



南京大學

NANJING UNIVERSITY

Introduction to

# *Algorithm Design and Analysis*

## [14] Minimum Spanning Tree



*Yu Huang*

<http://cs.nju.edu.cn/yuhuang>  
Institute of Computer Software  
Nanjing University



# In the last class...

- **Undirected and Symmetric Digraph**
  - DFS skeleton
- **Biconnected Components**
  - Articulation Point
  - Bridge
- **Other undirected graph problems**
  - Orientation for undirected graphs
  - MST based on graph traversal



# 当年我们学贪心算法

**定义 7** 一个图论算法的计算量 $f(v, \varepsilon) = O(P(v, \varepsilon))$ 时, 则称此算法为有效算法或好算法, 其中 $P(v, \varepsilon)$ 是某个多项式,  $v$ 与 $\varepsilon$ 分别是图的顶点数与边数.

**Dijkstra 算法** ( $u, v$  不相邻时,  $w(uv) = \infty$ )

(1) 令 $l(u_0) = 0$ ;  $l(v) = \infty, v \neq u_0$ ;  $S_0 = \{u_0\}, i = 0$ .

(2) 对每一个 $v \in \bar{S}_i$  ( $\bar{S}_i$  指  $S_i$  以外的顶所成之集合), 用 $\min\{l(v), l(u_i) + w(u_i, v)\}$ 代替 $l(v)$ ; 设 $u_{i+1}$ 是使 $l(v)$ 取最小值的 $\bar{S}_i$ 中的顶, 令 $S_{i+1} = S_i \cup \{u_{i+1}\}$ ;

(3) 若 $i = v - 1$ , 止; 若 $i < v - 1$ , 用 $i + 1$ 代替 $i$ , 转(2).

由上述算法知:

(1)  $S_i$ 中各顶标 $l(u)$ 即为 $u_0$ 到 $u$ 的距离. 又因 $v < \infty$ , 故有限步之后,  $V(G)$ 中每一顶都标志了与 $u_0$ 的距离, 从而可以找到各顶到 $u_0$ 的最短轨.

(2) Dijkstra 算法的时间复杂度 $f(v, \varepsilon) = O(v^2)$ , 所以是有效算法.



# Greedy Strategy

- **Optimization Problem**
- **Greedy Strategy**
- **MST Problem**
  - Prim's Algorithm
  - Kruskal's Algorithm
- **Single-Source Shortest Path Problem**
  - Dijkstra's Algorithm



# Greedy Strategy for Optimization Problems

- **Coin change Problem**
  - [candidates] A finite set of coins, of 1, 5, 10 and 25 units, with enough number for each value
  - [constraints] Pay an exact amount by a selected set of coins
  - [optimization] a smallest possible number of coins in the selected set
- **Solution by greedy strategy**
  - For each selection, choose the highest-valued coin as possible.



# Greedy Fails Sometimes

We have to pay 15 in total

- If the available types of coins are  $\{1,5,12\}$ 
  - The greedy choice is  $\{12,1,1,1\}$
  - But the smallest set of coins is  $\{5,5,5\}$
- If the available types of coins are  $\{1,5,10,25\}$ 
  - The greedy choice is always correct



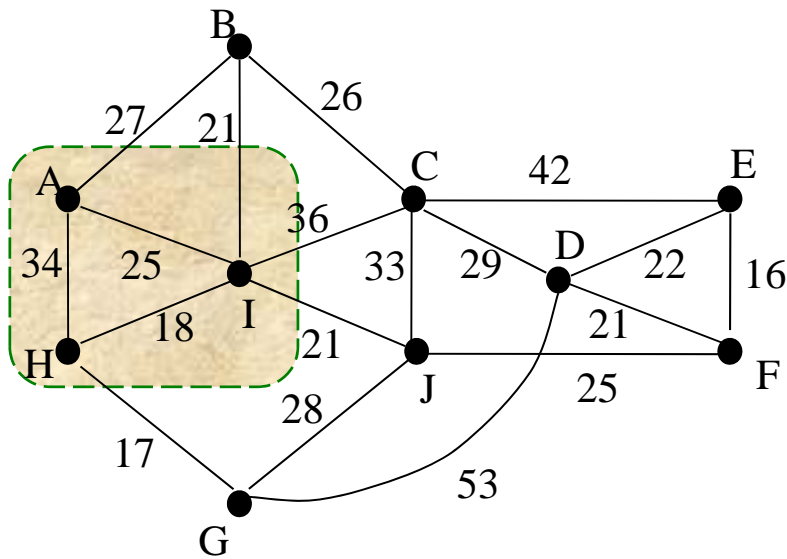
# Greedy Strategy

- Expanding the partial solution **step by step**.
- In each step, a selection is made from a set of candidates. The choice made **must** be:
  - [Feasible] it has to satisfy the problem's constraints
  - [Locally optimal] it has to be the best local choice among all feasible choices on the step
  - [Irrevocable] the choice cannot be revoked in subsequent steps

```
set greedy(set candidate)
  set S=∅;
  while not solution(S) and candidate≠∅
    select locally optimizing x from candidate;
    candidate=candidate-{x};
    if feasible(x) then S=S∪{x};
  if solution(S) then return S
  else return ("no solution")
```



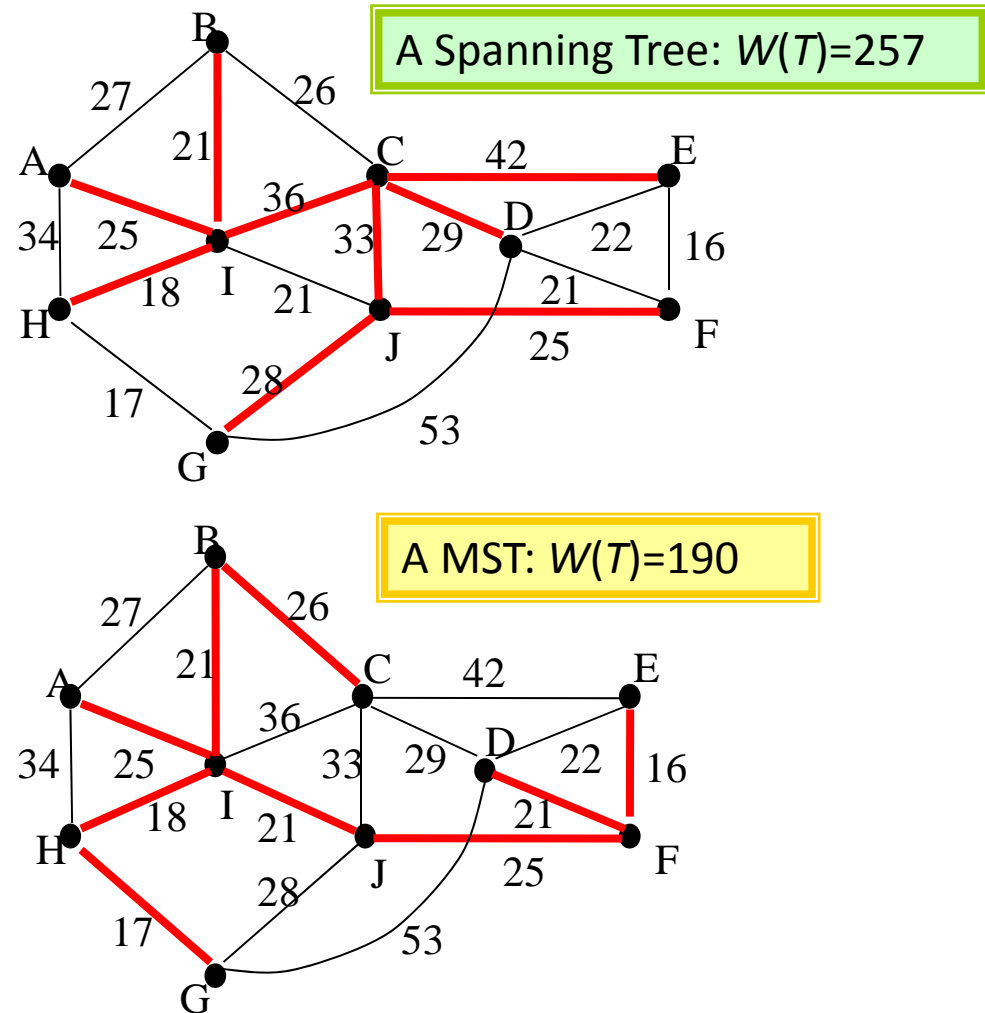
# Weighted Graph and MST



A weighted graph

The nearest neighbor of vertex *I* is *H*

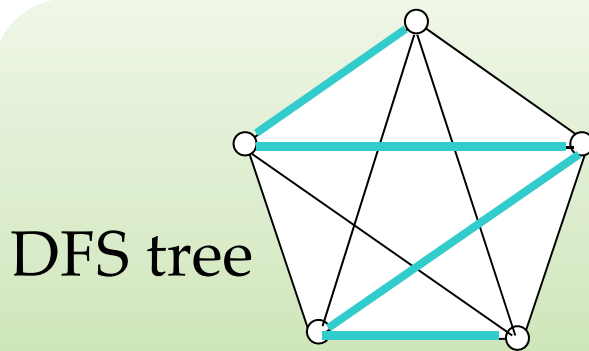
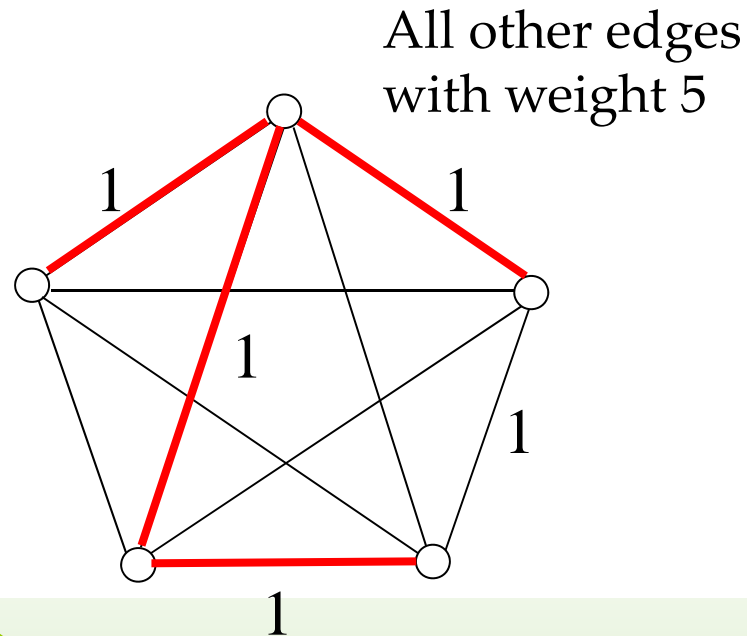
The nearest neighbor of shaded  
subset of vertex is *G*



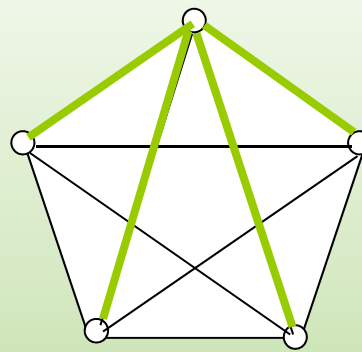


# Graph Traversal and MST

There are cases that graph traversal tree **cannot** be minimum spanning tree, with the vertices explored in any order.



DFS tree



BFS tree

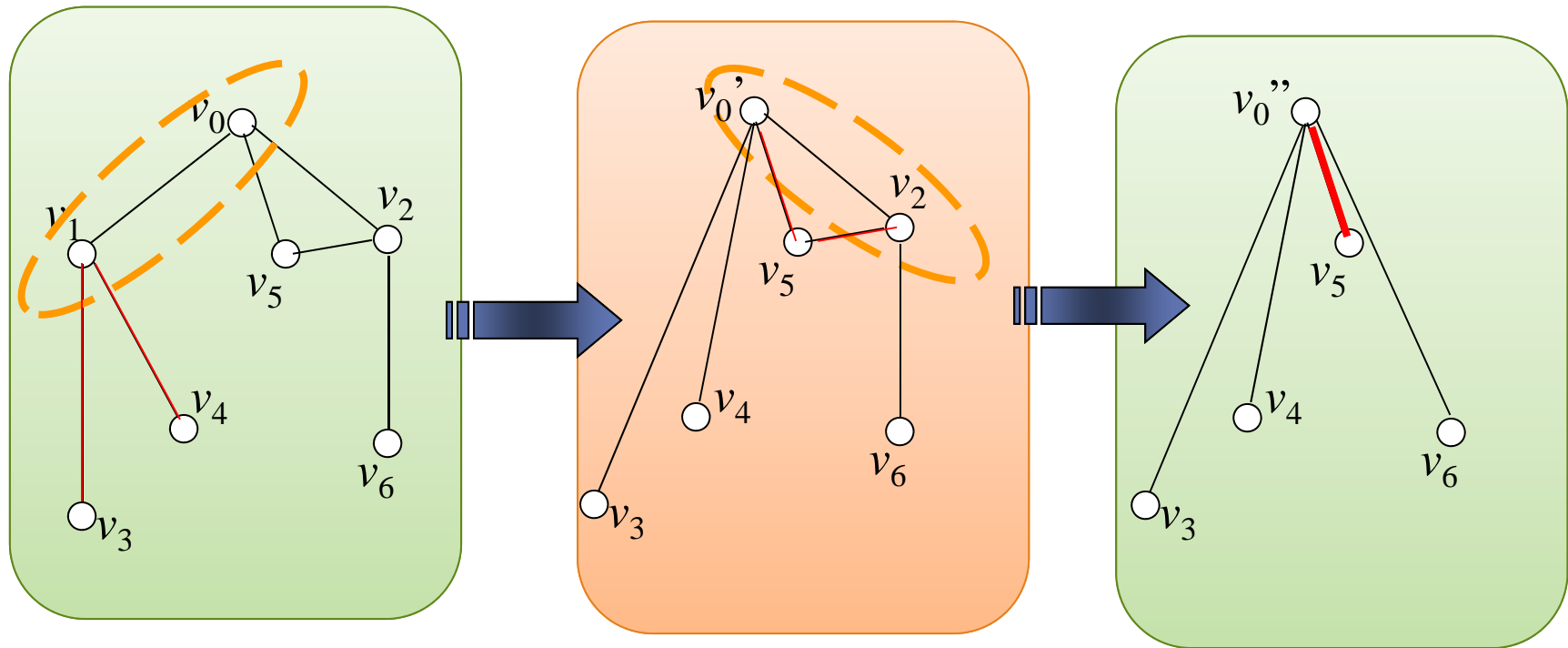
in any ordering of vertex

# Greedy Algorithms for MST

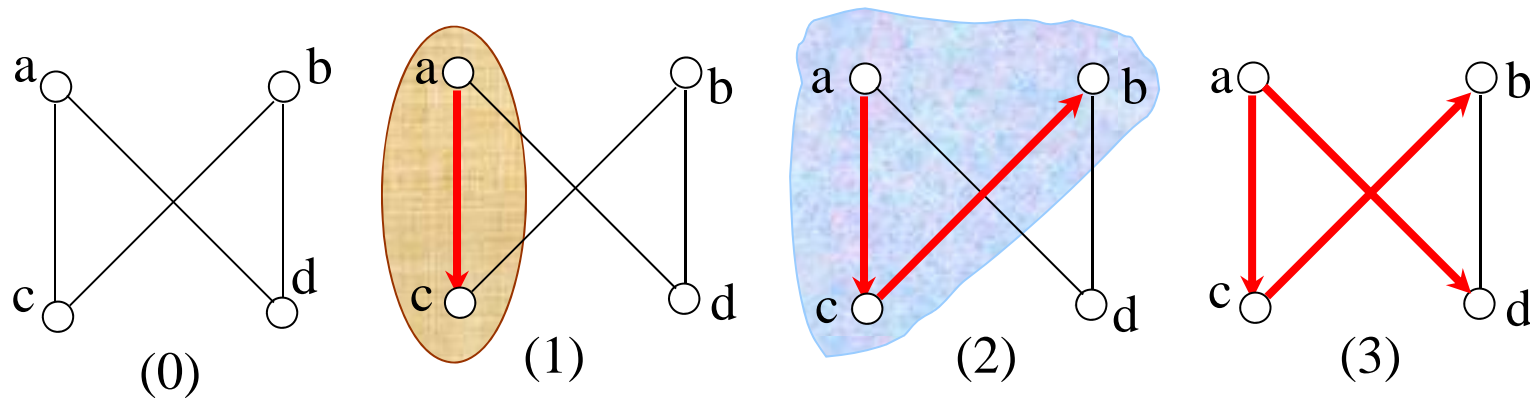
- **Prim's algorithm:**
  - Difficult selecting: “best local optimization means **no cycle and small weight under limitation**.”
  - Easy checking: doing nothing
- **Kruskal's algorithm:**
  - Easy selecting: smallest in primitive meaning
  - Difficult checking: **no cycle**



# Merging Two Vertices

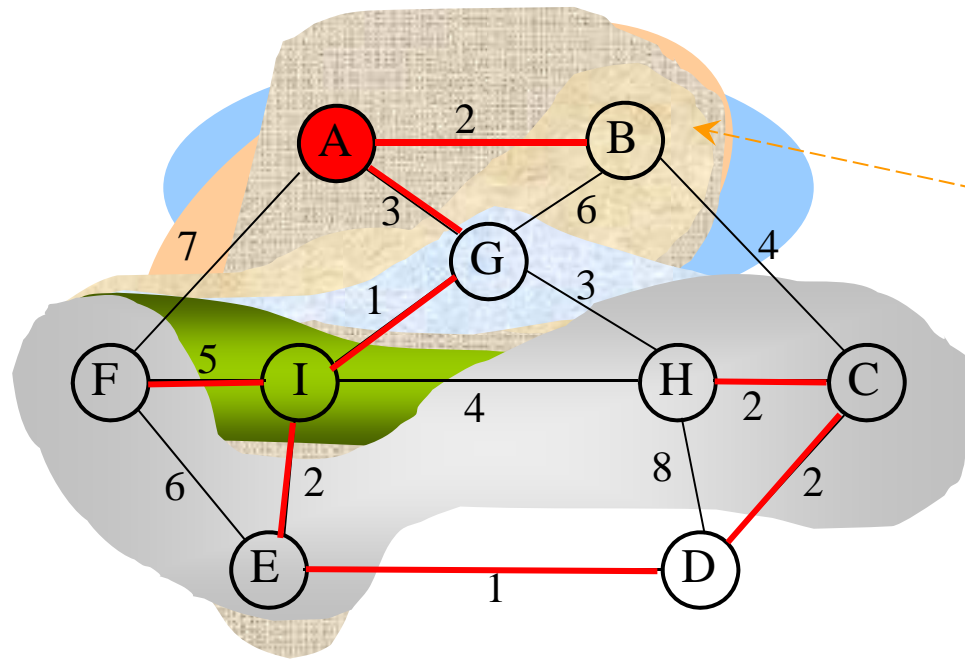


# Constructing a Spanning Tree by Merging Vertices



0. Let a be the starting vertex, selecting edges one by one in original graph
  1. Merging a and c into  $a'(\{a,c\})$ , selecting (a,c)
  2. Merging  $a'$  and b into  $a''(\{a,c,b\})$ , selecting (c,b)
  3. Merging  $a''$  and d into  $a'''(\{a,c,b,d\})$ , selecting (a,d) or (d,b)
- Ending, as only one vertex left

# Prim's Algorithm

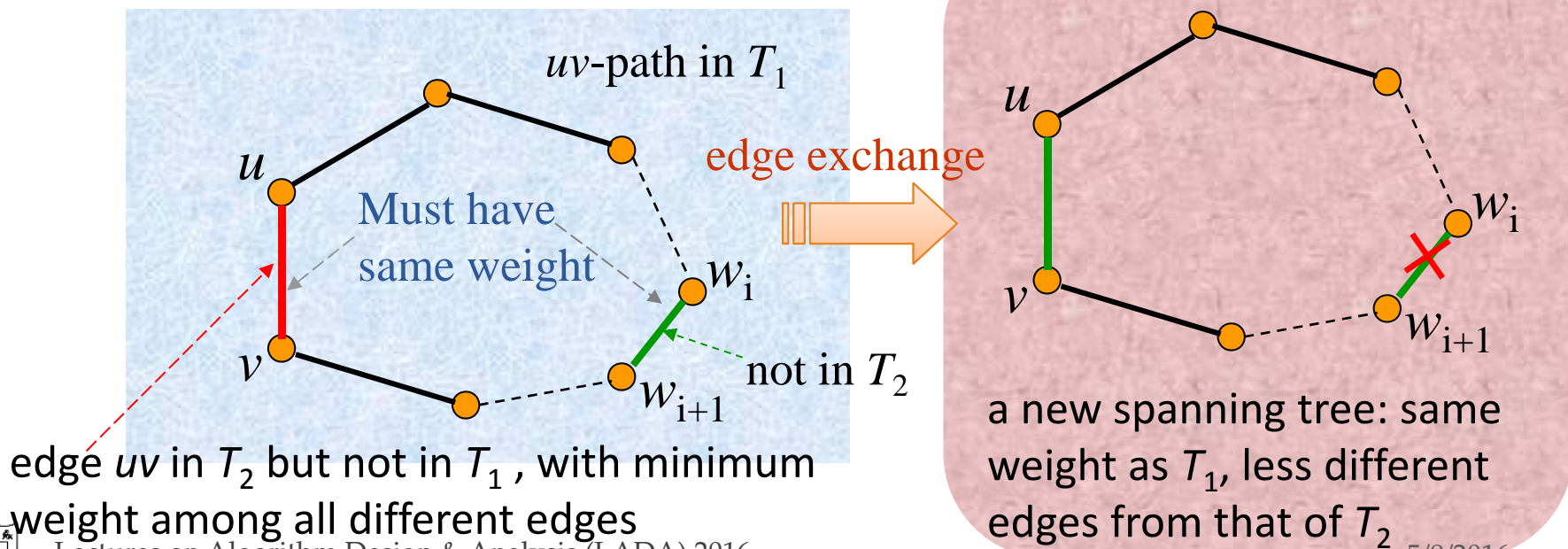


Greedy strategy:  
For each set of fringe vertex,  
select the edge with the  
minimal weight, that is,  
local optimal.

edges included in the MST

# Minimum Spanning Tree Property

- A spanning tree  $T$  of a connected, weighted graph has MST property if and only if for any non-tree edge  $uv$ ,  $T \cup \{uv\}$  contain a cycle in which  $uv$  is **one of** the maximum-weight edge.
- All the spanning trees having MST property have the same weight.



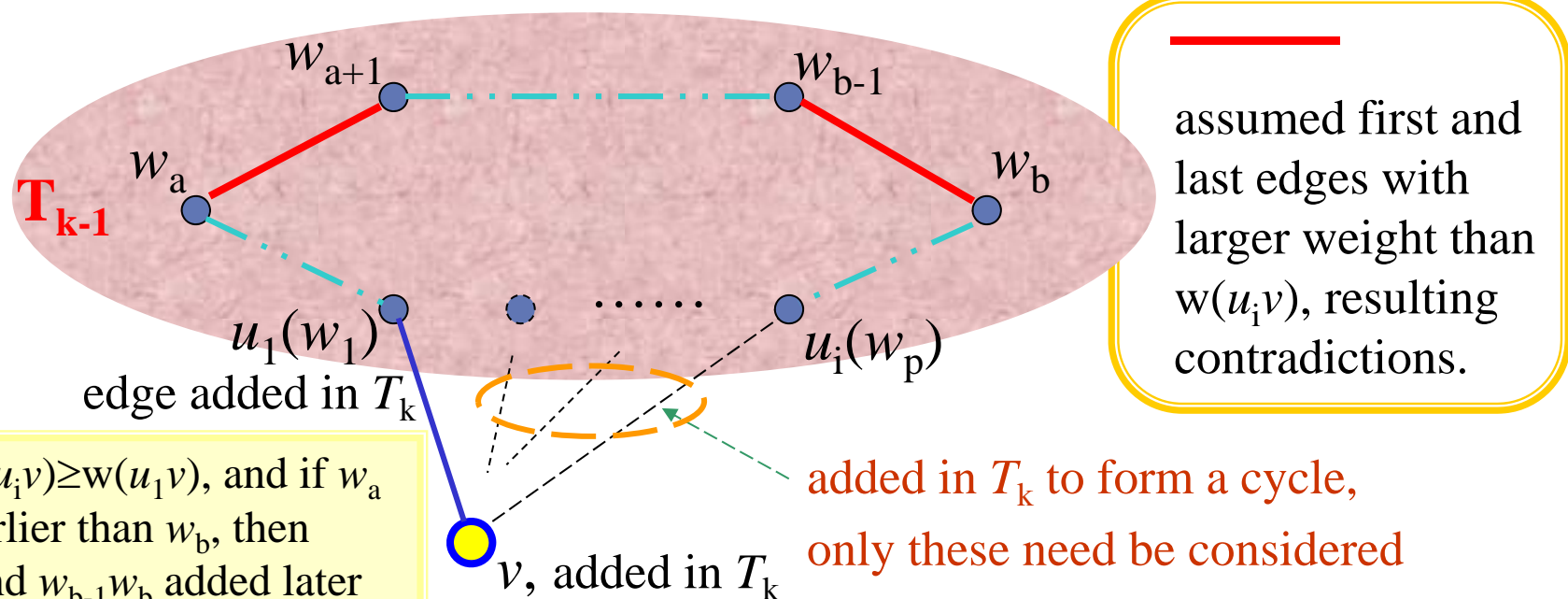
# MST Property and Minimum Spanning Tree

- In a connected, weighted graph  $G=(V,E,W)$ , a tree  $T$  is a minimum spanning tree if and only if  $T$  has the MST property.
- **Proof**
  - $\Rightarrow$  For a minimum spanning tree  $T$ , if it doesn't have MST property. So, there is a non-tree edge  $uv$ , and  $T \cup \{uv\}$  contains an edge  $xy$  with weight larger than that of  $uv$ . Substituting  $uv$  for  $xy$  results in a spanning tree with less weight than  $T$ . Contradiction.
  - $\Leftarrow$  As claimed above, any minimum spanning tree has the MST property. Since  $T$  has MST property, it has the same weight as any minimum spanning tree, i.e.  $T$  is a minimum spanning tree as well.



# Correctness of Prim's Algorithm

- Let  $T_k$  be the tree constructed after the  $k^{\text{th}}$  step of Prim's algorithm is executed. Then  $T_k$  has the MST property in  $G_k$ , the subgraph of  $G$  induced by vertices of  $T_k$ .



Note:  $w(u_1v) \geq w(u_1w_a)$ , and if  $w_a$  added earlier than  $w_b$ , then  $w_a w_{a+1}$  and  $w_{b-1} w_b$  added later than any edges in  $u_1 w_a$ -path, and  $v$  as well.



# Key Issue in Implementation

- **Maintaining the set of fringe vertices**
  - Create the set and update it after each vertex is “selected” (*deleting* the vertex having been selected and *inserting* new fringe vertices)
  - Easy to decide the vertex with “highest priority”
  - Changing the priority of the vertices (*decreasing key*).
- **The choice: priority queue**



# Implementation

## Main Procedure

primMST( $G, n$ )

Initialize the priority queue  $pq$  as empty;

Select vertex  $s$  to start the tree;

Set its candidate edge to  $(-1, s, 0)$ ;

**insert**( $pq, s, 0$ );

**while** ( $pq$  is not empty)

$v = \text{getMin}(pq)$ ;  $\text{deleteMin}(pq)$ ;

add the candidate edge of  $v$  to the tree;

**updateFringe**( $pq, G, v$ );

**return**

$\text{getMin}(pq)$  always be the vertex with the smallest key in the fringe set.

ADT operation executions:

**insert**, **getMin**, **deleteMin**:  $n$  times

**decreaseKey**:  $m$  times

## Updating the Queue

**updateFringe**( $pq, G, v$ )

For all vertices  $w$  adjacent to  $v$  *// 2m loops*

$\text{newWgt} = w(v, w)$ ;

**if**  $w$ .status is unseen **then**

Set its candidate edge to  $(v, w, \text{newWgt})$ ;

**insert**( $pq, w, \text{newWgt}$ )

**else**

**if**  $\text{newWgt} < \text{getPriority}(pq, w)$

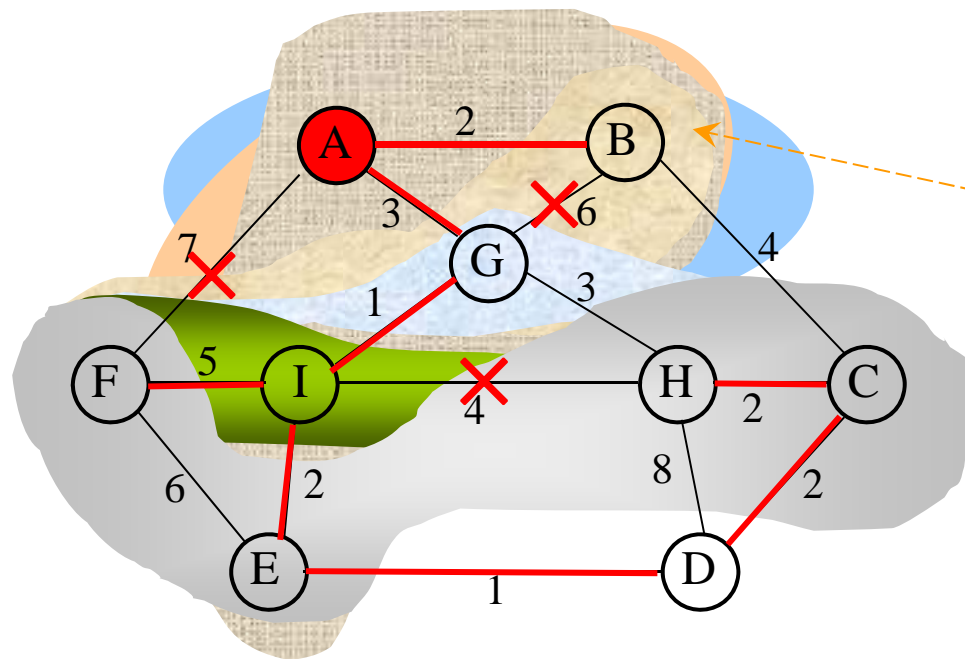
Revise its candidate edge to  $(v, w, \text{newWgt})$ ;

**decreaseKey**( $pq, w, \text{newWgt}$ )

**return**



# Prim's Algorithm



Greedy strategy:  
For each set of fringe vertex,  
select the edge with the  
minimal weight, that is,  
local optimal.

# Complexity

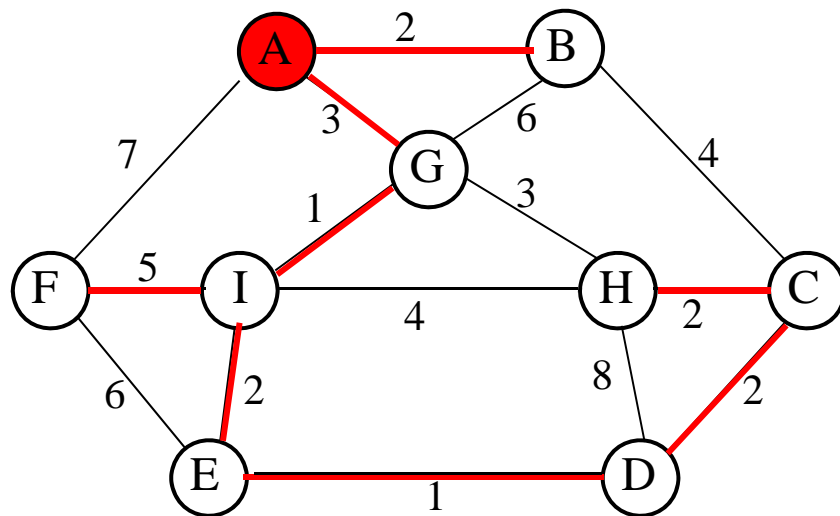
- Operations on ADT priority queue: (for a graph with  $n$  vertices and  $m$  edges)
  - insert:  $n$  ;      getMin:  $n$  ;      deleteMin:  $n$  ;
  - decreaseKey:  $m$  (appears in  $2m$  loops, but execute at most  $m$ )
- So,

$$T(n,m) = O(nT(\text{getMin}) + nT(\text{deleteMin} + \text{insert}) + mT(\text{decreaseKey}))$$

- Implementing priority queue using array, we can get  $\Theta(n^2 + m)$



# Kruskal's Algorithm



edges included in the MST

Also Greedy strategy:  
From the set of edges not yet included in the partially built MST, select the edge with the minimal weight, that is, local optimal, in another sense.

# Key Issue in Implementation

- How to know an insertion of edge will result in a cycle *efficiently*?
- For correctness: the two endpoints of the selected edge *can not* be in the same connected components.
- For the efficiency: connected components are implemented as dynamic equivalence classes using union-find.



# Kruskal's Algorithm: the Procedure

- `kruskalMST(G,n,F)` //outline
- `int count;`
- Build a minimizing priority queue, `pq`, of edges of `G`, prioritized by weight.
- Initialize a Union-Find structure, `sets`, in which each vertex of `G` is in its own set.
- 
- `F =  $\phi$ ;`
- `while (isEmpty(pq) == false)`
- `vwEdge = getMin(pq);`
- `deleteMin(pq);`
- `int vSet = find(sets, vwEdge.from);`
- `int wSet = find(sets, vwEdge.to);`
- `if (vSet  $\neq$  wSet)`
- `Add vwEdge to F;`
- `union(sets, vSet, wSet)`
- `return`

Simply sorting, the cost will be  $\Theta(m \log m)$



# Prim vs. Kruskal

- **Lower bound for MST**
  - For a correct MST, each edge in the graph should be examined at least once.
  - So, the lower bound is  $\Omega(m)$
- **$\Theta(n^2+m)$  and  $\Theta(m\log m)$ , which is better?**
  - Generally speaking, depends on the density of edge of the graph.





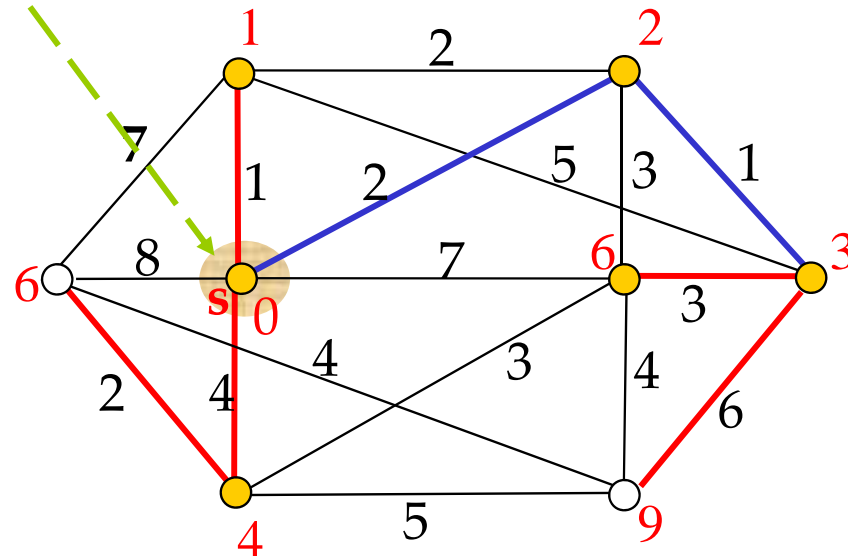
# Single Source Shortest Paths

## The single source

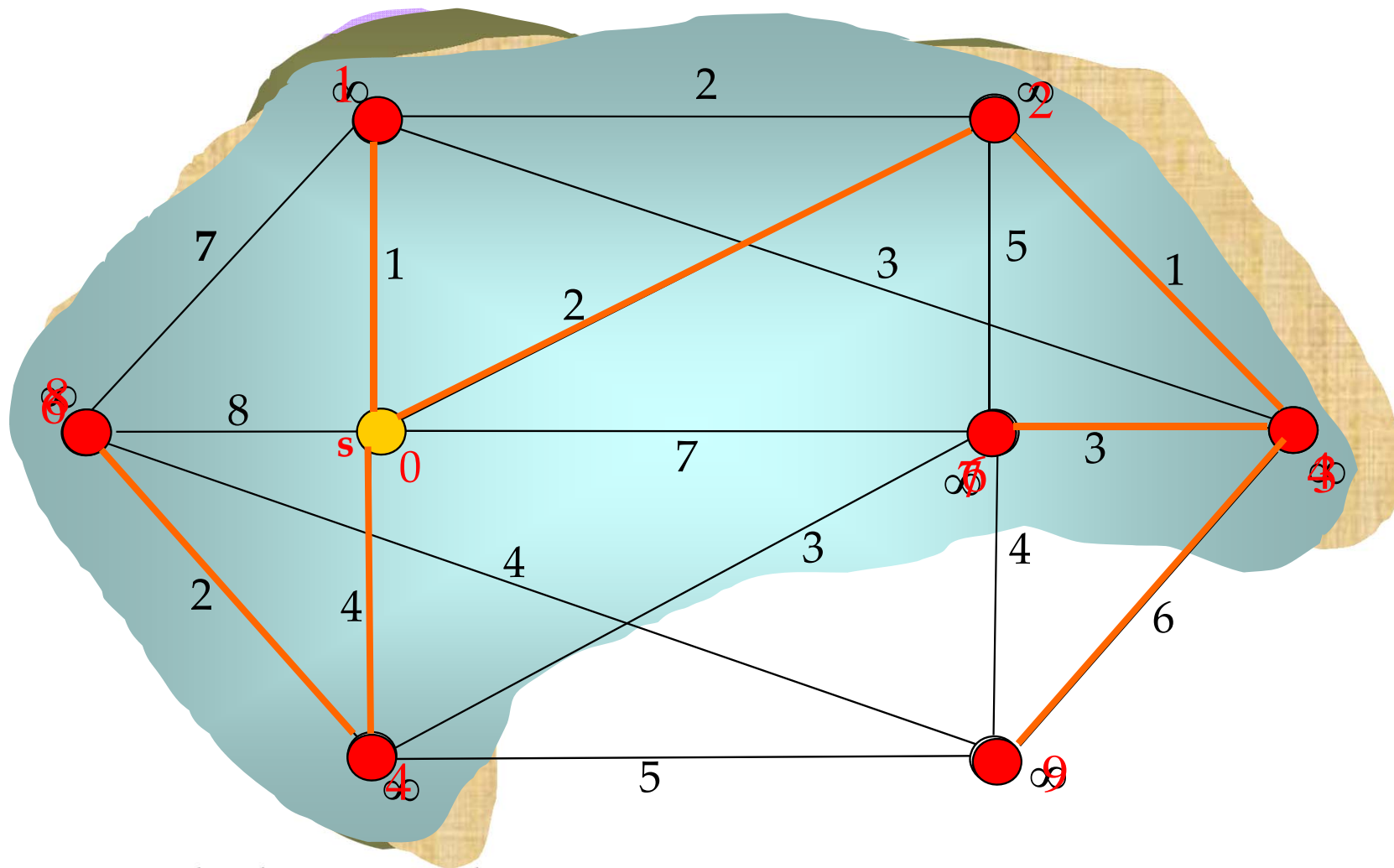
**Red labels** on each vertex is the length of the shortest path from  $s$  to the vertex.

Note:

The shortest  $[0, 3]$ -path  
doesn't contain the shortest  
edge leaving  $s$ , the edge  $[0,1]$



# Dijkstra's Algorithm



*Thank you!*

*Q & A*

*Yu Huang*

<http://cs.nju.edu.cn/yuhuang>

