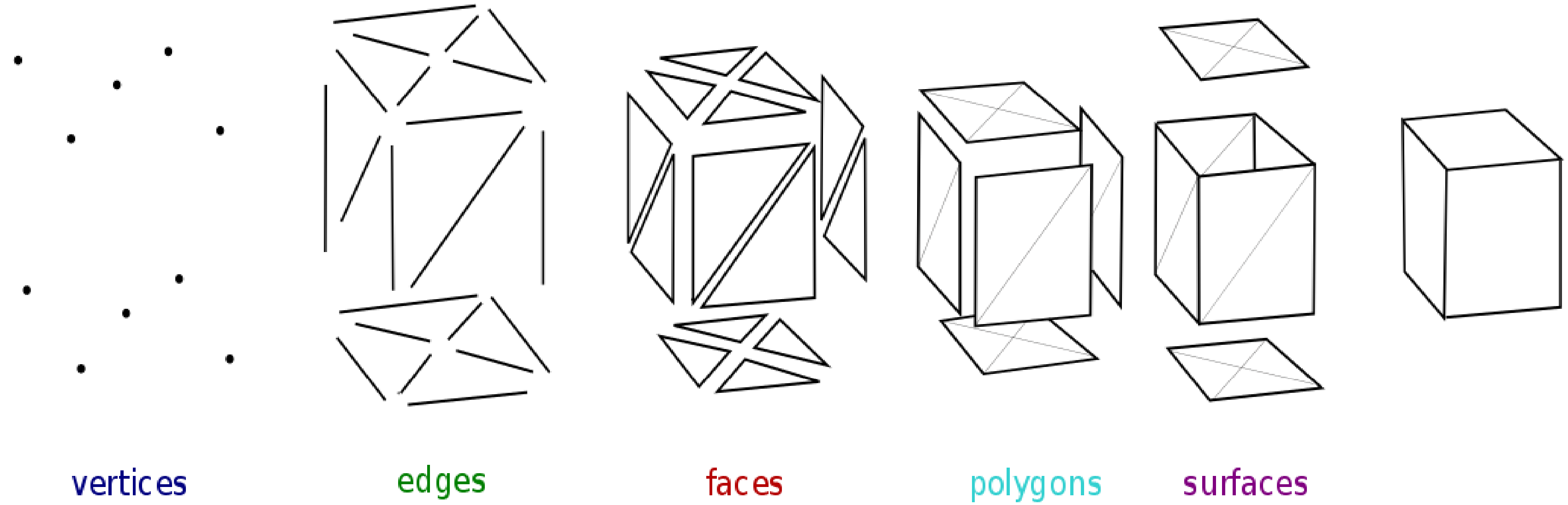
Design and Analysis of Data Structures and Algorithms :: Graph part 2

อ.ดร.วรินทร์ วัฒนพรพรหม

Polygon 3d Mesh

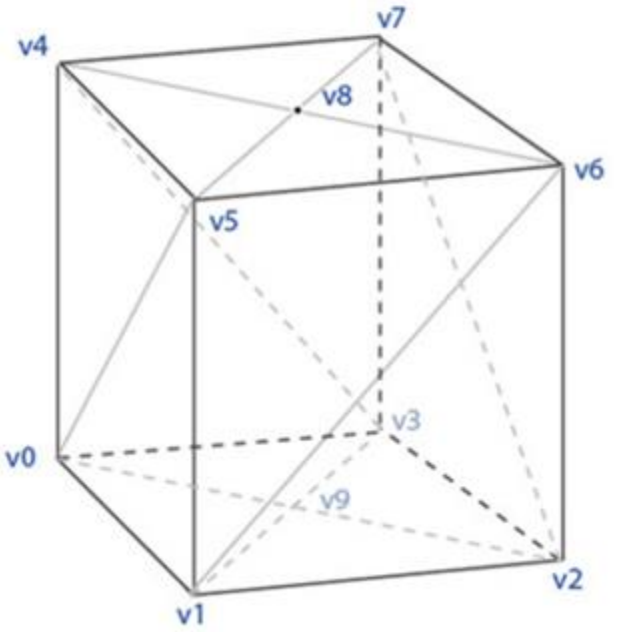


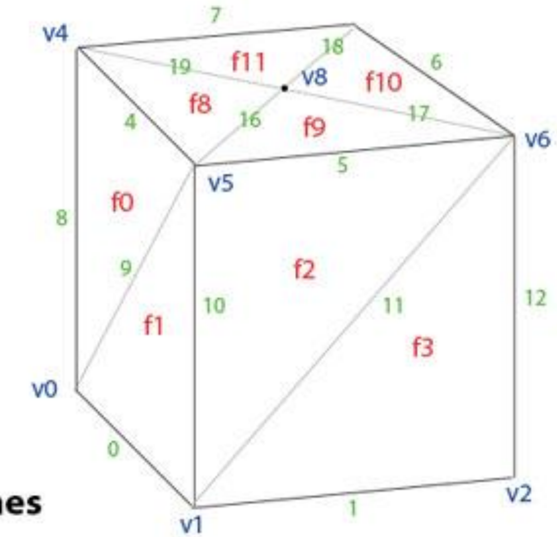
Polygon 3d Mesh

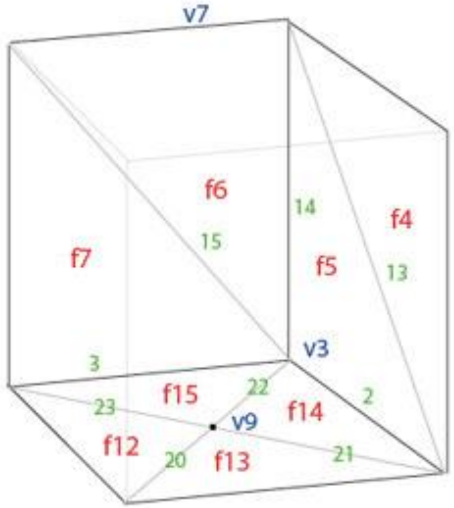
Vertex-Vertex Meshes (VV)

Vertex List

v0	0,0,0	v1 v5 v4 v3 v9
v1	1,0,0	v2 v6 v5 v0 v9
v2	1,1,0	v3 v7 v6 v1 v9
v3	0,1,0	v2 v6 v7 v4 v9
v4	0,0,1	v5 v0 v3 v7 v8
v5	1,0,1	v6 v1 v0 v4 v8
v6	1,1,1	v7 v2 v1 v5 v8
v7	0,1,1	v4 v3 v2 v6 v8
v8	.5,.5,1	v4 v5 v6 v7
v9	.5,.5,0	v0 v1 v2 v3







Winged-Edge Meshes

Face List

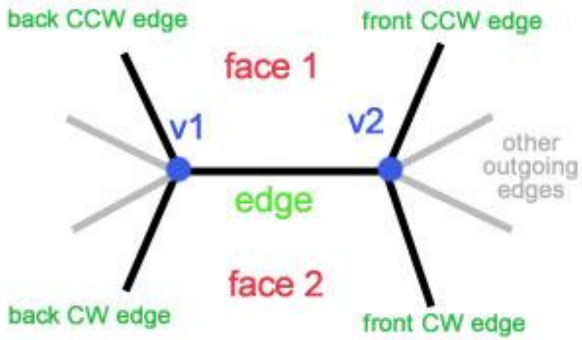
f0	4 8 9
f1	0 10 9
f2	5 10 11
f3	1 12 11
f4	6 12 13
f5	2 14 13
f6	7 14 15
f7	3 8 15
f8	4 16 19
f9	5 17 16
f10	6 18 17
f11	7 19 18
f12	0 23 20
f13	1 20 21
f14	2 21 22
f15	3 22 23

Edge List

e0	v0 v1	f1 f12	9 23 10 20
e1	v1 v2	f3 f13	11 20 12 21
e2	v2 v3	f5 f14	13 21 14 22
e3	v3 v0	f7 f15	15 22 8 23
e4	v4 v5	f0 f8	19 8 16 9
e5	v5 v6	f2 f9	16 10 17 11
e6	v6 v7	f4 f10	17 12 18 13
e7	v7 v4	f6 f11	18 14 19 15
e8	v0 v4	f7 f0	3 9 7 4
e9	v0 v5	f0 f1	8 0 4 10
e10	v1 v5	f1 f2	0 11 9 5
e11	v1 v6	f2 f3	10 1 5 12
e12	v2 v6	f3 f4	1 13 11 6
e13	v2 v7	f4 f5	12 2 6 14
e14	v3 v7	f5 f6	2 15 13 7
e15	v3 v4	f6 f7	14 3 7 15
e16	v5 v8	f8 f9	4 5 19 17
e17	v6 v8	f9 f10	5 6 16 18
e18	v7 v8	f10 f11	6 7 17 19
e19	v4 v8	f11 f8	7 4 18 16
e20	v1 v9	f12 f13	0 1 23 21
e21	v2 v9	f13 f14	1 2 20 22
e22	v3 v9	f14 f15	2 3 21 23
e23	v0 v9	f15 f12	3 0 22 20

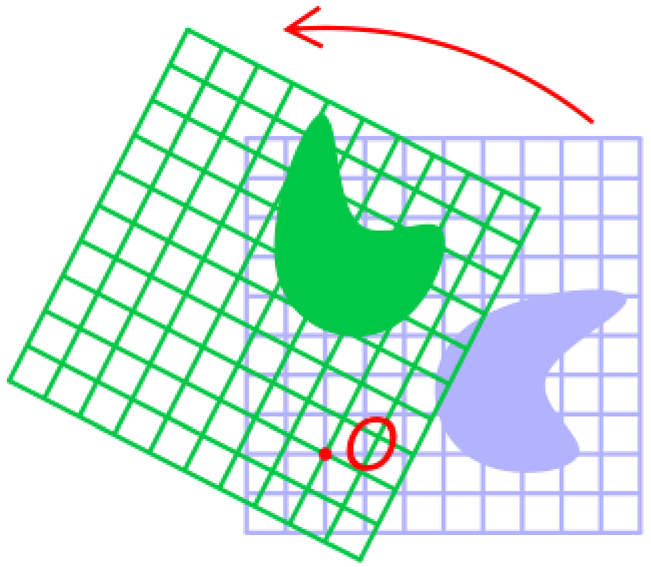
Vertex List

v0	0,0,0	8 9 0 23 3
v1	1,0,0	10 11 1 20 0
v2	1,1,0	12 13 2 21 1
v3	0,1,0	14 15 3 22 2
v4	0,0,1	8 15 7 19 4
v5	1,0,1	10 9 4 16 5
v6	1,1,1	12 11 5 17 6
v7	0,1,1	14 13 6 18 7
v8	.5,.5,0	16 17 18 19
v9	.5,.5,1	20 21 22 23



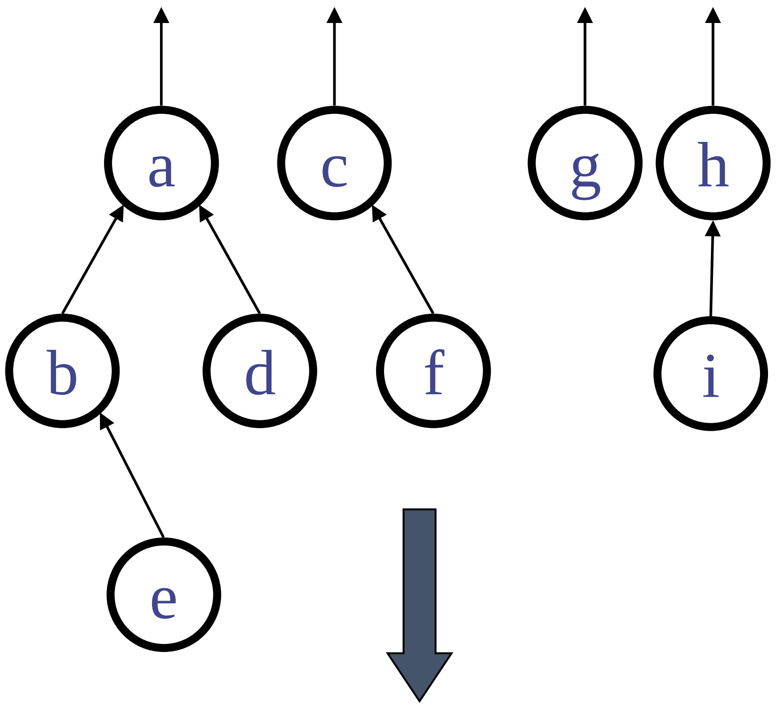
Winged Edge Structure

Rotation



Disjoint set data structure

- ใช้ forest ของ up tree
- ใช้หมายเลขโหนดเป็นตัวเลขยิ่งเร็ว



	0 (a)	1 (b)	2 (c)	3 (d)	4 (e)	5 (f)	6 (g)	7 (h)	8 (i)
up-index:	-1	0	-1	0	1	2	-1	-1	7

Disjoint set

```
typedef ID int;
ID find(Object x) {
  assert(hTable.contains(x));
  ID xID = hTable[x];

  while(up[xID] != -1) {
    xID = up[xID];
  }

  return xID;
}
```

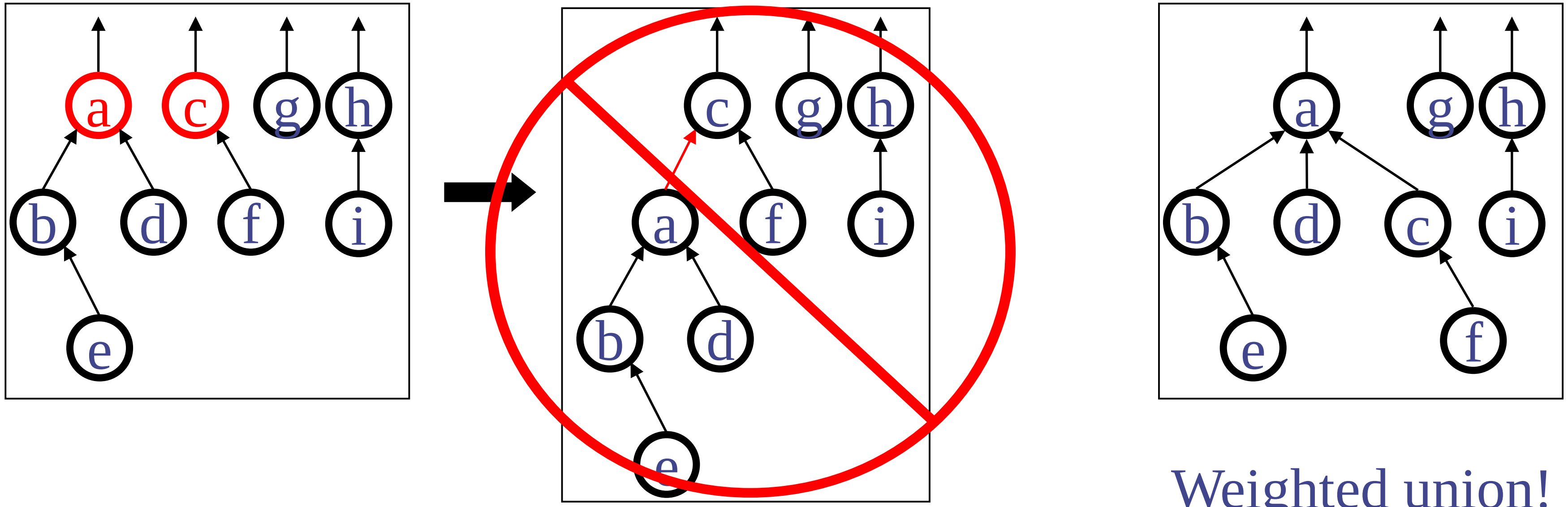
```
ID union(ID x, ID y) {
  assert(up[x] == -1);
  assert(up[y] == -1);

  up[y] = x;
}
```

runtime: O(depth) or …

runtime: O(1)

Room for Improvement: Weighted Union

- Union ตามความสูงของ tree จะทำให้ tree ไม่สูงมาก
- คำเตือน บางปัญหาอย่า union ตามความสูงเพราะจะทำให้เส้นทางหายไป



Weighted union!

Weighted Union

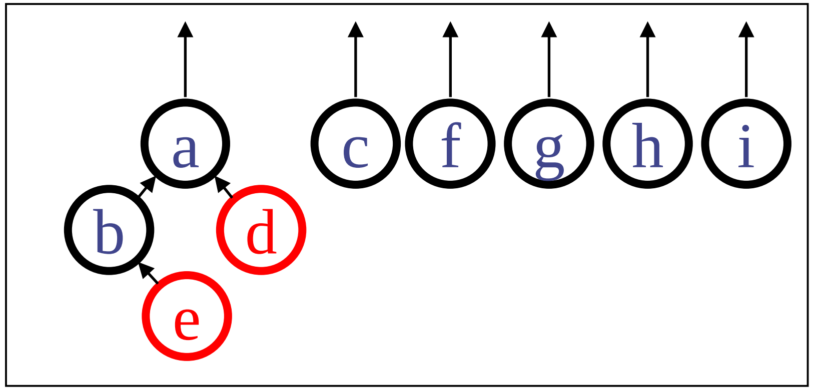
```
typedef ID int;
ID union(ID x, ID y) {
  assert(up[x] == -1);
  assert(up[y] == -1);

  if (weight[x] > weight[y]) {
    up[y] = x;
    weight[x] += weight[y];
  } else {
    up[x] = y;
    weight[y] += weight[x];
  }
}
```

new runtime of union:

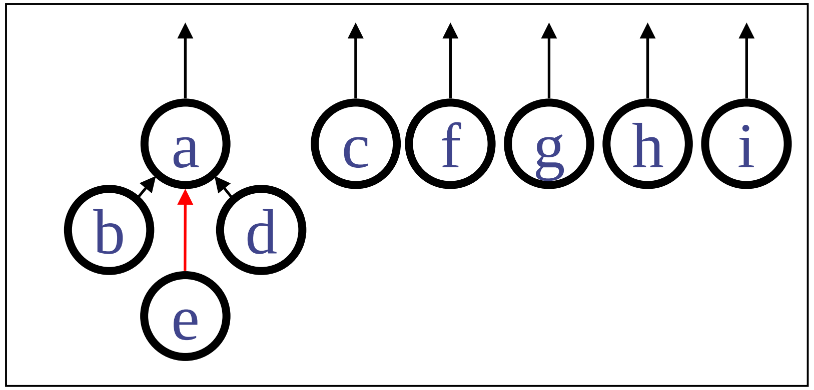
new runtime of find:

Room for Improvement: Path Compression

- Find เมื่อไหร่ชี้โหนดกลางทางไปหา root
- ลดความสูงให้เหลือ 1
- คำเตือน บางปัญหาอย่า compress เพราะจะทำให้เส้นทางหายไป

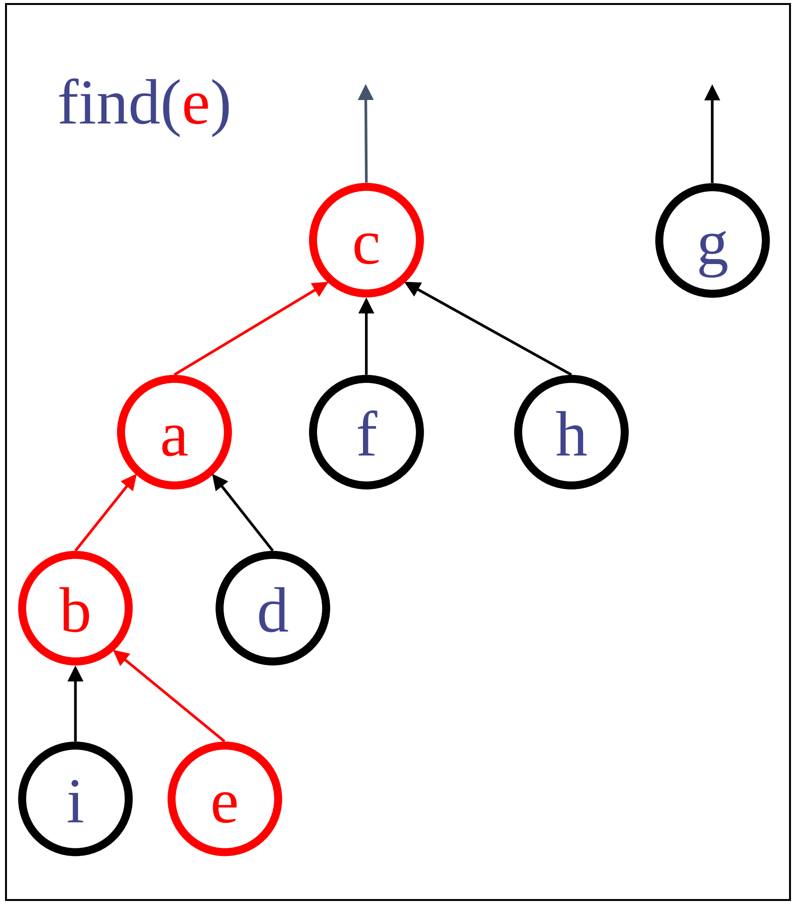


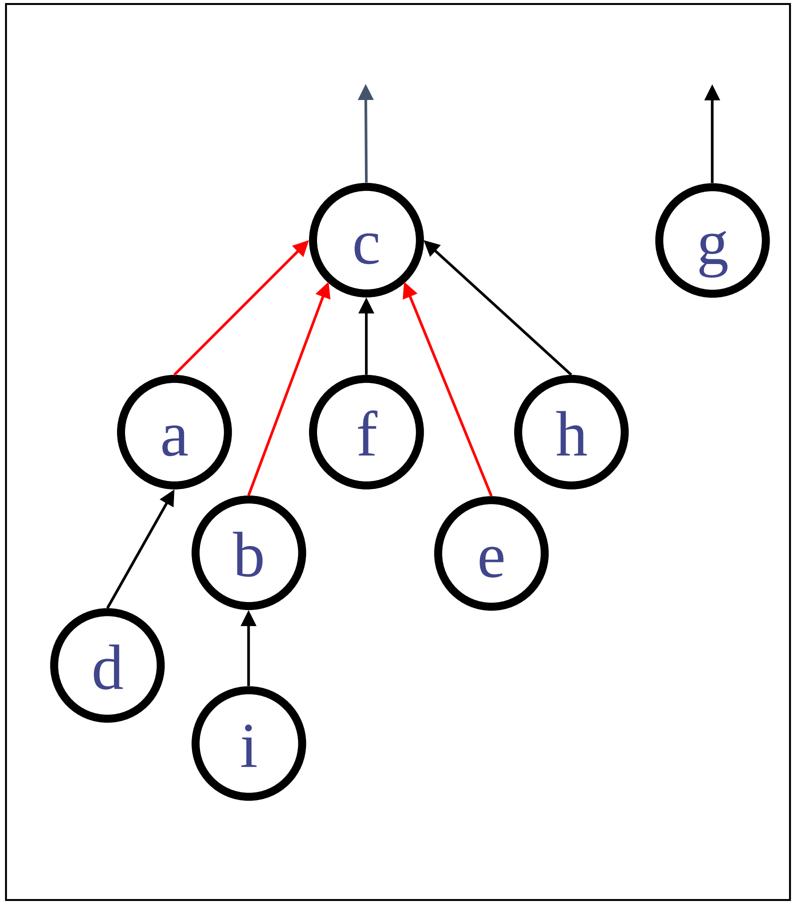
While we're finding ***e***,
could we do anything else?



Path compression!

Path Compression Example





Path Compression Code

```
typedef ID int;
ID find(Object x) {
  assert(hTable.contains(x));
  ID xID = hTable[x];
  ID hold = xID;
  while(up[xID] != -1) {
    xID = up[xID];
  }
  while(up[hold] != -1) {
    temp = up[hold];
    up[hold] = xID;
    hold = temp;
  }
  return xID;
}
```

runtime:

Disjoint Sets

ADT	Time complexity
`Make-Set(x)`	$\Theta(1)$
`Union(s1,s2)`	$\Theta(1)$
`Find-Set(x)`	$O(\log n)$*

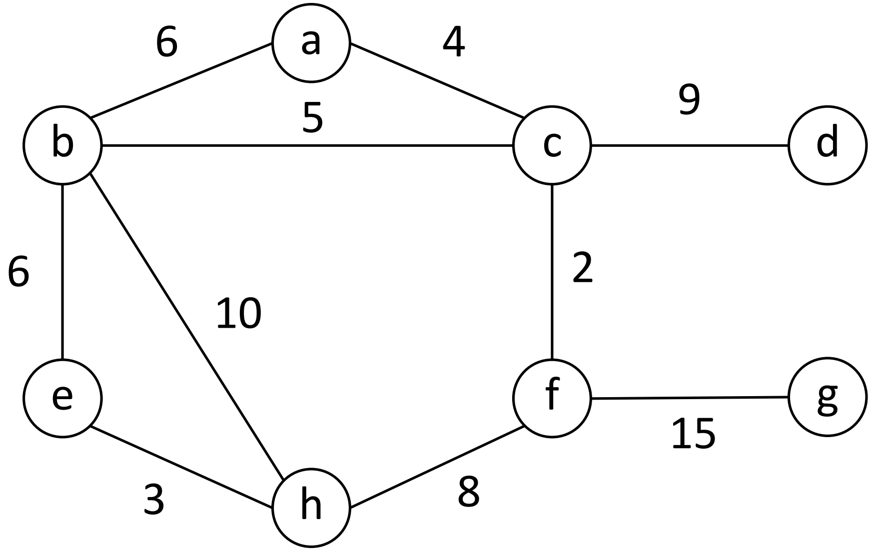
Kruskal’s MST Algorithm

- แนวคิดคือสร้าง tree ด้วยวิธีละโมบ (greedy)
  - ไต่ไปตามขอบของ Edge แล้วสร้าง forest เก็บไว้
  - แต่ละ vertex ทำการ sort edge (ใช้ heap จะเร็ว)
  - Edges ที่มีน้ำหนักน้อยโดยจับใส่ก่อน
  - Edges ที่โดนจับใส่ต้องไม่ทำลายโครงสร้างต้นไม้ กล่าวคือไม่เกิดการ short circuit วิธีการคือใช้ disjoint set ช่วย

Kruskal's Algorithm

```
Kruskal(G,w)                    ; Graph G, with weights w
    A ← {}                      ; Our MST starts empty
    for each vertex v ∈ V[G] do Make-Set(v)    ; Make each vertex a set
    Sort edges of E by increasing weight
    for each edge (u,v) ∈ E in order
            ;  Find-Set returns a representative (first vertex) in the set
            do if Find-Set(u) ≠ Find-Set(v)
                then A ← A ∪ {(u,v)}
                    Union(u,v)                 ; Combines two trees
    return A
```

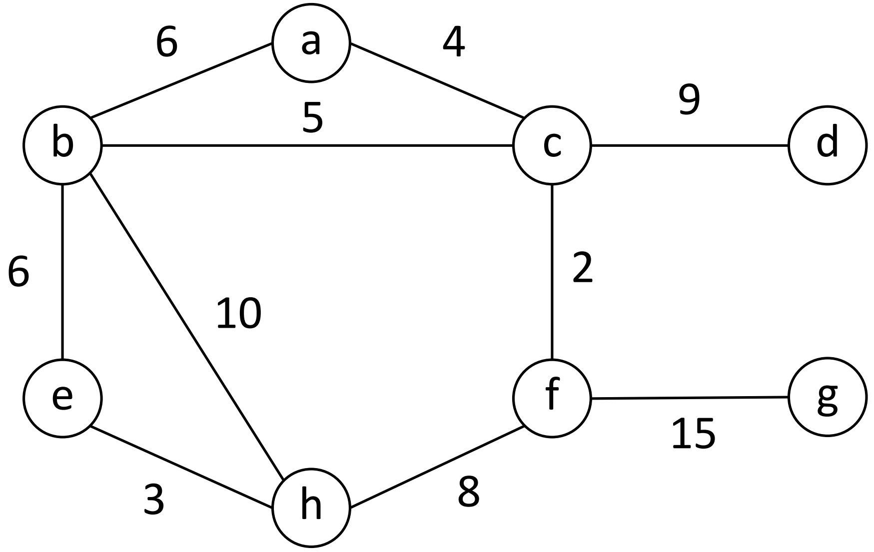
Kruskal's Example



Graph (tabular form)

From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	6	10	9	2	3	8	15

Kruskal’s Example



Graph (tabular form)

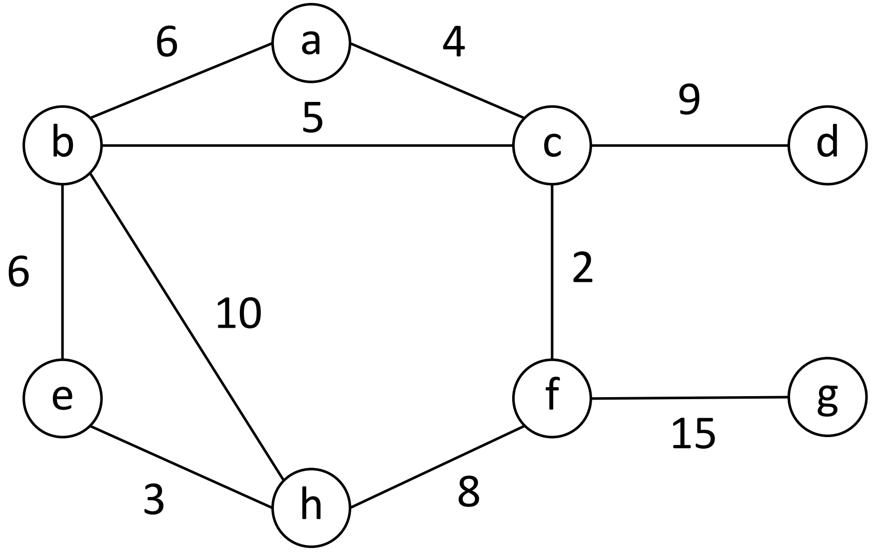
From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	6	10	9	2	3	8	15

Disjoint set

Index	0	1	2	3	4	5	6	7
Node	a	b	c	d	e	f	g	h
Parent	0	1	2	3	4	5	6	7

- A={ }, Make each node element its own set. {a} {b} {c} {d} {e} {f} {g} {h}

Kruskal's Example



Graph (tabular form)

From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	6	10	9	2	3	8	15

↓ Sort

Heap (simplified view)

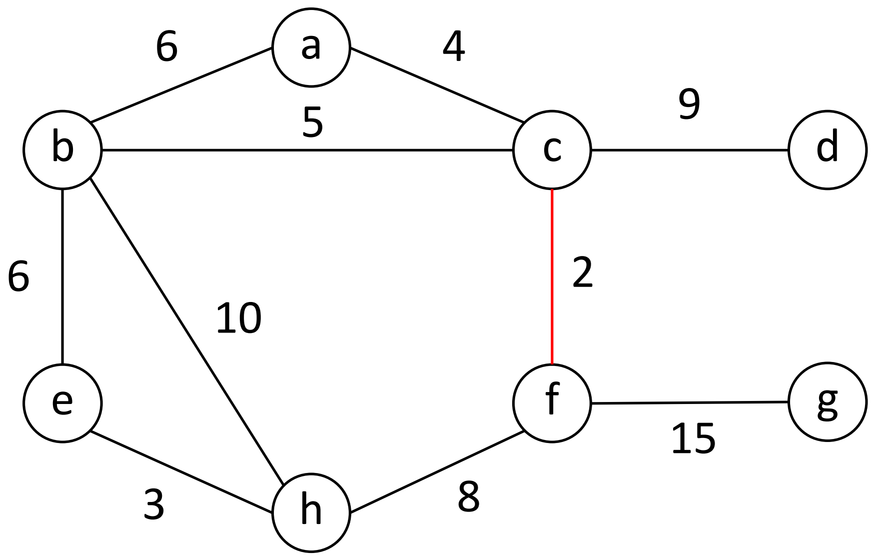
From	c	e	a	b	a	b	f	c	b	f
To	f	h	c	c	b	e	h	d	h	g
Weight	2	3	4	5	6	6	8	9	10	15

Disjoint set

Index	0	1	2	3	4	5	6	7
Node	a	b	c	d	e	f	g	h
Parent	0	1	2	3	4	5	6	7

- A={ }, Make each node element its own set. {a} {b} {c} {d} {e} {f} {g} {h}
- Sort edges using heap.

Kruskal’s Example



Graph (tabular form)

From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	6	10	9	2	3	8	15

Sort

Heap (simplified view)

From	c	e	a	b	a	b	f	c	b	f
To	f	h	c	c	b	e	h	d	h	g
Weight	2	3	4	5	6	6	8	9	10	15

Disjoint set

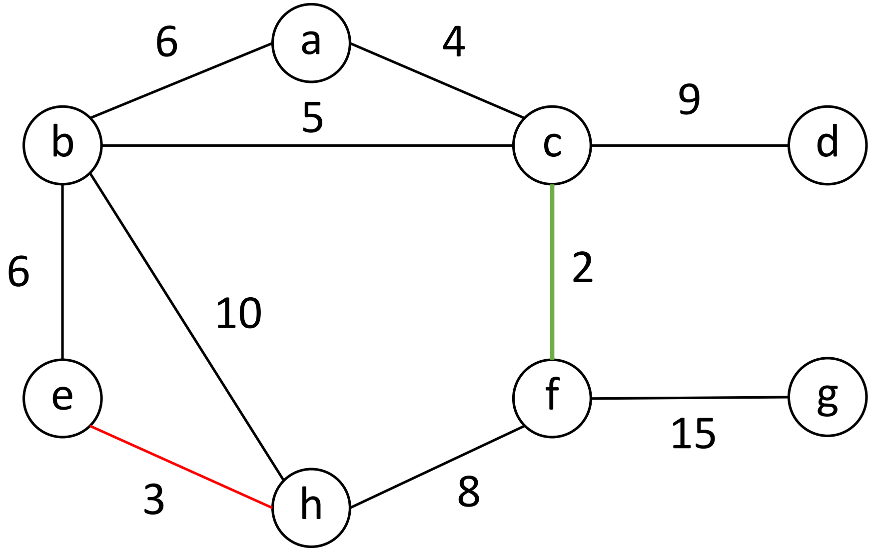
Index	0	1	2	3	4	5	6	7
Node	a	b	c	d	e	f	g	h
Parent	0	1	2	3	4	2	6	7

Tree (tabular form)

From	c									
To	f									
Weight	2									

- A={ }, Make each node element its own set. {a} {b} {c} {d} {e} {f} {g} {h}
- Sort edges edges using heap.
- Extract the smallest edge first:
  - if {c} and {f} not in same set, add it to A, union together.
- Now get {a} {b} {c f} {d} {e} {g} {h}

Kruskal’s Example



Graph (tabular form)

From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	6	10	9	2	3	8	15

Sort

Heap (simplified view)

From	[redacted]	e	a	b	a	b	f	c	b	f
To	[redacted]	h	c	c	b	e	h	d	h	g
Weight	[redacted]	3	4	5	6	6	8	9	10	15

Disjoint set

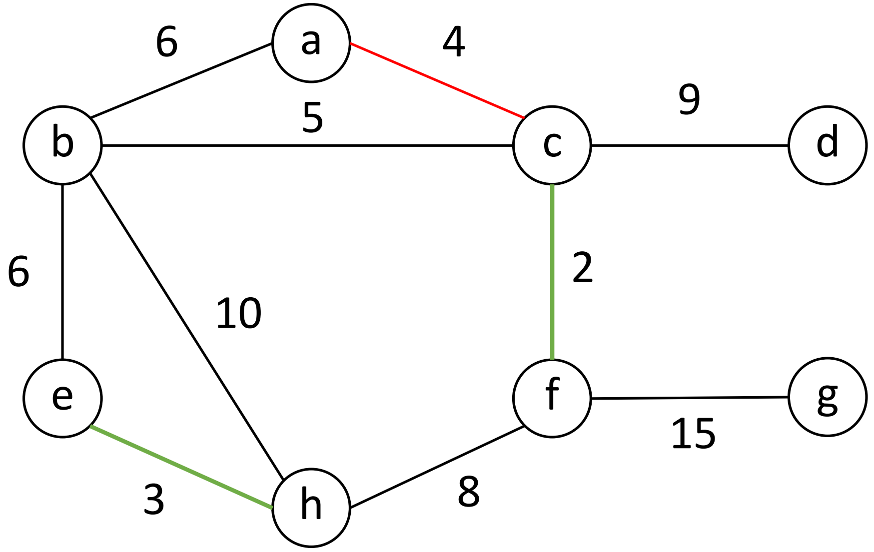
Index	0	1	2	3	4	5	6	7
Node	a	b	c	d	e	f	g	h
Parent	0	1	2	3	4	2	6	4

Tree (tabular form)

From	c	e								
To	f	h								
Weight	2	3								

- Keep going, checking next smallest edge.
- Had: {a} {b} {c f} {d} {e} {g} {h}
- {e} ≠ {h}, add edge.
- Now get {a} {b} {c f} {d} {e h} {g}

Kruskal's Example



Graph (tabular form)

From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	6	10	9	2	3	8	15

↓ Sort

Heap (simplified view)

From			a	b	a	b	f	c	b	f
To			c	c	b	e	h	d	h	g
Weight			4	5	6	6	8	9	10	15

Disjoint set

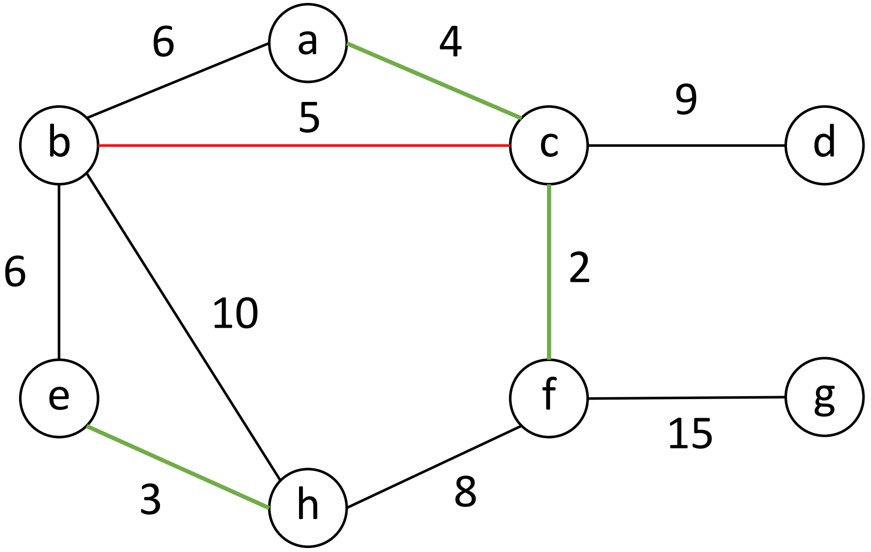
Index	0	1	2	3	4	5	6	7
Node	a	b	c	d	e	f	g	h
Parent	0	1	0	3	4	2	6	4

Tree (tabular form)

From	c	e	a							
To	f	h	c							
Weight	2	3	4							

- Keep going, checking next smallest edge.
- Had: {a} {b} {c f} {d} {e h} {g}
- {a} ≠ {c}, add edge.
- Now get {a c f} {b} {d} {e h} {g}

Kruskal’s Example



Graph (tabular form)

From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	6	10	9	2	3	8	15

↓ Sort

Heap (simplified view)

From	[redacted]	b	a	b	f	c	b	f
To		c	b	e	h	d	h	g
Weight		5	6	6	8	9	10	15

Disjoint set

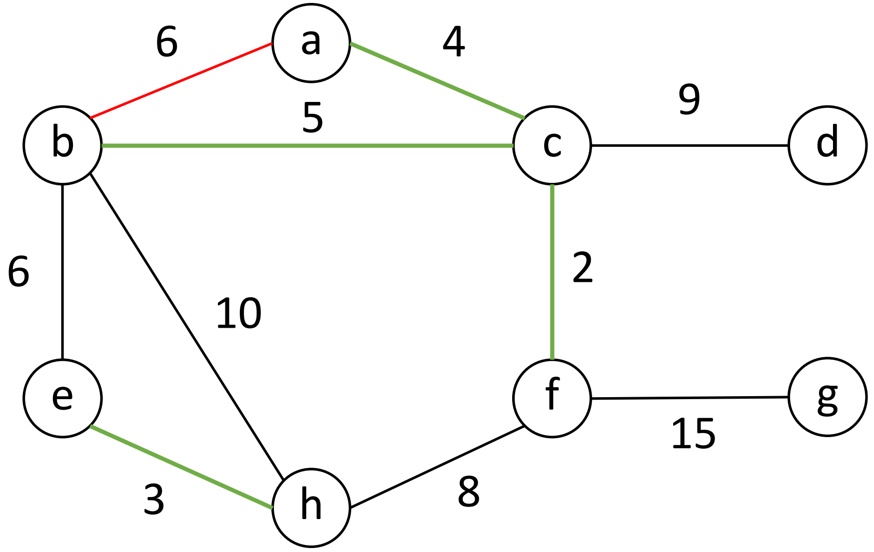
Index	0	1	2	3	4	5	6	7
Node	a	b	c	d	e	f	g	h
Parent	0	0	0	3	4	2	6	4

Tree (tabular form)

From	c	e	a	b						
To	f	h	c	c						
Weight	2	3	4	5						

- Keep going, checking next smallest edge.
- Had: {a c f} {d} {e h} {g}
- {b} ≠ {c}, add edge.
- Now get {a b c f} {d} {e h} {g}

Kruskal’s Example



Graph (tabular form)

From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	6	10	9	2	3	8	15

Sort

Heap (simplified view)

From	[redacted]				a	b	f	c	b	f
To	[redacted]				b	e	h	d	h	g
Weight	[redacted]				6	6	8	9	10	15

Disjoint set

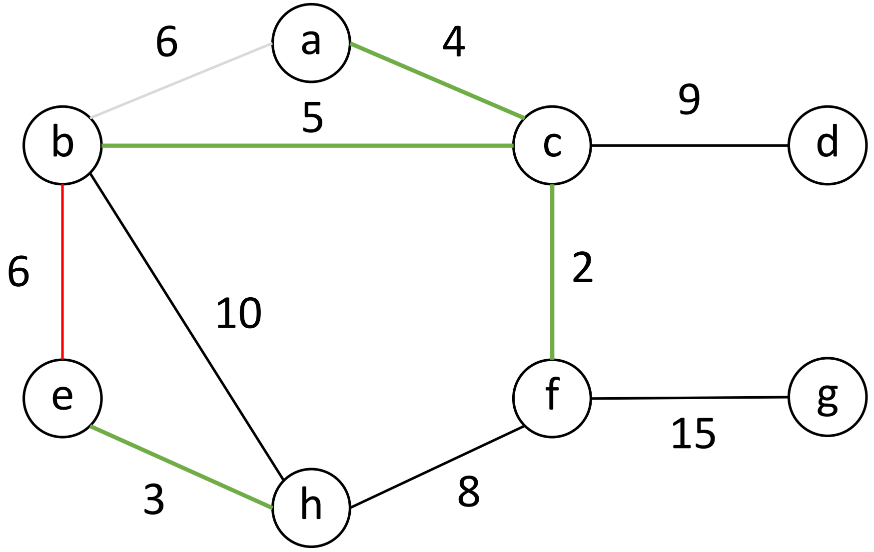
Index	0	1	2	3	4	5	6	7
Node	a	b	c	d	e	f	g	h
Parent	0	0	0	3	4	2	6	4

Tree (tabular form)

From	c	e	a	b						
To	f	h	c	c						
Weight	2	3	4	5						

- Keep going, checking next smallest edge.
- Had: {a b c f} {d} {e h} {g}
- {a} = {b}, do nothing.

Kruskal’s Example



Graph (tabular form)

From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	6	10	9	2	3	8	15

↓ Sort

Heap (simplified view)

From	[redacted]					b	f	c	b	f
To	[redacted]					e	h	d	h	g
Weight	[redacted]					6	8	9	10	15

Disjoint set

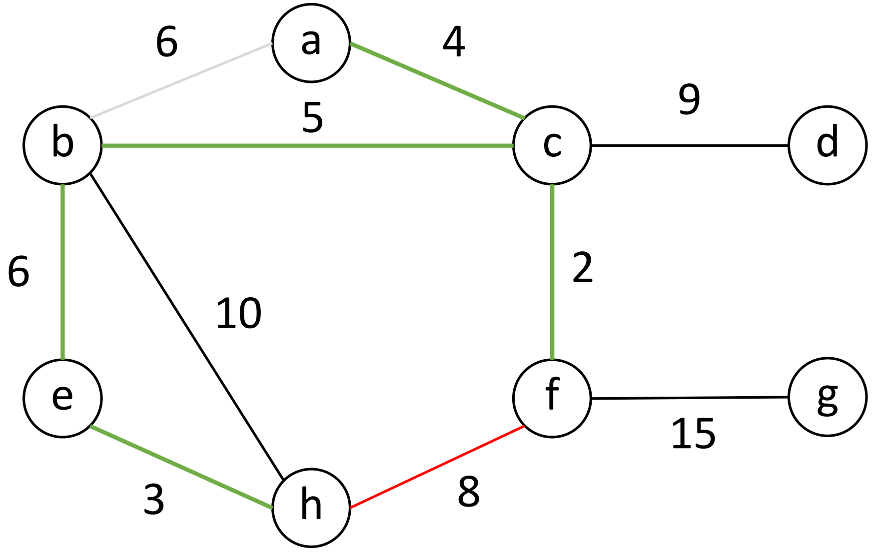
Index	0	1	2	3	4	5	6	7
Node	a	b	c	d	e	f	g	h
Parent	0	0	0	3	0	2	6	4

Tree (tabular form)

From	c	e	a	b	b					
To	f	h	c	c	e					
Weight	2	3	4	5	6					

- Keep going, checking next smallest edge.
- Had: {a b c f} {d} {e h} {g}
- {b} ≠ {e}, add edge.
- Now get {a b c e f h} {d} {g}

Kruskal’s Example



Graph (tabular form)

From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	6	10	9	2	3	8	15

Sort

Heap (simplified view)

From	[redacted]	f	c	b	f
To		h	d	h	g
Weight		8	9	10	15

Disjoint set

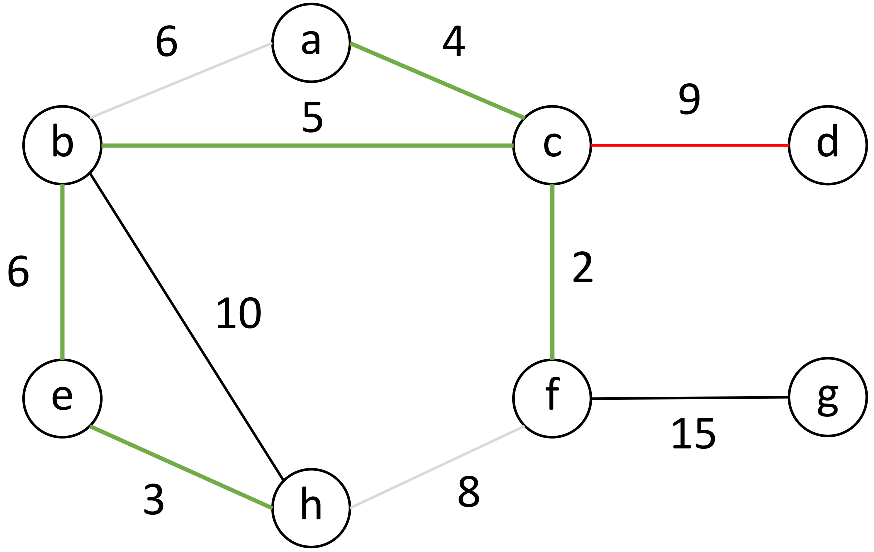
Index	0	1	2	3	4	5	6	7
Node	a	b	c	d	e	f	g	h
Parent	0	0	0	3	0	2	6	4

Tree (tabular form)

From	c	e	a	b	b					
To	f	h	c	c	e					
Weight	2	3	4	5	6					

- Keep going, checking next smallest edge.
- Had: {a b c e f h} {d} {g}
- {f} = {h}, do nothing.

Kruskal’s Example



Graph (tabular form)

From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	6	10	9	2	3	8	15

Sort

Heap (simplified view)

From	[redacted]	c	b	f
To	[redacted]	d	h	g
Weight	[redacted]	9	10	15

Disjoint set

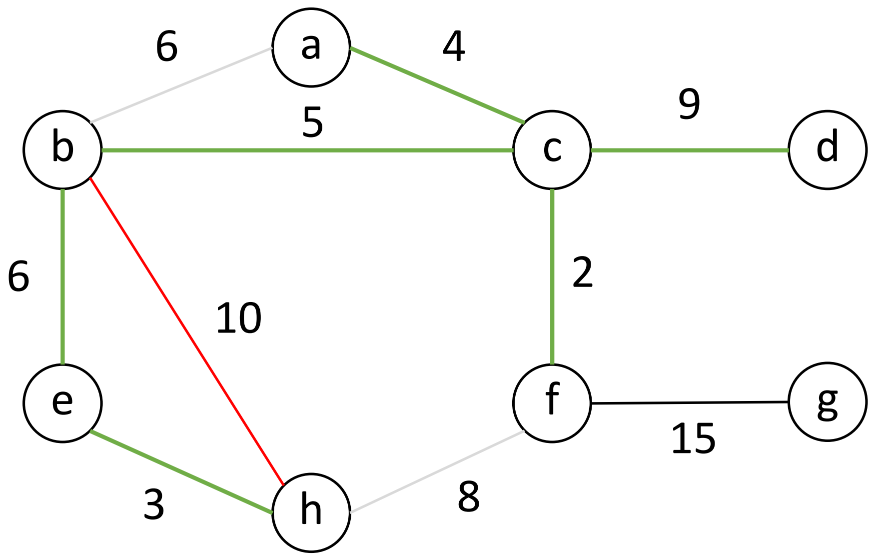
Index	0	1	2	3	4	5	6	7
Node	a	b	c	d	e	f	g	h
Parent	0	0	0	0	0	2	6	4

Tree (tabular form)

From	c	e	a	b	b	c				
To	f	h	c	c	e	d				
Weight	2	3	4	5	6	9				

- Keep going, checking next smallest edge.
- Had: {a b c e f h} {d} {g}
- {c} ≠ {d}, add edge.
- Now get {a b c d e f h} {g}

Kruskal’s Example



Graph (tabular form)

From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	6	10	9	2	3	8	15

↓ Sort

Heap (simplified view)

From	[redacted]	b	f
To		h	g
Weight		10	15

Disjoint set

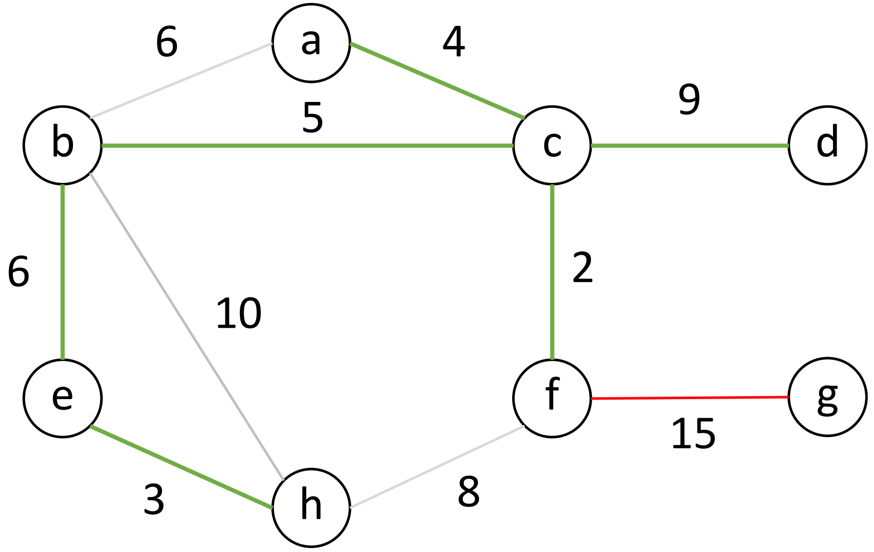
Index	0	1	2	3	4	5	6	7
Node	a	b	c	d	e	f	g	h
Parent	0	0	0	0	0	2	6	4

Tree (tabular form)

From	c	e	a	b	b	c				
To	f	h	c	c	e	d				
Weight	2	3	4	5	6	9				

- Keep going, checking next smallest edge.
- Had: {a b c d e f h} {g}
- {b} = {h}, do nothing.

Kruskal’s Example



Graph (tabular form)

From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	6	10	9	2	3	8	15

Sort

Heap (simplified view)

From	[redacted]	f
To	[redacted]	g
Weight	[redacted]	15

Disjoint set

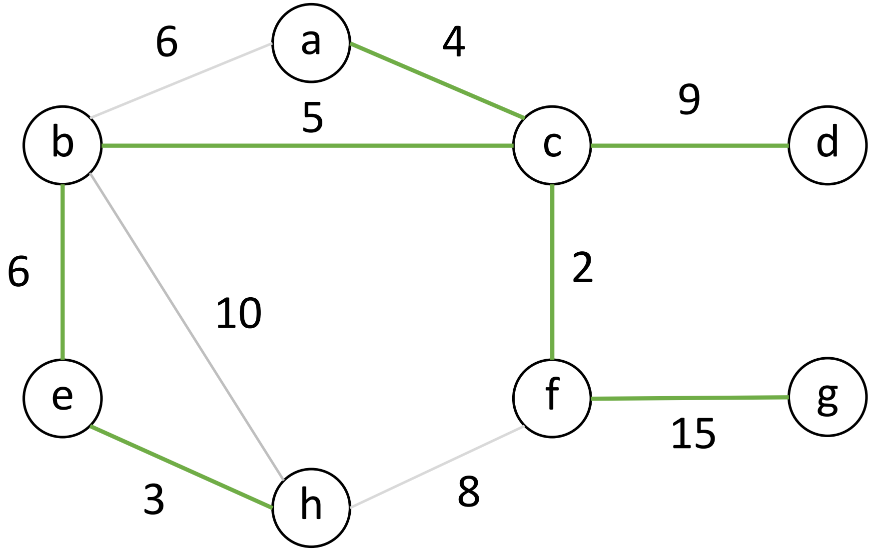
Index	0	1	2	3	4	5	6	7
Node	a	b	c	d	e	f	g	h
Parent	0	0	0	0	0	2	6	4

Tree (tabular form)

From	c	e	a	b	b	c	f			
To	f	h	c	c	e	d	g			
Weight	2	3	4	5	6	9	15			

- Keep going, checking next smallest edge.
- Had: {a b c d e f h} {g}
- {f} ≠ {g}, add edge.
- Now get {a b c d e f g h}

Kruskal's Example



Graph (tabular form)

From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	6	10	9	2	3	8	15

↓ Sort

Heap (simplified view)

From	[redacted]
To	
Weight	

Disjoint set

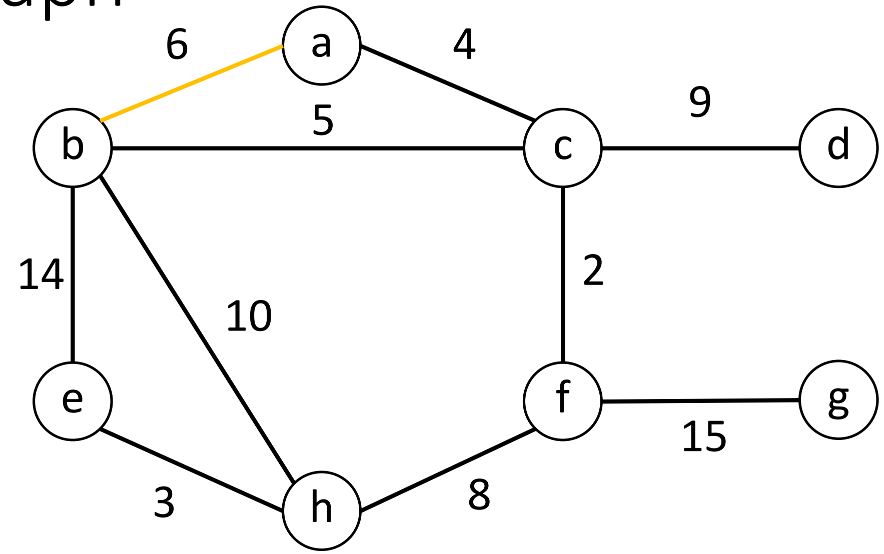
Index	0	1	2	3	4	5	6	7
Node	a	b	c	d	e	f	g	h
Parent	0	0	0	0	0	2	0	4

Tree (tabular form)

From	c	e	a	b	b	c	f			
To	f	h	c	c	e	d	g			
Weight	2	3	4	5	6	9	15			

- All edges were extracted!!!

Exercise :: Find the Kruskal’s minimum spanning tree of the given graph



Overall Runtime

Kruskal(G,w) ; Graph G, with weights w

A $\leftarrow$ {} ; Our MST starts empty

for each vertex $v \in V[G]$ do Make-Set(v) ; Make each vertex a set

Sort edges of E by increasing weight

for each edge $(u,v) \in E$ in order

; Find-Set returns a representative (first vertex) in the set

do if Find-Set(u) $\neq$ Find-Set(v)

then A $\leftarrow$ A $\cup \{(u,v)\}$

Union(u,v) ; Combines two trees

return A

O(V)

O(ElgE) – using heapsort

O(E)

O(1)

O(V)

Total runtime: O(V)+O(ElgE)+O(E*(1+V)) = O(E*V)

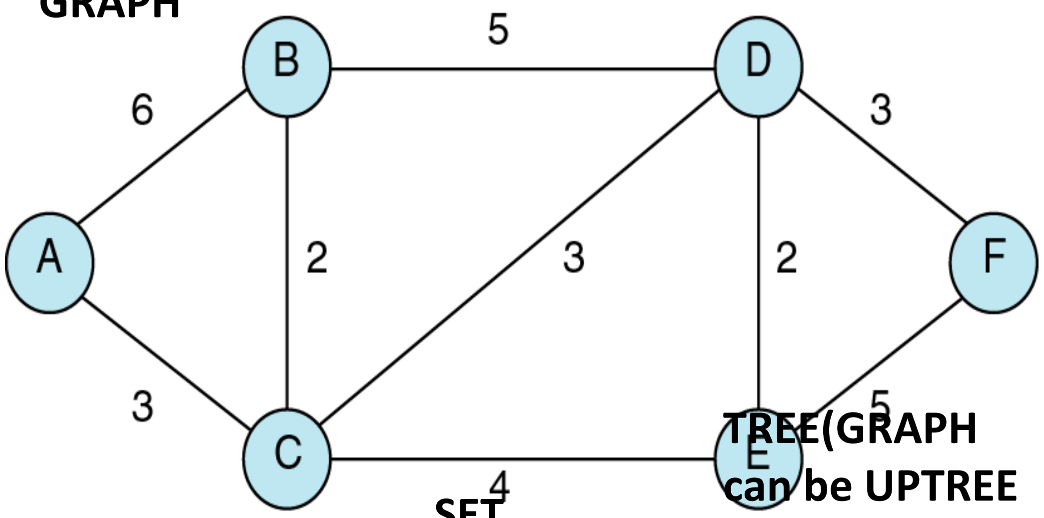
Book describes a version using disjoint sets that runs in O(E*lgE) time

Kruskal

- สร้าง Adjacency Matrix ก่อน
- ใช้ heap เก็บ edge ทุก edge ไว้
- สร้าง disjoint set
- ดึง edge ออกจาก heap ทีละ edge
  - ถ้า find(source)!=find(destination) →union(source,destination)

จงใช้ Kruskal หา Minimum Spanning Tree

GRAPH



Heap

SET

TREE(GRAPH can be UPTREE Non -compress)

SET

Sorted edge list
- B-C = 2
- D-E = 2
- A-C = 3
- C-D = 3
- D-F = 3
- C-E = 4
- B-D = 5
- E-F = 5
- A-B = 6

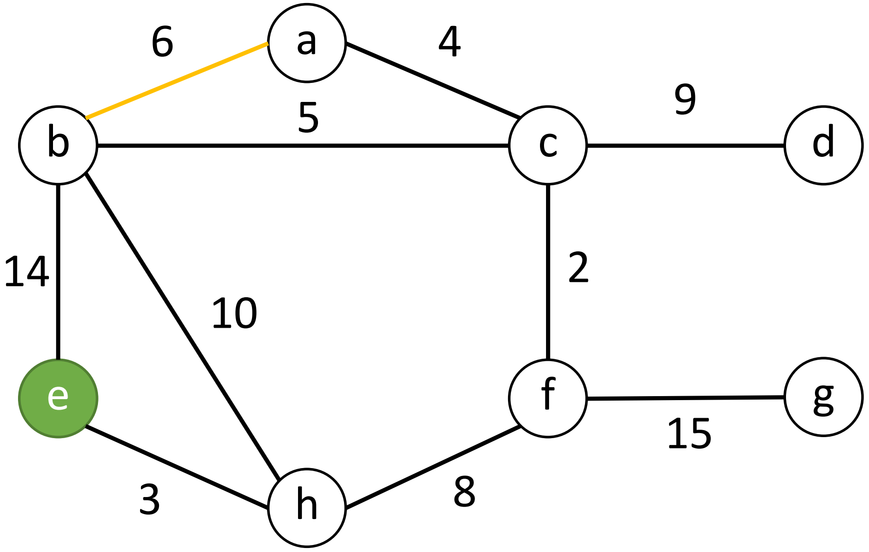
Steps
1. Find(B) != Find(C) , AddEdge(B,C), Union(B,C)
2. Find(D) != Find(E) , AddEdge(D,E), Union(D,E)
3. Find(A) != Find(C) , AddEdge(A,C), Union(A,B)
4. Find(C) != Find(D) , AddEdge(C,D), Union(A,D)
5. Find(D) != Find(F) , AddEdge(D,F), Union(A,F)
6. Find(C) == Find(E) , do nothing
7. Find(B) == Find(D) , do nothing
8. Find(E) == Find(F) , do nothing
9. Find(A) == Find(B) , do nothing

Prim’s MST Algorithm

- เหมือน Kruskal
- อาจได้ผลลัพธ์ที่แตกต่างถ้ามีคำตอบที่ดีที่สุดมากกว่าหนึ่งคำตอบ สามารถเขียนได้ทั้งแบบเลือกโหนดใกล้และโหนดแรกที่พบ กรณีเลือกโหนดใกล้จำเป็นต้องเปรียบเทียบจำนวน hop ด้วย

```
MST-Prim(G,w,r)                                   ; Graph G, weights w, root r
    Q ← V[G]
    for each vertex u ∈ Q do key[u] ← ∞           ; infinite “distance”
    key[r] ← 0
    P[r] ← NIL
    while Q<>NIL do
        u ← Extract-Min(Q)                        ; remove closest node
        ; Update children of u so they have a parent and a min key val
        ; the key is the weight between node and parent
        for each v ∈ Adj[u] do
            if v ∈ Q & w(u,v)<key[v] then
                P[v] ← u
                key[v] ← w(u,v)
```

Prim’s Example



Graph (tabular form)

From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	14	10	9	2	3	8	15

Heap (simplified view)

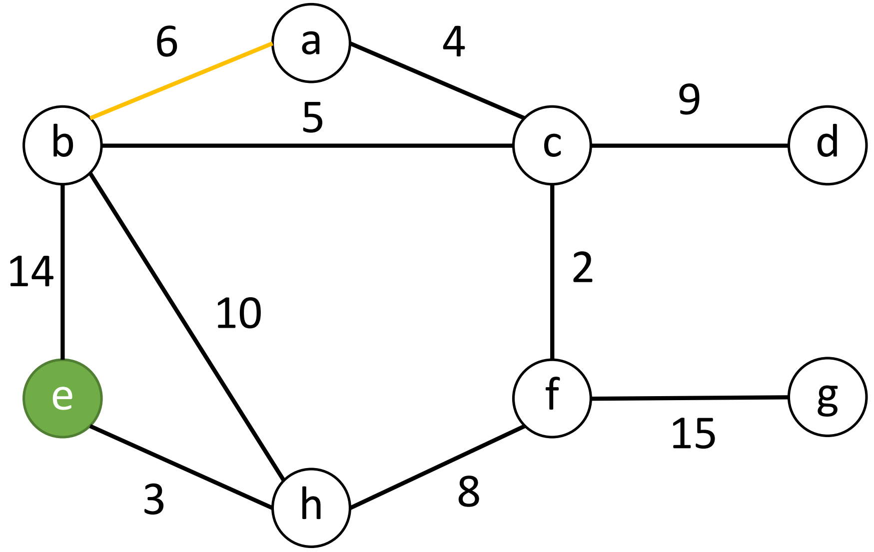
From	a	b	c	d	e	f	g	h
Weight	∞	∞	∞	∞	0	∞	∞	∞

Tree (tabular form)

From										
To										
Weight										

Extract min, vertex e. Update neighbor if in Q and weight < key.

Prim’s Example



Graph (tabular form)

From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	14	10	9	2	3	8	15

Heap (simplified view)

From	a	b	c	d	e	f	g	h
Weight	∞	14	∞	∞	0	∞	∞	3

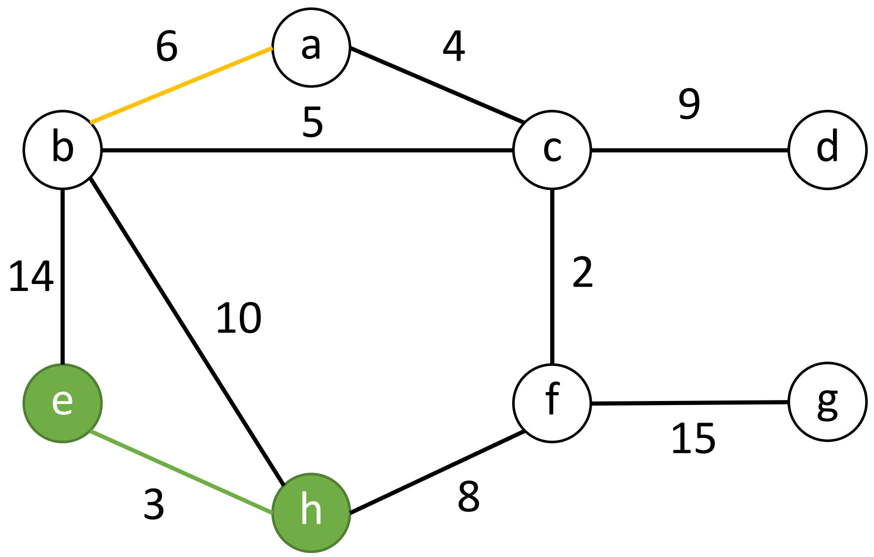
Tree (tabular form)

From	e									
To	e									
Weight	0									

Extract min, vertex e. Update neighbor if in Q and weight < key.

DecreaseKey(b,14)
DecreaseKey(h,3)

Prim’s Example



Graph (tabular form)

From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	14	10	9	2	3	8	15

Heap (simplified view)

From	a	b	c	d	e	f	g	h
Weight	∞	10	∞	∞	0	8	∞	3

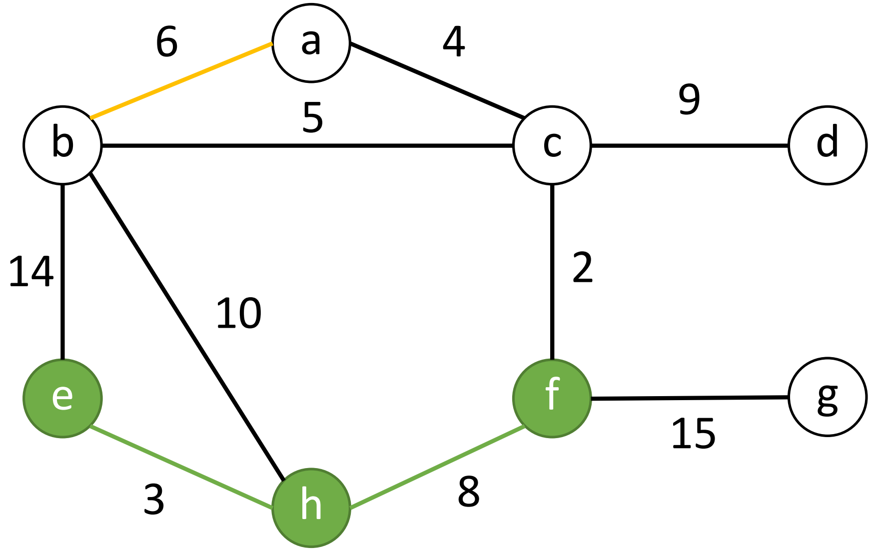
Tree (tabular form)

From	e	e								
To	e	h								
Weight	0	3								

Extract min, vertex h. Update neighbor if in Q and weight < key.

DecreaseKey(b,10)
DecreaseKey(f,8)

Prim’s Example



Graph (tabular form)

From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	14	10	9	2	3	8	15

Heap (simplified view)

From	a	b	c	d	e	f	g	h
Weight	∞	10	2	∞	0	8	15	3

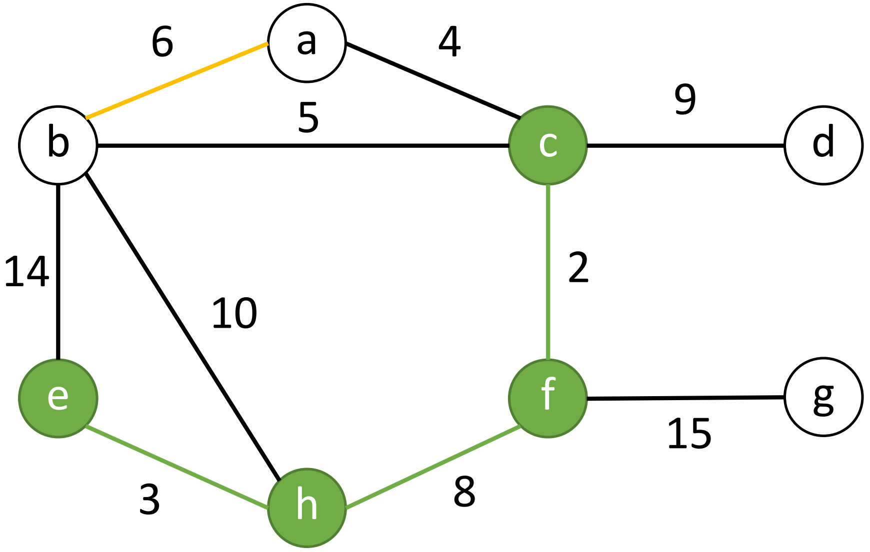
Tree (tabular form)

From	e	e	h							
To	e	h	f							
Weight	0	3	8							

Extract min, vertex f. Update neighbor if in Q and weight < key.

DecreaseKey(c,2)
DecreaseKey(g,15)

Prim’s Example



Graph (tabular form)

From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	14	10	9	2	3	8	15

Heap (simplified view)

From	a	b	c	d	e	f	g	h
Weight	4	5	2	9	0	8	15	3

Tree (tabular form)

From	e	e	h	f						
To	e	h	f	c						
Weight	0	3	8	2						

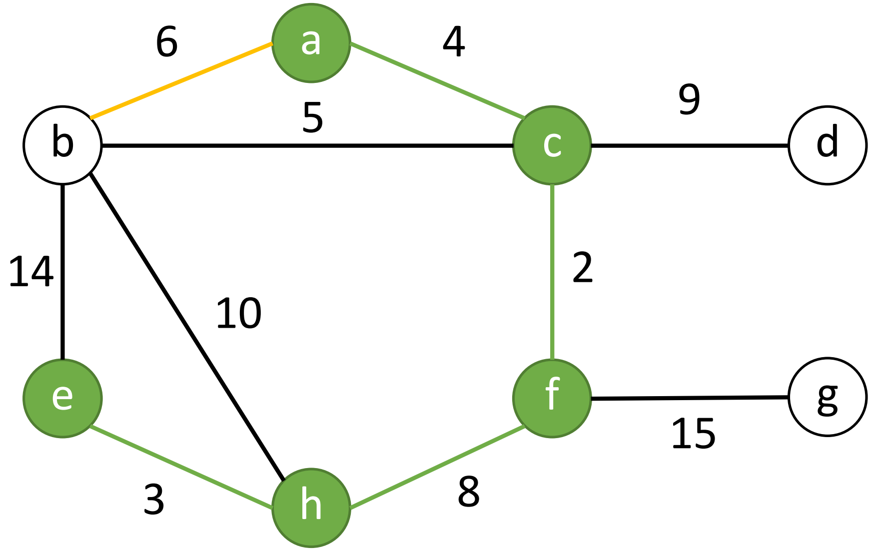
Extract min, vertex c. Update neighbor if in Q and weight < key.

DecreaseKey(a,4)

DecreaseKey(b,5)

DecreaseKey(d,9)

Prim’s Example



Graph (tabular form)

From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	14	10	9	2	3	8	15

Heap (simplified view)

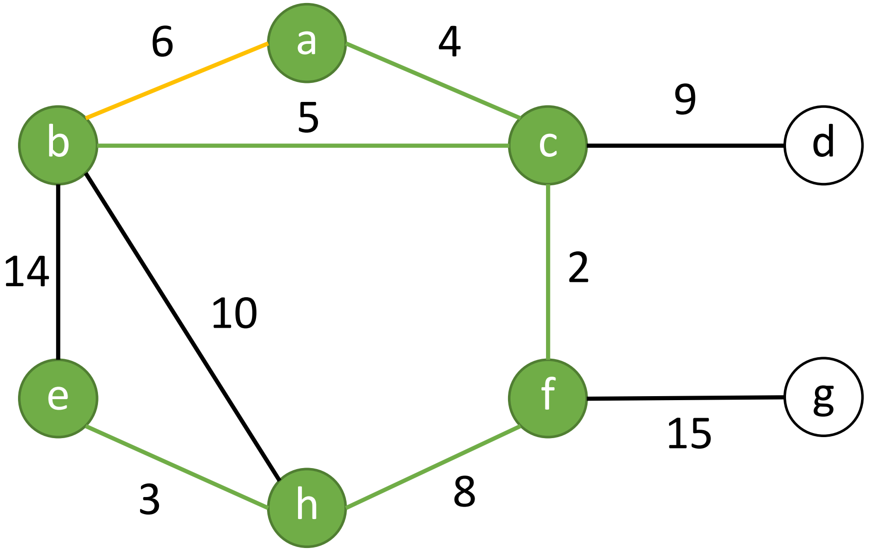
From	a	b	c	d	e	f	g	h
Weight	4	5	2	9	0	8	15	3

Tree (tabular form)

From	e	e	h	f	c					
To	e	h	f	c	a					
Weight	0	3	8	2	4					

Extract min, vertex a. Update neighbor if in Q and weight < key.

Prim’s Example



Graph (tabular form)

From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	14	10	9	2	3	8	15

Heap (simplified view)

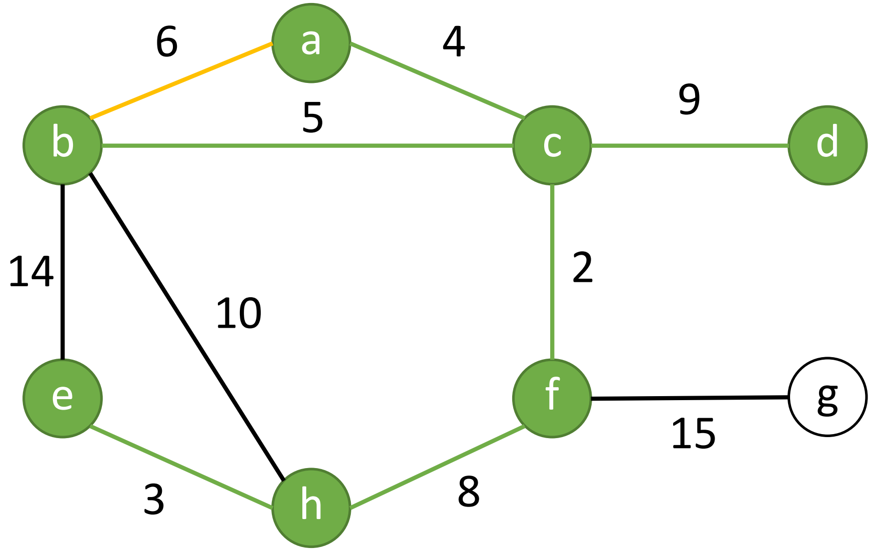
From	a	b	c	d	e	f	g	h
Weight	4	5	2	9	0	8	15	3

Tree (tabular form)

From	e	e	h	f	c	c				
To	e	h	f	c	a	b				
Weight	0	3	8	2	4	5				

Extract min, vertex b. Update neighbor if in Q and weight < key.

Prim’s Example



Graph (tabular form)

From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	14	10	9	2	3	8	15

Heap (simplified view)

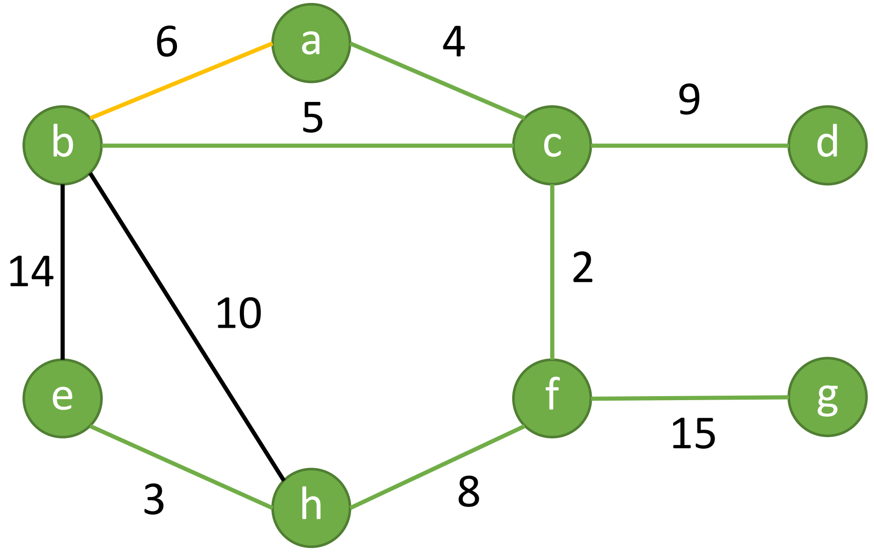
From	a	b	c	d	e	f	g	h
Weight	4	5	2	9	0	8	15	3

Tree (tabular form)

From	e	e	h	f	c	c	c			
To	e	h	f	c	a	b	d			
Weight	0	3	8	2	4	5	9			

Extract min, vertex d. Update neighbor if in Q and weight < key.

Prim’s Example



Graph (tabular form)

From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	14	10	9	2	3	8	15

Heap (simplified view)

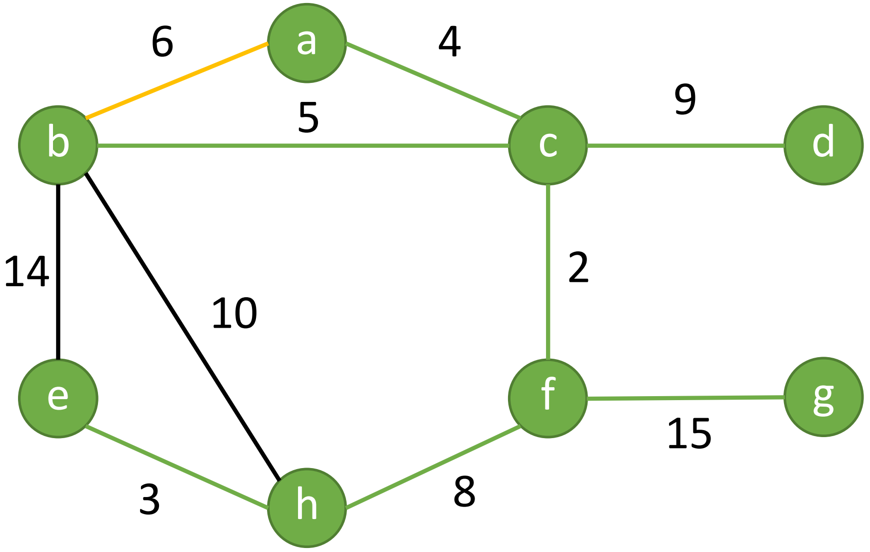
From	a	b	c	d	e	f	g	h
Weight	4	5	2	9	0	8	15	3

Tree (tabular form)

From	e	e	h	f	c	c	c	f		
To	e	h	f	c	a	b	d	g		
Weight	0	3	8	2	4	5	9	15		

Extract min, vertex g. Update neighbor if in Q and weight < key.

Prim’s Example



Heap is empty!!!

Graph (tabular form)

From	a	a	b	b	b	c	c	e	f	f
To	b	c	c	e	h	d	f	h	h	g
Weight	6	4	5	14	10	9	2	3	8	15

Heap (simplified view)

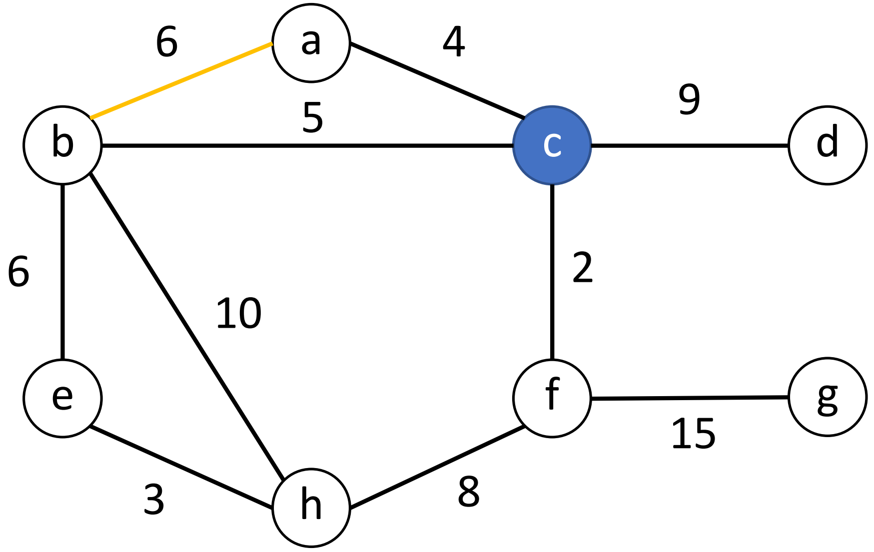
From	a	b	c	d	e	f	g	h
Weight	4	5	2	9	0	8	15	3

Tree (tabular form)

From	e	e	h	f	c	c	c	f		
To	e	h	f	c	a	b	d	g		
Weight	0	3	8	2	4	5	9	15		

Exercise :: Find the Prim's minimum spanning tree of the given graph

Begin with node c



Runtime for Prim's Algorithm

```
MST-Prim(G,w,r)                               ; Graph G, weights w, root r
    Q ← V[G]
    for each vertex u ∈ Q do key[u] ← ∞       ; infinite "distance"
    key[r] ← 0
    P[r] ← NIL
    while Q<>NIL do
        u ← Extract-Min(Q)                    ; remove closest node
        ; Update children of u so they have a parent and a min key val
        ; the key is the weight between node and parent
        for each v ∈ Adj[u] do
            if v ∈ Q & w(u,v)<key[v] then
                P[v] ← u
                key[v] ← w(u,v)
```

O(V) if using a heap

O(lgV) if using a heap

O(V)

O(E) over entire while(Q<>NIL) loop

O(lgV) to update if using a heap!

The inner loop takes O(E lg V) for the heap update inside the O(E) loop.
This is over all executions, so it is not multiplied by O(V) for the while loop
(this is included in the O(E) runtime through all edges.

The Extract-Min requires O(V lg V) time.
O(lg V) for the Extract-Min and O(V) for the while loop.

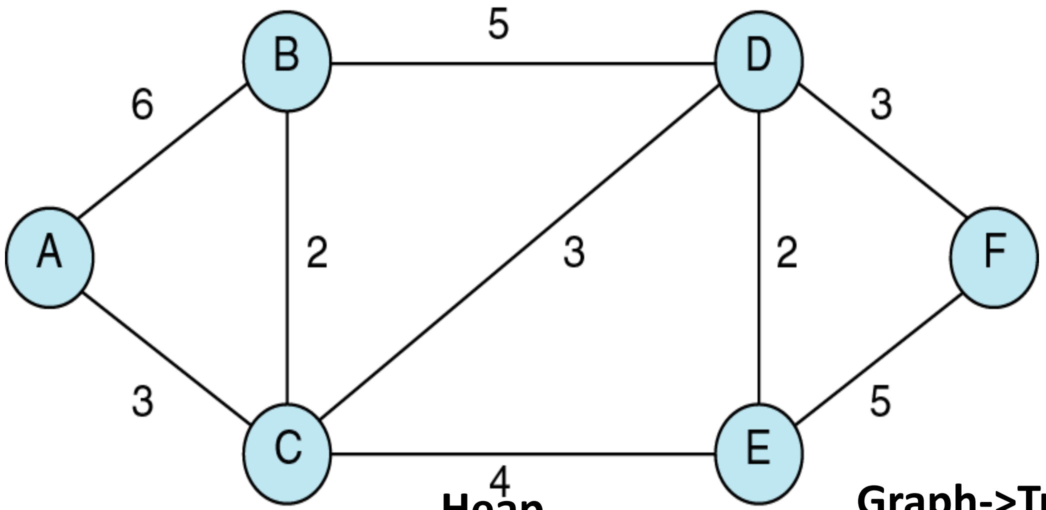
Total runtime is then O(V lg V) + O(E lg V) which is O(E lg V)
in a connected graph
(a connected graph will always have at least V-1 edges).

Prim

- สร้าง Adjacency Matrix ก่อน
- สร้าง tree ให้ node แรกเป็น root node
- Span แล้วใช้ heap เก็บ open node ทุก node ไว้บวกระยะทาง
- ดึง node ออกจาก heap ไปใส่ในอีก Adjacency Matrix แล้ว span จาก node ที่ดึง
    - Union node ใน disjoint set

จงใช้ Prim หา Minimum Spanning Tree

เริ่มจากโหนด A ทำการสร้างตารางจัดเก็บระยะจากแต่ละปลายโหนดเก็บใส่ heap

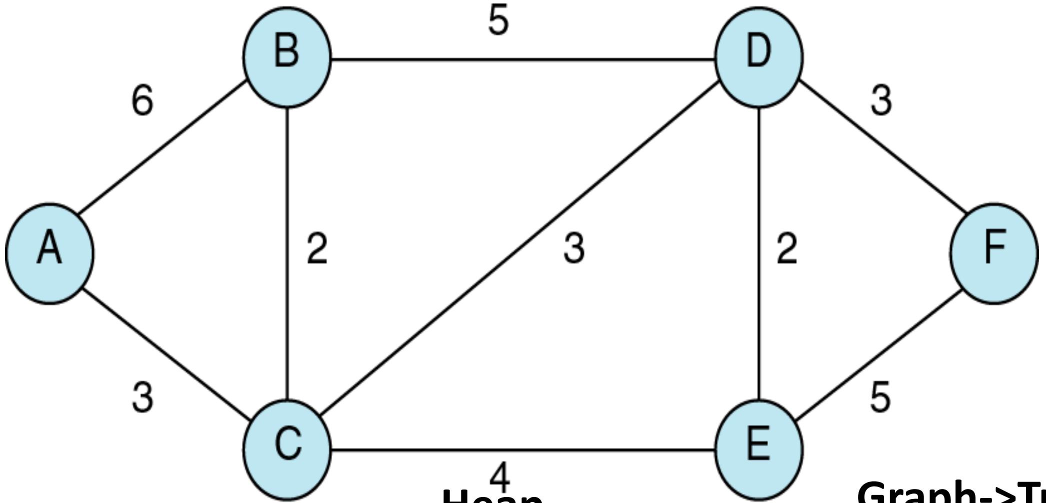


Heap **Heap** **Graph->Tree** **Heap**

Steps
1. [A,A,0] Span(A)
2. [B,A,6],[C,A,3] ExtractMin = [C,A,3], Span(C), [B,C,2]<[B,A,6], decreaseKey(B,C,2)
3. [B,C,2],[D,C,3],[E,C,4] ExtractMin = [B,C,2], Span(B), [D,B,5]>[D,C,3], do nothing
4. [D,C,3],[E,C,4] ExtractMin = [D,C,3], Span(D), [E,D,2]<[E,C,4], decreaseKey(E,D,2)
5. [E,D,2],[F,D,3] ExtractMin = [E,D,2], Span(E), [F,E,5]>[F,D,3], do nothing
6. [F,D,3] ExtractMin = [F,D,3], Span(F)

จงใช้ Prim หา Minimum Spanning Tree

เริ่มจากโหนด A ทำการสร้างตารางจัดเก็บระยะจากแต่ละปลายโหนดเก็บใส่ heap



Heap **Heap** **Graph->Tree** **Heap**

Steps

1.	[D,D,0]		Span(D)
2.	[B,D,5],[C,D,3],[E,D,2],[F,D,3]	ExtractMin = [E,D,2],	Span(E), [C,E,4]>[C,D,3],[F,E,5]>[F,D,3] do nothing
3.	[B,D,5],[C,D,3],[F,D,3]	ExtractMin = [F,D,3],	Span(F)
4.	[B,D,5],[C,D,3]	ExtractMin = [C,D,3],	Span(C), [B,C,2]<[B,D,5], decreaseKey [B,C,2]
5.	[B,C,2],[A,C,3]	ExtractMin = [B,C,2],	Span(B), [A,B,6]>[A,C,2], do nothing
6.	[A,C,3]	ExtractMin = [A,C,3],	Span(A)

Edsger W. Dijkstra (1930-2002)

- Dutch Computer Scientist
- Received Turing Award for contribution to developing programming languages.

Contributed to :

- Shortest path-algorithm, also known as Dijkstra's algorithm;
- Reverse Polish Notation and related Shunting yard algorithm; t
- THE multiprogramming system;
- Banker's algorithm;
- Self-stabilization – an alternative way to ensure the reliability of the system.

Dijkstra’s algorithm

- Dijkstra’s algorithm for finding the shortest path in a graph
  - Always takes the *shortest* edge connecting a known node to an unknown node
- Kruskal’s algorithm for finding a minimum-cost spanning tree
  - Always tries the *lowest-cost* remaining edge
- Prim’s algorithm for finding a minimum-cost spanning tree
  - Always takes the *lowest-cost* edge between nodes in the spanning tree and nodes not yet in the spanning tree

Dijkstra’s shortest-path algorithm

- Dijkstra’s algorithm finds the shortest paths from a given node to all other nodes in a graph
  - Initially,
    - Mark the given node as *known* (path length is zero)
    - For each out-edge, set the distance in each neighboring node equal to the *cost* (length) of the out-edge, and set its *predecessor* to the initially given node
  - Repeatedly (until all nodes are known),
    - Find an unknown node containing the smallest distance
    - Mark the new node as known
    - For each node adjacent to the new node, examine its neighbors to see whether their estimated distance can be reduced (distance to known node plus cost of out-edge)
      - If so, also reset the predecessor of the new node

Dijkstra’s shortest-path algorithm

```
function Dijkstra(Graph, source):
    for each vertex v in Graph.Vertices:
        dist[v] ← INFINITY
        prev[v] ← UNDEFINED
        add v to Q
    dist[source] ← 0
    while Q is not empty:
        u ← vertex in Q with min dist[u]
        remove u from Q
        for each neighbor v of u still in Q:
            alt ← dist[u] + Graph.Edges(u, v)
            if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u
    return dist[], prev[]
```

Repeatedly (until all nodes are known),

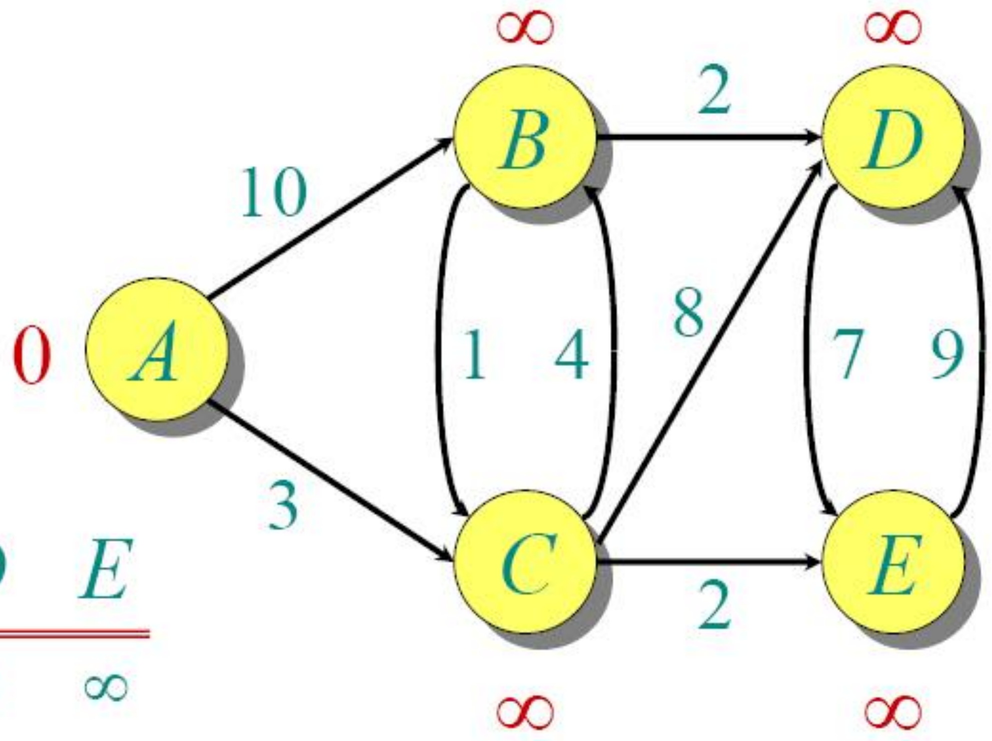
- Find an unknown node containing the smallest distance
- Mark the new node as known
- For each node adjacent to the new node, examine its neighbors to see whether their estimated distance can be reduced (distance to known node plus cost of out-edge)
    - If so, also reset the predecessor of the new node

แบบฝึกหัด จงเปลี่ยน pseudocode นี้จาก Queue เป็น Heap แล้ววิเคราะห์ Time Complexity ด้วยตัวเอง

```
function Dijkstra(Graph, source):
    for each vertex v in Graph.Vertices:
        dist[v] ← INFINITY
        prev[v] ← UNDEFINED
        add v to Q
    dist[source] ← 0
    while Q is not empty:
        u ← vertex in Q with min dist[u]
        remove u from Q
        for each neighbor v of u still in Q:
            alt ← dist[u] + Graph.Edges(u, v)
            if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u
    return dist[], prev[]
```

Dijkstra’s shortest-path example

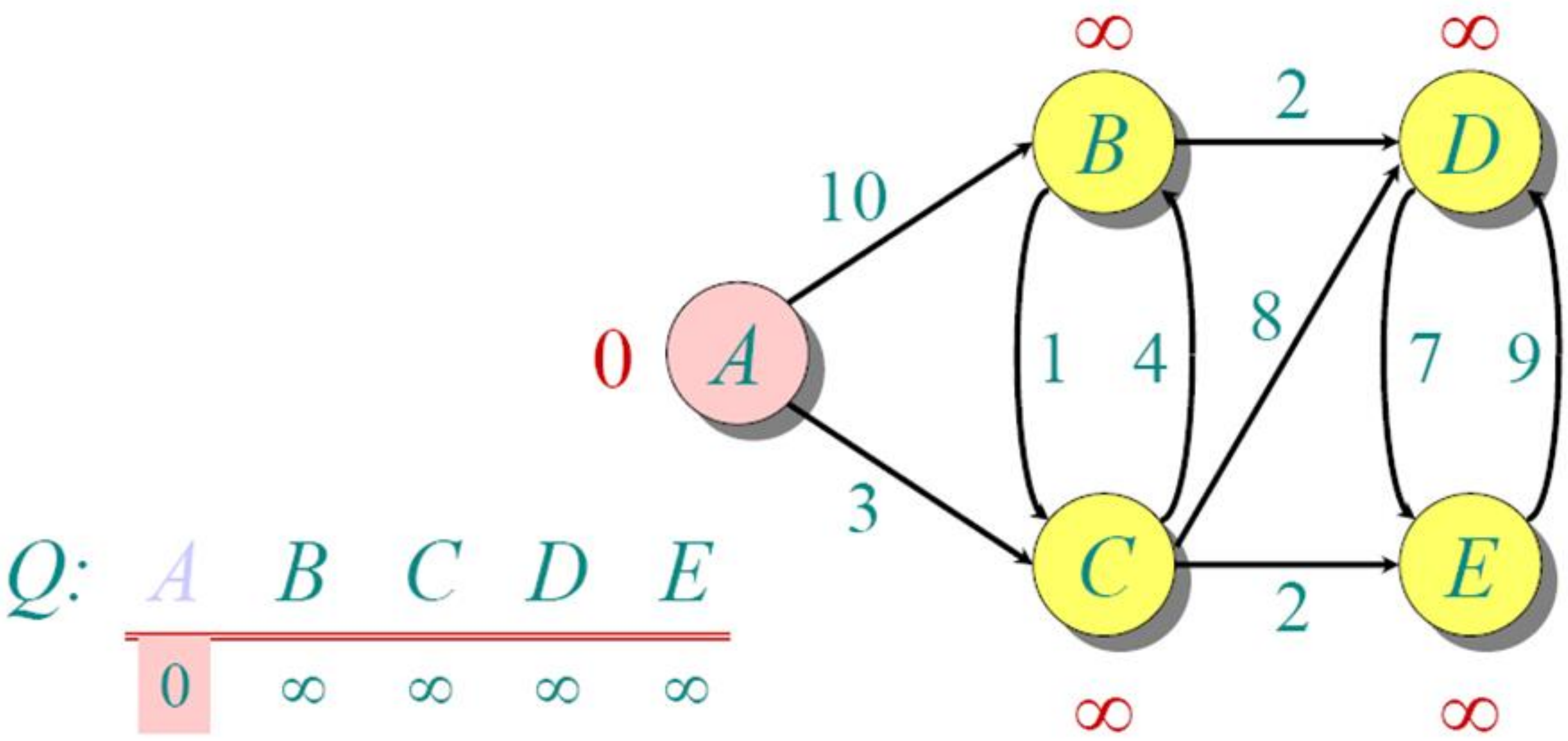
Initialize:



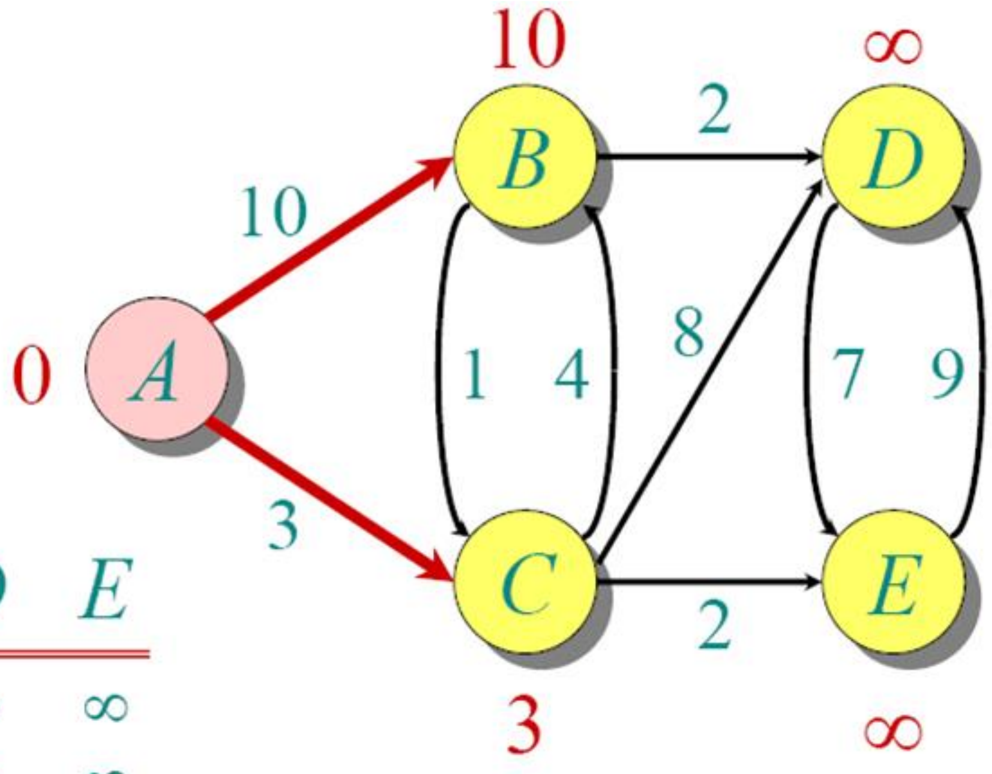
Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: {}

Dijkstra’s shortest-path example



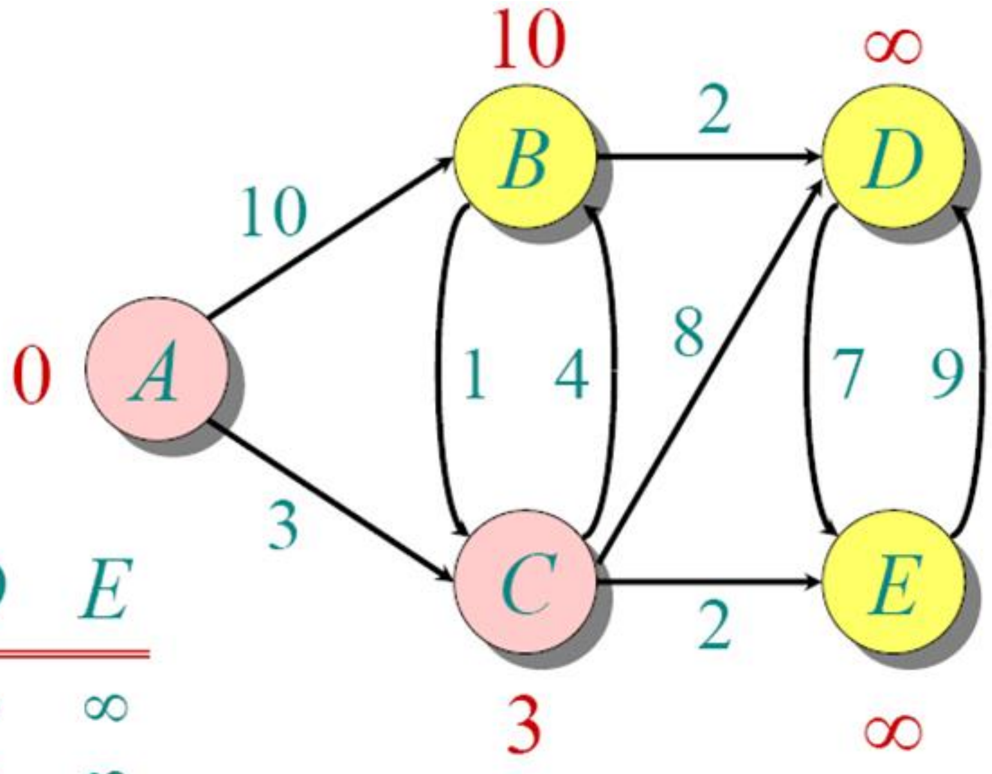
Dijkstra’s shortest-path example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

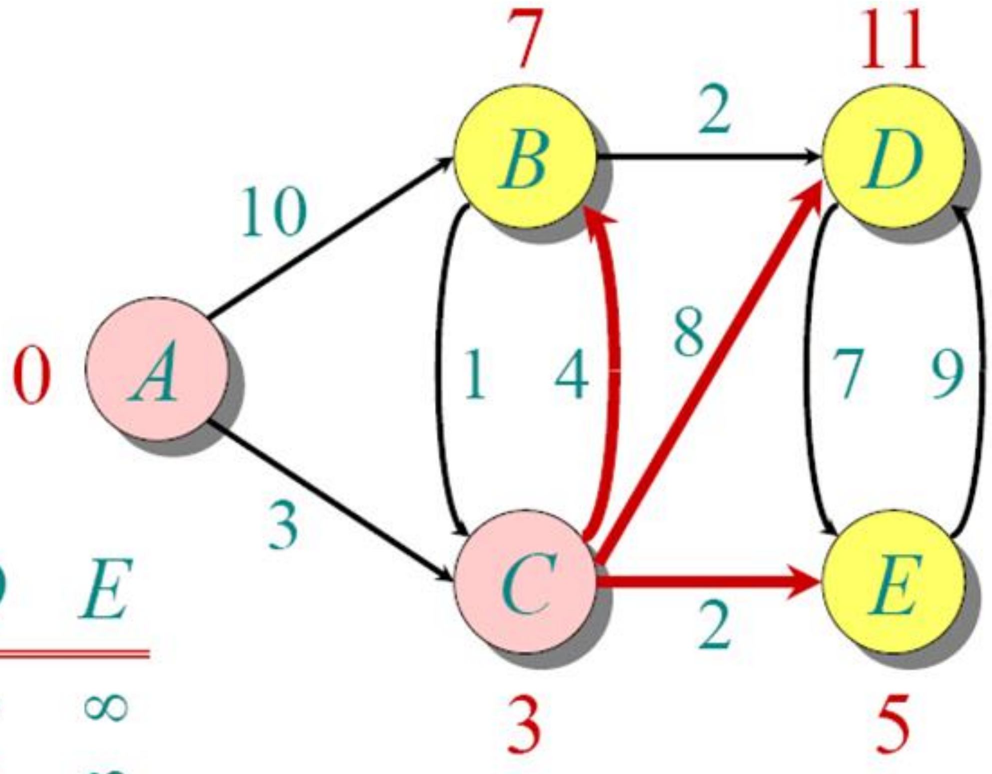
Dijkstra’s shortest-path example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A, C }

Dijkstra’s shortest-path example

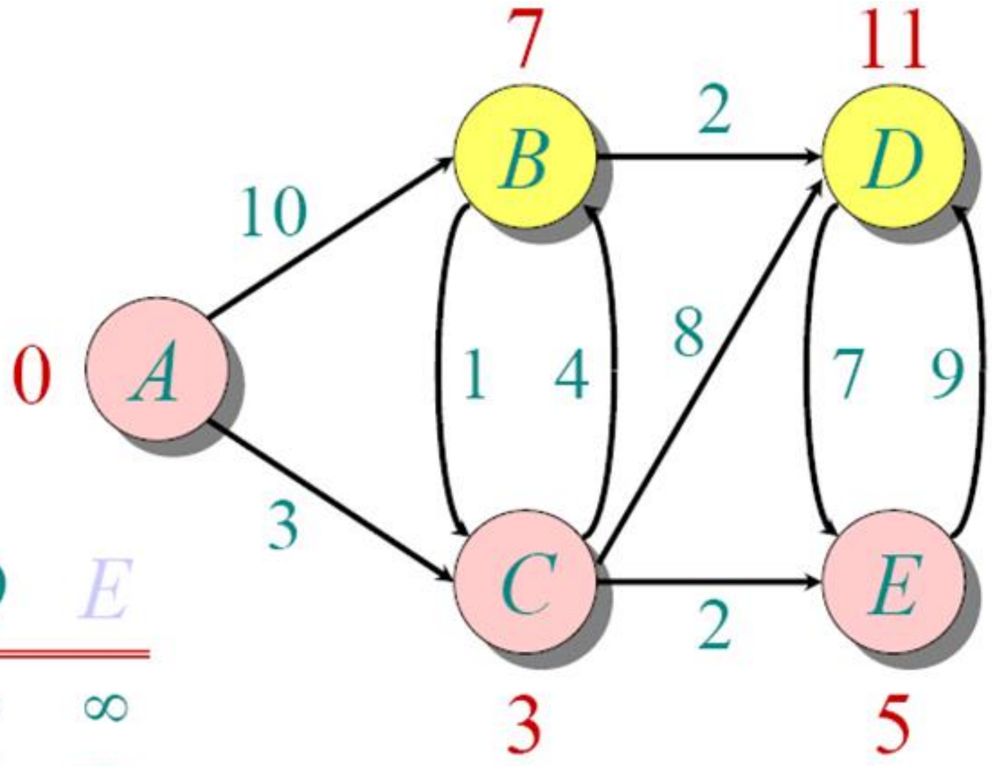


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C }

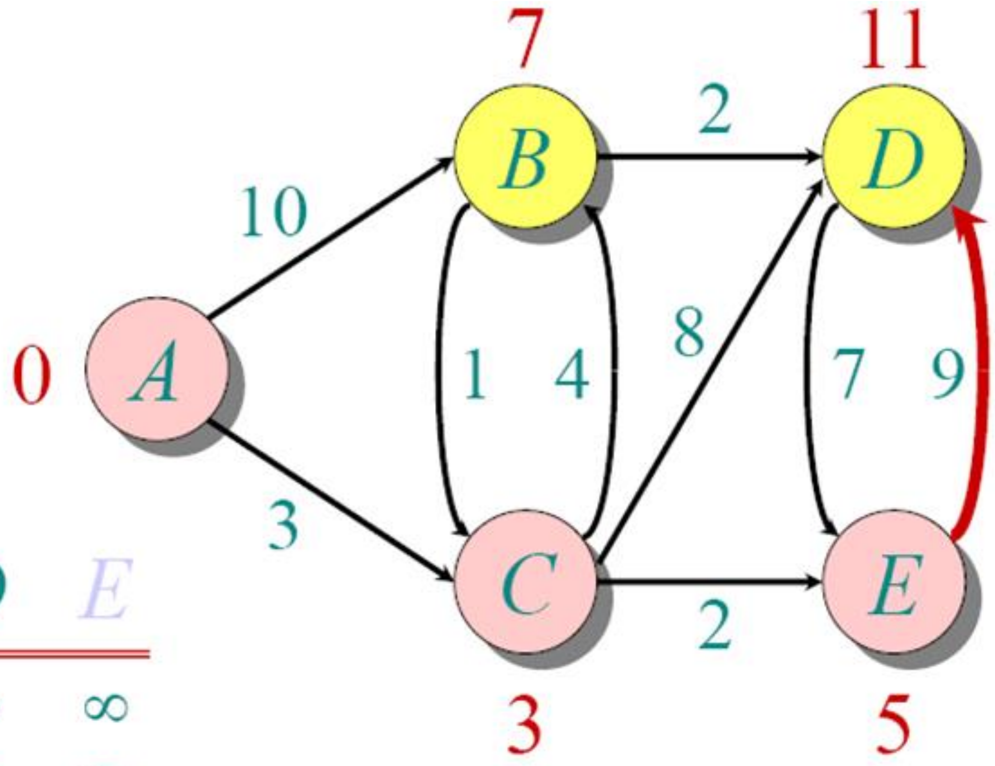
Dijkstra’s shortest-path example

Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5



S: { A, C, E }

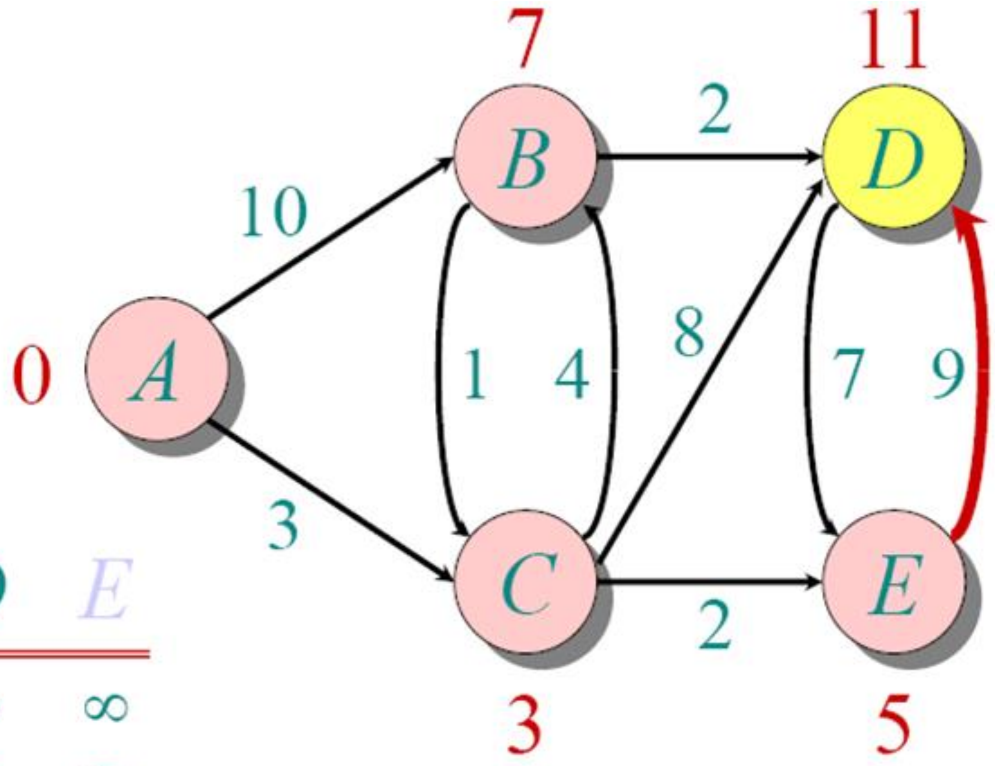
Dijkstra’s shortest-path example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E }

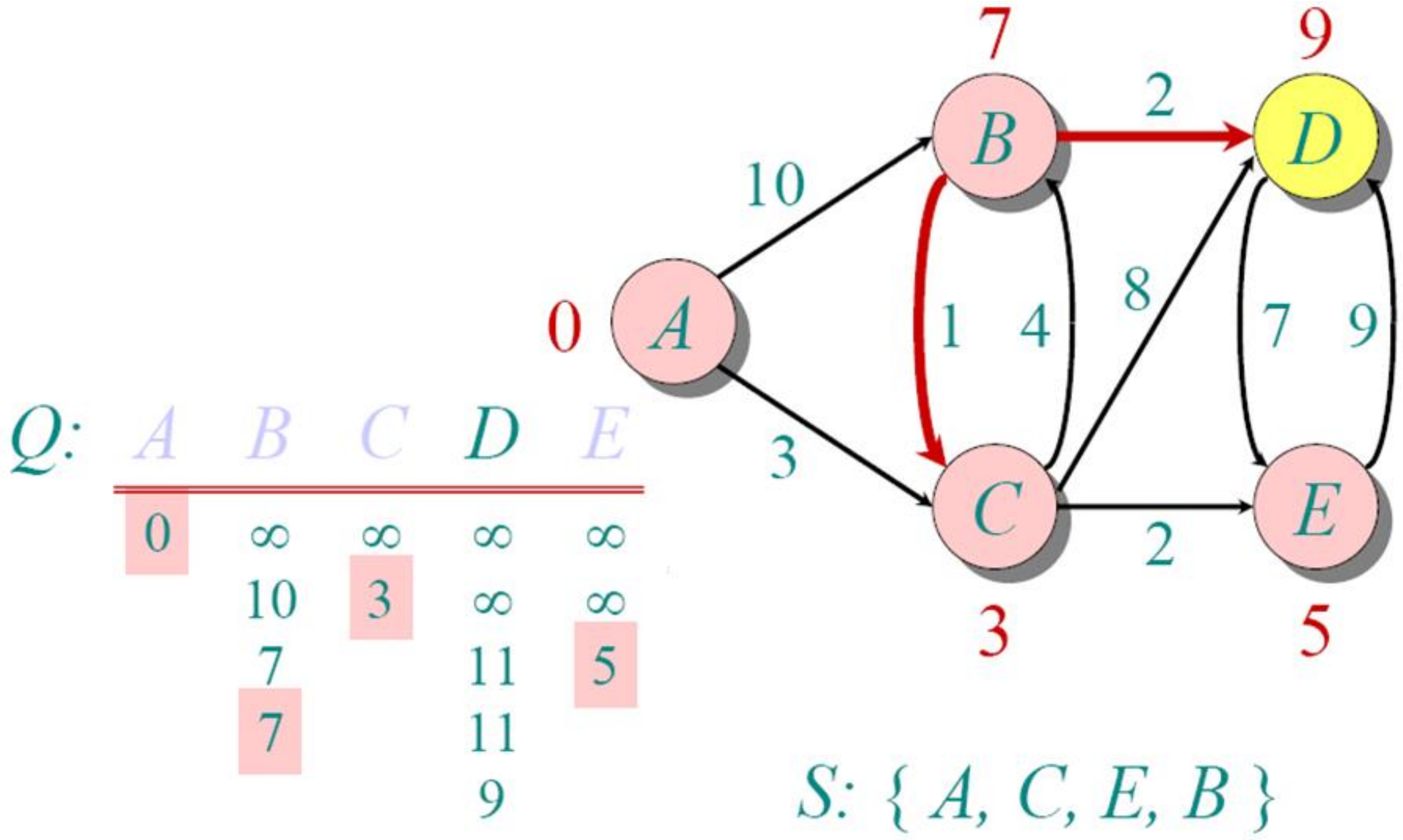
Dijkstra’s shortest-path example



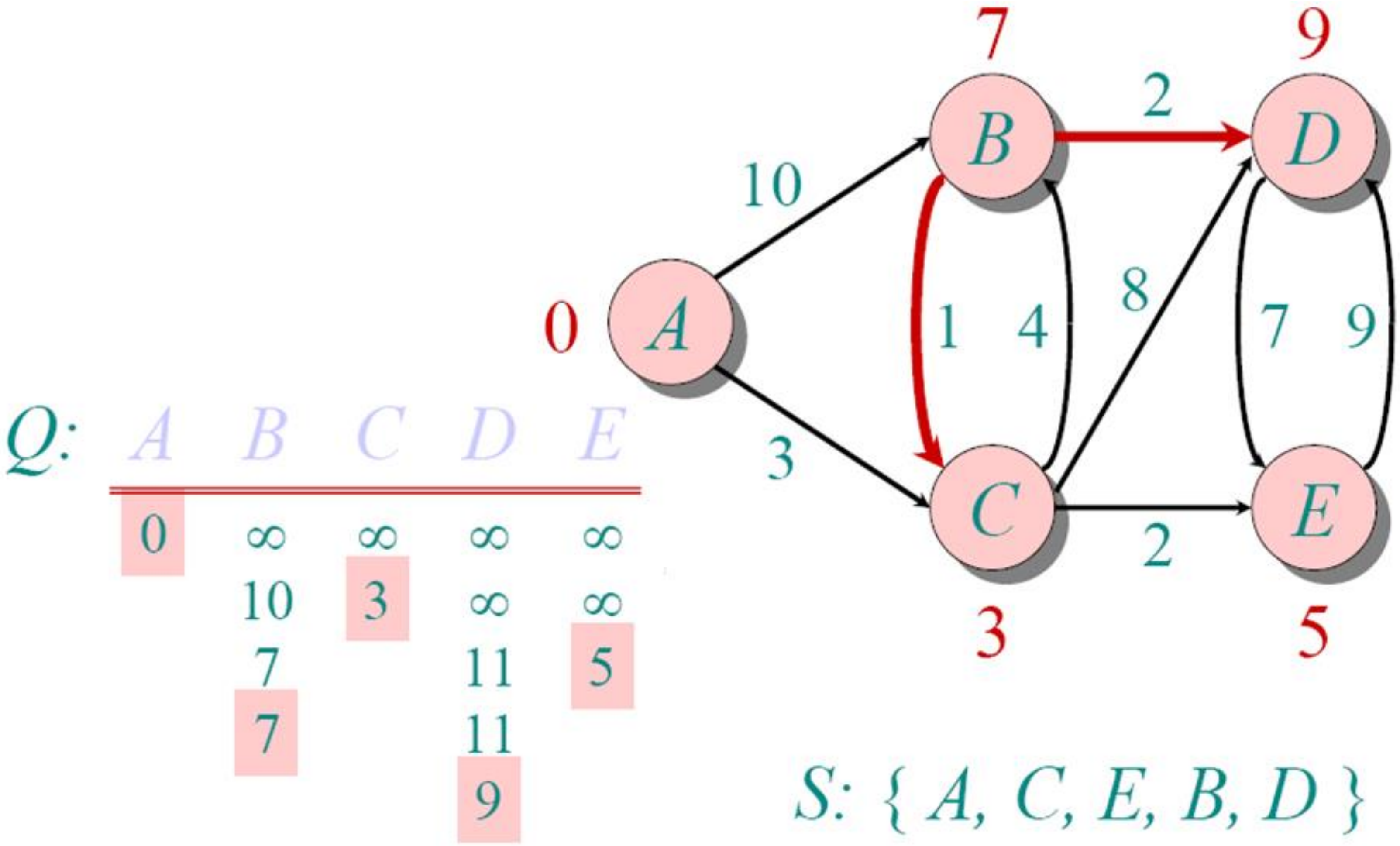
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E, B }

Dijkstra’s shortest-path example



Dijkstra’s shortest-path example



Implementations and Running Times

การออกแบบโปรแกรมที่ง่ายที่สุดคือการจัดเก็บ node หรือ vertex ใน array list หรือ linked list จะใช้เวลาทำงานเท่ากับ

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

สำหรับกราฟที่ไม่ค่อยหนาแน่น (กราฟที่มี edge น้อยมากและจำนวน node มาก) สามารถออกแบบให้มีประสิทธิภาพมากขึ้นโดยใช้ priority queue (อาจจะเป็น binary heap) จะใช้เวลาทำงานเท่ากับ

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

Dijkstra's Algorithm - Why It Works

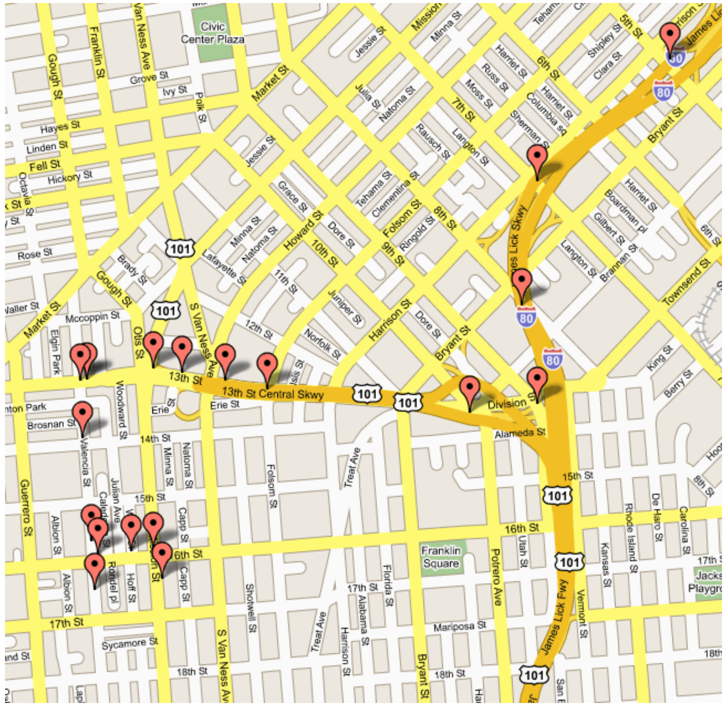
- To understand how it works, we'll go over the previous example again. However, we need two mathematical results first:

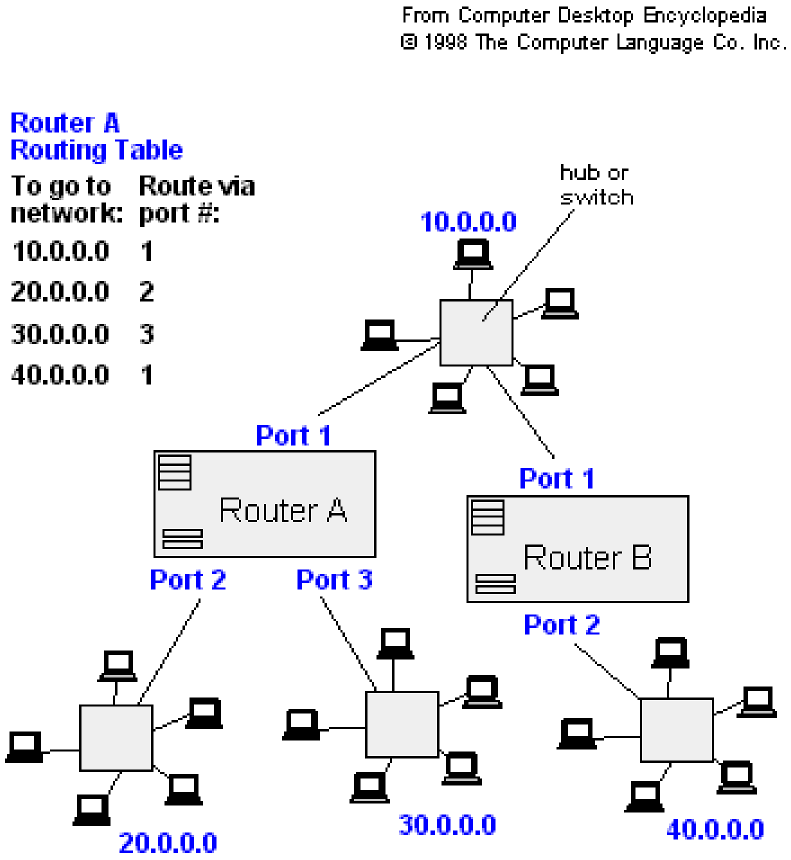
- **Lemma 1**: Triangle inequality
  If $\delta(u,v)$ is the shortest path length between u and v,
  $$\delta(u,v) \le \delta(u,x) + \delta(x,v)$$
- **Lemma 2**:
  The subpath of any shortest path is itself a shortest path.

- 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.

Applications of Dijkstra's Algorithm

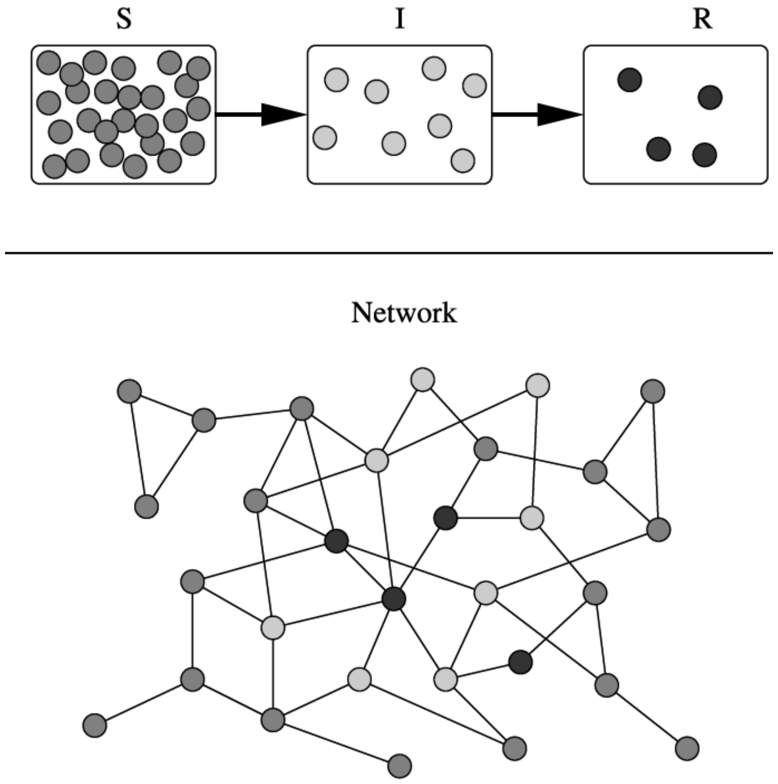
- Traffic Information Systems are most prominent use
- Mapping (Map Quest, Google Maps)
- Routing Systems





Applications of Dijkstra's Algorithm

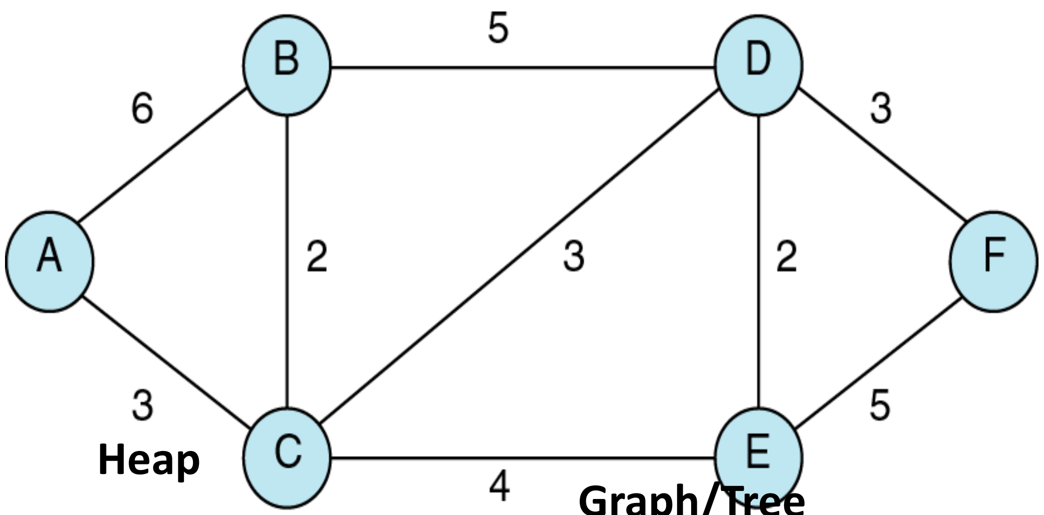
- We can use networks to model the spread of infectious diseases and design prevention and response strategies.
- Vertices represent individuals, and edges their possible contacts. It is useful to calculate how a particular individual is connected to others.
- Knowing the shortest path lengths to other individuals can be a relevant indicator of the potential of a particular individual to infect others.



Dijkstra

- สร้าง Adjacency Matrix ก่อน
- สร้าง up tree ให้ node แรกเป็น root node
- Span แล้วใช้ heap เก็บ open node ทุก node ไว้บวกระยะทาง
- ดึง node ออกจาก heap ไปใส่ในอีก Adjacency Matrix แล้ว span จาก node ที่ดึง
  - เมื่อพบ node เป้าหมายให้ trace back path จาก up tree

จงหา Shortest Path จาก A ไป F



Heap

Heap

Graph/Tree

Heap

Steps	
1. [A,A,0]	Span(A)
2. [B,A,6],[C,A,3]	ExtractMin = [C,A,3], Span(C), [B,C,5]<[B,A,6], decreaseKey(B,C,5)
3. [B,C,5],[D,C,6],[E,C,7]	ExtractMin = [B,C,5], Span(B), [D,B,10]>[D,C,6], do nothing
4. [D,C,6],[E,C,7]	ExtractMin = [D,C,6], Span(D), [E,D,8]>[E,C,7], do nothing
5. [E,C,7],[F,D,9]	ExtractMin = [E,C,7], Span(E), [F,E,12]>[F,D,9], do nothing
6. [F,D,9]	ExtractMin = [F,D,9], Span(F)

Trace Back

C→A

B→C→A

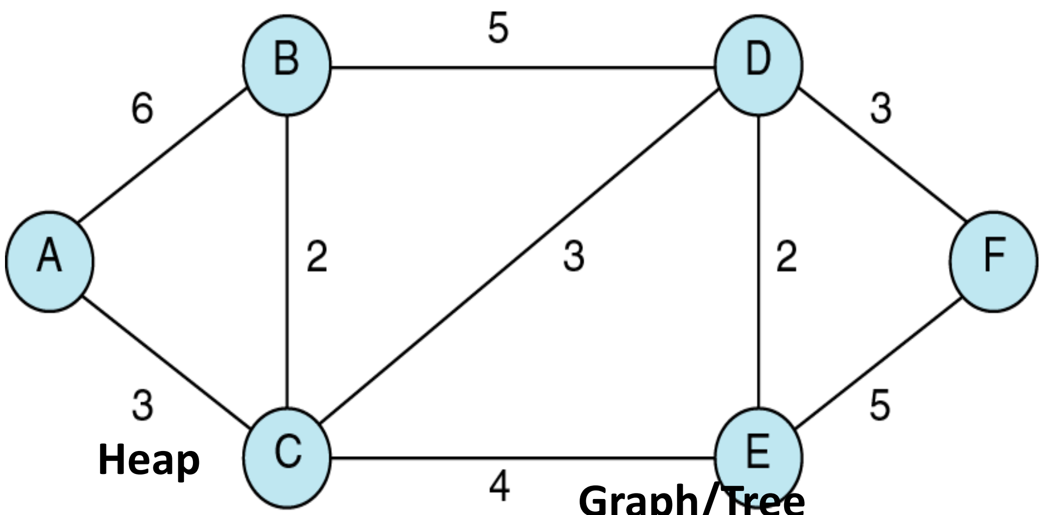
D→C→A

E→C→A

F→D→C→A

UpTree

จงหา Shortest Path จาก F ไป B



Heap

Heap

Graph/Tree

Heap

Steps
1. [F,F,0] Span(F)
2. [D,F,3],[E,F,5] ExtractMin = [D,F,3], Span(D), [ED5]=[EF5], do nothing
3. [E,F,5],[B,D,8],[C,D,6] ExtractMin = [E,F,5], Span(E), [C,D,6]<[C,E,9] do nothing
4. [B,D,8],[C,D,6] ExtractMin = [C,D,6], Span(C),[B,D,8]=[B,C,8]
5. [B,D,8],[A,C,9] ExtractMin = [B,D,8], Span(B)

Trace Back

D→F

E→F

C→D→F

B→D→F

UpTree

Huffman Coding

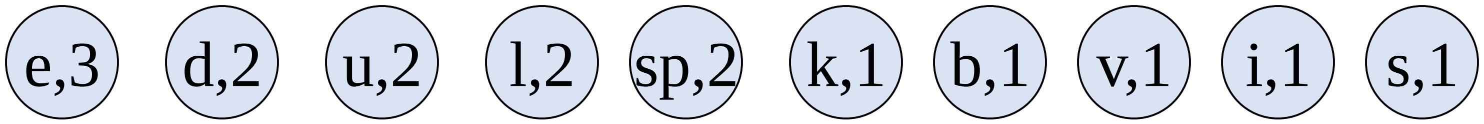
- Huffman codes ใช้สำหรับการบีบอัดข้อมูลทั้งการจัดเก็บและการส่งข้อมูล
  - JPEGs นำมาใช้ในการบีบอัดรูปภาพ
- แนวคิดคือแทนที่จะเก็บข้อมูลด้วย ASCII เราจะใช้ตัวอักษรที่มีความถี่มากๆด้วยการใช้จำนวนบิตน้อยๆ ในการเข้ารหัส โดยเฉลี่ยแล้วเราสามารถลดขนาดของไฟล์ได้ประมาณครึ่งนึง

Huffman Coding

- As an example, lets take the string:

  “duke blue devils”

- นับความถี่ของตัวอักษรเอาไว้: ใช้ heap จะเร็วดีเพราะต้อง sort
  - e:3, d:2, u:2, l:2, space:2, k:1, b:1, v:1, i:1, s:1
- แล้วใช้ขั้นตอนวิธีเชิงละโมบในการสร้าง Huffman Tree

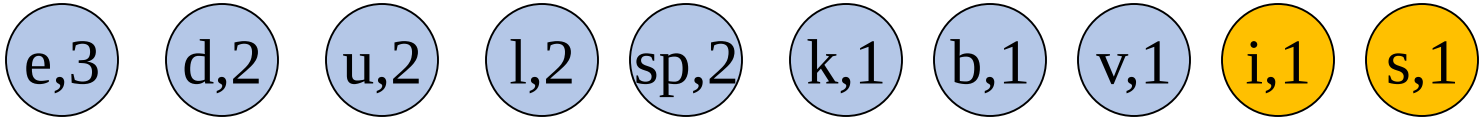


Huffman Coding

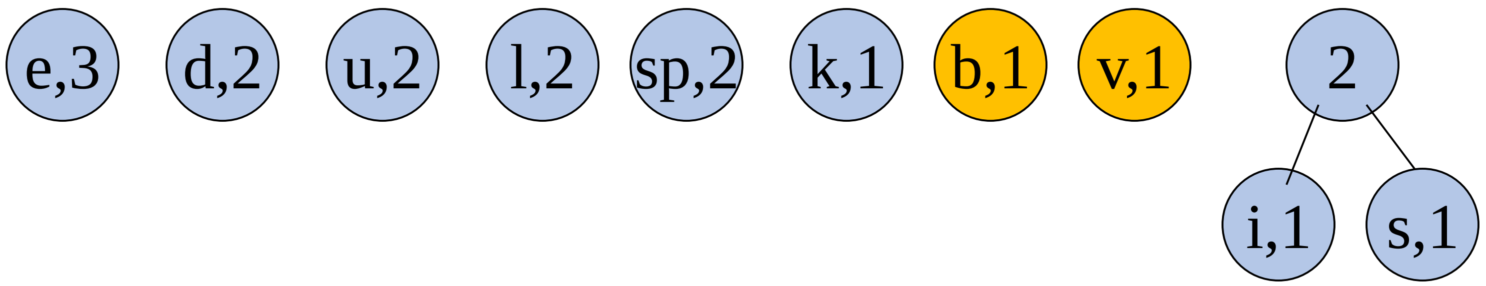
- ทำการเลือกโหนดที่มีความถี่น้อยที่สุดก่อน
- เลือกมาอีกโหนดแล้วจับรวมกัน เก็บผลรวมไว้ที่ root
- ทำซ้ำจนกว่าจะเหลือโหนดเพียงหนึ่งโหนดในเซ็ต

```
H = new Heap()
for each w_i
      T = new Tree(w_i)
      H.Insert(T)
while H.Size() > 1
      T_1 = H.Extract_Min()
      T_2 = H.Extract_Min()
      T_3 = Merge(T_1, T_2)
      H.Insert(T_3)
```

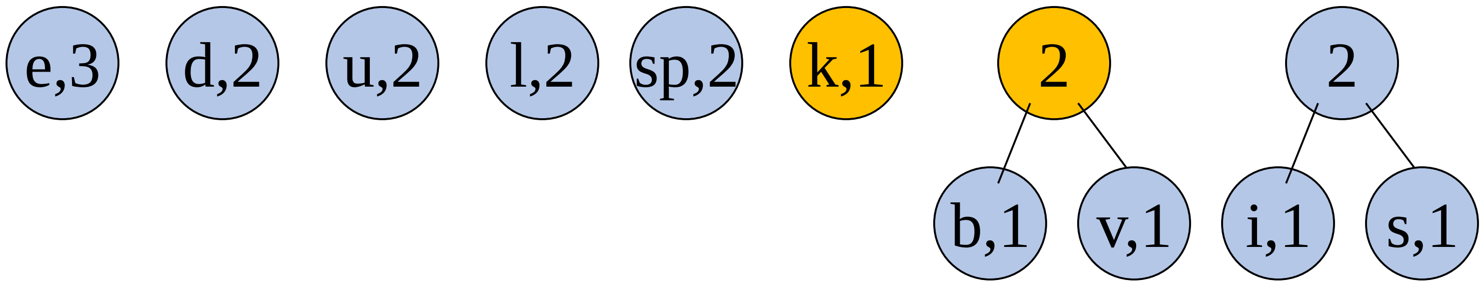
Huffman Coding



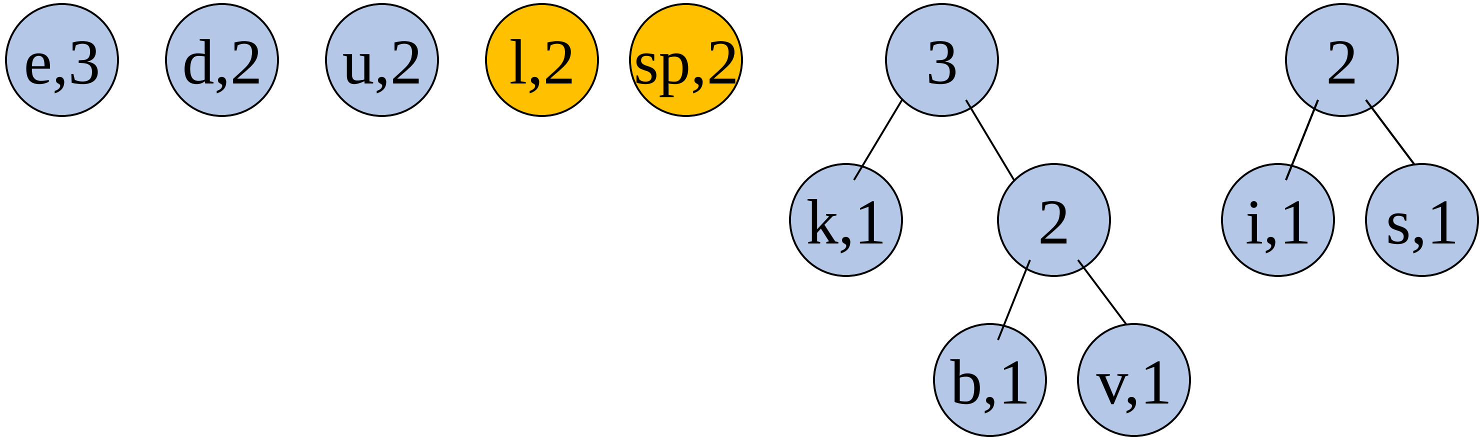
Huffman Coding



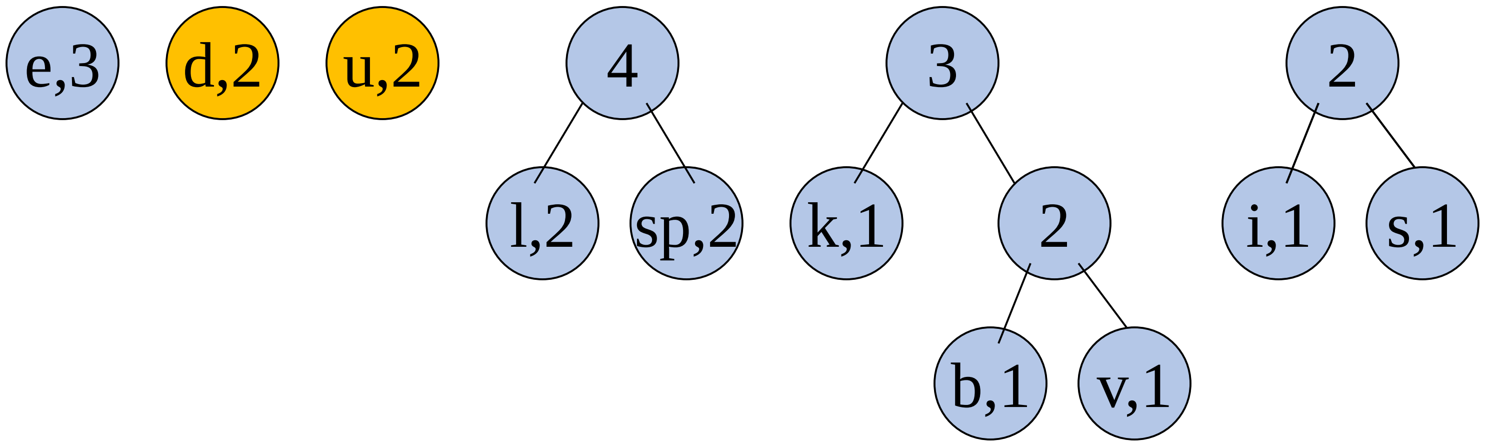
Huffman Coding



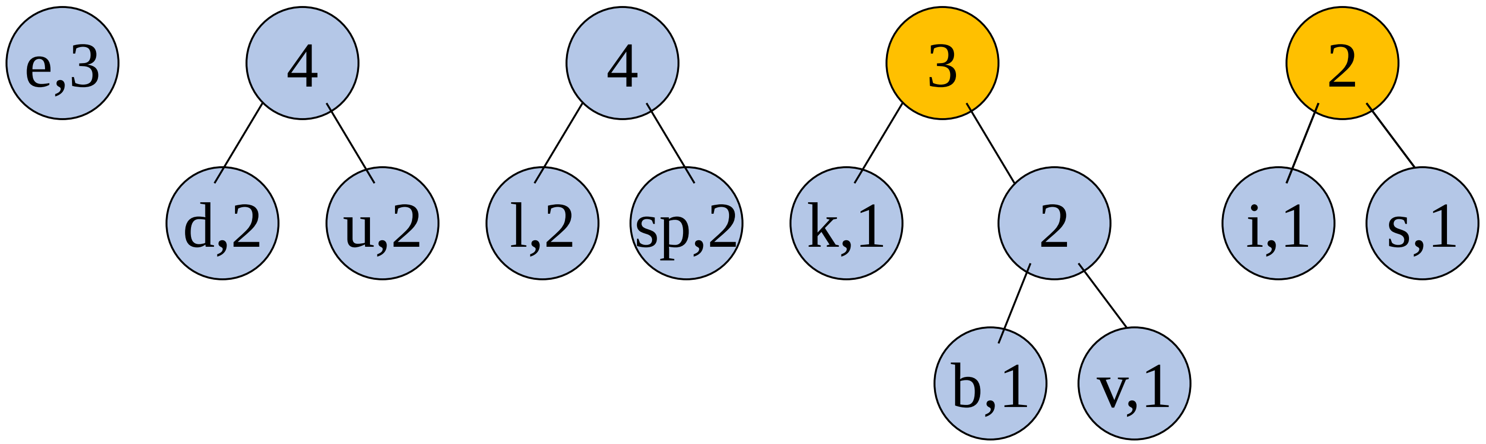
Huffman Coding



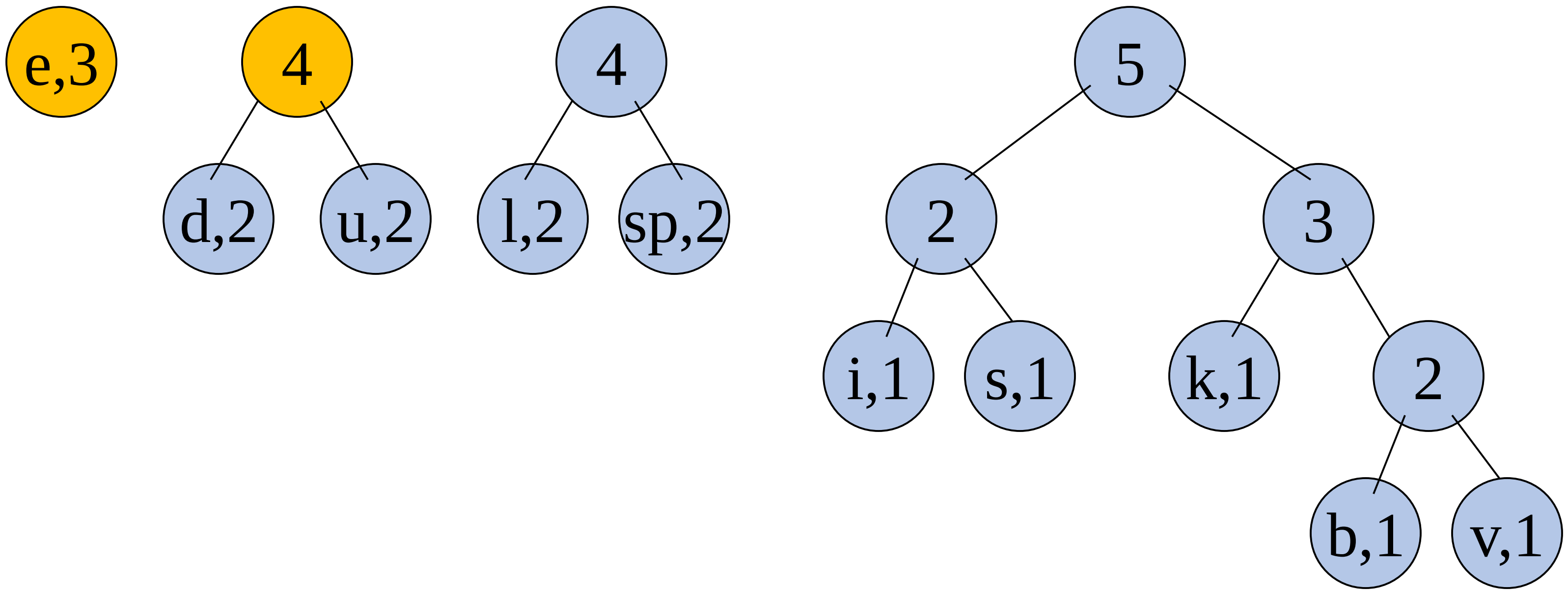
Huffman Coding



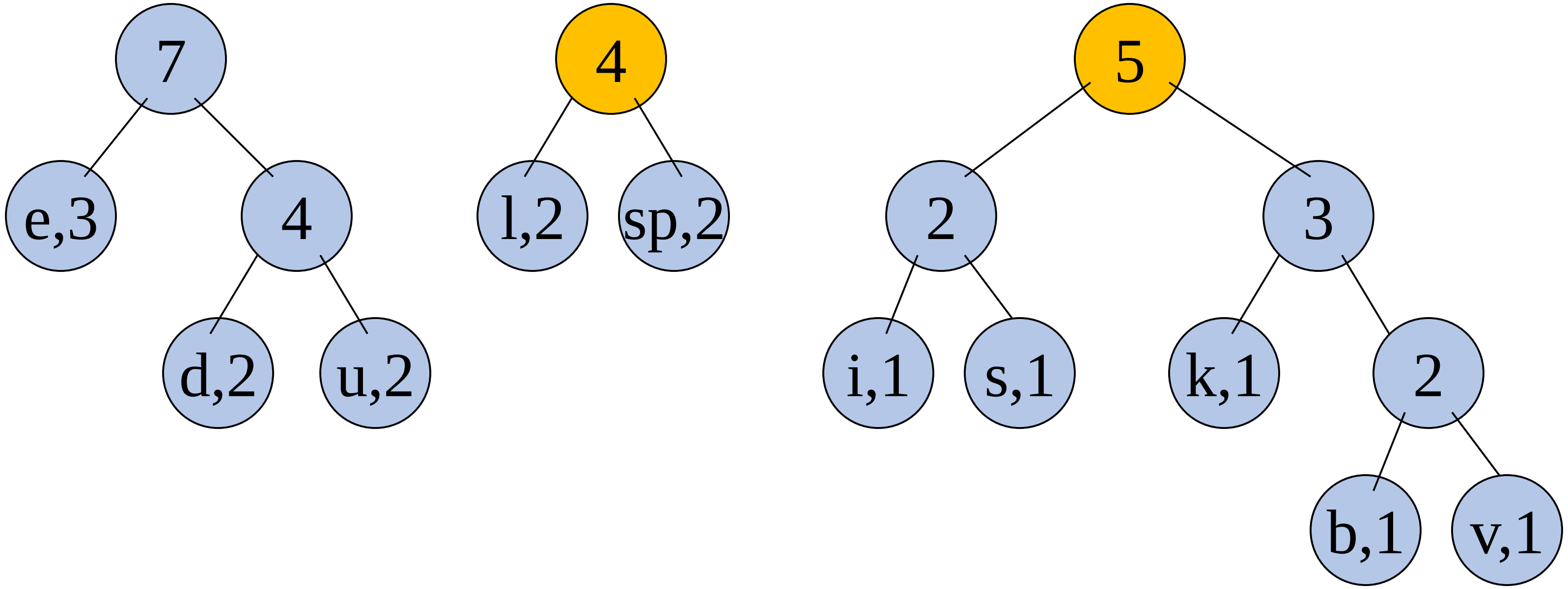
Huffman Coding



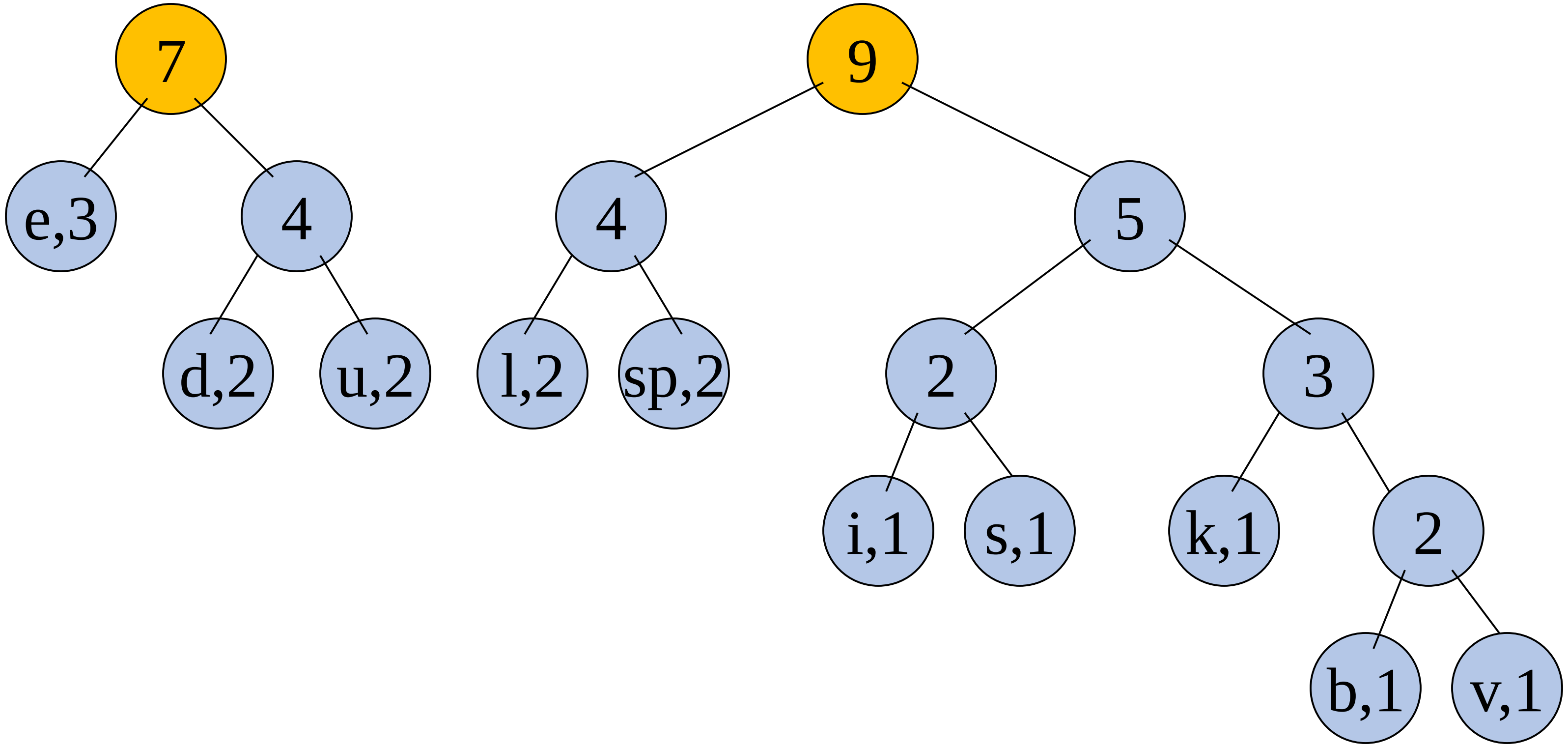
Huffman Coding



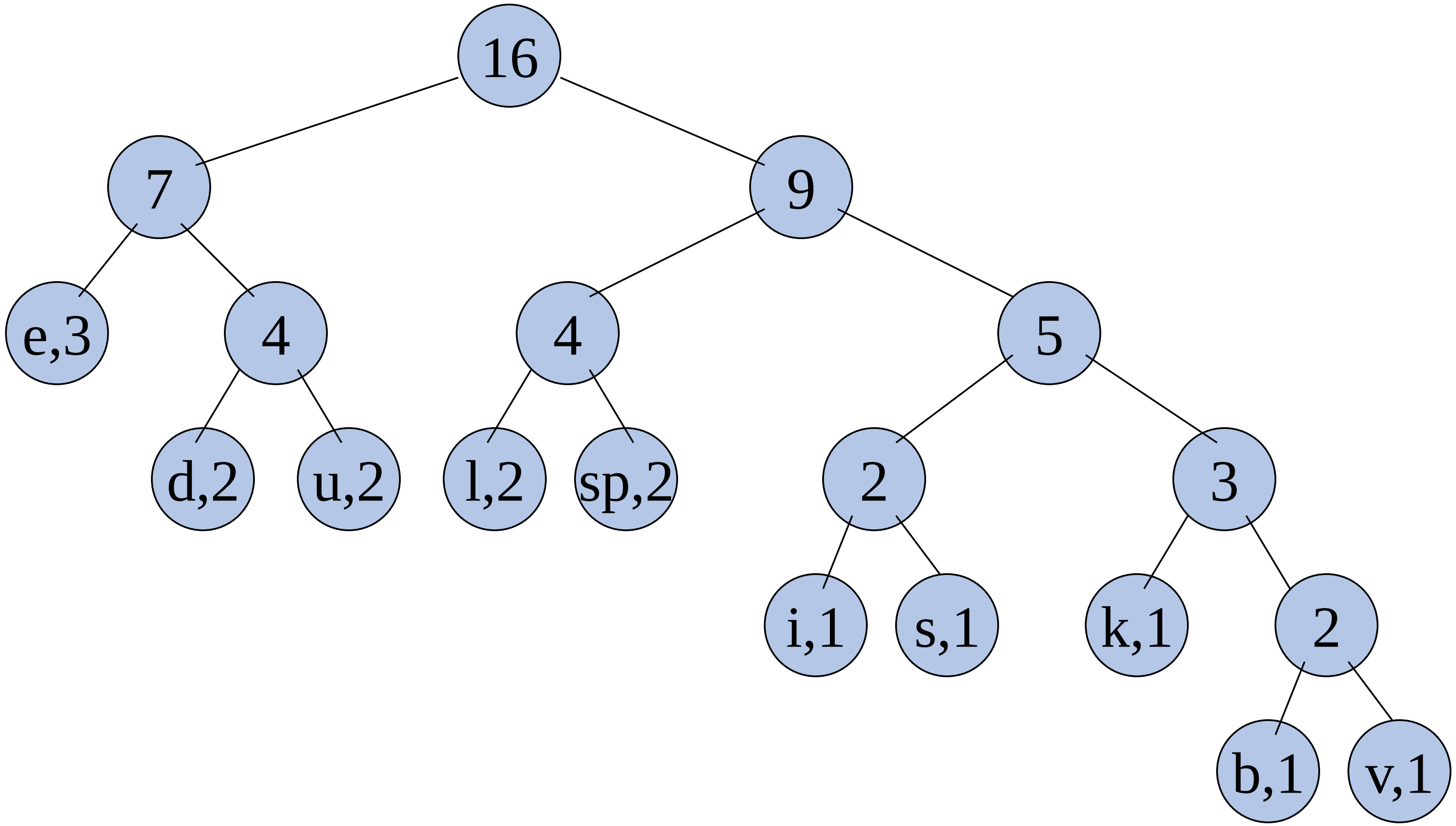
Huffman Coding



Huffman Coding



Huffman Coding

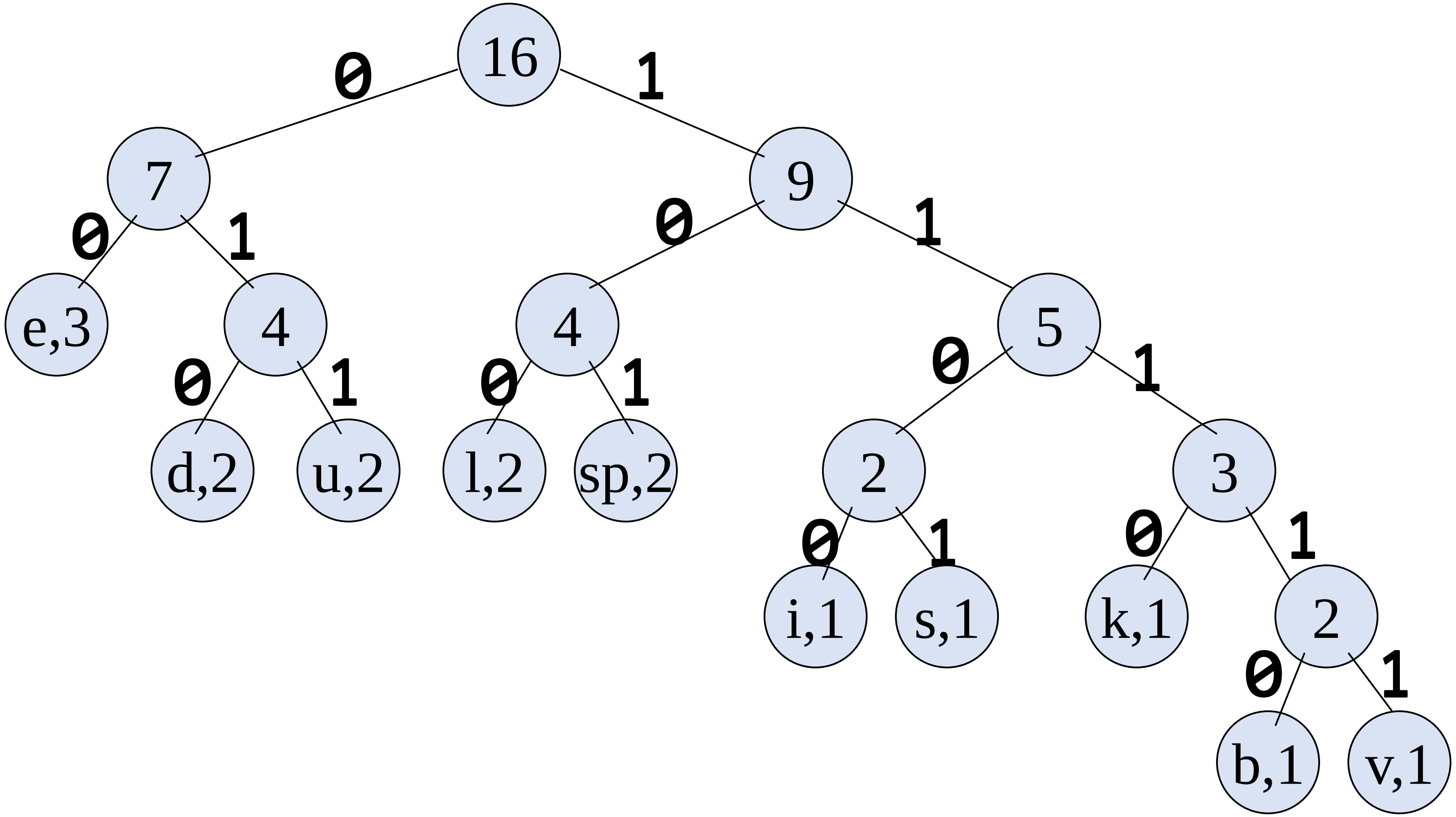


Huffman Coding

- ทำการเข้ารหัสโดยการวาง 0 ไว้ทางซ้ายและ 1 ไว้ทางขวา
- ทำ tree traversal จนครบทุกโหนด
- เวลาถอดรหัสทำจนสุดปลาย leaf แล้วค่อยตัดคำ

Huffman Coding

e	00
d	010
u	011
l	100
sp	101
i	1100
s	1101
k	1110
b	11110
v	11111



Huffman Coding

e	00
d	010
u	011
l	100
sp	101
i	1100
s	1101
k	1110
b	11110
v	11111

- Thus, “duke blue devils” turns into:

010 011 1110 00 101 11110 100 011 00 101 010 00 11111 1100 100 1101

- When grouped into 8-bit bytes:

01001111 10001011 11101000 11001010 10001111 11100100 1101xxxx

- Thus it takes 7 bytes of space compared to 16 characters * 1 byte/char = 16 bytes uncompressed

Huffman Coding

- เวลาถอดรหัสเราค่อยๆ ถอดบิตต่อบิตเริ่มจาก root ถึง leaf

```
Start at the root of the tree
      if a 0 is read,
            head left
      else if a 1 is read,
            head right
When a leaf is reached decode that character
and start over again at the root of the tree
```

- เราจำเป็นต้องเก็บ Huffman table ไว้ใน header ของ file ด้วย

แบบฝึกหัดสร้าง Huffman Tree จากประโยค

“she sells sea shell on the sea shore”

- สร้าง array based tree ก่อนเพิ่มอีก column ใส่ความถี่
- สร้าง heap เก็บ A-Z และ white_space ตาม ความถี่ ทุกครั้งที่รับข้อมูลเข้า ให้ decrease_key
- หลังจบทุกประโยคให้ดึงโหนดออกจาก heap มาสองโหนด union กันแบบซ้ายขวา บวกความถี่แล้วโยนกลับเข้า heap
- วนซ้ำจนกว่าจะไม่มีของเหลือใน heap พอให้ union