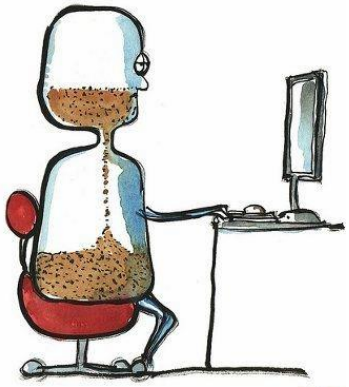
数据结构与算法分析



人工智能与自动化学院 陶文兵

wenbingtao@hust.edu.cn

贪心算法

- 贪心算法的概念
- 贪心算法的基本要素
  - (1) 最优子结构性质
  - (2) 贪心选择性质
- 贪心算法与动态规划算法的差异
- 通过例子学习贪心算法策略
  - (1) 活动安排问题
  - (2) 最优装载问题;
  - (3) 哈夫曼编码;
  - (4) 最小生成树;
  - (5) 单源最短路径;
  - (6) 多机调度问题

贪心法基本思想

- 作出在当前看来最好的选择
  - 非整体考虑，而是某种意义上的局部最优选择
  - 但希望最终达到全局最优
  - 并非所有问题可得到整体最优解，但对许多问题它能产生整体最优解，如：单源最短路经问题、最小生成树问题等
  - 在一些情况下，即使贪心算法不能得到整体最优解，其最终结果却是最优解的很好近似

活动安排问题–贪心算法的示例

- 问题简述：要在所给的活动集合中选出最大的相容活动子集合
  - 要求高效地安排一系列争用某一公共资源的活动
  - 贪心算法可有效求解

- 贪心算法求解：
  - 使得尽可能多的活动能兼容地使用公共资源
  - 简单、漂亮

活动安排问题

- 问题定义：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，如演讲会场等，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
- 目标：选择最多的互不冲突的活动，使相容活动集合最大找最大相容子集合

最多三个事件

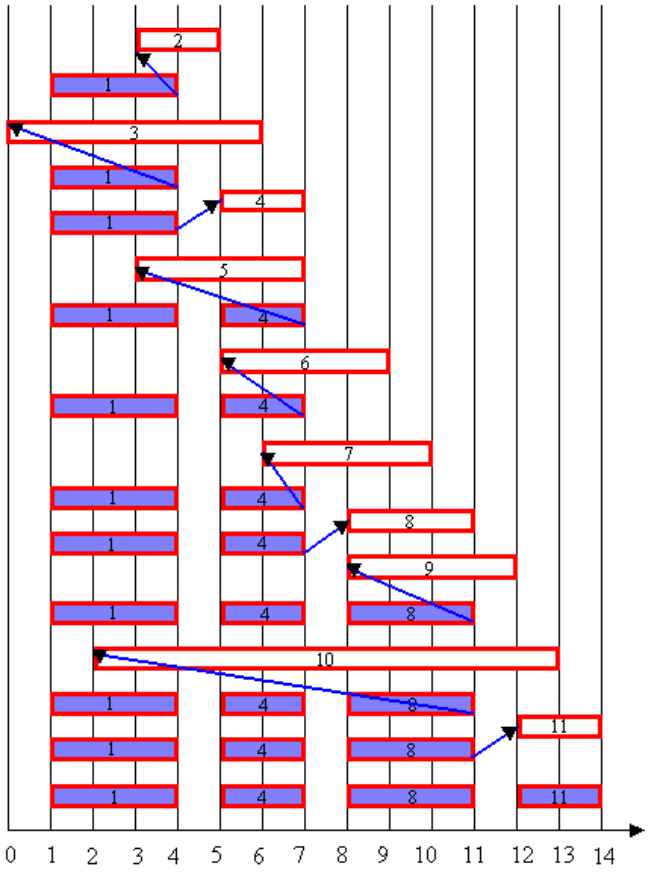
活动安排问题

- 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



- 算法greedySelector 的计算过程如左图所示
- 每行相应于算法的一次迭代
- 阴影长条表示的活动是已选入集合A的活动，而空白长条表示的活动是当前正在检查相容性的活动。
- 若被检查的活动i的开始时间Si小于最近选择的活动j的结束时间fj，则不选择活动i，否则选择活动i加入集合A中

贪心算法并不总能求得问题的**整体最优解**。但**对于活动安排问题，贪心算法greedySelector却总能求得整体最优解，即它最终所确定的相容活动集合A的规模最大**。这个结论可以用数学归纳法证明。

活动安排问题

- 算法分析
  - **输入的活动按照其完成时间的非减序排列**，所以算法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**。直观上，**按这种方法选择相容活动为未安排活动留下尽可能多的时间。**也就是说，**该算法的贪心选择的意义是使剩余的可安排时间段极大化，以便安排尽可能多的相容活动**
  - 算法greedySelector的效率极高
    - 当输入的活动已按结束时间的非减序排列，算法只需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
    - 如果所给出的活动未按非减序排列，可以用O(nlogn)的时间重排

活动安排问题

- 贪心算法实现-GeedySelector

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

各活动的起始时间和结束时间存储于数组s和f中且按结束时间的非减序排列。

贪心算法的基本要素

对于一个具体的问题，怎么知道是否可用贪心算法解此问题，以及能否得到问题的最优解呢?

- 很难给予肯定回答
- 但可分析问题是否具备两个基本性质
  - 贪心选择性质
  - 最优子结构性质

贪心算法的基本要素

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，也是贪心算法与动态规划算法的主要区别。

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

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。

问题的最优子结构性质是该问题可用动态规划算法或贪心算法求解的关键特征。

活动安排问题贪心算法正确性

(1)贪心选择性质

首先证明活动安排问题有一个最优解以贪心选择开始，即该最优解中包含活动1。

设A $\subseteq E$是所给活动安排问题的一个最优解，且A中活动也按时间结束顺序时间非减序排列，A中第一个活动是k，若k=1则结论成立。

若k>1，则设B=A-{k}∪{1}。由于$f_1 \leq f_k$，且A中活动是相容的，故B中活动也是相容的。由于A和B活动数相同，故B和A一样也是最优活动安排。因此**总存在以贪心选择开始的最优解**，结论成立！

(2)最优子结构性质

需要证明：若**A**是原问题的最优解，则**A'** =A-{1}是活动安排问题$E' = \{i \in E: s_i \geq f_1\}$的最优解。

如果能够找到的一个解**B'**，比**A'**包含更多的活动，则将1加入**B'**中将产生E的一个解**B**，它包含比**A**更多的活动。**这与A最优产生矛盾。**

因此，每一步所作出的贪心选择都将问题简化为一个更小的与原问题具有相同形式的子问题。

对贪心选择次数采用**数学归纳法**即知：**贪心算法最终产生原问题的最优解。**

最小生成树

对于一个无向网络——无向加权连通图N=(V,E,C)(C表示该图为权图)，其顶点个数为|V|=n，图中边的个数为|E|，我们可以从它的|E|条边中选出n-1条边，使之满足

（1）这n-1条边和图的n个顶点构成一个连通图。

（2）该连通图的代价是所有满足条件(1)的连通图的代价的最小值。

这样的连通图被称为网络的**最小生成树**(Minimum-cost Spanning Tree)。

1. Prim贪心算法基本步骤

普里姆(Prim)算法(逐点加入)

设N=(V,E,C)为连通网，TE是N的最小支撑树MST的边的集合，U为MST顶点集。

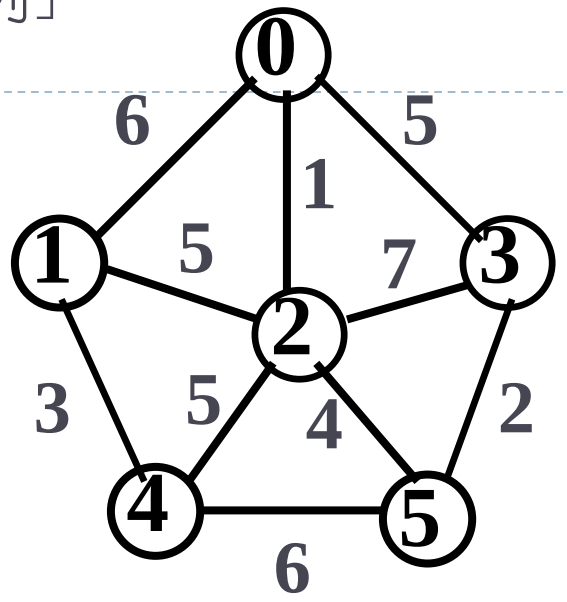
① 初始设$U=\{u_o\}(u_o \in V)$, $TE=\Phi$；

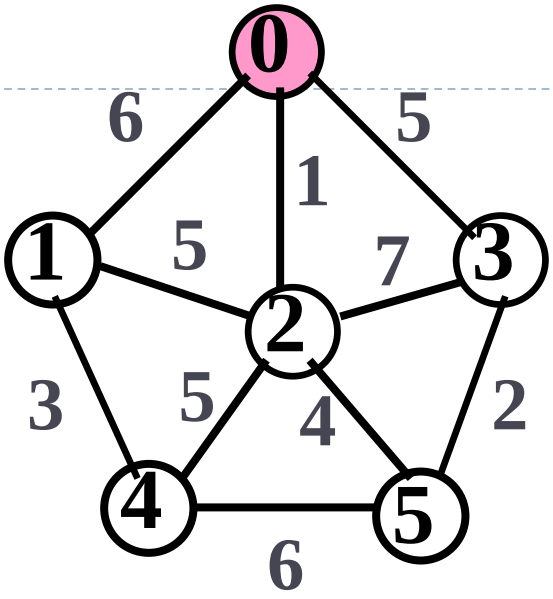
② 找到满足

$weight(u,v)=\min\{weight(u_1,v_1)|u_1 \in U, v_1 \in V-U\}$，的边，把它并入TE，同时v并入U；

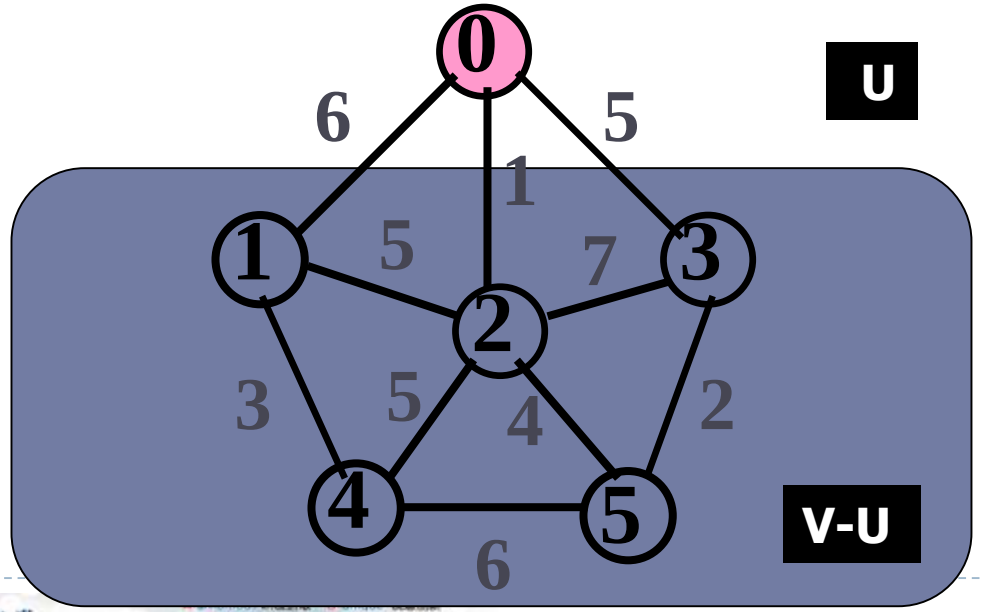
③ 反复执行②，直至 V=U，则T=(V,{TE})为N的最小生成树，算法结束。

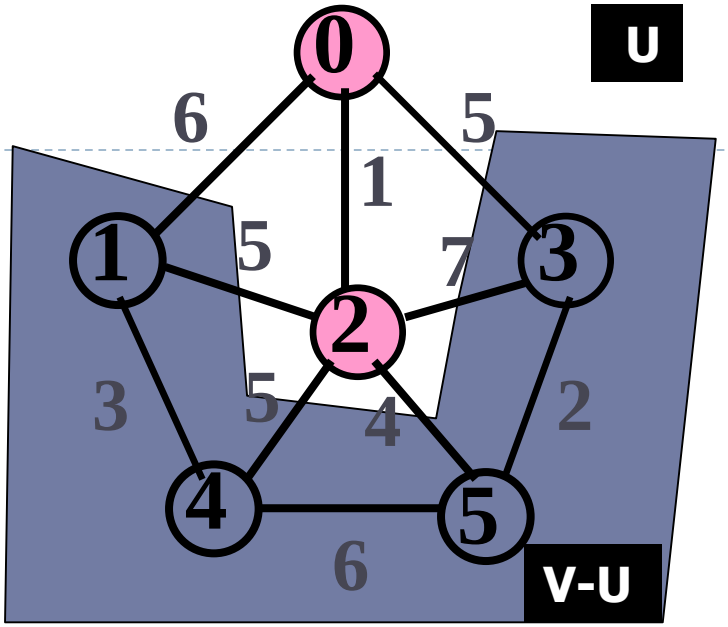
[例]


0
1
2
3
4
5
6
5
1
5
7
3
5
4
2
6

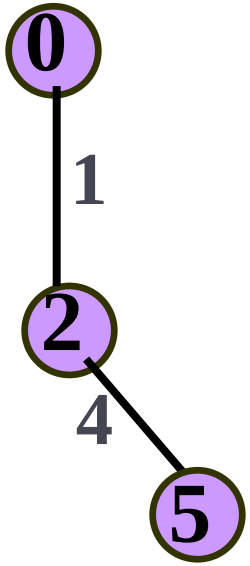

0
1
2
3
4
5
6
5
1
5
7
3
5
4
2
6

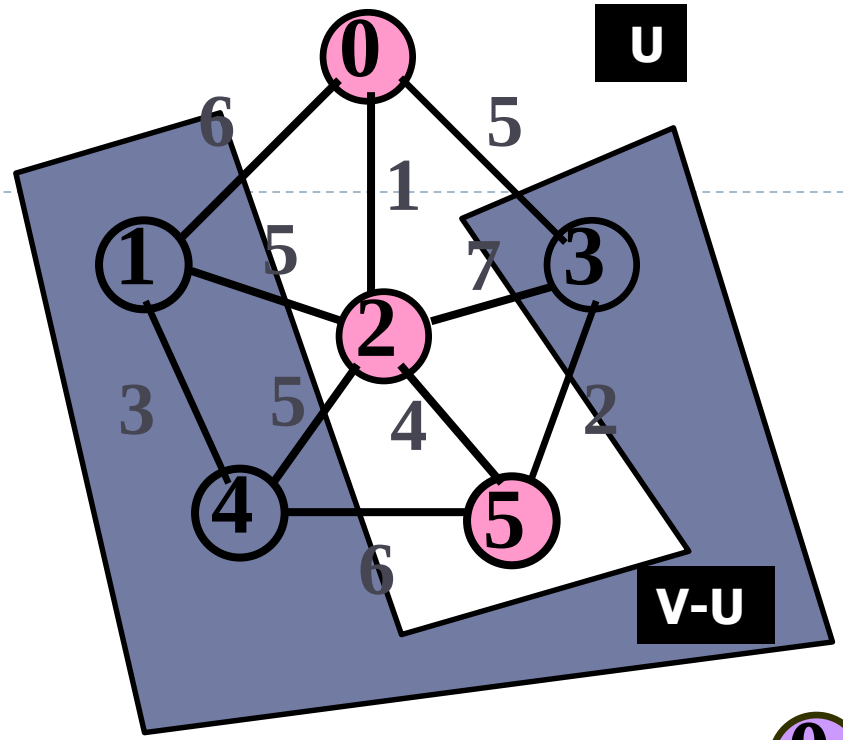

0
1
2

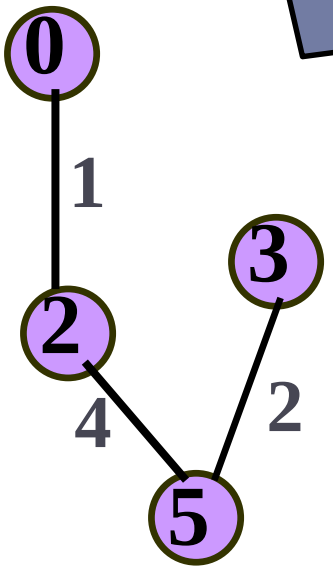

U
V-U
0
1
2
3
4
5
6
5
1
5
7
3
5
4
2
6

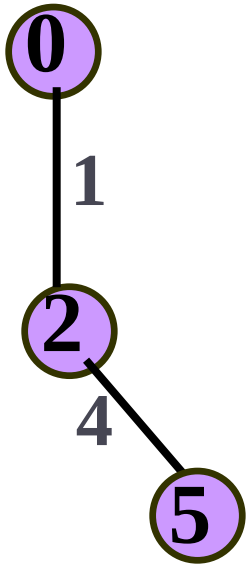

U
V-U
0
1
2
3
4
5
6
5
1
5
7
3
5
4
2
6

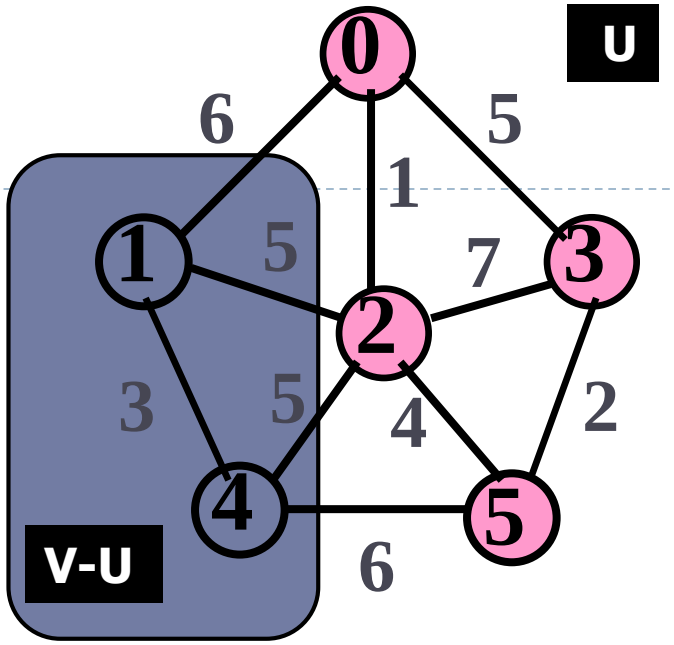

0
1
2

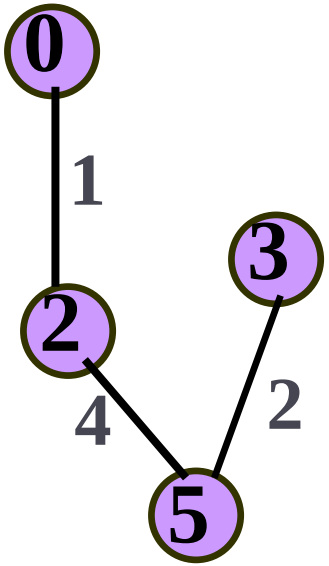

0
1
2
4
5

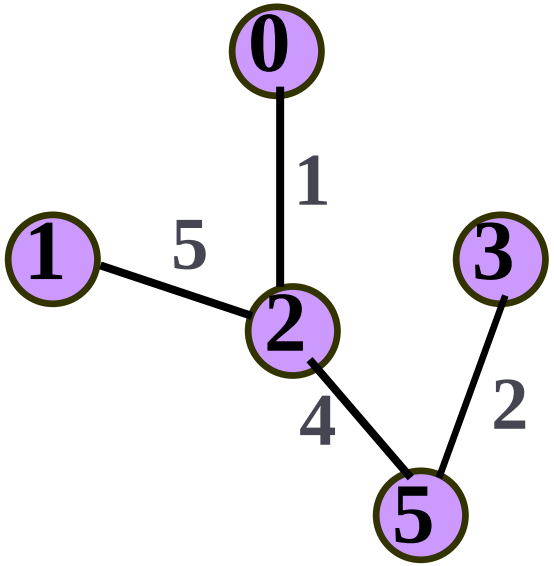

U
V-U
0
1
2
3
4
5
6
5
1
5
7
3
5
4
2
6

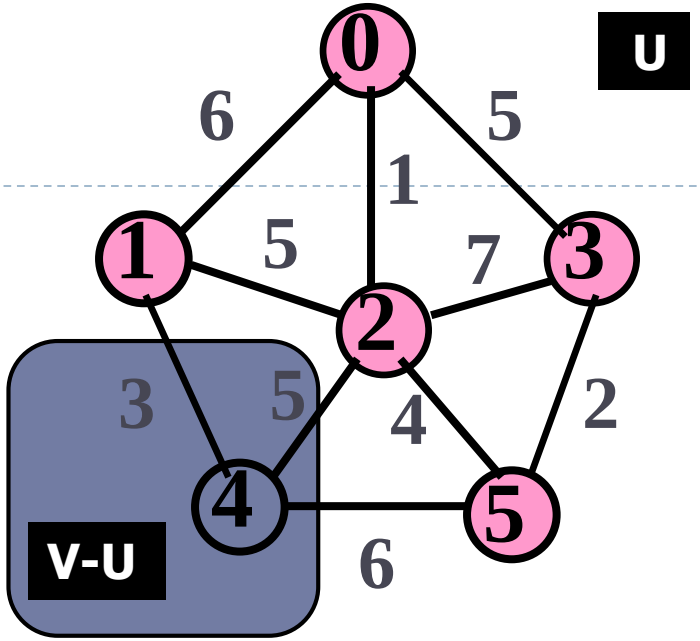

0
1
2
3
4
5

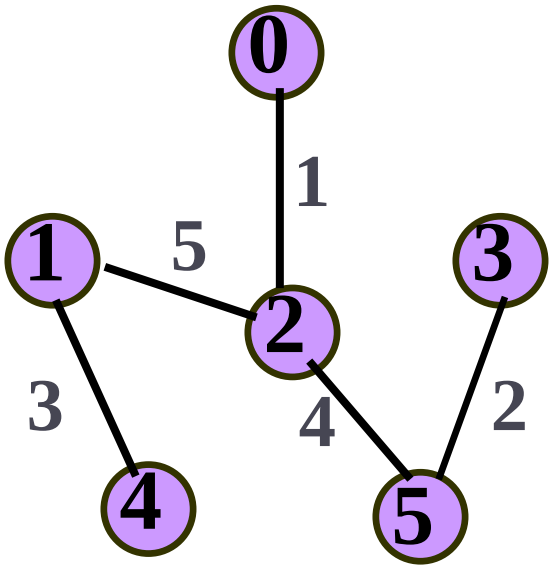

0
1
2
4
5

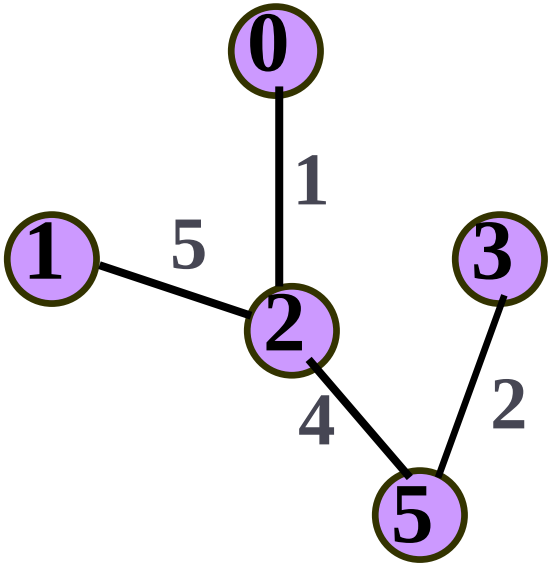

U
V-U
0
1
2
3
4
5
6
5
1
5
7
3
5
4
2
6


0
1
2
3
4
5
1
4
2

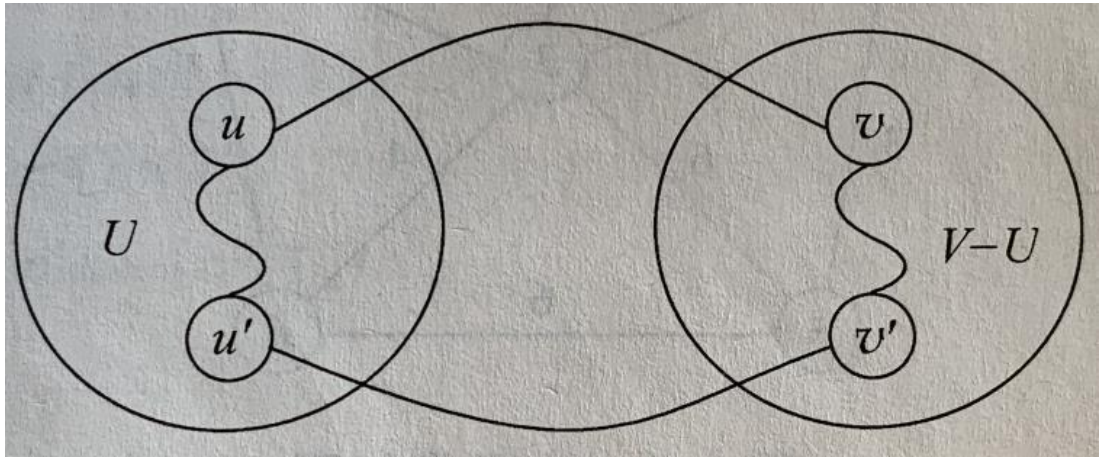

0
1
2
3
4
5
1
5
4
2


U
V-U
0
1
2
3
4
5
6
5
1
5
7
3
5
4
2
6


0
1
2
3
4
5
1
5
3
4
2


0
1
2
3
4
5
1
5
4
2

2. Prim贪心算法的正确性



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的一颗最小生成树，它以(u,v) 为其中一条边。

由这一特性，可证明(1)贪心选择性质和 (2)最优子结构性质

哈夫曼编码

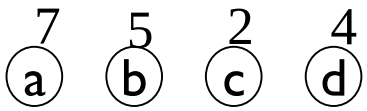
- 根据给定的n个权值$\{w_1, w_2, \cdots\cdots w_n\}$，构造n棵只有根结点的二叉树，令其权值为$w_j$
- 在森林中选取两棵根结点权值最小的树作左右子树，构造一棵新的二叉树，置新二叉树根结点权值为其左右子树根结点权值之和
- 在森林中删除这两棵树，同时将新得到的二叉树加入森林中
- 重复上述两步，直到只含一棵树为止，这棵树即哈夫曼树

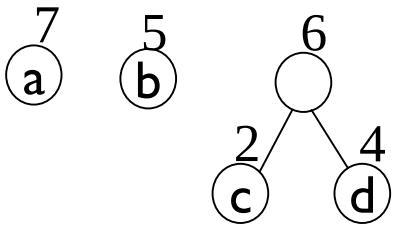
$$WPL=\sum_{k=1}^{n} W_K L_K$$

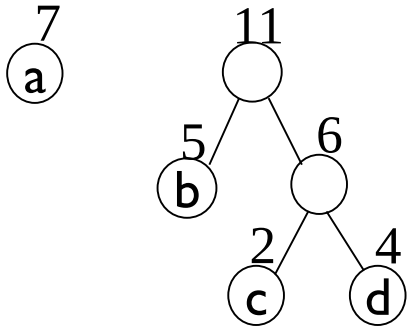
哈夫曼编码——示例

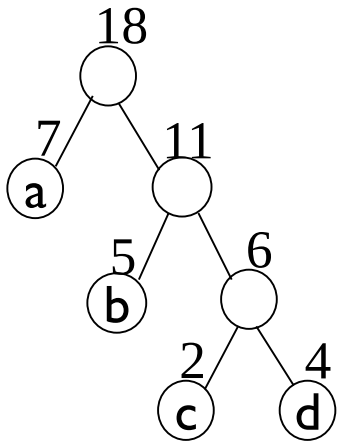
有4个结点，权值分别为7，5，2，4，构造有4个叶子结点的二叉树

例







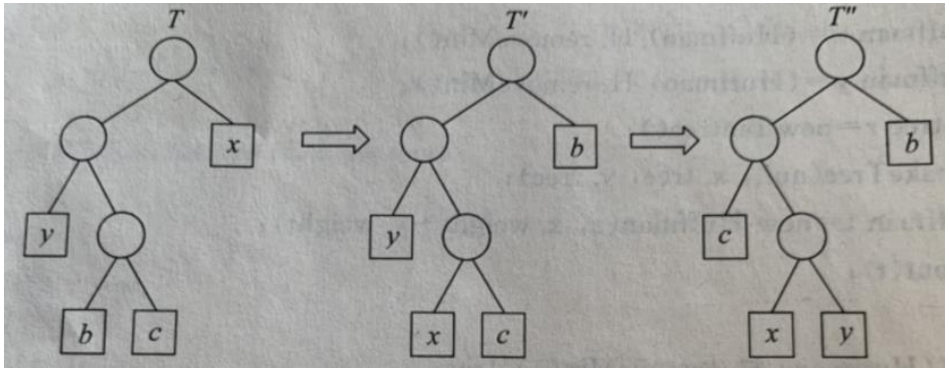


$$WPL=\sum_{k=1}^{n} W_K L_K$$

哈夫曼贪心算法的正确性

(1)贪心选择性质

设C是编码字符集，C中字符c的频率为f(c)。又设x和y是C中具有最小频率的两个字符，存在C的最优前缀码使x和y具有相同码长且仅最后一位不同



设二叉树T表示C的任意一个最优前缀码，如果能证明对T作适当修改得到T”，使得T”表示的前缀码也是C的最优前缀码。则结论成立。

$$B(T)-B(T')=\sum_{c\in C}f(c)d_T(c)-\sum_{c\in C}f(c)d_{T'}=f(x)d_T(x)+f(b)d_T(b)-f(x)d_{T'}(x)-f(b)d_{T'}(b)$$

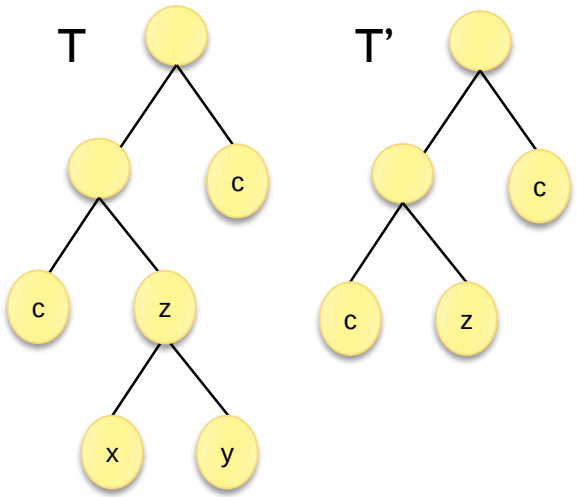
$$=f(x)d_T(x)+f(b)d_T(b)-f(x)d_T(b)-f(b)d_T(x)=(f(b)-f(x))(d_T(b)-d_T(x))\geq 0$$

可类似证明在T’交换y与c的位置也不增加平均码长，即B(T’)-B(T”)也是非负的。由此可知B(T”) ≤B(T’) ≤ B(T)。另一方面，由于T所表示的前缀码是最优的，所以B(T) ≤ B(T”) 。因此B(T) = B(T”)，即T”表示的前缀码也是最优的，且x和y具有相同的码长，仅最后一位不同。

哈夫曼贪心算法的正确性

(2)最优子结构性质

设T是字符集C一个最优前缀码。C中字符c的频率为f(c)，设x和y是T中两个叶子且为兄弟，z是它们的父亲。若将z看作是具有频率f(z)=f(x)+f(y)字符，则树T’=T−{x,y}表示的字符集C’=C−{x,y}+{z}的一个**最优前缀码**。



首先证明T的平均码长B(T)可用T’的平均码长B(T’)表示。

对任意$c \in C-\{x,y\}$有$d_T(c)=d_{T'}(c)$，故$f(c)d_T(c)=f(c)d_{T'}(c)$，且$d_T(x)=d_T(y)=d_{T'}(z)+1$

因此：$f(x)d_T(x)+f(y)d_T(y)=\big(f(x)+f(y)\big)(d_{T'}(z)+1)=f(x)+f(y)+f(z)d_{T'}(z)$

则： $\mathrm{B(T)}=\mathrm{B}(T')+f(x)+f(y)$

如果T'所表示的字符集C'的前缀码不是最优的，则T''表示的C'的前缀码使得$\mathrm{B}(T'')<\mathrm{B}(T')$。由于z被看作是$C'$中一个字符，故z在$T''$中是一颗叶子。若将x和y加入树中作为z的儿子，则得到表示字符集C的前缀码的二叉树T'''，且有：

$B(T''')=B(T'')+f(x)+f(y)<B(T')+f(x)+f(y)=B(T)$ 产生矛盾，因为T是最优的

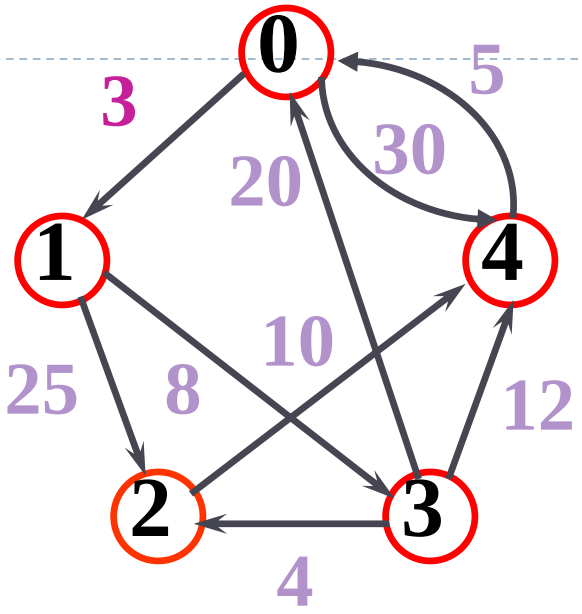
单源最短路径

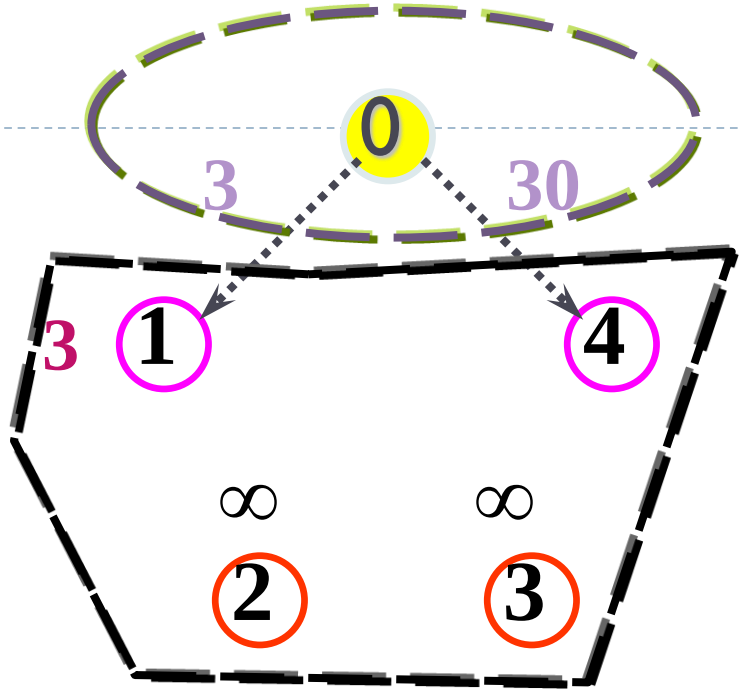
- 问题描述：给定一个图G=(V, E)，其上的边权函数w为非负数。给定V中的一个顶点，称为源。现在要计算从源到所有其它各顶点的最短路长度。这里路的长度是指路上各边权之和。这个问题通常称为单源最短路径问题。

- 算法思想：贪心算法求解
  - 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并对其不断扩充，从源点到S中 顶点的最短距离已知
  - 每一步添加u∈$V-S$中具有最小距离的顶点到S中
  - 更新剩余顶点u 的距离估计

1. 算法基本思想和步骤

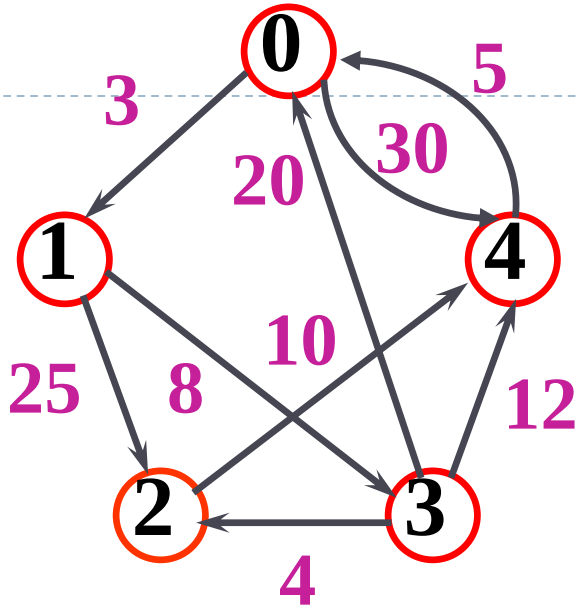
- 初使时令 **S={V0}, T={其余顶点}**，T中顶点对应的**距离值**：
  - 若**存在**<V0, Vi>，为<V0, Vi>弧上的**权值**
  - 若**不存在**<V0, Vi>，**为∞**
- 从**T中**选取一个其**距离值为最小**的顶点**W**，**加入S**
- 对**T中**顶点的距离值进行**修改**：若**加进W**作**中间顶点**，从**V0到Vi的距离值**比**不加W**的路径**要短**，则**修改此距离值**
- 重复上述步骤，直到S中包含所有顶点，即**S=V**为止

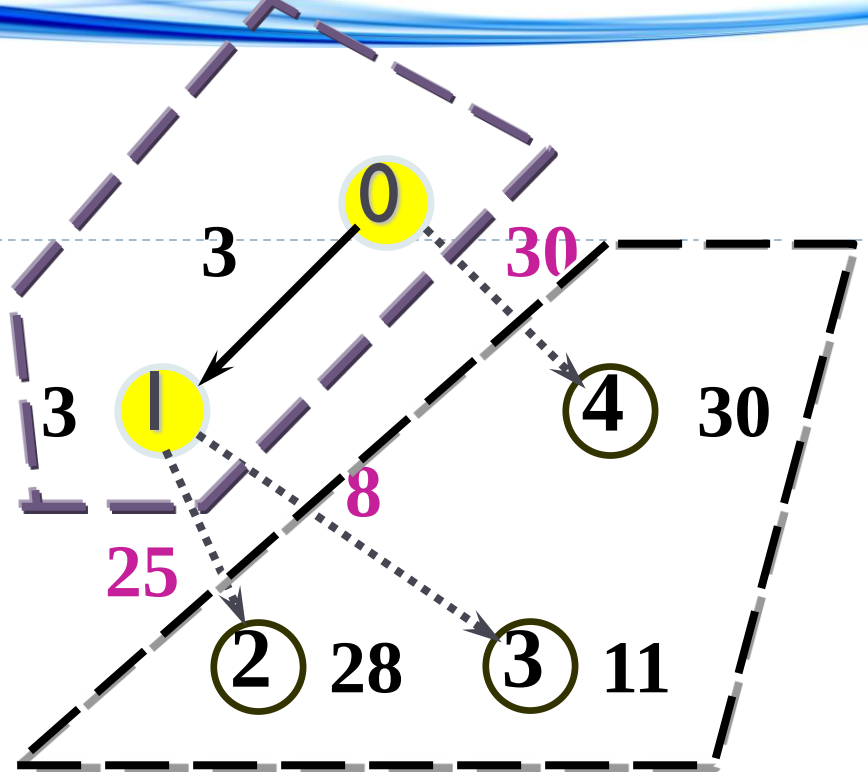




最小最短特殊路径

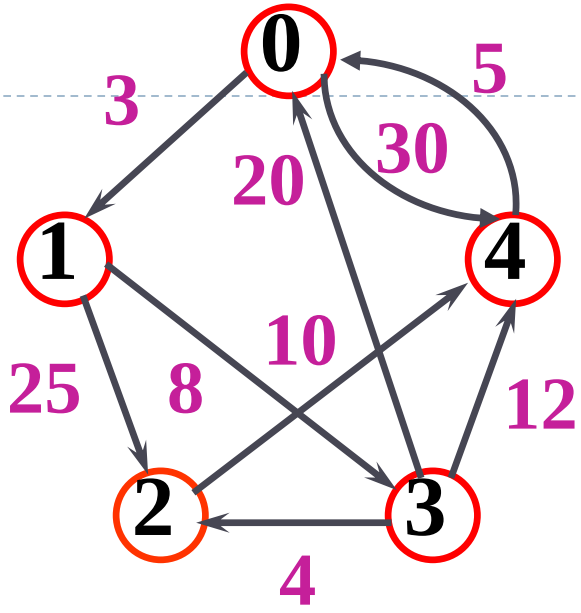
Dijkstra算法按递增次序依次得到各顶点的最小路径长度

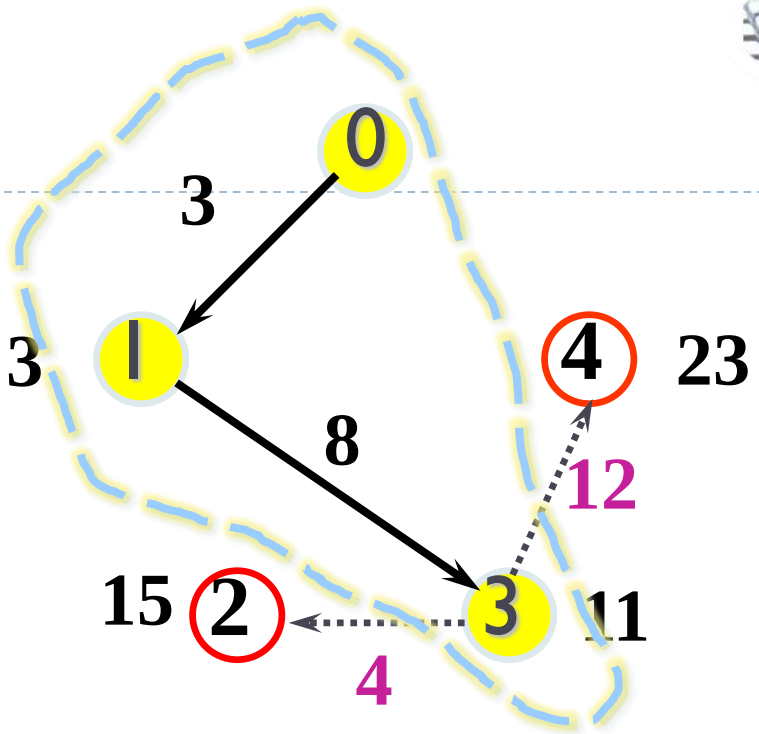




次小最短特殊路径

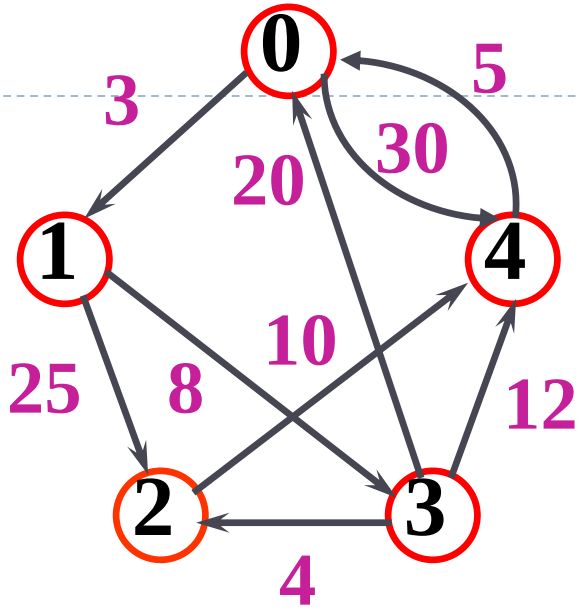
Dijkstra算法按递增次序依次得到各顶点的最小路径长度

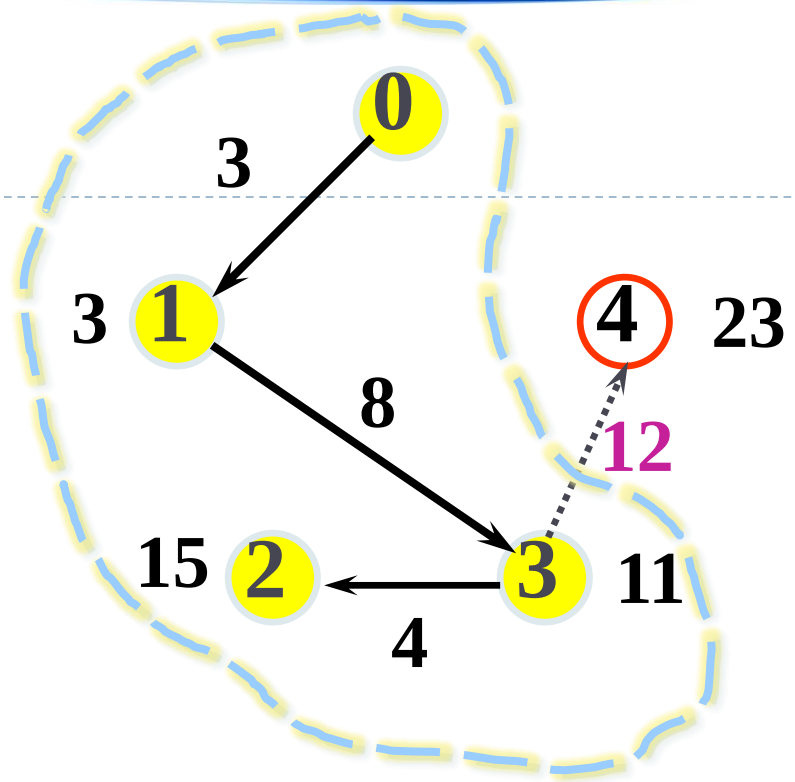




第三小最短特殊路径

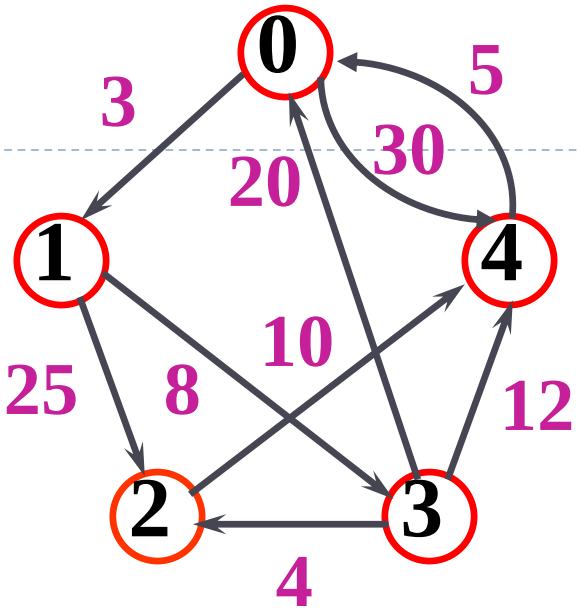
Dijkstra算法按递增次序依次得到各顶点的最小路径长度

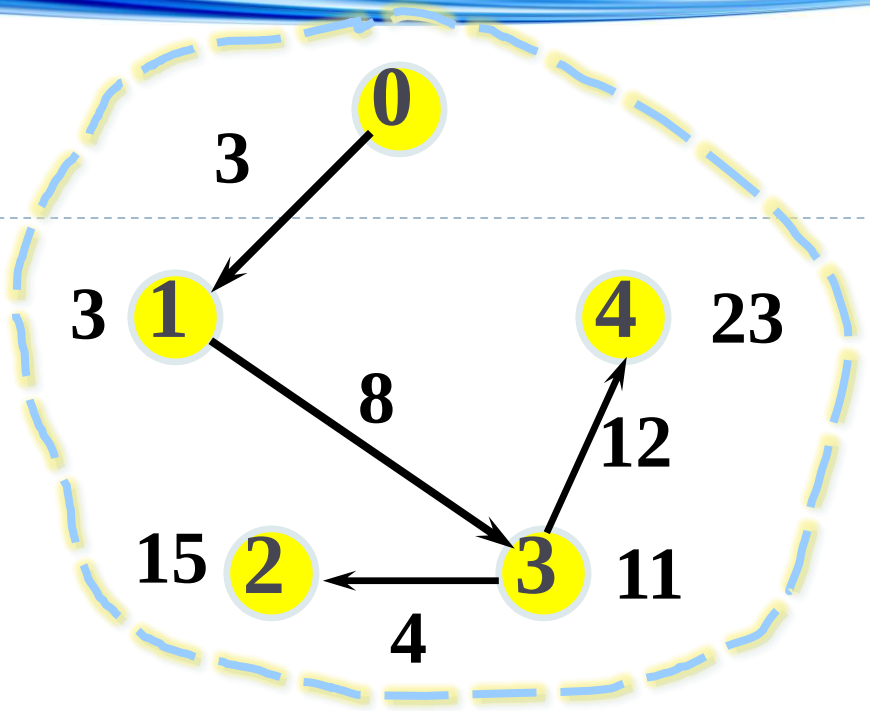




第四小最短特殊路径

Dijkstra算法按递增次序依次得到各顶点的最小路径长度





依次得到所有最短路径

Dijkstra算法按递增次序依次得到各顶点的最小路径长度

Dijkstra算法描述

初始时（S为初始顶点），$D_s=0$且 $\forall i \neq S$，有$D_i=+\infty$。

①在未访问顶点中选择D_v最小的顶点v，访问v，令 S[v]=1。

②依次考察v的邻接顶点w，若

$D_v+weight(<v,w>)< D_w$，

则改变D_w的值，使$D_w = D_v + weight(<v,w>)$。

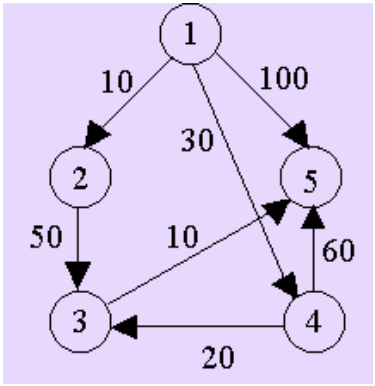
③重复①②，直至所有顶点被访问。

说明

- 引入一个**辅助数组*dist***。它的每一个分量*dist[i]*表示当前找到的从源点*s*到顶点*i* 的最短路径的长度。初始状态：

  dist[*s*]=0, 对其它节点*i*有*dist*[*i*]=+∞ 。
- 引入path[*i*]记录*s*到*i*最短路径中*i*的前驱节点编号

单源最短路径



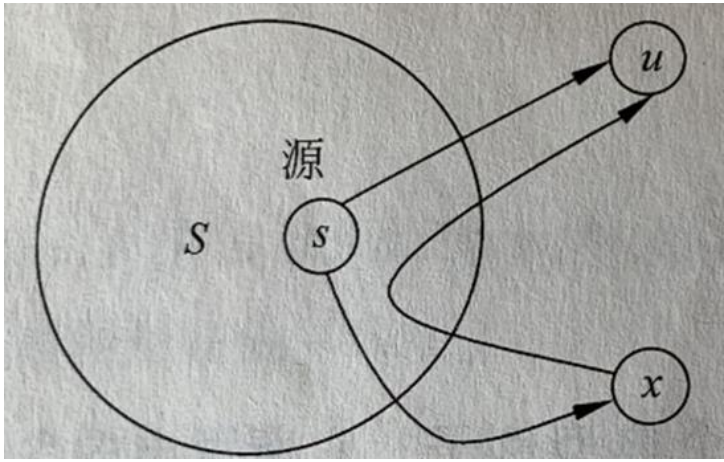
- 应用Dijkstra算法计算从源顶点1到其它顶点间最短路径的过程

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

2. Dijkstra贪心算法的正确性

(1)贪心选择性质

贪心策略是从V-S中选择具有最短特殊路径的顶点u，从而确定从源s到u的最短路径长度dist[u]，为何能导致最优解呢?



假设存在一条从s到u且长度小于dist[u]的路径$d(s,u)$，该路径初次走出S到达x，设这段初次走出S到达x的路径长度为$d(s,x)$，然后从x开始徘徊S内外若干次到达u，设这段路径长度为$d(x,u)$，则有:

$$dist[x] \le d(s,x) \le d(s,x) + d(x,u) = d(s,u) < dist[u]$$

矛盾，因为dist[u]是当前最短的特殊路径，否则节点x就应该先u加入S，而不是在u加入时x还在S外面。

2. Dijkstra贪心算法的正确性

(2)最优子结构性质

如果**S(i,j)={Vi….Vk..Vs…Vj}**是从顶点i到j的最短路径，**k和s**是这条路径上的一个中间顶点，那么**S(k,s)**必定是从**k到s的最短路径**

假设**S(i,j)={Vi….Vk..Vs…Vj}**是从顶点**i到j**的最短路径，

则有：**S(i,j)=S(i,k)+S(k,s)+S(s,j)**

若**S(k,s)**不是从**k**到**s**的最短距离，则：

必定存在另一条从**k**到**s**的最短路径**S'(k,s)**，那么**S'(i,j)=S(i,k)+S'(k,s)+S(s,j)<S(i,j)**

该结论与**S(i,j)**是从**i**到**j**的最短路径相矛盾

最优装载

- 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Wi。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

- 算法思想：最优装载问题可用贪心算法求解
  - 采用重量最轻者先装的贪心选择策略，可产生最优装载问题的最优解

最优装载算法

```
template<class Type>
void Loading(int x[],  Type w[], Type c, int n)
{
        int *t = new int [n+1];
        Sort(w, t, n); // t(i)=物品i的排序序号,按重量的升序排序
        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]];
        }
}
```

- 算法主要计算量在于将集装箱依其重量从小到大排序
- 算法复杂性为 O(nlogn)。

最优装载问题贪心算法正确性

(1)贪心选择性质

设集装箱已按照重量从小到大排序，$(x_1, x_2, \cdots\cdots, x_n)$是最优装载问题的一个最优解。又设$k = \min_{1 \le i \le n} \{i | x_i = 1\}$。易知，如果给定的最优装载问题有解，则$1 \le k \le n$。

（1）当k=1时，$(x_1, x_2, \cdots\cdots, x_n)$是一个满足贪心选择性质的最优解。

（2）当k>1时，取$y_1 = 1, y_k = 0, y_i = x_i, 1 < i \le n, i \ne k$，则

$\sum_{i=1}^{n} w_i y_i = w_1 - w_k + \sum_{i=1}^{n} w_i x_i \le \sum_{i=1}^{n} w_i x_i \le c$

因此，$(y_1, y_2, \cdots\cdots, y_n)$是所给最优装载问题的可行解。

另外，由$\sum_{i=1}^{n} y_i = \sum_{i=1}^{n} x_i$，知道$(y_1, y_2, \cdots\cdots, y_n)$是满足贪心选择的最优解。

(2)最优子结构性质

设$(x_1, x_2, \cdots\cdots, x_n)$是最优装载问题的满足贪心选择的最优解，则易知，$x_1 = 1$，且$(x_2, \cdots\cdots, x_n)$是轮船载重量为$c - x_1$，待装船集装箱为{2, 3, ……. , n}时相应最优装载问题的最优解。也就是说，最优装载问题具有最优子结构性质。

多机调度问题

问题描述：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
- 是NP完全问题，到目前为止还没有有效的解法
- 贪心选择策略有时可以设计出较好的近似算法

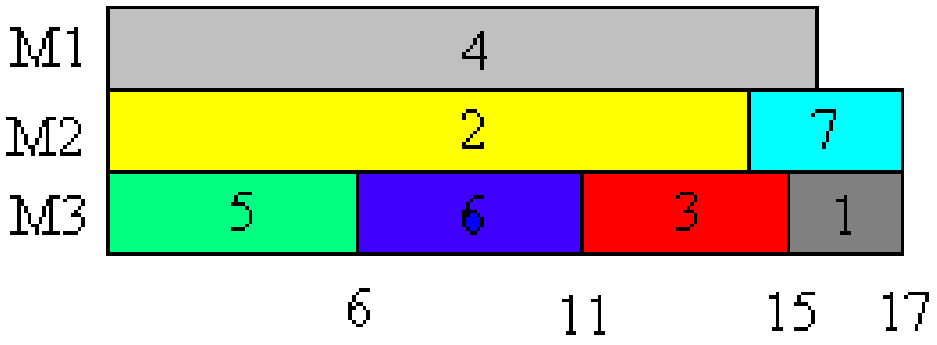
多机调度问题

解决思路： 采用最长处理时间作业优先的贪心选择策略设计较好的近似算法。

- 当 $n \leqslant m$ 时，只要将机器 i 的 $[0, t_i]$ 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)。

多机调度问题

- 设7个独立作业{1,2,3,4,5,6,7}由3台机器M_1，M_2和M_3加工处理，各作业所需的处理时间分别为{2,14,4,16,6,5,3}
  - 按算法greedy产生的作业调度如下图所示，所需的加工时间为17。



贪心算法与动态规划

1. 贪心选择性质

所谓**贪心选择性质是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**，也是贪心算法与动态规划算法的**主要区别**。

动态规划算法通常以自底向上的方式解各子问题，而**贪心算法则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**。

2. 最优子结构性质

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**。

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**。

问题的最优子结构性质是该问题可用动态规划算法或贪心算法求解的关键特征。

贪心算法与动态规划算法的差异

- 动态规划思想：分治法类似，也是将待求解问题分解成若干个子问题。子问题往往不是互相独立的。

- 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
  - 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?
  - 是否能用动态规划算法求解的问题也能用贪心算法求解?

以研究经典的背包和0-1背包组合优化问题说明贪心算法与动态规划算法的主要差别。

0-1背包问题

问题描述：直接或间接地给定n种物品和一个背包。物品i的重量是W_i，其价值为V_i，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?

限制：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，也不能只装入物品i的一部分。**

假定一背包10公斤容量，物体重量分别为7,5,4公斤，价值分别为6,4,3.

背包问题

问题描述：直接或间接地给定n种物品和一个背包。物品i的重量是W_i，其价值为V_i，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?

限制：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类问题都具有**最优子结构**性质，极为相似，但**背包问题可以用贪心算法求解，而0-1背包问题却不能用贪心算法求解**。

假定一背包10公斤容量，物体重量分别为7,5,4公斤，价值分别为6,4,3.

贪心算法解背包问题的基本步骤

- 计算每种物品单位重量的价值 Vi / Wi；
- 依贪心选择策略，将尽可能多的单位重量价值最高的物品装入背包；
- 若将这种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；
- 依此策略一直地进行下去，直到背包装满为止。

用贪心算法解背包问题的算法

```
void Knapsack(int n,float M,float v[],float w[],float x[]){
        Sort(n,v,w);//将各种物品依其单位重量的价值从大到小排序
        int i;
        for (i=1;i<=n;i++) x[i]=0;
        float c=M;
        for (i=1;i<=n;i++) {
            if (w[i]>c) break;
            x[i]=1;
            c-=w[i];
            }
        if (i<=n) x[i]=c/w[i];
      //如果有剩余空间，就将i物品的部分装入
}
```

算法knapsack的主要计算时间在于**将各种物品依其单位重量的价值从大到小排序**。因此，算法的计算时间上界为O(nlogn)。

为了证明算法的正确性，还必须证明背包问题具有贪心选择性质。

0-1背包问题算法选择分析

- 贪心选择之所以不能得到最优解是因为在这种情况下，**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。
- **在考虑0-1背包问题时，应比较选择该物品和不选择该物品所导致的最终方案，然后再作出最好选择。**由此就导出许多**互相重叠的子问题**。这正是该问题可用动态规划算法求解的另一重要特征。
- **动态规划算法的确可以有效地解0-1背包问题。**

End of Course