第四章.贪心算法(Greed method)

例 题

用以求解最优化问题

4.1 基本思想

将问题的求解过程看作是一系列选择,每次选择一个输入,每次选择都是当前状态下的最好选择(局部最优解).每作一次选择后,所求问题会简化为一个规模更小的子问题.从而通过每一步的最优解逐步达到整体的最优解。

[适用问题] 具备贪心选择和最优子结构性质的最优化问题

整体的最优解可通过一系列局部最优解达到.每次的选择可以依赖以前作出的选择,但不能依赖于后面的选择

问题的整体最优解中包含着它的子问题的最优解.

[常见应用]背包问题,最小生成树,最短路径,作业调度等等

[算法优点]求解速度快,时间复杂性有较低的阶.

[算法缺点]需证明是最优解.

4. 2. 活动安排问题

[问题陈述]设有n个活动E={1,2,…,n}要使用同一资源,同一时间内只允许一个活动使用该资源. 设活动*i*的起止时间区间[s_i, f_i) ,如果选择了活动*i*,则它在时间区间[s_i, f_i)内占用该资源;若区间[s_i, f_i)与[s_J, f_J)不相交，则称活动i与j是相容的. 求解目标是在所给的活动集合中选出最大相容活动子集.

[算法思路]将n个活动按结束时间非减序排列,依次考虑活动i，若i与已选择的活动相容,则添加此活动到相容活动子集.

[例] 设待安排的11个活动起止时间按结束时间的非减序排列

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

最大相容活动子集(1， 4， 8， 11)，

也可表示为等长n元数组:(1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1)

活动安排问题贪心算法

```
template< class Type >
void GreedySelector(int n, Type s[ ], Type f[ ], bool A[] )
{ A[ 1 ] = true;
   int j = 1;
  //从第二个活动开始检查是否与前一个相容
  for (int i=2;i< =n;i+ +) {
    if (s[i]>=f[j]) {
        A[i] = true;
        j=i;}
     else A[ i] = false;} }
```

[算法证明] 算法达到最优解.

[算法分析] T(n)=O(n) (排序时)

T(n)=O(nlogn) (未排序时)

4.3 最优装载

[问题描述] 输入:(x1,x2,...xn), xi=0,货箱i不装船; xi=1,货箱i装船

可行解: 满足约束条件 $\sum_{i=1}^{n} w_i x_i \leqslant c$ 的输入

优化函数: $\sum_{i=1}^{n} x_i$

最优解:使优化函数达到最大值的一种输入.

[算法思路] 将装船过程划为多步选择，每步装一个货箱，每次从剩下的货箱中选择重量最轻的货箱.如此下去直到所有货箱均装上船或船上不能再容纳其他任何一个货箱。

[例]

设n=8，[w1，…w8]=[100, 200, 50, 90, 150, 50, 20, 80]，c=400。所考察货箱的次序为 :7, 3, 6, 8, 4, 1, 5, 2。货箱7, 3, 6, 8, 4, 1的总重量为390个单位且已被装载, 剩下的装载能力为10 ,小于任意货箱.所以得到解[x1,...x8]=[1, 0, 1, 1, 0, 1, 1, 1]

最优装载的贪心算法

```
template < class Type >
void Loading(int x[], Type w[], Type c, int n )
{ int *t = new int [n + 1];
   Sort(w, t, n) ; //按货箱重量排序/
   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]];}} //调整剩余空间/
```

算法证明:该算法能得到最优解.

算法分析: 排序为主要算法时间,所以 T(n)=O(nlogn)

4-4 背包问题 (Knapsack Problem)

[问题描述]设有n个物体和一个背包,物体i的重量为w_i,价值为v_i,背包的容量为C.若将物体i的x_i部分$(1\le i\le n,\ 0\le x_i\le 1)$装入背包,则具有价值为$v_i x_i$. 目标是找到一个方案,使放入背包的物体总价值最高.

[最优化描述] 找一个n元向量$(x_1,\ldots x_n)$ $0\le xi\le 1$ 使得 $\sum_{i=1}^{n} w_i x_i \le c$ (约束条件)

且$\max \sum_{i=1}^{n} v_i x_i$. 其中 C, Wi, vi>0 , $1 \le i \le n$ (优化函数)

[例] n=3, c=20 (v1, v2, v3)=(25, 24, 15), (w1, w2, w3)=(18, 15, 10)

{x1, x2, x3}	{1, 2/15, 0}	{ 0, 2/3, 1}	{0, 1, 1/2}	{. . .}
$\sum_{i=1}^{n} w_i x_i$	20	20	20	...
$\sum_{i=1}^{n} v_i x_i$	28.2	31	31.5	···

[算法思路] 1). 将各物体按单位价值由高到低排序.
2). 取价值最高者放入背包.
3). 计算背包剩余空间.
4). 在剩余物体中取价值最高者放入背包.
若背包剩余容量=0或物体全部装入背包为止

背包问题的贪心算法

```
void Knapsack(int n,float M,float v[ ],float w[ ] ,float x[ ])
{ Sort(n, v, w); //按单位价值排序/
   int i;
   for (i =1;i <= n;i++) x[i] = 0;
   float c = M; //c为背包剩余空间/
   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]; }
```

算法证明:该算法能得到在最优解

算法分析: 排序为主要算法时间,所以 T(n)=O(nlogn)

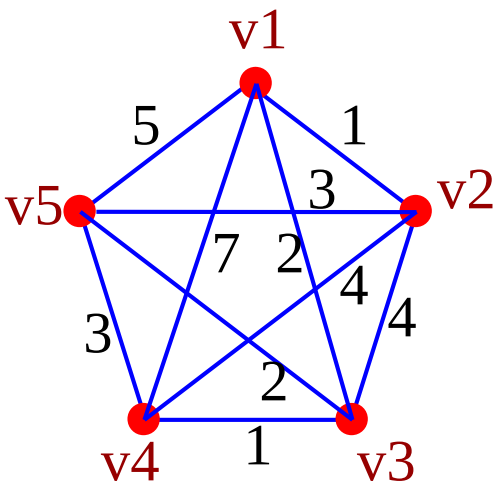
背包问题中的物体不能分拆,只能整个装入称为0-1背包问题.

用贪心算法能得到0-1背包的最优解吗?

*旅行商问题(货郎担问题)

问题：设一个由N个城市$v_1,v_2,...v_n$组成的网络, $c_{i,j}$为从v_i到v_j的代价不妨设$c_{i,j} = c_{j,i}$,且$c_{i,i}=\infty$.一推销员要从某城市出发经过每城市一次且仅一次后返回出发地问如何选择路线使代价最小。

抽象描述:将城市以及之间的道路抽象为一个无向图G, G中每边的权值表示这段线路的代价. 问题转化为求一条最佳周游路线:从一点出发,经过每点一次且仅一次并返回原点,且该路线的总代价最小.



C=

∞	1	2	7	5
1	∞	4	4	3
2	4	∞	1	2
7	4	1	∞	3
5	3	2	3	∞

输入:城市的数目n, 代价矩阵c=c(1..n,1..n).
输出: 最小代价路线

```
1.      tour:=0;    // tour  纪录路线/
2.      cost:=0;    //  cost  纪录到目前为止的花费/
3.      v:=N;       // N为起点城市, v为当前出发城市/
4.      for  k:=1 to  N-1 do
5.          { tour:= tour+(v,w) //(v,w)为从v到其余城市代价中值最小的边/
6.            cost:= cost+c(v,w)
7             v:=w}
8       tour:= tour+(v,N)
9       cost:= cost+c(v,N)
        print tour, cost   }
```

*该算法不能求的最优解. 算法的最坏时间复杂性为$O(n^2)$

该问题为NP难问题.

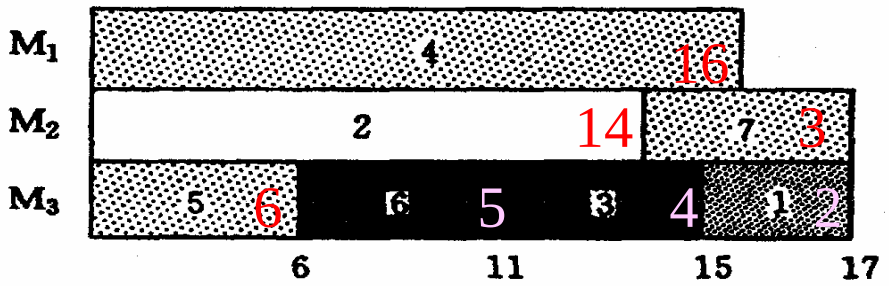
4.7 多机调度问题

问题: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所需时间为ti. 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。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。

该问题为NP完全问题.

贪心近似算法: 采用最长处理时间作业优先的贪心策略:
当n≤m时, 只要将机器i的[0, ti]时间区间分配给作业i即可。
当n>m时, 将n个作业依其所需的处理时间从大到小排序,然后依次将作业分配给空闲的处理机。

7个独立作业{1, 2, 3, 4, 5, 6, 7}
由M1,M2和M3来加工处理
各作业所需时间间分别为
{2, 14, 4, 16, 6, 5, 3}。



多机调度问题的贪心近似算法

```
class JobNode {
    friend void Greedy(JobNode * , int, int);
    friend void main(void);
    public:
        operator int () const {return time; }
    private:
        int ID,
            time;
  };
class MachineNode {
    friend void Greedy(JobNode *, int, int);
    public:
        operator int( ) const { return avail; }
    private:
        int ID,
            avail; }
```

多机调度问题的贪心近似算法

```
template<class Type>
void Greedy(Type a[]，int n，int m)
{if (n<=m){
   cont<<”为每个作业分配一台机器.“<<endl
   return；}
Sort(a，n)；
MinHeap<MachineNode>H(m)；
MachineNode x；
for(int：i=1；i<=m; i++){
  x．avail=0；
  x．ID=i;
  H．Insert(x)；}
for(inti：n；i>=1；i--){
   H．DeleteMin(x)；
   cout<<”将机器”<<x．ID<<“从” <<x．avail<<"到”
        <<(x．avail十a[i]．time)
        <<”的时间段分配给作业”<<a[i]. ID<<endl；
   x．avail+=a[i]．time；
  H．insert(x)；}}
```

4.5 哈夫曼编码

问题：通讯过程中需将传输的信息转换为由字母组成的字符串。由于英文字母使用频率不同，若频率高的字母对应短编码，频率低的字母对应长的编码，传输的数据总量就会降低。要求找到一个编码方案，使传输的数据量最少.

为能正确译码,编码需采用前缀码，前缀码和二叉树一一对应.

问题可化为求一棵最优二叉树

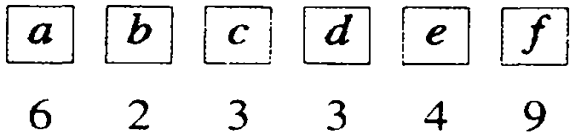
序列集合，其任一序列不能是另一序列的前缀

算法思路：

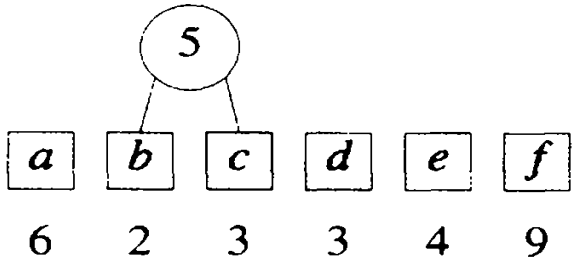
1)以n个字母为结点构成n棵仅含一个点的二叉树集合,字母的频率即为结点的权.

2)每次从二叉树集合中找出两个权最小者合并为一棵二叉树:
增加一个根结点将这两棵树作为左右子树.新树的权为两棵子树的权之和.

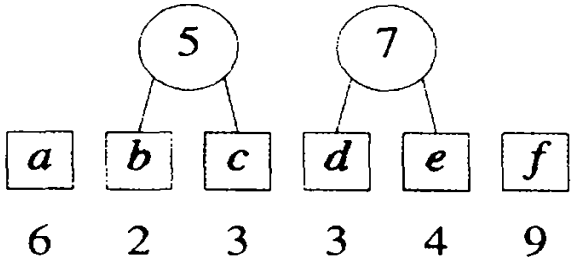
3) 反复进行步骤2)直到只剩一棵树为止.


a b c d e f
6 2 3 3 4 9

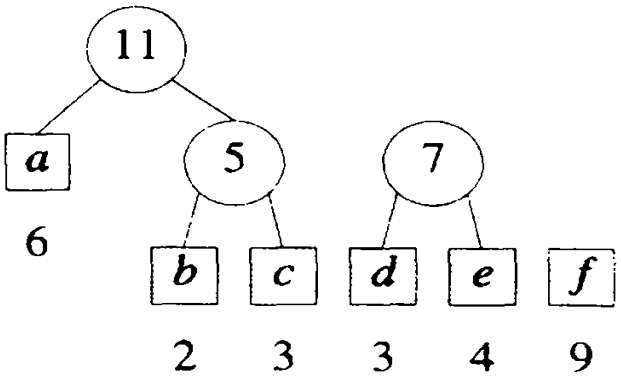
a)


5
a b c d e f
6 2 3 3 4 9

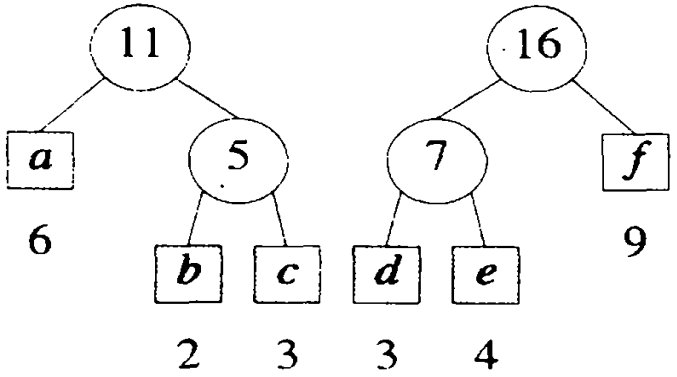
b)


5 7
a b c d e f
6 2 3 3 4 9

c)


11
a 5 7
6 b c d e f
2 3 3 4 9

d)


11 16
a 5 7 f
6 b c d e 9
2 3 3 4

e)

霍夫曼树算法

```
template<class T>
BinaryTree<int>HuffmanTree(T f[], int n)
{//根据权f [1: n]构造霍夫曼树
  //创建一个单节点树的数组
  Huffman <T>*W=newHuffman<T> [n+1];
  BinaryTree<int> z,zero;
  for(int i=1;  i<=n;  i++){
     z.MakeTree(i, zero, zero);
     W[i].weight=f[i];
     W[i].tree=z: }
 //数组变成一个最小堆
  MinHeap<Huffman<T>>Q(1);
  Q.Initialize(w,n,n);
  //将堆中的树不断合并
  Huffman<T> x, y
  for(i=1;i<n;i++){
     Q.DeleteMin(x);
     Q.DeleteMin(y);
     z.MakeTree(0, x.tree, y.tree);
     x.weight+=y.weight;  x.tree=z;
     Q.Insert(x);   }
  Q. DeleteMin(x);  //最后的树
  Q. Deactivate();
  delete[] w;
  return x.tree;
```

算法证明

贪心选择性： 设T为带权$w_1 \leqslant w_2 \leqslant \ldots \leqslant w_t$的最优树,

a).带权w_1和w_2的树叶v_{w_1}和v_{w_2}是兄弟.

b).以v_{w_1}和v_{w_2}为儿子的分枝点,其通路长度最长.

最优子结构: 设T为带权$w_1 \leqslant w_2 \leqslant \ldots \leqslant w_t$的最优树,若将以带权$w_1$和$w_2$的树叶为儿子的分枝点改为带权$w_1+w_2$的树叶,得到一棵新树T', 则T'也是最优树。

算法分析

HuffmanTree初始化优先队列Q需要O(n)

DeleteMin和Insert需O(logn). n-1次的合并总共需要O(nlogn)

所以n个字符的哈夫曼算法的计算时间为O(nlogn)

例 题

4.6 单源最短路径

问题：给定带权有向图G=(V,E), 其中每条边的权是一个非负实数. 要计算从V的一点v_0(源)到所有其他各顶点的最短路长度. 路长指路上各边权之和。

算法思路(Dijkstra) :设最短路长已知的终点集合为S,初始时$v_0 \in S$, 其最短路长为0,然后用贪心选择逐步扩充S :每次在V-S中,选择路径长度值最小的一条最短路径的终点x加入S.

构造路长最短的最短路径:设已构造i条最短路径,则下一个加入S的终点u必是V-S中具有最小路径长度的终点, 其长度或者是弧(v_0,u), 或者是中间只经过S中的顶点而最后到达顶点u的路径.

算法描述：

- (1) 用带权的邻接矩阵c来表示带权有向图, c[i][j]表示弧<vi,vj>上的权值. 若 <vi, vj>∉V,则置c[i][j]为∞

  设S为已知最短路径的终点的集合,它的初始状态为空集.

  从源点v到图上其余各点vi的当前最短路径长度的初值为：

  dist[i]=c[v][i] vi∈V
- (2) 选择vj, 使得dist[j]=Min{dist[i] | vi∈V-S}

  vj就是长度最短的最短路径的终点。令S=SU{j}
- (3) 修改从v到集合V-S上任一顶点v_k的当前最短路径长度:

  如果 dist[j]+c[j][k]< dist[k] 则修改 dist[K]= dist[j]+c[j][k]
- (4) 重复操作(2),(3)共n-1次.

单源最短路径问题的Dijkstra算法

```
voidDijkstra(int n, int v,Type dist[], int prev[], Type **c)
{ bool s[maxint];
  for (int i=1;i<=n; i++){
    dist[i]=c[v][i];
    s[i]=false;
    if(dist[i]= =maxint) prev[i]=0;
    else   prev[i]=v ; }
  dist[v]=0;  s[v]=true;
  for (int i=1;i<n;  i++){
    int temp=maxint;
    int u= v;
    for (int j = 1;  j<=n;  j++)
      if ((!s[j])&&(dist[j]<temp)){
        u=j;
        temp=dist[j];  }
    s[u]=true;
    for (int j=1;  j<=n;j++)  ;
      if((!s[j])& &(c[u][j]<maxint)){
        Type  newdist=dist[u)+c[u][j];
      if (newdist<dist[j]){
        dist[]]=newdist;
        prev[j]=u;  }}}}
```

分析:用带权邻接矩阵表示有n个顶点和e条边的带权有向图,主循环体需要O(n)时间,循环需要执行n-1次,所以完成循环需要$O(n^2)$.

4.7 最小生成树

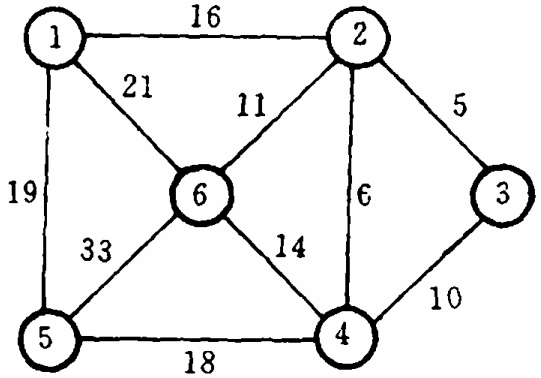
问题陈述:设G(V,E)是一个无向连通带权图。E中每条边(v, w)的权为c[v][w],若G的一个子图G'是一棵包含G的所有顶点的树，则称G'为G的生成树。生成树各边权的总和称为该生成树的耗费。在G的所有生成树中,耗费最小的生成树称为G的最小生成树.

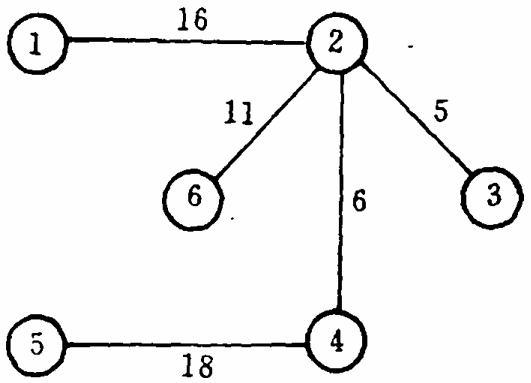
抽象描述: 输入:任一连通生成子图 (该子图的边集合)

可行解:图的生成树,

优化函数:生成树的各边权值之和

最优解:使优化函数达到最小的生成树.





例 题

Prim算法

设G=(V,E)是一个连通带权图, y={l, 2, …, n}。

算法思路: 首先置S={1}, T= Ø .若S⊂V, 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(i, j),将顶点j添加到S中边(i, j)添加到T中.这个过程一直进行到S=V时为止. T中的所有边构成G的一棵最小生成树。

算法描述

```
void Prim(int n, Type  * * c)
  { T= Ø;
    S ={1};
     while (S!= V) {
            (i, j) = i ∈ S且 j∈V- S的最小权边;
            T=TU{(i,j)};
             S= S U{j}; }
          }
```

Prim算法

```
template < class Type >
void Prim(int n, Type * * c)
  { Type lowcost[maxint];
     int closest[ maxim];
    bool s[maxint];
    s[1] = true;
    for(int i = 2;i<= n; i++){
      lowcost[i] = c[l][i];
      closest[ i] = 1;
      s[i]= false; }
    for (int i = 1; i< n; i++){
      Type min = inf;
      int j = 1;
      for (int k = 2;k<= n; k++)
        if ((lowcost[k] < min)&&(!s[k]))
          min = lowcost[ k];
          j=k;}
      cout< < j < < ' ' < < closest[j] < < end};
      s[j] = true;
      for(intk= 2; k<= n; k++)
        if ((c[j][k] < lowcost[k])&&(!s[k]))
          lowcost[i] = c[j][k];
          closest[ k] =j; }}}
```

算法分析: $O(n^2)$.

Kruskal算法

例 题

G=(V, E), V=<1, 2, …, n}。

算法思路: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，并按下述方法连接两个通分支：当查看到第k条边(v,w)时,如果端点u和w分别是当前两个不同的连通分支T1, T2中的顶点时,就用边(u,w)将TI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十1条边。直到只剩下一个连通分支为止。

Kruska算法

```
template < class Type >
bool Kruskal(int n, int e, EdgeNode < Type > E[ ], EdgeNode < Type > t[ ] )
        MinHeap < EdgeNode < Type > > H(1);
        H. Initialize(E, e, e);
        UnionFind U(n);
       iht k = 0;
       while (e && k < n- 1) {
            EdgeNode <int > x;
            x.u = 2;
            x.v = 1;
            x. weight = 5;
            H. DeleteMin(x);
            e--;
            int a = U.Find(x.u);
           int b = U.Eind(x.v);
           if (a! = b) {
               t[k ++] = x;
         U. Union(a, b);
        H. Deactivate( )
renturn (k = = n-1)
```

算法分析:当图的边数为e时，将其组成优先队列需要时间O(e) while循环中,DeleteMin需要O(loge)时间因此关于优先队列所作运算需时间O(eloge)。实现UnionFind所需的时间为O(eloge) 所以Kruska的时间是O(eloge),适合求边数较少的最小生成树。

```
template<classType>
  void Greedy(Type a[]，int n，int m)
  {  if(n<=m){
          cout<<”为每个作业分配一台机器．”<<endl；
           return；}|
  Sort(a，n)；
     MinHeap<MachineNode> H(m)；
     MachlneNode X；
     for(int i=1；i<=m；i++){
         x.avail=0；
         x.1D=i;
         H.Insert(x)；}
    for (int i=n；i>=1；i--){
         H.DeleteMin(x)；
```

```
cout<<”将机器”<<x.ID<<”从”
      << x. avail <<”到”
      <<(x.avail+a[i]].time)
      <<”的时间段分配给作业”
     <<a[i].ID<<endl；
x.avail+=a[i].time；
H. Insert(x)；  }}
```

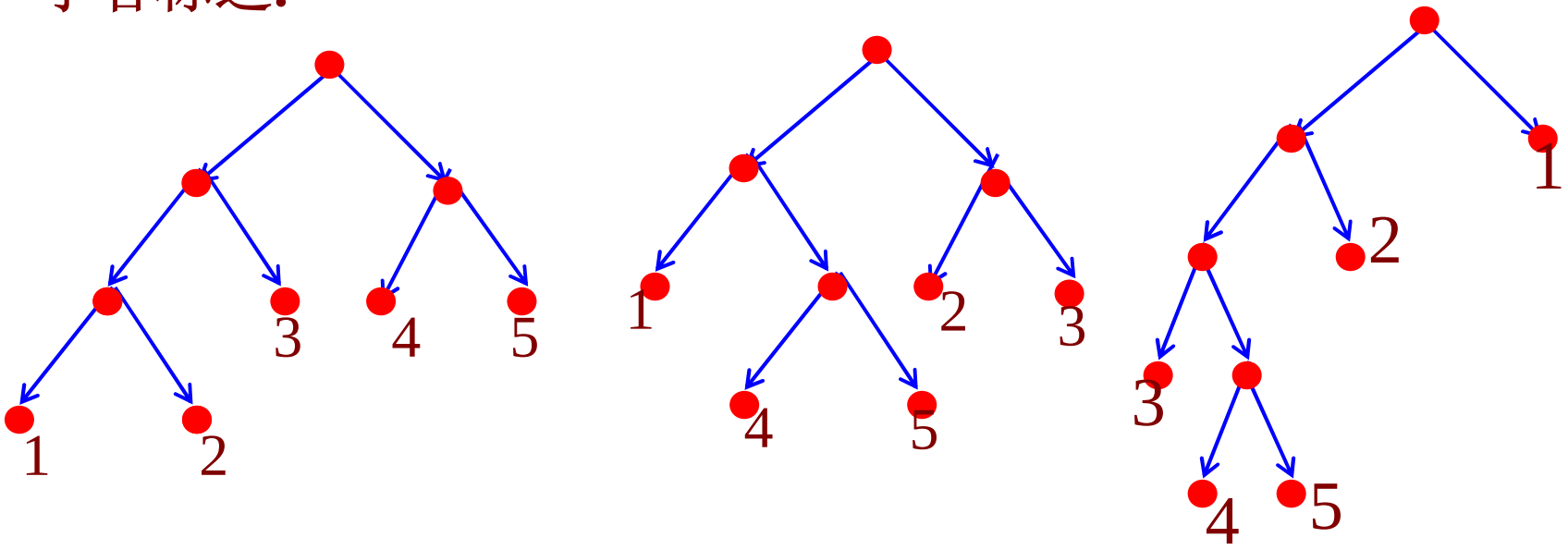
分析:n>m时, 排序耗时O(nlogn). 初始化堆耗时O(m). 堆的DeleteMin和insert 共需O(nlogm).因此算法Greedy 所需时间:O(nlogn)。

最优二叉树

叶带权二叉树:若二叉树T的每片叶子v都对应一个正实数w.

树的权:在叶带权二叉树中, 若带权为w_i的叶,其通路长度为$L(w_i)$, 则称 $w(T)=\Sigma\, w_i \bullet L(w_i)$ 为该叶带权二叉树的**权**.

最优二叉树:在所有叶带权为$w_1,w_2,\ldots,w_t$的二叉树T中$w(T)$最小者称之.

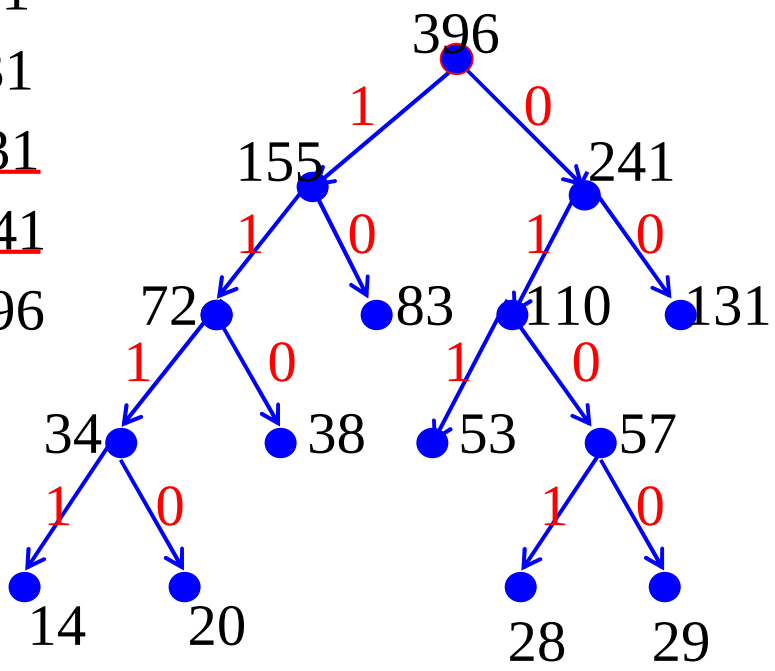


例 题

设在1000个字母的文章中各字母出现的频率为:
a:83, b:14, c:28, d:38, e:131, f:29, g:20, h:53.....

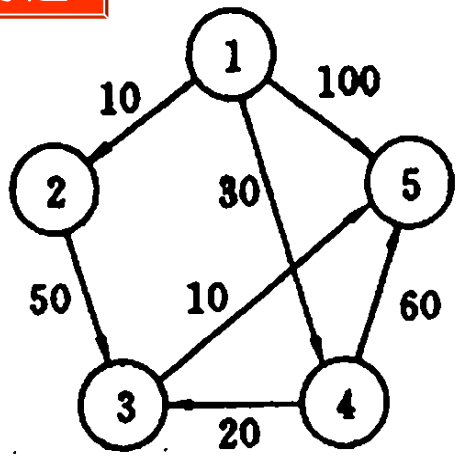
14	20	28	29	38	53	83	131
	34	28	29	38	53	83	131
		34	57	38	53	83	131
			57	72	53	83	131
				72	110	83	131
					110	155	131
						155	241
							396

1)将权从小到大排序
2)每次选取最小权合并



最佳编码:
a:10 ; b:1111; c:0101; d:110;
e:00; f:0100; g:1110; h:011

例 题



∞	10	∞	30	100
∞	∞	50	∞	∞
∞	∞	∞	∞	10
∞	∞	20	∞	60
∞	∞	∞	∞	∞

1) v1 → v2: 10

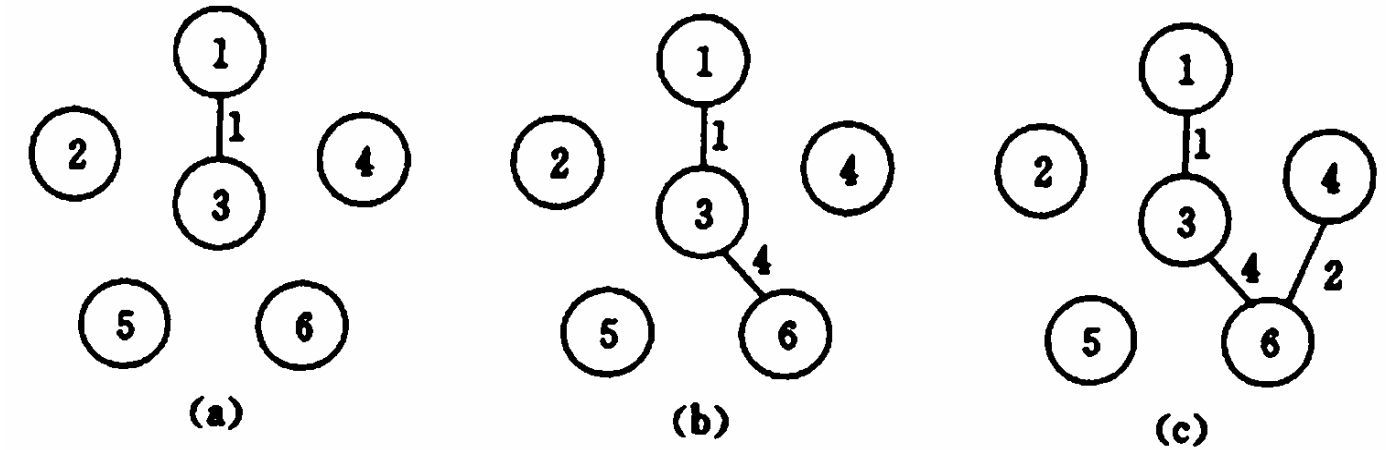
2) v1 → v3: 50

3) v1 → v4: 30

4) v1 → v5: 60

迭代	S	vi	d[2]	d[3]	d[4]	d[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	{1,2,3,4}	3	10	50	30	60
4	{1,2,3,4,5}	5	10	50	30	60

例 题



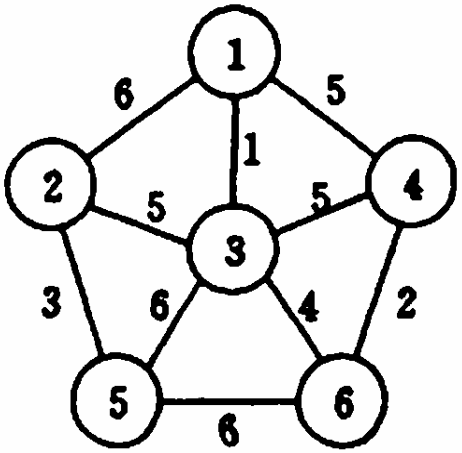


图 4 - 10　连通带权图

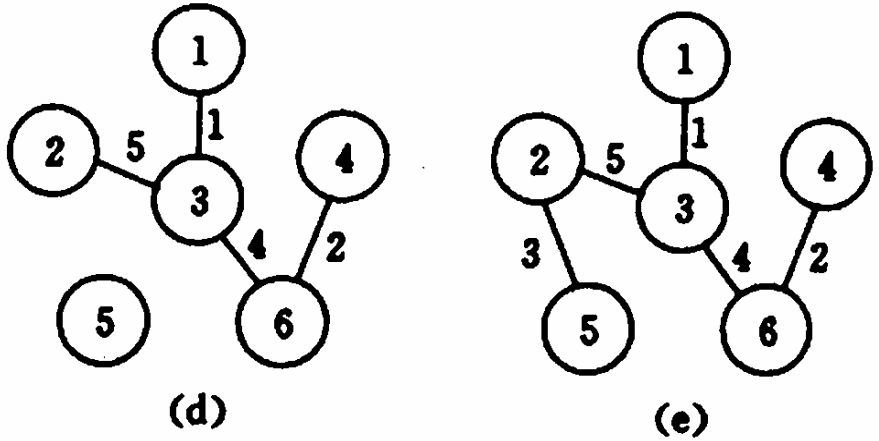


图 4 - 11　Prim 算法选边过程

例 题

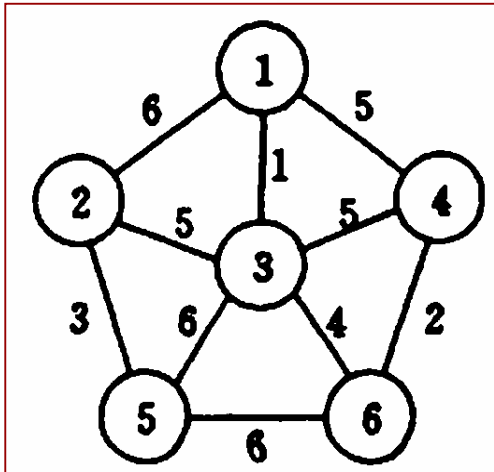


图 4 - 10　连通带权图

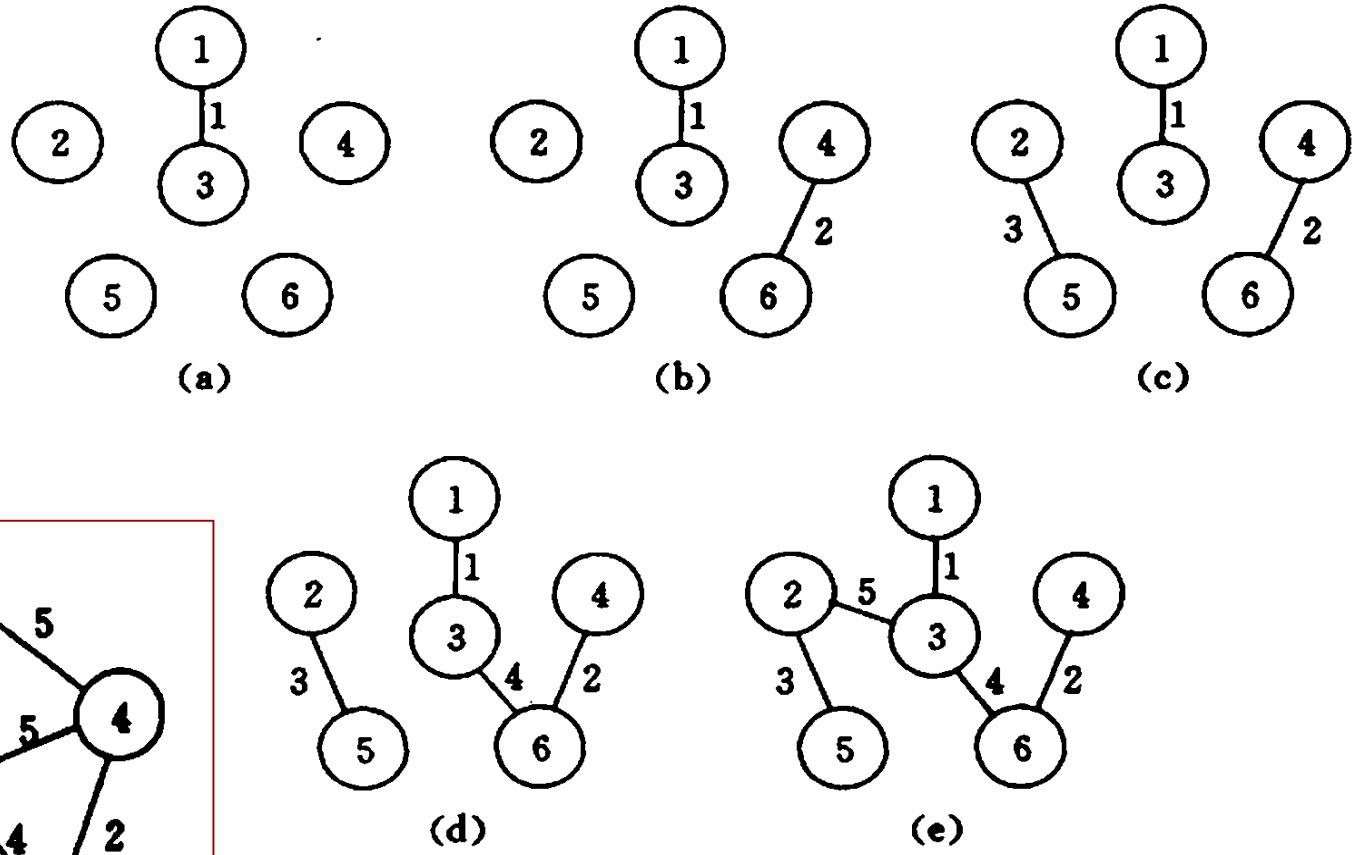


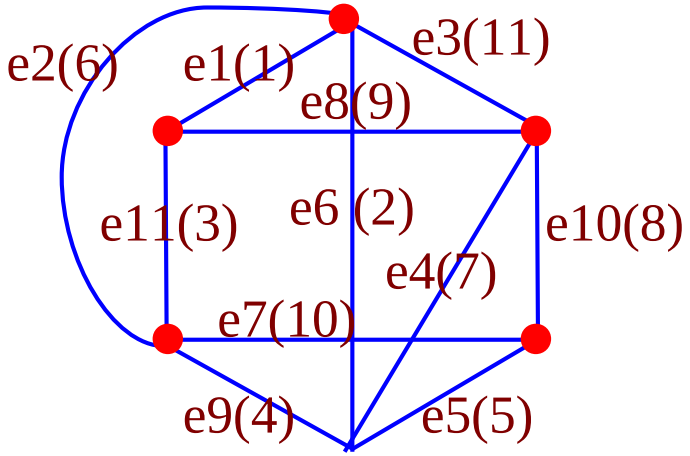
图 4 - 12　Kruskal 算法选边过程

求最小生成树(Kruscal)

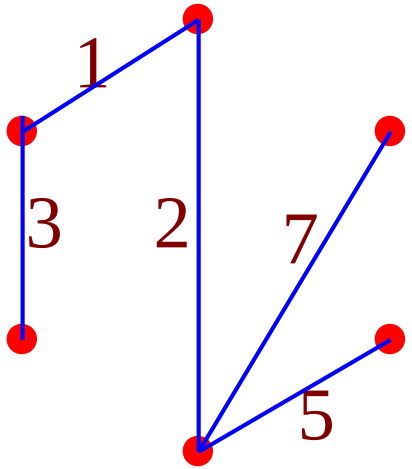
1) 以G 中全部点为点作图

2) 按权的大小次序依次添加各边,若出 现回路则忽略此边.

3) 加入n-1条边后就得到最小生成树.



最优解: (e1, e6, e11, e5, e4)



最优化问题(optimization problem):

问题可描述为有n个输入(x1, x2, . . . xn), 一组约束条件和一个优化(目标)函数。满足约束条件的输入称为可行解,它是输入的一个子集.使优化函数取得极值的可行解称为最优解.

例1

[最小代价通讯网络] 在N城市之间架设通讯线路,要求造价最低.

城市之间所有可能的通讯连接视作一个无向图G, G中每边的权值表示建成这段线路的代价. 问题转化为求一棵最小生成树.

问题描述: 输入:任一连通图G (该图的边集合)

可行解:图G的生成树

优化函数:生成树的各边权值之和

最优解:使优化函数达到最小值的生成树.

例2

[装载问题]有一艘大船准备用来装载货物。所有待装货物都装在n个大小一样的集装箱中,集装箱的重量各不相同。设第i个集装箱的重量为w*i* (1≤*i*≤n). 船的最大载重量为c, 目标是在船上装入最多货物.

问题描述: 输入:(x1,x2,...xn), xi=0:货箱i不装船; xi=1,货箱i装船

可行解: 满足约束条件 $\sum_{i=1}^{n} w_i x_i \leqslant c$ 的输入

优化函数: $\sum_{i=1}^{n} x_i$

最优解:使优化函数达到最大值的一种输入.

```
template<class T>
    TSelect(T a[]，int n，int k)
    { / / 返回a[0：n-1]中第k小的元素
     / / 假定a[n]是—个伪最大元素
    if (k<1||k>n)throwOUtOfBoUndS()：
    return Select(a，0，n-1，k)；
    }   template<classT>
    TSeleCt(Ta[]，intI，intr，intk)
    {//a[1：r]中选择第k小的元素
    if(l>=r)retUrnaI，]；
    inti=I，      / / 从左至右的游标
    j=r+1；  / / 从右到左的游标
    Tpivot=a[1)；
     / / 把左侧>=pivot的元素与右侧<=pivot的元素进行交
```

```
’h ¨ e(1rUe){
d0们在左侧寻找>=pivot的元素
   i二i+飞；
   }wh ¨ e(a[门<pivot)；
do们在右侧寻找<=酬vot的元素
   j 二 j · 下；
   }wh ¨ e(aU]>pivot)；
汁(i>=Dbrea叱 / / 未发现交换对象
Swap(a㈠]， a[j])；
}
(j·0+1==k)retUrnpivot；
设置pivot
1]二aU]；     ，
¨ =pivot；
对一个段进行递归调用
(j·0+下<k)
returnSetect(a， j+下， r， k·j+I-下)；
SereturnSelect(a， I， j·下， k)
```

例 题 1-1 算法1-2:二分查找

```
function b-search(c)
    {  L:=1;          U:=m;
       found:=false;
       while not found  and U>=L do                    3(logm+1)
          { i:=(L+U)div2;                                  3(logm+1)
            if c=A[i]                                       logm+1
            then   found:=true
            else if c>A[i]                                  logm+1
                  then L:=i+1
                  else  U:=i-1                              2(logm+1)
           }
       if found
       then b-search:=1
       else b-search:=0
      }
```

最坏情况T*max*(m) =O(logm)

第三章.动态规划(Dynamic Programming)

用以求解最优化问题

3.1 基本思想

将问题的求解过程化为多步选择或决策的结果, 在每一步决策上, 列出各种可能的选择(各子问题的可行解), 舍去那些肯定不能成为最优解的局部解. 最后一步得到的解必是最优解.

适用问题: 具备最优子结构性质和子问题重叠性的最优化问题.

问题的整体的最优解中包含着它的子问题的最优解

第i+1步问题的求解中包含第i步子问题的最优解, 形成递归求解.

与贪心算法比较:都是将问题的求解过程化为多步决策.区别是:

贪心法每采用一次贪心策略便做出唯一决策,求解过程只产生一个决策序列;求解过程为自顶向下,不一定得到最优解.

动态规划的求解过程产生多个决策序列, 下一步的选择总是依赖上一步的结果.求解过程多为自底向上.总能得到最优解.

算法的步骤

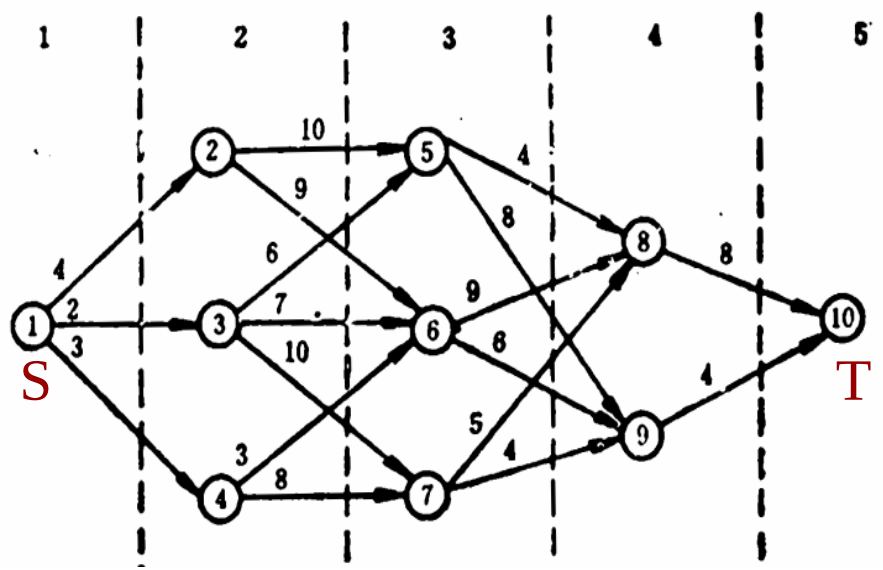
1). 分析最优解的结构.
2). 给出计算局部最优解值的递归关系.
3). 自底向上计算局部最优解的值.
4). 根据最优解的值构造最优解.

常见应用:

0-1背包问题,图像压缩,最短路径,矩阵连乘,作业调度等等.

例 题 **最短路问题**

问题描述:在一带权有向连通图中找一条从s到t的路径使该路径的权(各边权之和)最小.



算法思路:1)将问题化为多步决策,从最后一段开始从后向前依次计算各点到t的最短路径.即先求边数为1的各点到t的最短路径,边数为2的各点到t的最短路径... 设1是i到t的最短路径,则s经过i到达t的最短路径中必包含1为其子路径.(**最优子结构**)

2)当作第i步选择时(计算第i层各点的最短路径),会用到上一步的选择结果(i+1层各点的最短路径)(**子问题重叠性**).

3)计算i层点的递归公式:设ci:从vi到t的最短距离,则

$ci=\min\{c(i,j)+cj\}$, c(i,j):边<vi, vj>的权, vj为vi的下一步各终点

算法效率:16个加法,9次比较

最短路径算法的伪代码

```
for (int i=1； i<=n； i++)
    d(i)=0;
    for(int j=n-1； j<=1； j--)
      {若顶点r使<j,r>∈E,且d( j,r)+d(r)最小
        d(j)= d( j,r)+d(r)
        c(j)=r}
p(1)=1;
p(k)=n; //k为段数
 for(int j=2;j<=k-1； j++)
   p(j)=c(p(j-1))
```

θ(n+|E|)

θ(k)

算法复杂性： θ(n+|E|)

旅行商问题(货郎担问题)

问题陈述:设一个由N个城市$v_1,v_2,...v_N$组成的网络, $d_{i,j}$ 为从v_i到v_j的代价,一般设$d_{i,j} \neq d_{j,i}$,且$d_{i,i}=\infty$.一推销员要从某城市出发经过每城市一次且仅一次后返回出发地.问如何选择路线使代价最小

算法思路:化为多步决策.先求出边数为1时各点到v1的最短路;边数为2(有一个中途点)时各个点到v1的最短路...如此下去最后得到经过n条边的到v1的最短路.(最优子结构)

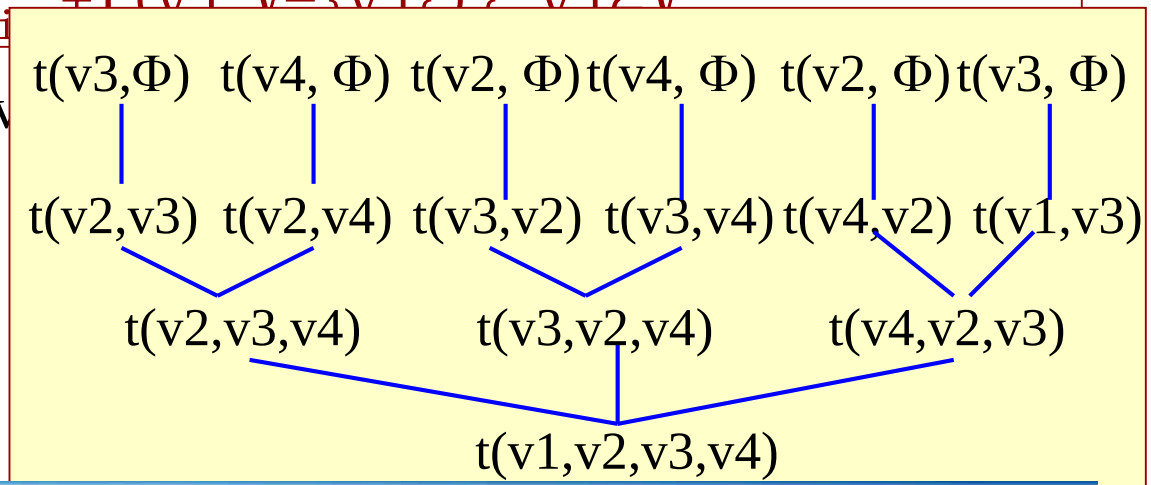
设T(vi,V):从vi出发遍历V中所有点后返回v1的最短路长

其中V是顶点集合,v1∉v,则多步决策的递推公式为:

$$T(vi,V)=\min\{d_{i,j}+T(v_j,V-\{v_j\})\}\quad v_j\in V$$

例如:四个城市v1,v2,v3,v

$$D=\begin{bmatrix} \infty & 2 & 5 & 8 \\ 4 & \infty & 7 & 6 \\ 8 & 5 & \infty & 6 \\ 7 & 5 & 8 & \infty \end{bmatrix}$$



3-10 0-1背包问题(0-1Knapsack Proble

问题陈述：设有n个物体和一个背包,物体
背包的容量为C. 目标是找一个方案,使放

例： n=3, w={100, 14, 10}
v={20, 18, 15}, C=116
求m(1, 116)

形式化描述：找一个n元 0-1向量 $(x_1,...x_n)$

$$\sum_{i=1}^{n} w_i x_i \le C \quad \text{其中 C, W}i\text{, v}i>0\ ,1 \le i \le \text{n}$$

最优子结构：设$(x_1,...x_n)$是所给问题的一个最优解,则$(x_2,...x_n)$是如下子问题的一个最优解:

$$\textbf{max} \quad \sum_{i=2}^{n} v_i x_i \qquad \sum_{i=2}^{n} w_i x_i \le M - w_1 x_1 \qquad x_i \in \{0,1\},\ 2 \le i \le \text{n}$$

递归公式: 设m(i,j):剩余背包容量为j, 剩余物品为i,i+1,..n时的问题的最优解,则计算m(i,j)的递归公式为:

$$m(i,j)=\begin{cases} \max\{\ M((i+1, j),\ M(i+1, j-w_i) + \text{v}i\ \} & j \ge w_i \\ m(\ i+1, j) & 0 \le j < w_i \end{cases}$$

$$m(n,j)=\begin{cases} \text{v}n & \text{j} \ge w_n \\ 0 & 0 \le \text{j} < w_n \end{cases}$$

3.1 矩阵乘法链

问题描述:给定n个矩阵$\{A_1, ...,A_n\}$, 其中A_i与A_{i-1}是可相乘的. 确定一个计算次序, 使得计算矩阵连乘积$A_1...A_n$所需的计算量最少.

例　三个矩阵相乘,有两种计算顺序 (A*B)*C, A*(B*C).

假定A为100×1矩阵，B为1 × 100矩阵，C为100 × 1矩阵，

则 D=A*B为100 ×100矩阵, A*B所需乘法数为10000, D再与C相乘所需乘法数为1000000, 计算(A*B)*C的总时间为**1010000.**

E=B*C所需乘法数为10000, B*C 为1 × 1矩阵, E再与A相乘所需乘法数为100, 因而计算A*(B*C)的时间耗费只有**10100**.

而且计算(A*B)*C时, 还需10000个单元来存储A*B, 而A*(B*C)计算过程中,只需用1个单元来存储B*C。

穷举法求最小连乘积的代价为指数阶的.

算法思路: 化为多步决策, 自底向上, 先求出矩阵链长为1的最优计算次序, 链长为2的最优次序, 链长为3的最优次序. . . . . .

1)最优解结构:设A[1: n]= $A_1 \ldots A_n$, A[1: n]的一个最优计算次序是在矩阵A_k和A_{k+1}, $1 \leqslant k < n$, 间将矩阵链断开, 则总计算量为:

A[1:k]的计算量+A[k+1:n]的计算量+A[1:k]*A[k+1:n]的计算量

则矩阵子链A[1:k]和 A[k+1:n]的计算次序也必是最优(最优子结构)

2)递推关系:设计算A[i: j] = $A_i \ldots A_j$所需最少数乘为m[i][j] (最优解值)

$$m[i][j]=\begin{cases} 0 & i=j \\ \min\limits_{i \le k < j}\{\, m[i][k]+m[k+1][j]+ p_{i-1}p_k p_j \,\}, & i=j \end{cases}$$

Ai的维数设为$p_{i-1} \times p_i$

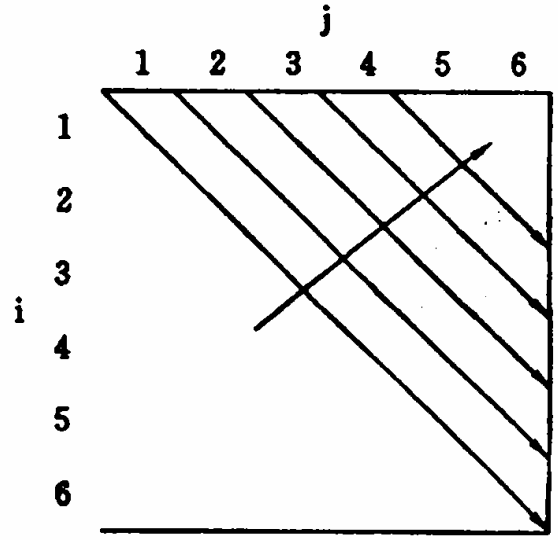
3)构造最有解:记m[i][j]的断开位置为s[i][j], 在计算出m[i][j]后, 可由s[i][j]递归构造相应的最优解.

矩阵乘法链动态规划算法

```
void MatrixChain(int p, int n, int ** m, int ** s)
 { for (int i= 1; i <= n; i ++) m[i][i] =0;
   for (int r= 2; r <= n; r++)
     for (int i= 1; i<= n-r+l; i++) {
          int j= i+r-1;
          m[i][j] = m[i+1][j] + p[i-1] * p[i]* p[j];
          s[i][j] = i;
         for(int k = i+1; k< j;k++) {
              int t = m[i][k] + m[k+1][j] + p[i- 1] * p[k] * p[j];
              if ( t < m[i][j] ) {
                   m[i][j] = t;
                   s[i][j] = k; }}}
   }
```

算法分析: $T(n)=O(n^3)$

A_1	A_2	A_3	A_4	A_5	A_6
30×35	35×15	15×5	5×10	10×20	20×25



计算次序
(a)

i \ j	1	2	3	4	5	6
1	0	15750	7875	9375	11875	15125
2		0	2625	4375	7125	10500
3			0	750	2500	5375
4				0	1000	3500
5					0	5000
6						0

m[i,j]
(b)

i \ j	1	2	3	4	5	6
1	0	1	1	3	3	3
2		0	2	3	3	3
3			0	3	3	3
4				0	4	5
5					0	5
6						0

s[i,j]
(c)

$$m[2][5] = \min\begin{cases} m[2][2] + m[3][5] + p_1p_2p_5 = 0 + 2500 + 35 \times 15 \times 20 = 13\,000 \\ m[2][3] + m[4][5] + p_1p_3p_5 = 2625 + 1000 + 35 \times 5 \times 20 = 7125 \\ m[2][4] + m[5][5] + p_1p_4p_5 = 4375 + 0 + 35 \times 10 \times 20 = 11\,375 \end{cases}$$

$$= 7125$$

例 设n=5和 p=(10, 5, 1, 10, 2, 10)， 由动态规划的递归式得：

m(1, 5)=min{m(1, 1)+ m(2, 5)+ 500, m(1, 2)+ m(3, 5)+ 100，
m(1, 3)+ m(4, 5)+ 1000, m(1, 4)+ c(5, 5)+ 200 }

其中 m(1, 1)= m(5, 5)= 0; m(1, 2)= 50; m(4, 5)= 200.

m(2, 5)= min{ m(2, 2) + m(3, 5) + 50, m(2, 3)+m(4, 5)+500,
m(2 ,4) + m(5, 5) + 100 }

其中 m(2, 2) = m(5, 5)=0； m(2, 3)=50； m(4, 5)=200。

m(3, 5) = min{m(3, 3)+m(4, 5) +100, m(3, 4)+m(5, 5)+20}
= min{ 0 +200 +100, 20 +0 +20 }=40

m(2, 4) = min{m(2, 2) m(3, 4)+l0, m(2, 3)+m(4, 4)+100 }
= min{ 0+20+10, 50+10+20 }=30

s(3, 5)=4, s(2, 4)=2。

m(2,5)=min{0+40+50, 50+200+500, 30+0+100}=90, s[2][5]=2

同理 m(1,3)=150, s(1, 3)=2 , m(1,4)=90, s(1, 4)=2。

最优解值: m[1:5]=min{502, 190 , 1350, 290}=190

3.7 图像压缩

例 题

问题陈述:数字化图像是n×n的像素阵列. 假定每个像素有一个0~255的灰度值, 因此存储一个像素至多需8位. 为了减少存储空间, 采用变长模式, 即不同像素用不同位数来存储, 步骤如下.

1) 图像线性化:将$n \times n$维图像转换为$1 \times n^2$向量 $\{p1,p2,...pn^2\}$

2) 分段: 将像素分成连续的m段s1,s2,...sm,使每段中的像素存储位数相同. 每个段是相邻像素的集合且每段最多含256个像素, 若相同位数的像素超过 256个的话, 则用两个以上段表示。

3) 创建三个表

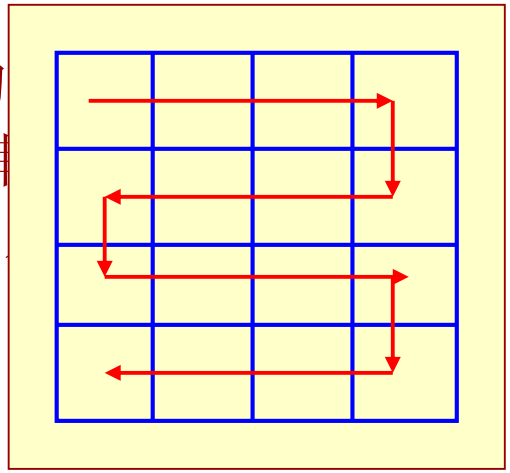
l: l[i]存放第i段长度, 表中各项均为8位长

B: b[i]存放第i段中像素的存储位数,表中各项均

P: $\{p1,...p\ n^2\}$以变长格式存储的像素的二进制串

设产生了m个段,则存储第i段像素所需要的空间

总存储空间为 $11m+\sum l[i]*b[i]$;



问题要求找到一个最优分段。使存储空间最少

例: 考察4×4图像

例 题

灰 度 值	10	9	12	40	50	35	15	12	8	10	9	15	11	130	160	240
像素位数	4	4	4	6	6	6	4	4	4	4	4	4	4	8	8	8

分段: [10, 9, 12], [40, 50, 35], [15, 12, 8, 10, 9, 15, 11], [130, 160, 240]

L={2，2， 6，2}；B={3，5，3，7}；

P包含16个灰度值，其中头三个各用4位存储，接下来三个各用6位再接下来的七个各用4位，最后三个各用8位存储。

1010 1001 1100 111000 110010 100011 ...

10	9	12	40
12	15	35	50
8	10	9	15
240	160	130	11

三个表需要存储空间分别为:

L:32位
B:12位
P:82位

} 共需126位。

将段1,2合并，则l={5, 6, 2} ; b={5, 3, 7 }; 合并后P的第一段为

001010 001001 001100 111000 110010 100011

其余不变.三个表需要存储空间分别为:

L:24位 B:9位 P:88位 (共121位。)

算法思路: 化为多步决策, 先求有一个段时的最优分段,再求有2个段时的最优分段,有3个段时的最优分段...

1)最优解结构 :设n个段的最优合并为C, 若在合并C中,第n段与n-1,n-2,...n-k+1段合并, 则n, n-1,n-2,...n-k+1段的合并也必是最优的.

最优合并C所需要的空间消耗为:

第一段到第n-k段最优合并空间+lsum(n-k+1,n)*bmax(n-k+1,n)+11

其中lsum(a, b)= $\sum_{j=a}^{b} l[j]$ bmax(a, b)=max{ b[a],..., b[b] }

2) 递推关系:令s[i]为前i个段最优合并的存储位数,则

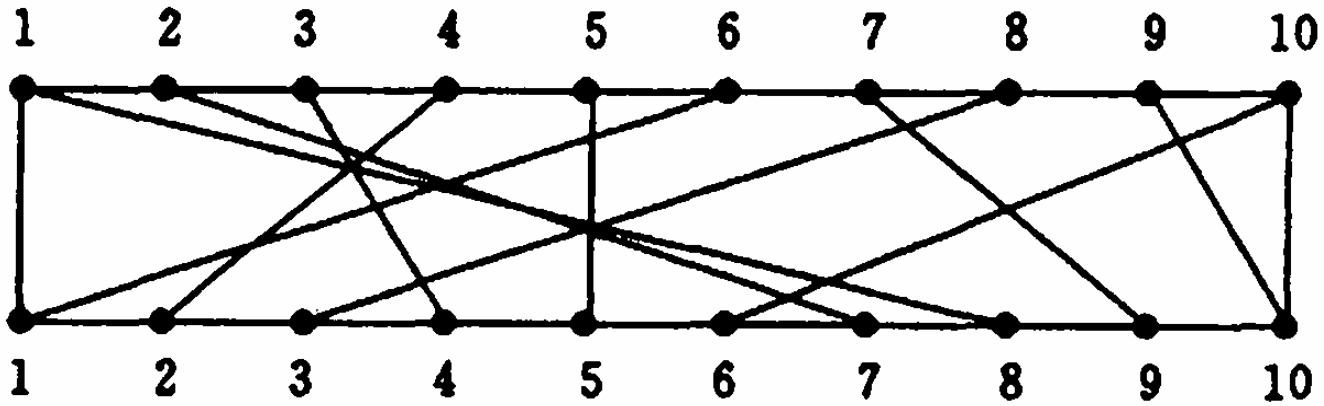
$$s[i]=\min_{\substack{1\le k\le i \\ lsum(i-k+1,i)\le 256}} \{s[i-k]+ lsum(n-k+1,n)*bmax(n-k+1,n)\} +11$$

3)构造最优解:记k[i]为s[i]取得最小值时k的值, 可由k的值构造相应的最优解.

例 题

3．8 电路布线

问题陈述:在一块电路板的上、下两端分别有n个接线柱, 要求用导线(i, π(i))将上端接线柱i与下端接线柱π(i)相连. 对于任何1≤i<j≤n, i, j两条连线相交的充分必要条件是π(i) >π(j)



在制作电路板时，要求将这n条连线分布到若干个绝缘层上。在同一层上的连线不相交.

电路布线问题就是要确定将哪些连线安排在第一层(优先层)上, 使得该层上有尽可能多的连线。

即确定导线集Nets={(i, π(i)), 1≤i≤n} 的最大不相交子集MNS.

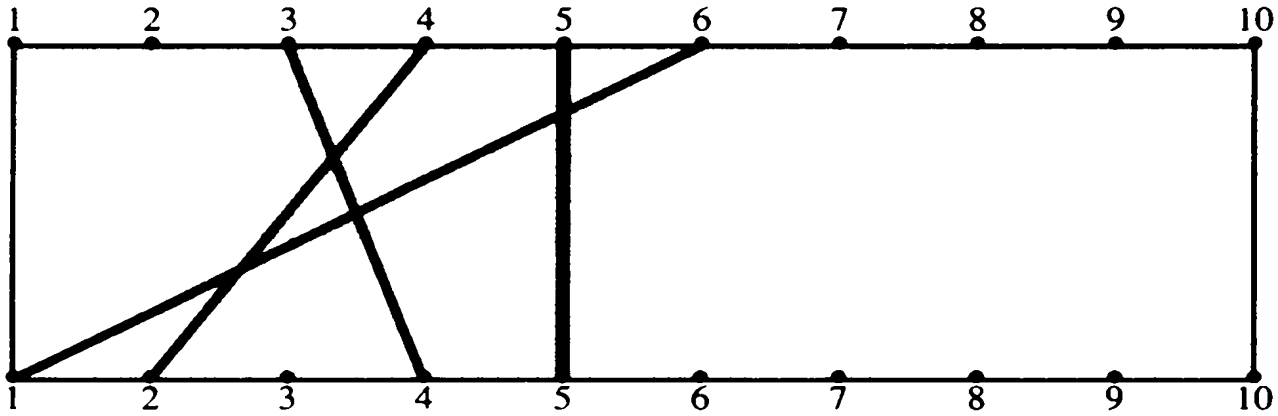
上图中 MNS={(3, 4), (5, 5), (7, 9), (9, 10) }

算法思路： 化为多步决策, 自底向上, 先求出只有一条连线的最大不相交子集, 再求有2条连线的最大不相交子集. . .

1)最优解结构: 设nets(i, j) ={(t, π(t)) | t≤ i, π(t) ≤ j }⊆ nets, MNS(i, j)为nets(i, j)的最大不相交子集, 则所求即为MNS(n,n), 且MNS(i, j) ⊆MNS(n,n).

例如　MNS(10, 10)=MNS={ (3, 4), (5, 5), (7, 9), (9, 10) }

nets(7, 6)={ (3, 4), (4, 2), (5, 5), (6, 1) }



MNS(7, 6) ={ (3, 4) ,(5, 5) }, Size (7, 6) =2

2. 递归计算

当i=1时, nets(1, j)={(1, π(1)}, $MNS(1,j)=\begin{cases}\phi & j<\pi(1)\\ \{(1,\pi(1))\} & j\geq\pi(1)\end{cases}$

当i>l时

1)若j<π(i), (i, π(i)) ∉MNS(i, j),

此时MNS(i, j)=MNS(i-1, j), Size(i, j)=Size(i-1, j);

2)若j≥π(i), (i,π(i))可在也可不在MNS(i, j)中:

若(i,π(i)) ∈MNS(i, j), 对∀(t, π(t))∈MNS(i, j) 有t≤i, π(t) ≤π(i)

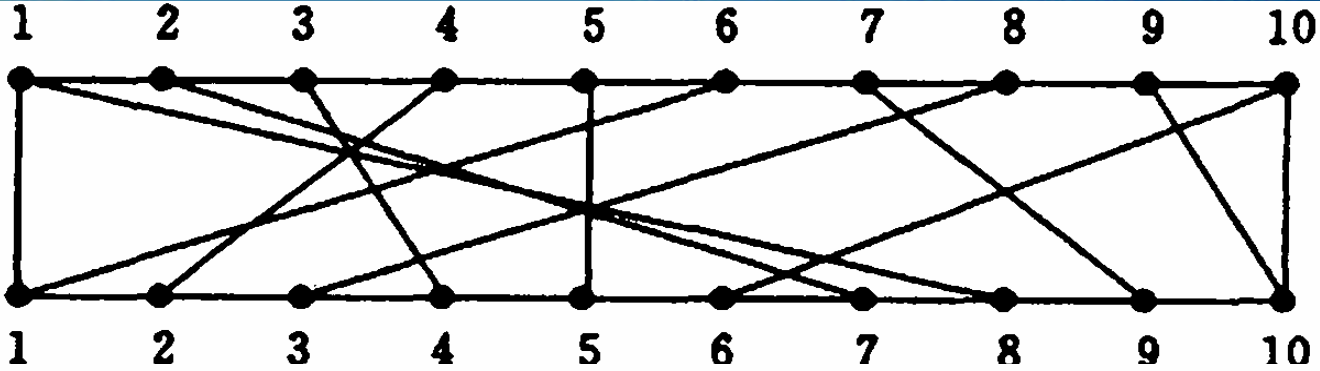
此时MNS(i, j)=MNS(i-1, j-1)∪(i,π(i)), Size(i, j)= Size(i-1,π(i)-1)+1;

若(i, π(i)) ∉MNS(i, j), 则对任意(t, π(t))∈MNS(i, j), 有t<i.

此时MNS(i, j)=MNS(i-1, j), Size(i, j)=Size(i-1, j)。

(1)当i=1时 $Size(i,j)=\begin{cases}0 & j<\pi(1)\\ 1 & j>\pi(1)\end{cases}$

(2)当i>l时 $Size(i,j)=\begin{cases}Size(i-1,j) & j<\pi(i)\\ \max\{Size(i-1,j),Size(i-1,\pi(i)-1)+1\} & j>\pi(i)\end{cases}$



i	j									
	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	1	1	1
2	0	0	0	0	0	0	1	1	1	1
3	0	0	0	1	1	1	1	1	1	1
4	0	1	1	1	1	1	1	1	1	1
5	0	1	1	1	2	2	2	2	2	2
6	1	1	1	1	2	2	2	2	2	2
7	1	1	1	1	2	2	2	2	3	3
8	1	1-	2	2	2	2	2	2	3	3
9	1	1	2	2	2	2	2	2	3	4
10	1	1	2	2	2	3	3	3	3	4

求最大无交叉子集算法

```
void MNS(int C[]，int n，int **size)
//初始化size[1][*]
for(int j=0；j<C[l]；j++)
size[l][j]=0；
for(j=c[1])；j<=n ; j++)
size[1][j]=1；
//计算size[i][*]，1<i<n
for(int i=2；i<n；i++){
   for(int j=0；j<C[i]；j++)
       size[i-1[j]；size[i-1][j]；
   for(j=C[i]；i<=n：j++)
     size[i][j]=max(size[i-1)[j]，
size[n][n]=max(size[n-1][n]，si
```

```
voidTraceback(int C[], int**size, int n,
                        int Net[], int &m)
{//在Net[0:m-1)中返回MMS
 int j=n；//所允许的底部最大编号
 m=0；  //连线的游标
 for(int i=n；i>1；i--)
  //i号net在MNS中?
  if (size[i][j]!=size[i-1][j]{ //在MNS中
      Net[m++]=i;
       j=C[i]-1；}
 //1号网组在MNS中?
 if (j>=C[1])
   Net[m++]=1；//在MNS中
  }
```

算法复杂性: $\theta(n^2)$