算法设计与分析

第4章 贪心算法
(3)

谢晓芹
哈尔滨工程大学计算机科学与技术学院

学习要点

- 理解贪心算法的概念
- 掌握贪心算法的基本要素
  - 最优子结构性质
  - 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

学习要点

- 通过应用范例学习贪心设计策略
  - 活动安排问题
  - 哈夫曼编码
  - 最优装载问题
  - 单源最短路径
  - 最小生成树

最优装载

- 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
- 问题描述
  - 输入: n个集装箱, 重为w_i, 轮船载重量为c.
  - 输出: $(x_1, x_2, \ldots, x_n)$, 满足以下条件

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

$$\sum_{i=1}^{n} w_i x_i \leq c$$

$$x_i \in \{0,1\}, 1 \leq i \leq n$$

最优装载

- 分析
  - 与0-1背包问题的区别？

	最优装载	**0-1背包**
相同点	重量受限	重量受限
	集装箱要么装，要么不装	物品要么装，要么不装
目标不同	总数目最大	总价值最大
贪什么不同	贪重量最小	贪单位重量价值最大 **（分数背包）**

最优装载

- 贪心算法描述:
  - 采用重量最轻者先装的贪心选择策略

```
void Loading(int x[],  Type w[], Type c, int n){
    int *t = new int [n+1];
    Sort(w, t, n);
    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;    c -= w[t[i]];}
}
```

t:按重量排序后的物品序列.
核心： 建立排序前后关系映射表

c:当前背包重量限制

新编号 i	1	2	3	4	5
原始编号 t[i]	5	1	3	4	2

最优装载

- 举例： n=5, C=300斤
  - 物品编号 i = 1, 2, 3, 4, 5
  - 物品重量wi =30, 70, 50, 60, 10
  - 新排名：2， 5， 3，4，1

按重量从轻到重排序的编号：5，1，3，4，2

新编号 i	**1**	**2**	**3**	**4**	**5**
排序后编号	2	5	3	4	1

按t[i]取得顺序： t[i]={5,1,3,4,2 }

最优装载

- 复杂性和正确性
  - 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(nlogn)。
  - 由最优装载问题的贪心选择性质和最优子结构性质，容易证明算法loading的正确性。

最优装载

- 贪心选择性质
  - 贪什么：贪重量最小的集装箱
  - 要证明：存在一个最优解，其包含贪心选择的结果
    - 分析：构造出一个解，然后证明其是最优解（是解：满足约束条件， 最优：货品个数最多）
- 证：设贪心选择的结果k= $\min_{1\le i\le n}\{i|x_i=1\}$, 这是选装的第1个集装箱
  - 1) k=1时，$(x_1,x_2,\ldots,x_n)$是满足贪心选择性质的最优解
  - 2) k>1时, 对$(x_1,x_2,\ldots,x_n)$做修改，改成贪心最优解

    取$y_1=1,\ y_k=0,\ y_i=x_i,\ 1\le i\le n,\ i\ne k$

  有：$\sum_{i=1}^{n} w_i y_i=\sum_{i=1}^{n} w_i x_i+w_1-w_k\le\sum_{i=1}^{n} w_i x_i\le c$ （因为$w_1\le w_k$）

  所以，$(y_1,y_2,\ldots,y_n)$是所给装载问题的一个贪心选择得到的可行解

最优装载

- 最优子结构性质
  - 原问题：假设已排好序的n个集装箱 {1,2,…,n}，　≤C
    - 满足贪心选择的最优解：（ $1, x_2, \ldots, x_n$ ）
  - 子问题: {2,3,…, n}，　≤C-w_1
    - 最优解？　（ $x_2, \ldots, x_n$ ）？

- 反证+构造法

单源最短路径

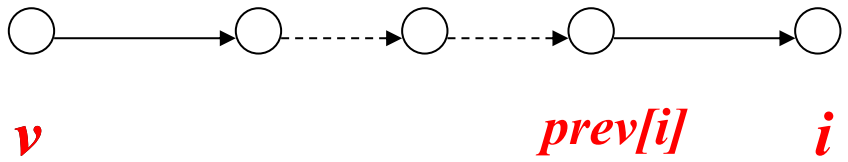
- 带权有向图G =(V,E)，其中每条边的权c[i][j]是**非负**实数。还给定V中的一个顶点，称为源
- 单源最短路径问题:
  - 是计算从源到所有其它各顶点的最短路长度，这里路的长度是指路上各边权之和
- 算法基本思想
  - Dijkstra算法是解单源最短路径问题的贪心算法
  - 设置顶点集合S，作贪心选择来扩充这个集合。一个顶点属于集合S当且仅当从源到该顶点的最短路径长度已知

最短特殊路经长度

单源最短路径

- 输入
  - 带权有向图G=(V,E),源顶点v, c[i][j]表示边ij的权
- 输出
  - 从源v到**所有其它各点**的最短路径长度
  - dist[i]:从v到i的最短（特殊）路径长度
  - Prev[i]: 保存从v到各顶点之间最短路径上的各边的连接信息,即最短路径上i的前一个顶点

单源多点最短路径问题

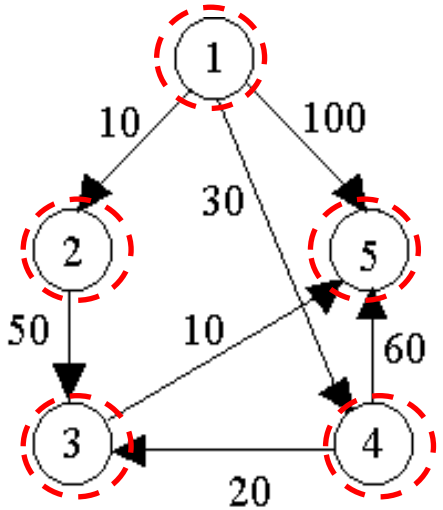


单源最短路径

- Dijkstra算法步骤
  - 初始时，S中仅含有源点*v*
  - 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
    - 设u是G的某一个顶点，把从源v到u且中间只经过S中顶点的路称为从源v到u的特殊路径，并用数组dist记录当前每个顶点所对应的最短特殊路径长度
  - 结束条件:当S包含了V中所有顶点,dist就记录了从源到所有其它顶点间的最短路径长度

贪什么

特殊: 路径上的点是所有贪心选择后的点



迭代	S	u	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	-	10	+∞	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	70
4	{1,2,4,3,5}	5	10	50	30	70

单源最短路径

- 分析：
  - 最优子结构：最短路径包含的子路径也是一条最短路径
  - 贪心策略
    - 设置顶点集合S, 作**贪心选择**来扩充这个集合，初始只包含{s}
    - 每一步把某结点$v \in V - S$加入S，保证从s到v 的特殊路径长度最小
    - 更新与v邻接的点的最短路径长度

    - 贪什么
    - 怎么贪
    - 何时结束

单源最短路径

```
Dijkstra(){
    dist[s]← 0;
    for each v ∈ V − {s} do
        dist[v] ← ∞
    S ← ∅;
    Q ←V;
    while Q!=∅ do{
        u←ExtractMin(Q);
        S←S∪{u};
        for each v ∈ Adj[u] do{
            if dist[v] > dist[u]+w(u,v) then
                dist[v]=dist[u]+w(u,v);
        }

    }
}
```

用极小堆Q来组织 V-S 结点，key值为dist[]

初始化

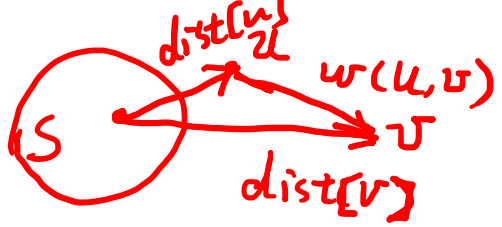
贪什么

取点　取最小值实现

加点

改权

Adj[u]：所有和u相邻的结点



T(n)=O(nlogn)

单源最短路径

```
Dijkstra(int n, int v, long dist[],int prev[], long **c){
      ……//初始化
     for(int i=1; i<n; i++){   //何时终止                                  O(n)
        for (int j=1; j<=n;j++){      //1）贪心选点                       O(n)
           if (!s[j]&&(dist[j]<temp)){   //特殊路径最短
              u=j;  temp=dist[j]
           }
        }
        s[u]=true;                          //2）加点
        for(int j=1; j<=n;j++){       //3）改权                           O(n)
           if((!s[j])&&(c[u][j]<maxint))  { //从u到j有路径
              long newdist=dist[u]+c[u][j];
              if(newdist<dist[j]){dist[j]=newdist; pre[j]=u}
           }
        }
     }
  }
```

用列表来组织 V-S 结点

找Min

顺序扫描的方式

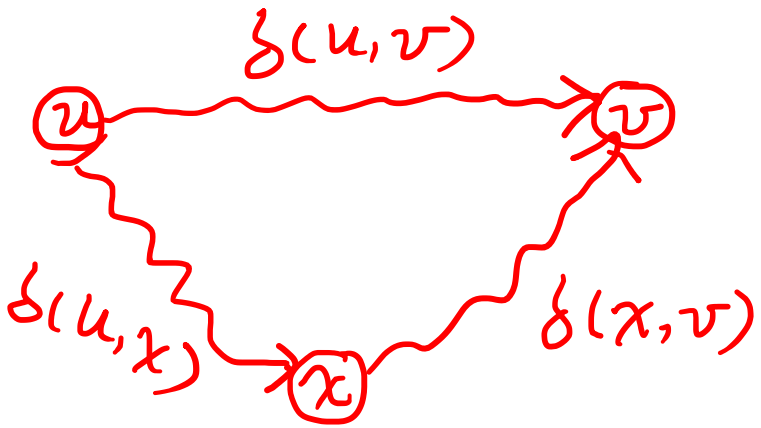
j点没有被选过

松弛技术

$T(n)=O(n^2)$

单源最短路径

- 松弛技术
  - 三角不等式：对于任意顶点 u,v,x ∈V. 设$\delta(u,v)$表示u到v的最短路径，有$\delta(u,v) \leq \delta(u,x) + \delta(x,v)$



单源最短路径

- 算法的复杂性
  - Dijkstra算法运行时间依赖于具体数据结构
  - 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$O(n^2)$

单源最短路径

```
Dijkstra(){
        dist[s]← 0;
        for each v ∈ V − {s} do
            dist[v] ← ∞
        S ← ∅;
        Q ←v;
        while Q!=∅ do{
            u←ExtractMin(Q);
            S←S∪{u};
            for each v ∈ Adj[u] do{
                if dist[v] > dist[u]+w(u,v) then
                        dist[v]=dist[u]+w(u,v);
            }

        }
}
```

分析：

初始化

取点 $|V|*T_{extract}$

加点

改权 $|E|*T_{降低dist时间}$

$T(n)= V|*T_{extract}+|E|*T_{降低dist时间}$

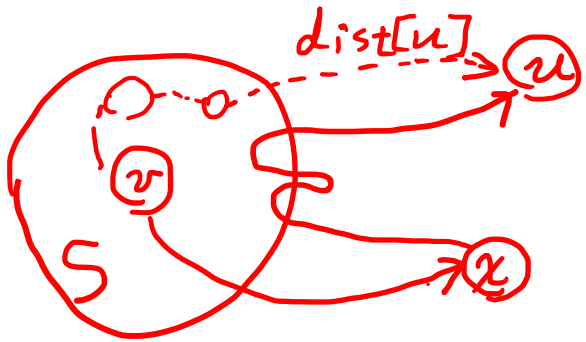
单源最短路径

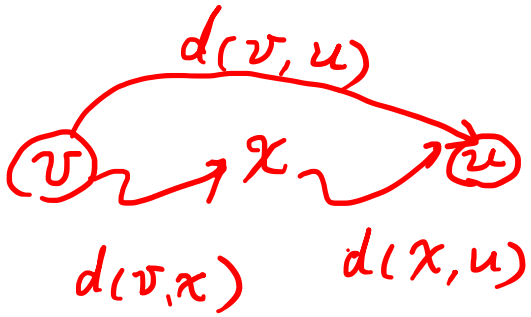
- 算法的正确性
  - (1)贪心选择性质
  - (2)最优子结构性质

单源最短路径

- 贪心选择性质证明，即是要证明贪心选择的结果能得到一个最优解，或：存在一个最优解，其最短路径长度为dist[u]
- 反证法
  - 假设dist[u]不是从源到顶点u的最短路径长度
  - 设有一条从源到u的更短路，v→x→u, 分别记路长为:d(v,x),d(x,u),d(v,u)，x为S之外的点， 有
    - dist[x]<=d(v,x)
    - d(v,x)+d(x,u)=d(v,u)<dist[u]

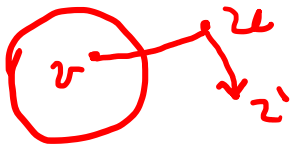
      因为d(x,u)>0 =>dist[x]<=d(v,x) <= d(v,x)+d(x,u)<dist[u]，产生矛盾，假设错误。

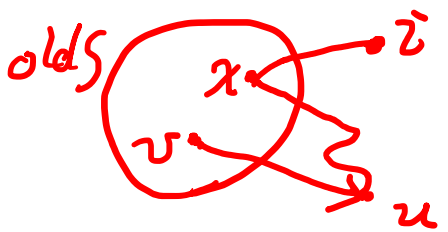




单源最短路径

- 最优子结构性质证明
  - 原问题：从源点到所有其他各点的最短路径长度
  - 子问题：从源点到一个顶点u的最短路径长度
  - 问题转化为:算法在添加u到s中之后，dist[i]的值会发生什么变化？
- 分析证明法：当u加到S后，有两种情况
  - 1）从u到i有直接边：newpath=v->u->i,该路径最短为：dist[u]+c[u][i]
    - 若dist[u]+c[u][i]<dist[i],则替换
  - 2）从u到i没有直接边：从oldS到u再经oldS中某点x，才到i
    - 因为x比u先加入S，Len(v->x)<len(v->u->x)
    - dist[i]<len(v->x->i)(根据dist定义)
    - 有：dist[i]<len(v->u->x->i)
    - 此类路径不考虑
  - 不管dist[i]是否变化，它总是关于顶点集S到顶点i的最短路径。





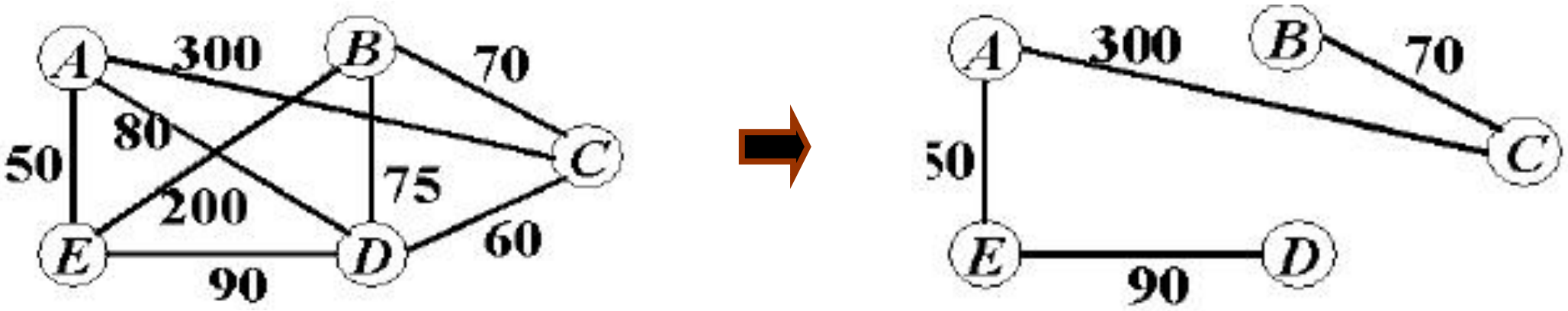
最小生成树

- 问题的提出
  - 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，则需要给出了建立通信网络的最经济的方案--最小生成树
- 网络的最小生成树在实际中有广泛应用

最小生成树

- 定义1（生成树）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T
- 生成树上各边权的总和称为该生成树的耗费W(T)

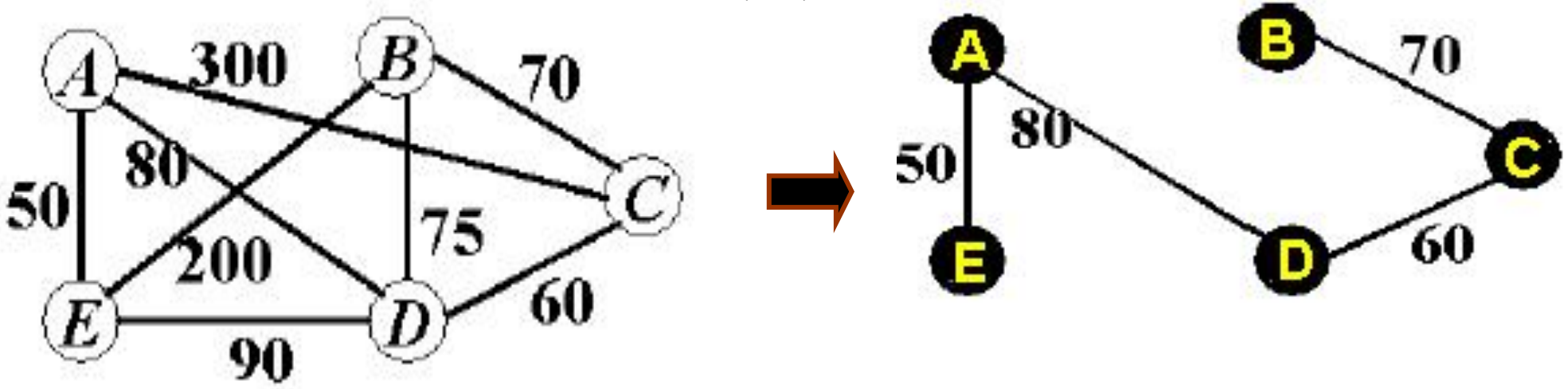
结点之间连通，没有环路



最小生成树

- 定义2 最小生成树（Minimal Spanning Tree, MST） 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
- 最小生成树问题
  - 输入: 连通无向图G=(V,E),加权函数W:E→R, 假设所有边权值是唯一的
  - 输出: 一棵生成树T（连接了所有点且具有**最小**权值）

$$W(T) = \sum_{(u,v)\in T} w(u,v)$$



最小生成树

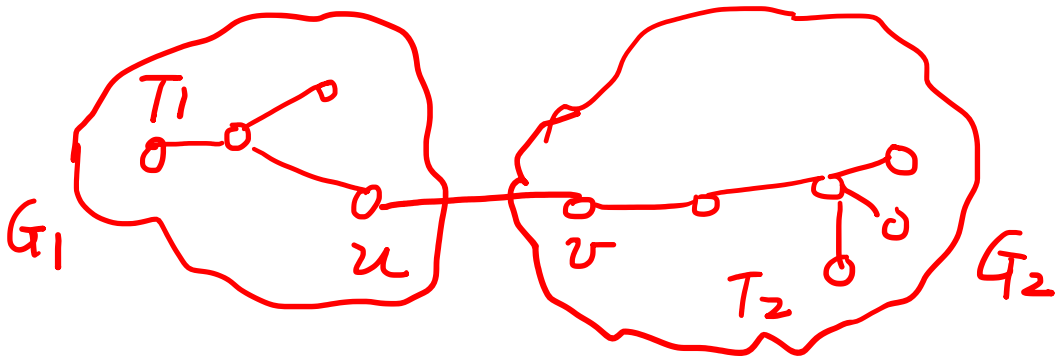
- 枚举法
  - 列举出所有生成树，计算各个生成树的权重和，选出最小的
  - 图中的每条边要么在生成树上，要么不在，有2种可能。
  - 图共有|E|条边
  - 列举出所有生成树的复杂度是O($2^{|E|}$)
- 动态规划算法？

最小生成树

- 最优子结构性质
  - 原问题：G
    - 其MST为T
  - 划分：
    - T中的一条边(u,v), 将T分为T_1和T_2
  - 子问题：G_1（ 由T_1中的顶点导出的G的子图）
    - 其MST为T_1' ？
- 要证明：T_1是$G_1=(V_1, E_1)$的MST
- 反证法
  - $w(T)=w(u,v)+w(T_1)+w(T_2)$

  若存在T_1'优于T_1，是G_1的MST

  有： $T' = \{u,v\} \cup T_1' \cup T_2$比T更优，产生矛盾。



V_1=T_1的所有结点

$E_1=\{(x,y)\in E|x,y \in V_1\}$

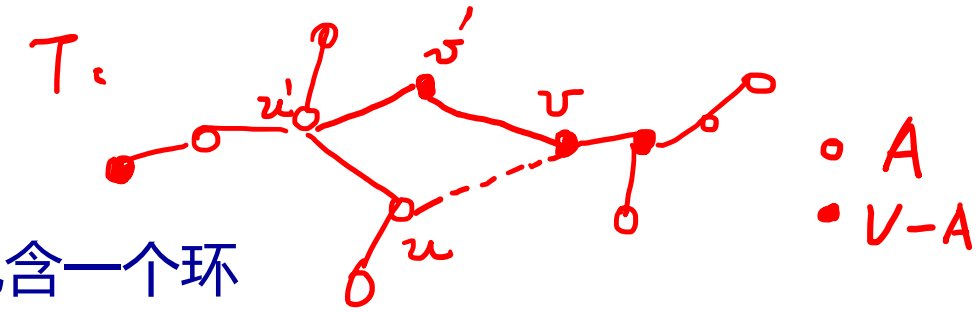
最小生成树

- 最小生成树性质(MST性质): 贪心选择性质
  - 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
- 分析
  - 贪什么： 权值最小， c[u][v]最小
  - 贪心选择得到的解包含(u,v)边
  - 证明: 反证法, Cut-paste法

最小生成树

- 最小生成树性质 (MST性质)
  - 设G=(V,E)是连通带权图, S是V的真子集.如果(u,v) ∈E,且u ∈S,v ∈V-S, 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.

➢ 证：1) 如果(u, v)属于T，证毕。



2) 假设(u, v)不属于T，

则添加(u, v)到最小生成树T中，导致T包含一个环

删除与(u, v)形成环的边(u’, v’),得到另一个生成树T’

T’ =T - {(u’, v’)} ∪{(u, v)}。只需证T’是最小生成树。

因为c(u, v)≤c(u’, v’)

W(T’)=W(T)- c(u’, v’) + c(u, v)≤W(T)

又因为T最优，有W(T)≤W(T’)

所以，W(T’)=W(T)。于是，T’是最小生成树

最小生成树

- 构造MST的贪心算法
  - Prim算法
  - Kruskal算法

最小生成树-Prim算法

- 设G=(V,E)是连通带权图，V={1,2,…,n}
- Prim算法
  - 基本思想

贪心性：每次添加到树中的边都是使树的权尽可能小的边

  - 首先置集合S={1}
  - 然后，只要S是V的真子集，就作如下贪心选择：

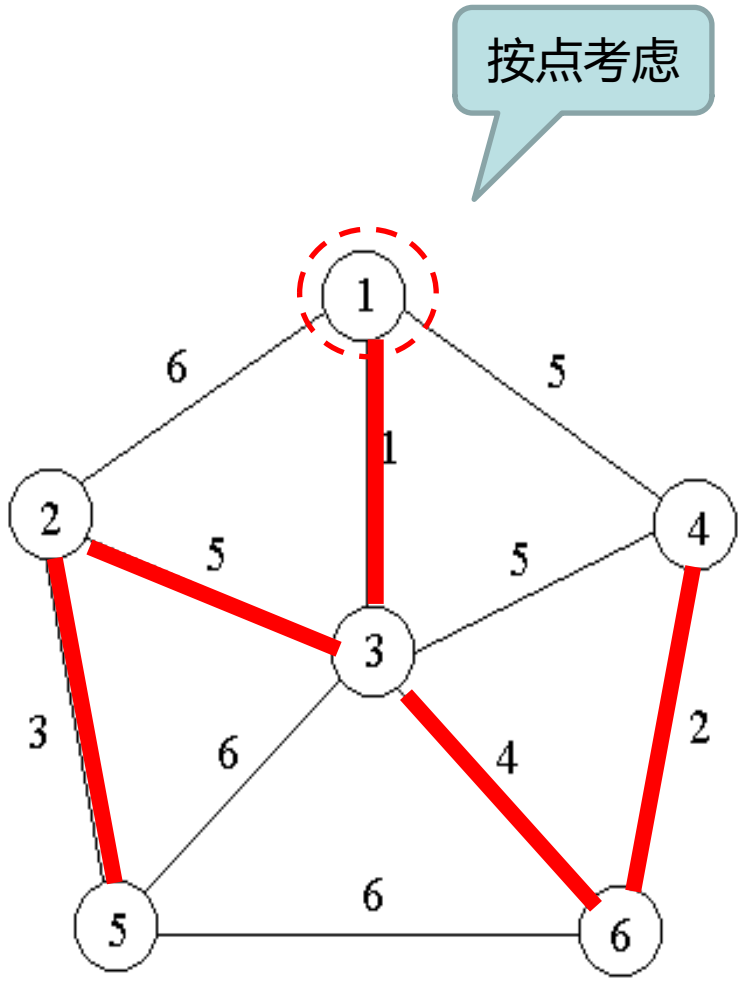
贪什么

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

怎么贪

  - 贪心过程
    - 选边（点）
    - 加点
    - 改值
  - 何时终止： 所有点都加完

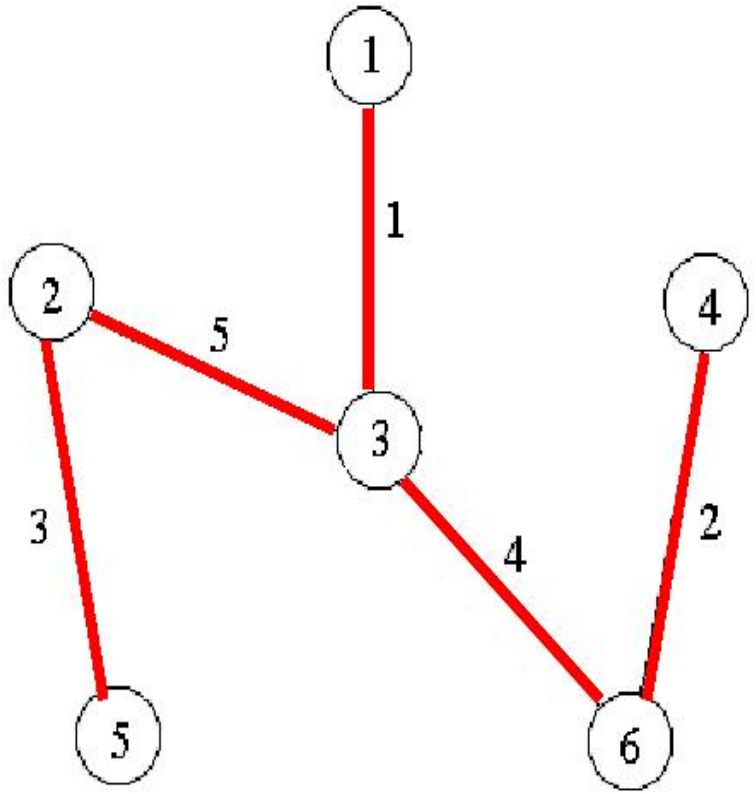


最小生成树-Prim算法

- Prim算法

输入：G=(V,E), c[][]
输出： T

```
void Prim(G, float c[][])
    { T=ϕ;
      S={1};
      while(S!=V){
          (i,j)=i∈S且j∈V-S 的最小权边; //选边(点)
          T=T∪{(i,j)};   //加边, T为边集合
          S=S ∪{j};      //S为点集合
      }//end of while
    }
```



最小生成树-Prim算法

- Prim算法

输入：G=(V,E), c[][]
输出： T

```
void Prim(G, float c[][])
    { T=ϕ;
      S={1};
      while(S!=V){
          (i,j)=i∈S且j∈V-S 的最小权边; //选边(点)
          T=T∪{(i,j)};   //加边, T为边集合
          S=S ∪{j};      //S为点集合
      }//end of while
    }
```

分析： 采用堆管理结点

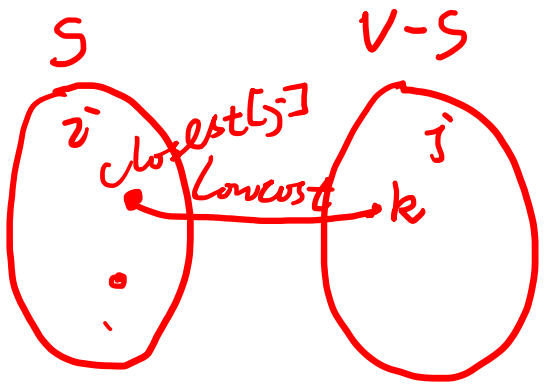
O(|V|)

O(log|V|)

O(|V|log|V|)

最小生成树-Prim算法

- 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)?
  - 选择数据结构：设置2个数组closest和lowcost
  - closest是 j在S中的一邻接顶点, j属于V-S
  - c[j][closest[j]] ≤c[j][k]
  - lowcost[j]=c[j][closest[j]]



- Prim算法执行过程

三步曲：选边(点)

- 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])

加点

- 将j添加到S中

改值

- 并对closest和lowcost作必要的修改

动态规划里有贪心的思想吗？

- 该办法实现的Prim算法的计算时间为O(n^2) (前提：使用数组)

N=|V|

最小生成树-Kruskal算法

- Kruskal算法基本思想

贪心性：添加到森林中的边的权值尽可能小

  - 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

1贪什么

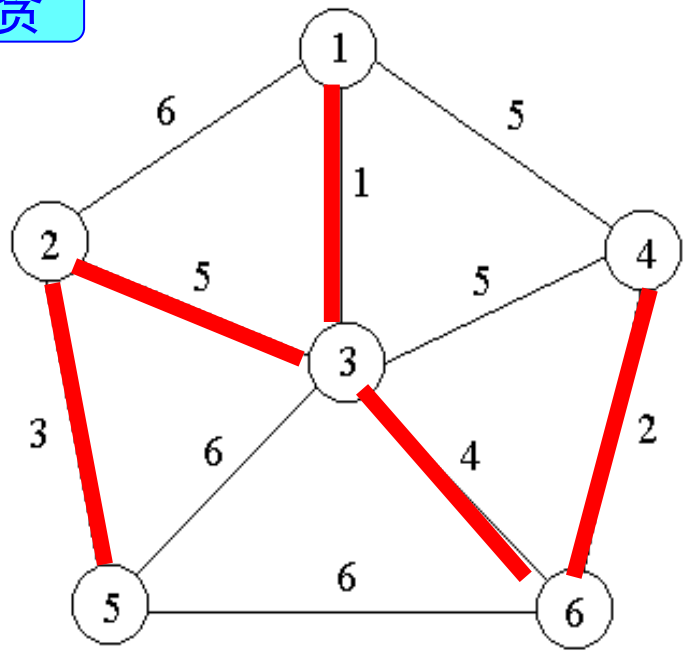
  - 从第一条边开始，依边权递增的顺序查看每一条边，并按下述方法连接2个不同的连通分支：

2怎么贪

    - 1) 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T_1和T_2中的顶点时，就用边(v,w)将T_1和T_2连接成一个连通分支，然后继续查看第k+1条边
    - 2) 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
    - 3) 这个过程一直进行到只剩下一个连通分支时为止

3 何时终止

按边考虑



最小生成树-Kruskal算法

输入: 连通带权图G, 权W
输出: 集合A (存储最小生成树的边的集合)

```
MST-Kruskal(G,W){
1. A=Φ;
2. For ∀v⊆ V[G] Do
3.        Make-Set(v);
4. 按照W值的递增顺序排序E[G];
5. For ∀(u, v) ⊆ E[G] (按W 值的递增顺序) Do
6.        If Find-Set(u)≠Find-Set(v)
7.        Then  {A=A∪{(u, v)}; Union(u, v); }
8. Return A
}
```

n: 节点个数
m: 边个数

第2-3 步执行*O(n)*个*Make-Set* 操作（初始化）

第4 步执行*O(mlogm)*排序操作（贪什么）

第5-7步执行*O(m)*个*Find-set*和*Union*操作(怎么贪)：选边，加点，改值

第6步:判断是否属于同一连通分支

最小生成树-Kruskal算法

- Kruskal算法时间复杂度分析
  - 当图的边数为e时，Kruskal算法所需的计算时间是O(eloge)。
  - 当$e=\Omega(n^2)$时（边多时）
    - Kruskal算法差，Prim算法好
  - 当$e=O(n^2)$时（边少时）
    - Kruskal算法好，Prim算法差

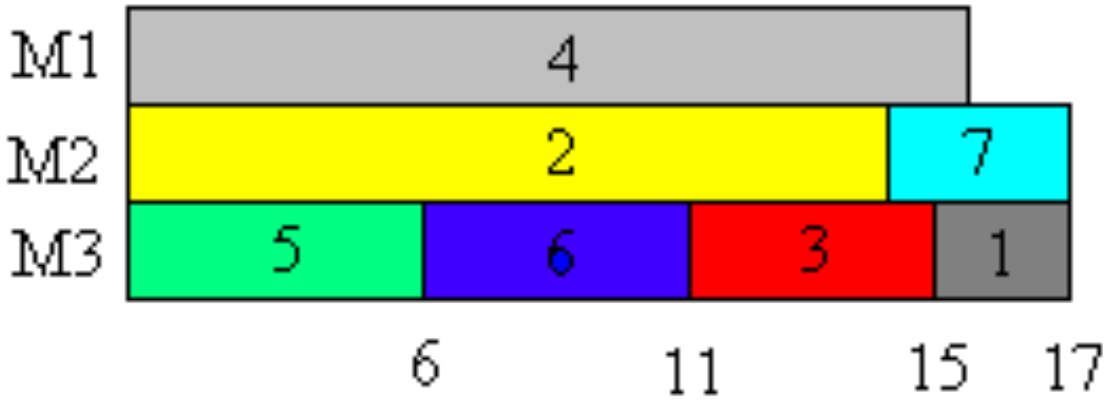
多机调度问题

- 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。每个作业均可在任何一台机器上加工处理，但未完工前不允许中断处理。作业不能拆分成更小的子作业
- 分析
  - 这个问题是NP完全问题，到目前为止还没有有效的解法。对于这一类问题,用贪心选择策略有时可以设计出较好的近似算法
  - 采用最长处理时间作业优先的贪心选择策略可以设计出解多机调度问题的较好的近似算法
    - 当作业数n≤机器数m 时，只要将机器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。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(nlogn)

n: 独立作业个数
m: 机器个数

多机调度问题

- 例如，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}。



作业编号	1	2	3	4	5	6	7
处理时间	2	14	4	16	6	5	3
按时间排序	7	2	5	1	3	4	6

➢ 按**贪心**算法产生的作业调度所需的加工时间为17

贪心算法的理论基础

- 借助于拟阵工具，可建立关于贪心算法的较一般的理论。这个理论对确定何时使用贪心算法可以得到问题的整体最优解十分有用
- 定义——拟阵(Matroid)
  - 拟阵M定义为一个序对(S,I)，满足下面的条件：

(1)S是非空有限集。

(2)I是S的子集的集族，I中的子集合称为S的独立子集合

(3)遗传性：若$B \in I, A \subseteq B$,则$A \in I$。空集$\varnothing$必为I的成员。

(4)交换性：若$A \in I, B \in I$且$|A|<|B|$，则$\exists x \in B-A$，使得$A \cup \{x\} \in I$。

贪心算法的理论基础

- 定义—图拟阵(Graphic Matroid)
  - 设G=(V，E)是一个无向图，由G 确定的图拟阵MG=(SG,IG)定义如下：SG是G 的边集合E，IG={A | A $\subseteq$ E, (V, A)是一个森林。
- 定义—可扩展元素
  - 给定拟阵M=(S，I)，对于I中的独立子集A$\in$I，若S有一元素x$\notin$ A，使得A$\cup${x}$\in$ I,则称x为A的可扩展元素。
- 定义—极大独立子集
  - 当拟阵M中的独立子集A没有可扩展元素时，称A为极大独立子集。

贪心算法的理论基础

- 定义—图拟阵(Graphic Matroid)
  - 设G=(V，E)是一个无向图，由G 确定的图拟阵MG=(SG,IG)定义如下：SG是G 的边集合E，IG={A | A ⊆ E}, (V, A)是一个森林。 A是无回路的
- 定义—可扩展元素
  - 给定拟阵M=(S, I)，对于I中的独立子集A∈I，若S有一元素x∉ A，使得A∪{x}∈ I, 则称x为A的可扩展元素。
- 定义—极大独立子集
  - 当拟阵M中的独立子集A没有可扩展元素时，称A为极大独立子集。

定理4.1：拟阵M中所有极大独立子集大小相同。

证：反证法证明。

贪心算法的理论基础

- 定义——带权拟阵
  - 对于拟阵M=(S, I)，若存在权函数W，使得∀x∈S，W(x)>0，则称拟阵M为带权拟阵。依此权函数，S的任一子集A的权定义为：

$$W(A)=\sum_{x\in A}W(x)$$

- 定义——最优子集
  - 给定带权拟阵M=(S,I)，确定S的独立子集A∈I，使得W(A)达到最大。这种使W(A)最大的独立子集A称为拟阵M的最优子集
- 由于S中任一元素x的权W(x)是正的，因此，最优子集也一定是极大独立子集。

✓ 许多可以用贪心算法求解的问题都可以归结为求带权拟阵的**最优子集问题。**

小结

- 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
- 案例分析
  - 最优装载问题
  - 单元最短路径问题
  - 最小生成树问题

实践创新能力训练

- 会场安排问题：
  - 假设要在足够多的会场里安排一批活动，并希望使用尽可能少的会场。请设计有效的贪心算法。

- 提示：
  - 输入：给定k个活动，每个活动有一个开始时间和终止时间si，fi，分别排序s1<s2<…<sn，f1<f2<…,<fn
  - 输出：给出使用最少会场场数和相应的时间表