算法设计与分析

4. 贪心算法

本章主要知识点

4.1 活动安排问题
4.2 贪心算法的基本要素
4.3 最优装载
4.4 哈夫曼编码
4.5 单源最短路径—自学
4.6 最小生成树
4.7 多机调度问题

学习要点

- 理解贪心算法的概念
- 掌握贪心算法的基本要素

（1）最优子结构性质

（2）贪心选择性质

- 理解贪心算法与动态规划算法的差异
- 理解贪心算法的一般理论
- 通过应用范例学习贪心设计策略

引言

- 当一个问题具有最优子结构性质时，如何求解?

- 假设有四种硬币，它们的面值分别为0.25元、0.10元、0.05元、0.01元，现在要给某顾客0.63元。如何找，使所拿出的硬币个数是最少?

- 假设有四种硬币，它们的面值分别为一元、五角、一角、五分，去给顾客2元7角5分，又如何找钱?

贪心策略

- 贪心策略不从整体最优考虑，而总是某种意义上是局部最优的方面作出选择。
- 即：贪心策略总是作出在当前看来最好的选择。
- 目标：得到的最终结果也是整体最优的。

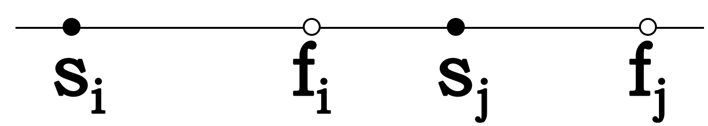
局限性：贪心算法未必能对所有问题都得到整体最优解

贪心算法对许多问题能产生整体最优解，且效率较高。
如单源最短路径问题，最小生成树问题等

4.1 活动安排问题

问题描述

- 设有n个活动的集合E={1,2,…,n}，其中每个活动都要求使用同一资源，如会场、教室等，而在同一时间内只有一个活动能使用这一资源。
- 每个活动i都有一个要求使用该资源的起始时间s_i和一个结束时间f_i,且$s_i<f_i$。
- 如果选择了活动i，则它在半开时间区间$[s_i, f_i)$内占用资源。若区间$[s_i, f_i)$与区间$[s_j, f_j)$不相交，则称活动i与活动j是相容的。



即：当$s_i \geq f_j$或$s_j \geq f_i$时，活动i与活动j相容。

1、求解策略

如何求解?

- 预备：将输入的活动以其完成时间的非减序排列（即升序排列）
- 策略：从队列中每次总是选择具有最早完成时间的相容活动加入活动集合A中

2、贪心算法GreedySelector

在下面所给出的解活动安排问题的贪心算法greedySelector：

```
int GreedySelector(int n, int  s[], int
  f[], bool A[]){
      A[1]=true;
      int j=1, count=1;
      for (int i=2;i<=n;i++) {
          if (s[i]>=f[j]) {
              A[i]=true;
              j=i;   count++;
          }
          else A[i]=false;
      }
      return count;
 }
```

3、复杂性

- 先对活动的完成时间作从小到大排列
- 算法GreedySelector每次总是选择具有最早完成时间的相容活动加入集合A中。
- 算法的贪心选择目的：使剩余可安排时间段尽量大，以便安排尽可能多的相容活动。
- 算法GreedySelector的效率： O(nlogn)

1. 对给出的活动按非减序排列，可用O(nlogn)时间
2. 对已按结束时间作非减序排列的活动，只需O(n)的时间安排n个活动，使最多的活动能相容地使用公共资源。

活动安排问题求解启示

- 活动安排问题要求在所给的活动集合中选出最大的相容活动子集合。
- 贪心算法可提供一个简单、漂亮的方法，使得尽可能多的活动能兼容地使用公共资源。

4、举例

例： **设待安排的11个活动的开始时间和结束时间按结束时间的非减序排列如下：**

i	1	2	3	4	5	6	7	8	9	10	11
S[i]	1	3	0	5	3	5	6	8	8	2	12
f[i]	4	5	6	7	8	9	10	11	12	13	14

活动安排问题贪心算法小结及正确性证明

结论：活动安排问题的贪心算法可求得整体最优解

策略：1、将输入的活动以其完成时间的非减序排列

2、若被检查的活动i的开始时间S_i小于最近选择的活动j的结束时间f_i，则不选择活动i，否则选择活动i加入集合A中。

正确性证明步骤：

该结论可用数学归纳法证明，最优性证明：须分两步进行：

1、开始时的贪心选择　　2、剩下的活动可继续作贪心选择（实际上证最优子结构性质）

4.2 贪心算法的基本要素

对一般的其它问题，贪心算法未必总能求得整体最优解。

具有什么性质的问题可用贪心算法?

1、贪心算法的基本要素

1.贪心选择性质

2.最优子结构性质

2、贪心选择性质与最优子结构性质

贪心选择性质

指所求问题的**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。每一步都可通过局部最优达到。

最优子结构性质

当一个问题的最优解包含其子问题的最优解时，称此问题具有**最优子结构性质**。

问题具有贪心选择性质的证明方法

1. 明确所说的贪心选择策略S
2. 按该贪心选择策略，可选择一个局部最优解，确定第一步选择（假定有一个最优解A）
3. 考察问题的最优解A，并证明它的第一步必可通过贪心选择策略开始（或通过A构造的一个最优解B的第一步必可通过贪心选择策略开始）。注：两个最优解A和B对最优性应一致，否则矛盾。

3、贪心算法与动态规划算法的差异

- 贪心选择：问题的整体最优解可以通过一系列局部最优的选择求得
- 动态规划：每步选择依赖与相关子问题，待子问题求解后，才作出选择

共同点：

贪心算法和动态规划算法都要求问题具有最优子结构性质

都通过求解一系列子问题的解求得原问题的解

4、贪心算法与动态规划算法的差异

动态规划： 通常以**自底向上**的方式解各子问题

贪心算法： 通常以**自顶向下**的方式进行，以迭代的方式作出相继的贪心选择，每作一次贪心选择就将所求问题简化为规模更小的子问题。

两个问题：

1. 对于具有**最优子结构**的问题应该选用贪心算法还是动态规划算法求解?
2. 是否能用动态规划算法求解的问题也能用贪心算法求解?

5、举例——0-1背包问题

- **0-1背包问题：**

  给定n种物品和一个背包。物品i的重量是Wi，其价值为Vi，背包的容量为C。应如何选择装入背包的物品，使得装入背包中物品的总价值最大？

在选择装入背包的物品时，对每种物品i只有2种选择，即装入背包或不装入背包。不能将物品i装入背包多次，也不能只装入部分的物品i。

6、举例——一般背包问题

- 与0-1背包问题类似，所不同的是在选择物品i装入背包时，可以选择物品i的一部分，而不一定要全部装入背包，$1 \leqslant i \leqslant n$。

这2类背包问题都具有最优子结构性质，极为相似。

➢一般背包问题可以用贪心算法求解。

➢0-1背包问题不能用贪心算法求解。

一般背包问题的贪心算法求解

用贪心算法解背包问题的基本步骤：

1. 计算每种物品单位重量的价值Vi/Wi，
2. 确定贪心选择策略：尽可能多地将单位重量价值最高的物品装入背包。
3. 若将一物品全部装入背包后，背包内的物品总重量未超过C，则选择单位重量价值次高的物品并尽可能多地装入背包。依此策略一直进行下去，直到背包装满为止。

0-1背包问题不能用贪心选择

- 在考虑0-1背包问题时，要么选择、要么不选择该物品，然后作出最好选择。
- 对0-1背包问题，贪心选择不能得到最优解的原因是在这种情况下，它无法保证最终能将背包装满，部分闲置的背包空间使每公斤背包空间的价值降低了。需用事例说明，见后。

  - 动态规划算法的确可有效地解0-1背包问题。

7、实例

1．设　v: 2 4 5 2　　　取c=5

　　　w: 2 3 4 7

其0-1背包问题不能用贪心算法求解。

但显然，可选物品1、2，对应的价值为6

2．再设　　v: 60 100 120

　　　　　w: 10 20 30

现 c=50，其一般背包问题能用贪心算法求解。

显然，可选物品1、2全部，物品3部分，对应的价值为240

一般背包问题的选择策略

例：设　$n = 3$，　$c = 20$

$(v_1, v_2, v_3) = (24, 15, 25)$，$(w_1, w_2, w_3) = (15, 10, 18)$

1. 以“v_i从大到小排列输入”为选择标准：

$$(x_1, x_2, x_3) = (2/15, 0, 1) \qquad v = 28.2$$

2. 以“w_i从小到大排列输入”为选择标准：

$$(x_1, x_2, x_3) = (10/15, 1, 0) \qquad v = 31$$

3. 以“v_i/w_i从大到小排列输入”为选择标准：

$$(x_1, x_2, x_3) = (1, 5/10, 0) \qquad v = 31.5$$

一般背包问题的贪心算法复杂性

- 算法：knapsack
- 复杂性：O(n log n)，排序

4.3 最优装载

1、最优装载问题描述

- **问题描述：**

　有一批集装箱要装上一艘载重量为c的轮船。其中集装箱i的重量为w_i。最优装载问题要求确定在装载体积不受限制的情况下，将尽可能多的集装箱装上轮船。

2、算法描述

- **如果用x_i=1表示将第i件集装箱装入船，用x_i=0表示未放入，则问题变为选择一组x_i（i=0,1）使得**

$$\max \sum_{i=1}^{n} x_i$$

$$\begin{cases} \sum_{i=1}^{n} w_i x_i \le C \\ x_i \in \{0,1\}, 1 \le i \le n \end{cases}$$

与0-1背包问题的差异在哪儿?

3、策略

- 最优装载问题可用贪心算法求解

1. 确定贪心选择策略：采用重量最轻者先装，可产生最优装载问题的最优解。
2. 步骤：

- 预备：先对货物按重量从轻到重排序
- 策略：依次按最轻者装入

4、两个性质需证明

1. 贪心选择性质

可以证明最优装载问题具有贪心选择性质。

2. 最优子结构性质

最优装载问题具有最优子结构性质。

证明需掌握

5、最优算法

```
void  Loading(int x[],  int w[], int c, int n)
   {
      int *t=new int [n+1],  val;
      Sort(w, t, n); val=0;
      for (int i = 1; i <= n; i++)
          x[i] = 0;
      for (int i = 1; i <= n&&w[t[i]]<=c;  i++) {
          x[t[i]] = 1;val=val+ v[t[i]];
          c - = w[t[i]];
      }
      return val;
   }
```

由最优装载问题的贪心选择性质和最优子结构性质，证明算法loading的正确性。

6、复杂性

- 算法loading的主要计算量在于将集装箱依其重量从小到大排序，故算法所需的计算时间为 $O(n\log n)$。

4.4 哈夫曼编码

问 题

- 一个文件含100,000个字符，共有6个字母a,b,c,d,e,f出现，频率如下：

  45 13 12 16 9 5

现用0、1串表示字母对文件进行压缩

用定长码：需300,000位

用变长码：	a	b	c	d	e	f
	0	101	100	111	1101	1100

需224,000位

编码：cad → 1000111

1、哈夫曼编码

- **哈夫曼编码:**用于数据文件压缩,其压缩率在20%～90%之间。

- **哈夫曼编码特点：** 给出现频率高的字符较短的0、1编码，出现频率较低的字符以较长的编码，可以大大缩短总码长。

编码方法

1）前缀码

对每一个字符规定一个0,1串作为其代码，并要求任一字符的代码都不是其他字符代码的前缀。这种编码称为前缀码。

2）译码方式：取前缀码

3）编码方法：构造二叉树

最优前缀码、平均码长

表示最优前缀码的二叉树总是一棵完全二叉树，即树中任一非叶结点都有2个儿子结点。

1） 平均码长定义为：

$$B(T)=\sum_{c\in C} f(c)d_T(c)$$

C为字符集，f(c)为字符c在文件中出现的频率，$d_T(c)$为深度。

2）最优前缀码：使平均码长达到最小的前缀码编码方案称为给定编码字符集C的最优前缀码。

2、构造哈夫曼编码

- 哈夫曼提出构造最优前缀码的贪心算法，由此产生的编码方案称为哈夫曼编码。

- 哈夫曼算法以自底向上的方式构造表示最优前缀码的二叉树T。

- 算法思想：算法以|C|个叶结点开始，执行|C|－1次的“合并”运算后产生最终所要求的树T。

3、哈夫曼树-算法实现HuffmanTree

- 算法步骤

1. 用C中每一字符c的频率f(c)初始化一个队列Q
2. 对优先队列Q用贪心选择：取出具有最小频率的2棵树x,y,并将这2棵树合并为新树z，其频率为合并的2棵树的频率之和，并将新树插入优先队列Q。
3. 作n－1次类似的合并。优先队列中只剩下一棵树，即所要求的树T。

举例

例：在由8个字母a,e,i,o,u,t,h,b构成的文件中，频率分别为：

e	t	a	b	i	o	u	H
0.6	0.2	0.05	0.05	0.03	0.03	0.03	0.01

求对应的Huffman树

4、哈夫曼树的实现方式与复杂性

- 用最小堆实现优先队列Q。
- 初始化优先队列需要O(n)计算时间，由于最小堆的DeleteMin和Insert运算均需O(log n)时间，n－1次的合并总共需要O(nlog n)计算时间。
- n个字符的哈夫曼算法的计算时间为O(nlog n)
- （或简单取最小数，复杂性增加）

5、哈夫曼算法的正确性

要证明哈夫曼算法的正确性，需证明最优前缀码问题具有贪心选择性质和最优子结构性质：

(1)贪心选择性质

(2)最优子结构性质

上机实验题4

- 哈夫曼编码问题

任意输入若干个字母（或字）的频率值，输出对应的Huffman编码

具体内容见 实验四 哈夫曼编码

4.5 单源最短路径

- 自学计算原理
- 会求并编程

单源最短路径问题描述

- 给定带权有向图G =(V,E)，其中每条边的权是非负实数。给定V中的一个顶点v，称为**源**。
- **单源最短路径问题：**计算从源v到所有其他各顶点u的**最短路长度**d[u]。

这里路径的长度是指路上各边权之和。

特殊路径

设S为顶点集合，$v \in S$。

设$u \in V$，把从源v到u且中间只经过S中顶点的路径C称为从源v到u的特殊路径。

dist[u]：记顶点u所对应的最短特殊路径长度。

注：从源v到u的最短路径长度≤ dist[u]

1. 算法基本思想

- 采用Dijkstra提出的解单源最短路径问题算法

Dijkstra算法基本思想:

设置顶点集合S并不断地作贪心选择来扩充这个集合。u∈S，当且仅当，从源v到该顶点u的最短路径长度已知。

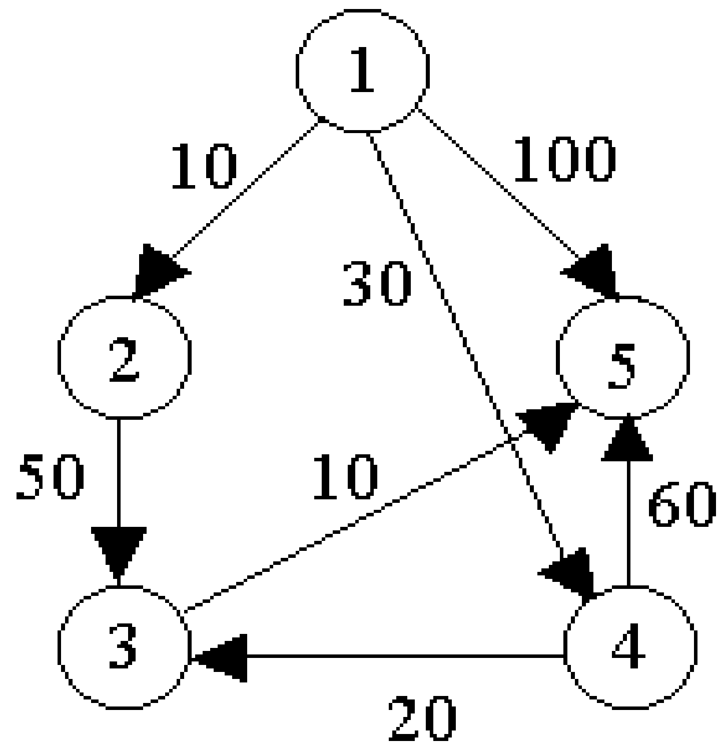
Dijkstra算法策略

初始时，S={v}。

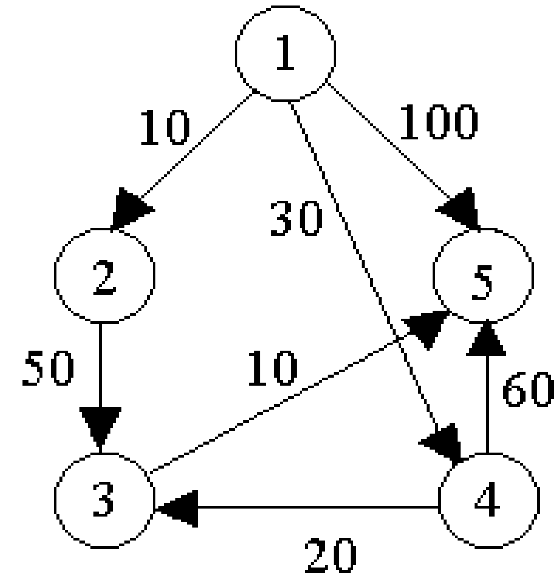
算法每次从V-S中取出具有最短特殊路长度的顶点u，将u添加到S中，同时对数组dist作必要的修改。若S=V，则算法完成。

算法举例

- 应用Dijkstra算法计算从源顶点1到其他顶点间最短路径



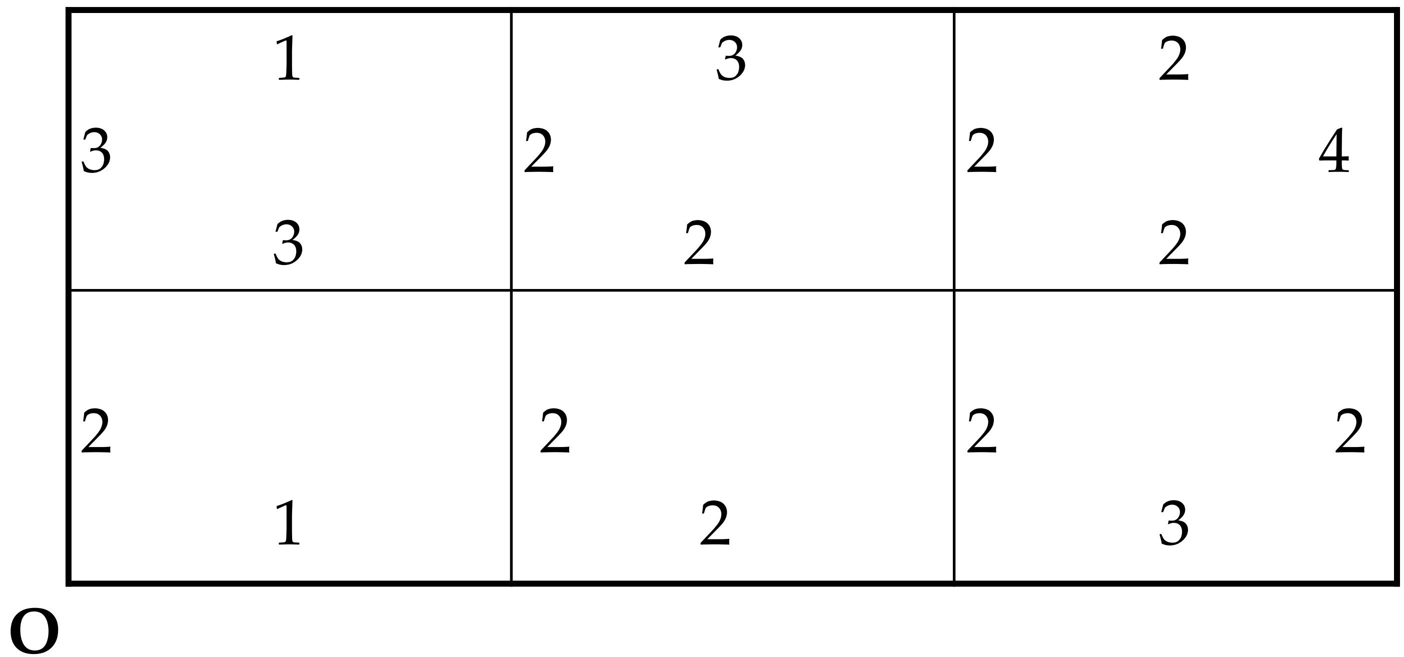
Dijkstra算法的迭代过程



迭代	S	u	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	-	10	maxint	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90
3	{1,2,4,3}	3	10	50	30	60
4	{1,2,4,3,5}	5	10	50	30	60

练习

- 从源顶点O到其他顶点间最短路径



2.算法过程

用a[i][j]记边（i,j）的权，数组dist[u]记从源v到顶点u所对应的最短特殊路径长度

S1:初始化，S←∅, T←∅,对每个y∉S，

{ dist[y]=a[v][y];prev[y]=nil; }

S2:若S=V，则输出dist,prev,结束

S3：u←V\S中有最小dist值的点，S←S∪{u}

S4：对u的每个相邻顶点x，调整dist[x]：即

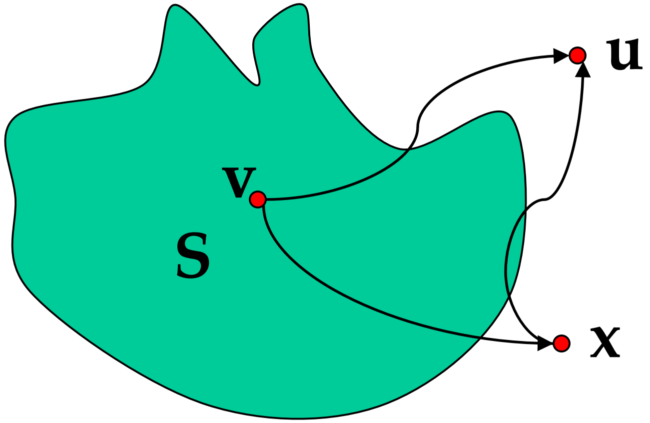
若dist[x]>dist[u]+a[u][x],则

{ dist[x]=dist[u]+a[u][x];prev[x]=u },转S2

3. 算法的正确性

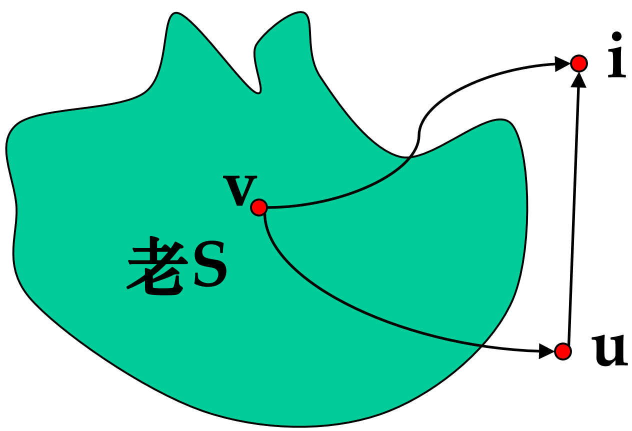
(1) 贪心选择性质

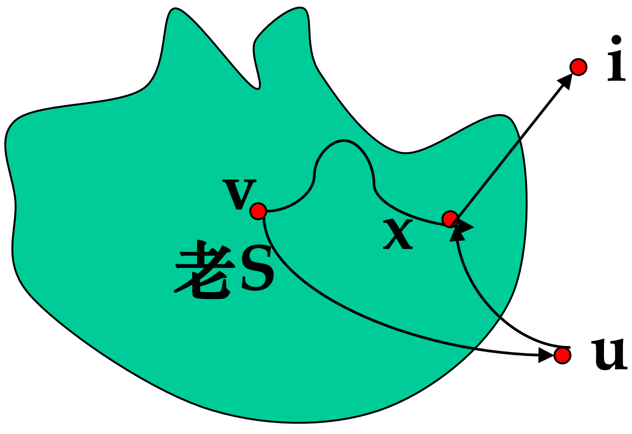
从S外具有最短特殊路径的点u必定使dist[u]最短



(2) 最优子结构性质

添加u后其它外部点的距离调整仅与u有关





4. 计算复杂性

对于具有n个顶点和e条边的带权有向图，如果用带权邻接矩阵表示这个图，那么Dijkstra算法的主循环体需要O(n)时间。这个循环需要执行n-1次，所以完成循环需要$O(n^2)$时间。算法的其余部分所需要时间不超过$O(n^2)$。

4.6 最小生成树

概 念

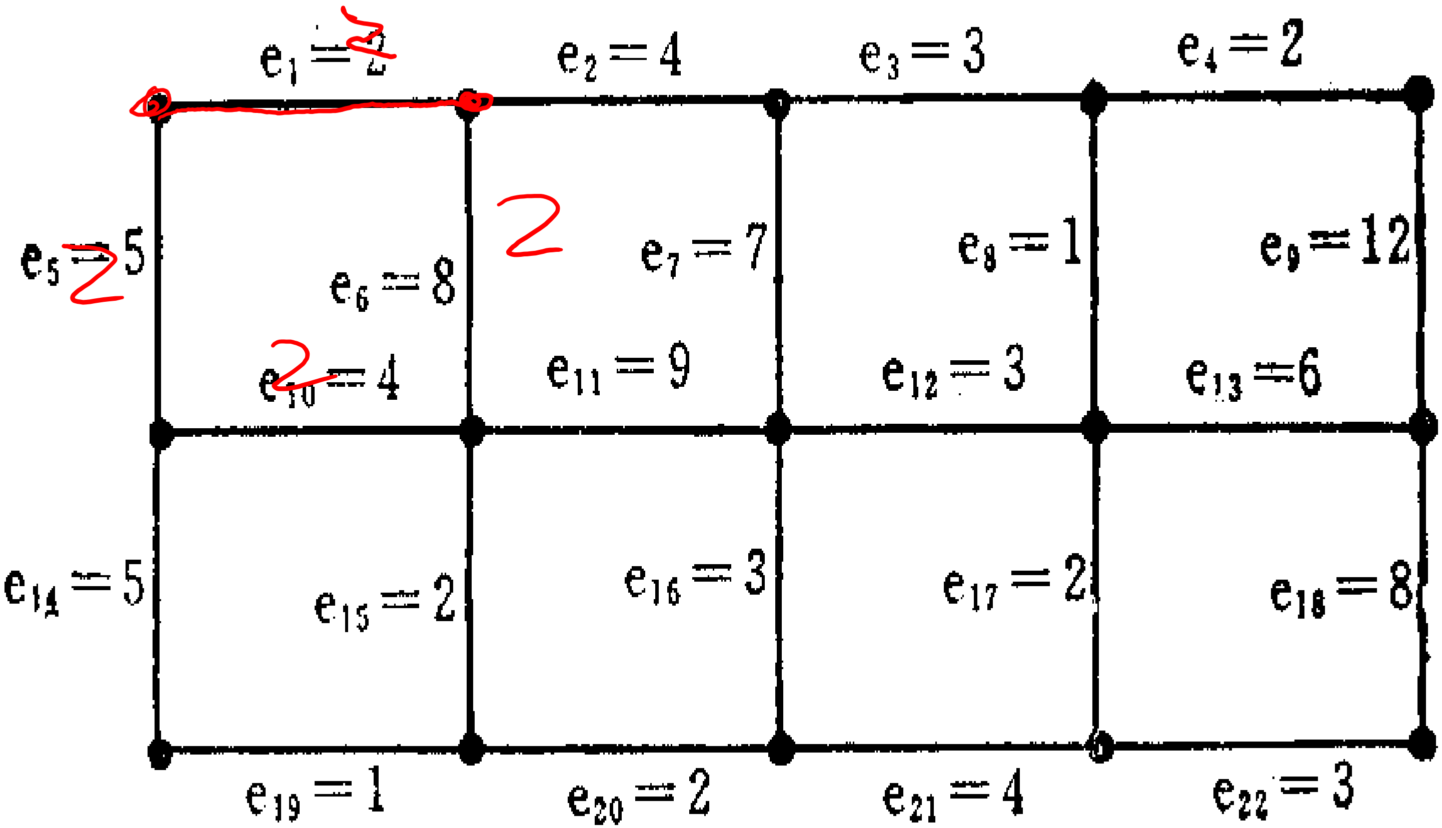
设G =(V,E)是无向连通带权图， |V|＝n，|E|=m，E中每条边e_i=(v,w)的权为c[v][w]，(在不引起混淆的条件下，为简单起见，用e_i表示|e_i|)。

1）**生成树：**如果G的一个子图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。

2）**最小生成树：**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。

3）**最短树：**是最小生成树问题的特例，指从G的边集合E中寻求一子集E_T，使得T＝（V，E_T）是一连通图，而且E_T边的长度之和最小。此时T必成一棵树.

例：求图G的最短生成树T



Prim算法

1.求最小生成树的Prim算法的基本思想

任取一顶点，设为V_1，作为起始点，让它进入集合S。只要S是V的真子集，就作如下的贪心选择：

每一次总选择一不属于S，但和S中顶点最为接近的顶点，设为v，即选择（u，v）$\in$L，使得v$\notin$S，u$\in$S，且（u，v）达到最小。将（u，v）作为一树枝，如此反复连续进行n-1次（此时S=V）。

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T。

2. Prim算法过程

S1．T← Ø；q[1]← -1.

S2．对i从2到n做　　//赋初值

　　始 p[i]←1；q[i]← d_{i1}；终；k←1

S3．若k＞n 转S5，否则min←∞；转S4

S4．对j从2到n做　　//找最短距离的点

　　若 0<q[j]<min 则做

　　始 min←q[j]；h←j 终

S5．T←T∪{(h，p(h))}；q[h]← -1

S6．j从2到n做　　//只调整与h有关的距离

　　若d_{hj}<q[j] 则做

　　始 q[j]←d_{hj}；p[j]←h 终

S7．k←k+1；转S3

S8．输出T；终止

算法中，

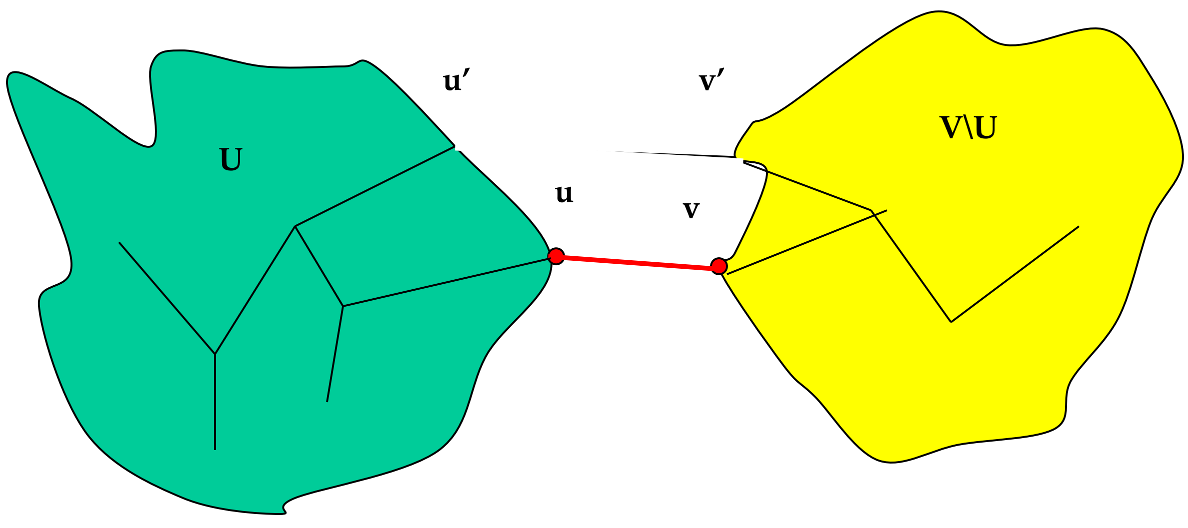
p[i]：记录和i点最接近的属于S的点

q[i]：记录i$\notin$S且与S中点的最短距离

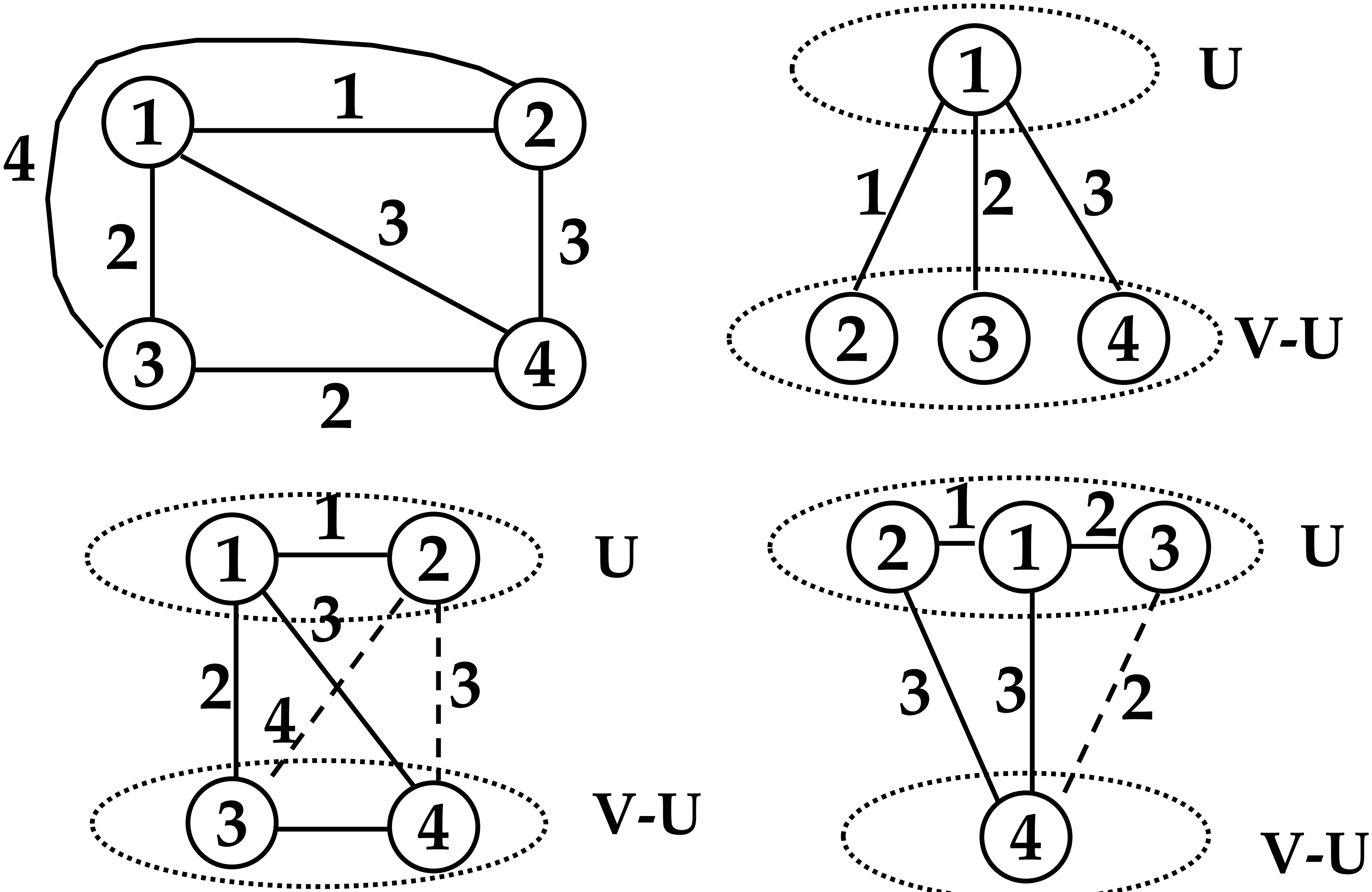
q[i]← -1：用以表示i点已进入集合S

3. 最小生成树性质—— MST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。这一性质有时也称为MST性质。

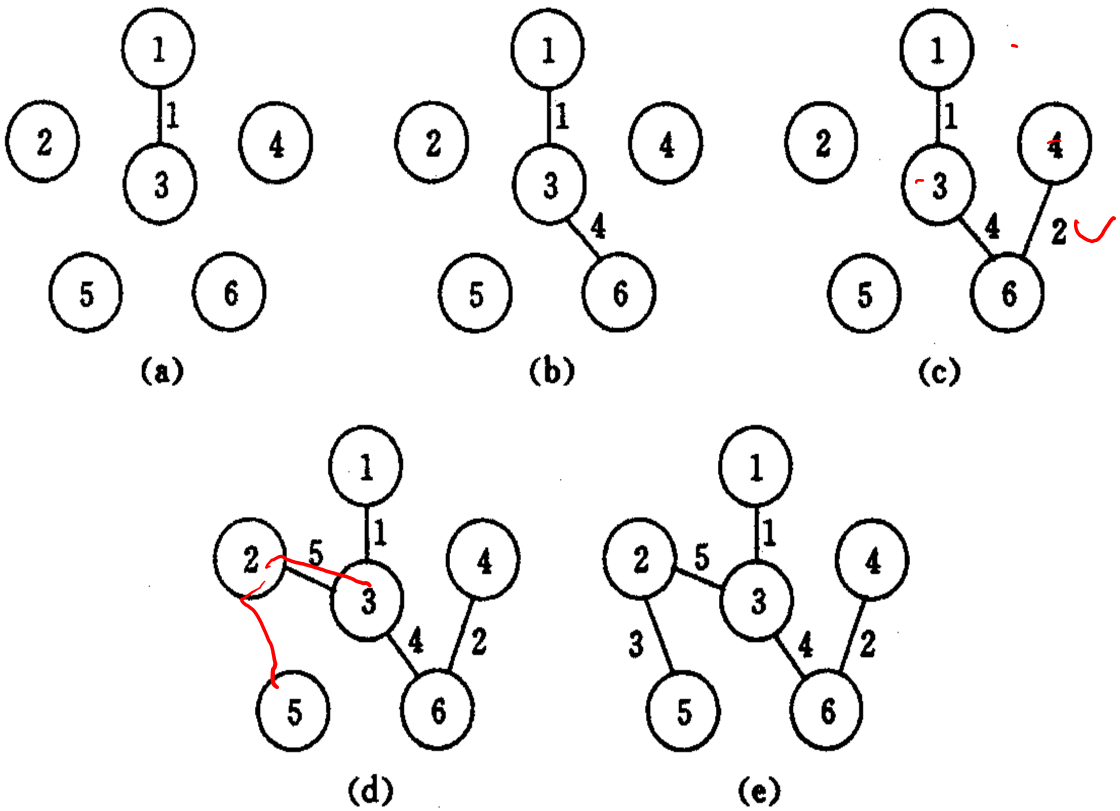


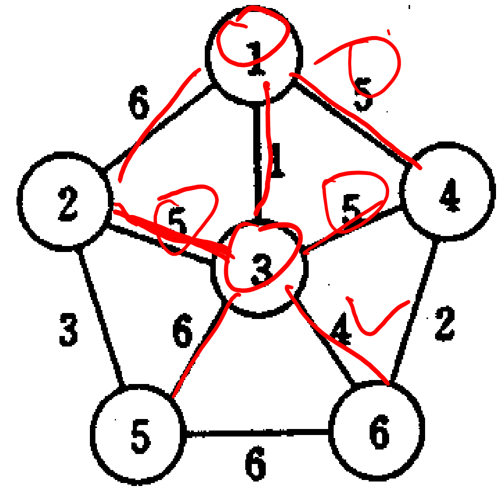
贪心选择举例

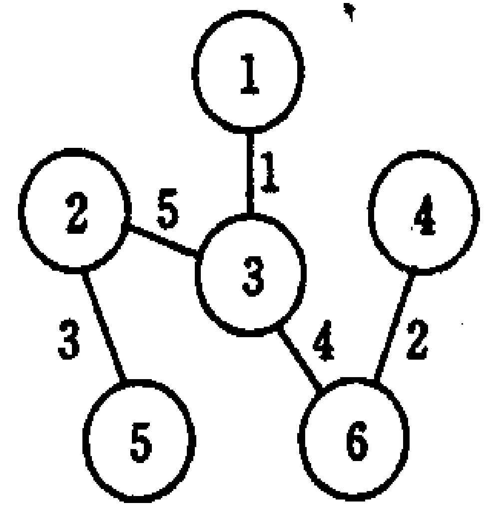


4.Prim算法的应用

- 用Prim算法求G的最短树







5.程序实现与复杂性

- 程序实现：略
- 复杂性：$O(n^2)$

Kruskal算法

1.Kruskal算法基本思想

1）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（或边长）从小到大排序。

2）从第一条边开始，依边权递增的顺序查看每一条边，并按下述方法连接两个不同的连通分支：

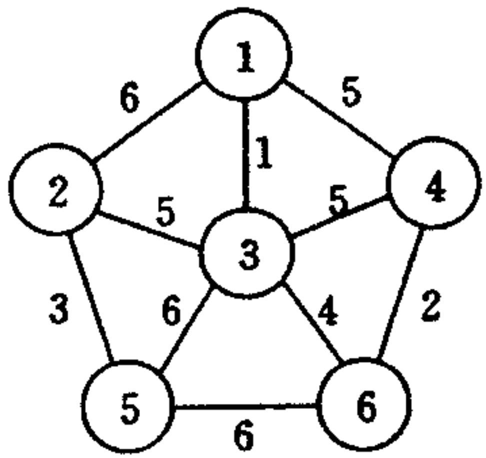
当查看到第k条边（v，w）时，若v和w分别在两个不同的连通分支T1和T2中，用边（v，w）将T1和T2连接成一个连通分支，然后继续查看第k+1条边；

若v和w在当前的同一个连通分支T′中，就直接查看第k+1条边（即构成圈，就放弃）。

这个过程一直进行到只剩下一个连通分支时为止。此时，这个连通分支就是G的一棵最小生成树。

2.实 例

- 求下面带权图G中的最小生成树

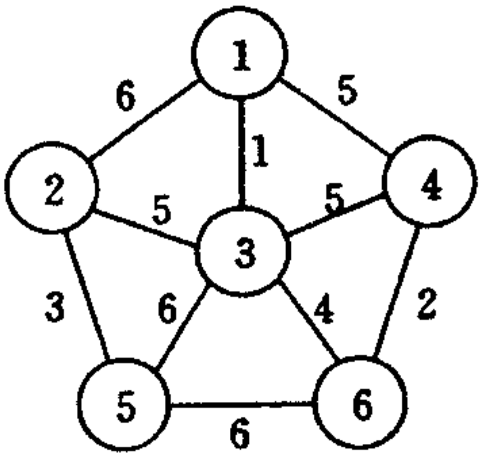


各边按权值排序为:

d_{13}=1　　d_{46}=2　　d_{25}=3　　$d_{36=}$4　　d_{14} = 5

d_{34}=5　　d_{23}=5　　d_{12}=6　　d_{35}=6　　d_{56}=6

求解过程



各边按权值排序:

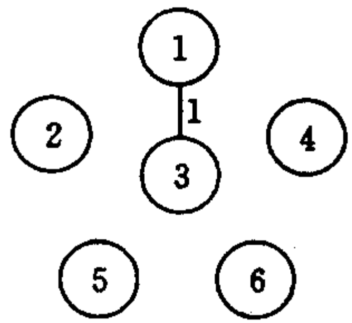
$d_{13}=1$ $d_{46}=2$

$d_{25}=3$ $d_{36}=4$

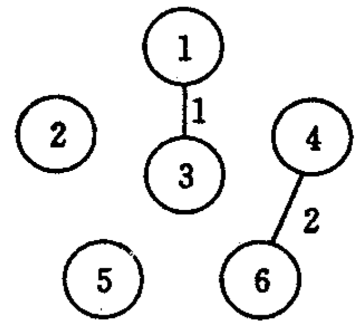
$d_{14}=5$ $d_{34}=5$

$d_{23}=5$ $d_{12}=6$

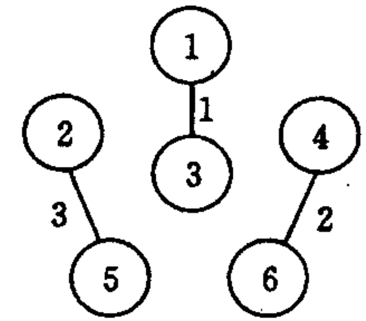
$d_{35}=6$ $d_{56}=6$



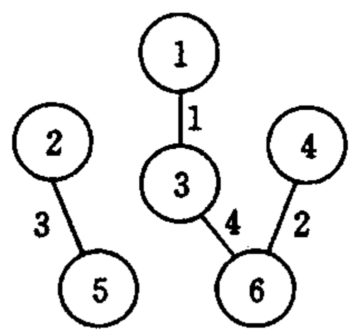
(a)



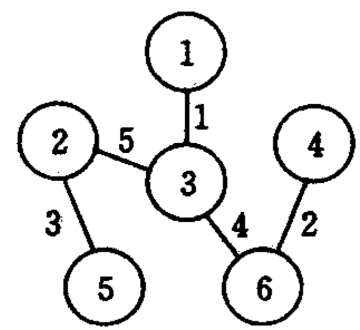
(b)



(c)



(d)



(e)

3. Kruskal算法

S1．对属于E的边按长度进行排序，不失一般性，设$e_1 <= e_2 <= \ldots <= e_m$。

S2．S←0；T← Ø ；i←1；t←0。

S3．若t= n-1，则转S6，否则转S4。

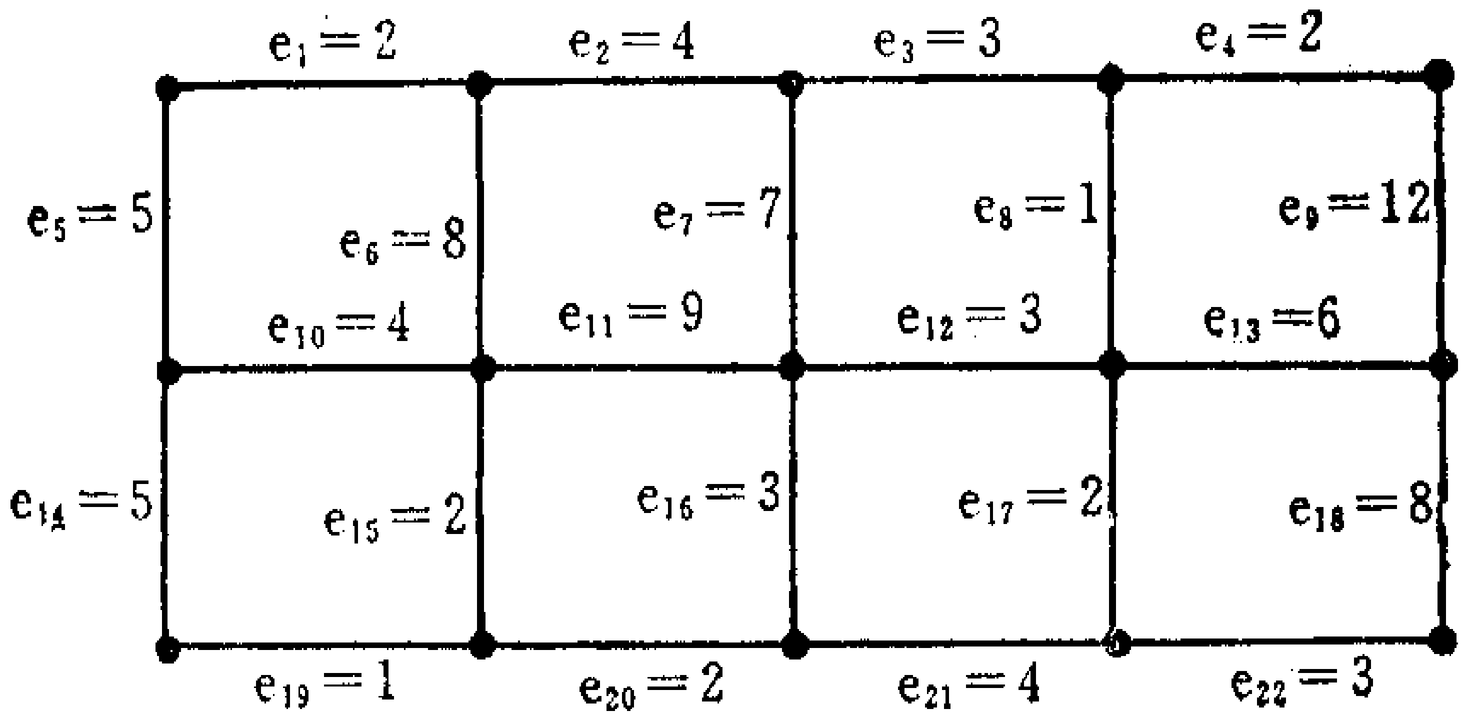
S4．若$T \cup \{e_i\}$构成一回路，则做

始i←i+l；转S4 终，否则转S5。

S5．$T \leftarrow T \cup \{e_i\}$；$S \leftarrow S+e_i$；t←t+1；i←i+1；转S3。

S6．输出T，S；终止。

练习：用kruskal算法求G的最小生成树



4. Kruskal算法的实现

- 问题：如何判断一些边构成了圈?

- 按权（边长）的递增顺序对边序列作优先队列。
- 用堆实现这个优先队列

- 按权的递增顺序查看等价于对优先队列执行DeleteMin运算，可以用堆实现。
- 对一个由连通分支组成的集合不断进行修改，需要用到抽象数据类型并查集UnionFind所支持的基本运算。

5. Prim算法与Kruskal算法的优劣

1. Prim算法无需对边进行从小到大的排序，而Kruskal算法先要对图G的边进行排序。
2. 设|E|=m，

Kruskal算法的时间复杂性为O（$m*\log_2 m$）

Prim算法的时间复杂性为O（n^2）。

3. 当$m=\Omega(n^2)$时，Kruskal算法比Prim算法差；

但当$m=O(n^2)$时，Kruskal算法却比Prim算法好得多。

4.7 多机调度问题

1. 多机调度问题描述

- 设有n个独立的作业{1,2,…,n}，由m台相同的机器进行加工处理。作业i所需的处理时间为t_i。
- 约定：任何作业可以在任何一台机器上加工处理，但未完工前不允许中断处理。任何作业不能拆分成更小的子作业。
- 多机调度问题要求给出一种作业调度方案，使所给的n个作业在尽可能短的时间内由m台机器加工处理完成。

多机调度问题是NP完全问题

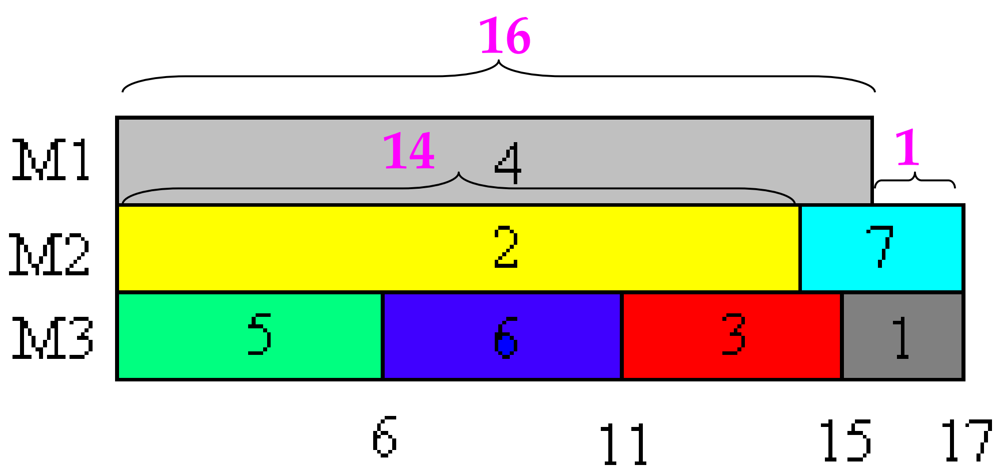
- 多机调度问题是NP完全问题
- 尚没有有效的解法
- 处理方式：近似算法
  - 对于这一类问题,可用贪心选择策略有时可以设计出较好的近似算法。

贪心选择策略

- 贪心选择策略：最长处理时间作业优先
- 按此策略
  - 当n≤m时，机器多，作业少，只要将机器i的[0,t_i]时间区间分配给作业i即可。
  - 当n＞m时，首先将n个作业依其所需的处理时间从大到小排序。然后依此顺序将作业分配给空闲的处理机。

实例

例： 设7个独立作业{1,2,3,4,5,6,7}由3台机器M1，M2和M3加工处理。各作业所需的处理时间分别为{2,14,4,16,6,5,3}。按**贪心选择策略**产生的作业调度如下图所示，所需的加工时间为17。



复杂性

- 按此策略，当n≤m时，将机器i的[0, ti]时间区间分配给作业i即可，算法只需要O(1)时间。
- 当n>m时，首先将n个作业依其所需的处理时间从大到小排序，需计算时间为O(nlog n) 。然后依此顺序将作业分配给空闲的处理机。算法所需的计算时间为O(nlog n)。
- 如采用建堆运算及堆的操作DeleteMin和Insert，本工作需时间O(nlog m)。总耗时仍为O(nlog n)

补充：一个有用的不等式

- **排序不等式**
- 设有n个非负数构成的两个数组a_1、 a_2、 … a_n与b_1、 b_2、 … b_n。设$a_1 \le a_2 \le \ldots \le a_n$与 $b_1 \le b_2 \le \ldots \le b_n$。如果$i_1$、 i_2、 … i_n是1、2、…n的一个排列，那么

$$a_1b_n + a_2b_{n-1} + \ldots + a_nb_1$$

$$\le a_1b_{i_1} + a_2b_{i_2} + \ldots + a_nb_{i_n}$$

$$\le a_1b_1 + a_2b_2 + \ldots + a_nb_n$$