Algorithms
演算法

Graphs (2)
Minimum Spanning Tree

Professor Chien-Mo James Li 李建模
Graduate Institute of Electronics Engineering
National Taiwan University

Outline

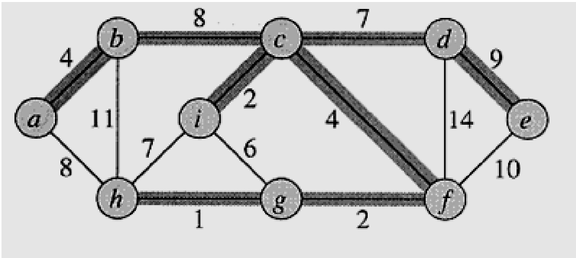
- **Elementary Graph Algorithms, CH22**
- **<u>Minimum Spanning Trees, CH23</u>**
- **Single Source Shortest Paths, CH24**
- **All-pairs Shortest Paths, CH25**
- **Maximum Flow, CH26**

Minimum Spanning Tree (MST)

- Input: Given an connected, undirected graph $G = (V,E)$, and weight function $w(u,v)$ on each edge $(u,v) \in E$
- Output: Find $T \subseteq E$ such that
  - 1. T connects all vertices (T is a spanning tree), and
  - 2. summation of weight is minimum

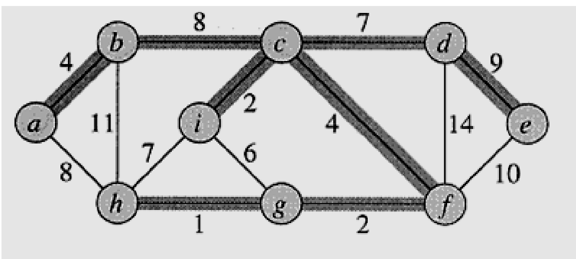
$$w(T) = \sum_{(u,v)\in T} w(u,v)$$

- Example: Fig 23.1
  - edges in T are highlighted
  - minimum weight = 37



MST (2)

- Properties of MST
  - MST has $|V|$ - 1 edges
  - MST has no cycles
  - MST might not be unique
    - Example: (b,c) can be replaced by (a,h)
- Applications of MST
  - construction of networks such as rail way, circuit interconnects



Growing MST

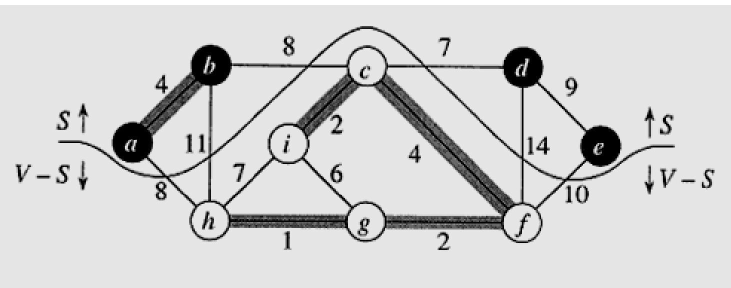
- Grow MST by adding one *safe edge* at a time
  - If *A* is a subset of some MST, an edge (*u, v*) is *safe* for A if and only if $A \cup \{(u,v)\}$ is also a subset of some MST

```
GENERIC-MST(G, w)
A ← Ø
while A does not form a spanning tree
      do find an edge (u, v) that is safe for A
            A ← A ∪ {(u, v)}
return A
```

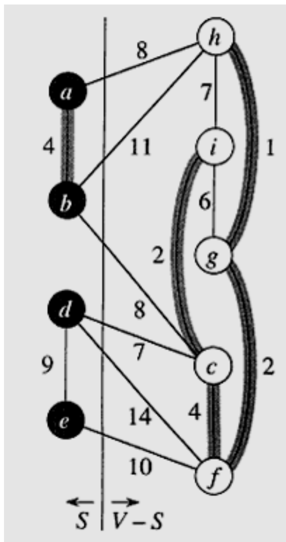
- Loop invariant: Prior to each iteration, *A* is a subset of some MST
  - Initialization: The empty set trivially satisfies the loop invariant.
  - Maintenance: Since we add only safe edges, *A* remains a subset of some MST
  - Termination: All edges added to *A* are in an MST, so when we stop, *A* is a spanning tree that is also an MST

Cut

- Let $S \subset V$ and $A \subseteq E$
- A *cut* (*S*, *V-S*) is a partition of vertices into disjoint sets *V* and *S-V*
  - A cut *respects A* if and only if no edge in *A* crosses the cut
  - (*u*,*v*) *crosses* the cut if *u* is in *S* and *v* is in *V-S*
  - *light edge crossing a cut* is the minimum-weighted edge over all edges crossing the cut
    - not unique
- Figure 23.2
  - light edge is (*c*, *d*)

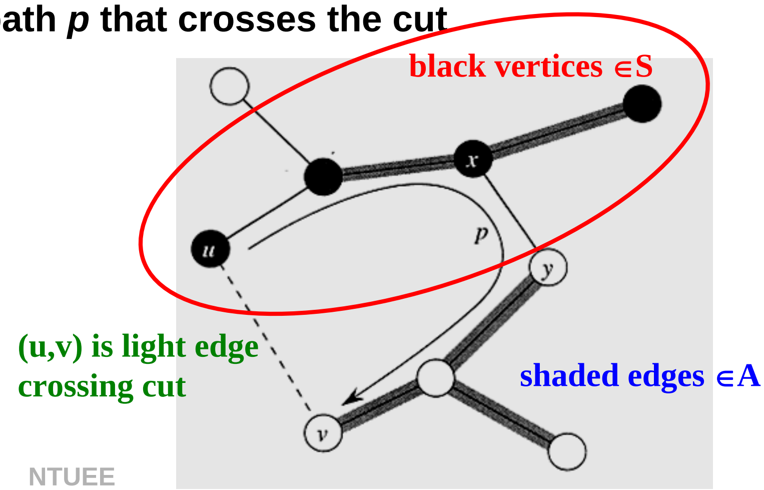


(a)



Light Edge is Safe

- (Theorem 23.1)Let A be a subset of some MST, (S, V-S) be a cut that respects A, and (u,v) be a light edge crossing (S, V-S). Then (u,v) is safe for A
  - Proof: "cut and paste" method
  - Let T be an MST that includes A but does not contain (u,v)
  - Since T is an MST, it contains a unique path p between u and v
    - Path p must cross the cut (S, V-S) at least once.
    - T contains (x,y) on path p that crosses the cut
      - w(x,y)≥ w(u,v)

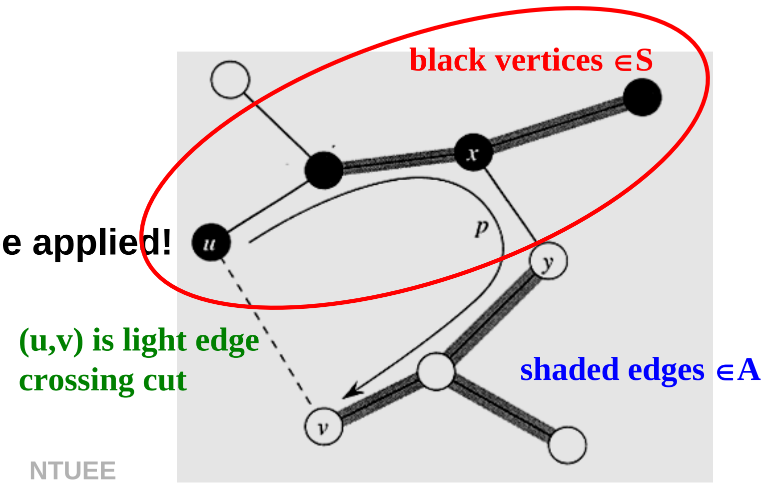


Light Edge is Safe (cont'd)

- construct a different MST T' that contains (u,v)
  - $T' = T - \{(x,y)\} \cup \{(u,v)\}$

$$w(T') = w(T) - w(x,y) + w(u,v)$$
$$\leq w(T)$$

- Since T' is a spanning tree, $w(T') \leq w(T)$, and T is an MST
  - then T' must also be an MST
- (u,v) is safe for A, why?
  - $A \subseteq T$ and $(x,y) \notin A$
  - $A \subseteq T'$, $A \cup \{(u,v)\} \subseteq T'$
  - Since T' is an MST
    - (u,v) is safe for A
- So, greedy algorithm can be applied!



Corollary 23.2

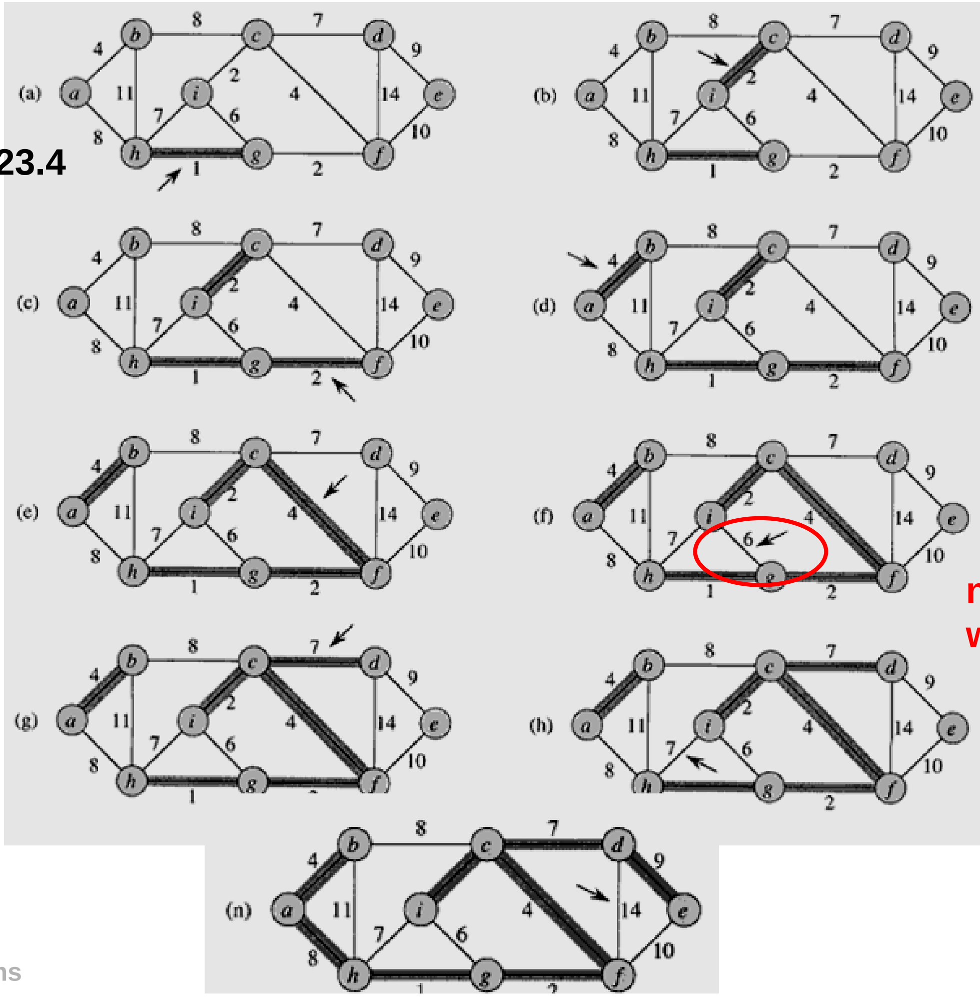
- (Corollary 23.2) If C =(V_C, E_C) is a connected component (tree) in the forest G_A=(V,A). If (u,v) is a light edge connecting C to some other component in G_A, then (u,v) is safe for A
  - Proof:
    - Set S = V_C in the theorem
    - (u,v) is a light edge crossing the cut (V_C, V-V_C)

Kruskal's Algorithm

- Initially, every vertex is a tree
- Repeatedly, add a safe edge to the growing forest
  - by finding an edge of least weight
  - new edge connects two different trees in the forest
- Is this a greedy algorithm?

```
MST-KRUSKAL(G,w)
 A=0
for each vertex v ∈ G.V
    MAKE-SET(v)     // CH21
sort the edges of G.E into nondecreasing order by weight w
for each edge (u,v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u)≠ FIND-SET(v)     //different trees (CH21)
       A=A∪{(u,v)}
       UNION (u,v)
return A
```

- **Fig 23.4**



not chosen
why?

Time Complexity

- **line 2-3: O(*V*)**
- **line 4: O(*E* lg *E*)**
- **line 5-8: O((*V*+*E*) α(*V*)), α is a slow growing function, see CH21**
  - **=O(*E* α(*V*)) , because graph is connected, |*E*| ≥ |*V*|-1**
  - **=O(*E* lg *V*) , because α(*V*) = O(lg *V*)**
  - **=O(*E* lg *E*) , because *E*<|*V*²|**

```
MST-KRUSKAL(G,w)
1  A=0
2  for each vertex v ∈ G.V
3      MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u,v) ∈ G.E, taken in nondecreasing order by weight
6       if FIND-SET(u)≠ FIND-SET(v)
7          A=A∪{(u,v)}
8          UNION (u,v)
9  return A
```

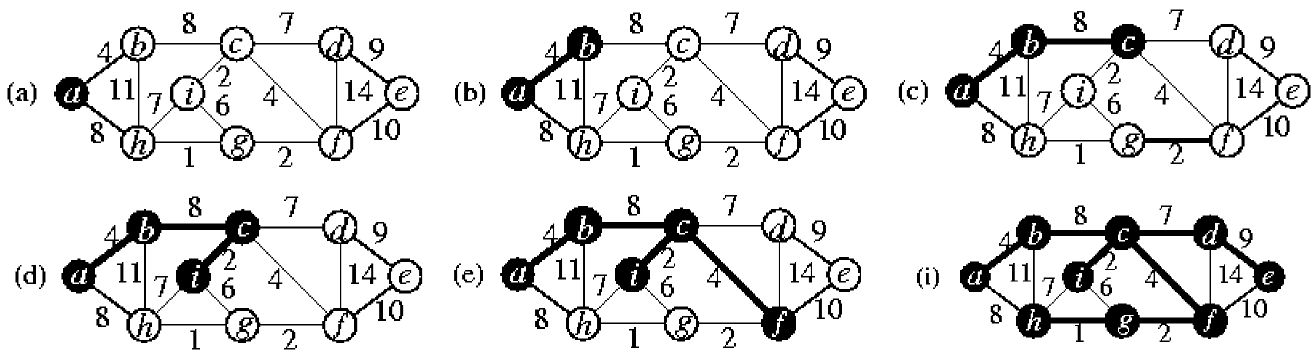
Prim's Algorithm

- *Q*=priority queue for vertices NOT in the tree *A*, sorted by their keys
  - edges in *A* always form a single tree $A = \{(v, v.\pi) : v \in V - \{r\} - Q\}$
  - *v.key*=min weight of edge connecting to vertex *v*
  - *v*.π = parent of vertex *v* in the tree
- At each step, the minimum edge connecting a vertex in *A* to a vertex in *V-A* is added to the tree

```
MST-PRIM(G,w,r)
  for each u ∈ G.V
     u.key = ∞
     u.π = NIL
  r.key = 0
  Q = G.V
  while Q ≠ ∅
     u = EXTRACT-MIN(Q)
        for each v ∈ G.Adj[u]
           if v ∈ Q and w(u,v) < v.key
              v.π = u
              v.key = w(u,v)
```


Example

- Fig 23.5
- (a) *b.key* = 4, *b*.π = *a* ; *h.key* = 8, *h*.π = *a*
  - select *b*
- (b) *c.key* = 8, *c*.π = *b* ; *h.key* = 8, *h*.π = *a*
  - select *c* (select *h* is also fine)
- (c) *d.key* =? , *d*.π =? ; *i.key* =? , *i*.π =? ; *f.key* =? , *f*.π =?
  - select ?
- (d) *h.key* =?, *h*.π =?



Loop Invariant

- **prior to each iteration of while loop**
  - **1.** $A = \{(v, v.\pi) : v \in V - \{r\} - Q\}$
  - **2. vertices in MST are in *V-Q***
  - **3. for all vertices *v* ∈ *Q*, if *v*.π ≠ NIL, then *v.key* < ∞ and *v.key* is the weight of a light edge (*v*, *v*.π) connecting to some vertex already in the MST**

Time Complexity

```
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 ≠ ∅
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)
```

	Binary Heap	**Fibonacci heap**
Line1-5	**O(*V*)**	**O(*V*)**
Line6-7 While	**O(*V* lg*V*)**	**O(*V* lg*V*)**
Line8-11 For	**O(*E* lg*V*)**	**O(*E*)***
Total	**O(*V*lg*V*+*E*lg*V*) =O(*E*lg*V*)**	**O(*E*+*V*lg*V*)**

***Fibonacci heap decrease key in O(1) time**

Mergeable Heaps* not in exam

- CH19, P.506
- In theory, Fibonacci heap is fast to decrease-key (good for MST)
  - but Fibonacci heap needs too much work so not very practical

	Binary heap	Fibonacci Heap
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg n)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$\Theta(1)$
EXTRACT-MIN	$\Theta(\lg n)$	$O(\lg n)$
UNION	$\Theta(n)$	$\Theta(1)$
DECREASE-KEY	$\Theta(\lg n)$	$\Theta(1)$
DELETE	$\Theta(\lg n)$	$O(\lg n)$

Fibonacci Heap

- A collection of rooted trees that are min-heap ordered
  - key of a node is greater than or equal to key of its parent
- *H.min* points to the root of a tree with minimum key
- Fig. 19.3

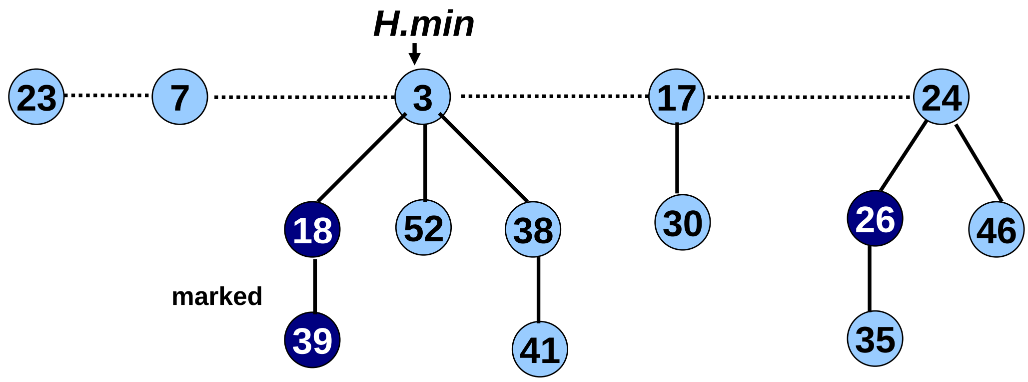
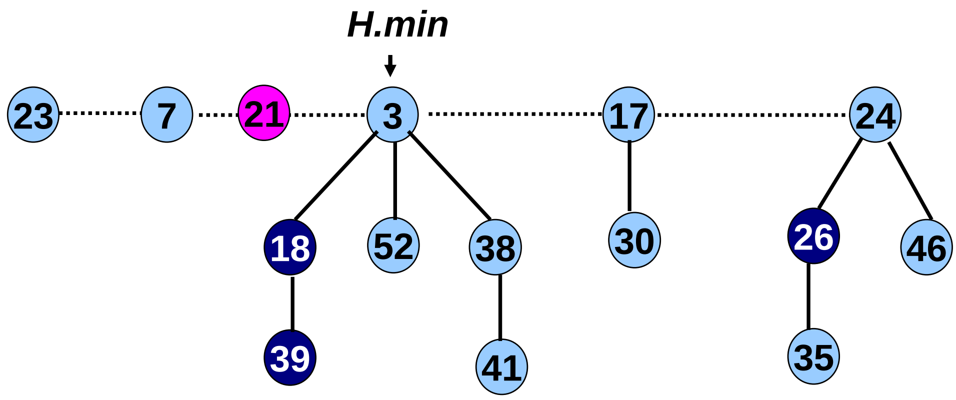


Fig. source : Prof. SC Tsai, NCTU

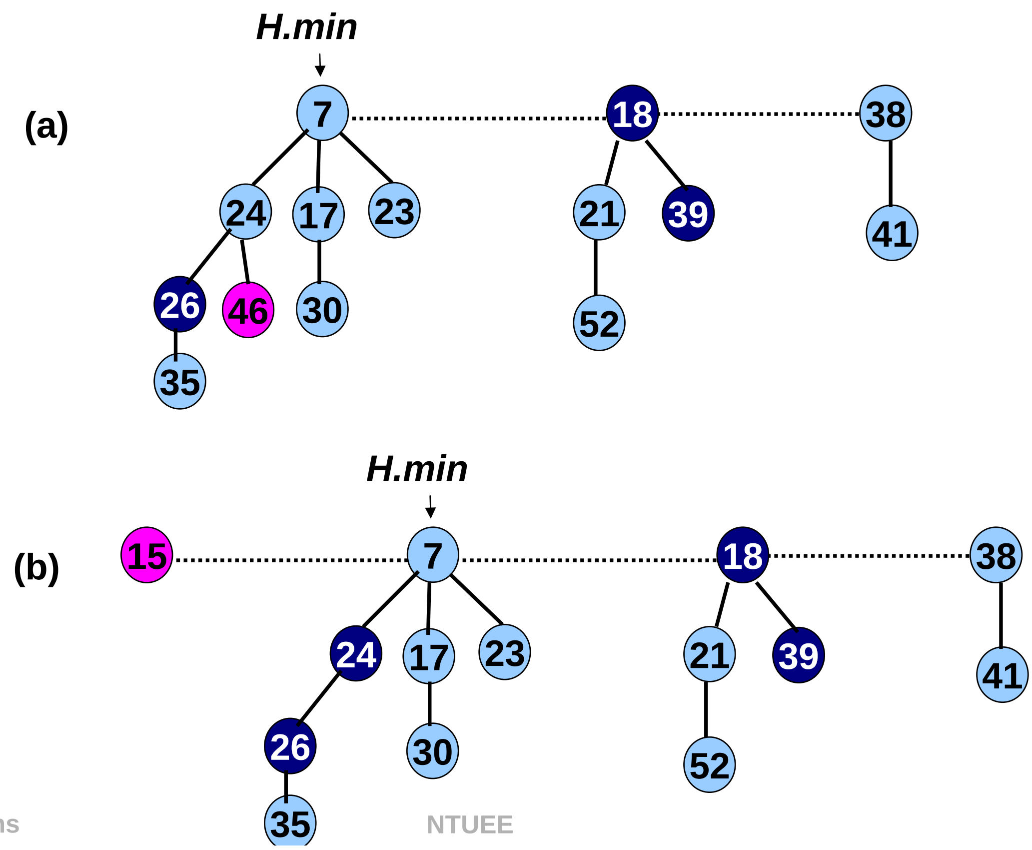
INSERT

- insert 21
- O(1)



DECREASE-KEY

- Fig 19.5 decrease 46→15



Reading

- CH 23