SLIIT ACADEMY

BSc (IT) – Year 2, Semester 2

Design and Analysis of Algorithms
Greedy Algorithm

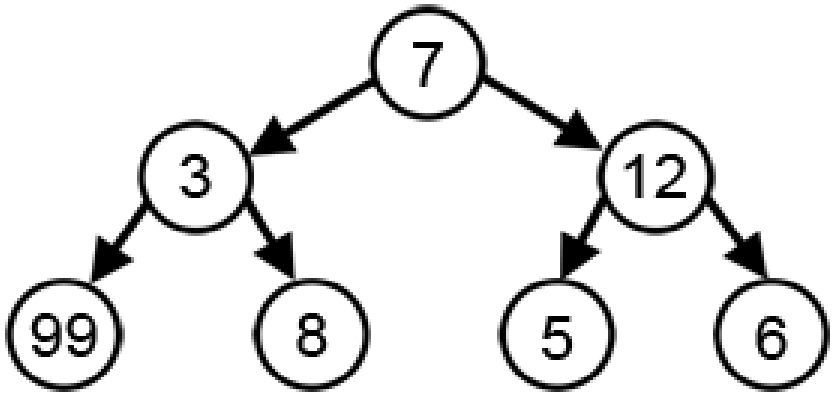
Anuruddha Abeysinghe
anuruddha.a@sliit.lk

Contents

- Greedy Method
- Greedy Graph Algorithms
  - Kruskal's Algorithm - To find MCST
  - Prim's Algorithm - To find MCST
  - Dijkstra's Algorithm - To find Shortest Paths

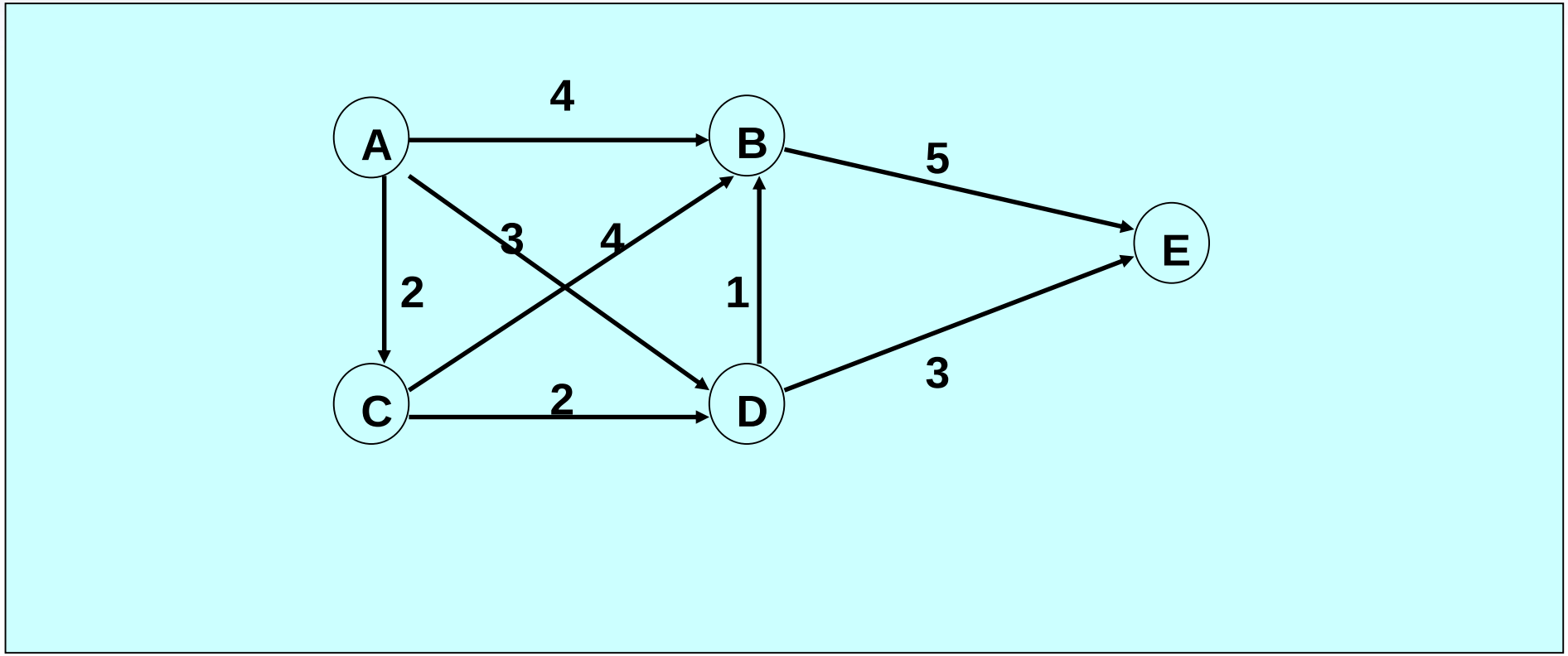
Greedy Method

Follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum solution.



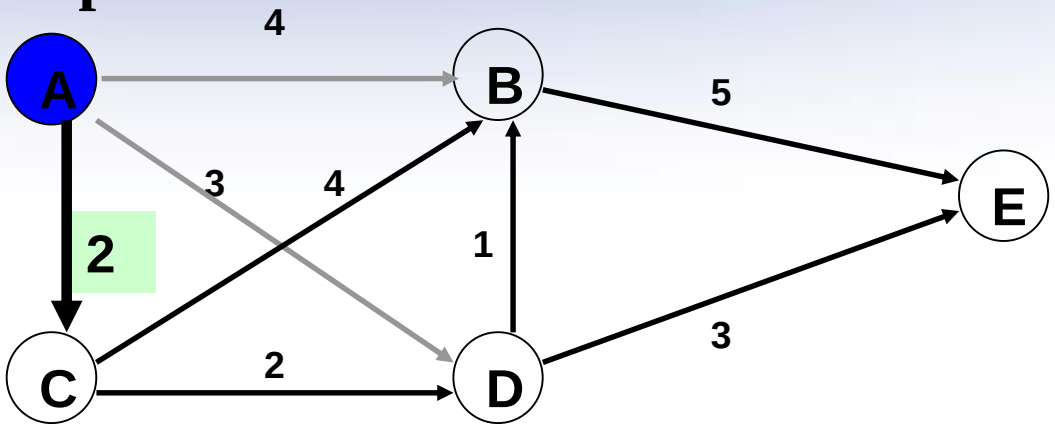
What is Greedy Method?

- Find the shortest path from ***A* to *E*** using Greedy method.
- **Greedy method**:Makes the choice that looks best at the moment

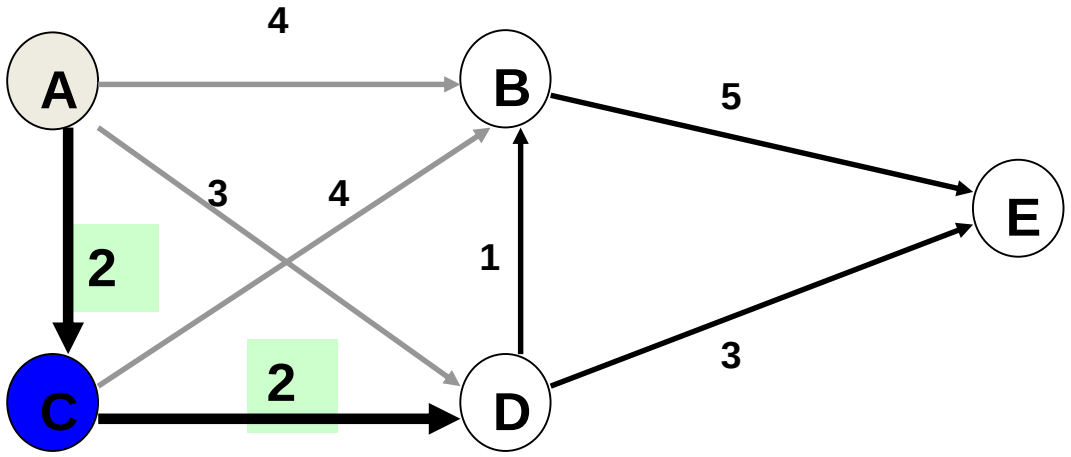


Solution

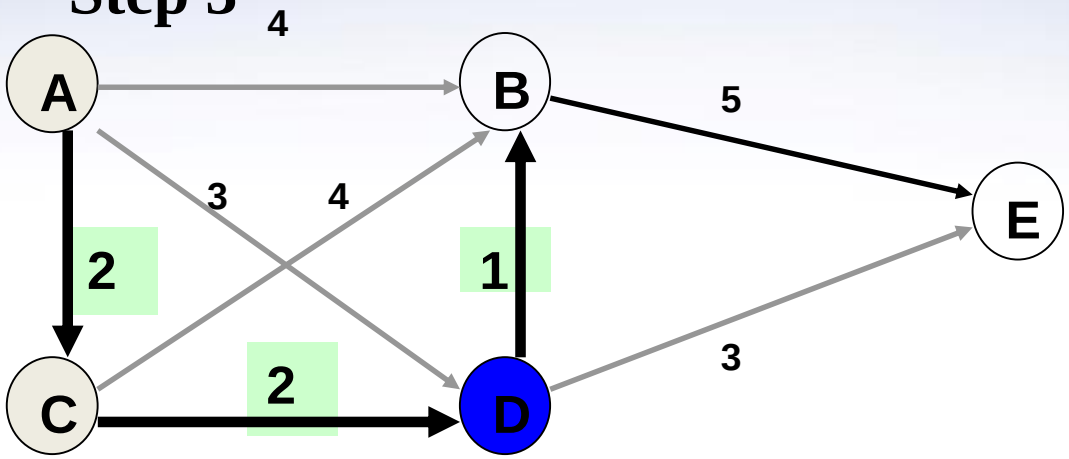
Step 1



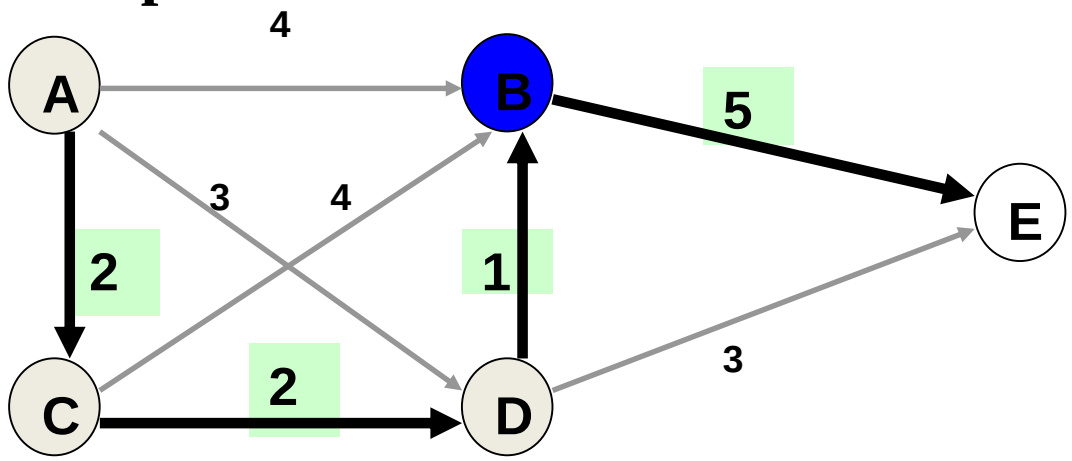
Step 2



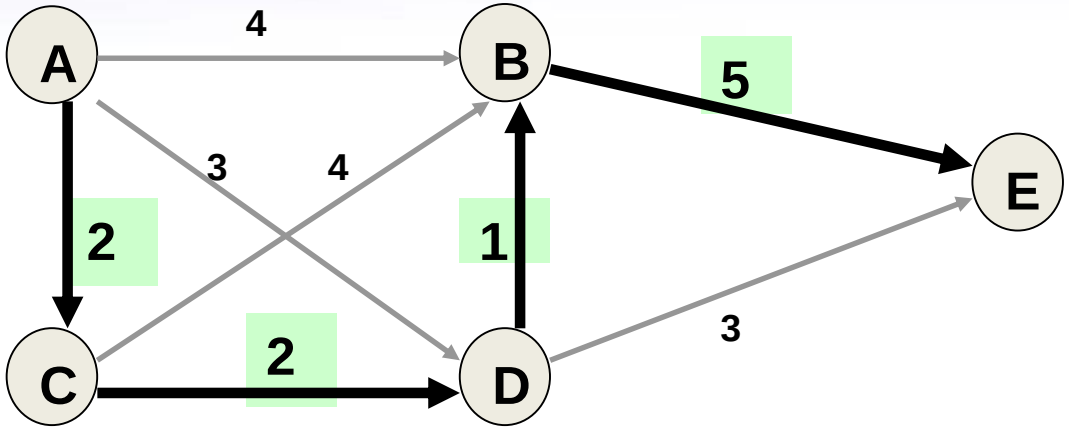
Step 3



Step 4



Final Result is:



- Order of visit is A→ C → D → B→ E path cost is 2+2+1+5 = **10**

Greedy Method(contd.)

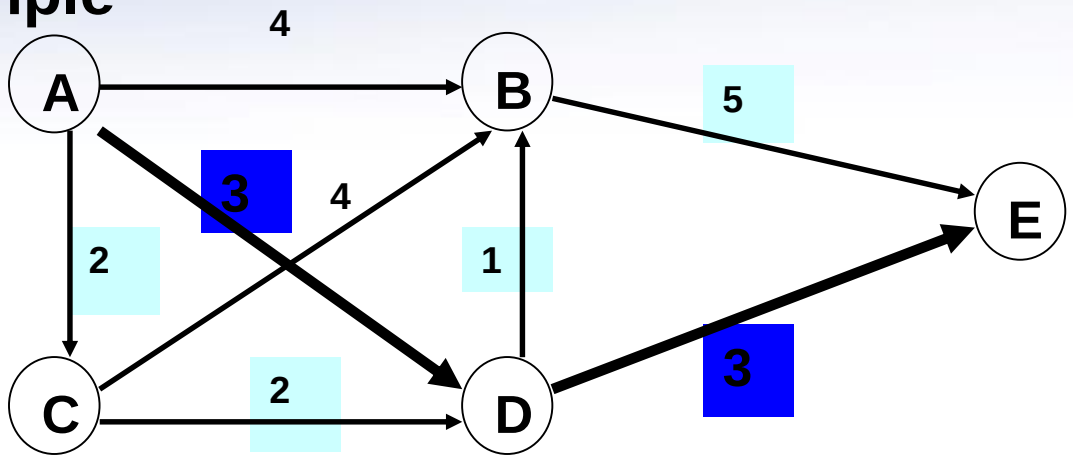
- Makes the choice that looks best at the moment
- Makes locally optimal choice hoping that it will lead to globally optimal solution.
- Do not always yield optimal solutions
- Powerful and works well for many problems.

What is mean by Optimal Solution?

- Optimal mean the best we can achieve.
- If it is a maximization problem the optimal solution mean the maximum value we can achieve.
- If the problem is a minimization problem optimal solution mean the minimum value we can achieve.
- Check whether, in above example it yields optimal solution

Greedy method does not yield optimal solution in always.

Example



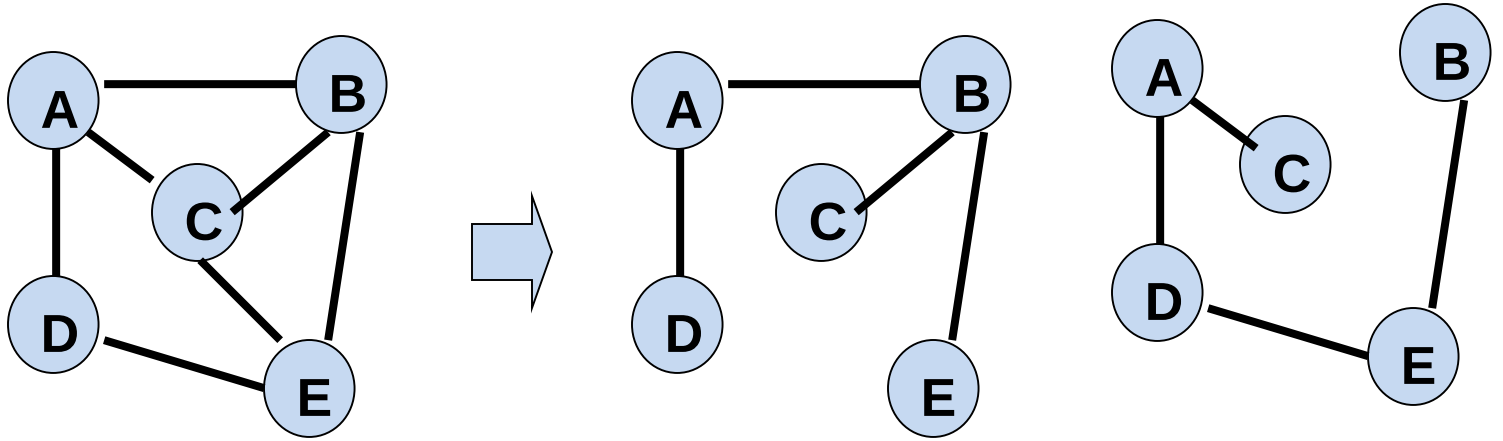
- Here the optimal solution is A →D →E path cost is **3+3 = 6** and it is the minimum value
- Greedy method solution is A→ C → D → B→ E path cost is 2+2+1+5 = 10

Greedy Algorithm

- Kruskal's Algorithm - To find *MCST*
- Prim's Algorithm - To find *MCST*
- Dijkstra's Algorithm - To find *Shortest Paths*

What is a Spanning Tree?

- **Tree** : A connected undirected graph that contains no cycles is called a tree.
- **Spanning Tree** : A spanning tree of a graph G is a subgraph of G that is a tree and contains all the vertices of G.



Undirected Graph

some Spanning Trees

Properties of spanning Tree

- The spanning tree of a ***n* –vertex undirected graph has exactly *n*-1 edges**.
- Spanning tree has **no cycles**
- It **Connects all vertexes in the graph**.

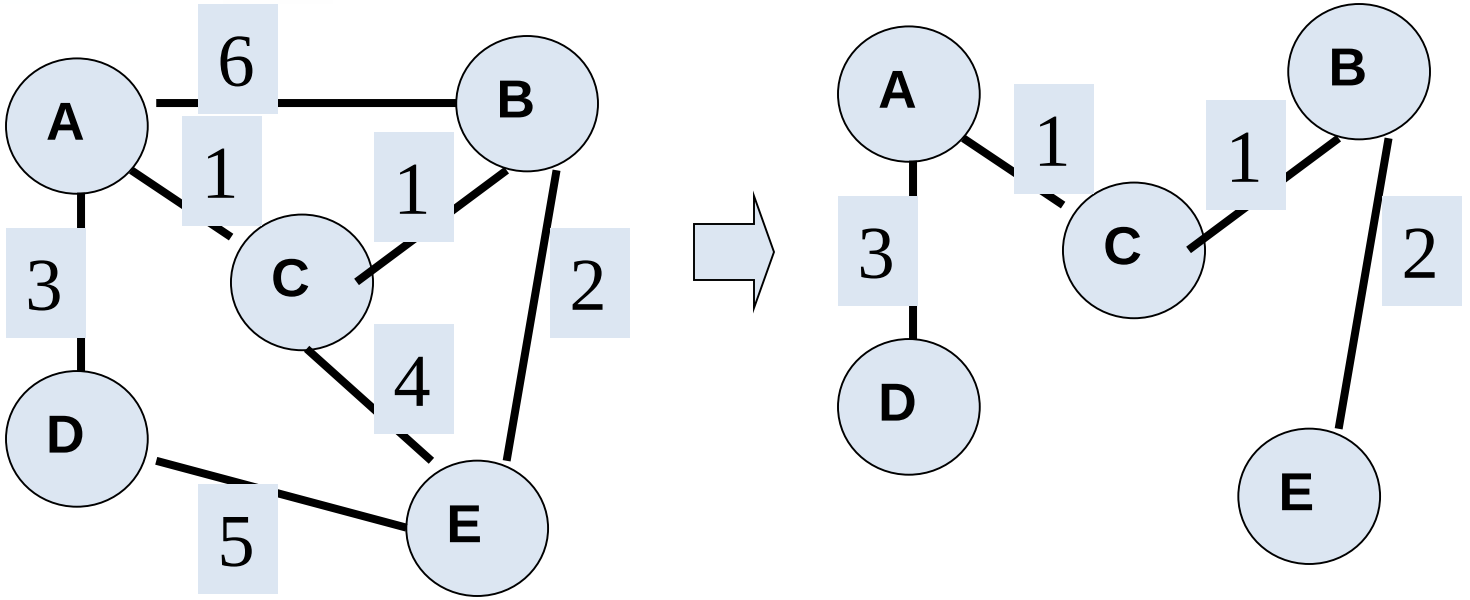
- Input: A connected, undirected graph G = (V, E)

  with weight function w : E → R.
- Output: A spanning tree T —a tree that connects all vertices of minimum weight:

$$w(T) = \sum_{(u,v)\in T} w(u,v)$$

What is a Minimum Cost Spanning Tree?

- **M**inimum **C**ost **S**panning **T**ree is the tree amongst all spanning trees with the smallest cost



Weighted Undirected Graph

MCST

Where do we use MCST?

- Computer Networks

  Minimum spanning trees are useful in constructing networks, by describing the way to connect a set of sites using the smallest total amount of wire.

- Ship/ Aero plane lines

  we can connect all the cities using the MCST and using MCST, will reduce amount of cost & time to travel.

- In the design of electronic circuitry

  it is often necessary to make the pins of several components electrically equivalent by wiring them together. To interconnect a set of n pins, we can use an arrangement of n - 1 wires, each connecting two pins. Of all such arrangements, the one that uses the least amount of wire is usually the most desirable

How to build MCST?

- Two Greedy Algorithms
  -Kruskal's algorithm
  - Prim's algorithm

Growing trees with the greedy method

INPUT: Connected weighted graph *G*

While no more edges

1. Take edge of least weight
2. If it will not make a cycle in *T*, add it to *T*

OUTPUT: *T*

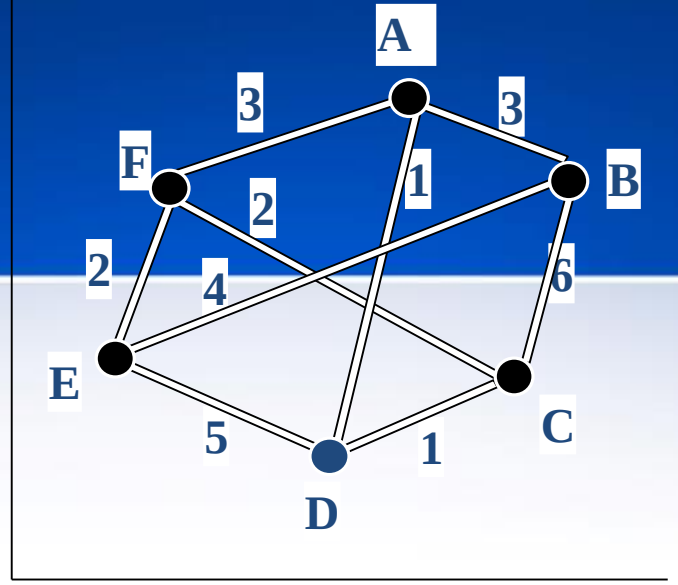
Kruskal's algorithm

Input: An undirected graph $G(V,E)$ with a cost function c on the edges

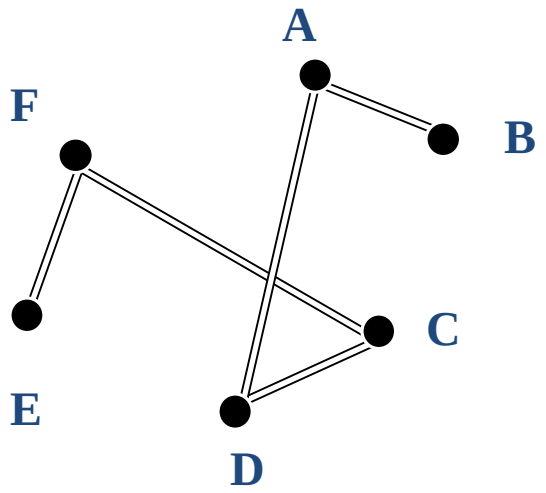
Output: T the minimum cost spanning tree for G

```
MST-KRUSKAL(G, w)
1 A ← Ø
2 for each vertex v ∈ V[G]
3         do MAKE-SET (v)
4 sort the edges of E into non decreasing order by weight w
5 for each edge (u, v) ∈ E, taken in non decreasing order by weight
6         do if FIND-SET (u) ≠ FIND-SET (v)
7                 then A ← A ∈ {(u, v)}
8                         UNION(u, v)
9 return A
```

Example 01:



			{A} {B} {C} {D} {E} {F}
1	(A,D)	merge	{A,D} {B} {C} {E} {F}
1	(C,D)	merge	{A,C,D} {B} {E} {F}
2	(C,F)	merge	{A,C,D,F} {B} {E}
2	(E,F)	merge	{A,C,D,E,F} {B}
3	(A,F)	reject	{A,C,D,E,F} {B}
3	(A,B)	merge	{A,C,D,E,F,B}
4	(B,E)	reject	{A,C,D,E,F,B}
5	(D,E)	reject	{A,C,D,E,F,B}
6	(B,C)	reject	{A,C,D,E,F,B}

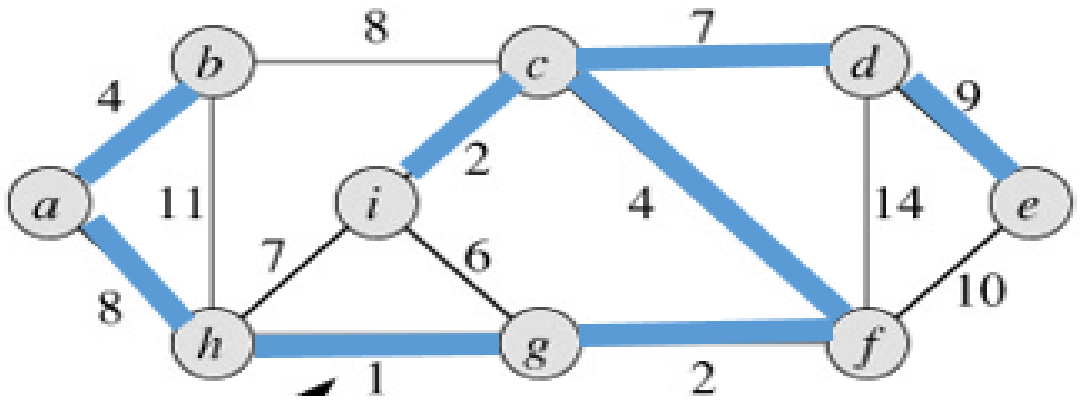


Kruskal's algorithm(contd.)

- Each vertex is labeled as being in a set
- Initially each is in it's own set
- Each set of vertices forms a MCST for the subgraph it connects
- It is safe to join two vertices from different sets
- The edge that joins the sets goes into *T*
- Note that Kruskal's algorithm is greedy:

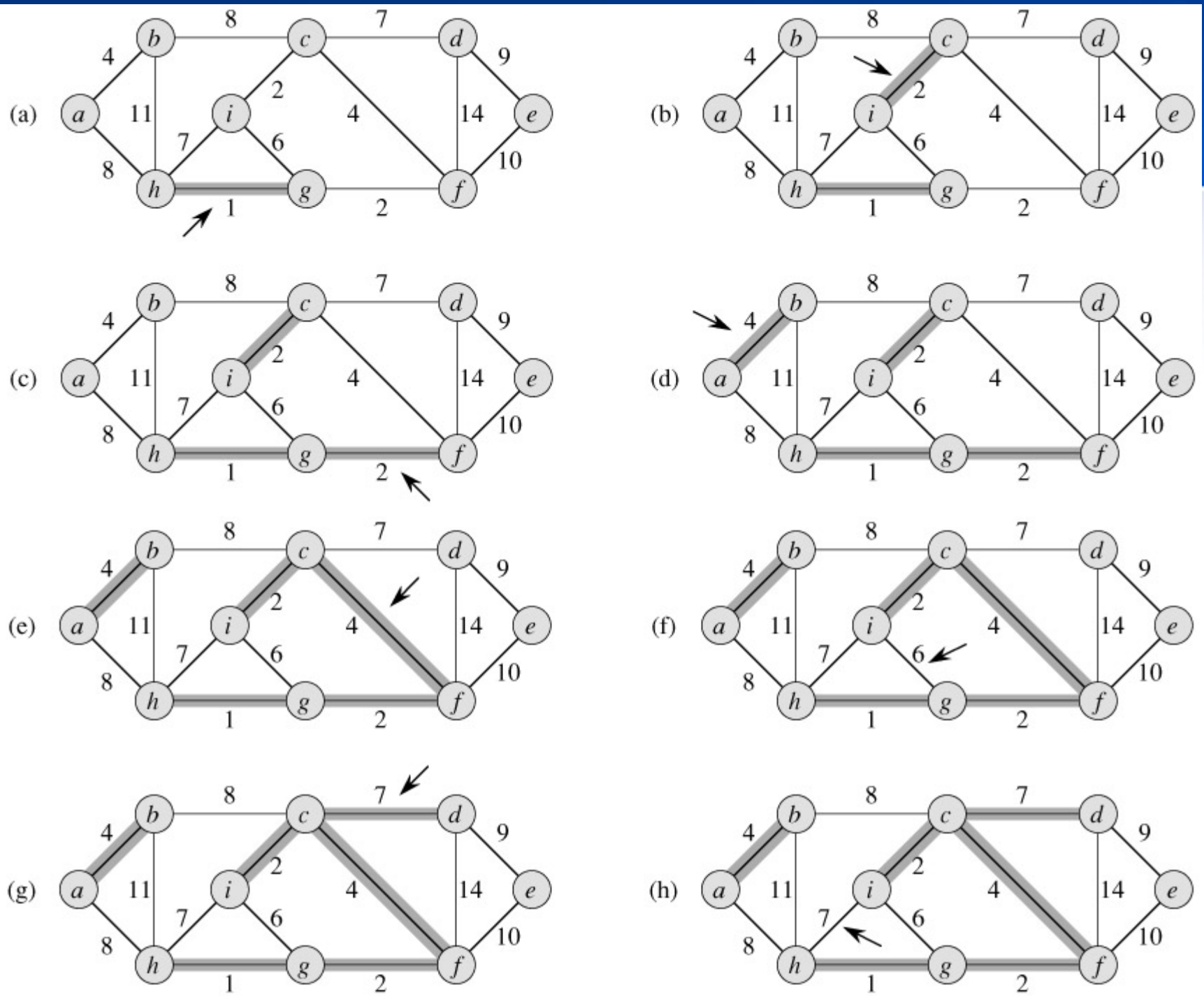
  Just adds a shortest edge at the moment without worrying about the overall structure
- MCST resulting from this algorithm is optimal

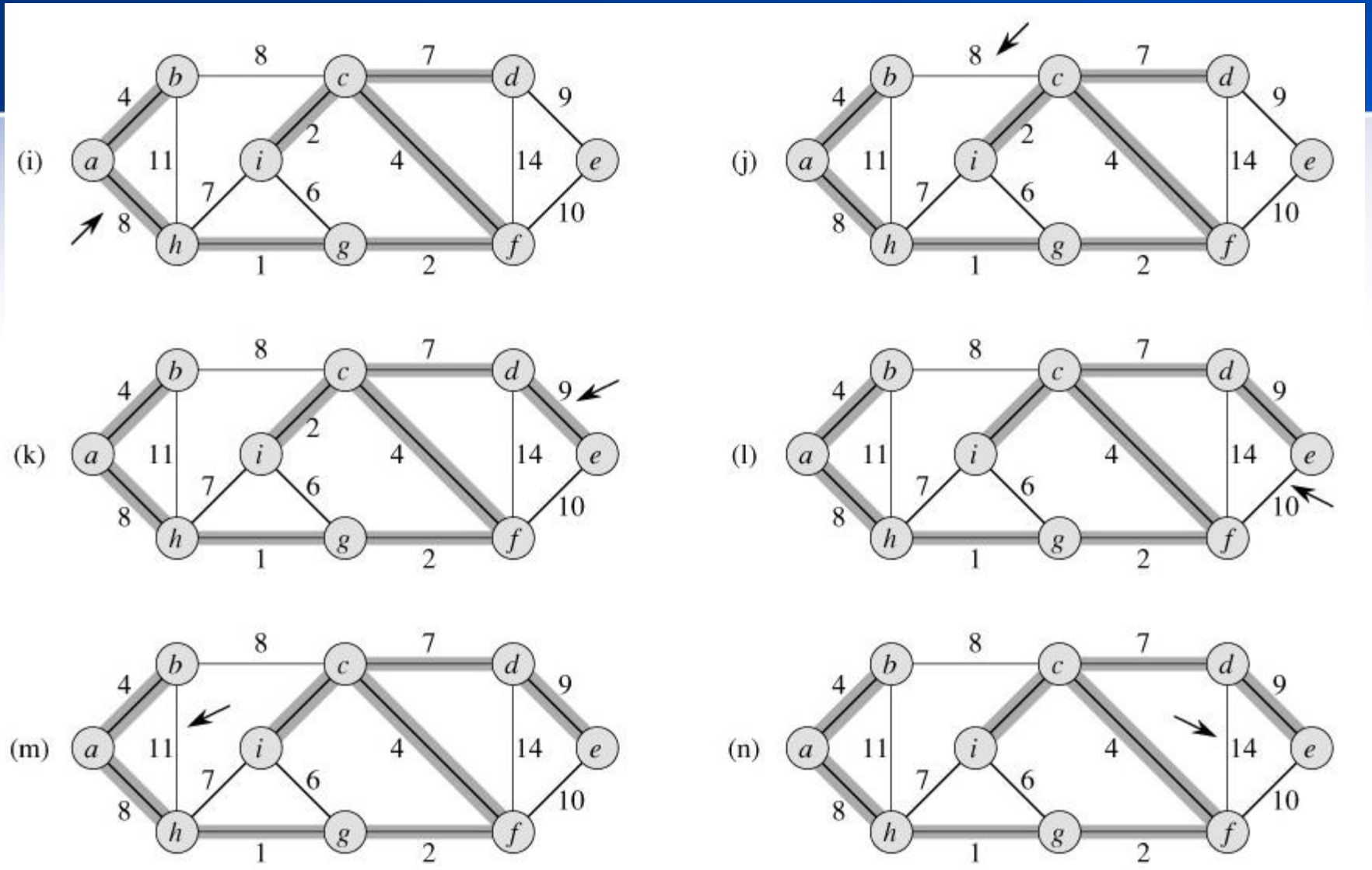
Example 02:



{a} {b} {c} {d} {e} {f} {g} {h} {i}

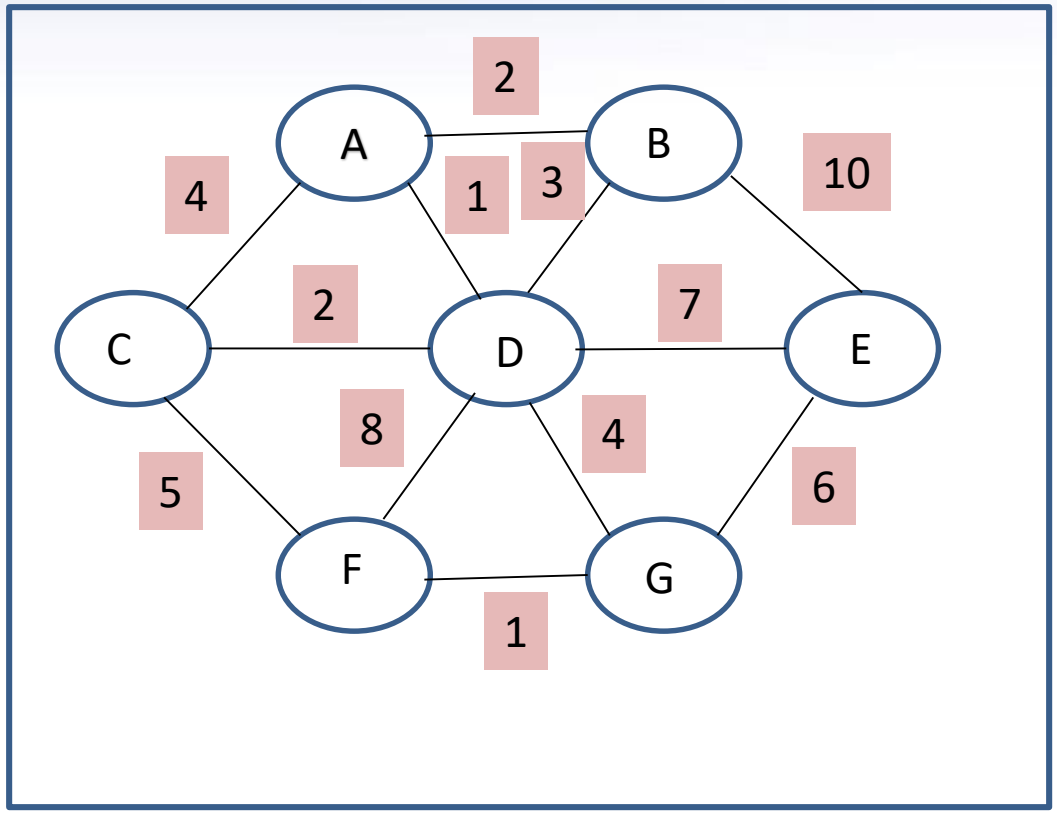
• (h,g) *1 – merge	{g,h} {a} {b} {c} {d} {e} {f} {i}
• (c,i) *2 – merge	{c,i} {g,h} {a} {b} {d} {e} {f}
• (g,f) *2 –merge	{c,i} {f,g,h} {a} {b} {d} {e}
• (a,b) *4 –merge	{a,b} {c,i} {f,g,h} {d} {e}
• (c,f) *4 –merge	{a,b} {c,f,g,h,i} {d} {e}
• (i,g) *6 – *reject*	{a,b} {c,f,g,h,i} {d} {e}
• (c,d) *7 – merge	{a,b} {c,d,f,g,h,i} {e}
• (i,h) *7 – *reject*	{a,b} {c,d,f,g,h,i} {e}
• (a,h) *8 – merge	{a,b,c,d,f,g,h,i} {e}
• (b,c) *8 – *reject*	{a,b,c,d,f,g,h,i} {e}
• (d,e) *9 – merge	{a,b,c,d,e,f,g,h,i}
• (f,e) *10 – *reject*	{a,b,c,d,e,f,g,h,i}
• (b,h) *11 – *reject*	{a,b,c,d,e,f,g,h,i}
• (d,f) *14 – *reject*	{a,b,c,d,e,f,g,h,i}


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

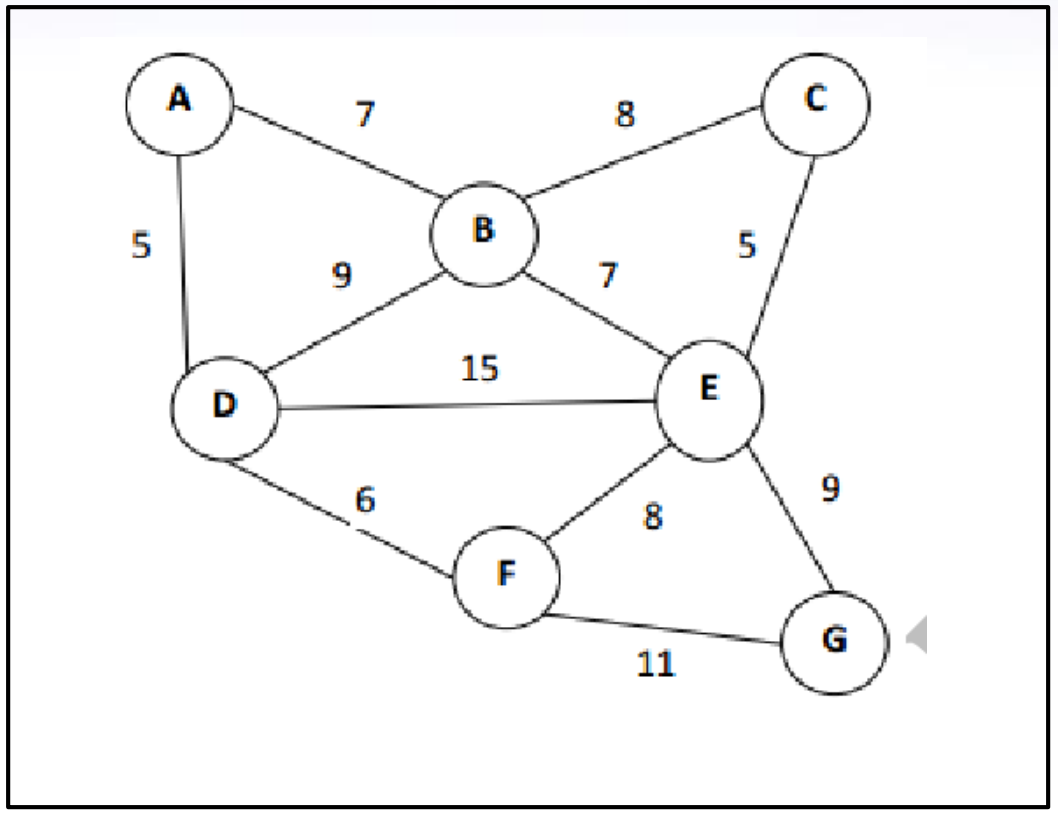

(i)
(j)
(k)
(l)
(m)
(n)
a
b
c
d
e
f
g
h
i

Exercise

- Final the Minimum Cost weight of following graphs.



Question 01



Question 02 – 2015 Pass paper question

Prim's algorithm

- The property that the edges in the set A always form a single tree. The tree starts from an arbitrary root vertex r and grows until the tree spans all the vertices in V. At each step, a light edge is added to the tree A that connects A to an isolated vertex of $GA = (V, A)$.

- The key to efficiently is to make it easy to select a new edge to be added to the tree formed by the edges in A.

- In the pseudocode below, the connected graph G and the root r of the minimum spanning tree to be grown are inputs to the algorithm. During execution of the algorithm, all vertices that are *not* in the tree reside in a min-priority queue Q based on a *key* field.

- For each vertex v, $key[v]$ is the minimum weight of any edge connecting v to a vertex in the tree; by convention, $key[v] = \infty$ if there is no such edge. The field $\pi[v]$ names the parent of v in the tree

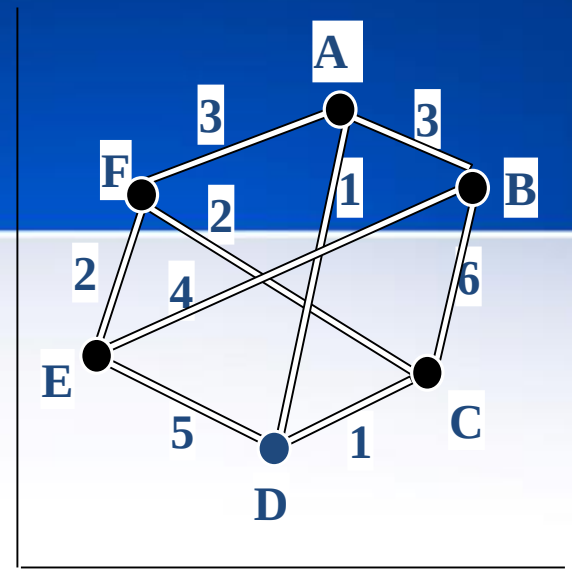
Prim's algorithm

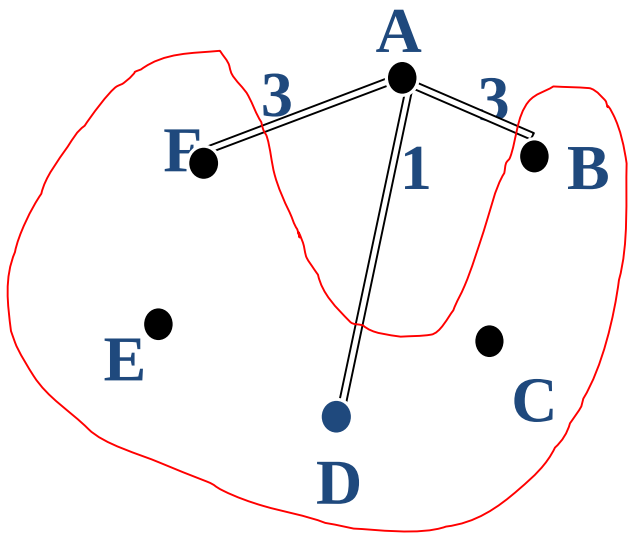
```
MST-PRIM(G, w, r)
1 for each u ∈ V [G]
2     do key[u] ← ∞
3           π[u] ← NIL
4 key[r] ← 0
5 Q ← V [G]
6  while Q ≠ Ø
7       do u ← EXTRACT-MIN(Q)
8             for each v ∈ Adj[u]
9                 do if v ∈ Q and w(u, v) < key[v]
10                      then π[v] ← u
11                   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

example

Choose vertex A, V={B,C,D,E,F}

Candidate edges = (A,F) (A,D) (A,B)





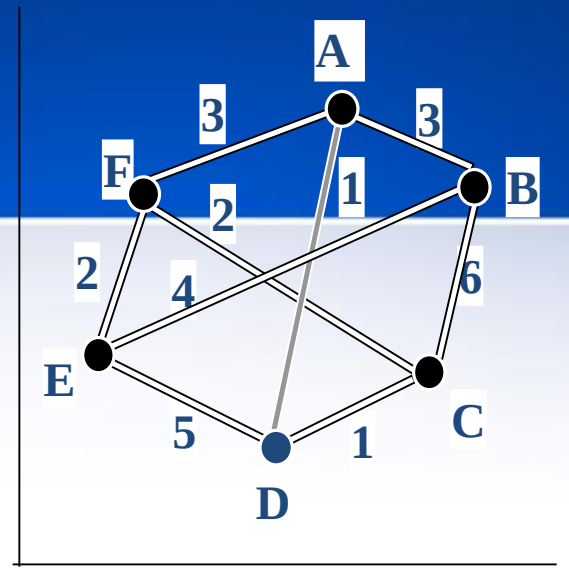
example

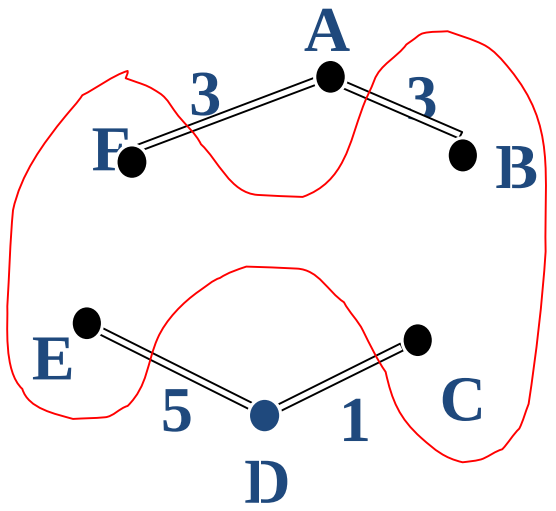
Choose vertex A, V={B,C,D,E,F}

Candidate edges = (A,F) (A,D) (A,B)

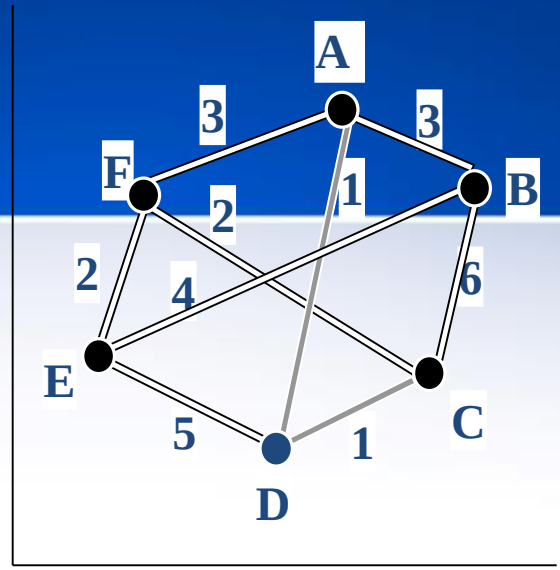
Choose edge (A,D) V={B,C,E,F}

Candidate edges = (A,F) (A,B) (D,C) (D,E)





example



Choose vertex A, V={B,C,D,E,F}

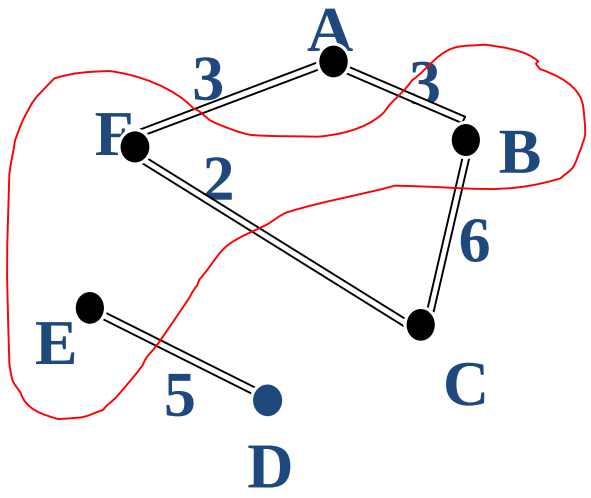
Candidate edges = (A,F) (A,D) (A,B)

Choose edge (A,D) V={B,C,E,F}

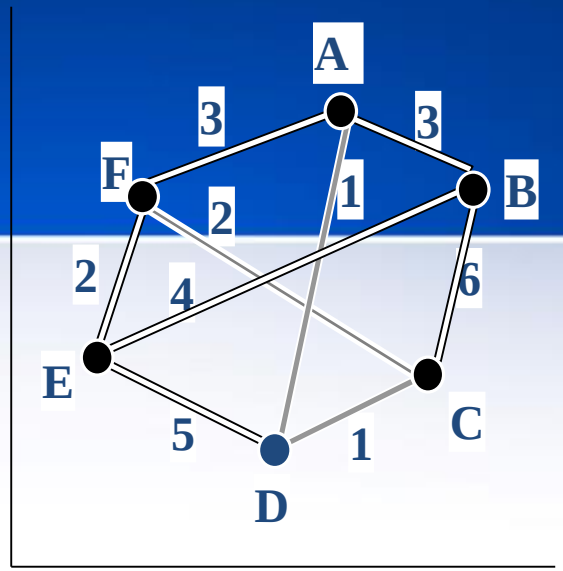
Candidate edges = (A,F) (A,B) (D,C) (D,E)

Choose edge (D,C) V={B,E,F}

Candidate edges = (A,F) (A,B) (D,E) (C,F) (C,B)



example



Choose vertex A, V={B,C,D,E,F}

Candidate edges = (A,F) (A,D) (A,B)

Choose edge (A,D) V={B,C,E,F}

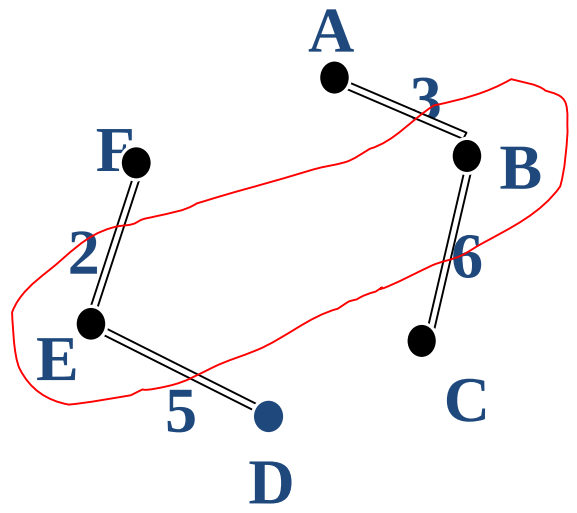
Candidate edges = (A,F) (A,B) (D,C) (D,E)

Choose edge (D,C) V={B,E,F}

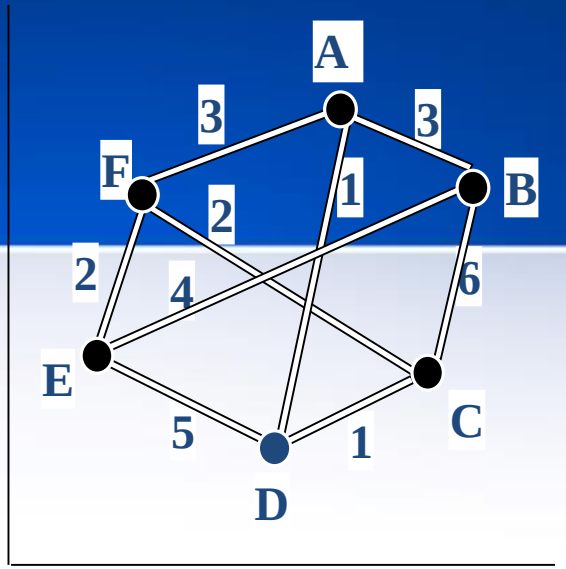
Candidate edges = (A,F) (A,B) (D,E) (C,F) (C,B)

Choose edge (F,C) V={B,E}

Candidate edges = (A,B) (D,E) (C,B) (E,F)



example



Choose vertex A, V={B,C,D,E,F}

Candidate edges = (A,F) (A,D) (A,B)

Choose edge (A,D) V={B,C,E,F}

Candidate edges = (A,F) (A,B) (D,C) (D,E)

Choose edge (D,C) V={B,E,F}

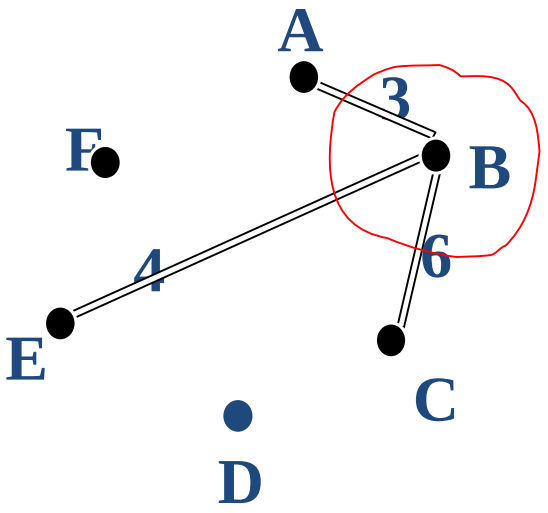
Candidate edges = (A,F) (A,B) (D,E) (C,F) (C,B)

Choose edge (F,C) V={B,E}

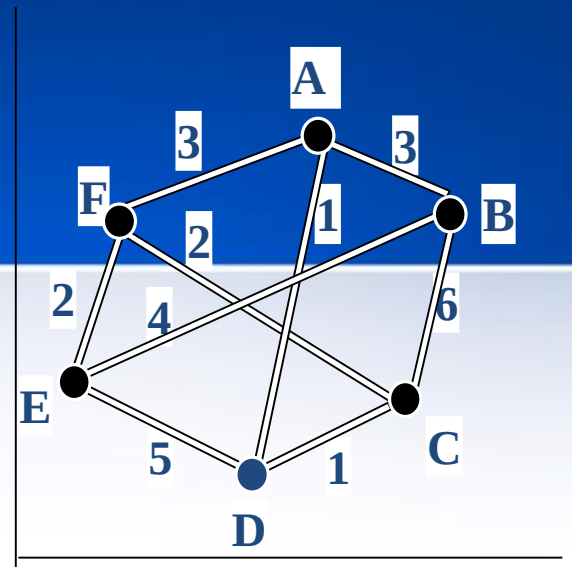
Candidate edges = (A,B) (D,E) (C,B) (E,F)

Choose edge (E,F) V={B}

Candidate edges = (A,B) (C,B) (B,E)



example



Choose vertex A, V={B,C,D,E,F}

Candidate edges = (A,F) (A,D) (A,B)

Choose edge (A,D) V={B,C,E,F}

Candidate edges = (A,F) (A,B) (D,C) (D,E)

Choose edge (D,C) V={B,E,F}

Candidate edges = (A,F) (A,B) (D,E) (C,F) (C,B)

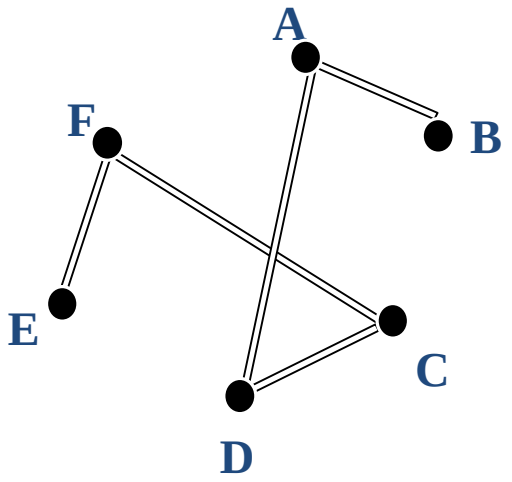
Choose edge (F,C) V={B,E}

Candidate edges = (A,B) (D,E) (C,B) (E,F)

Choose edge (E,F) V={B}

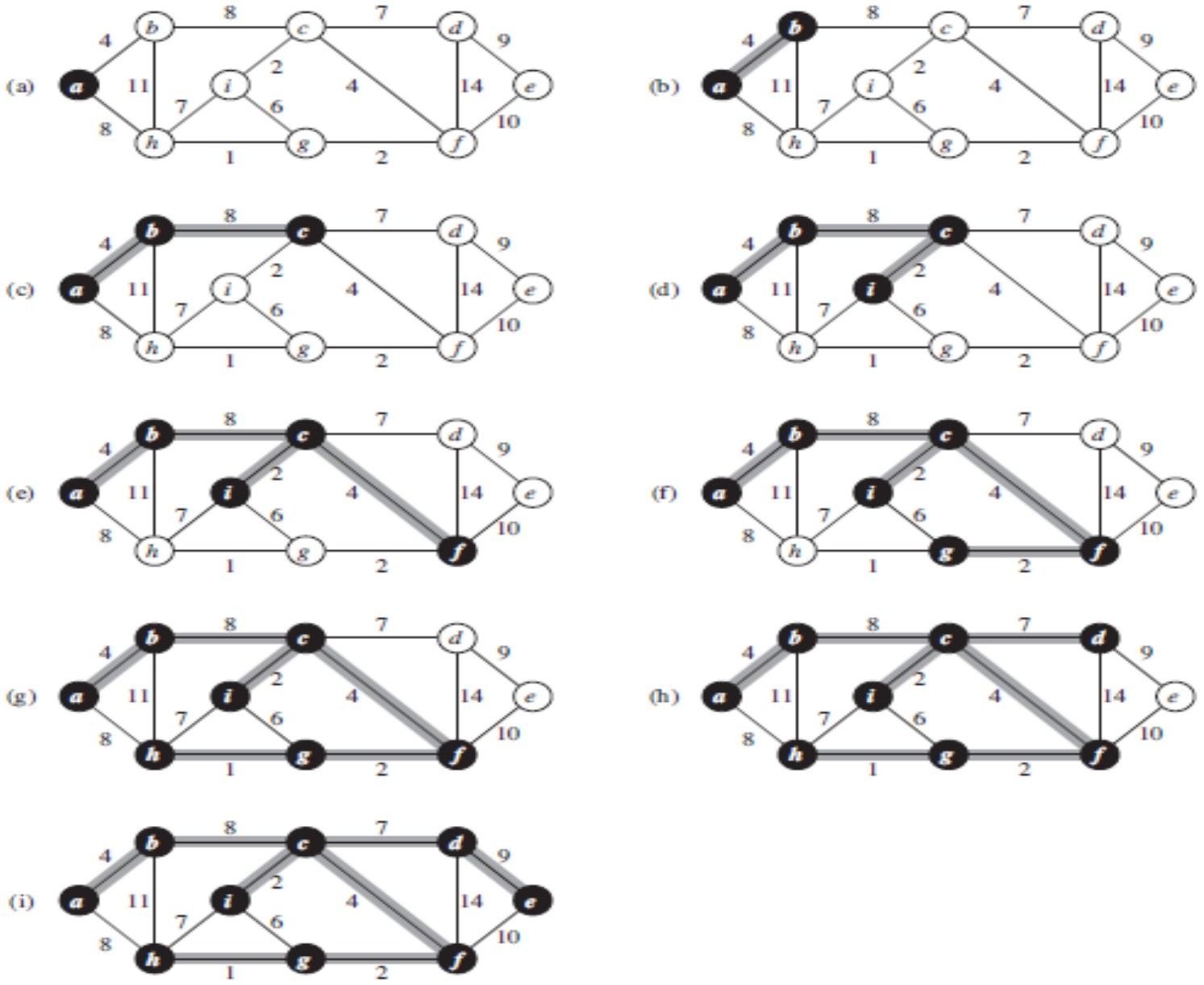
Candidate edges = (A,B) (C,B) (B,E)

Choose edge (A,B) V={}



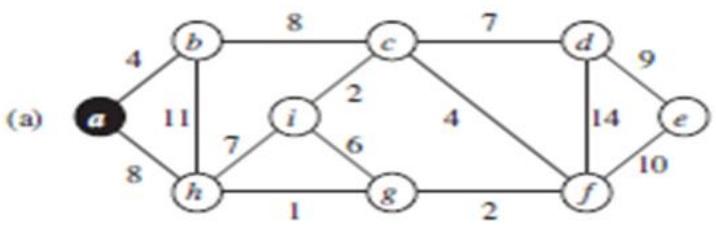
Properties of Prim’s algorithm

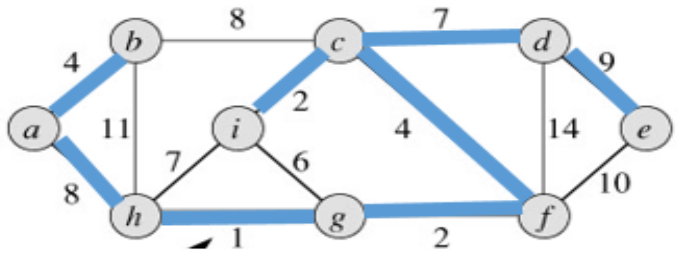
- Note that Prim’s algorithm is also Greedy: just adds a shortest edge without worrying about the overall structure
- The MCST resulting from Prim’s algorithm is also optimal
- The execution of Prim's algorithm on the graph. The root vertex is a. Shaded edges are in the tree being grown, and the vertices in the tree are shown in black. At each step of the algorithm, the vertices in the tree determine a cut of the graph, and a light edge crossing the cut is added to the tree. In the second step, for example, the algorithm has a choice of adding either edge (b, c) or edge (a, h) to the tree since both are light edges crossing the cut.


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

example

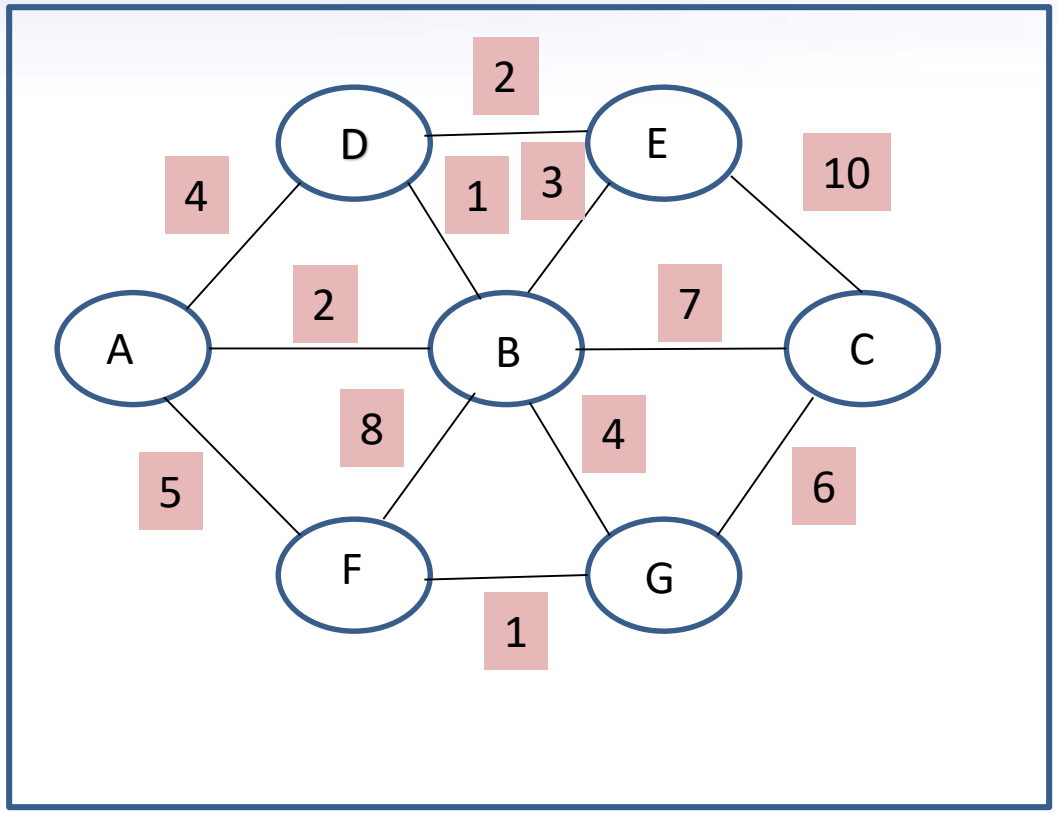
Choose vertex A, V={B,C,D,E,F,G,H,I}
Candidate edges = (A,B) (A,H)
Choose edge (A,B) ={C,D,E,F,G,H,I}
Candidate edges = (A,H) (B,H) (B,C)
Choose edge (A,H) ={C,D,E,F,G,I}
Candidate edges = (B,C) (H,I) (H,G)
Choose edge (H,G) ={C,D,E,F,I}
Candidate edges = (B,C) (H,I) (G,I) (G,F)
Choose edge (G,F) ={C,D,E,I}
Candidate edges = (B,C) (H,I) (G,I) (F,C) (F,D) (F,E)
Choose edge (F,C) ={D,E,I}
Candidate edges = (C,I) (H,I) (G,I) (C,D) (F,D) (F,E)
Choose edge (C,I) ={D,E}
Candidate edges = (C,D) (F,D) (F,E)
Choose edge (C,D) ={E}
Candidate edges = (D,E) (F,E)
Choose edge (D,E) ={}



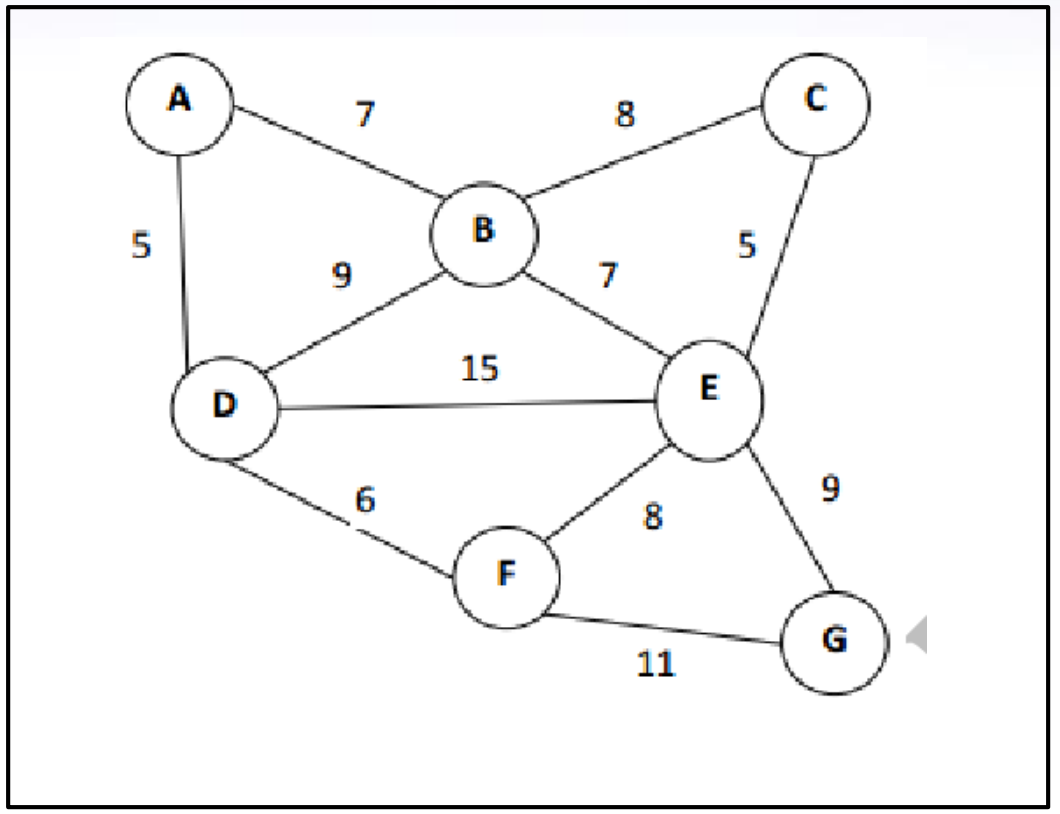


Exercise

- Final the Minimum Cost weight of following graphs.



Question 01



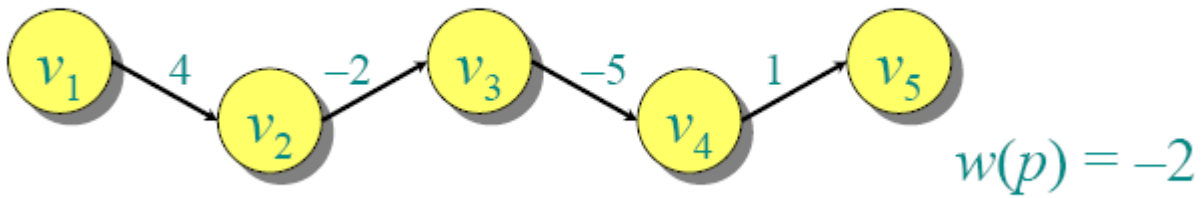
Question 02 – 2015 Pass paper question

Dijkstra's Single-source shortest path

- Introduce in 1959
- Provides the most efficient algorithm for solving *shortest-path* problem.
- In a shortest-paths problem, we are given a weighted, directed graph G = (V, E), with weight function w : E → R mapping edges to real-valued-weights. The weight of path p = $\langle v_0, v_1, \ldots, v_k \rangle$ is the sum of the weights of its constituent edges:

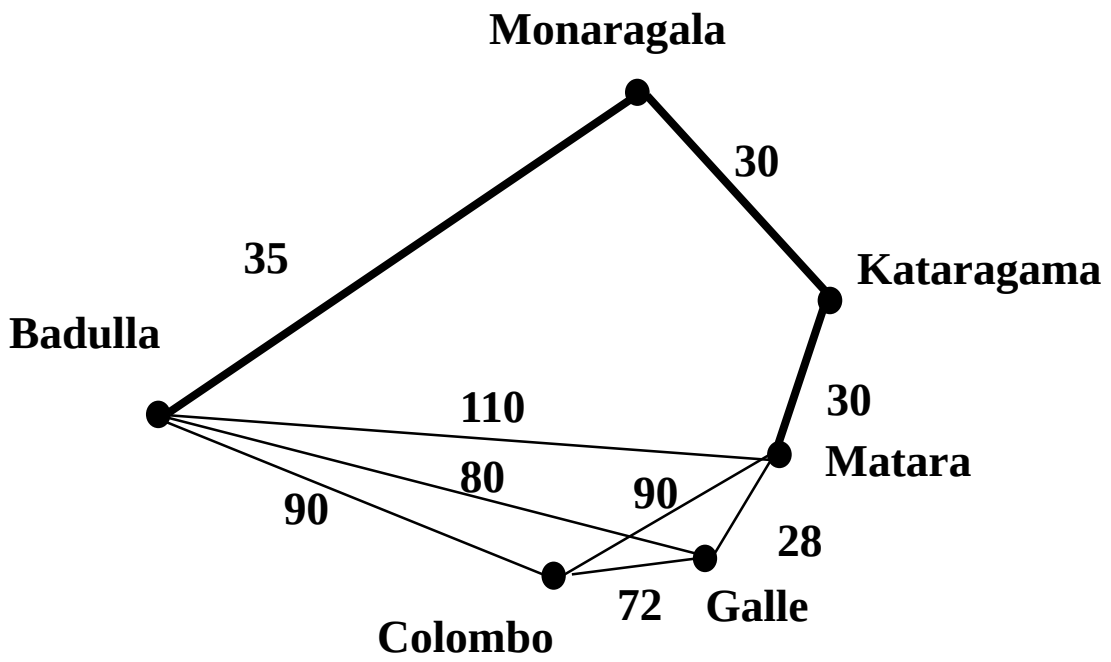
$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$

Example:

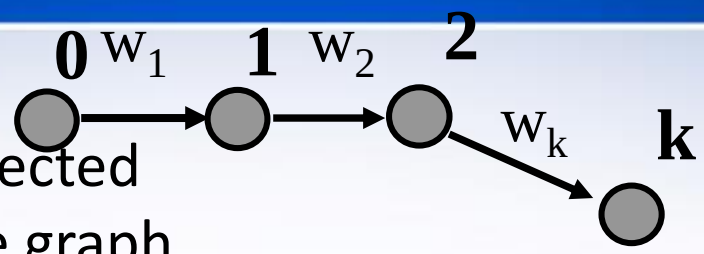


Shortest Paths

A motorist wishes to find the shortest possible route from Badulla to Matara. Given the map of Sri lanka, on which the distance between each pair of cities is marked, how can we determine the shortest route?



Shortest Paths



In a shortest-paths problem, we are given a weighted, directed graph G = (V,E), with weights assigned to each edge in the graph.

The weight of the path p = (v0, v1, v2, …, vk) is the sum of the weights of its constituent edges:

$v_0 \rightarrow v_1 \rightarrow v_2 \ . \ . \ . \rightarrow v_{k-1} \rightarrow v_k$

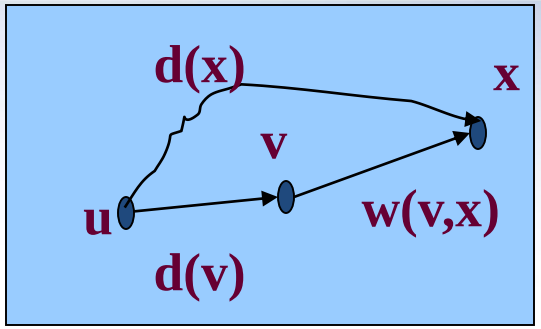
The shortest-path from u to v is given by

d(u,v) = min {weight (p) : if there is a path from u to v

= ∞ otherwise

The single-source shortest paths problem

Given G (V,E), find the shortest path from a given vertex
$u \in V$ to every vertex $v \in V$ ($u \neq v$).



For each vertex $v \in V$ in the weighted directed graph, d[v] represents the distance from u to v.

Initially, d[v] = 0 when u = v.
d[v] = ∞ if (u,v) is not an edge
d[v] = weight of edge (u,v) if (u,v) exists.

Dijkstra's Algorithm : At every step of the algorithm, we compute,
d[x] = min {d[x], d[v] + w(v,x)}, where $v,x \in V$.

Dijkstra's algorithm is based on the greedy principle because at every step we pick the path of least cost.

Dijkstra's algorithm

- The algorithms use the technique of **relaxation**. For each vertex $v \in V$, we maintain an attribute d[v], which is an upper bound on the weight of a shortest path from source s to v.
- We call d[v] a **shortest-path estimate**.
- We initialize the shortest-path estimates and predecessors by the following $\Theta(V)$-time procedure.

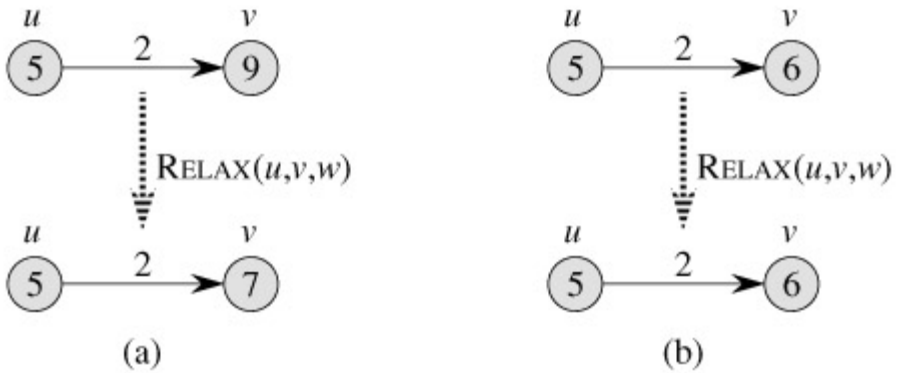
INITIALIZE-SINGLE-SOURCE(G, s)
1 **for** each vertex $v \in V[G]$
2 **do** $d[v] \leftarrow \infty$
3 $\pi[v] \leftarrow$ NIL
4 $d[s] \leftarrow 0$

After initialization, $\pi[v]$ = NIL for all $v \in V$, $d[s] = 0$, and $d[v] = \infty$ for $v \in V - \{s\}$.

Dijkstra's algorithm

- The process of **relaxing** an edge (u, v) consists of testing whether we can improve the shortest path to v found so far by going through u and, if so, updating d[v] and π[v]. A relaxation step may decrease the value of the shortest-path esti-mate d[v] and update v's predecessor field π[v]. The following code performs a relaxation step on edge (u, v).

```
RELAX(u, v, w)
1  if d[v] > d[u] + w(u, v)
2        then d[v] ← d[u] + w(u, v)
3             π[v] ← u
```



Relaxation of an edge (*u, v*) with weight *w*(*u, v*) = 2. The shortest-path estimate of each vertex is shown within the vertex. (a) Because *d*[*v*] > *d*[*u*] + *w*(*u, v*) prior to relaxation, the value of *d*[*v*] decreases. (b) Here, *d*[*v*] ≤ *d*[*u*] + *w*(*u, v*) before the relaxation step, and so *d*[*v*] is unchanged by relaxation.

Dijkstra's Single-source shortest path

Input: G =(V,E), the weighted directed graph and u the source vertex
Output: for each vertex, v, d[v] is the length of the shortest path from u to v.

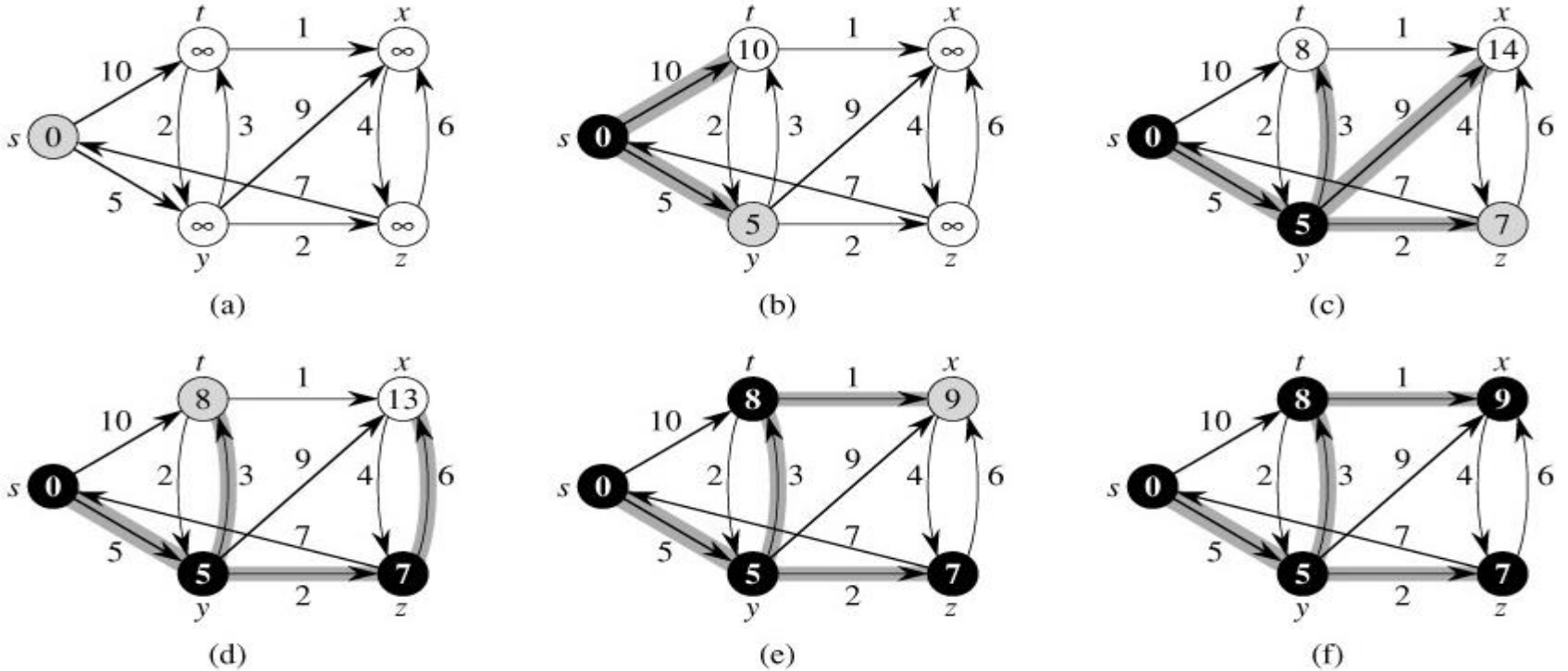
Dijkstra's algorithm solves the single-source shortest-paths problem on a weighted, directed graph $G = (V, E)$ for the case in which all edge weights are nonnegative.

we assume that $w(u, v) \geq 0$ for each edge $(u, v) \in E$, the running time of Dijkstra's algorithm is lower than that of the Bellman-Ford algorithm.

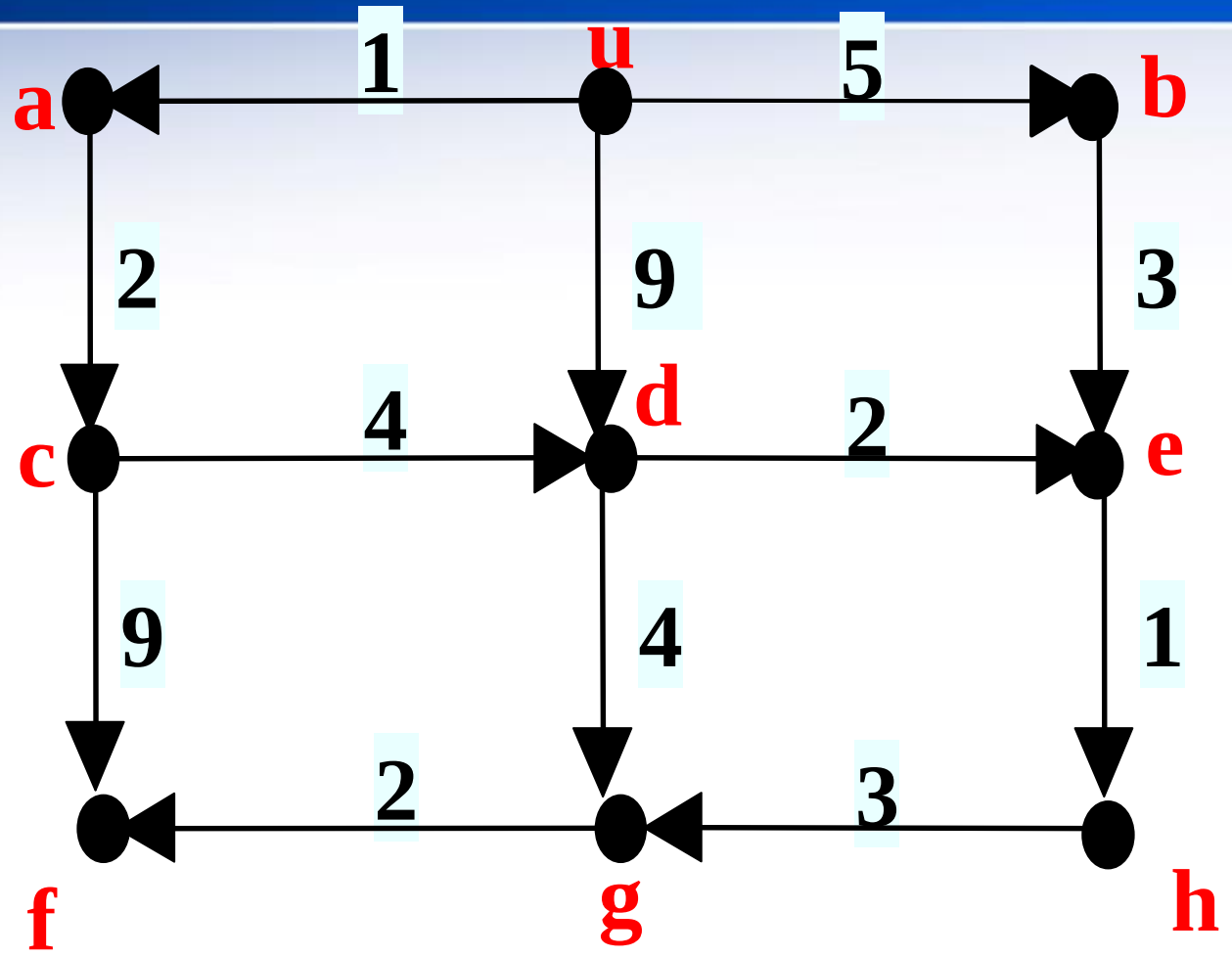
Dijkstra's algorithm maintains a set S of vertices whose final shortest-path weights from the source s have already been determined. The algorithm repeatedly selects the vertex $u \in V - S$ with the minimum shortest-path estimate, adds u to S, and relaxes all edges leaving u. In the following implementation, we use a min-priority queue Q of vertices, keyed by their d values.

```
DIJKSTRA(G, w, s)
1     INITIALIZE-SINGLE-SOURCE(G, s)
2     S ← Ø
3     Q ← V[G]
4      while Q ≠ Ø
5           do u ← EXTRACT-MIN(Q)
6                S ← S ∈ {u}
7                     for each vertex v ∈ Adj[u]
8                          do RELAX(u, v, w)
```

The execution of Dijkstra's algorithm. The source s is the leftmost vertex. The shortest-path estimates are shown within the vertices, and shaded edges indicate predecessor values. Black vertices are in the set S, and white vertices are in the min-priority queue Q = V - S. (a) The situation just before the first iteration of the while loop of lines 4-8. The shaded vertex has the minimum d value and is chosen as vertex u in line 5. (b)-(f) The situation after each successive iteration of the while loop. The shaded vertex in each part is chosen as vertex u in line 5 of the next iteration. The d and π values shown in part (f) are the final values.

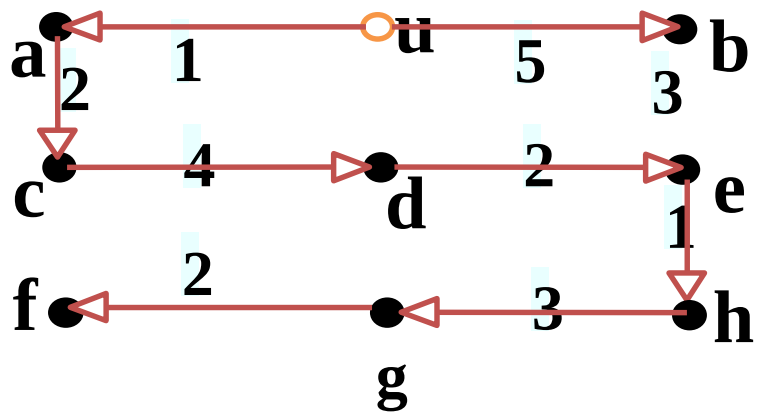


Example



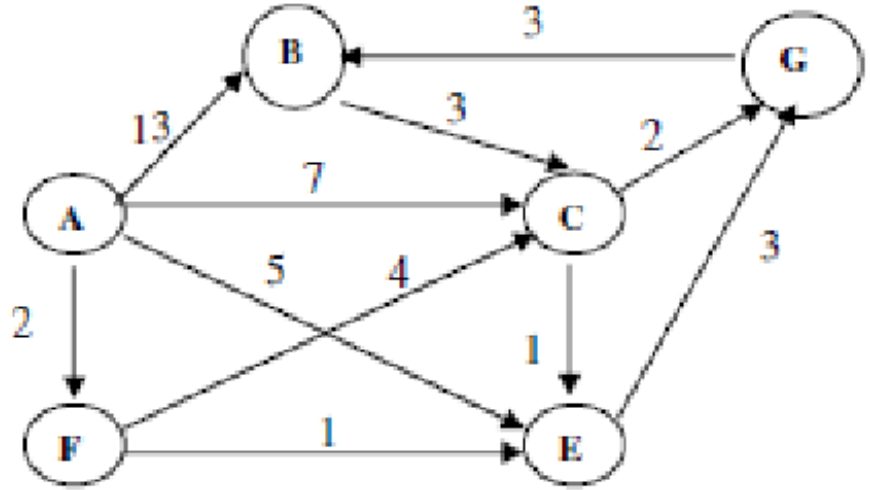
Find the shortest path from *u* to all the vertexes.

Step #	Vertex to be marked	Distance to vertex								Unmarked vertices
		d[u]	d[a]	d[b]	d[c]	d[d]	d[e]	d[f]	d[g]	d[h]	
0	u	0	1	5	∞	9	∞	∞	∞	∞	a,b,c,d,e,f,g,h
1	a	0	1	5	3	9	∞	∞	∞	∞	b,c,d,e,f,g,h
2	c	0	1	5	3	7	∞	12	∞	∞	b,d,e,f,g,h
3	b	0	1	5	3	7	8	12	∞	∞	d,e,f,g,h
4	d	0	1	5	3	7	8	12	11	∞	e,f,g,h
5	e	0	1	5	3	7	8	12	11	9	f,g,h
6	h	0	1	5	3	7	8	12	11	9	f,g
7	g	0	1	5	3	7	8	12	11	9	f
8	f	0	1	5	3	7	8	12	11	9	--------



Exercise:

Apply the **Dijkstra's algorithm** to find the shortest path from the vertex **A** to all the other vertices of the graph. **Source vertex : A**



2015 – Final Exam paper Question

Summery

- Greedy Method
- Greedy Graph Algorithms
  - Kruskal's Algorithm to find MST
  - Prim's Algorithm to find MST
  - Dijkstra's Algorithm to find Shortest Paths

Questions ???

KEEP
CALM
AND
GOOD LUCK
ON YOUR EXAM

anuruddha.a@sliit.lk