湖南大學
HUNAN UNIVERSITY

第十一章 最小生成树

湖南大学信息科学与工程学院

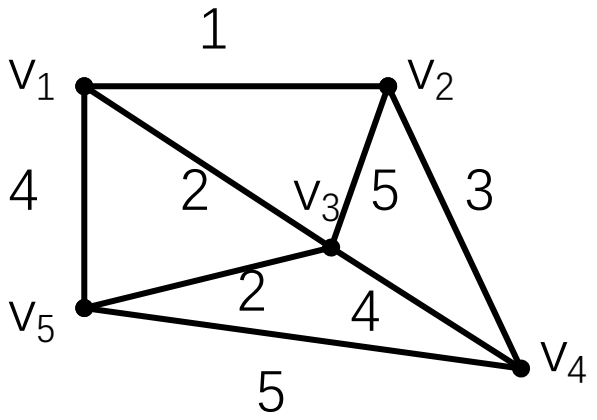
湖南大学 HUNAN UNIVERSITY

目录

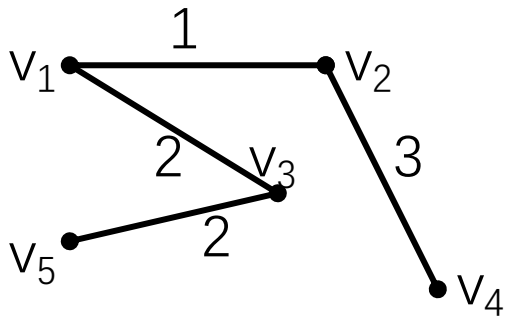
最小生成树的生成

最小连接问题：

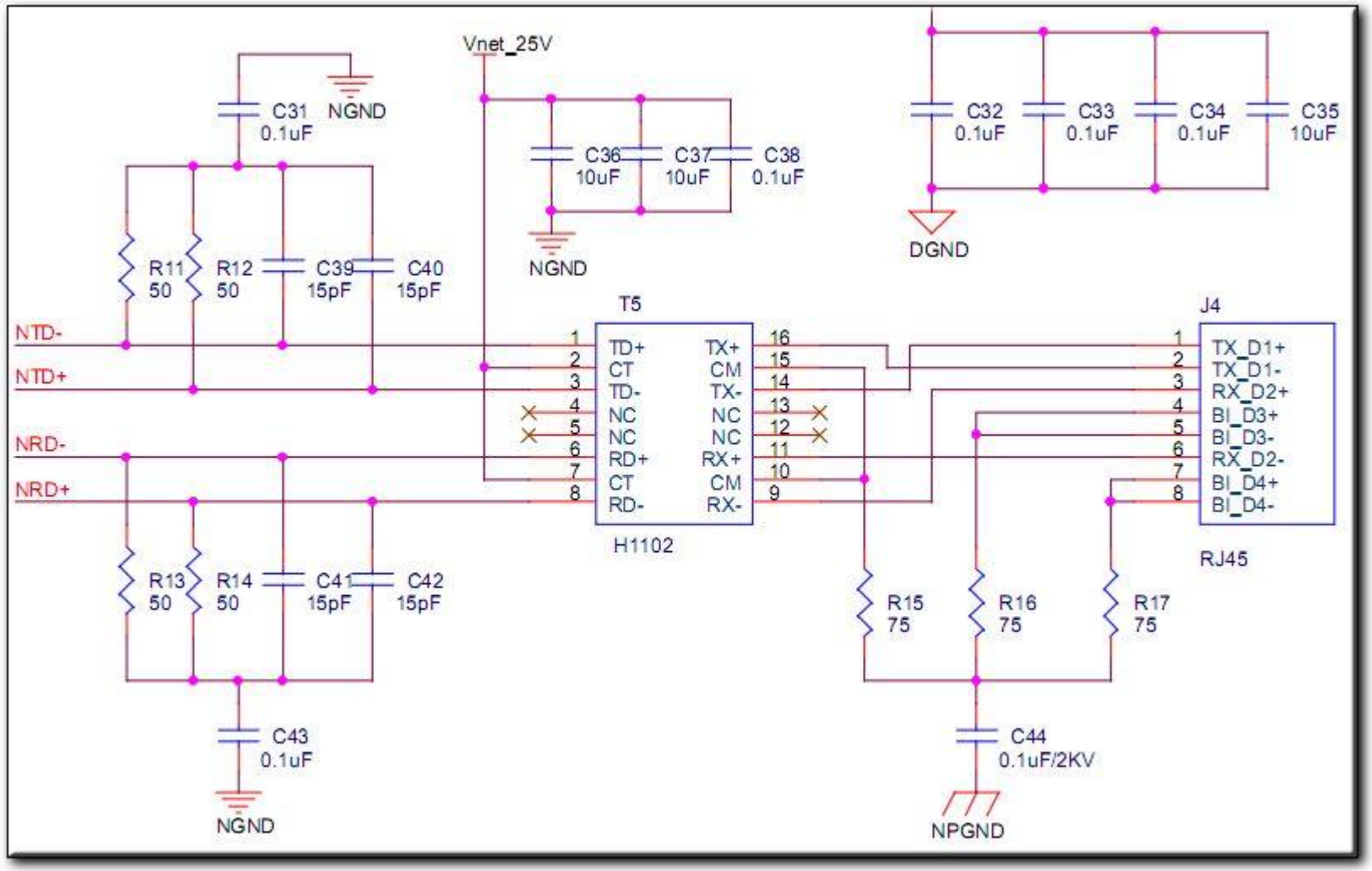
交通网络中，常常关注能把所有站点连接起来的生成树，使得该生成树各边权值之和为最小。例如：



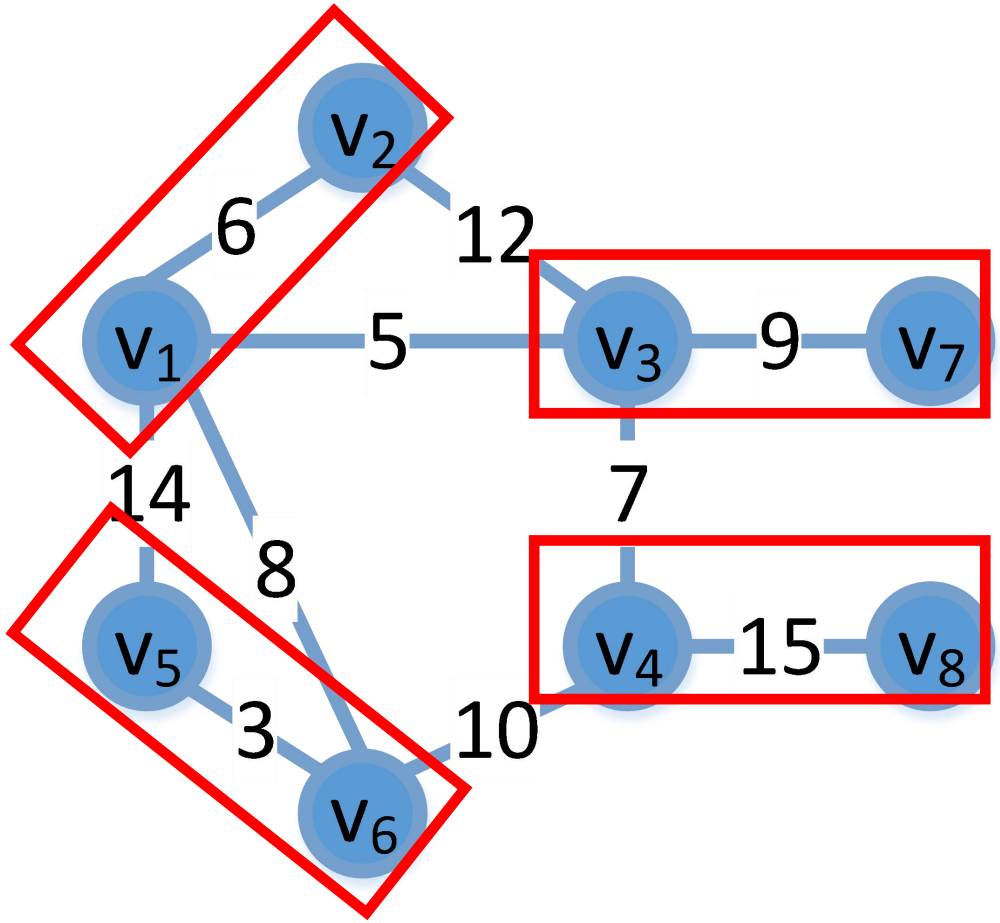
假设要在某地建造5个工厂，拟修筑道路连接这5处。经勘探，其道路可按下图的无向边铺设。现在每条边的长度已经测出并标记在图的对应边上，如果我们要求铺设的道路总长度最短，这样既能节省费用 ，又能缩短工期 ，如何铺设？



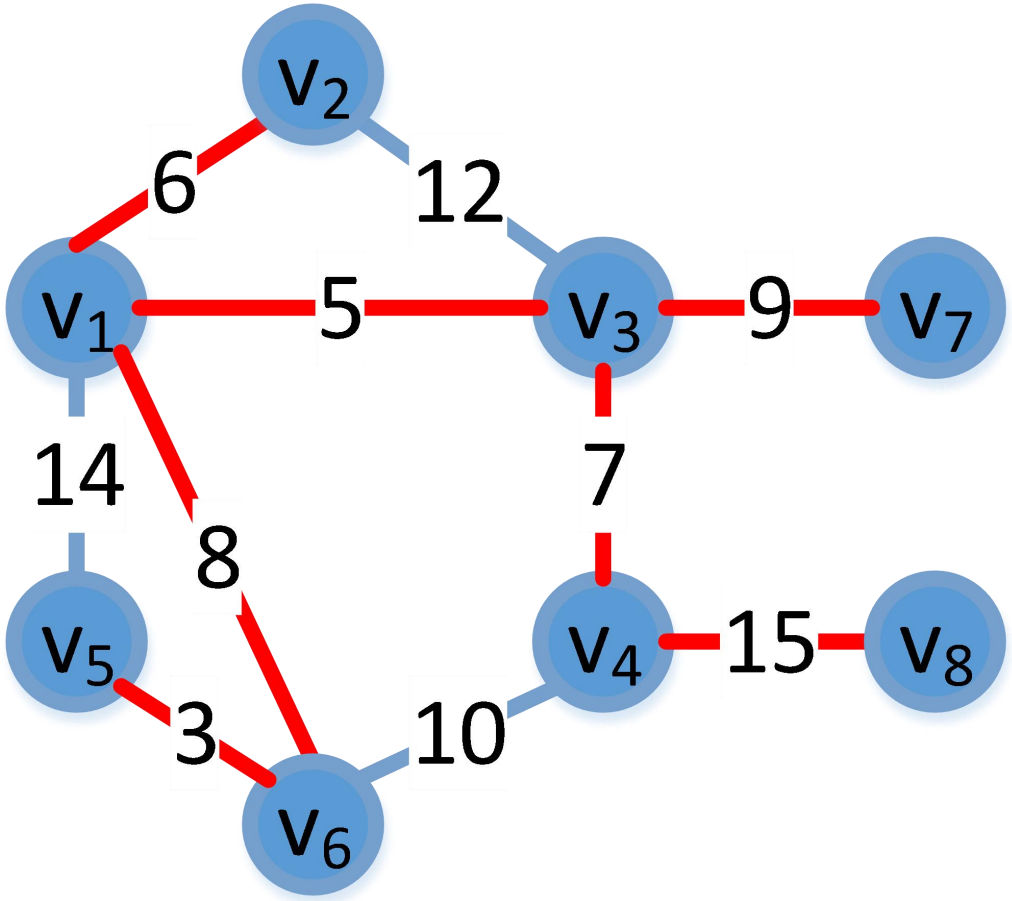
最小生成树的生成


Vnet_25V
C31
0.1uF
NGND
C36
10uF
C37
10uF
C38
0.1uF
NGND
C32
0.1uF
C33
0.1uF
C34
0.1uF
C35
10uF
DGND
R11
50
R12
50
C39
15pF
C40
15pF
T5
NTD-
NTD+
NRD-
NRD+
1
2
3
4
5
6
7
8
TD+
CT
TD-
NC
NC
RD+
CT
RD-
TX+
CM
TX-
NC
NC
RX+
CM
RX-
16
15
14
13
12
11
10
9
H1102
J4
TX_D1+
TX_D1-
RX_D2+
BI_D3+
BI_D3-
RX_D2-
BI_D4+
BI_D4-
RJ45
R13
50
R14
50
C41
15pF
C42
15pF
R15
75
R16
75
R17
75
C43
0.1uF
NGND
C44
0.1uF/2KV
NPGND

最小生成树的生成



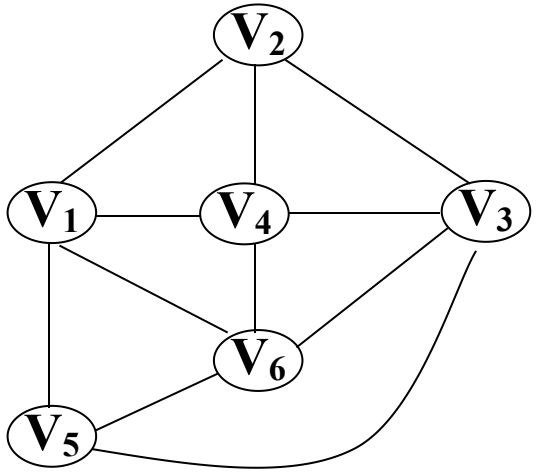
最小生成树的生成



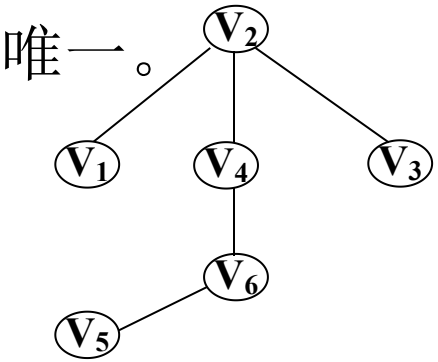
最小生成树的生成

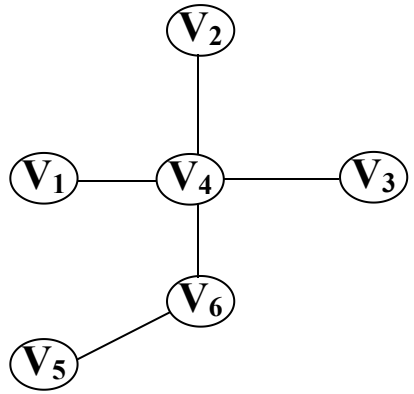
生成树定义：

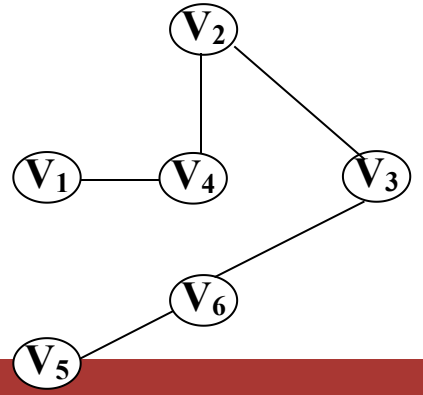
一个连通图的生成树是一个极小连通子图，它含有图中全部顶点，但只有足以构成一棵树的n-1条边。生成树不唯一。



生成树







最小生成树的生成

最小生成树定义（Minimum Spanning Tree, MST）

给定一个无向图G=(V, E, W)，它的最小生成树T是它的极小连通子图，且包含G中的所有|V|个结点，并且有保持整个树的权重之和最小，即

$$W(T) = \min_{T \subseteq G} \sum_{e \in T} w(e)$$

最小生成树的生成

- 两种常用的构造最小生成树的方法(贪心)：
  - Prim(普里姆)算法
  - Kruskal(克鲁斯卡尔)算法

湖南大学 HUNAN UNIVERSITY

目录

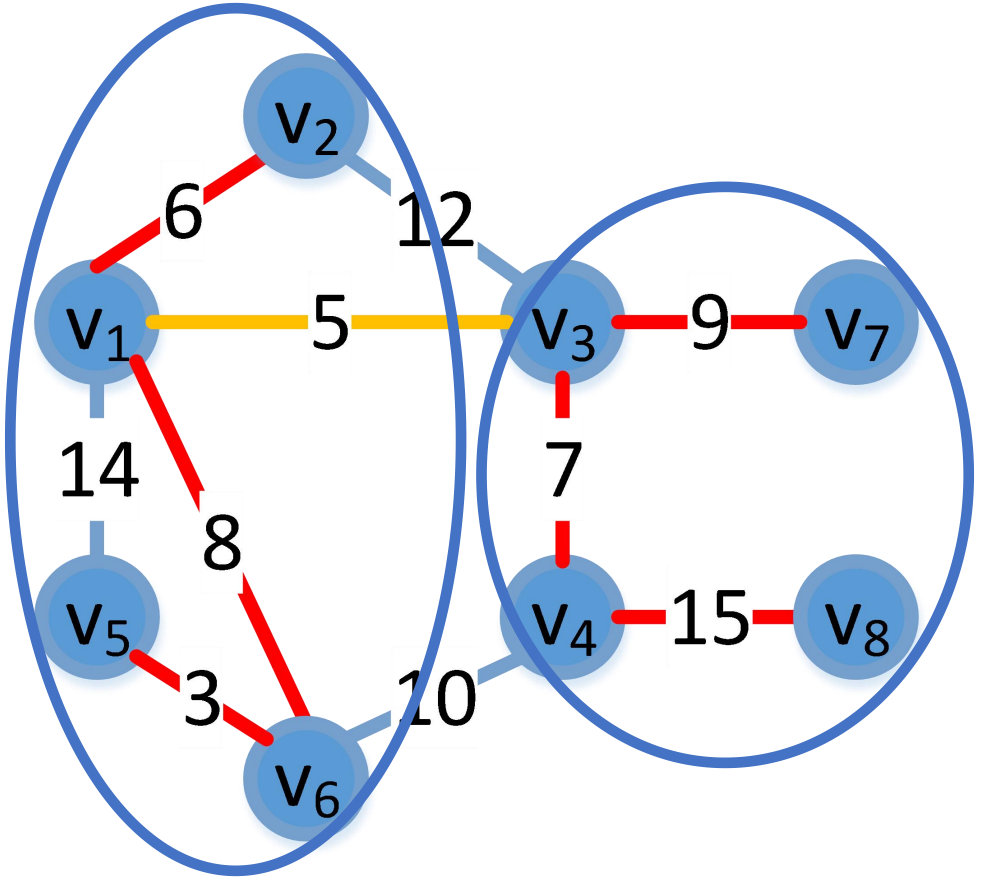
最小生成树的生成

最优子结构性质

给定一个无向图G=(V, E, W)以及它的最小生成树T，如果$A \subseteq V$且边(u,v)是连接A和V-A的所有边中权最小的边，那么边(u,v)在T中。

最小生成树的生成

最优子结构示例

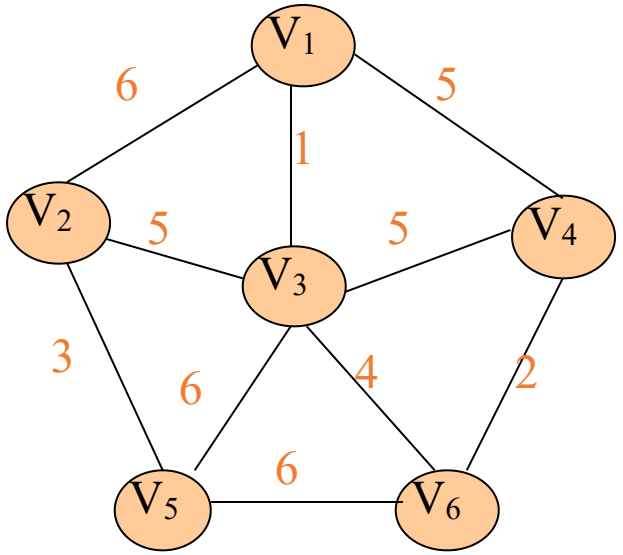


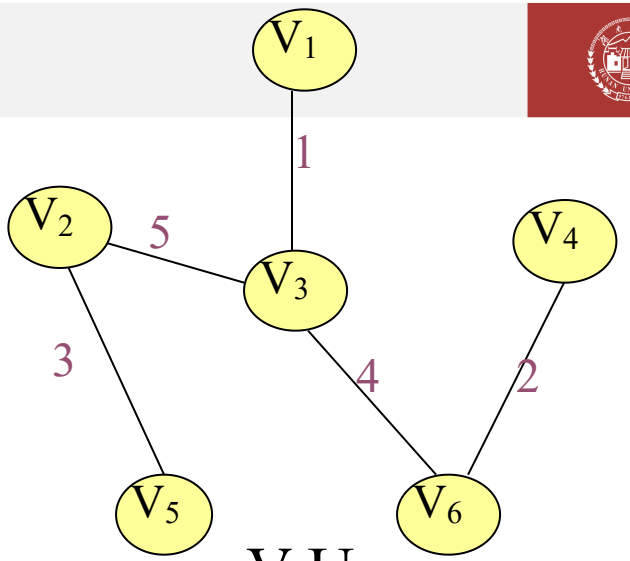
Prim算法(普里姆)

- 假设N=(V，E)是连通网，TE是N上最小生成树中边的集合。
- 算法从U={u_0}(u_0∈V)，TE={}开始，重复执行下述操作：
  - 在所有u∈U，v∈V-U的边(u，v)中找一条代价最小的边(u_0,v_0)，将其并入集合TE，同时将v_0并入U集合。
  - 当U=V则结束，此时TE中必有n-1条边，则T=(V，{TE})为N的最小生成树。
- 普里姆算法构造最小生成树的过程是从一个顶点U={u_0}作初态，不断寻找与U中顶点相邻且代价最小的边的另一个顶点，扩充到U集合直至U=V为止。

Prim算法(普里姆)

- 普里姆算法求最小生成树：从生成树中只有一个顶点开始，到顶点全部进入生成树为止。

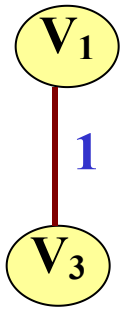


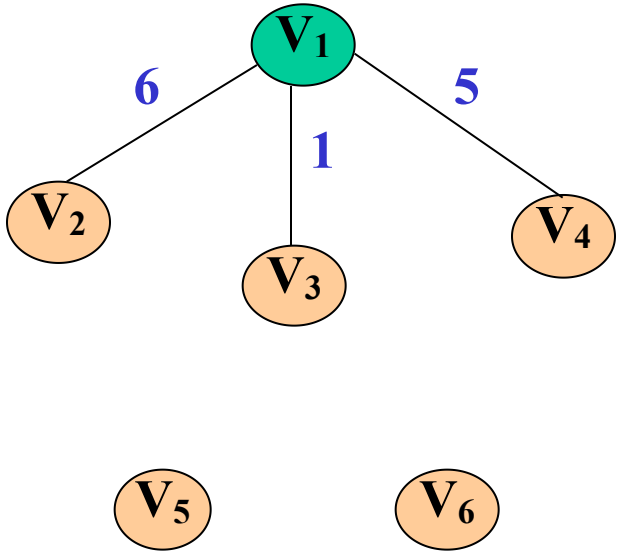


步骤	U	V-U
(0)	{ V_1 }	{ V_2 , V_3 , V_4 , V_5 , V_6 }
(1)	{ V_1 , V_3 }	{ V_2 , V_4 , V_5 , V_6 }
(2)	{ V_1 , V_3 , V_6 }	{ V_2 , V_4 , V_5 }
(3)	{ V_1 , V_3 , V_6 , V_4 }	{ V_2, V_5 }
(4)	{ V_1 , V_3 , V_6 , V_4 , V_2 }	{ V_5 }
(5)	{ V_1 , V_3 , V_6 , V_4 , V_2 , V_5 }	{ }

Prim算法(普里姆)

- 普里姆算法求最小生成树：从生成树中只有一个顶点开始，到顶点全部进入生成树为止

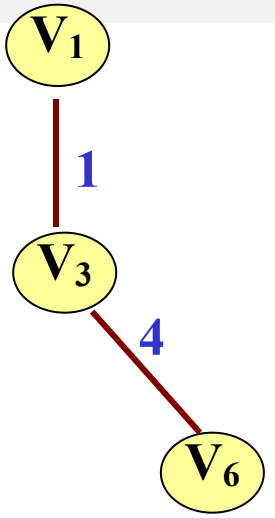


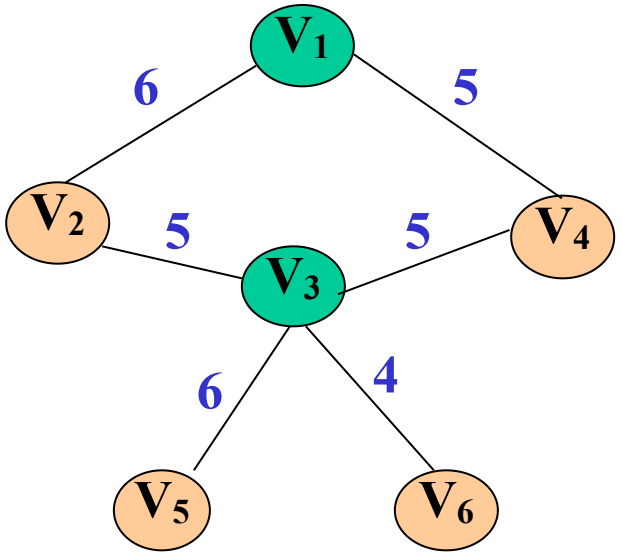


步骤	U	V-U
(0)	{ V_1 }	{ V_2 ,V_3 ,V_4 , V_5 ,V_6 }
(1)	{ V_1 ,V_3 }	{ V_2 ,V_4 , V_5 ,V_6 }

Prim算法(普里姆)

- 普里姆算法求最小生成树：从生成树中只有一个顶点开始，到顶点全部进入生成树为止

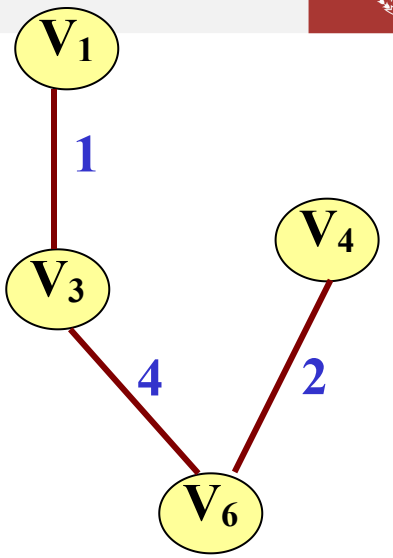


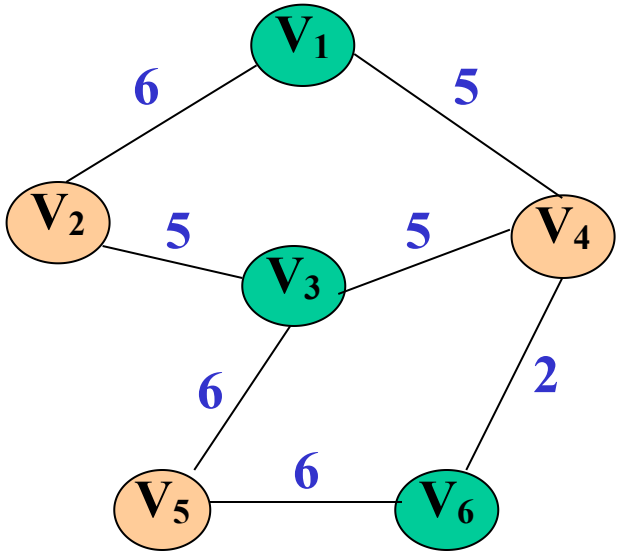


步骤	U	V-U
(0)	$\{V_1\}$	$\{V_2, V_3, V_4, V_5, V_6\}$
(1)	$\{V_1, V_3\}$	$\{V_2, V_4, V_5, V_6\}$
(2)	$\{V_1, V_3, V_6\}$	$\{V_2, V_4, V_5\}$

Prim算法(普里姆)

- 普里姆算法求最小生成树：从生成树中只有一个顶点开始，到顶点全部进入生成树为止

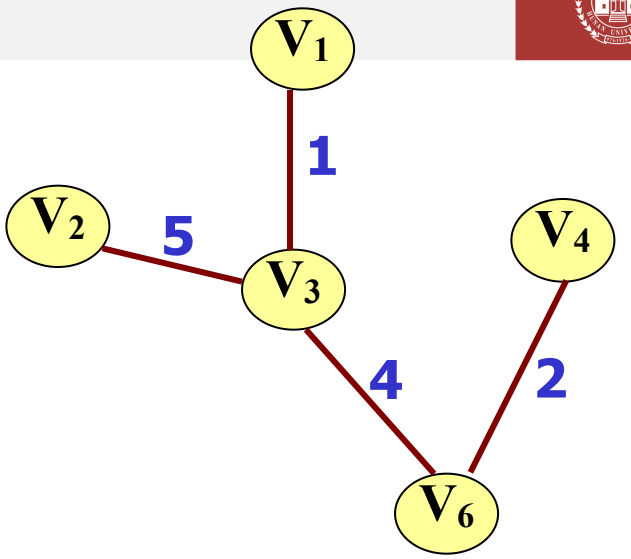


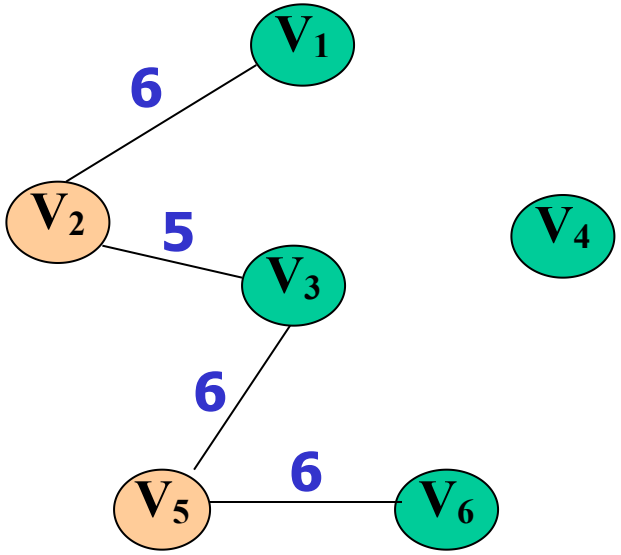


步骤	U	V-U
(0)	$\{V_1\}$	$\{V_2, V_3, V_4, V_5, V_6\}$
(1)	$\{V_1, V_3\}$	$\{V_2, V_4, V_5, V_6\}$
(2)	$\{V_1, V_3, V_6\}$	$\{V_2, V_4, V_5\}$
(3)	$\{V_1, V_3, V_6, V_4\}$	$\{V_2, V_5\}$

Prim算法(普里姆)

- 普里姆算法求最小生成树：从生成树中只有一个顶点开始，到顶点全部进入生成树为止

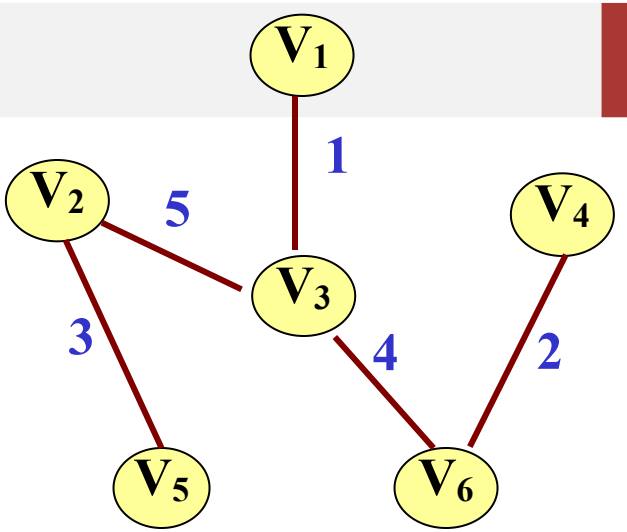


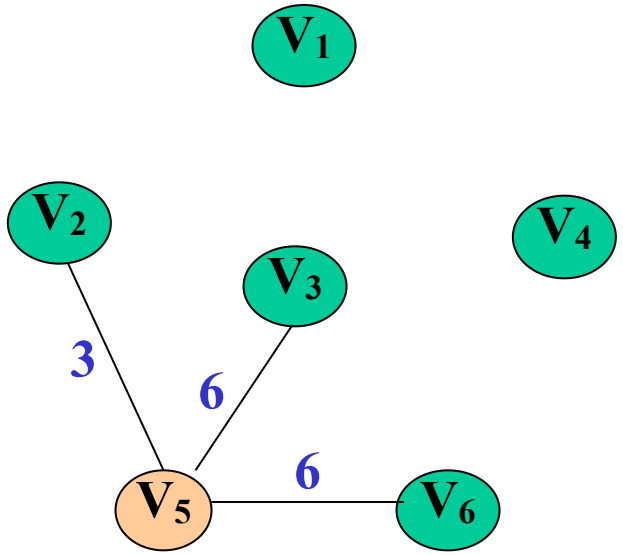


步骤	U	V-U
(0)	$\{V_1\}$	$\{V_2, V_3, V_4, V_5, V_6\}$
(1)	$\{V_1, V_3\}$	$\{V_2, V_4, V_5, V_6\}$
(2)	$\{V_1, V_3, V_6\}$	$\{V_2, V_4, V_5\}$
(3)	$\{V_1, V_3, V_6, V_4\}$	$\{V_2, V_5\}$
(4)	$\{V_1, V_3, V_6, V_4, V_2\}$	$\{V_5\}$

Prim算法(普里姆)

- 普里姆算法求最小生成树：从生成树中只有一个顶点开始，到顶点全部进入生成树为止

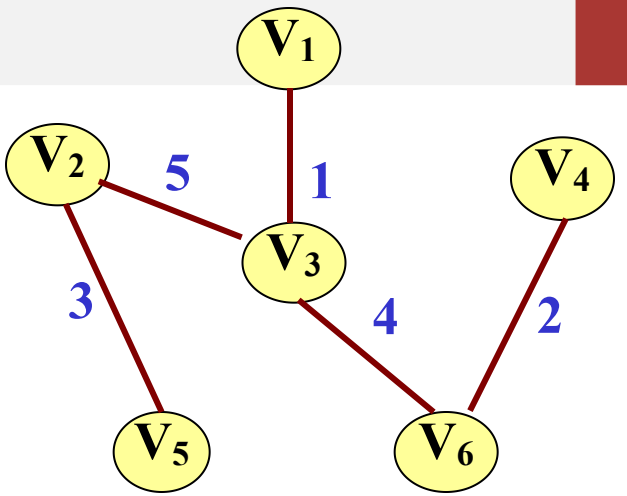


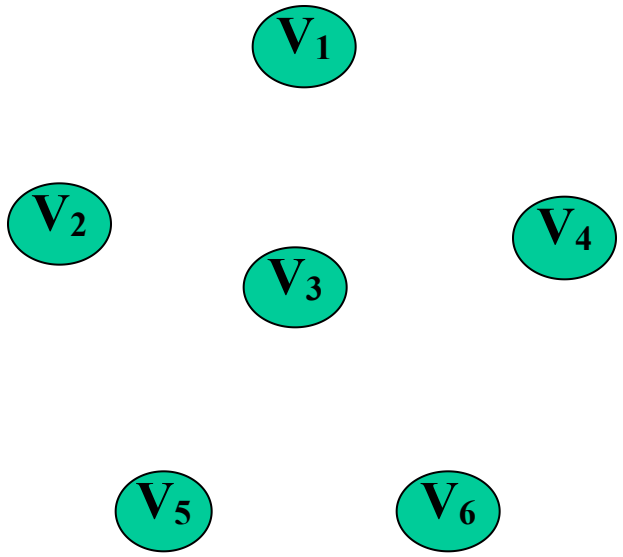


步骤	U	V-U
(0)	$\{V_1\}$	$\{V_2, V_3, V_4, V_5, V_6\}$
(1)	$\{V_1, V_3\}$	$\{V_2, V_4, V_5, V_6\}$
(2)	$\{V_1, V_3, V_6\}$	$\{V_2, V_4, V_5\}$
(3)	$\{V_1, V_3, V_6, V_4\}$	$\{V_2, V_5\}$
(4)	$\{V_1, V_3, V_6, V_4, V_2\}$	$\{V_5\}$
(5)	$\{V_1, V_3, V_6, V_4, V_2, V_5\}$	$\{\}$

Prim算法(普里姆)

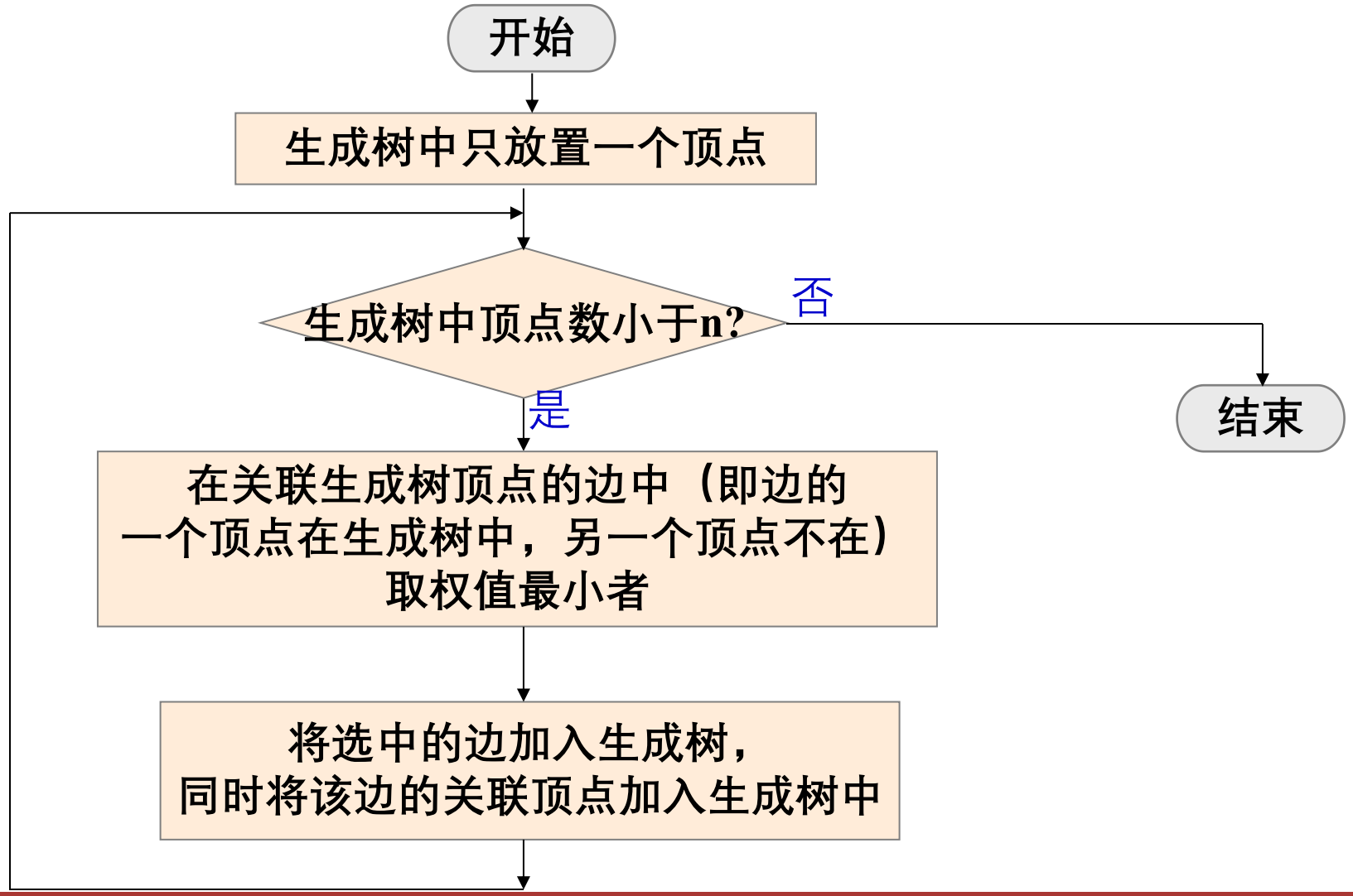
- 普里姆算法求最小生成树：从生成树中只有一个顶点开始，到顶点全部进入生成树为止





步骤	U	V-U
(0)	$\{V_1\}$	$\{V_2, V_3, V_4, V_5, V_6\}$
(1)	$\{V_1, V_3\}$	$\{V_2, V_4, V_5, V_6\}$
(2)	$\{V_1, V_3, V_6\}$	$\{V_2, V_4, V_5\}$
(3)	$\{V_1, V_3, V_6, V_4\}$	$\{V_2, V_5\}$
(4)	$\{V_1, V_3, V_6, V_4, V_2\}$	$\{V_5\}$
(5)	$\{V_1, V_3, V_6, V_4, V_2, V_5\}$	$\{\}$

Prim算法(普里姆)


开始
生成树中只放置一个顶点
生成树中顶点数小于n?
否
结束
是
在关联生成树顶点的边中（即边的
一个顶点在生成树中，另一个顶点不在）
取权值最小者
将选中的边加入生成树，
同时将该边的关联顶点加入生成树中

Prim算法(普里姆)

```
1  Prim(G, Adj, s)
2   key[s]=0;
3   for each v ∈ V – {s}
4      do key[v] = ∞
5   Q = V
6   while Q ≠ ∅  :
7      u = EXTRACT-MIN(Q);
8      for each v ∈ Adj[u]:
9         if key[v] > w(u, v)
10           then key[v] = w(u, v)
```

Prim算法(普里姆)

时间复杂度

$$\text{Time} = \Theta(V)\cdot T_{\text{EXTRACT-MIN}} + \Theta(E)\cdot T_{\text{DECREASE-KEY}}$$

Q	$T_{\text{EXTRACT-MIN}}$	$T_{\text{DECREASE-KEY}}$	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap	$O(\lg V)$ amortized	$O(1)$ amortized	$O(E + V \lg V)$ worst case

湖南大學 HUNAN UNIVERSITY

目录

最小生成树的生成

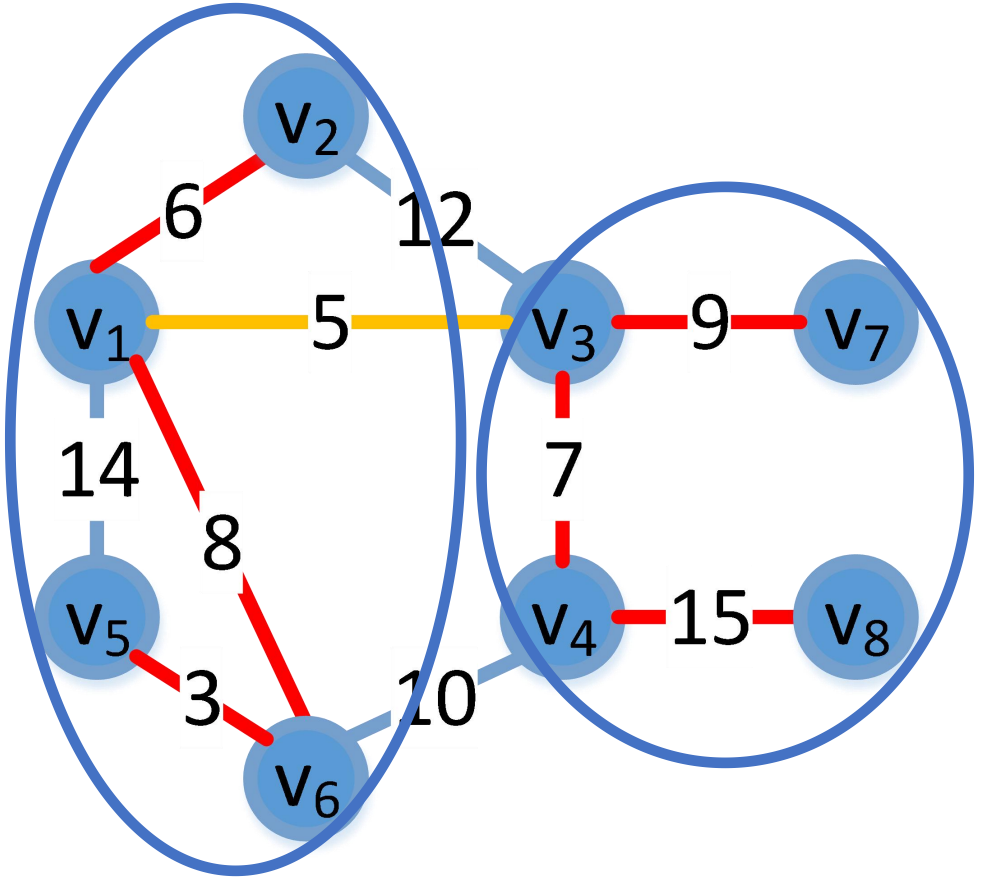
最优子结构性质

给定一个无向图G=(V, E, W)以及它的最小生成树T，如果从T中去除一条边(u, v)进而将T划分成两个子树T_1和T_2，那么T_1和T_2分别是图G_1 =(V_1, E_1, W)和G_2 =(V_2, E_2, W)的最小生成树，其中V_i（i=1或2）是T_i中的点且G_i是V_i的导出子图。

证明：根据W(T)=W(T1)+W(T2)+W(u, v)进行反证

最小生成树的生成

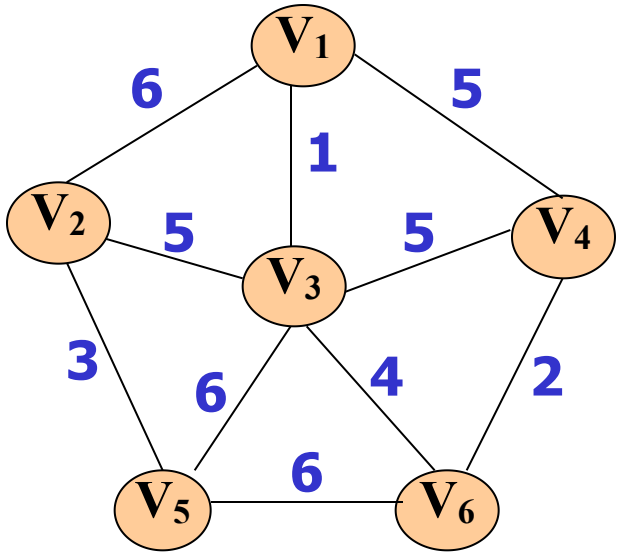
最优子结构示例

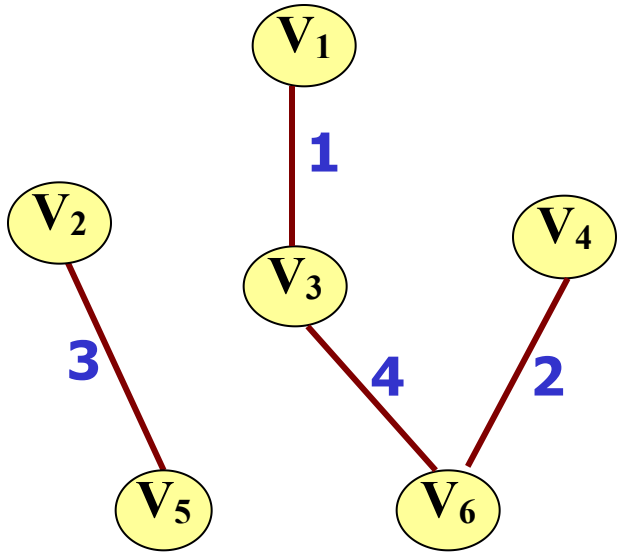


Kruskal算法(克鲁斯卡尔)

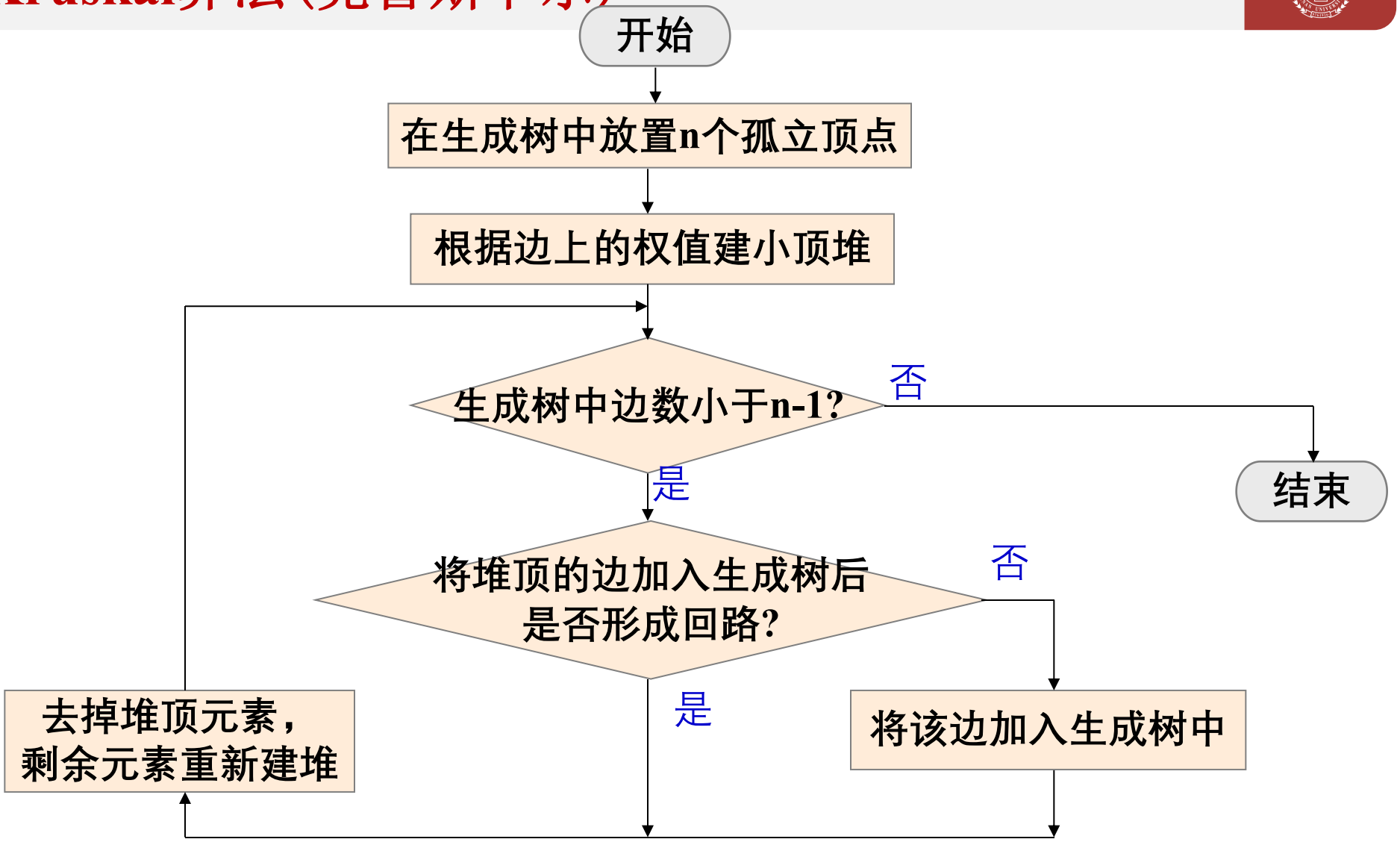
- 假设连通网N=(V，E)，则令最小生成树的初始状态为只有n个顶点而无边的非连通图T=(V，{})，图中每个顶点自成一个连通分量。
- 在E中选择代价最小的边，若该边依附的顶点落在T中不同的连通分量上，则将此边加入到T中，否则舍去此边而选择下一条代价最小的边。依次类推，直至T中所有顶点都在同一连通分量上为止。

Kruskal算法(克鲁斯卡尔)





Kruskal算法(克鲁斯卡尔)



Kruskal算法(克鲁斯卡尔)

- 从上述过程可知，实现克鲁斯卡尔(Kruskal)算法时，要解决以下两个问题：
  - 如何选择代价最小的边(堆排序，或简单选择排序)；
  - 如何判定边所关联的两个顶点是否在同一个连通分量中（集合）

Kruskal算法(克鲁斯卡尔)

并查集

在一些有N个元素的集合应用问题中，我们通常是在开始时让每个元素构成一个单元素的集合，然后按一定顺序将属于同一组的元素所在的集合合并，其间要反复查找一个元素在哪个集合中

Kruskal算法(克鲁斯卡尔)

并查集-基本算法

- **function** MakeSet(x)

  x.parent = x

- **function** Union(x, y)

  xRoot = Find(x)

  yRoot = Find(y)

  xRoot.parent = yRoot

➢**function** Find(x)

  if x.parent == x

    return x

  else

    return Find(x.parent)

Kruskal算法(克鲁斯卡尔)

并查集-按秩合并

总是将更小的树连接至更大的树上

```
function MakeSet(x)
    x.parent := x
    x.rank   := 0
```

```
function Union(x, y)
    xRoot = Find(x); yRoot = Find(y)
    if xRoot == yRoot
        return
    if xRoot.rank < yRoot.rank
        xRoot.parent = yRoot
    else if xRoot.rank > yRoot.rank
        yRoot.parent = xRoot
    else
        yRoot.parent := xRoot
        xRoot.rank := xRoot.rank + 1
```

Kruskal算法(克鲁斯卡尔)

并查集-路径压缩

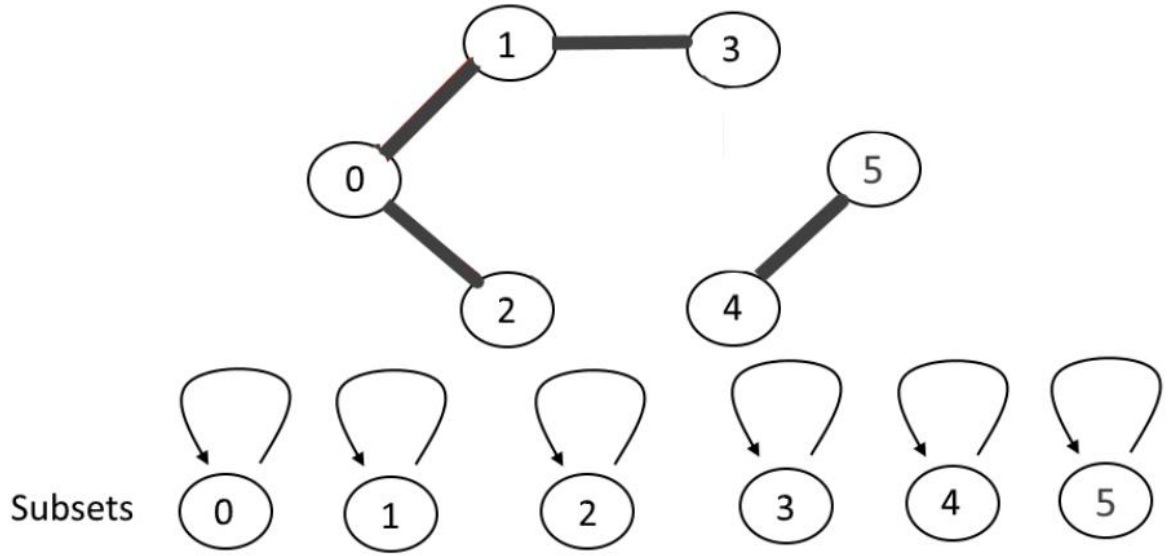
Find递归地经过树，改变每一个节点的引用到根节点

```
function Find(x)
    if x.parent != x
        x.parent = Find(x.parent)
    return x.parent
```

Kruskal算法(克鲁斯卡尔)

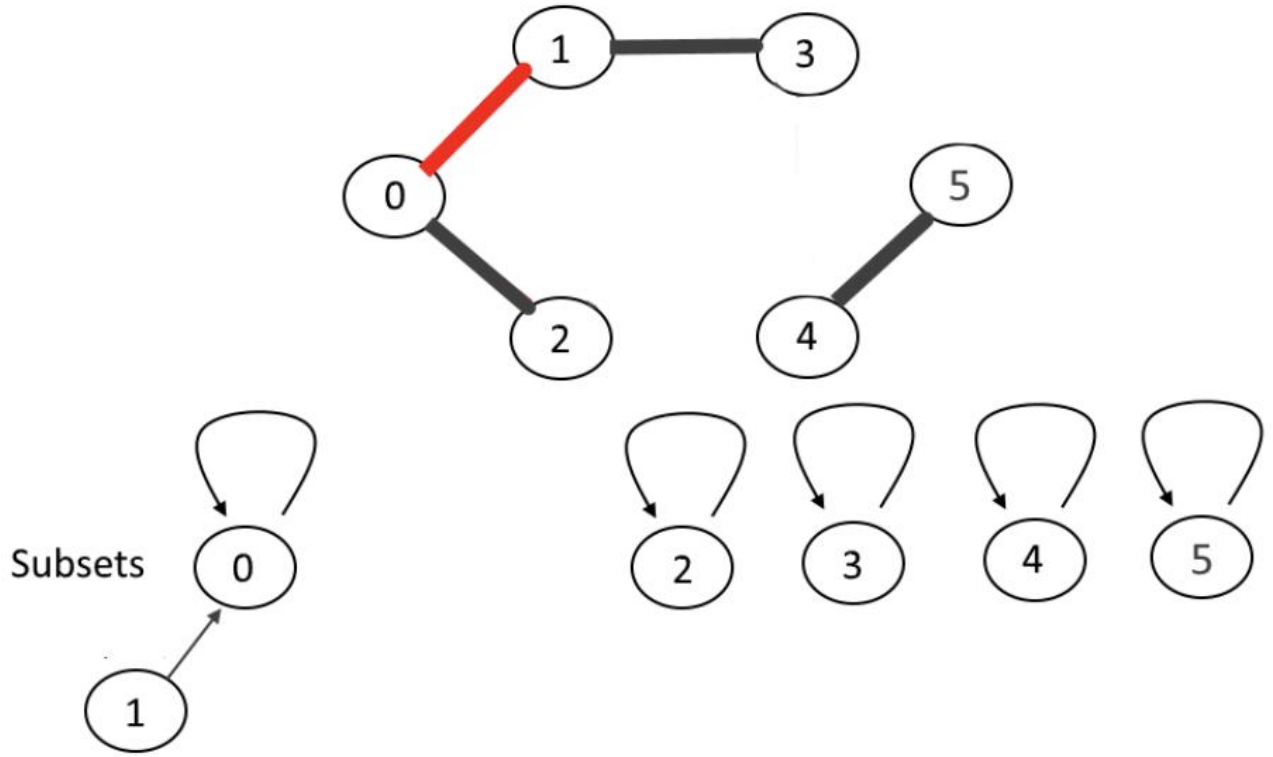
并查集-示例

初始的时候每个点的parent都是自己



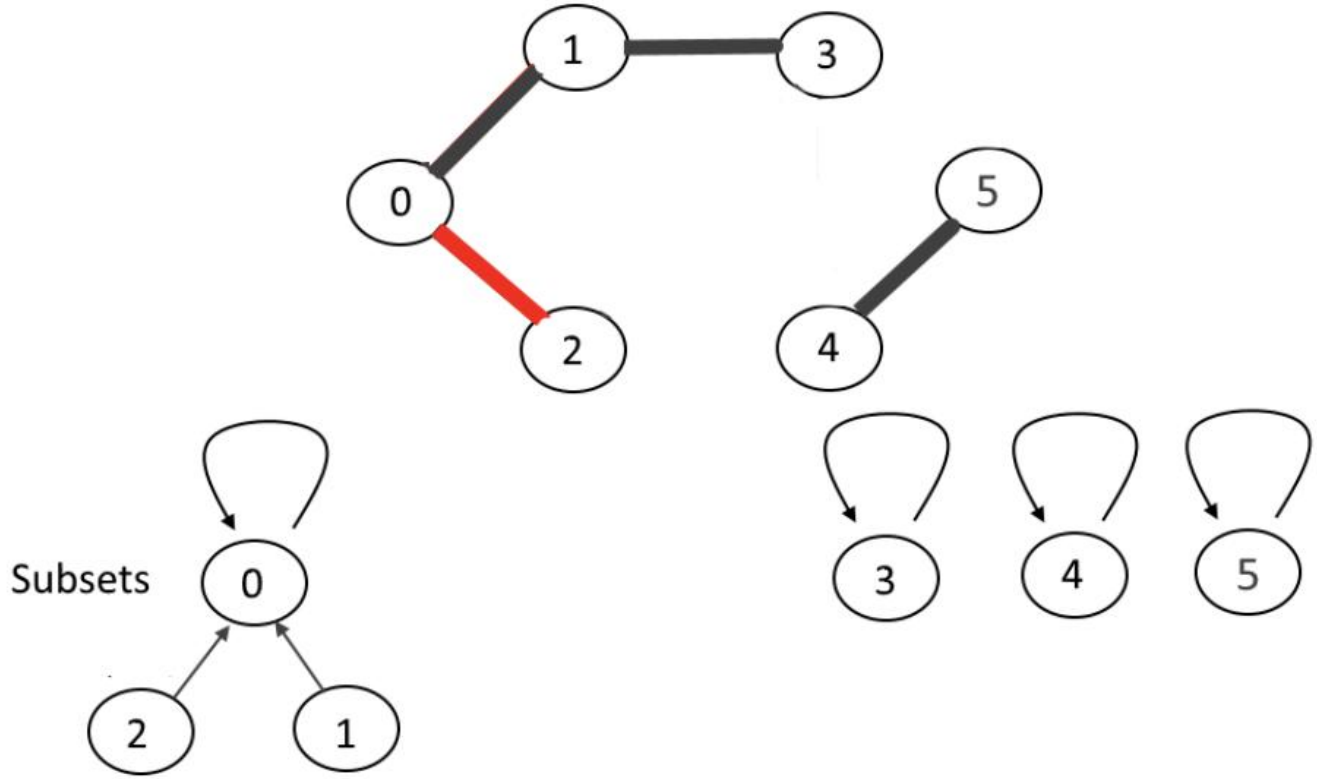
Kruskal算法(克鲁斯卡尔)

并查集-示例



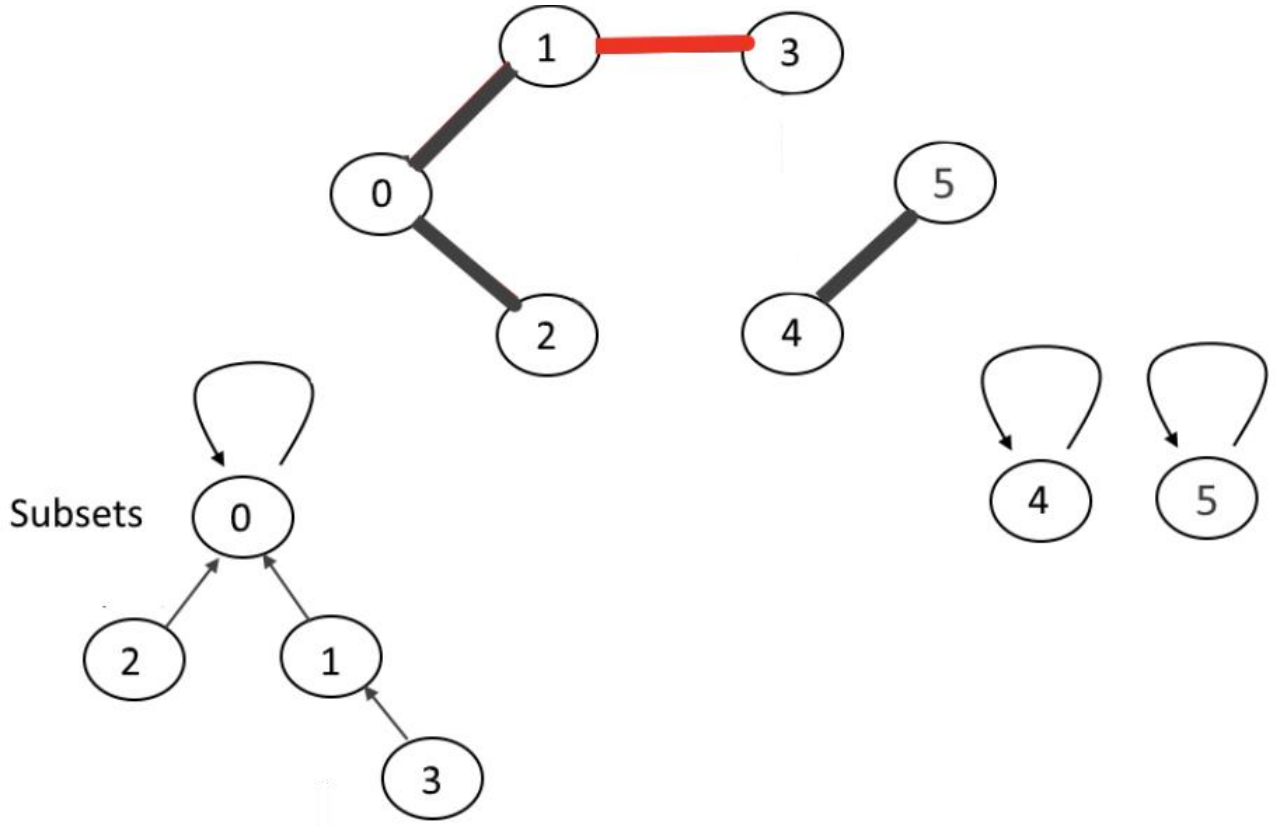
Kruskal算法(克鲁斯卡尔)

并查集-示例



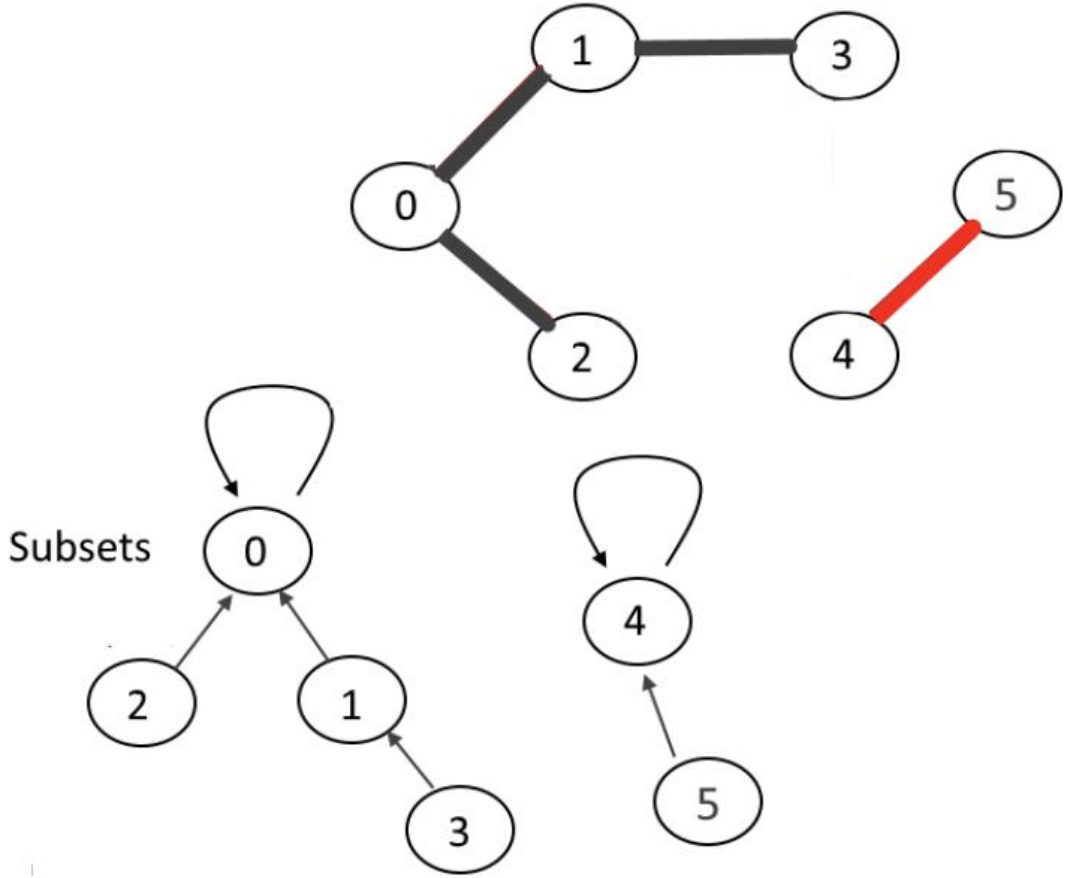
Kruskal算法(克鲁斯卡尔)

并查集-示例



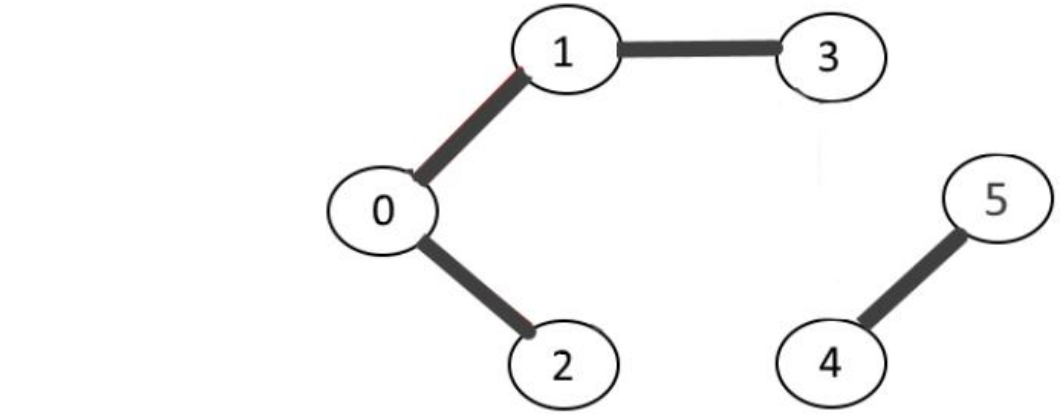
Kruskal算法(克鲁斯卡尔)

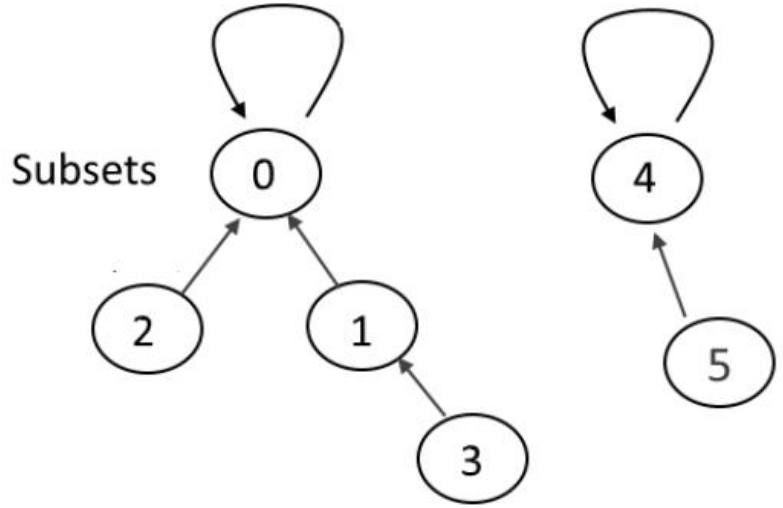
并查集-示例



Kruskal算法(克鲁斯卡尔)

并查集-示例





Kruskal算法(克鲁斯卡尔)

Kruskal算法伪代码

- Kruskal(G)

```
A= ∅ ;
for each v ∈ V
    MAKE-Tree(v)
sort E;
for each e = (u,v) in E in nondecreasing order:
    if FIND-ROOT(u)≠ FIND-ROOT(v)
        A= S ∪ {(u,v)};
        UNION(u, v)
```

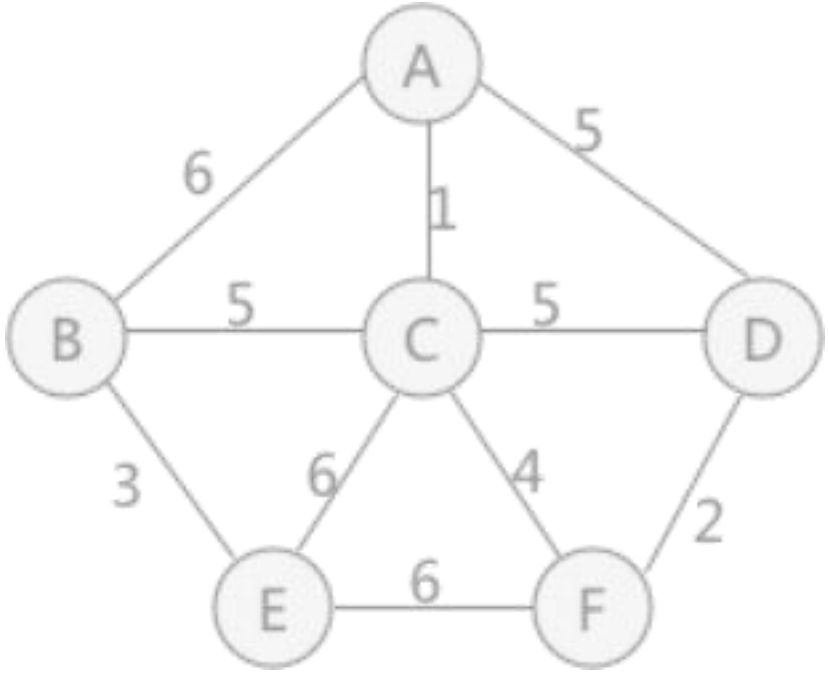
O(|E|log|E|)

O(α(|V|))

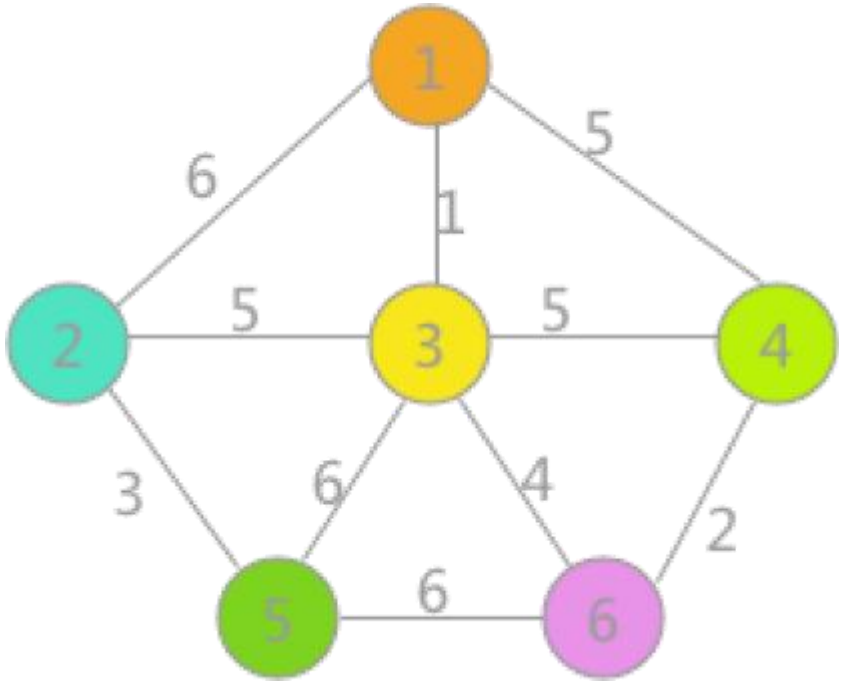
O(|E|)

于是，整体复杂度是O(|E|log|E|)= O(|E|log|V|)

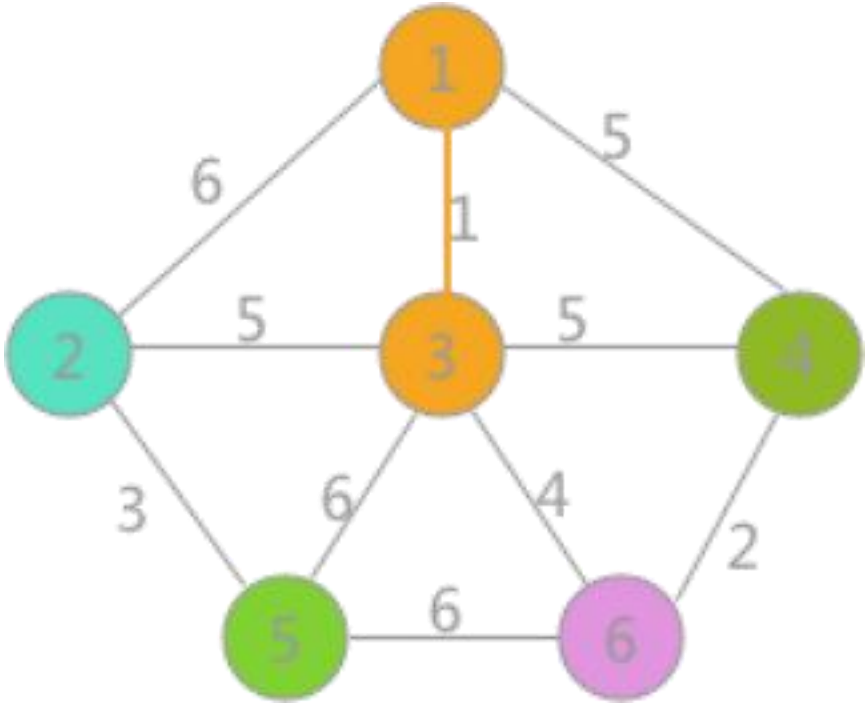
Kruskal算法(克鲁斯卡尔)



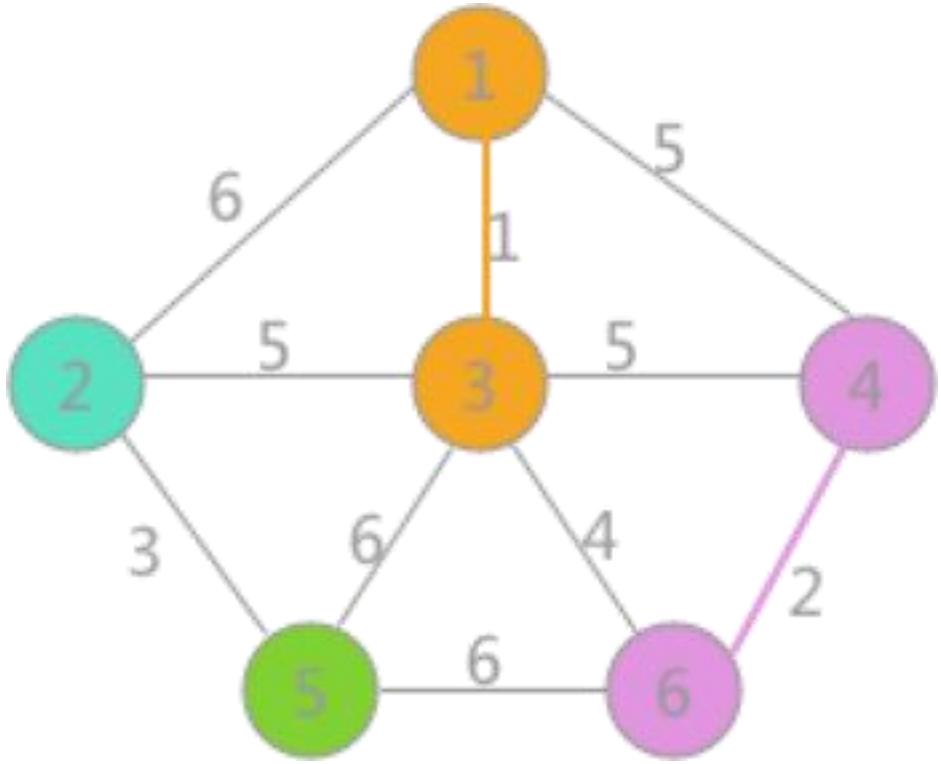
Kruskal算法(克鲁斯卡尔)



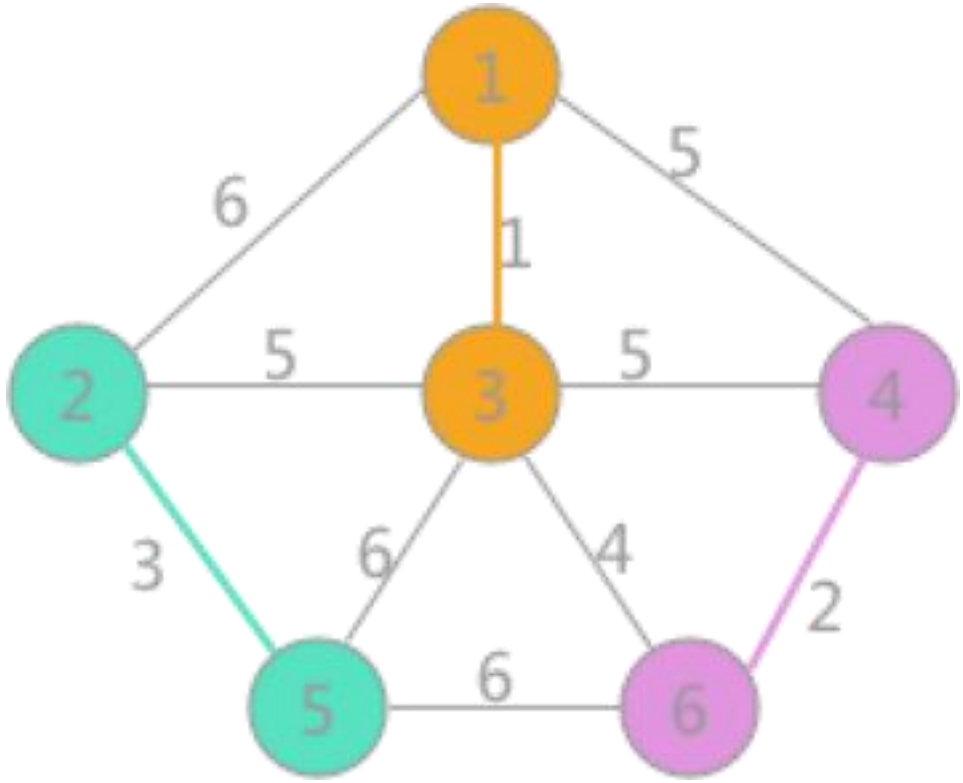
Kruskal算法(克鲁斯卡尔)



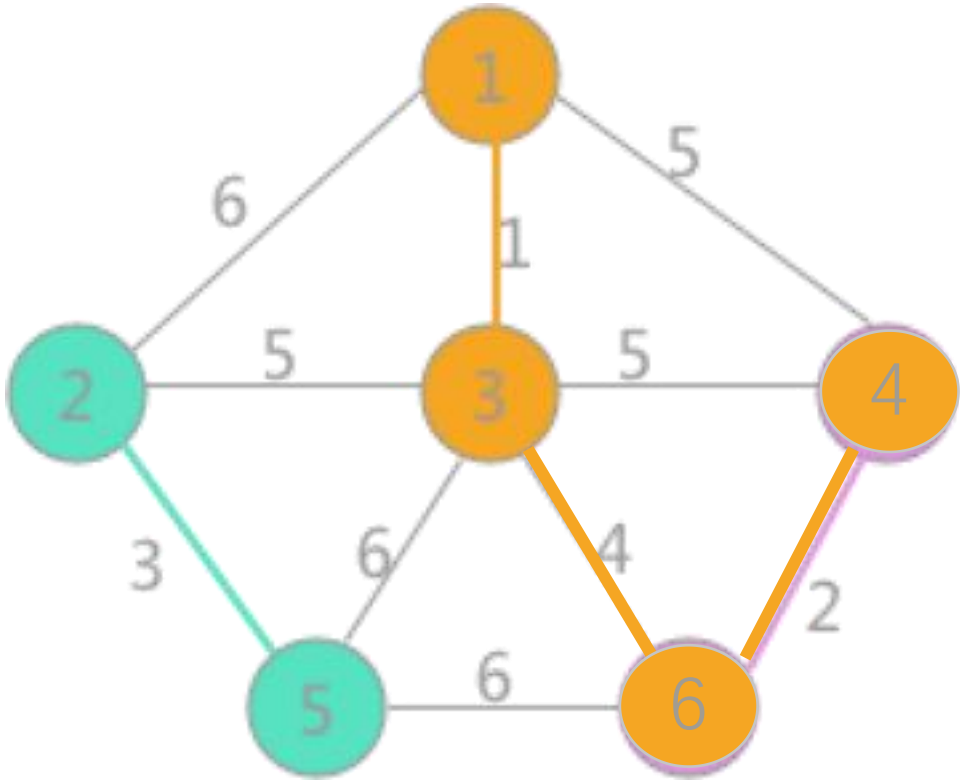
Kruskal算法(克鲁斯卡尔)



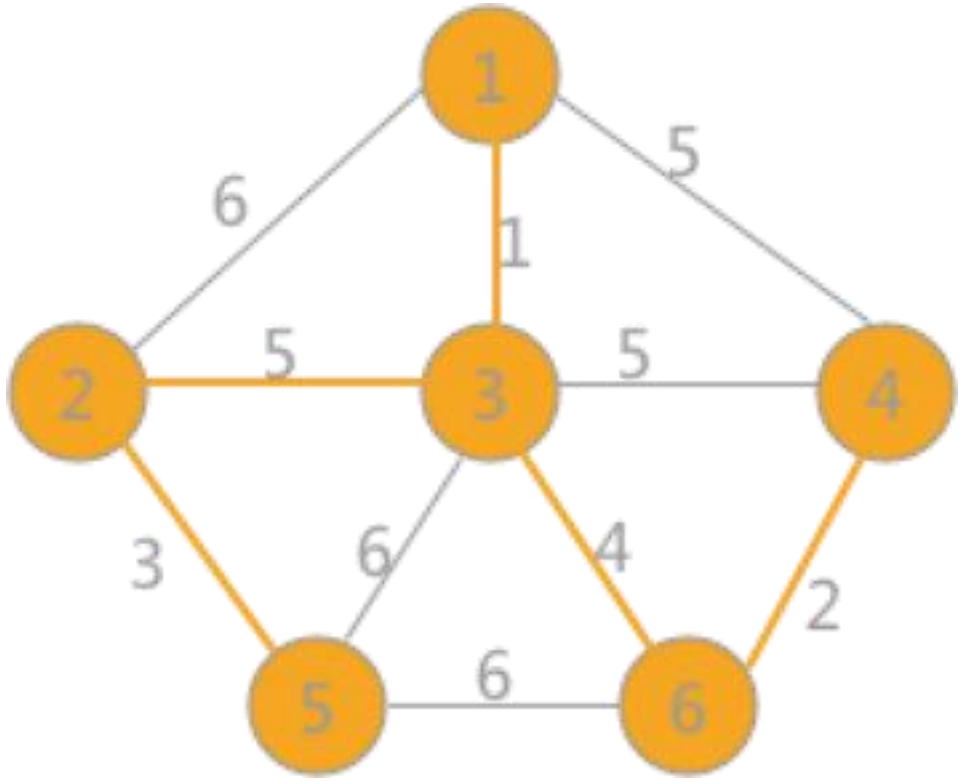
Kruskal算法(克鲁斯卡尔)



Kruskal算法(克鲁斯卡尔)

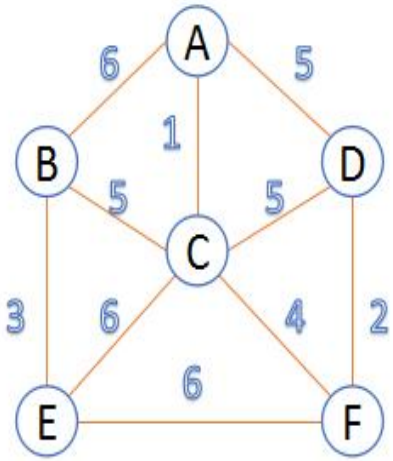


Kruskal算法(克鲁斯卡尔)

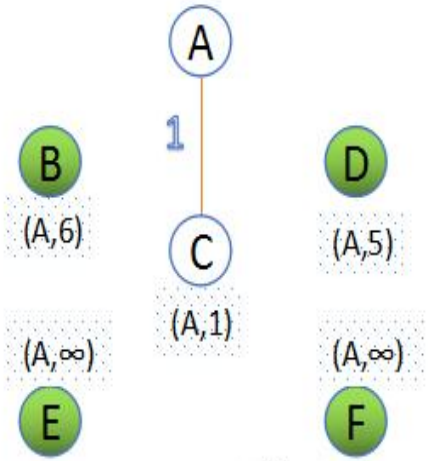


湖南大学 HUNAN UNIVERSITY

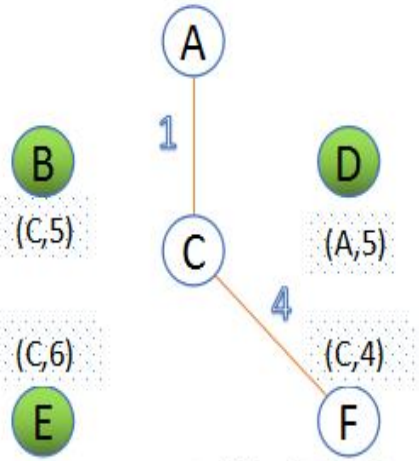
目录



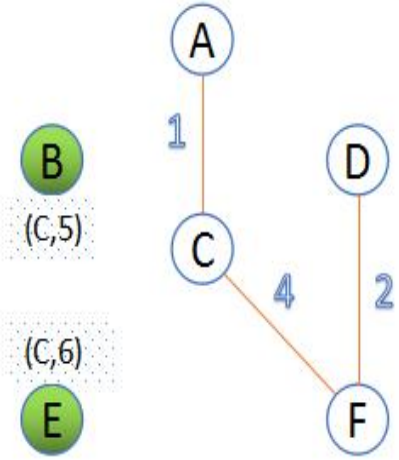
连通网G



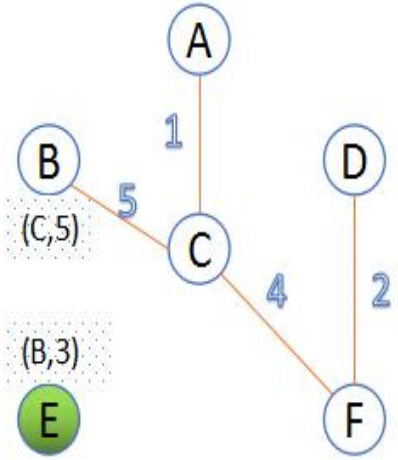
1.初始u={A},v={B,C,D,E,F};顶点B下方(A,6),表示与集合u中A的代价为6作为最小代价边。选择最小的代价边(A,C),把C并入到集合u中。



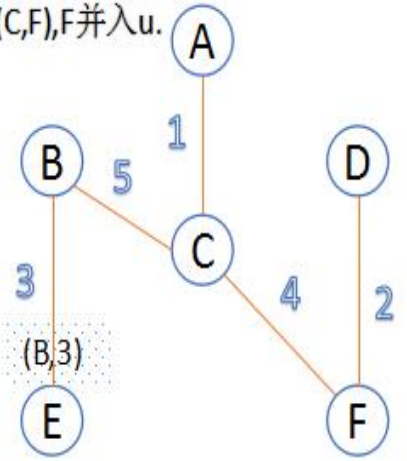
2.u={A,C},v={B,D,E,F};更新v中顶点与集合u的最小的代价边;例如:顶点E之前为(A,∞),更新为(C,6)；选择最小代价边(C,F),F并入u.



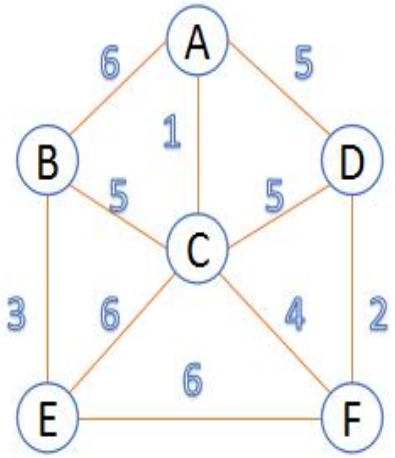
3.u={A,C,F},v={B,D,E};更新v中顶点与集合u的最小的代价边;选择最小代价边(F,D),D并入u.



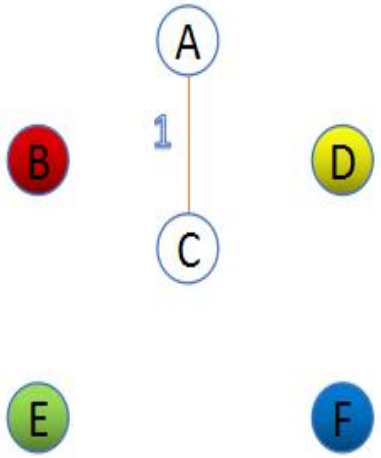
4.u={A,C,F,D},v={B,E};更新v中顶点与集合u的最小的代价边;选择最小代价边(C,B),B并入u.



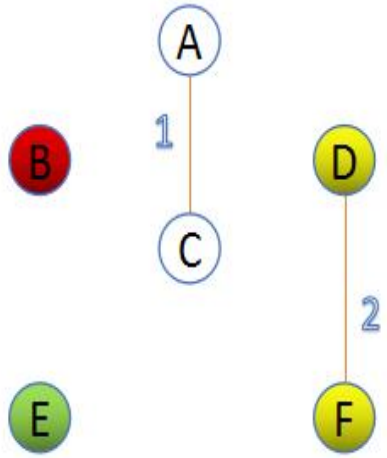
5.u={A,C,F,D,B},v={E};更新v中顶点与集合u的最小的代价边;选择最小代价边(B,E),E并入u.



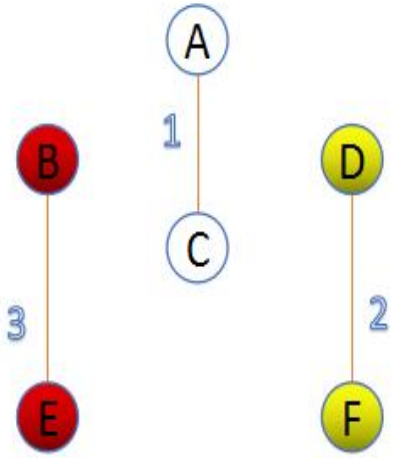
连通网G



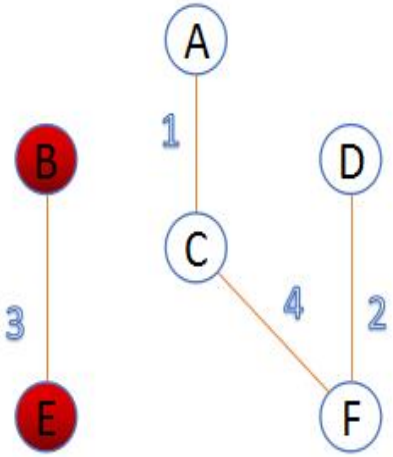
1.选择代价最小的边(A,C);并保证A,C不在同一颗树上,然后合并A,C



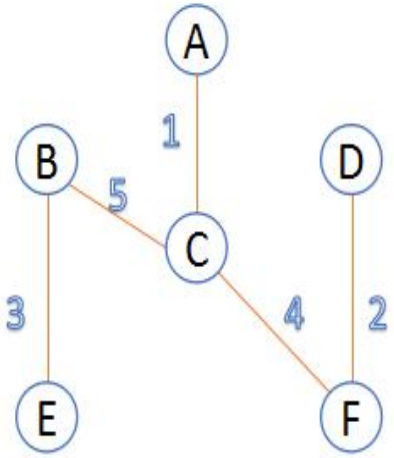
2.选择代价最小的边(D,F);并保证D,F不在同一颗树上,然后合并D,F



3.选择代价最小的边(B,E);并保证B,E不在同一颗树上,然后合并B,E



4.选择代价最小的边(C,F);并保证C,F不在同一颗树上,然后合并C,F所在的树



5.选择代价最小的边(A,D),顶点A,D在同一颗树上,丢弃;
选择最小的边(C,D),顶点C,D在同一颗树上,丢弃;
选择最小的边(B,C),顶点B,C不在同一颗树上,加入此边,然后合并B,C所在的树,此时所有顶点在同一颗树上，返回;

湖南大学 HUNAN UNIVERSITY

目录

并查集应用

最小边属性切割

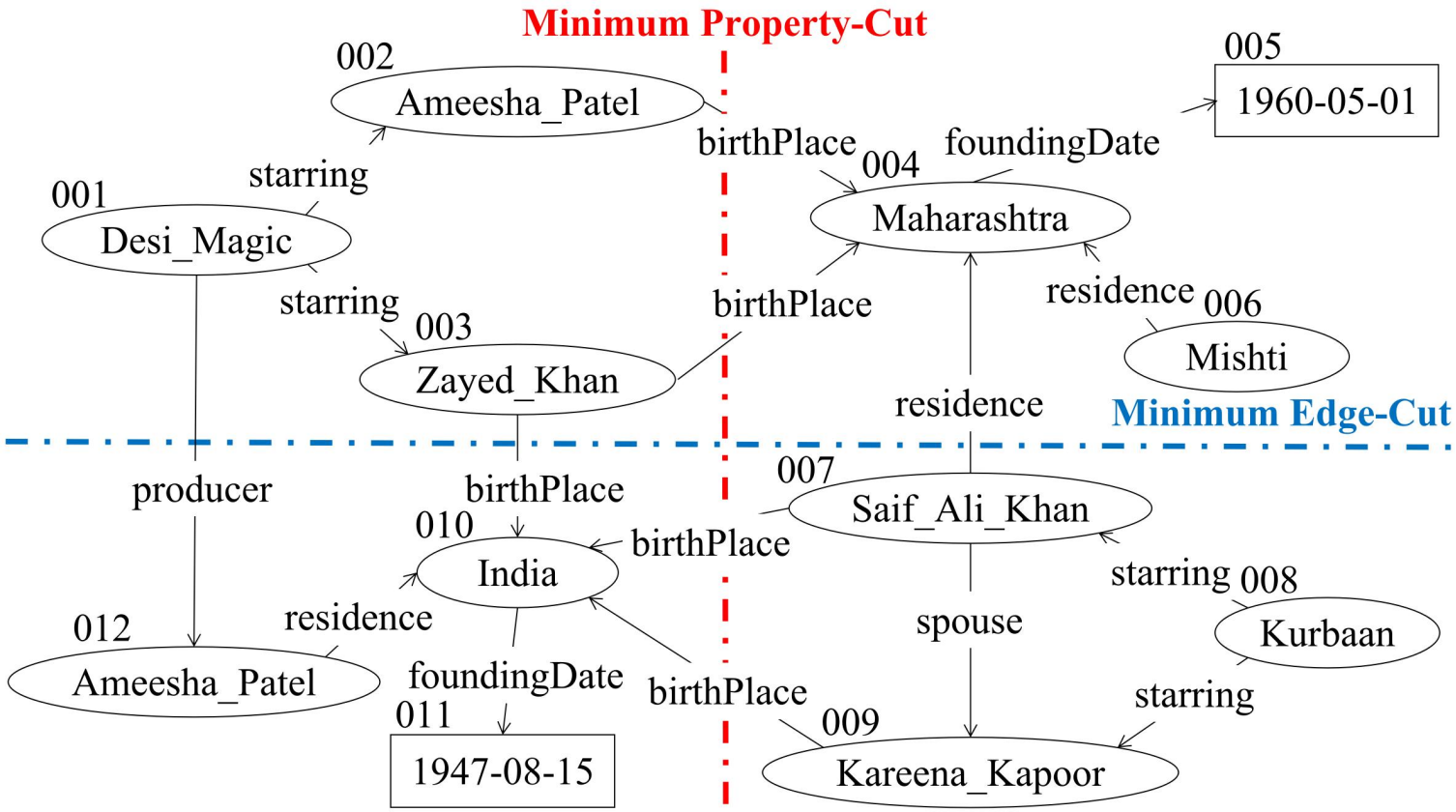
Peng Peng, M. Tamer Özsu, Lei Zou, Cen Yan, Chengjun Liu. MPC: Minimum Property-Cut RDF Graph Partitioning. Submitted in ICDE 2022

Motivations

In a typical decentralized RDF system, a graph G is divided into a set of subgraphs $\{F_1, ..., F_k\}$, called *partitions*.

Min Property-Cut vs Min Edge-Cut

Minimize the number of edges vs Minimize the number of properties.



Internal and Crossing Properties

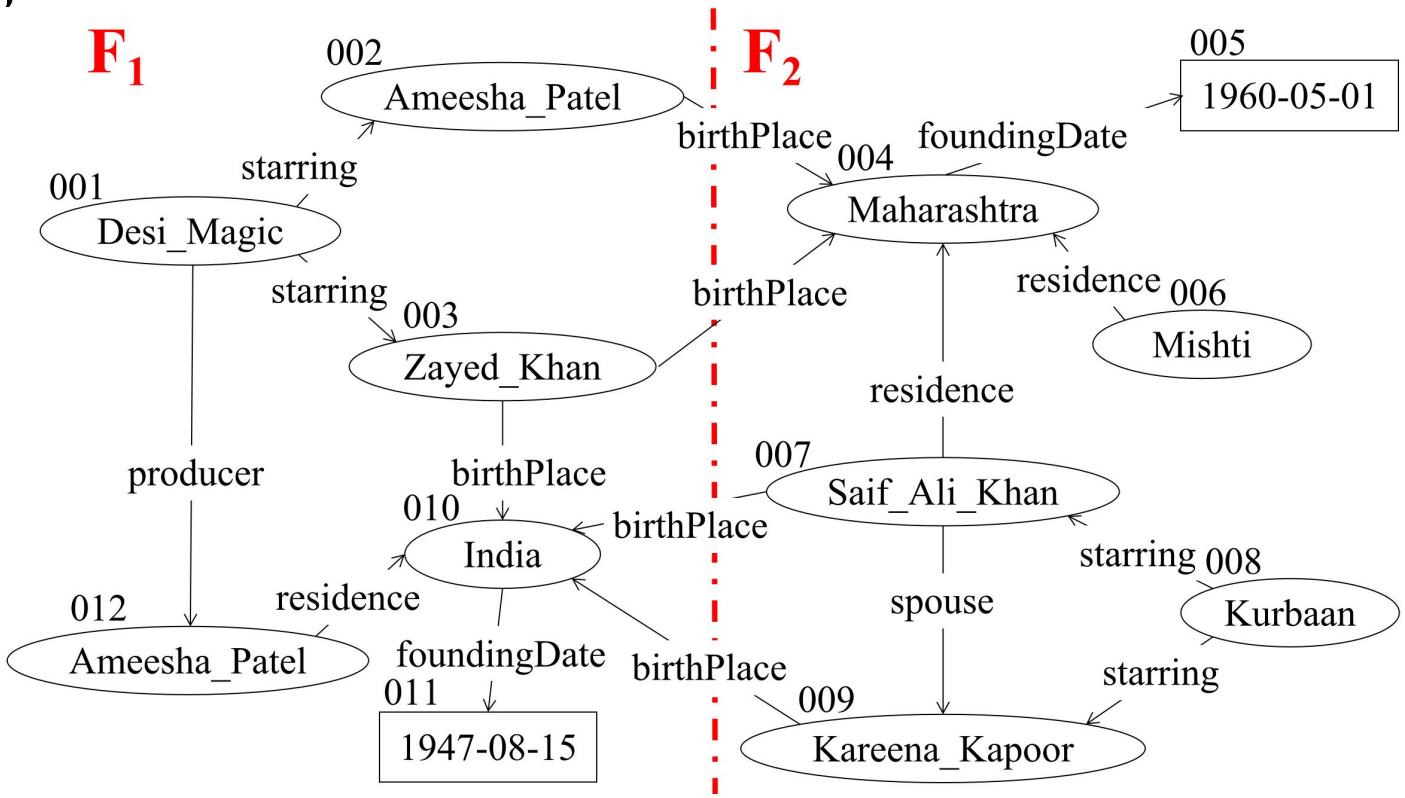
Let L be the set of all properties in graph G. The set of properties that only exist on the internal edges are called *internal properties*, denoted as L_{in}. Meanwhile, L-L_{in} is called the set of *crossing properties*.

Only edges with a crossing property can be crossing edges, and all edges with any internal property CANNOT be crossing edges.

Example L_{in}

Assume L_{in}={starring, producer, foundingDate, spouse, residence}, then

L-L_{in}={birthPlace}



Problem Transformations

Minimize crossing properties' number

⇓ Definitions of crossing properties

Maximize internal properties' number

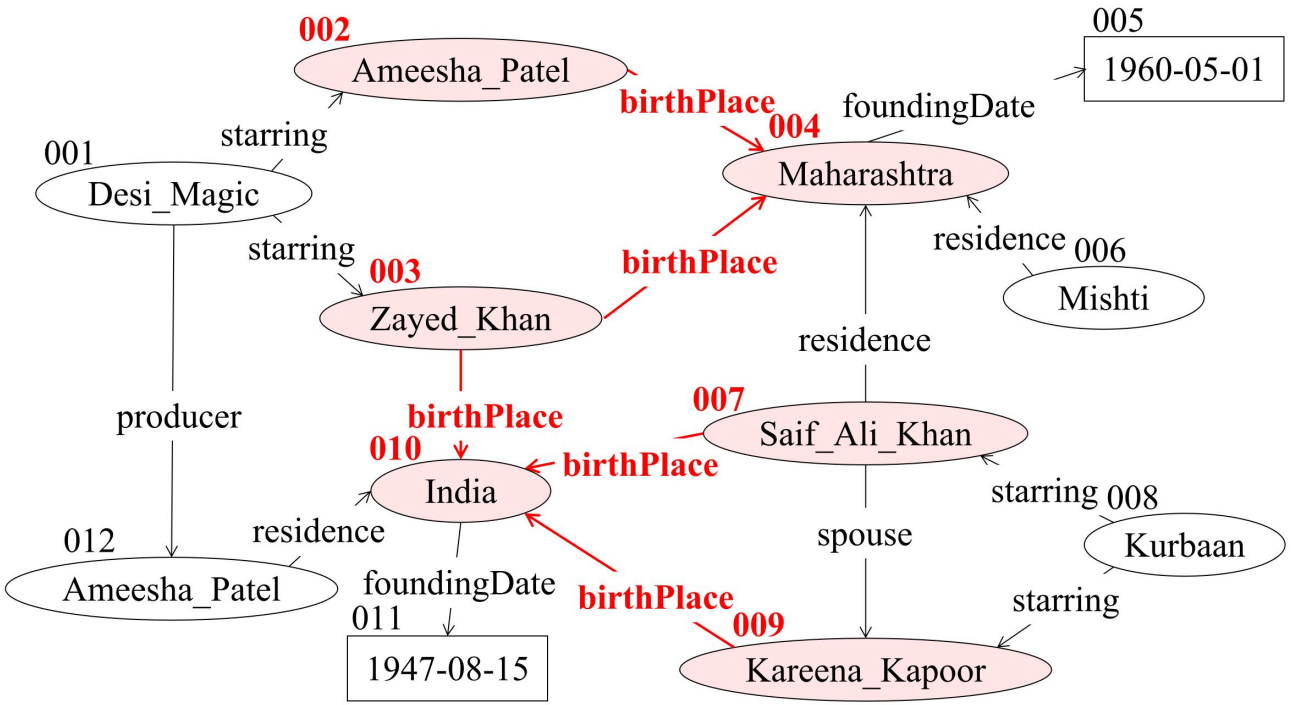
⇓

Select the largest set of properties as internal properties while its cost cannot be larger than $|V|/k$

Coarsen each weakly connected component of $G[L_{in}]$ into one supervertex

Constraint of Internal Properties

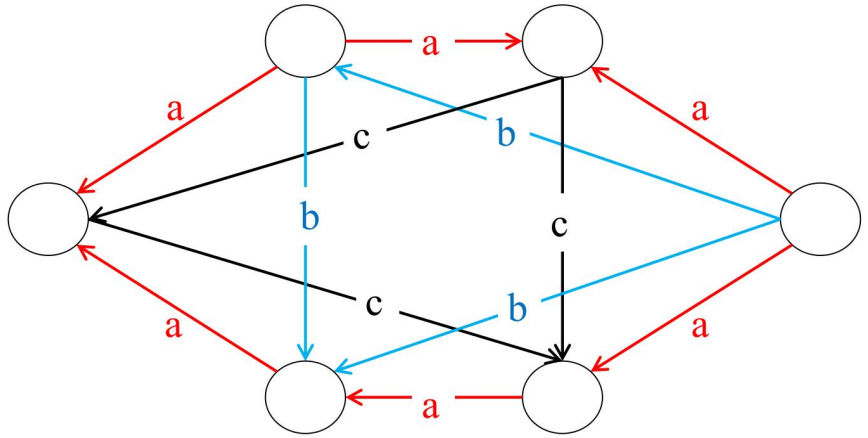
If a property p is an internal property, any two vertices in a weakly connected component of p's induced graph should be in one partition.



If we want *birthPlace* to be an internal property, all the six vertices should be in one partition

Extreme Case

For balance, no matter how frequently the property “a” is queried or how large weight the property “a” is assigned, it cannot be selected as internal property



When selecting internal properties, the size of largest weakly connected component in their induced graphs is the key factor

Cost of Selecting Properties as Internal

Given a property, we can use the size of the largest weakly connected component in its induced graphs to measure the cost of selecting it as internal property

Given a set of properties, we can use the size of the largest weakly connected component in their induced graphs to measure the cost of selecting them as internal properties

Problem Defintions

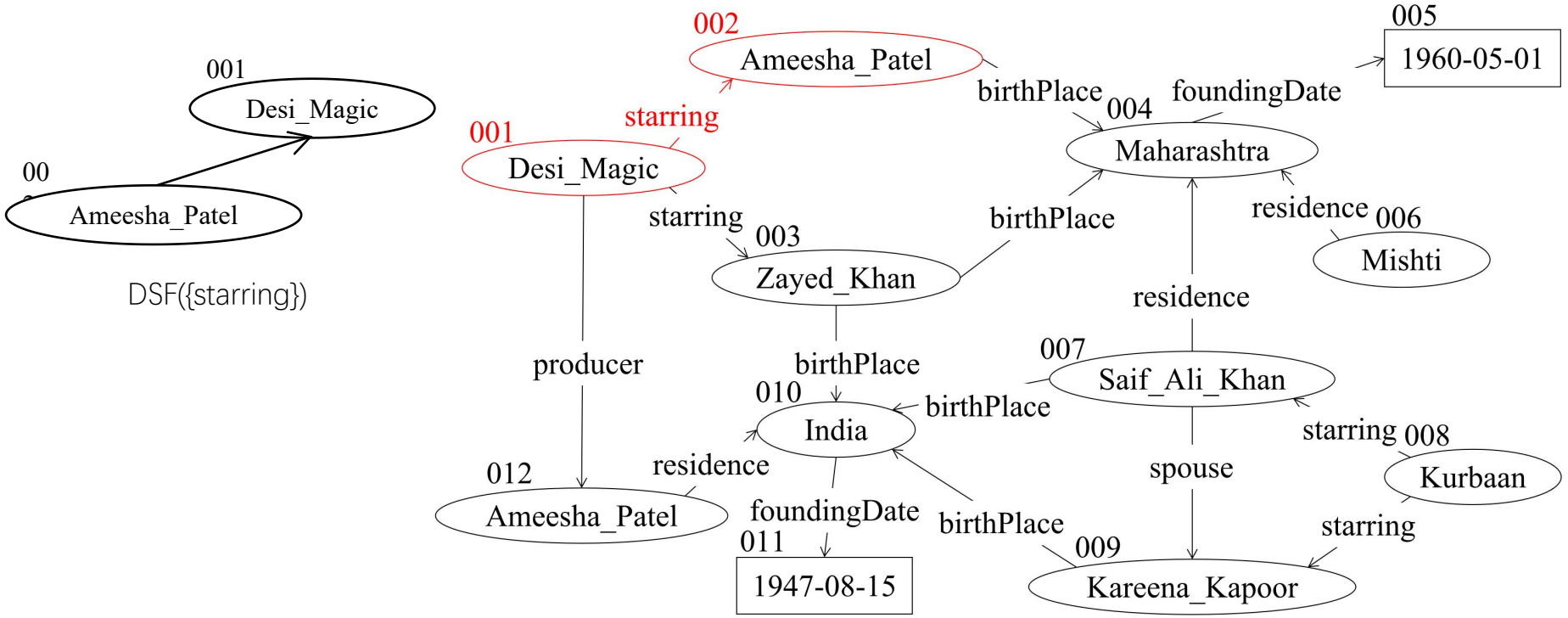
Selecting the maximal set of internal properties is equivalent to selecting the maximal set of internal properties while its cost cannot be larger than |V|/k

This is a knapsack problem

Each property maps to a item, its cost is the weight and |V|/k is the weight capacity in the knapsack problem.

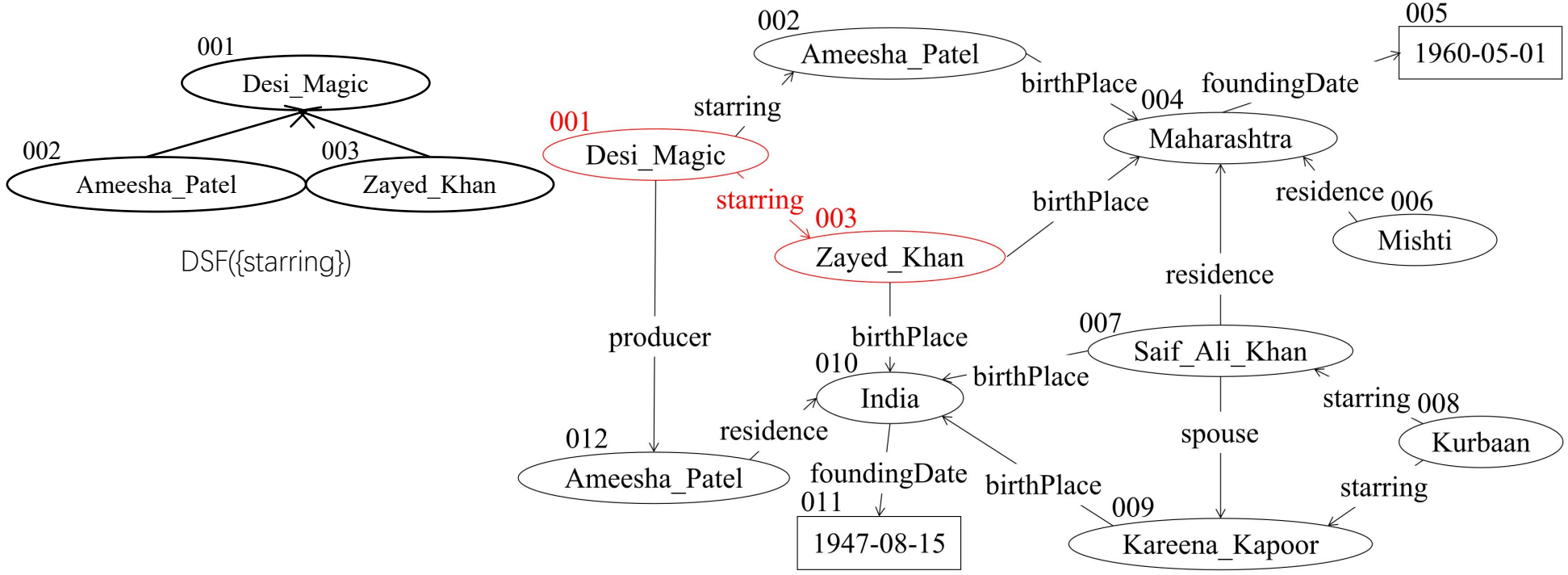
Example

For edge <001,002>, we find the subsets of “starring” and union them by making 001 as the parent of 002 for starring



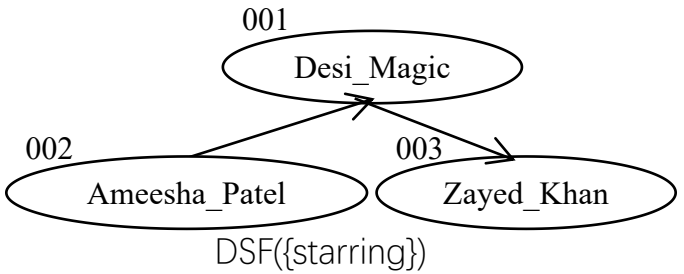
Example

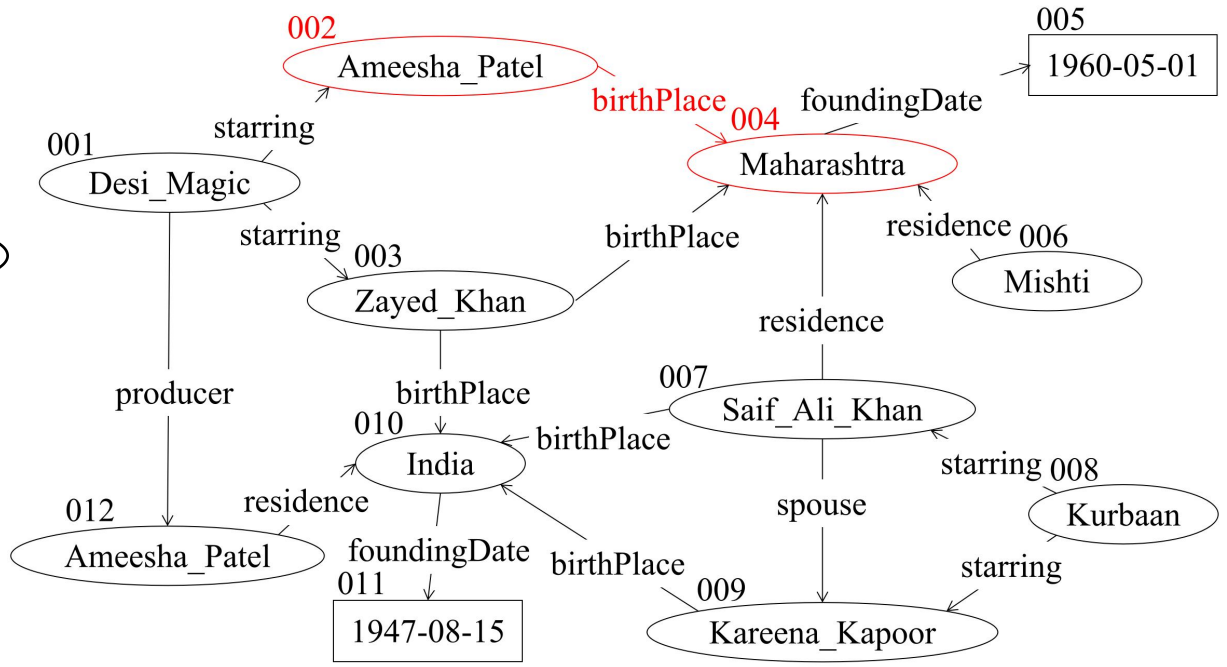
For edge <001,003>, we find the subsets of “starring” and union them by making 001 as the parent of 003 for starring



Example

For <002,004>, we find the subsets of “birthPlace” and union them by making 002 as the parent of 004 for birthPlace



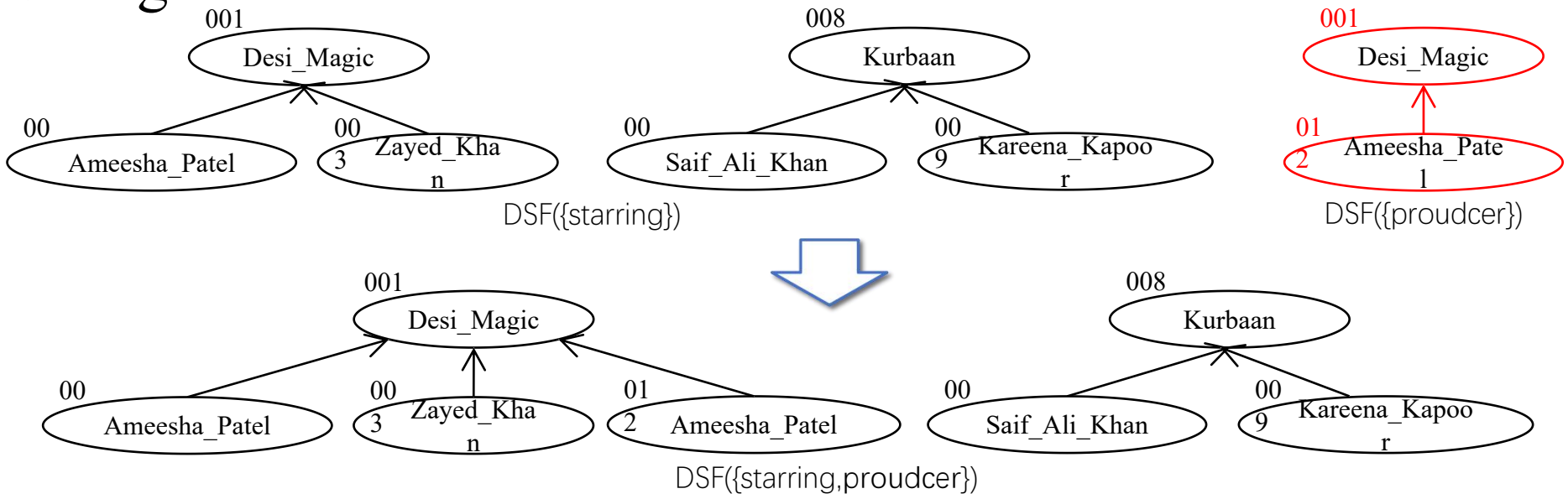


Computing the Cost of Multiple Properties

Two disjoint-set forests can join together by scanning one disjoint-set forest to union another one

Example

For 012 and its set representive 001 in DSF({proudcer), we update the subsets of “starring” and union them by making 001 as the parent of 012 for starring



湖南大學
HUNAN UNIVERSITY

HNU

谢谢观赏

实事求是　敢为人先