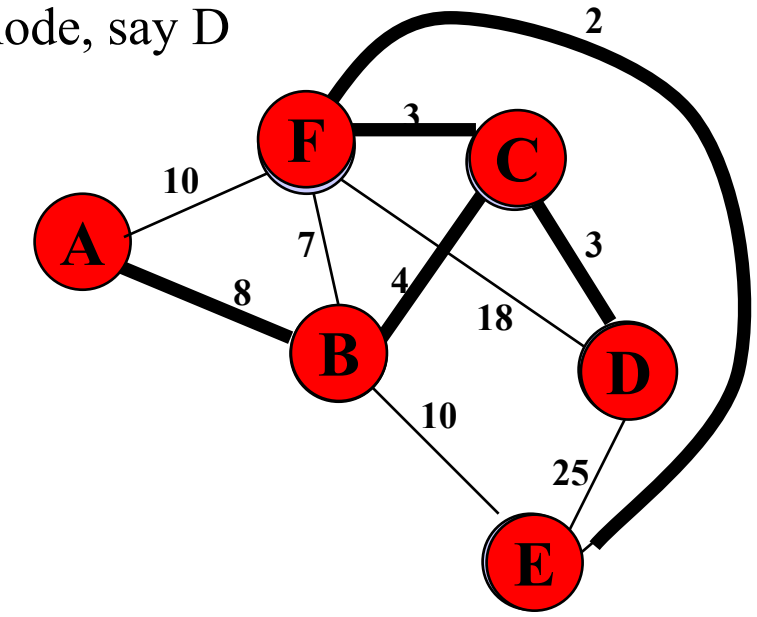
What is the cost of the minimum spanning tree? (1pt)

Student ID: _____ Name: _____
Student ID: _____ Name: _____
Student ID: _____ Name: _____

Prim's Algorithm

	K	d_v	p_v
A	T	8	B
B	T	4	C
C	T	3	D
D	T	0	–
E	T	2	F
F	T	3	C

Start with any node, say D



Fill in the above table (5pt)

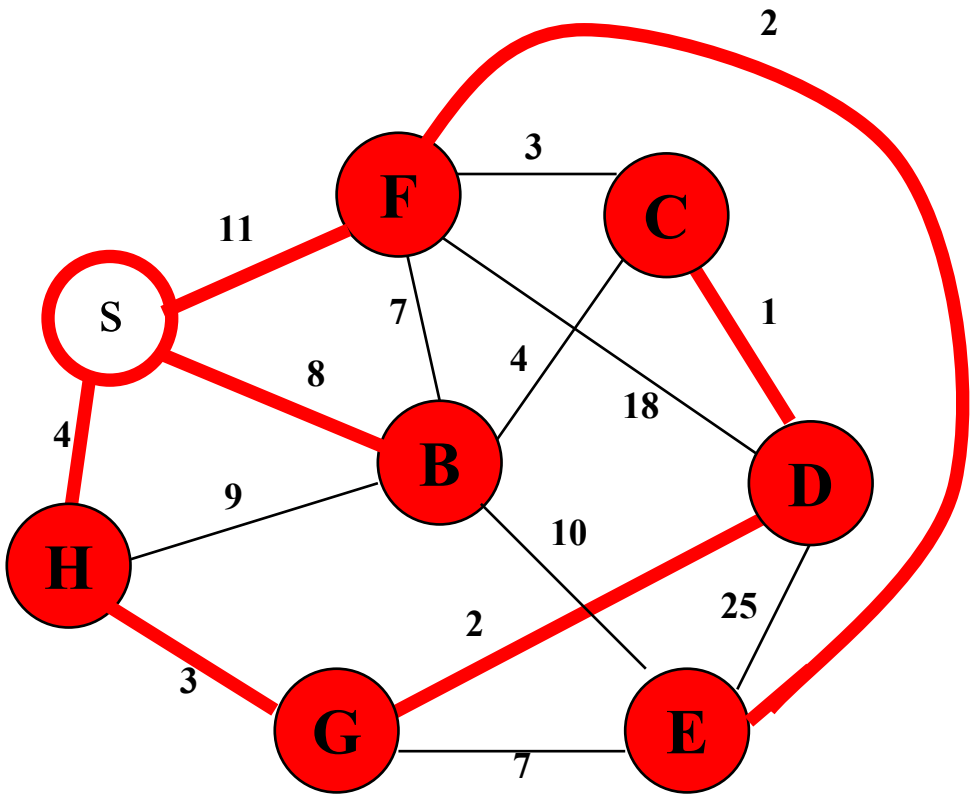
What is the cost of the minimum spanning tree? (1pt) 20

Design and Analysis of Algorithms Lecture-6: Shortest Path Trees

Dr. Chung-Wen Albert Tsao

Single-Source Shortest Path Problem

Finding shortest paths from source s to all other vertices in the graph.



Dijkstra's Algorithm For SPT

- Dijkstra's algorithm
  - Solution to the single-source shortest path problem in graph theory.
  - Works on both **directed** and **undirected** graphs.
  - All edges must have **nonnegative** weights.
- Approach: Greedy
- Input:
  - Weighted graph G={E,V} and source vertex s∈V, such that all edge weights are nonnegative
- Output:
  - Lengths of shortest paths (or the shortest paths themselves) from a given source vertex s∈V to all other vertices

Recall (Prim's Algorithm for MST)

- Start with any vertex s and greedily grow a tree T from s.
- Repeat the same process until no more nodes outside T
  - Among all nodes outside T, pick a node, say v, with the **cheapest edge to T** (denoted d_v)
  - Connect v to T

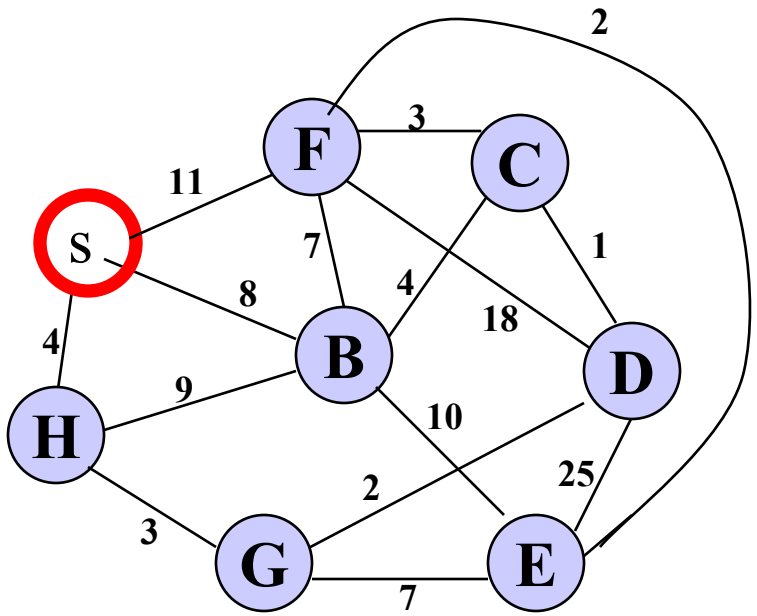
Dijkstra's Algorithm for SPT

- Start with any vertex *s* and greedily grow a tree *T* from *s*.
- Repeat the same process until no more nodes outside T
  - Among all nodes outside *T*, pick a node, say v, with the **shortest path length to *s*** (denoted d_v)
  - Connect *v* to *T*

Similar to Prim's Algorithm except that

- There is a source node s
- d_v records shortest path length to s, not the cheapest edge to T

Dijkstra's Algorithm: Walk-Through



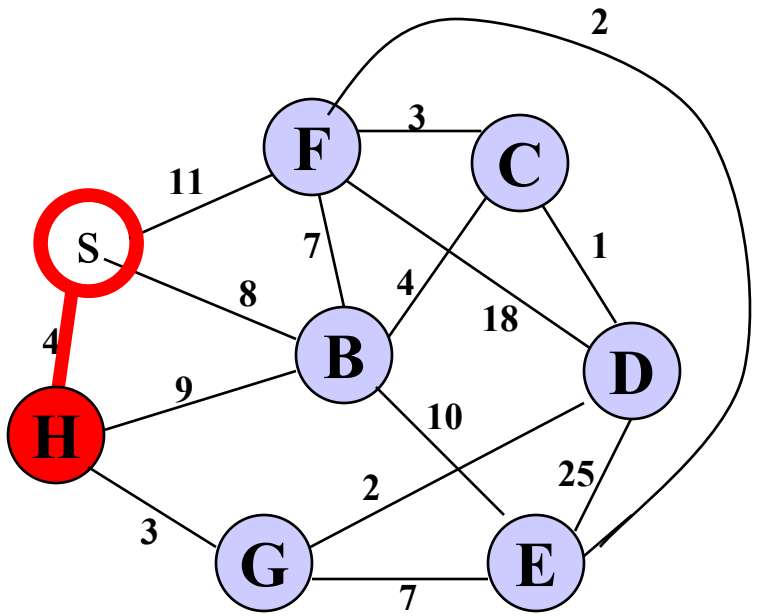
Initialize array

	K	d_v	p_v
s	T	0	–
B		8	σ
C		∞	–
D		∞	–
E		∞	–
F		11	σ
G		∞	–
H		4	σ

d_v = **Shortest path length from source node *s* to *v***

p_v = Node in T to which node v is connected

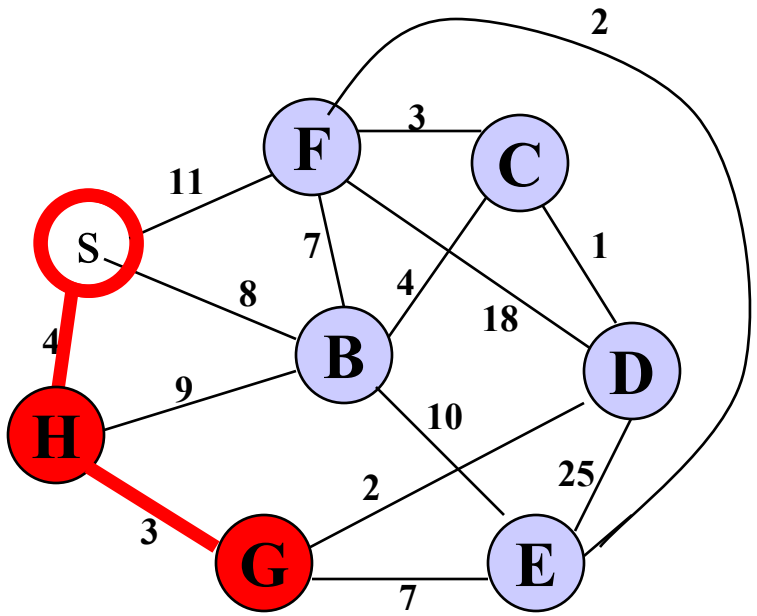
Dijkstra's Algorithm: Walk-Through



Initialize array

	K	d_v	p_v
s	χ	0	–
B		8	σ
C		∞	–
D		∞	–
E		∞	–
F		11	σ
G		7	H
H	√	4	σ

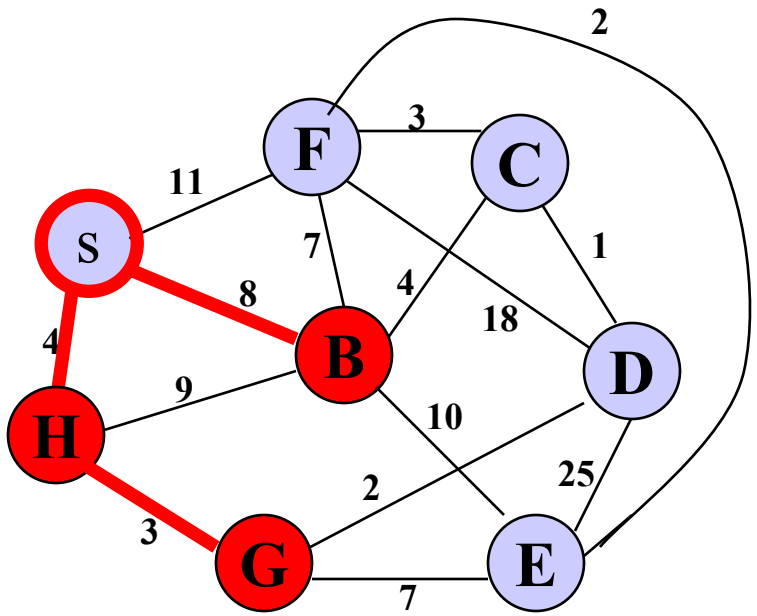
Dijkstra's Algorithm: Walk-Through



Initialize array

	K	d_v	p_v
s	χ	0	–
B		8	σ
C		∞	–
D		9	Γ
E		14	Γ
F		11	σ
G	√	7	H
H	χ	4	σ

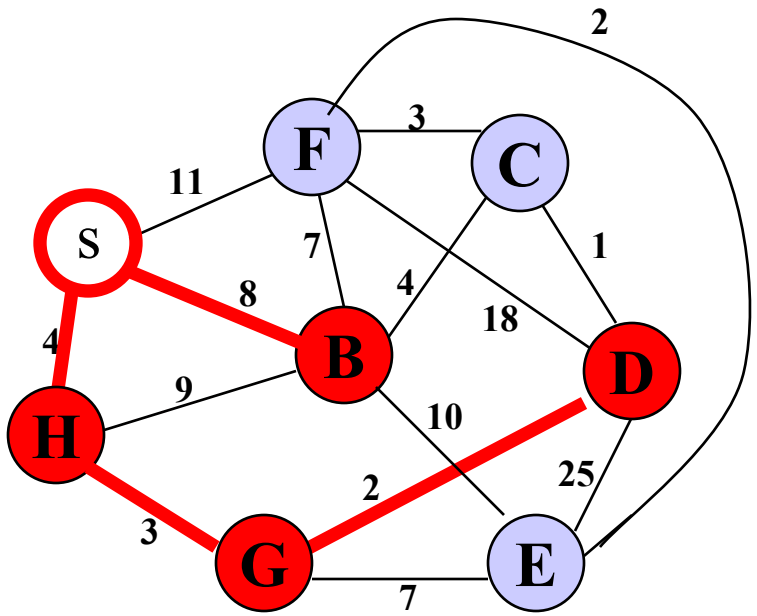
Dijkstra's Algorithm: Walk-Through



Initialize array

	K	d_v	p_v
s	χ	0	–
B	√	8	σ
C		12	B
D		9	Γ
E		14	Γ
F		11	σ
G	χ	7	H
H	χ	4	σ

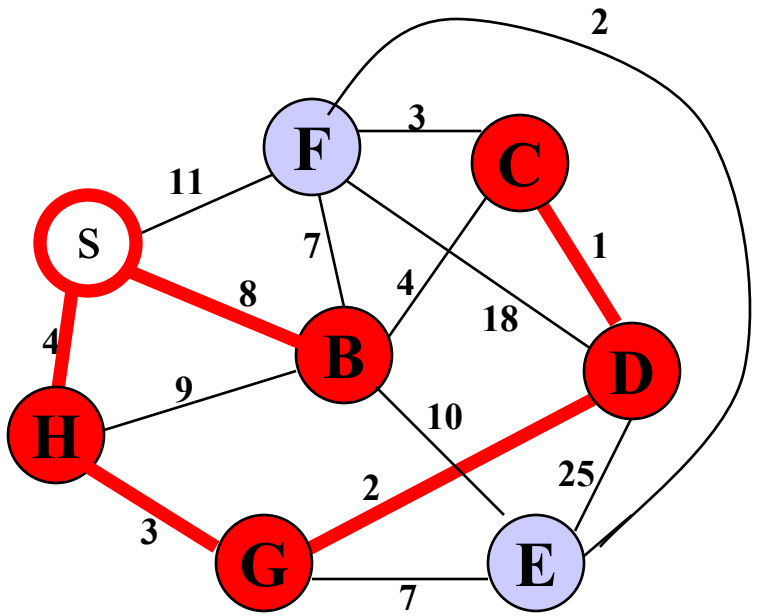
Dijkstra's Algorithm: Walk-Through



Initialize array

	K	d_v	p_v
s	χ	0	–
B	χ	8	σ
C		10	Δ
D	√	9	Γ
E		14	Γ
F		11	σ
G	χ	7	H
H	χ	4	σ

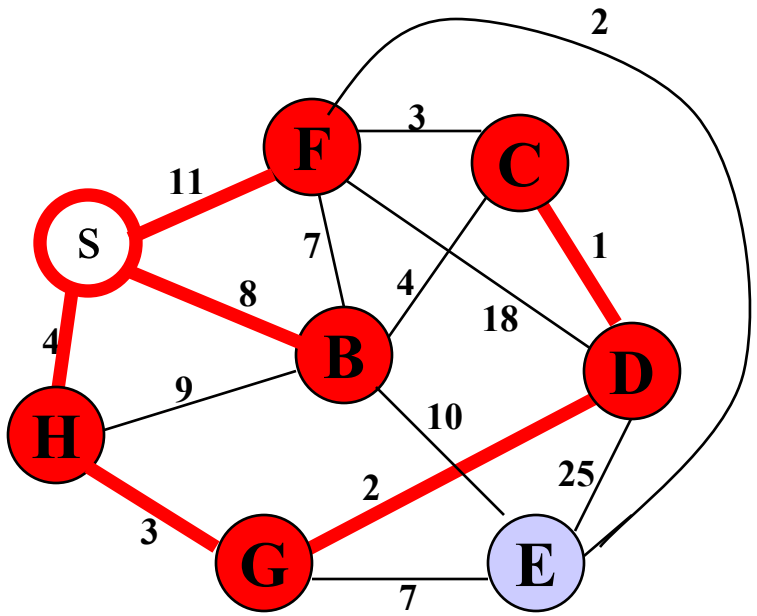
Dijkstra's Algorithm: Walk-Through



Initialize array

	K	d_v	p_v
s	χ	0	–
B	χ	8	σ
C	√	10	Δ
D	χ	9	Γ
E		14	Γ
F		11	σ
G	χ	7	H
H	χ	4	σ

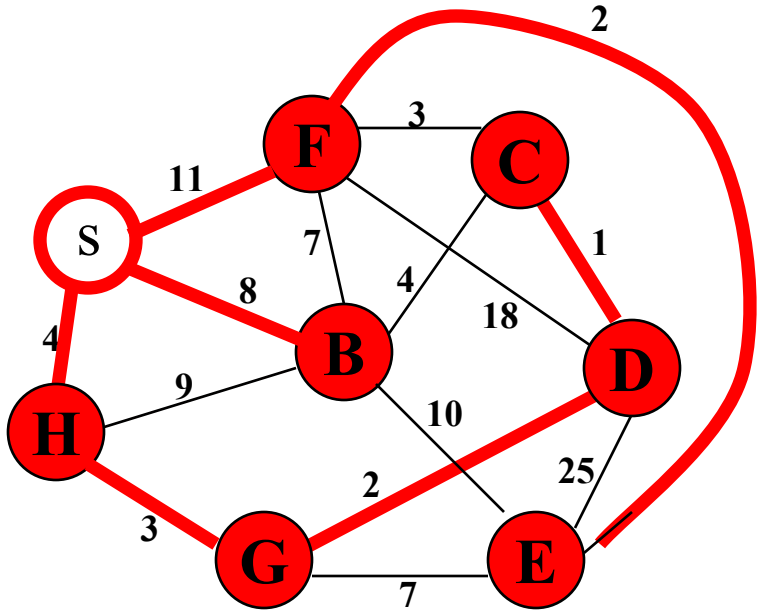
Dijkstra's Algorithm: Walk-Through



Initialize array

	K	d_v	p_v
s	χ	0	–
B	χ	8	σ
C	χ	10	Δ
D	χ	9	Γ
E		13	Φ
F	√	11	σ
G	χ	7	H
H	χ	4	σ

Dijkstra's Algorithm: Walk-Through



Initialize array

	K	d_v	p_v
s	χ	0	–
B	χ	8	σ
C	χ	10	Δ
D	χ	9	Γ
E		13	Φ
F	√	11	σ
G	χ	7	H
H	χ	4	σ

Dijkstra's algorithm - Pseudocode

```
dist[s] ←0                              (distance to source vertex is zero)
for  all v ∈ V−{s}
       do  dist[v] ←∞                   (set all other distances to infinity)
S←∅                                     (S, the set of visited vertices is initially empty)
Q←V                                     (Q, the queue initially contains all
vertices)
while Q ≠∅                              (while the queue is not empty)
do   u ← min_distance(Q,  dist)         (select the element of Q with the min. distance)
     S←S∪{u}                            (add u to list of visited vertices)
      for all v ∈ neighbors[u]          (v is a neighbor of  node u)
            do  if   dist[v] > dist[u] + w(u, v)          (if new shortest path found)
                     then      d[v] ←d[u] + w(u, v)       (set new value of shortest path)
                           (if desired, add traceback code)
return dist
```

IMPLEMENTATIONS AND RUNNING TIMES

The simplest implementation is to store vertices in an array or linked list. This will produce a running time of

$O(|V|^2 + |E|)$

For **sparse** graphs, it can be implemented more efficiently storing the graph in an adjacency list using a **binary heap** or **priority queue**. This will produce a running time of

$O((|E|+|V|) \log |V|)$

Dijkstra's Algorithm - Why It Works

- As with all greedy algorithms, we need to make sure that it is a correct algorithm (e.g., it always returns the right solution if it is given correct input).

- A formal proof would take longer than this presentation, but we can understand how the argument works intuitively.

- If you can’t sleep unless you see a proof, see the second reference or ask us where you can find it.

Dijkstra's Algorithm - Why It Works

- To understand how it works, we'll go over the previous example again. However, we need two mathematical results first:

- Lemma 1: Triangle inequality
  If $\delta(u,v)$ is the shortest path length between u and v,
  $\delta(u,v) \leq \delta(u,x) + \delta(x,v)$
- Lemma 2:
  The subpath of any shortest path is itself a shortest path.

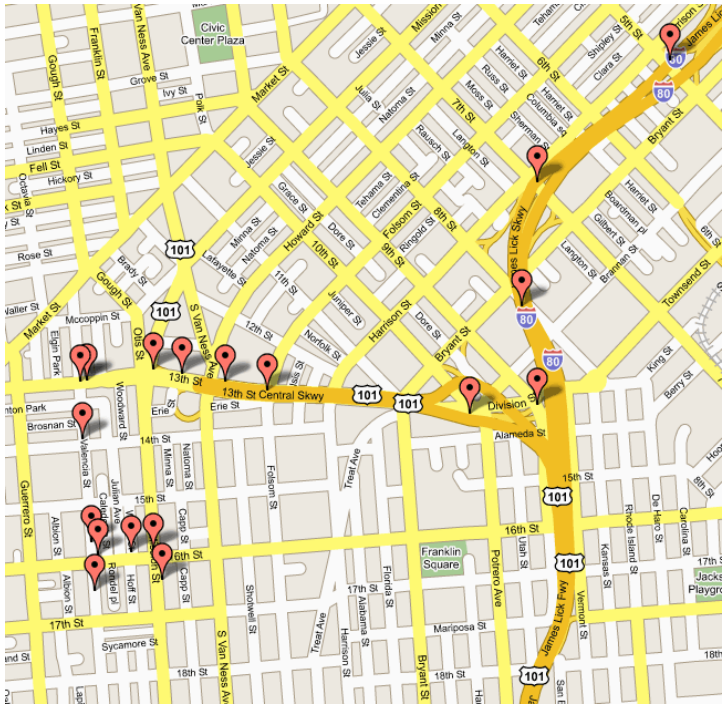
- The key is to understand why we can claim that anytime we put a new vertex in S, we can say that we already know the shortest path to it.
- Now, back to the example…

Dijkstra's Algorithm - Why use it?

- As mentioned, Dijkstra’s algorithm calculates the shortest path to every vertex.
- However, it is about as computationally expensive to calculate the shortest path from vertex u to every vertex using Dijkstra’s as it is to calculate the shortest path to some particular vertex v.
- Therefore, anytime we want to know the optimal path to some other vertex from a determined origin, we can use Dijkstra’s algorithm.

Applications of Dijkstra's Algorithm

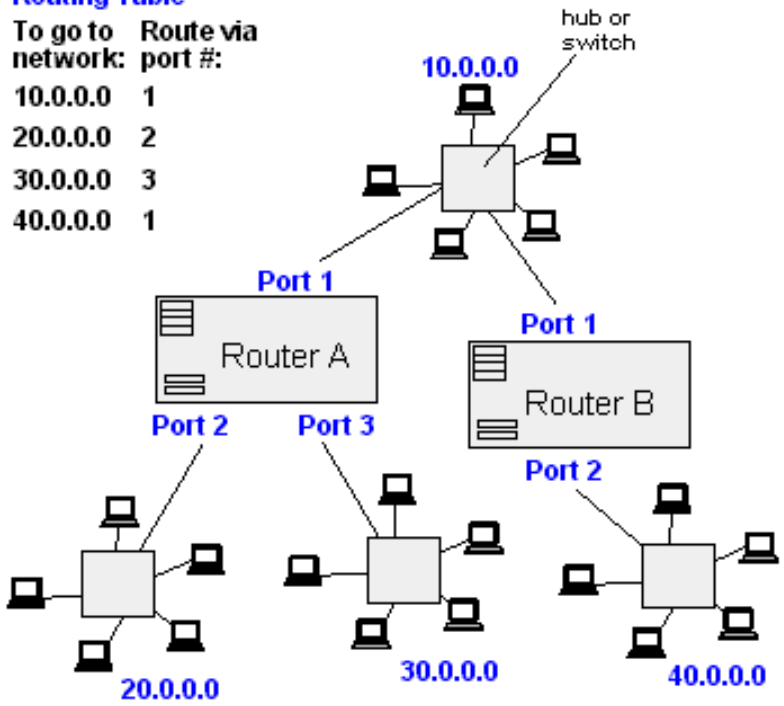
- Traffic Information Systems are most prominent use
- Mapping (Map Quest, Google Maps)
- Routing Systems



From Computer Desktop Encyclopedia
© 1998 The Computer Language Co. Inc.

Router A
Routing Table

To go to network:	Route via port #:
10.0.0.0	1
20.0.0.0	2
30.0.0.0	3
40.0.0.0	1

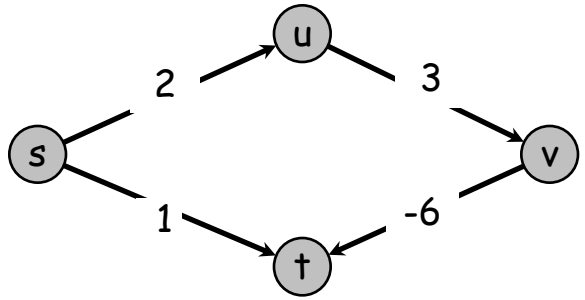


References

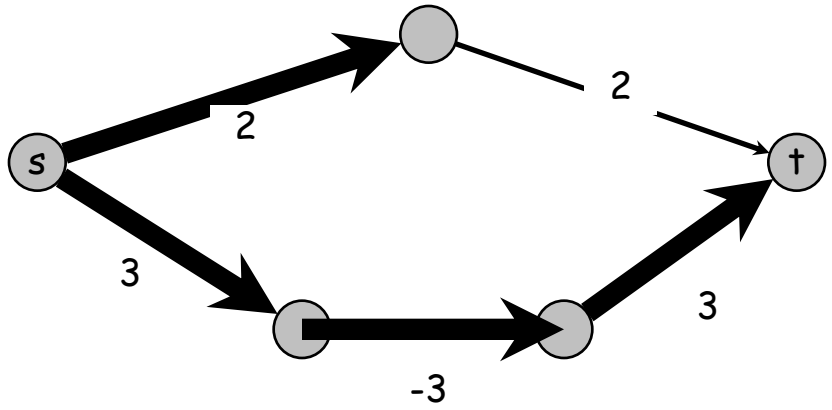
- Dijkstra's original paper:
  E. W. Dijkstra. (1959) A Note on Two Problems in Connection with Graphs. Numerische Mathematik, 1. 269-271.
- MIT OpenCourseware, 6.046J Introduction to Algorithms.
  < http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-046JFall-2005/CourseHome/> Accessed 4/25/09
- Meyers, L.A. (2007) Contact network epidemiology: Bond percolation applied to infectious disease prediction and control. Bulletin of the American Mathematical Society 44: 63-86.
- Department of Mathematics, University of Melbourne. Dijkstra's Algorithm.
  <http://www.ms.unimelb.edu.au/~moshe/620-261/dijkstra/dijkstra.html > Accessed 4/25/09

Shortest Paths: Failed Attempts

Dijkstra. Can fail if **negative** edge costs.

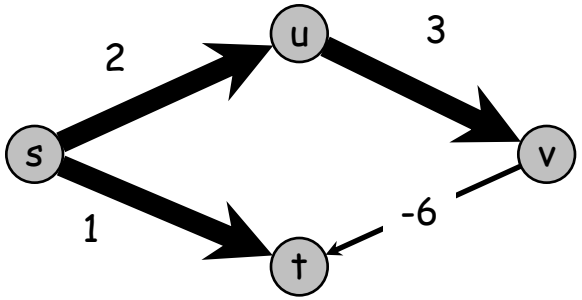


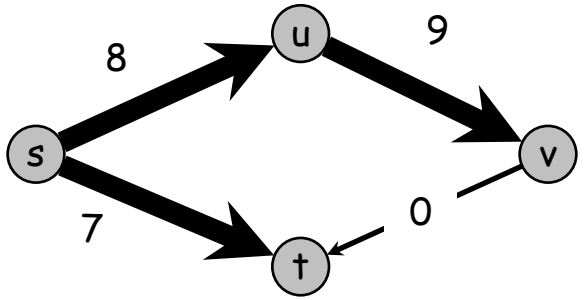
Re-weighting. Adding a constant to every edge weight can still fail.



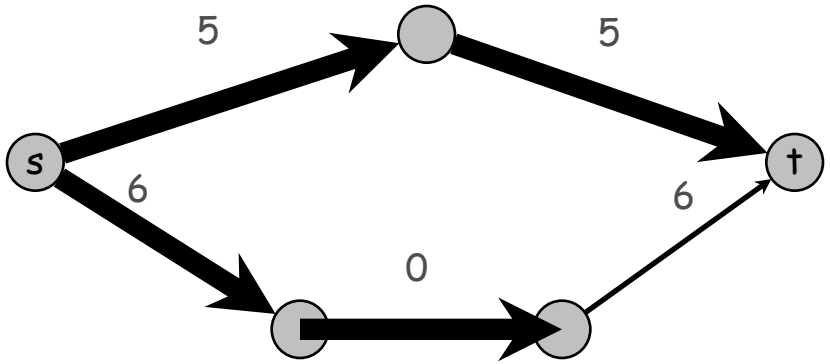
Shortest Paths: Failed Attempts

Dijkstra. Can fail if **negative** edge costs.



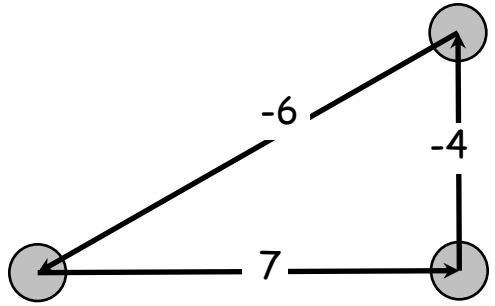


Re-weighting. Adding a constant to every edge weight can still fail.

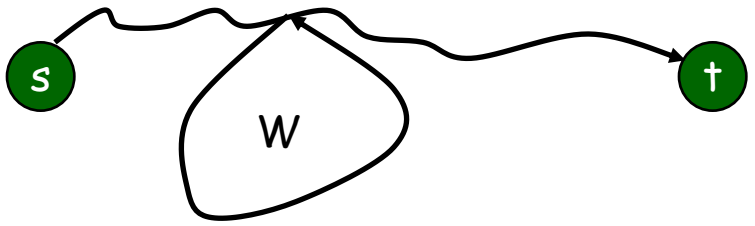


Shortest Paths: Negative Cost Cycles

Negative cost cycle.



Observation. If some path from s to t contains a negative cost cycle, there does not exist a shortest s-t path; otherwise, there exists one that is simple.



c(W) < 0

Bellman-Ford Algorithm

http://www.geeksforgeeks.org/dynamic-programming-set-23-bellman-ford-algorithm/

Can have negative-weight edges.
Will "detect" reachable negative-weight cycles.

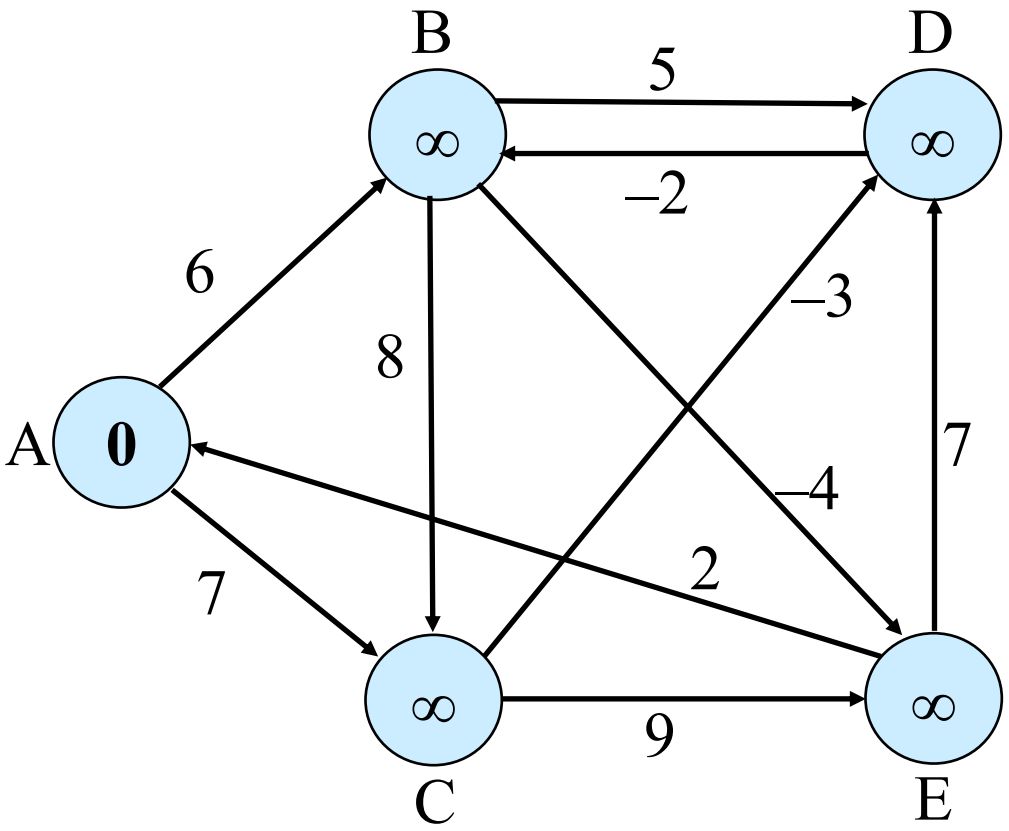
```
Initialize(G, s);
for i := 1 to V –1 do
    for each (u, v) in E do
        Relax(u, v, w)

for each (u, v) in E do
    if d[v] > d[u] + w(u, v) then
        return false
    fi
od;
return true
```

d[v]=min { d[v], d[u]+w(w,v) }

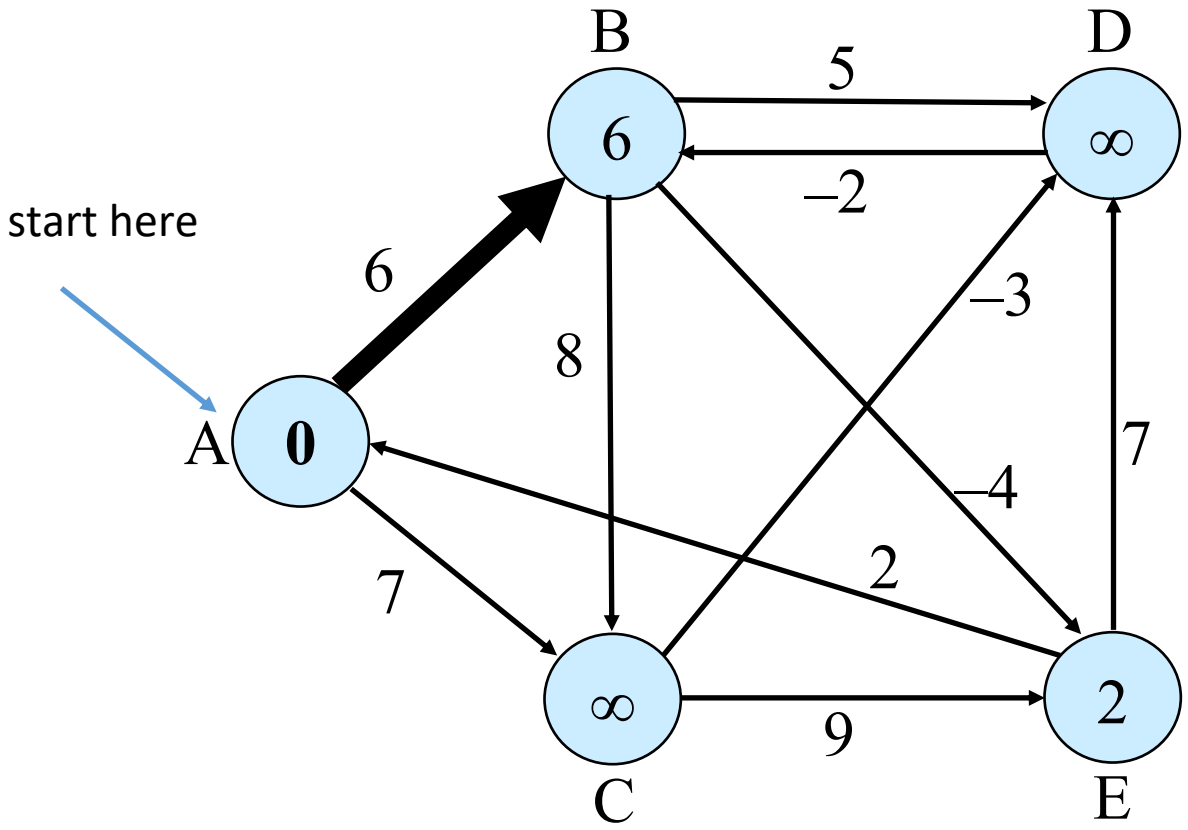
If Bellman-Ford has not converged after V(G) - 1 iterations, then there cannot be a shortest path tree, so there must be a negative weight cycle.

Time Complexity is O(VE).

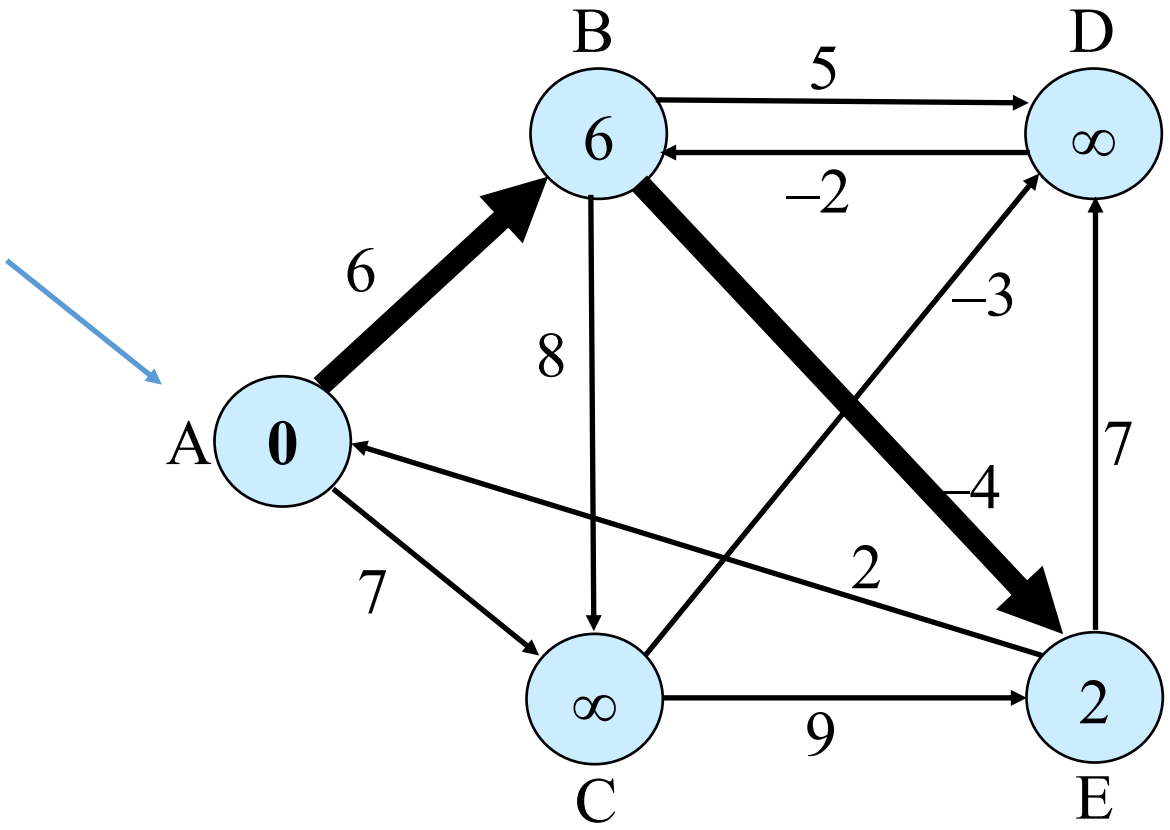
Example



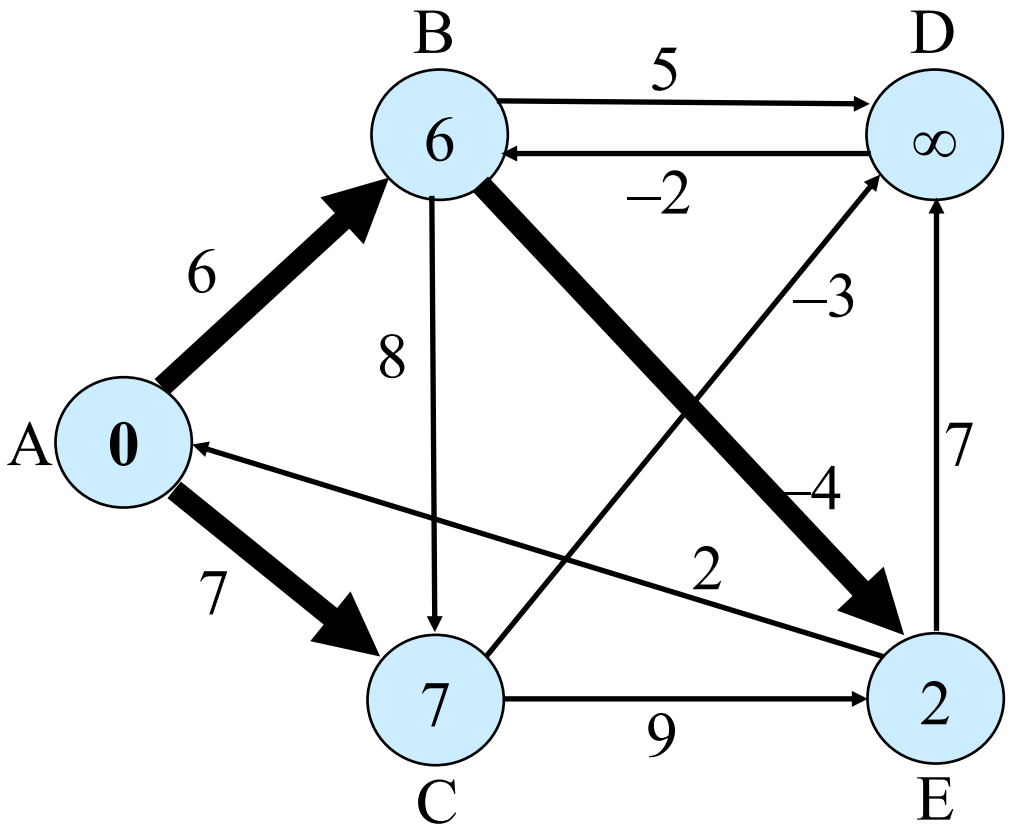
Example: Dijkstra's Algorithm Not Working



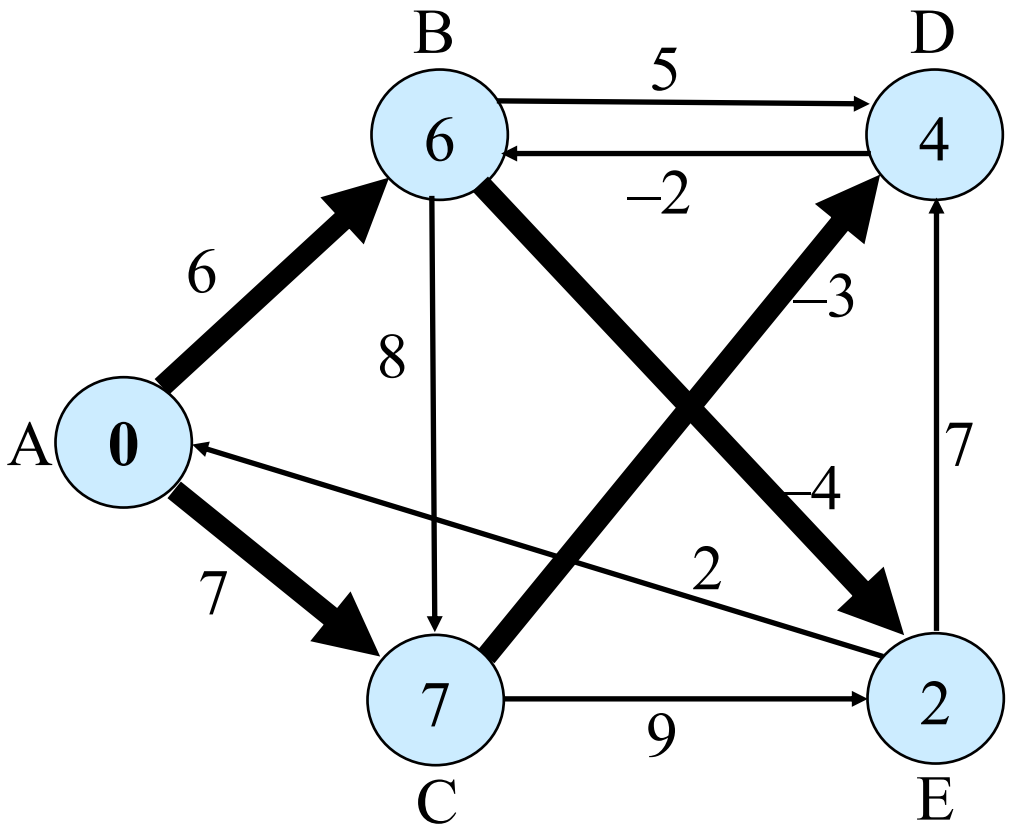
Example: Dijkstra's Algorithm Not Working



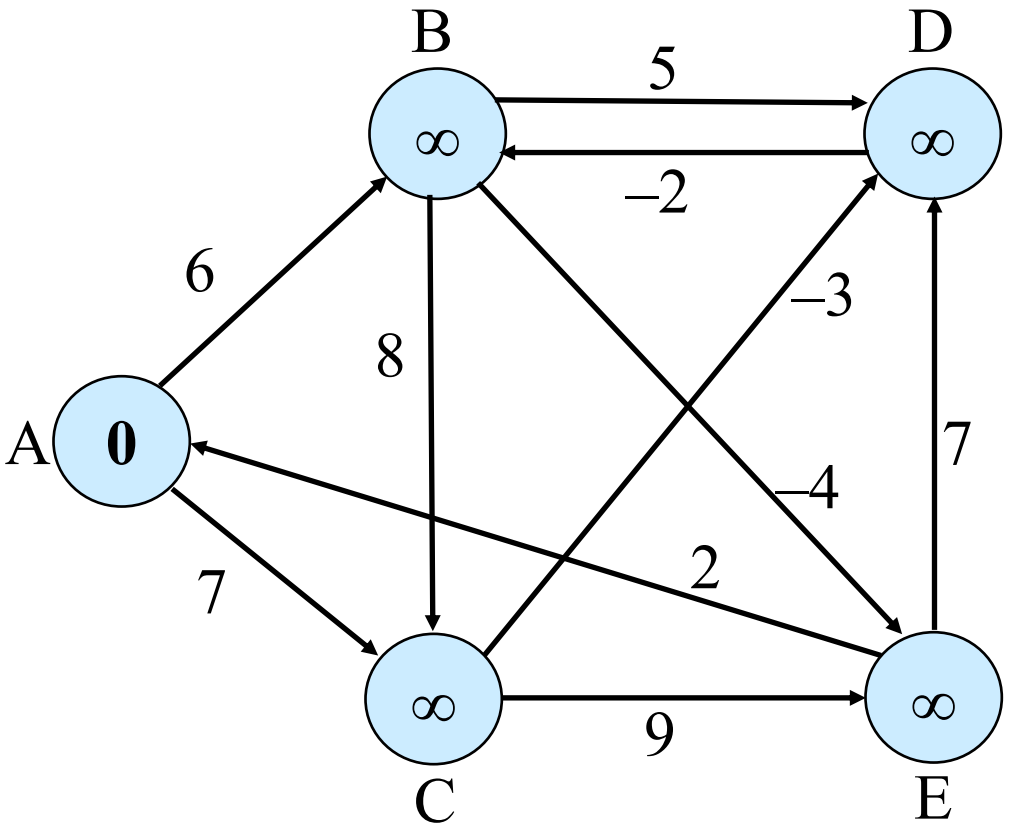
Example: Dijkstra's Algorithm Not Working



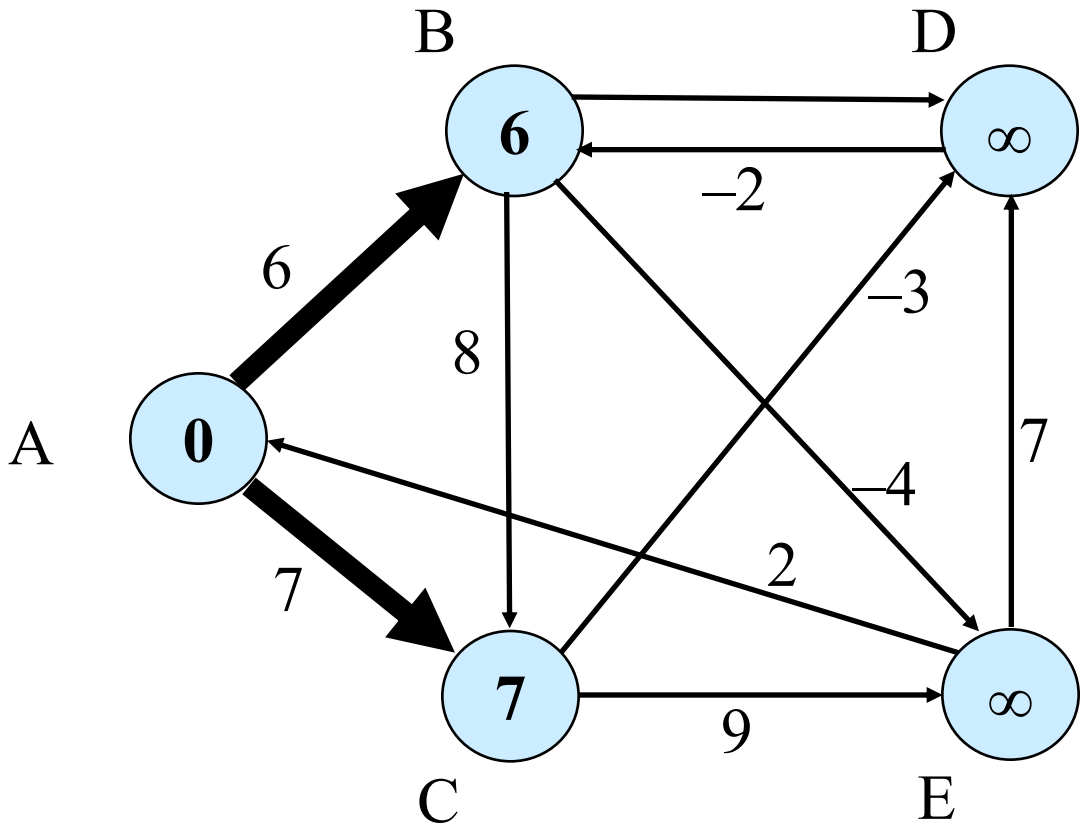
Example: Dijkstra's Algorithm Not Working



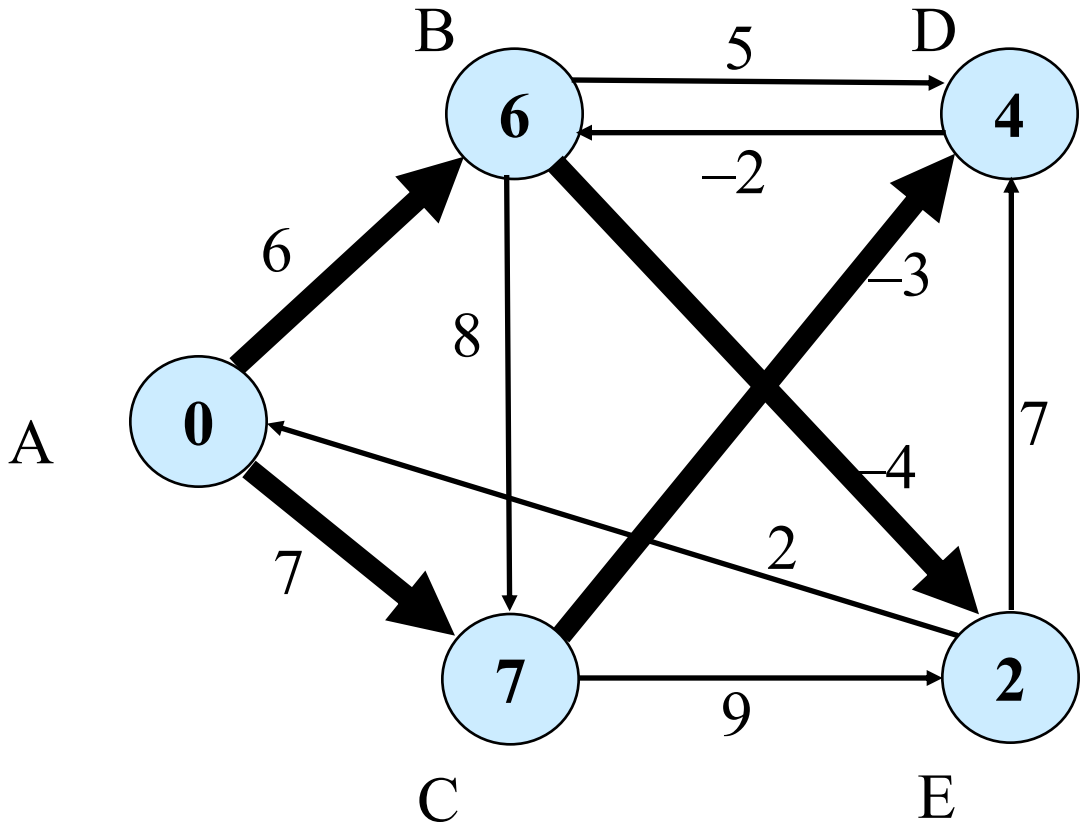
Example: Bellman-Ford



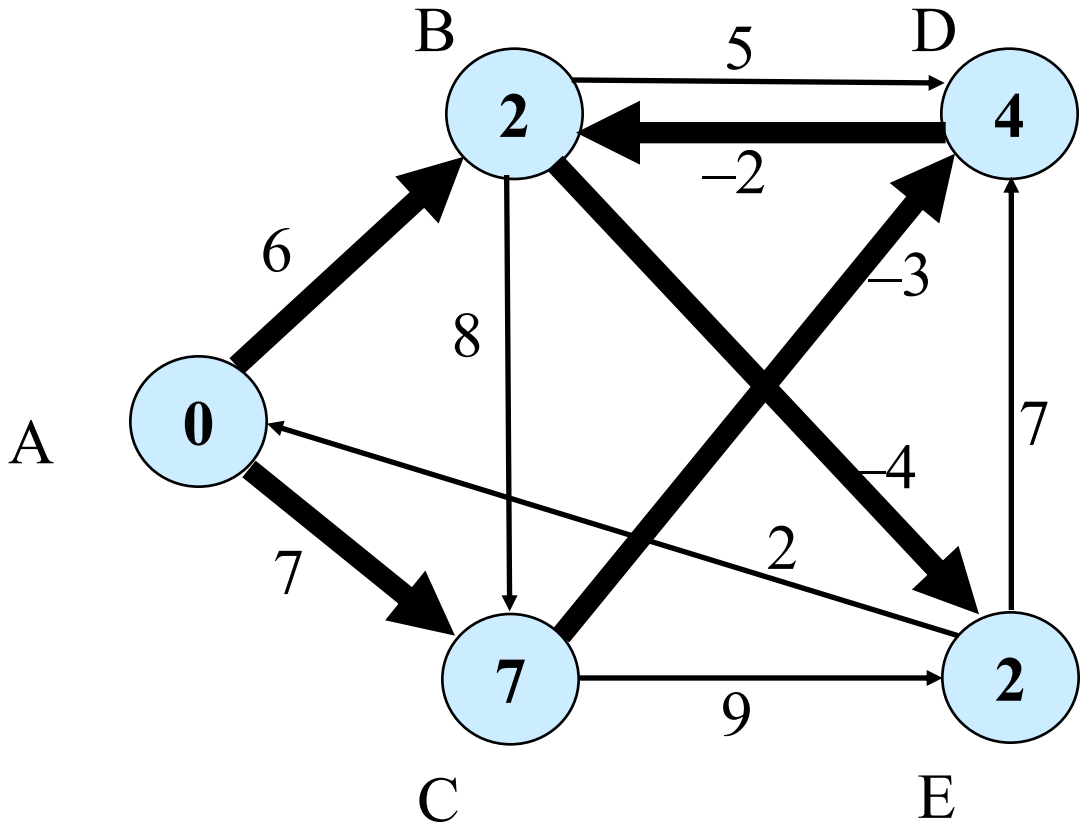
Example: Bellman-Ford



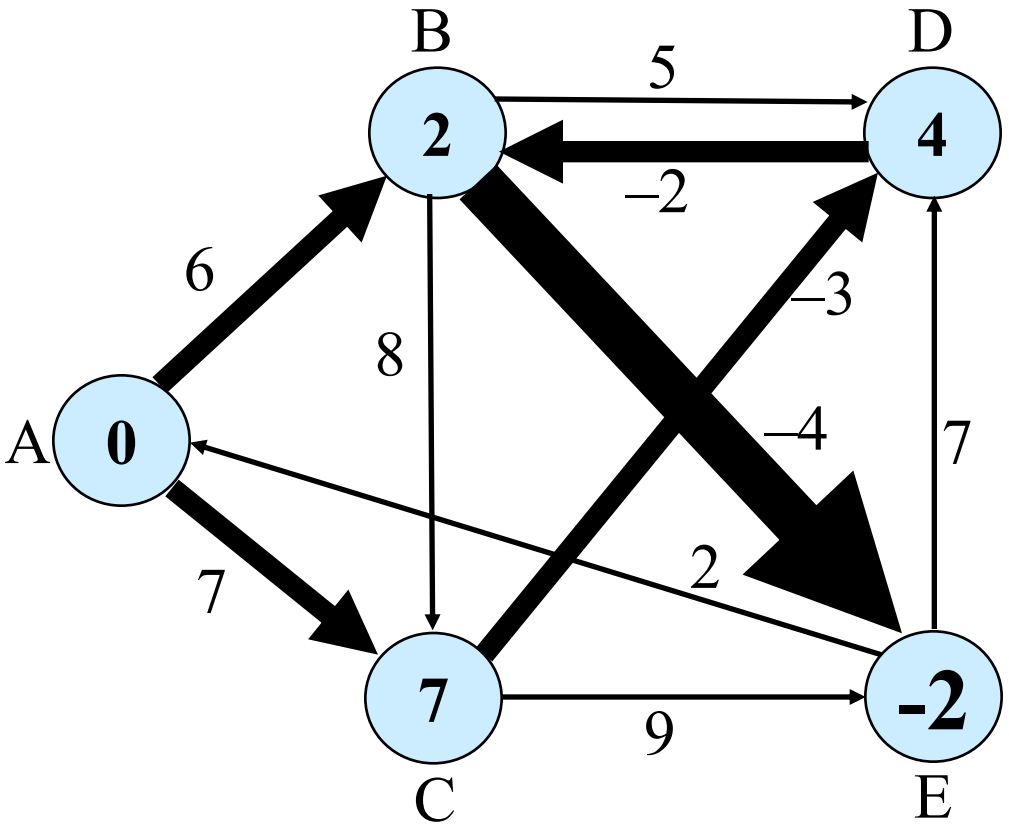
Example: Bellman-Ford



Example: Bellman-Ford



Example: Bellman-Ford



Another Look

Note: This is essentially **dynamic programming**.

Let d(i, j) = cost of the shortest path from s to i that is at most j hops.

$$d(i,j) = \begin{cases} 0 & \textbf{if } i = s \wedge j = 0 \\ \infty & \textbf{if } i \neq s \wedge j = 0 \\ \min(\{d(k, j-1) + w(k, i) : i \in \mathrm{Adj}(k)\} \cup \{d(i, j-1)\}) & \textbf{if } j > 0 \end{cases}$$

i →

j	z	u	v	x	y
	1	2	3	4	5
0	0	∞	∞	∞	∞
1	0	6	∞	7	∞
2	0	6	4	7	2
3	0	2	4	7	2
4	0	2	4	7	–2

j ↓

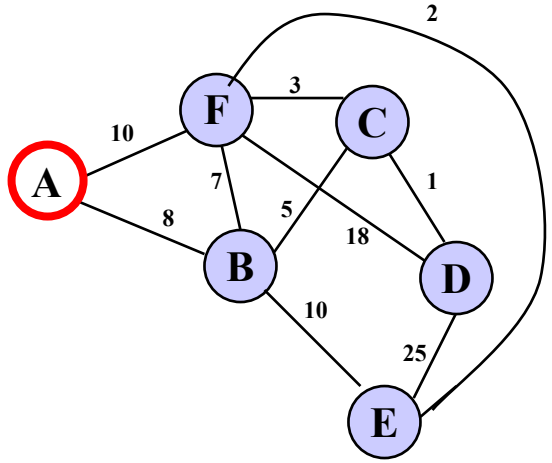
HOME WORK Review:
Prim’s, Kruskal’s, Dijkstra’s Algorithms

Prim's Algorithm

Prim's Algorithm: Fill in the following table. (1pt for each number)

	K	d_v	p_v	
A	T	0	-	
B				(1pt)
C				(1pt)
D				(1pt)
E				(1pt)
F				(1pt)
Total				(1pt)

Start with any node, say A



d_v = Cheapest edge **cost to connect node v to *tree T***

p_v = Node in T to which the cheapest edge is connected

Prim's Algorithm

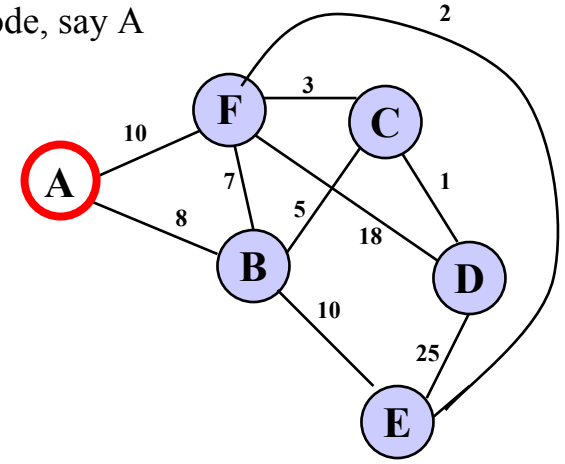
Home Work, Week 14, CS502, Design and Analysis of Algorithm, Spring 2016

Fill in the following table (5pt)

What is the cost of the minimum spanning tree? (1pt)

	K	d_v	p_v
A	T	0	-
B		8	A
C			
D			
E			
F		10	A
Total Cost			

Start with any node, say A

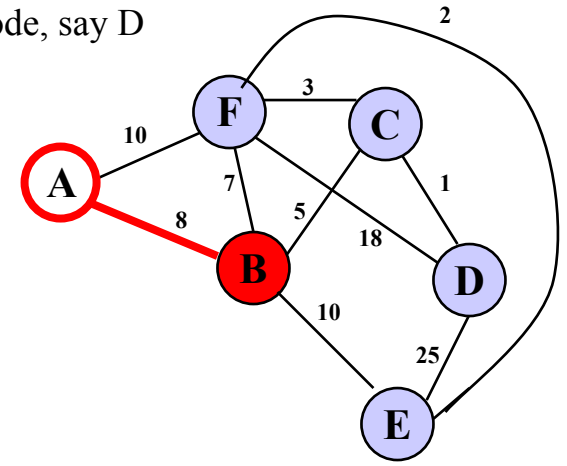


Prim's Algorithm

Home Work, Week 14, CS502, Design and Analysis of Algorithm, Spring 2016

	K	d_v	p_v
A	T	0	-
B	T	8	A
C		5	B
D			
E		10	B
F		7	B

Start with any node, say D

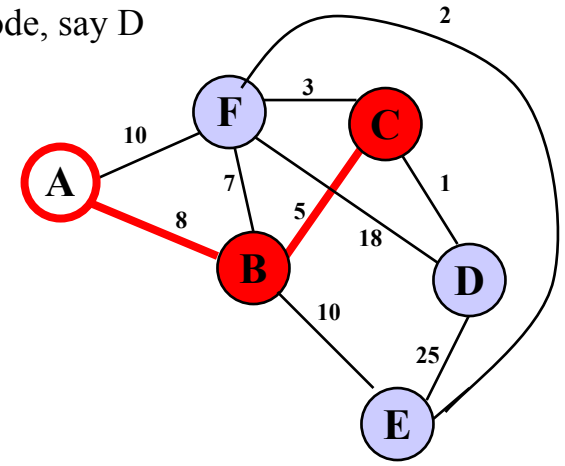


Prim's Algorithm

Home Work, Week 14, CS502, Design and Analysis of Algorithm, Spring 2016

	K	d_v	p_v
A	T	0	-
B	T	8	A
C	T	5	B
D		1	C
E		10	B
F		3	C

Start with any node, say D



Fill in the above table (5pt)

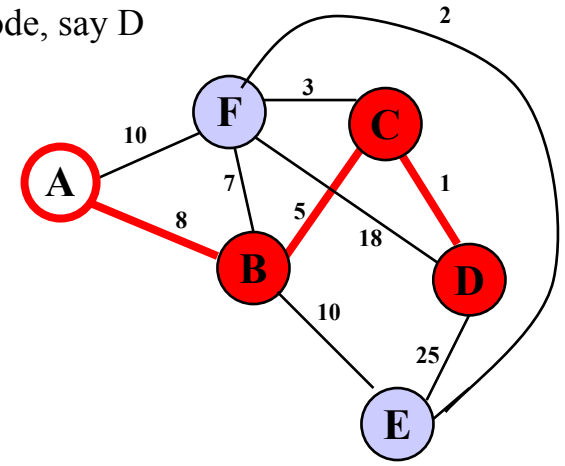
What is the cost of the minimum spanning tree? (1pt)

Prim's Algorithm

Home Work, Week 14, CS502, Design and Analysis of Algorithm, Spring 2016

	K	d_v	p_v
A	T	0	-
B	T	8	A
C	T	5	B
D	T	1	C
E		10	B
F		3	C

Start with any node, say D



Fill in the above table (5pt)

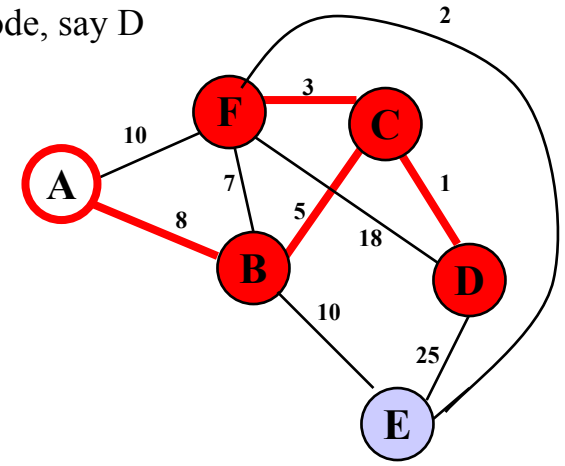
What is the cost of the minimum spanning tree? (1pt)

Prim's Algorithm

Home Work, Week 14, CS502, Design and Analysis of Algorithm, Spring 2016

	K	d_v	p_v
A	T	0	-
B	T	8	A
C	T	5	B
D	T	1	C
E		2	F
F	T	3	C

Start with any node, say D



Fill in the above table (5pt)

What is the cost of the minimum spanning tree? (1pt)

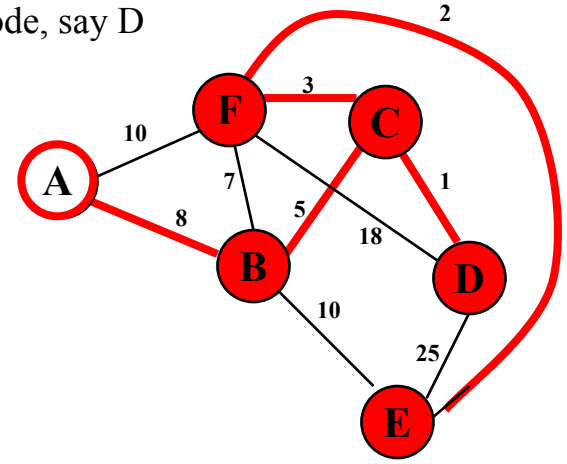
Prim's Algorithm

Home Work, Week 14, CS502, Design and Analysis of Algorithm, Spring 2016

Fill in the following table, and answer the questions.

	K	d_v	p_v	
A	T	0	-	
B	T	8	A	(1pt)
C	T	5	B	(1pt)
D	T	1	C	(1pt)
E	T	2	F	(1pt)
F	T	3	C	(1pt)

Start with any node, say D

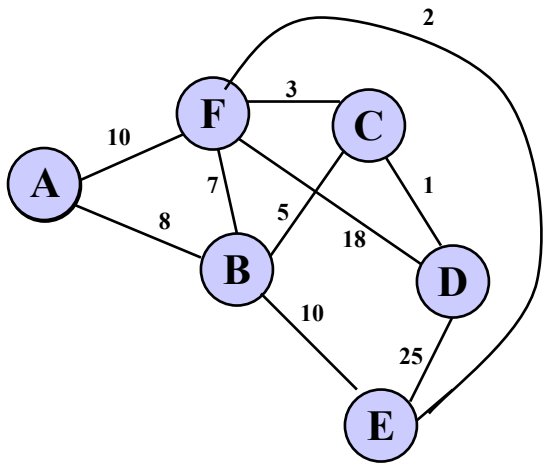


What is the cost of the minimum spanning tree? (1pt) 19

Kruskal's Algorithm

Kruskal's Algorithm

Kruskal's Algorithm: Fill in the following table. (1pt for each number)



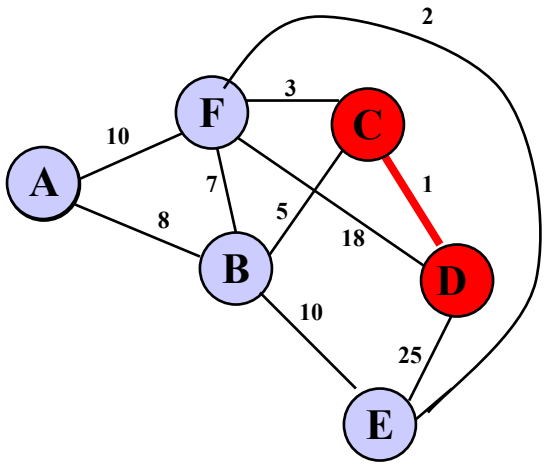
edge	*cost*	*OK?*
(C,D)	1	
(E,F)	2	
(F,C)	3	
(B,C)	5	
(B,F)	7	

Sort the edges by increasing edge weight

edge	*cost*	*ok?*
(A,B)	8	
(A,F)	10	
(B,E)	10	
(F,D)	18	
(D,E)	25	

Krusal's Algorithm

Kruskal's Algorithm: Fill in the following table. (1pt for each number)



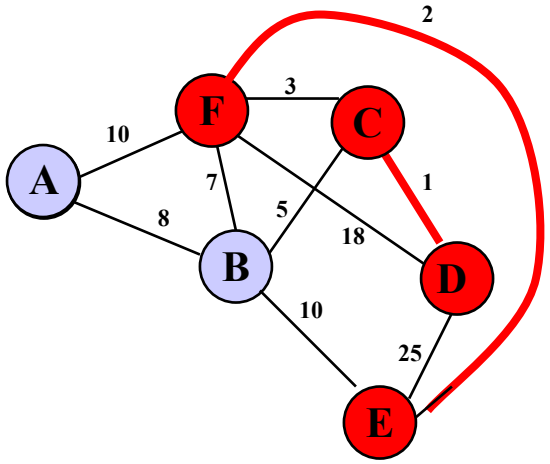
edge	*cost*	
(C,D)	1	√
(E,F)	2	
(F,C)	3	
(B,C)	5	
(B,F)	7	

edge	*cost*	
(A,B)	8	
(A,F)	10	
(B,E)	10	
(F,D)	18	
(D,E)	25	

Krusal's Algorithm

Home Work, Week 14, CS502, Design and Analysis of Algorithm, Spring 2016

Kruskal's Algorithm: Fill in the following table. (1pt for each number)



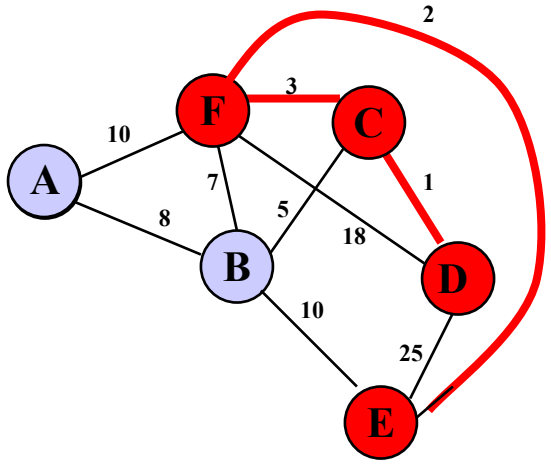
edge	*cost*	
(C,D)	1	√
(E,F)	2	√
(F,C)	3	
(B,C)	5	
(B,F)	7	

edge	*cost*	
(A,B)	8	
(A,F)	10	
(B,E)	10	
(F,D)	18	
(D,E)	25	

Krusal's Algorithm

Home Work, Week 14, CS502, Design and Analysis of Algorithm, Spring 2016

Kruskal's Algorithm: Fill in the following table. (1pt for each number)



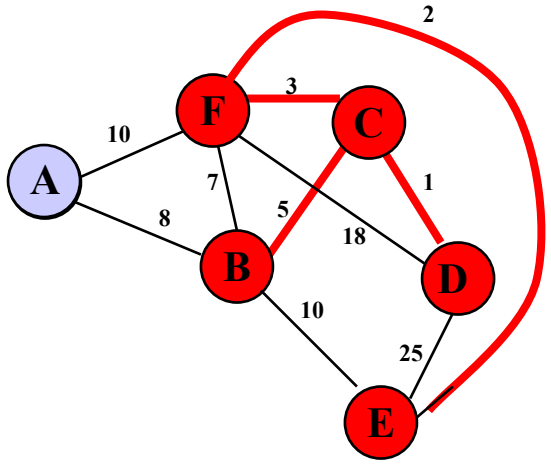
edge	*cost*	
(C,D)	1	√
(E,F)	2	√
(F,C)	3	√
(B,C)	5	
(B,F)	7	

edge	*cost*	
(A,B)	8	
(A,F)	10	
(B,E)	10	
(F,D)	18	
(D,E)	25	

Krusal's Algorithm

Home Work, Week 14, CS502, Design and Analysis of Algorithm, Spring 2016

Kruskal's Algorithm: Fill in the following table. (1pt for each number)



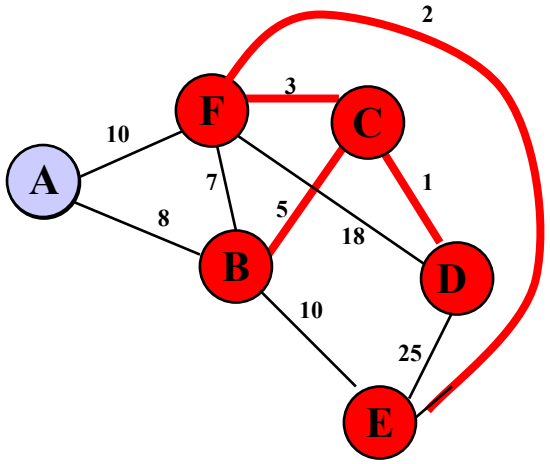
edge	*cost*	
(C,D)	1	√
(E,F)	2	√
(F,C)	3	√
(B,C)	5	√
(B,F)	7	

edge	*cost*	
(A,B)	8	
(A,F)	10	
(B,E)	10	
(F,D)	18	
(D,E)	25	

Krusal's Algorithm

Home Work, Week 14, CS502, Design and Analysis of Algorithm, Spring 2016

Kruskal's Algorithm: Fill in the following table. (1pt for each number)



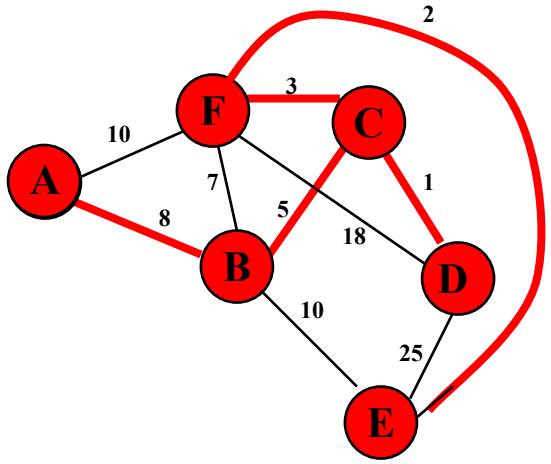
edge	*cost*	
(C,D)	1	√
(E,F)	2	√
(F,C)	3	√
(B,C)	5	√
(B,F)	7	χ

edge	*cost*	
(A,B)	8	
(A,F)	10	
(B,E)	10	
(F,D)	18	
(D,E)	25	

Krusal's Algorithm

Home Work, Week 14, CS502, Design and Analysis of Algorithm, Spring 2016

Kruskal's Algorithm: Fill in the following table. (1pt for each number)



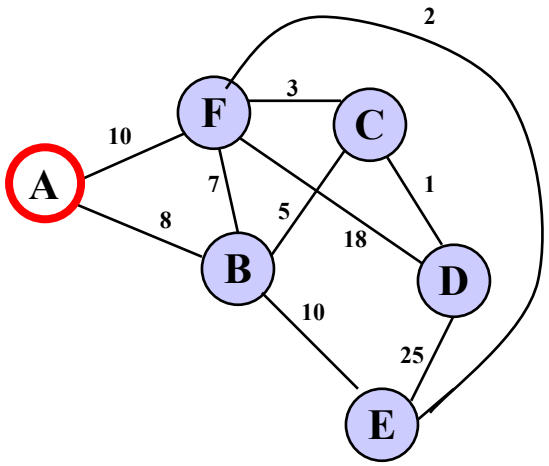
edge	*cost*	
(C,D)	1	√
(E,F)	2	√
(F,C)	3	√
(B,C)	5	√
(B,F)	7	χ

edge	*cost*	
(A,B)	8	√
(A,F)	10	
(B,E)	10	
(F,D)	18	
(D,E)	25	

Dijkstra's Algorithm

Dijkstra's Algorithm

Dijkstra's Algorithm: Fill in the table. (1pt for each number)

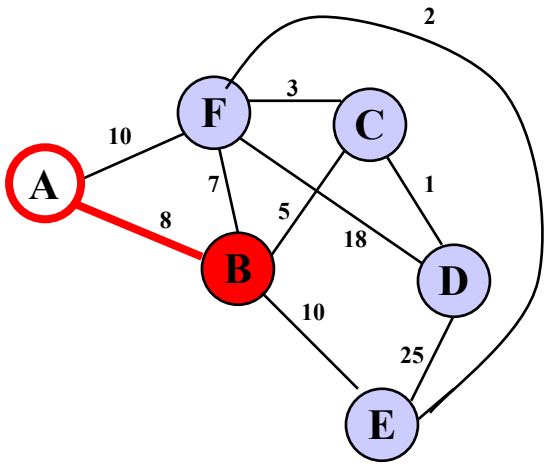


	K	d_v	p_v
A	T	0	-
B		8	A
C			
D			
E			
F		10	A

d_v = **Shortest path length from source node *s* to *v***

p_v = Node in T to which Node v is connected

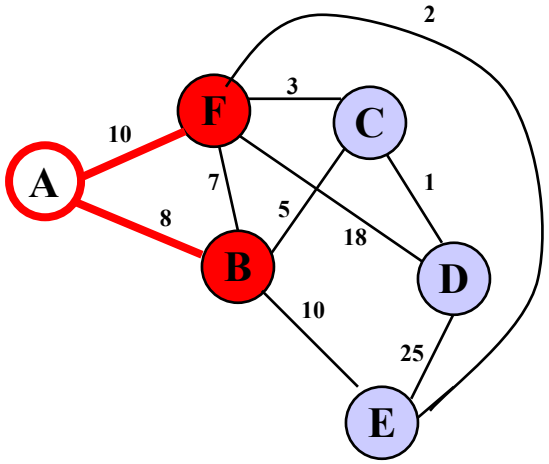
Dijkstra's Algorithm



	K	d_v	p_v
A	T	0	-
B	T	8	A
C		**13**	B
D			
E		**18**	B
F		**10**	A

Dijkstra's Algorithm

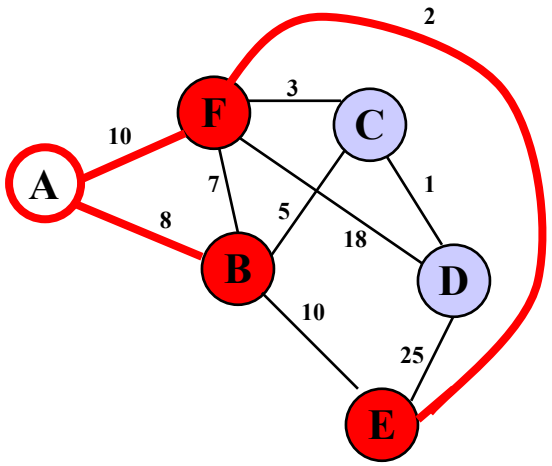
Home Work, Week 14, CS502, Design and Analysis of Algorithm, Spring 2016



	K	d_v	p_v
A	T	0	-
B	T	8	A
C		**13**	B
D		**28**	F
E		**12**	F
F	T	10	A

Dijkstra's Algorithm

Home Work, Week 14, CS502, Design and Analysis of Algorithm, Spring 2016



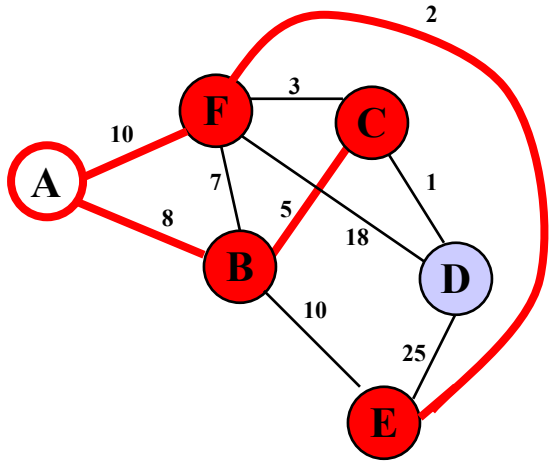
	K	d_v	p_v
A	T	0	-
B	T	8	A
C		13	B
D		28	F
E	T	12	F
F	T	10	A

Dijkstra's Algorithm

Home Work, Week 14, CS502, Design and Analysis of Algorithm, Spring 2016

Fill in the above table (5pt)

What is the cost of the minimum spanning tree? (1pt)



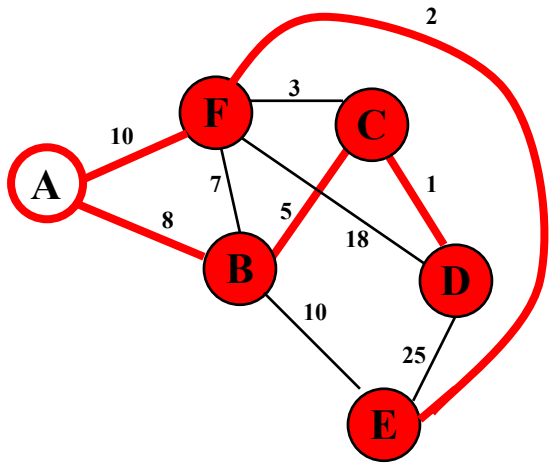
	K	d_v	p_v
A	T	0	-
B	T	8	A
C	T	13	B
D		14	C
E	T	12	F
F	T	10	A

Dijkstra's Algorithm

Home Work, Week 14, CS502, Design and Analysis of Algorithm, Spring 2016

Fill in the above table (5pt)

What is the cost of the minimum spanning tree? (1pt) 26



	K	d_v	p_v	
A	T	0	-	
B	T	8	A	(1pt)
C	T	13	B	(1pt)
D	T	14	C	(1pt)
E	T	12	F	(1pt)
F	T	10	A	

```
0-1  1-2 2-3 3-4
2-5 2-8 8-6 6-7
```