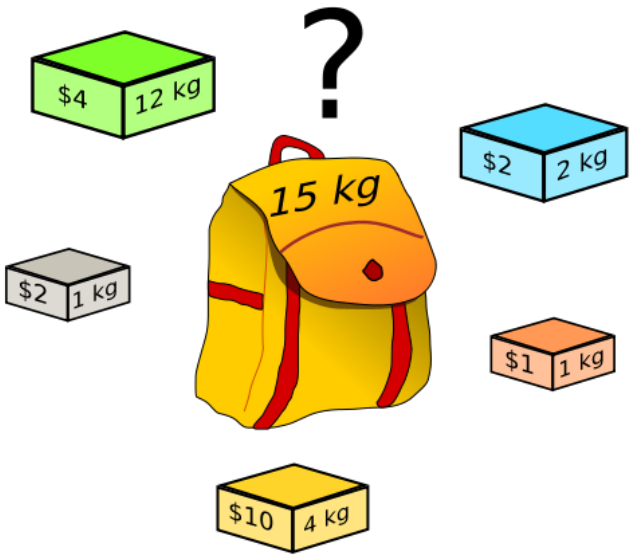
電子工業出版社
Publishing House of Electronics Industry

第5章　回溯法

5.5 0-1背包问题

- n个物体v_i，重量w_i、价值p_i，$0 \leqslant i \leqslant n-1$，背包的载重量C。
- x_i：物体v_i被装入背包的情况，$x_i = 0, 1$。
- 约束方程和目标函数：

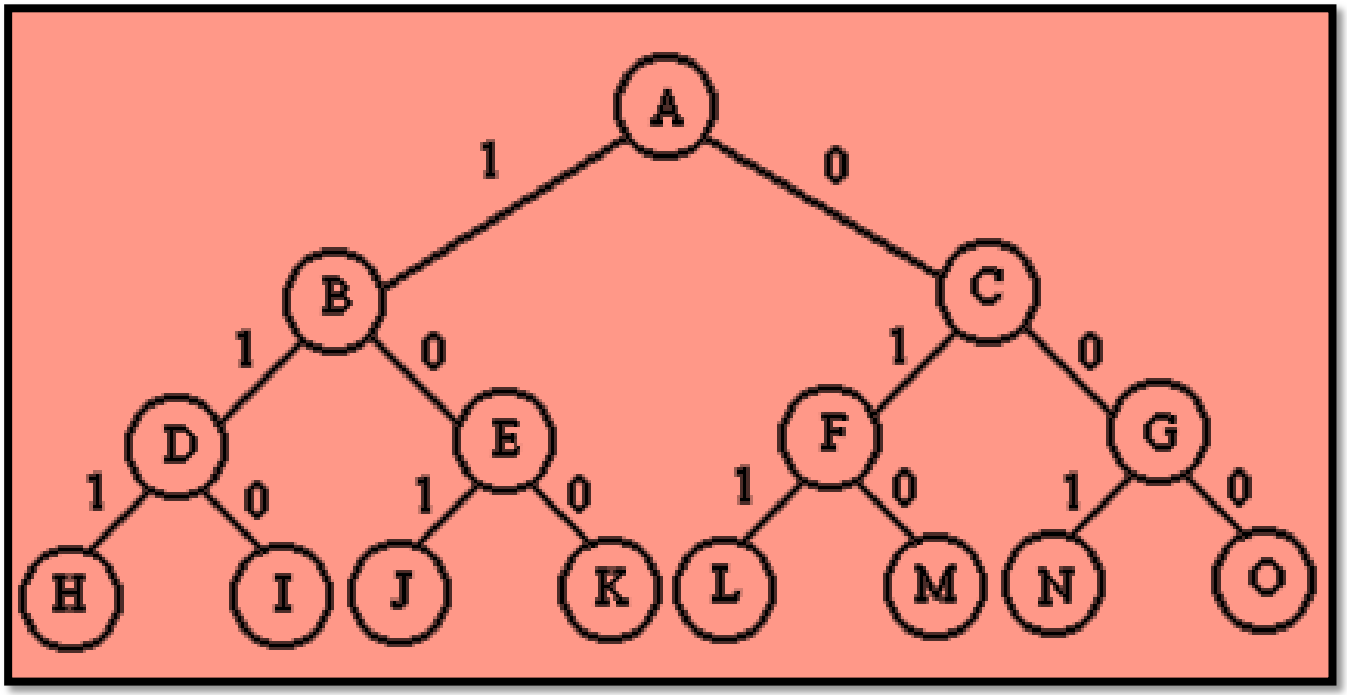


0-1背包问题

- 解向量：$(x_0, x_1, \cdots, x_{n-1})$
- 状态空间树：有2^n个叶子节点，其结点总数有$2^{n+1}-1$个。
- 根结点到叶结点的路径，是问题的可能解。
- 假定：第i层的左子树，表示物体X_i被装入背包的情况；右子树，表示物体X_i未被装入背包的情况。

0-1背包问题

- 解空间：子集树
- 可行性约束函数：$\sum_{i=1}^{n} w_i x_i \le c_1$



求解过程

- 初始化：目标函数上界为0，物体按价值重量比的非增顺序排序，
- 搜索过程：尽量沿左儿子结点前进，当不能沿左儿子继续前进时，就得到问题的一个部分解，并把搜索转移到右儿子子树。
- 估计由部分解所能得到的最大价值
  - 估计值高于当前上界：继续由右儿子子树向下搜索，扩大部分解，直到找到可行解；保存可行解，用可行解的值刷新目标函数的上界，向上回溯，寻找其它可行解；
  - 若估计值小于当前上界：丢弃当前正在搜索的部分解，向上回溯。

右子树中可能有最优解

右子树中不可能有最优解

0-1背包问题

- C——背包容量
- w[i]——第i个物品重量
- p[i]——第i个物品价值
- x[i]]——第i个物品是否装包
- bestp——当前最优价值
- cp——当前价值
- cw ——当前重量

0-1背包问题

```
Backtrack (int i)
{  if(i>n)      // 到达叶节点
    { if (bestp<cp) bestp=cp;
   return;}
   if(cw+w[i]<=c) // 进入左子树
    { x[i]=1;
      cw+=w[i];
       cp+=p[i];
      backtrack(i+1);
      cw-=w[i];
       cp-=p[i];   }
   // 装满背包
   if( bound(i+1)>bestp)
   { x[i]=0;
       backtrack(i+1);
       }
   }
```

求出最优值

可装入，试探进入左子树

0-1背包问题

- 上界函数：

```
Bound(int i)
{// 计算上界
  cleft = c - cw;  // 剩余容量
  b = cp;
  // 以物品单位重量价值递减序装入物品
  while (i <= n && w[i] <= cleft)
  {
    cleft -= w[i];
    b += p[i];
    i++;
    }
  // 装满背包
  if (i <= n)
      b += p[i]/w[i] * cleft;
  return b;
  }
```

cw当前装入包裹重量

cp当前装入包裹价值

练习：0-1背包问题

- 已知：
  - n=5, c=100,
  - w={30, 20, 20, 50, 40},
  - V={65, 40, 30, 60, 40}
  - 求最优解并画出搜索过程

```
Backtrack (int i)
{   if(i>n)      // 到达叶节点
     { if (bestp<cp) bestp=c
    return;}
     if(cw+w[i]<=c) // 进入左子树
      { x[i]=1;
        cw+=w[i];
         cp+=p[i];
        backtrack(i+1);
        cw-=w[i];
         cp-=p[i];   }
```

```
// 装满背包
if( bound(i+1)>bestp)
{ x[i]=0;
     backtrack(i+1);
     }
}
```

课堂练习：子集和的问题

- 假设有n个不同的正整数，找出这些数中所有使其和为正整数m的组合。
  - 例如：n=4，w={11,13,24,7}，m=31
  - 则相应的子集和数问题的解是{0,0,1,1}，{1,1,0,1}

解空间：

1、解的形式： $(x_1,x_2,\ldots,x_n)$

2、显约束条件： x_i=0或1

3、隐约束条件: 子集和等于m

4、解空间树

课堂练习：子集和的问题

```
Sumofsub()
{ for(i=1;i<=n;i++) x[i]=0;
  s=0; k=1; x[1]=1;
  do{ if(s+w[k]==m)        //输出解；
      if(k<n){if(s+w[k]<m) s=s+w[k];
              else x[k]=0;
              k=k+1;x[k]=1;}
          else{ x[n]=0;
                while(x[k-1]==0 and k>=1) k=k-1;
                x[k-1]=0;
                s=s-w[k-1];
                x[k]=1;}
  } while(k>0)
  if(k<=0)          //无解；  }
```

S:累加器，k：下标

试探

K>=n，回溯

前一项为0则一直回溯

正确算法

```
void Sumofsub(int m,int n, int w[5])
int i=1,s=0, k=1;
 int x[5];
 for(i=1;i<=n;i++) x[i]=0;
   x[1]=1;
do{
if(s+w[k]==m)for(i=1;i<=n;i++)
printf("%d ",x[i]);
      if(k<n){if(s+w[k]<m) s=s+w[k];
               else x[k]=0;
              k=k+1;x[k]=1; }
else
       { x[n]=0;
         while(x[k-1]==0 && k>=1)
k=k-1;
         x[k-1]=0;
         s=s-w[k-1];
         x[k]=1;}

    } while(k>0);

 if(k<=0)   printf("no resolve ")
 }
```

递归回溯

```
global integer M,n; global real W(1:n); global boolean X(1:n)
real  r,s; integer k,j
     //生成左儿子//
   X(k)←1
   if s+W(k)=M then
     print(X(j),j←1 to k)
   else
     if s+W(k)+W(k+1)<=M then
        call SUMOFSUB(S+W(k),k+1,r-W(k))
    endif
 endif
     //生成右儿子和计算Bk的值//
 if   s+r-W(k)>=M and s+W(k+1)<=M
 then X(k)←0
          call SUMOFSUB(s,k+1,r-W(k))
 endif
end SUMOFSUB
```

5.6 装载问题

- 装载问题是最优装载问题的变形。
- 有一批共 n 个集装箱要装上2艘载重量分别为c_1和c_2的轮船，其中集装箱 i 的重量为w_i，且 $\sum_{i=1}^{n} w_i \le c_1 + c_2$

  装载问题要求确定是否有一个合理的装载方案可将这 n 个集装箱装上这2艘轮船。如果有，找出一种装载方案。
- 当 $\sum_{i=1}^{n} w_i = c_1 + c_2$ 时，装载问题等价于**子集和问题**；
- 当$c_1=c_2$且 $\sum_{i=1}^{n} w_i = 2c_1$ 时，装载问题等价于**划分问题**。
- 二者都是NP难问题，故装载问题也是NP难的。

5.6 装载问题

- 容易证明，如果一个给定装载问题有解，则采用下面的策略可得到最优装载方案。
  - (1) 首先将第一艘轮船尽可能装满；
  - (2) 将剩余的集装箱装上第二艘轮船。

5.6 装载问题

- 将第一艘轮船**尽可能装满**等价于选取全体集装箱的一个子集，使该子集中集装箱**重量之和最接近**c_1。由此可知，装载问题等价于以下特殊的0-1背包问题。

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

$$\text{s.t.} \sum_{i=1}^{n} w_i x_i \le c_1$$

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

0-1背包中此处为价值*v*

5.6 装载问题

- 解空间：子集树
- 可行性约束函数(选择当前元素)：$\sum_{i=1}^{n} w_i x_i \le c_1$
- 限界函数(不选择当前元素)：当前载重量cw+剩余集装箱的重量r≤当前最优载重量$bestw$

- 用回溯法设计解装载问题的$O(2^n)$计算时间算法。在某些情况下(即$c_1>2^n$)，该算法优于动态规划算法$O(\min\{c_1, 2^n\})$

装载问题

采用递归回溯

子集树第i层

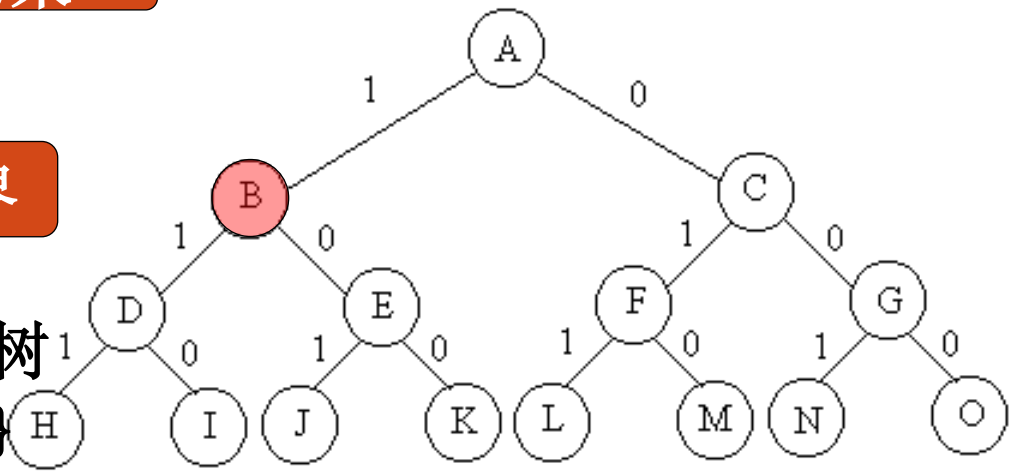
```
void backtrack (int i)
  {// 搜索第i层结点
    if (i > n)  // 到达叶结点
    更新最优解bestx,bestw;return;
    r -= w[i];
    if (cw + w[i] <= c) {// 搜索左子树
      x[i] = 1;
      cw += w[i];
      backtrack(i + 1);
      cw -= w[i];     }
    if (cw + r > bestw)  {
      x[i] = 0;  // 搜索右子树
      backtrack(i + 1);     }
    r += w[i];
  }
```

剩余集装箱重量

约束

恢复状态

限界



课堂练习

例3：在任意给定的字符表(例如：‘1’，‘2’，‘3’)上，生成一个由该字符表字符组成、含n个字符的序列，但要求生成的序列中没有两个相邻的子序列是相同的。

例如：对于n=5,序列“12321”是问题的一个解，而序列“12323”因有两个相邻的子序列都为“23”，所以该序列不是问题的解。

解题思路：为找到一个满足要求的长为n个字符的序列，从空序列开始，每次检查当前序列是否含有两个相邻的子序列。在没有两个相邻的子序列相同的情况下，在序列之后添加一个字符，让序列延长。如果当前序列有两个相邻的子序列一样时，就改变序列。如此重复执行延长、检查或修改、检查，直到找到一个满足问题要求的解。

练习

```
算法：{int n,m;
       int good;
       char s[MAXLEN];
       输入欲求序列长度n;
       m=0;
       good=1;
       do{
            if(good)     {延长序列;m++;}
            else         {改变序列;若所有字符都尝试过，则m--}
            good=检查当前序列是否合理的结果;
        }while((!good||m!=n)&&(m!=0));
       if(m!=0)      输出解;
       else          输出无解;
      }
```