L17：最小生成树

吉林大学计算机学院
谷方明
fmgu2002@sina.com

学习目标

- 掌握（自由）树的概念、性质和等价定义
- 掌握最小生成树的定义
- 掌握求解最小生成树问题的Prim算法和Kruskal算法

自由树

- 自由树是一个连通无环图。
  - 无向图
  - 图论通常省略“自由”，数据结构要根据语境区分
- 有根树是一棵自由树，有一个特别的称为根的结点。
  - 无根树
- 自由树转换成有根树
  - 定根
  - 定向（边：内向、外向）

树的等价性定义（图论）

令 G = (V,E) 是一个无向图，下面描述是等价的

1. G是自由树
2. G无环，且 | E | = | V | - 1
3. G连通，且 | E | = | V | - 1
4. G无环，但增加任意一条边后均有环
5. G连通，但移除任意一条边后均不连通
6. G中任意两点之间有唯一一条路

1=>2

- **G是自由树 =>G无环，且 | E | = | V | - 1**
- **证明: 数学归纳法（对结点数n）**
  - ✓ n=1时， e=0 ， 显然成立
  - ✓ 设n=k时成立， 当n=k+1时：

  无环，一定存在度为1的点u；

  设该边(u,v)，删掉该点及边，得到k个点的连通无环图，根据假设，其边数为k-1; 因此，原图边数为 (k-1) + 1 = k，结论成立。

2=>3

- **G无环且| E | = | V | - 1 => G连通且| E | = | V | - 1**

- **证明: 反证法**
  - ✓ 假设G不连通，设有k个连通分支，k>1。每个分支连通无环，$e_i = v_i-1$，则整个图有 $e = v-k$。与e=v-1矛盾

3=>4

- G连通且| E | = | V | - 1 => G无环，但增加任意一条边后均有环
- 数学归纳法（对结点数n）
  - ✓ n=2时，e=1，显然无环，增加一条边后有1个环；
  - ✓ 假设n<k成立，当n=k时：G连通，每个结点d(u)≥1。至少有一个结点u，d(u)=1。否则，2e≥2n,与 e=n-1矛盾；删掉u及其关联边得G’，利用归纳假设，G’无环，增加一条边均有环，补回u及其关联边，T也无环，任意增加一条边都会有1个环

4=>5:

- **G无环，但增加任意一条边后均有环 => G连通，但移除任意一条边后均不连通**

- 证明
  - ✓ 若G不连通，设u与v之间无路，增加边(u,v)不会产生环路，与假设矛盾。
  - ✓ 由于G中无环路，删除任意一条边，T就不连通
    。

5=>6

- **G连通，但移除任意一条边后均不连通 => G中任意两点之间有唯一一条路**

- **证明:**
  - ✓ 由连通性，任意两点之间都至少有一条路；
  - ✓ 如有两点间有多于一条路，则必有环路，删除该环路上任意一条边，图仍然连通，矛盾。

6=>1

- **G中任意两点之间有唯一一条路=>G是自由树**

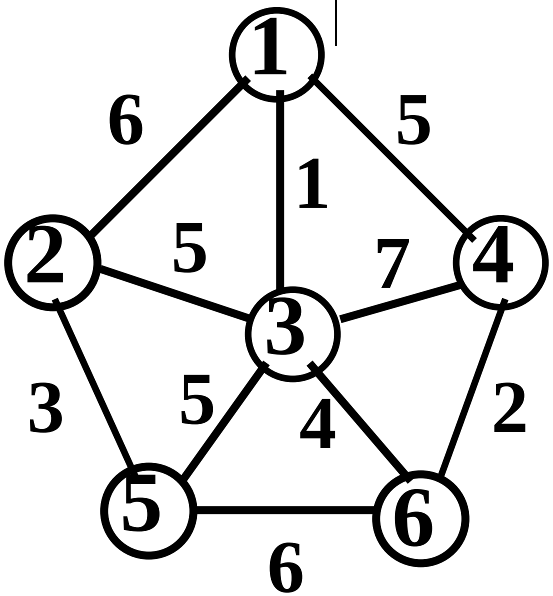
- **证明:**
  - ✓ 任意两点之间都有唯一一条路，则G必连通
  - ✓ 如有环路，则环路上任意两点都有两条路，矛盾，故G无环

树的性质（图论）

1. **n-1** 条边
2. 连通
3. 无环

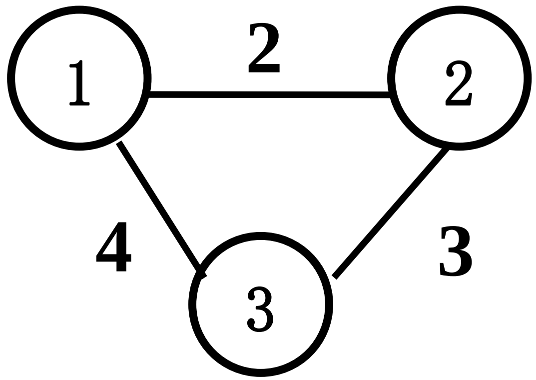
例题：网络工程规划

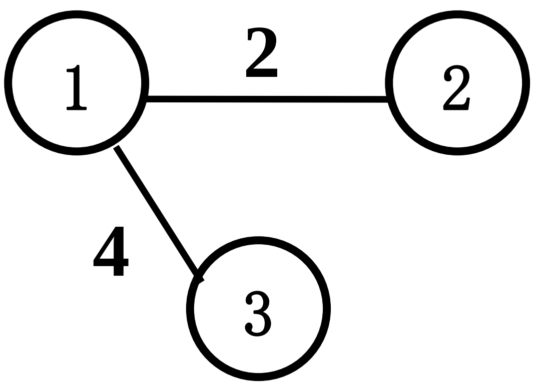
- 网络工程布线
  - ✓ 顶点表示服务器或基站
  - ✓ 边表示光纤或铜缆
  - ✓ 边上的权值表示铺线的代价



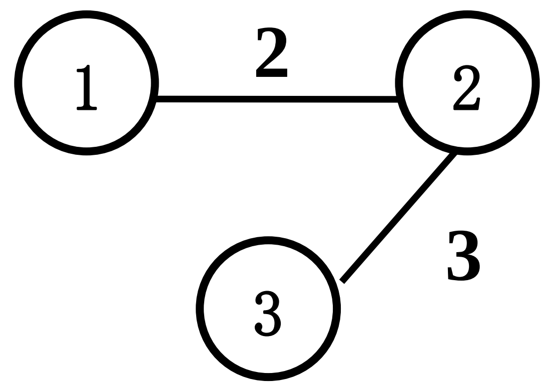
- 问题：构造任意两点都能通信且代价最小的网络。代价指边的权值和。

所求问题不是最短路问题





源为1的单源最短路



实际要求的问题

分析

- **问题是寻找包含全部n个顶点的连通子图，且代价最小。**
  - **n个顶点的子图：** 支撑子图或生成子图
  - **连通：** 连通n个顶点至少要n-1条边，代价最小要求只能是n-1条边；n-1条边的连通图就是树。
  - **代价最小：最小支撑（生成）树（Minimum Spanning Tree, MST）。**

最小生成（支撑）树

- 设连通无向网络**G=(*V*, *E*, *W*)**，***W*** 是 **G** 的边上权值集合。设**G** 的顶点数为***n***，若从***E***中选出 ***n*–1**条边, 满足:
  1. 这 *n*–1条边和 V 构成一个连通子图*G′* 。
  2. *G′*具有最小代价。

则称*G′*为网络*G*的<u>最小生成（支撑）树。</u>

求解策略

- 一种策略：逐步生成，每次确定**MST**的一条边
- 过程：管理边集**A**。**A**初始空，循环加**n-1**条边。
  - ✓ 增加边前，**A都是某棵MST的子集**（循环不变式）。
  - ✓ 增加边时，选择一条边(u,v)加入A中，使A不违反循环不变式，即**A∪ (u,v) 也是某棵MST的子集。**
- **(u,v)**加入不破坏循环不变式，称其为**A**的安全边
- 奥妙：安全边必然存在。关键：寻找安全边。

普里姆算法思想(Prim)

- **将顶点集 V 分成两组**
  - 一组为U，表示已在最小生成树的顶点集合；
  - 另一组为V-U，不在最小生成树的顶点集合；
- **每次从 V- U中，选择一个顶点v，放入U中，通过加入边(u,v)，(u,v)满足 weight(u,v)＝min{weight(u_1,v_1)|u_1∈U, v_1∈V-U}**
  - 标记法；

Prim算法描述（自然语言）

设G=(V,E,W)为连通网，TE是G的最小生成树MST的边的集合，U为MST顶点集。

① 初始化：$U=\{u_0\}$ ($u_0 \in V$)，TE= Φ ；

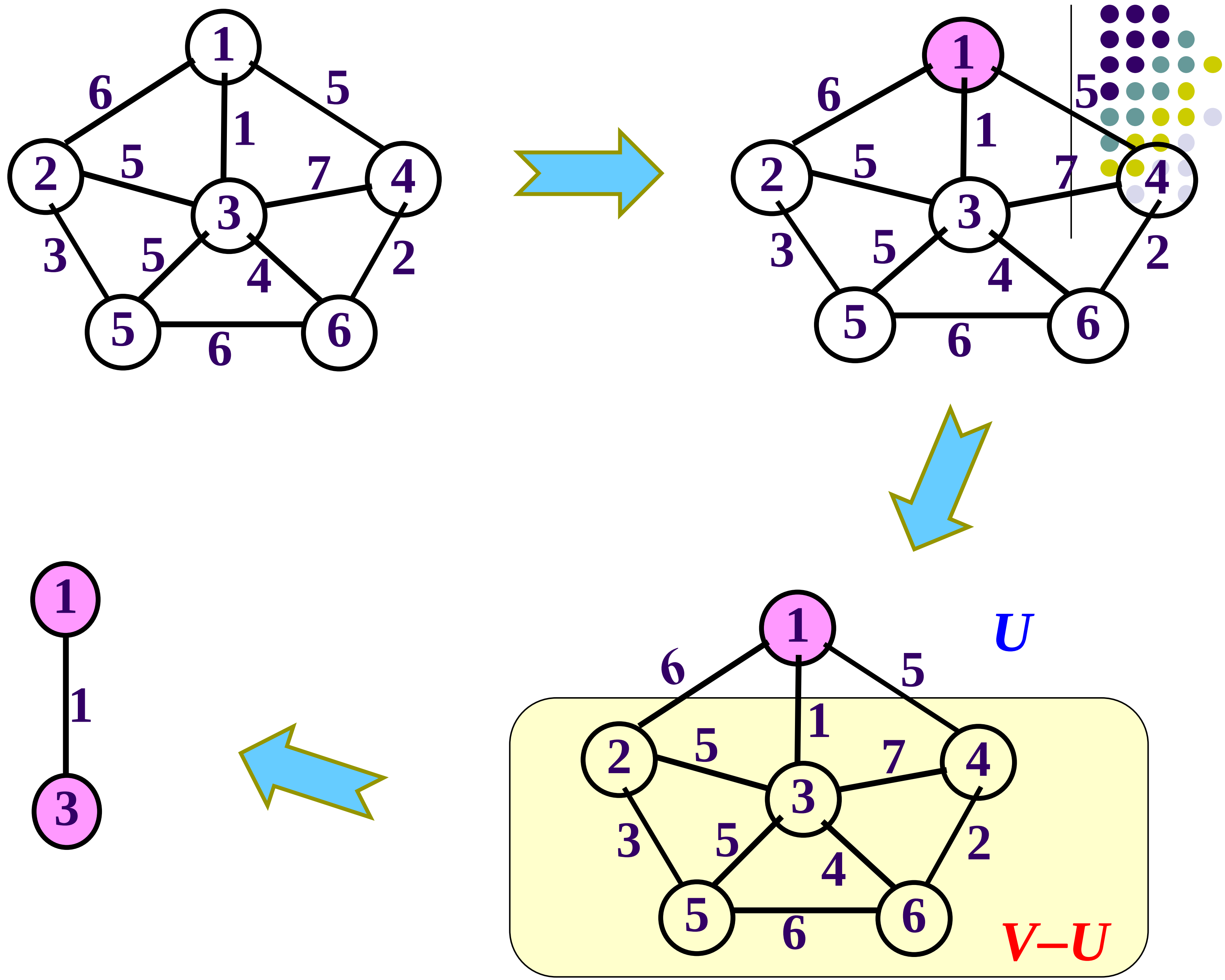
② 找到满足

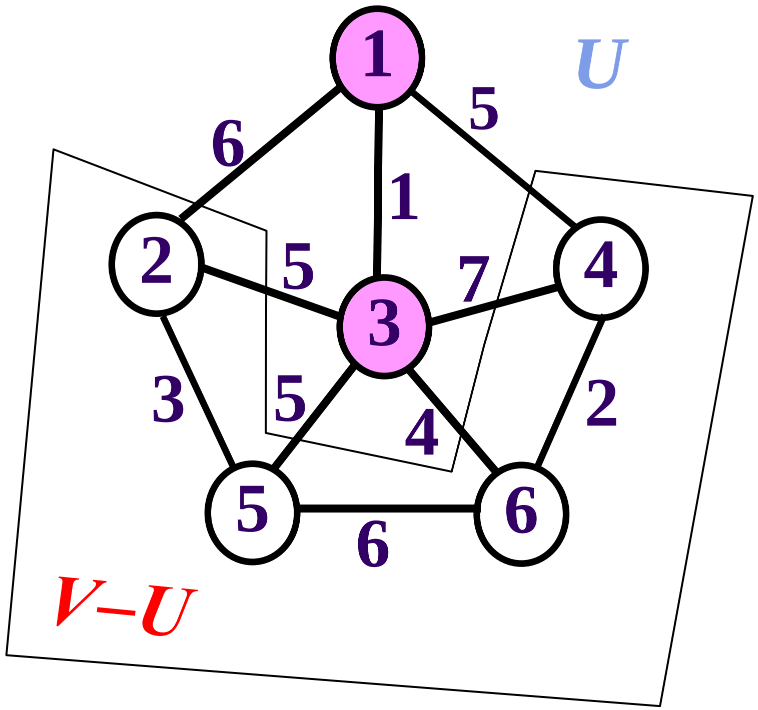
$weight(u,v)=min\{weight(u_1,v_1)|u_1 \in U, v_1 \in V-U\}$,

的边，把它并入TE，同时v并入U;

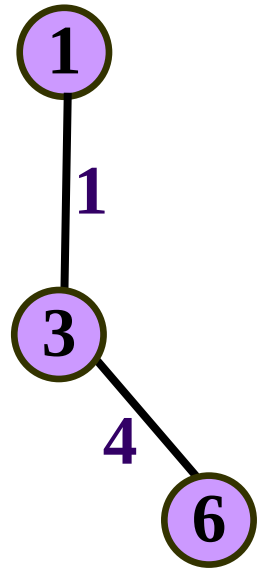
③ 反复执行② ,直至 V=U , 算法结束。

[例]

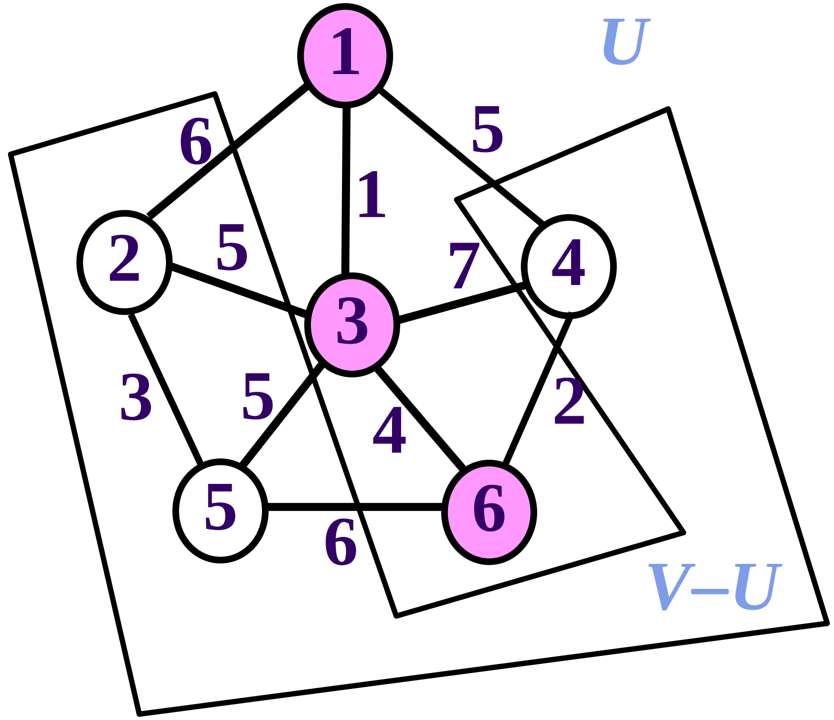


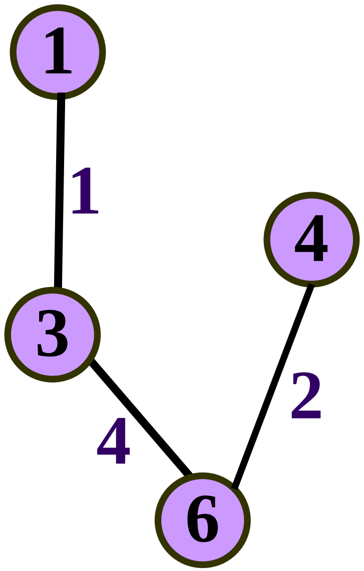


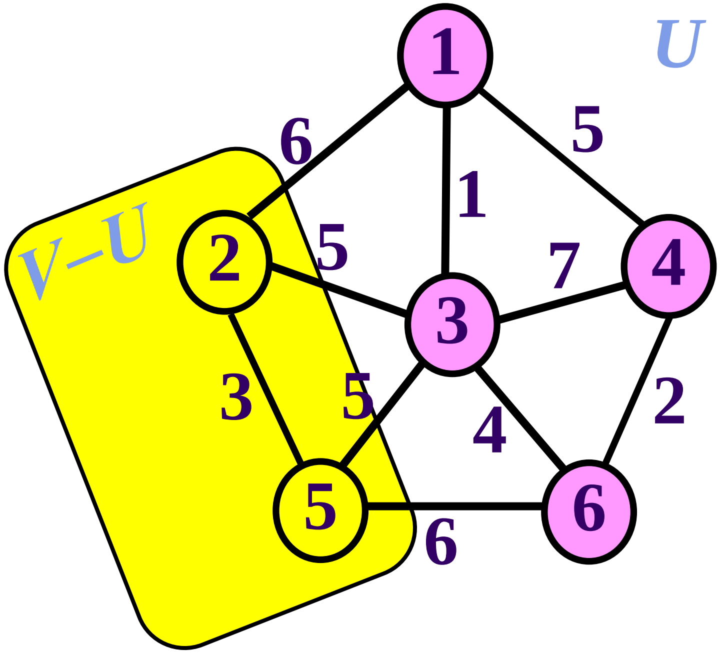
1
2
3
4
5
6
U
V–U
6
5
1
5
7
3
5
4
2
6

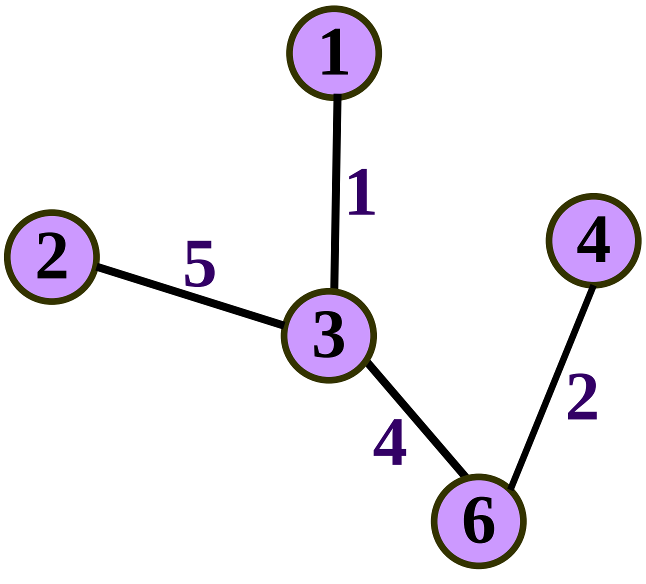


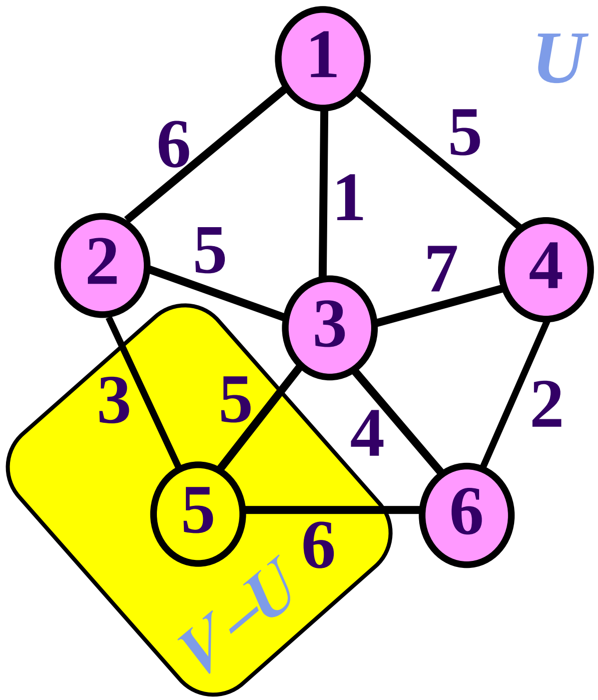
1
3
6
1
4

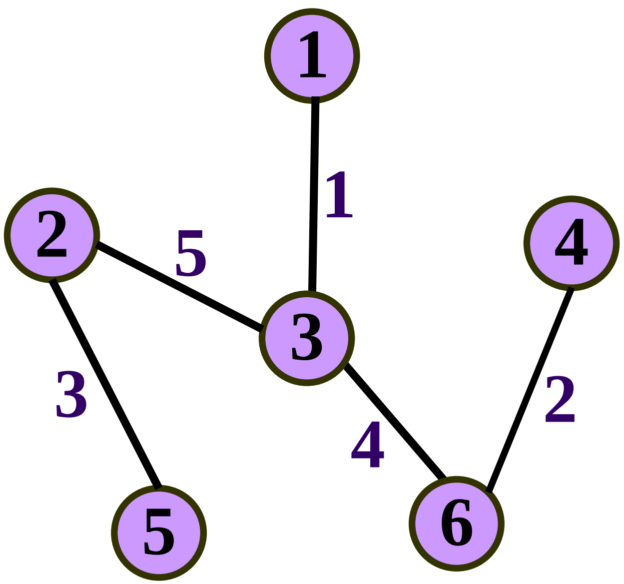

U
V–U
1
2
3
4
5
6
6
1
5
5
7
3
5
4
2
6


1
3
4
6
1
4
2


1
U
6
5
1
V–U
2
5
7
4
3
3
5
2
4
5
6
6


1
1
4
2
5
3
2
4
6

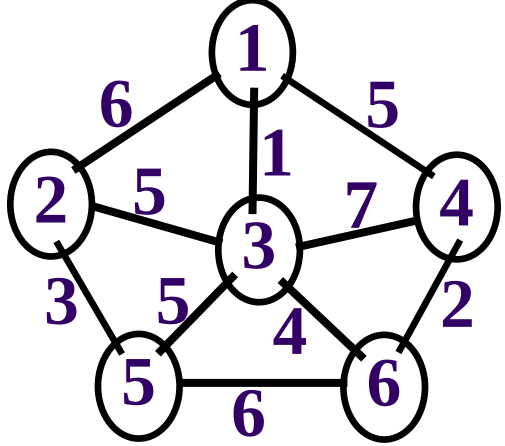

U
1
2
3
4
5
6
6
5
1
5
7
3
5
4
2
6
V–U


1
2
3
4
5
6
1
5
3
4
2

算法设计

- 图用邻接矩阵或邻接链表存储
- 跨集合边的最小值
  - ✓ 标记数组mark；
  - ✓ Lowcost[v] = $\min\{weight(u, v) | u \in U\}$
- 树边的存储
  - ✓ 法一：Vex[v] = u，$\min\{weight(u, v) | u \in U\}$的顶点；
  - ✓ 法二：引入存放在TE[n-1]中；*TE*[*i*]表示一条边, 由 *head*、*tail* 和 *cost* 三个域构成, 分别存放边的始点、终点和权值.

例



	1	2	3	4	5	6
lowcost	0	6	1	5	max	max
vex	-1	1	1	1	1	1
mark	1	0	0	0	0	0

	1	2	3	4	5	6
lowcost	0	5	1	5	5	4
vex	-1	3	1	1	3	3
mark	1	0	1	0	0	0

Prim算法描述（ADL）

算法Prim(head,n. vex)/* 连通图G使用邻接链表存储*/

Prim1[初始化]

```
 for( i =1 ; i <= n ; i ++) {
     lowcost[i] = max;
     mark[i] = 0;  // mark[ i ] 记录 i 是否已在MST
      vex[i] = -1;
 }
 mark[1] = 1; lowcost[1] = 0 ;
 for( p =head[1].adjacent ; p; p=p->link){
          k = p->VerAdj;
          lowcost[k] = p->cost; vex[k] = 1;
  }
```

Prim2[构造MST]

```
for( j =1 ; j < n ; j ++){
    minEW = INF; //确定轻边
    for( i =1 ; i <= n ; i ++)
        if (mark[i]==0 && lowcost[i] < minEW ) {
            minEW = lowcost[i] ;  u = i ;}
    mark[u] = 1;
    for( p = Head[u].adjacent ; p ; p = p->link ){
        v =  p -> VerAdj ;
        if (mark[v] == 0 && p->cost < lowcost[v] ){
            lowcost[v] = p->cost ;  vex[v] = u; }
        }//若vex存MST，必须判mark[v]防止污染
}
```

Prim算法的正确性——相关术语

- 无向图G(V,E)的一个切割(S,V-S)是V的一个划分
- 如果一条边(u,v)∈E的一个端点在S中，另一个端点在V-S中，称该边横跨切割(S,V-S)

- 若A中不存在横跨切割(S,V-S)的边，则称该切割尊重A
- 在横跨一个切割的所有边中，权重最小的边称为轻边。

Prim算法的正确性——定理

- 定理：设G=(V,E)是一个连通无向图，W定义了边上权值。设集合A是E的一个子集，且A包含在G的某棵最小生成树中。设(S,V-S)是G中尊重A的任意一个切割，(u,v)是横跨(S,V-S)的一条轻边，(u,v)对于A是安全的。

证明

- 设T是一棵包含A的最小生成树。假设T不包含轻边(u,v)，否则已证明完毕。
- 剪切粘贴：轻边(u,v)与T中从u到v的简单路径p形成一个环路。由于u、v分别处于切割(S,V-S)的两端，T中至少有一条边属于p且横跨该切割。设(x,y)是这样一条边，因为切割(S,V-S)尊重A，边(x,y)位于T中从u到v的唯一简单路径p上，删除(x,y)将导致T被分解到两个连通分量。将轻边(u,v)加上去，可将这两个连通分量连接成一棵新的生成树 T' = T- {(x,y)} ∪ {(u,v)}.

- **T’ 是一棵MST. 由于(u,v)是横跨切割(S,V-S)的一条轻边，且(x,y)也横跨切割(S,V-S), 则有W (u,v)≤ W (x,y)。因此， W (T’)≤ W (T)。由于T是MST，故T’也一定是MST。**

- **(u,v)对于A是安全的。因为 A $\subseteq$ T且(x,y)不属于A，所以A $\subseteq$ T’。因此，A $\cup$ {(u,v)} $\subseteq$ T’ 。 T’是MST，所以(u,v)对于集合A是安全的**

Prim算法小结

- 普里姆算法的时间复杂度为$O(n^2)$
- 使用堆优化，可达到O(e * logn)

- 适用于求边稠密网的最小生成树。

- 不要和Dijkstra算法弄混
  - 两者都是标记法，算法描述相似

克鲁斯卡尔(Kruskal)算法思想

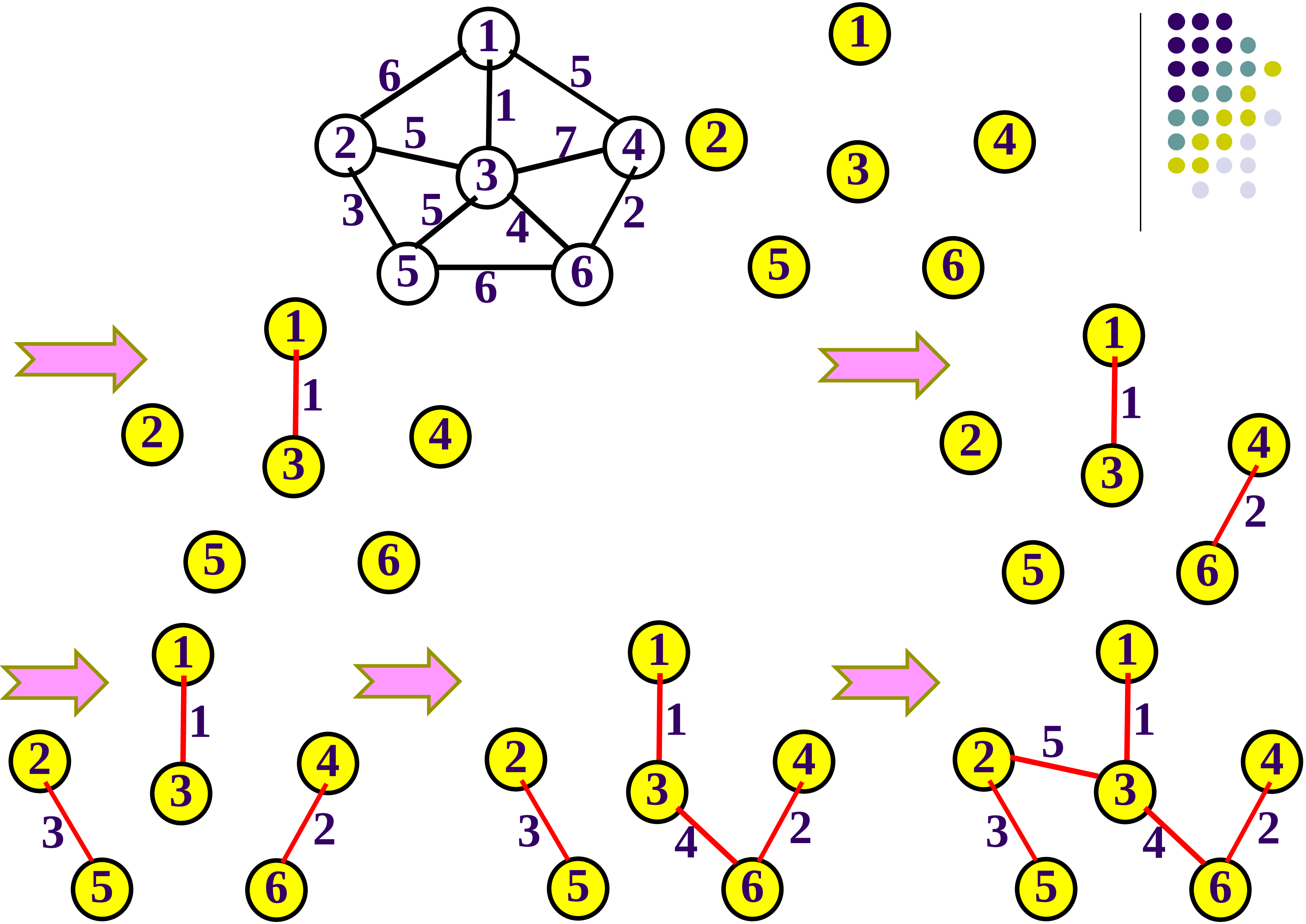
设连通网G=(V,E,W)，T为N的最小生成树。

初始时T={V,Φ}，即T中没有边，只有n个顶点

①在E中选择权值最小的边.

②如果将此边加入T中，不形成环，就加入；否则，不加入；

重复执行① ②，直至选够 n-1 条边。



Kruskal算法的正确性

- 定义： 图G = (V,E)的生成森林是各个不相交生成树$T_i=(V_i,E_i)$的集合，其中 $\cup V_i = V$, $\cup E_i \leqslant E$

- 定理：设G=(V,E)是一个连通无向图，W定义了边上权值。设集合A是E的一个子集，且A包含在G的某棵最小生成树中。设$C=(V_C,E_C)$为森林$G_A(V,A)$中的一个连通分量（生成树）。如果(u,v)是连接C和G_A中其它某个连通分量的一条轻边，则(u,v)对于集合A是安全的。

- 证明：切割$(V_C, V-V_C)$尊重集合A，(u,v)是横跨该切割的一条轻边，由前面定理可知，(u,v)对于集合A是安全的。

实现有多种方式

- 边取最小
  - ✓ 堆
  - ✓ 有序边表（权值递增）
- 判环（无向图）
  - ✓ 使用并查集判环
  - ✓ 遍历判环
  - ✓ ……
- 图的存储结构选取
  - ✓ 邻接表
  - ✓ 边表

Kruskal算法描述 (有序边表+并查集，ADL)

算法Kruskal(E, n,m. TE)/* 连通图G用边表E(u,v,w)存储*/

Kruskal1[初始化]

```
    sort(E，less). //按w递增排序，less：元素比较函数
    for( i =1 ; i <= n ; i ++) MAKE_SET(i)
```

Kruskal2[构造MST]

```
    for( i =1, j = 0  ; i <=m ; i ++) {
        u= E[i].u; v= E[i].v
        if( FIND(u) != FIND(v)){
            TE[++j] = E[i];  UNION(u, v); }
        if (j == n-1) break;
}
```

Kruskal算法时间效率分析

- 边表+并查集
  - 排序：O(eloge)
  - 判环：O(e*α(e))
  - 整体时间复杂度O(eloge)

- 堆实现
  - 取最小 O(eloge)

- 克鲁斯卡尔算法适用求边稀疏网的最小生成树;

小结

- **最小生成树基于无向图（连通）**

- **构造MST的算法**
  - ✓ 避圈法：增加权值最小的边，如Prim算法、Kruskal算法、Solin算法等
  - ✓ 破圈法：删除圈中权值最大的边，如管梅谷算法等

课后思考：

- 最大生成树？
  - ✓ 边权取相反数，利用最小生成树方法
  - ✓ 仿照MST，修改最小生成树方法

- 次小生成树？
- 指定点度为k的最小生成树？
- 最小生成树计数？