算法基础

主讲人： 庄连生

Email: { *lszhuang@ustc.edu.cn* }

***Spring 2019*，USTC**

算法设计复习

内容提要：

- 算法设计思想回顾（递归和分治、动态规划、贪心算法、回溯法、分支限界法、随机算法）
- 经典例子讲解

算法设计策略

已学过的算法设计策略：

- ✓ 递归和分治
- ✓ 动态规划
- ✓ 贪心算法
- ✓ 回溯法
- ✓ 分支限界法
- ✓ 随机算法

Sch1-4 分治法

- 基本思想：把一个规模大的问题划分为规模较小的子问题，然后分而治之，最后合并子问题的解得到原问题的解。
- 步骤：
  ① 分割原问题；
  ② 求解子问题；
  ③ 合并子问题的解为原问题的解。
- 在分治法中，子问题一般是相互独立的，因此，经常通过递归调用算法来求解子问题。

Sch1-4 分治法

分治法所能解决的问题一般具有以下几个特征：

① 该问题的规模缩小到一定的程度就可以容易地解决；

② 该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质

③ 利用该问题分解出的子问题的解可以合并为该问题的解；

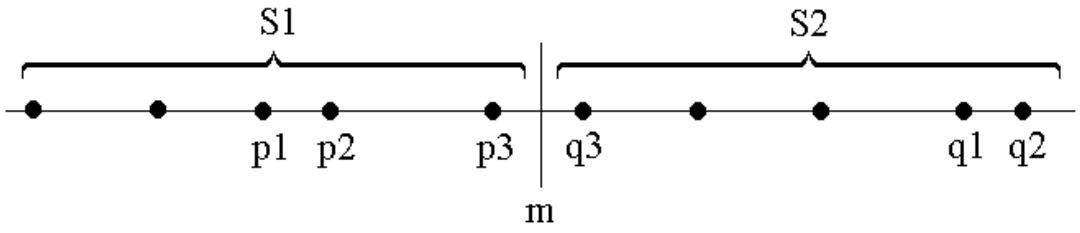
④ 该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子问题。

这条特征涉及到分治法的效率，如果各子问题是不独立的，则分治法要做许多不必要的工作，重复地解公共的子问题，此时虽然也可用分治法，但一般用动态规划较好。

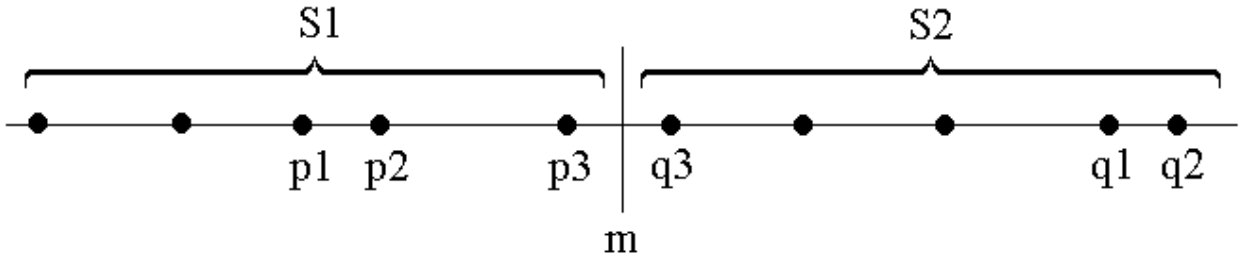
Sch1-4 分治法

例1：最近点对问题

- 为了使问题易于理解和分析，先来考虑一维的情形。此时，S中的n个点退化为x轴上的n个实数x1,x2,…,xn。最接近点对即为这n个实数中相差最小的2个实数。
- 假设我们用x轴上某个点m将S划分为2个子集S1和S2，基于平衡子问题的思想，用S中各点坐标的中位数来作分割点。
- 递归地在S1和S2上找出其最接近点对{p1,p2}和{q1,q2}，并设d=min{|p1-p2|,|q1-q2|}，S中的最接近点对或者是{p1,p2}，或者是{q1,q2}，或者是某个{p3,q3}，其中p3∈S1且q3∈S2。
- 能否在线性时间内找到p3,q3？



Sch1-4 分治法

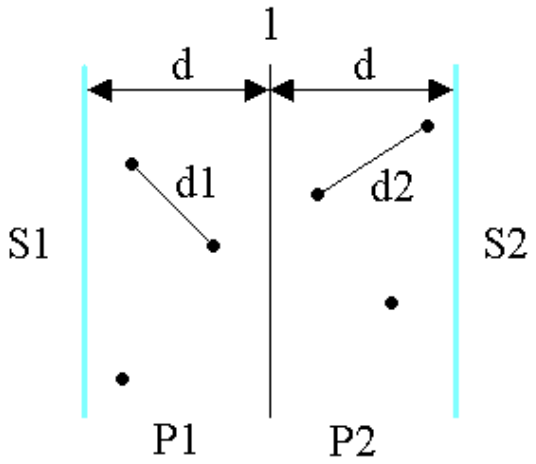


如果S的最接近点对是{p3,q3}，即|p3-q3|<d，则p3和q3两者与m的距离不超过d，即p3∈(m-d,m]，q3∈(m,m+d]。

- 由于在S1中，每个长度为d的半闭区间至多包含一个点（否则必有两点距离小于d），并且m是S1和S2的分割点，因此(m-d,m]中至多包含S中的一个点。由图可以看出，如果(m-d,m]中有S中的点，则此点就是S1中最大点。
- 因此，我们用线性时间就能找到区间(m-d,m]和(m,m+d]中所有点，即p3和q3。从而我们用线性时间就可以将S1的解和S2的解合并成为S的解。

Sch1-4 分治法

- 选取一垂直线l:x=m来作为分割直线。其中m为S中各点x坐标的中位数。由此将S分割为S1和S2。
- 递归地在S1和S2上找出其最小距离d1和d2，并设d=min{d1,d2}，S中的最接近点对或者是d，或者是某个{p,q}，其中p∈P1且q∈P2。
- 能否在线性时间内找到p,q?

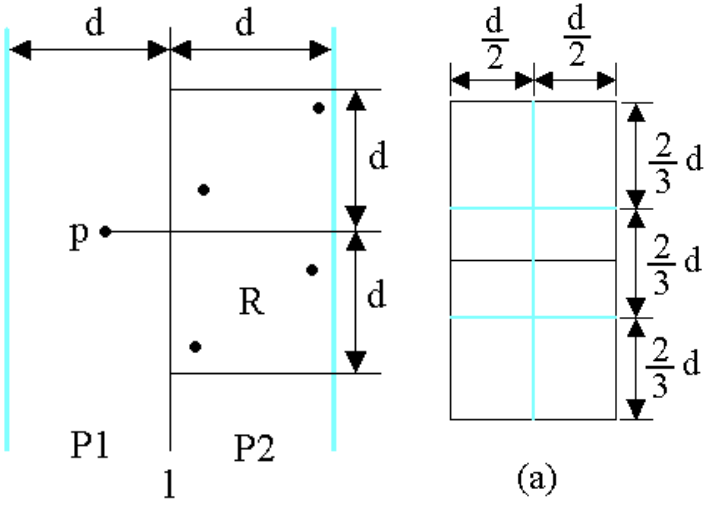


Sch1-4 分治法

能否在线性时间内找到p,q?

考虑P1中任意一点p，它若与P2中的点q构成最接近点对的候选者，则必有distance(p, q) < d。**满足这个条件的P2中的点一定落在一个d×2d的矩形R中**

- 由d的意义可知，P2中任何2个S中的点的距离都不小于d。由此可以推出**矩形R中最多只有6个S中的点**。
- 因此，**在分治法的合并步骤中最多只需要检查6×n/2=3n个候选者**



(a)

(b)

证明:将矩形R的长为2d的边3等分，将它的长为d的边2等分，由此导出6个(d/2)×(2d/3)的矩形。若矩形R中有多于6个S中的点，则由鸽舍原理易知至少有一个(d/2)×(2d/3)的小矩形中有2个以上S中的点。设u，v是位于同一小矩形中的2个点，则distance(u,v)<d。这与d的意义相矛盾。

Sch1-4 分治法

- 为了确切地知道要检查哪6个点，可以将p和P2中所有S2的点投影到垂直线l上。由于能与p点一起构成最接近点对候选者的S2中点一定在矩形R中，所以它们在直线l上的投影点距p在l上投影点的距离小于d。由上面的分析可知，这种投影点最多只有6个。
- 因此，若将P1和P2中所有S中点按其y坐标排好序，则对P1中所有点，对排好序的点列作一次扫描，就可以找出所有最接近点对的候选者。对P1中每一点最多只要检查P2中排好序的相继6个点。

Sch1-4 分治法

```
double cpair2(S)
{
    n=|S|;
    if (n < 2) return    ;
1、m=S中各点x间坐标的中位数;
    构造S1和S2;
    //S1={p ∈ S|x(p)<=m},
    S2={p ∈ S|x(p)>m}
2、d1=cpair2(S1);
    d2=cpair2(S2);
3、dm=min(d1,d2);
```

4、设P1是S1中距垂直分割线l的距离在dm之内的所有点组成的集合；
P2是S2中距分割线l的距离在dm之内所有点组成的集合；
将P1和P2中点依其y坐标值排序；
并设X和Y是相应的已排好序的点列；
5、通过扫描X以及对于X中每个点检查Y中与其距离在dm之内的所有点(最多6个)可以完成合并；
当X中的扫描指针逐次向上移动时，Y中的扫描指针可在宽为2dm的区间内移动；
设dl是按这种扫描方式找到的点对间的最小距离；

```
6、d=min(dm,dl);
    return d;
}
```

时间复杂度：T(n)=O(nlogn)

$$T(n)=\begin{cases} O(1) & n<4 \\ 2T(n/2)+O(n) & n\geq 4 \end{cases}$$

Sch1-4 动态规划

基本思想：

将待求解问题分解成若干个子问题,但是经分解得到的子问题往往不是互相独立的。不同子问题的数目常常只有多项式量级。在用分治法求解时，有些子问题被重复计算了许多次。通过保存已解决的子问题的答案，而在需要时再找出已求得的答案，就可以避免大量重复计算，从而得到多项式时间算法。

基本步骤：

① 找出最优解的性质，并刻划其结构特征。
② 递归地定义最优值。
③ 以自底向上的方式计算出最优值。
④ 根据计算最优值时得到的信息，构造最优解。

Sch1-4 动态规划

- 基本要素：

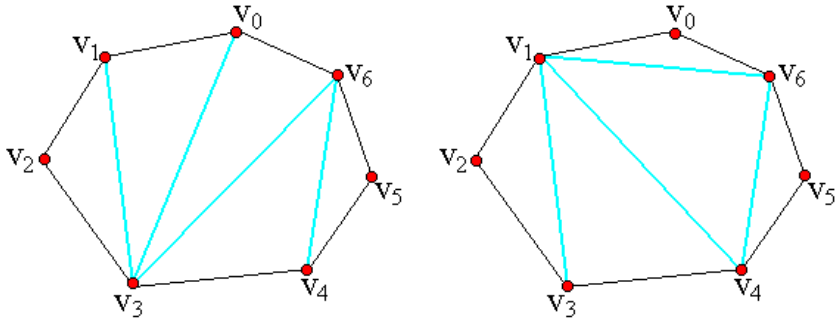
① 最优子结构性质：原问题的最优解包含着子问题的最优解，可以通过反证法来证明问题具有最优子结构性质。

② 重叠子问题：递归算法求解问题时，每次产生的子问题并不总是新问题，有些子问题被反复计算多次。对每一个子问题只解一次，而后将其解保存在一个表格中，当再次需要解此子问题时，只是简单地用常数时间查看一下结果。

- 问题的关键在于构造子问题空间。一个经验性规则就是，尽量保持这个空间简单，然后在需要时再扩充它。

例子2：凸多边形最优三角剖分问题

- 用多边形顶点的逆时针序列表示凸多边形，即$P=\{v_0,v_1,\cdots,v_{n-1}\}$表示具有n条边的凸多边形。
- 若v_i与v_j是多边形上不相邻的2个顶点，则线段v_iv_j称为多边形的一条弦。弦将多边形分割成2个多边形$\{v_i,v_{i+1},\cdots,v_j\}$和$\{v_j,v_{j+1},\cdots v_i\}$。
- 多边形的三角剖分是将多边形分割成互不相交的三角形的弦的集合T。



- 在凸多边形P的一个三角剖分T中，各弦互不相交，且弦数已达到最大，即P的任一不在T中的弦必与T中某一弦相交。在一个有n个顶点的凸多边形的三角剖分中，恰好有n-3条弦和n-2个三角形。

Sch1-4 动态规划

- 凸多边形最优三角剖分的问题是：给定凸多边形P，以及定义在由多边形的边和弦组成的三角形上的权函数w。要求确定该凸多边形的三角剖分，使得即该三角剖分中诸三角形上权之和为最小。

可以定义三角形上各种各样的权函数ω，如定义：

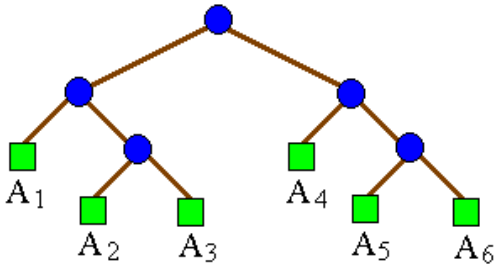
$$\omega(v_i v_j v_k)=|v_i v_j|+|v_i v_k|+|v_k v_j|$$

其中，$|v_i v_j|$是点v_i到v_j的欧氏距离。相应于此权函数的最优三角剖分即为最小弦长三角剖分

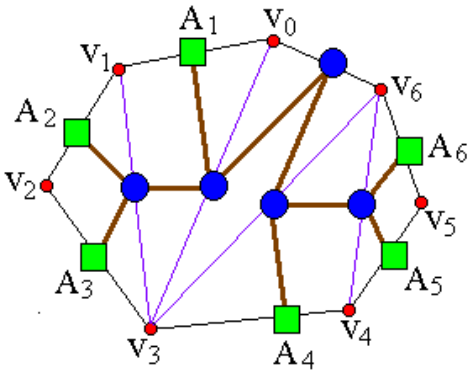
Sch1-4动态规划

三角形剖分的结构及其相关问题

- 一个表达式的完全加括号方式相应于一棵完全二叉树，称为表达式的语法树。例如，完全加括号的矩阵连乘积 $((A_1(A_2A_3))(A_4(A_5A_6)))$ 所相应的语法树如图 (a)所示。
- 凸多边形$\{v_0,v_1,\cdots v_{n-1}\}$的三角剖分也可以用语法树表示。例如，图 (b)中凸多边形的三角剖分可用图 (a)所示的语法树表示。
- 矩阵连乘积中的每个矩阵A_i对应于凸(n+1)边形中的一条边$v_{i-1}v_i$。三角剖分中的一条弦v_iv_j，i<j，对应于矩阵连乘积A[i+1:j]。



(a)



(b)

Sch1-4 动态规划法

最优子结构性质：

- 凸多边形的最优三角剖分问题有最优子结构性质。
- 证明：事实上，若凸(n+1)边形$P=\{v_0,v_1,\cdots,v_{n-1}\}$的最优三角剖分T包含三角形$v_0v_kv_n$，$1\leqslant k\leqslant n-1$，则T的权为3个部分权的和：三角形$v_0v_kv_n$的权，子多边形$\{v_0,v_1,\cdots,v_k\}$和$\{v_k,v_{k+1},\cdots,v_n\}$的权之和。可以断言，由T所确定的这2个子多边形的三角剖分也是最优的。因为若有$\{v_0,v_1,\cdots,v_k\}$或$\{v_k,v_{k+1},\cdots,v_n\}$的更小权的三角剖分将导致T不是最优三角剖分的矛盾。

Sch1-4 动态规划法

递归结构：

- 定义t[i][j]，1≤i<j≤n为凸子多边形$\{v_{i-1},v_i,\cdots,v_j\}$的最优三角剖分所对应的权函数值，即其最优值。为方便起见，设退化的多边形$\{v_{i-1},v_i\}$具有权值0。据此定义，要计算的凸(n+1)边形P的最优权值为t[1][n]。
- t[i][j]的值可以利用最优子结构性质递归地计算。当j-i≥1时，凸子多边形至少有3个顶点。由最优子结构性质，t[i][j]的值应为t[i][k]的值加上t[k+1][j]的值，再加上三角形$v_{i-1}v_kv_j$的权值，其中i≤k≤j-1。由于在计算时还不知道k的确切位置，而k的所有可能位置只有j-i个，因此可以在这j-i个位置中选出使t[i][j]值达到最小的位置。由此，t[i][j]可递归地定义为：

$$t[i][j]=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{t[i][k]+t[k+1][j]+w(v_{i-1}v_kv_j)\} & i<j\end{cases}$$

Sch1-4 动态规划

例3：图像压缩问题

图象的变位压缩存储格式将所给的象素点序列{$p_1,p_2,\cdots,p_n$},$0\leqslant p_i\leqslant 255$分割成m个连续段$S_1,S_2,\cdots,S_m$。第i个象素段Si中$(1\leqslant i\leqslant m)$，有l[i]个象素,且该段中每个象素都只用b[i]位表示。设 $t[i]=\sum_{k=1}^{i-1} l[k]$,则第i个象素段S_i为 $S_i=\{p_{t[i]+1},\cdots,p_{t[i]+l[i]}\}$

设 $h_i=\left\lceil \log\left(\max_{t[i]+1\leq k\leq t[i]+l[i]} p_k+1\right)\right\rceil$ ，则$h_i\leq b[i]\leq 8$。因此需要用3位表示b[i],如果限制$1\leq l[i]\leq 255$，则需要用8位表示l[i]。因此，第i个象素段所需的存储空间为l[i]*b[i]+11位。按此格式存储象素序列{p1,p2,⋯,pn}，需要 $\sum_{i=1}^{m} l[i]*b[i]+11m$ 位的存储空间。

图象压缩问题要求确定象素序列{$p_1,p_2,\cdots,p_n$}的最优分段，使得依此分段所需的存储空间最少。每个分段的长度不超过256位。

Sch1-4 动态规划算法

设l[i]，b[i]，是{p1,p2,…,pn}的最优分段。显而易见，l[1]，b[1]是$\{p_1,\cdots,p_{l[1]}\}$的最优分段，且l[i]，b[i]，是$\{p_{l[1]+1},\cdots,p_n\}$的最优分段。即图象压缩问题满足最优子结构性质。

设s[i]，1≤i≤n，是象素序列{p1,…,pn}的最优分段所需的存储位数。由最优子结构性质易知：

$$s[i]=\min_{1\le k\le \min\{i,256\}}\{s[i-k]+k*\mathrm{bmax}(i-k+1,i)\}+11$$

$$\mathrm{bmax}(i,j)=\left\lceil \log\left(\max_{i\le k\le j}\{p_k\}+1\right)\right\rceil$$

其中

算法复杂度分析：

由于算法compress中对k的循环次数不超这256，故对每一个确定的i，可在时间O(1)内完成的计算。因此整个算法所需的计算时间为O(n)。

Sch1-4 动态规划

例4：多边形游戏

多边形游戏是一个单人玩的游戏，开始时有一个由n个顶点构成的多边形。每个顶点被赋予一个整数值，每条边被赋予一个运算符“+”或“*”。所有边依次用整数从1到n编号。

游戏第1步，将一条边删除。

随后n-1步按以下方式操作：

(1)选择一条边E以及由E连接着的2个顶点V1和V2；

(2)用一个新的顶点取代边E以及由E连接着的2个顶点V1和V2。将由顶点V1和V2的整数值通过边E上的运算得到的结果赋予新顶点。

最后，所有边都被删除，游戏结束。游戏的得分就是所剩顶点上的整数值。

问题：对于给定的多边形，计算最高得分(边删除顺序不同会导致不同结果)。

Sch1-4动态规划

- 最优子结构性质：

在所给多边形中，从顶点i(1≤i≤n)开始，长度为j(链中有j个顶点)的顺时针链p(i，j)可表示为v[i]，op[i+1]，…，v[i+j-1]。

如果这条链的最后一次合并运算在op[i+s]处发生(1≤s≤j-1)，则可在op[i+s]处将链分割为2个子链p(i，s)和p(i+s，j-s)。

设m1是对子链p(i，s)的任意一种合并方式得到的值，而a和b分别是在所有可能的合并中得到的最小值和最大值。m2是p(i+s，j-s)的任意一种合并方式得到的值，而c和d分别是在所有可能的合并中得到的最小值和最大值。依此定义有a≤m1≤b，c≤m2≤d，则：

(1) 当op[i+s]='+'时，显然有a+c≤m≤b+d

(2) 当op[i+s]='*'时，有min{ac，ad，bc，bd}≤m≤max{ac，ad，bc，bd}

换句话说，主链的最大值和最小值可由子链的最大值和最小值得到！

Sch1-4 动态规划

递归求解

设m[i,j,0]表示链p(i,j)合并的最小值，m[i,j,1]是最大值。若最优合并在op[i+s]处将p[i,j]分为2个长度小于j的子链p(i,s)和p(i+s,j-s)，且从顶点i开始的长度小于j的子链的最大值和最小值均已计算出，记：

a = m[i,i+s,0]，b=m[i,i+s,1], c=m[i+s, j-s, 0], d = m[i+s, j-s, 1]

(1) 当op[i+s]= ‘+’ 时，m[i,j,0] = a + c，m[i,j,1]=b+d

(2) 当op[i+s]= ‘*’ 时，

m[i,j,0] = min{ac, ad, bc, bd}, m[i,j,1] = max{ac, ad, bc, bd}

综合(1)和(2)，将p(i,j)在op[i+s]处断开的最大值记为maxf(i,j,s)，最小值记为minf(i,j,s)，则

$$\min f(i,j,s)=\begin{cases} a+c & op[i+s]='+' \\ \min\{ac,ad,bc,bd\} & op[i+s]='*' \end{cases}$$

$$\max f(i,j,s)=\begin{cases} b+d & op[i+s]='+' \\ \max\{ac,ad,bc,bd\} & op[i+s]='*' \end{cases}$$

Sch1-4 动态规划

由于最优断开位置s有1≤s≤j-1的j-1中情况，由此可知：

$$m[i,j,0]=\min_{1\le s\le j}\{\min f(i,j,s)\},1\le i,j\le n$$

$$m[i,j,1]=\max_{1\le s\le j}\{\max f(i,j,s)\},1\le i,j\le n$$

初始边界值显然为：

$$m[i,1,0] = v[i],\ 1\leqslant i \leqslant n$$

$$m[i,1,1] = v[i],\ 1\leqslant i \leqslant n$$

由于多边形是封闭的，在上面的计算中，当i+s>n时，顶点i+s实际编号为(i+s) mod n。按上述递推式计算出的m[i,n,1]即为游戏首次删去第i条边后得到的最大得分。

Sch1-4 贪心算法

基本要素：

- 贪心选择性质：所求问题的整体最优解可以通过一系列局部最优的选择，即贪心选择来达到。这是贪心算法可行的第一个基本要素，也是贪心算法与动态规划算法的主要区别。
- 最优子结构性质：问题的最优解包含其子问题的最优解。

* 对于一个具体问题，要确定它是否具有贪心选择性质，必须证明每一步所作的贪心选择最终导致问题的整体最优解。

Sch1-4 贪心算法

- 例5：最小生成树问题

设G＝(V,E)是无向连通带权图，即一个网络。E中每条边(v,w)的权为c[v][w]。如果G的子图G’是一棵包含G的所有顶点的树，则称G’为G的生成树。生成树上各边权的总和称为该生成树的耗费。在G的所有生成树中，耗费最小的生成树称为G的最小生成树。

网络的最小生成树在实际中有广泛应用。例如，在设计通信网络时，用图的顶点表示城市，用边(v,w)的权c[v][w]表示建立城市v和城市w之间的通信线路所需的费用，则最小生成树就给出了建立通信网络的最经济的方案。

Sch1-4 贪心算法

- 最小生成树性质：

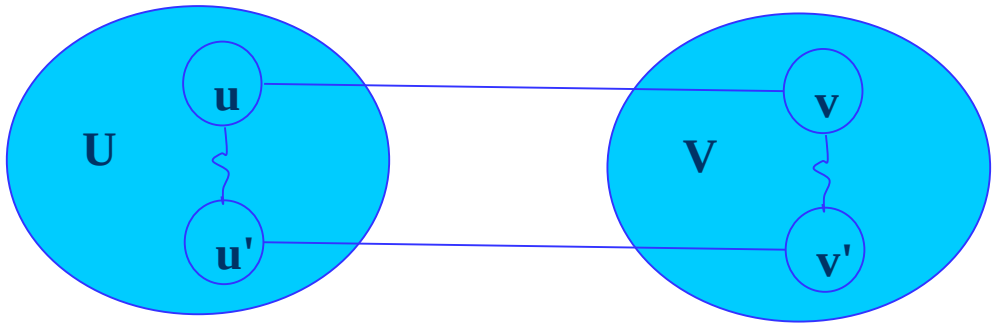
设G=(V,E)是连通带权图，U是V的真子集。如果(u,v)∈E，且u∈U，v∈V-U，且在所有这样的边中，(u,v)的权c[u][v]最小，那么一定存在G的一棵最小生成树，它以(u,v)为其中一条边。这个性质有时也称为MST性质。

用贪心算法设计策略可以设计出构造最小生成树的有效算法。本节介绍的构造最小生成树的Prim算法和Kruskal算法都可以看作是应用贪心算法设计策略的例子。尽管这2个算法做贪心选择的方式不同，它们都利用了上面的最小生成树性质：

Sch1-4 贪心算法

- **证明：**

假设G的任何一棵最小生成树都不含边(u, v)。将边(u, v)添加到G的一棵最小生成树T上，将产生含有边(u,v)的圈，并且在这个圈上有一条不同于(u, v)的边(u', v')，使得u'∈U, v'∈V-U。如下图所示，将边(u', v')删去，得到G的另一棵生成树T'。由于c[u][v]<=c[u'][v']，所以T'的耗费<=T的耗费。于是，T'是一颗含有边(u,v)的最小生成树，这与假设矛盾。



含边(u,v)的圈

Sch1-4 贪心算法

- Prime算法

设G=(V,E)是连通带权图，V={1,2,…,n}，算法思想为：首先置S={1}，然后，只要S是V的真子集，就作如下的贪心选择：选取满足条件i∈S，j∈V-S，且c[i][j]最小的边，将顶点j添加到S中。这个过程一直进行到S=V时为止。

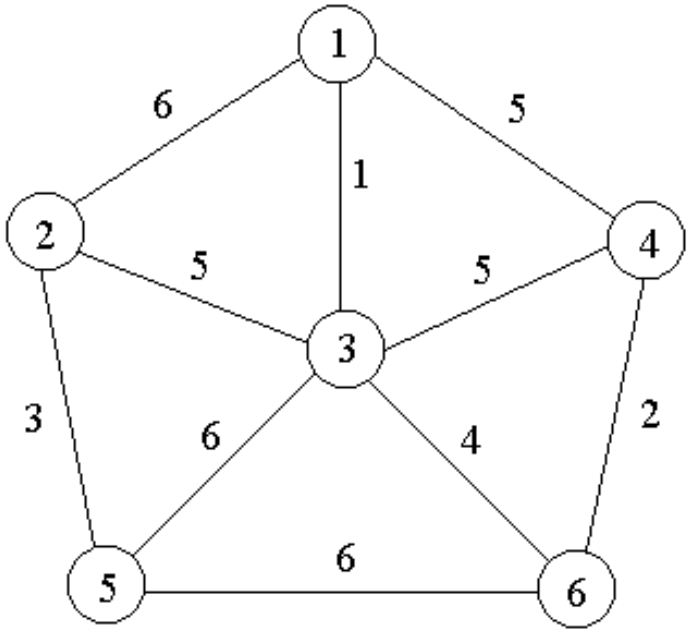
在这个过程中选取到的所有边恰好构成G的一棵最小生成树。

```
Void Prime( n, c[][])
{
    T= Φ;
    S = {1};
    while( S != V ){
        (i,j) = i ∈ S且j ∈ V-S的最小权边;
        T = T U {(i,j)};
        S = S U {j};
    }
}
```

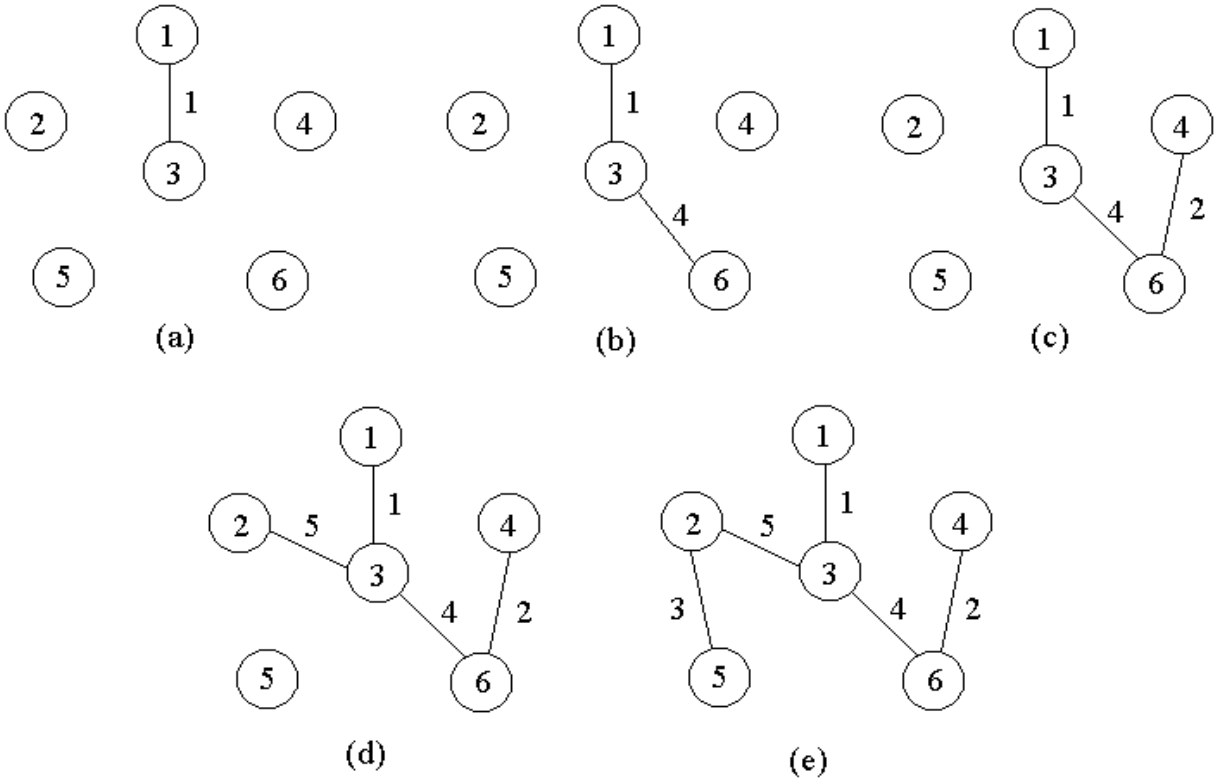
Sch1-4 贪心算法

利用最小生成树性质和数学归纳法容易证明，上述算法中的边集合T始终包含G的某棵最小生成树中的边。因此，在算法结束时，T中的所有边构成G的一棵最小生成树。

例如，对于右图中的带权图，按Prim算法选取边的过程如下页图所示。



Sch1-4 贪心算法



Sch1-4 贪心算法

在上述Prim算法中，还应当考虑如何有效地找出满足条件i∈S,j∈V-S，且权c[i][j]最小的边(i,j)。实现这个目的的较简单的办法是设置2个数组closest和lowcost。

在Prim算法执行过程中，先找出V-S中使lowcost值最小的顶点j，然后根据数组closest选取边(j,closest[j])，最后将j添加到S中，并对closest和lowcost作必要的修改。

用这个办法实现的Prim算法所需的计算时间为 $O(n^2)$

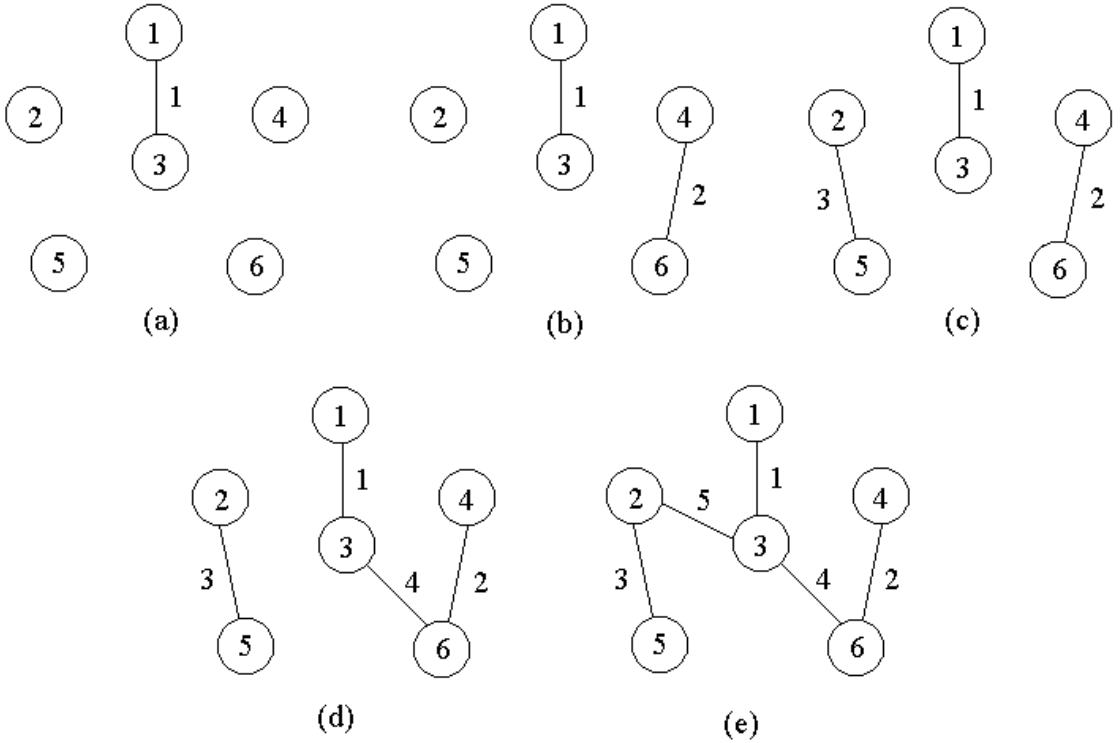
Sch1-4 贪心算法

- Kruskal算法

Kruskal算法构造G的最小生成树的基本思想是：首先将G的n个顶点看成n个孤立的连通分支。将所有的边按权从小到大排序。然后从第一条边开始，依边权递增的顺序查看每一条边，并按下述方法连接2个不同的连通分支：当查看到第k条边(v,w)时，如果端点v和w分别是当前2个不同的连通分支T1和T2中的顶点时，就用边(v,w)将T1和T2连接成一个连通分支，然后继续查看第k+1条边；如果端点v和w在当前的同一个连通分支中，就直接再查看第k+1条边。这个过程一直进行到只剩下一个连通分支时为止。

Sch1-4 贪心算法

- 例如，对前面的连通带权图，按Kruskal算法顺序得到的最小生成树上的边如下图所示。



Sch1-4 贪心算法

时间复杂度：

当图的边数为e时，Kruskal算法所需的时间是 $O(e\log e)$ ：

- 当 $e=\Omega(n^2)$ 时，Kruskal算法比Prim算法差；
- 当 $e=o(n^2)$ 时，Kruskal算法却比Prim算法好得多。

谢谢！

Q & A