算法设计与分析

第5章 回溯法
（1）

谢晓芹
哈尔滨工程大学计算机科学与技术学院

学习要点

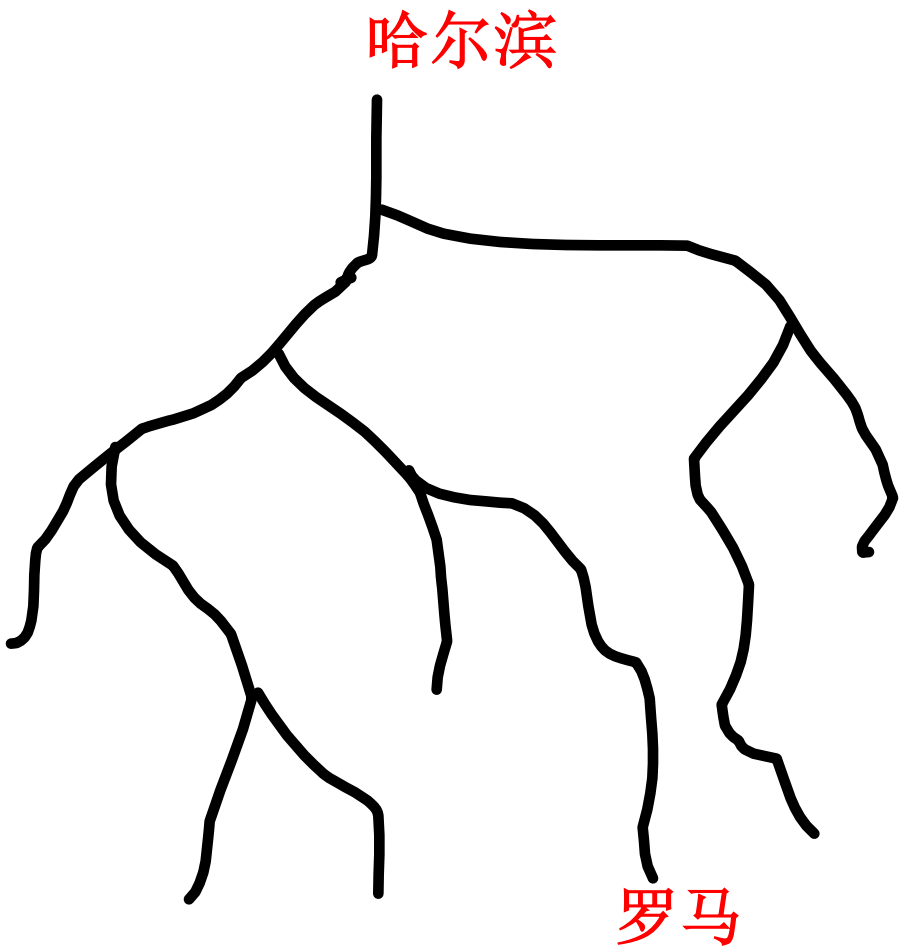
- 理解回溯法的深度优先搜索策略
- 掌握用回溯法解题的算法框架
  - 递归回溯
  - 迭代回溯
  - 子集树算法框架
  - 排列树算法框架

学习要点

- 通过应用范例学习回溯法的设计策略
  - 0-1背包问题
  - 旅行售货员问题
  - 装载问题
  - 批处理作业调度
  - n后问题
  - 图的m着色问题

回溯法 - 引言

- 实例
  - 哈尔滨走到罗马
  - 若完全不认识路，会怎样走



- **两类问题**
  - ✓ 存在性问题
  - ✓ 优化问题

可行解
最优解

To be or not to be
How to be

回溯法 - 引言

- 举例：0-1背包问题
  - 优化问题：给定n个物品，物品i的价值为v_i,重量w_i，以及一个总重量C；要求找出一个物品选择的组合，使其总重量不超过C的情况下，总价值最高
  - 决策问题：给定n个物品，物品i的价值为v_i,重量w_i，以及一个总重量C和**实数m**，**是否存在**一个物品组合，其价值大于等于m，而总重量小于等于C?
  - 问题的解能表示成一个n-元组$(x_1,\ldots, x_n)$, 其中x_i取自有穷集S_i={0,1}, 可能元组数2^n

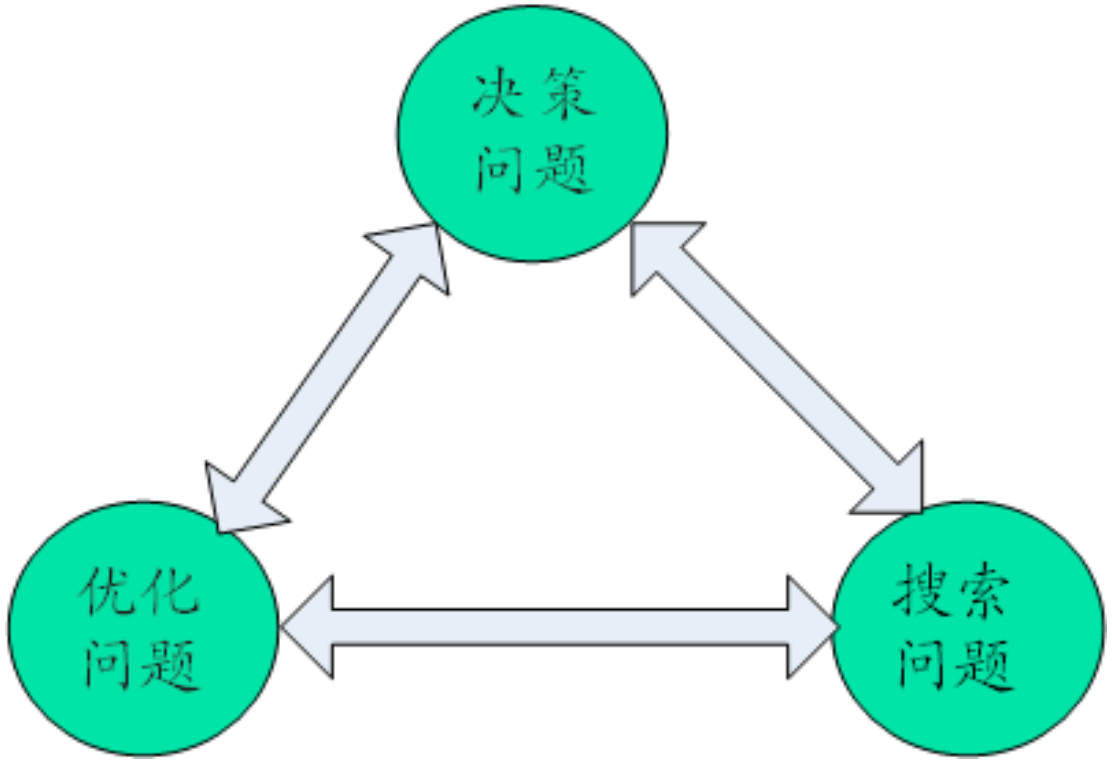
To be or not to be
How to be

回溯法 - 引言

- 问题描述
  - 假定问题的解能表示成一个n-元组$(x_1,\ldots, x_n)$,其中x_i取自某个有穷集S_i
  - 所有这些n-元组构成问题的解空间
  - 假设集合S_i的大小是m_i,可能元组数为m= $m_1 m_2 \ldots m_n$
- 两类问题
  - 存在性问题
    - 求满足某些条件的一个或全部元组，这些条件称为约束条件
    - 如果不存在这样的元组，算法应返回No
  - 优化问题
    - 给定一组约束条件，在满足约束条件的元组中求使某目标函数达到最大（小）值的元组
    - 满足约束条件的元组称为问题的可行解

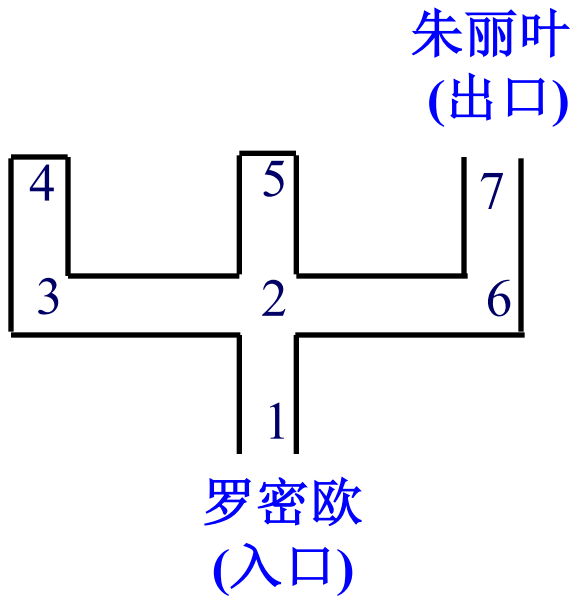
回溯法 - 引言

- 问题间的转化



回溯法 - 引言

- 实例——罗密欧与朱丽叶问题
  - 迷宫是一些互相连通的交叉路口的集合，给定一个入口、一个出口。当从入口到出口存在通路时，输出选中的一条通路；否则，输出无通路存在



(a) 交叉路口

✓ **路口动作结果**

1向前进入2
2向左进入3
3向右进入4
4（死路）回溯进入3
3（死路）回溯进入2
2向前进入5
5（死路）回溯进入2
2向右进入6
6向左进入7

(b)搜索过程

回溯法 - 引言

回溯法与穷举查找是一样的吗?

- 回溯法和分支限界法看成是穷举法的一个改进
  - 智能性
  - 该方法至少可以对某些组合难题的较大实例求解.
- 不同
  - 每次只构造侯选解的一个部分
  - 然后评估这个部分构造解
    - 如果加上剩余的分量也不可能求得一个解,就绝对不会生成剩下的分量
- 最坏情况: 指数级爆炸问题

回溯法 - 相关概念

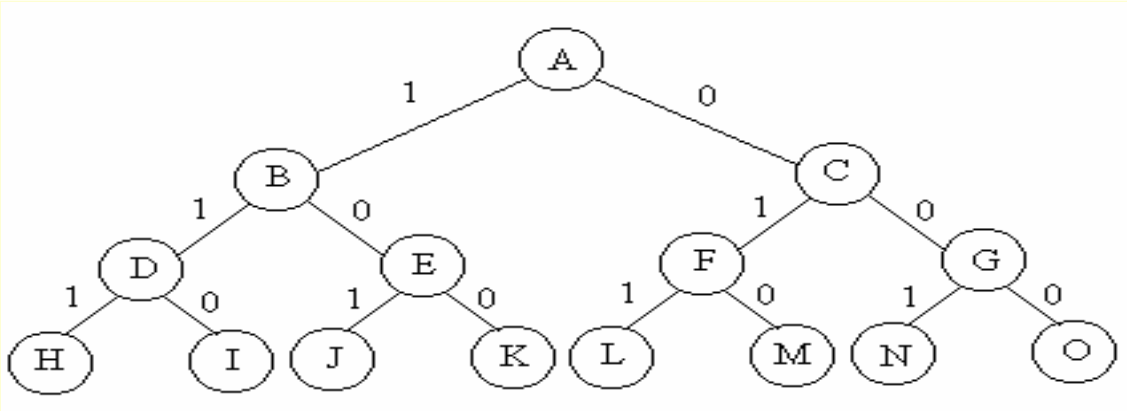
- 在包含问题的所有解的解空间树中，以深度优先的方式系统地搜索问题的解的算法称为回溯法
- 使用场合
  - 对于许多问题，当需要找出它的解的集合或者要求回答什么解是满足某些约束条件的最佳解时，往往要使用回溯法
  - 这种方法适用于解一些组合数相当大的问题，具有“通用解题法”之称
- 回溯法的基本做法
  - 是搜索，或是一种组织得井井有条的，能避免不必要搜索的穷举式搜索法

回溯法 - 相关概念

- 问题的解向量
  - 回溯法希望一个问题的解能够表示成一个n元式(x_1,x_2, …, x_n)的形式。
- 显约束
  - 对分量x_i的取值限定。
- 隐约束
  - 为满足问题的解而对不同分量之间施加的约束。
- 解空间
  - 对于问题的一个实例，解向量满足显式约束条件的所有多元组，构成了该实例的一个解空间。

回溯法 - 相关概念

- 例如: 有n种可选择物品的0-1背包问题
  - 解空间由长度为n的0-1向量组成
    - n=3 : {(0,0,0),(0,0,1),(0,1,0),(1,0,0),(0,1,1), (1,0,1),(1,1,0),(1,1,1)}
  - 显约束
    - $x_i \in \{0,1\}$
  - 隐约束 $\max \sum_{i=1}^{n} v_i x_i, \quad \sum_{i=1}^{n} w_i x_i \leq C$
  - 解空间结构: “树”或“图”



  - 注意
    - 同一个问题可以有多种表示，有些表示方法更简单，所需表示的状态空间更小（存储量少，搜索方法简单）

回溯法 - 相关概念

- 回溯法：
  - 为了避免生成那些不可能产生最佳解的问题状态，要不断地利用限界函数(bounding function)来处死那些实际上不可能产生所需解的活结点，以减少问题的计算量
  - 具有**限界函数**的**深度优先搜索**法称为回溯法
- 具体做法
  - 系统性
    - 回溯法在问题的解空间树中，按深度优先策略，从根结点出发搜索解空间树
  - 跳跃性
    - 算法搜索至解空间树的任意一点时，先判断该结点是否包含问题的解
      - 如果肯定不包含，则跳过对该结点为根的子树的搜索，逐层向其祖先结点回溯
      - 否则，进入该子树，继续按深度优先策略搜索

人工智能: 解空间树的不同生成策略

解空间树是实际存在的吗?

回溯法的基本思想

- 回溯法求解步骤
  - 1、针对所给问题，定义问题的解空间
  - 2、确定易于搜索的解空间结构
  - 3、以深度优先方式搜索解空间，并在搜索过程中用剪枝函数避免无效搜索
- 常用剪枝函数
  - 用约束函数在扩展结点处剪去不满足约束的子树；
  - 用限界函数剪去得不到最优解的子树。

0-1背包问题

- 给定n种物品和一背包。物品i的重量是w_i，其价值为v_i，背包的容量为C。问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?
  - 输入：C>0, w_i >0, v_i >0, 1≤ i≤n
  - 输出：$(x_1, x_2, \ldots, x_n)$, $x_i \in \{0, 1\}$, 满足 $\max \sum_{i=1}^{n} v_i x_i, \sum_{i=1}^{n} w_i x_i \leq C$
- 实例:
  - 物品个数为 n=3，背包的容量为 C=30
  - 物品的重量分别为 w={16，15，15}, 价值分别为 p={45，25，25}
  - 问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?
- 问题解空间：（x_1, x_2, x_3）
  - {(0,0,0),(0,1,0),(0,0,1),(1,0,0), (0,1,1),(1,0,1),(1,1,0),(1,1,1)}

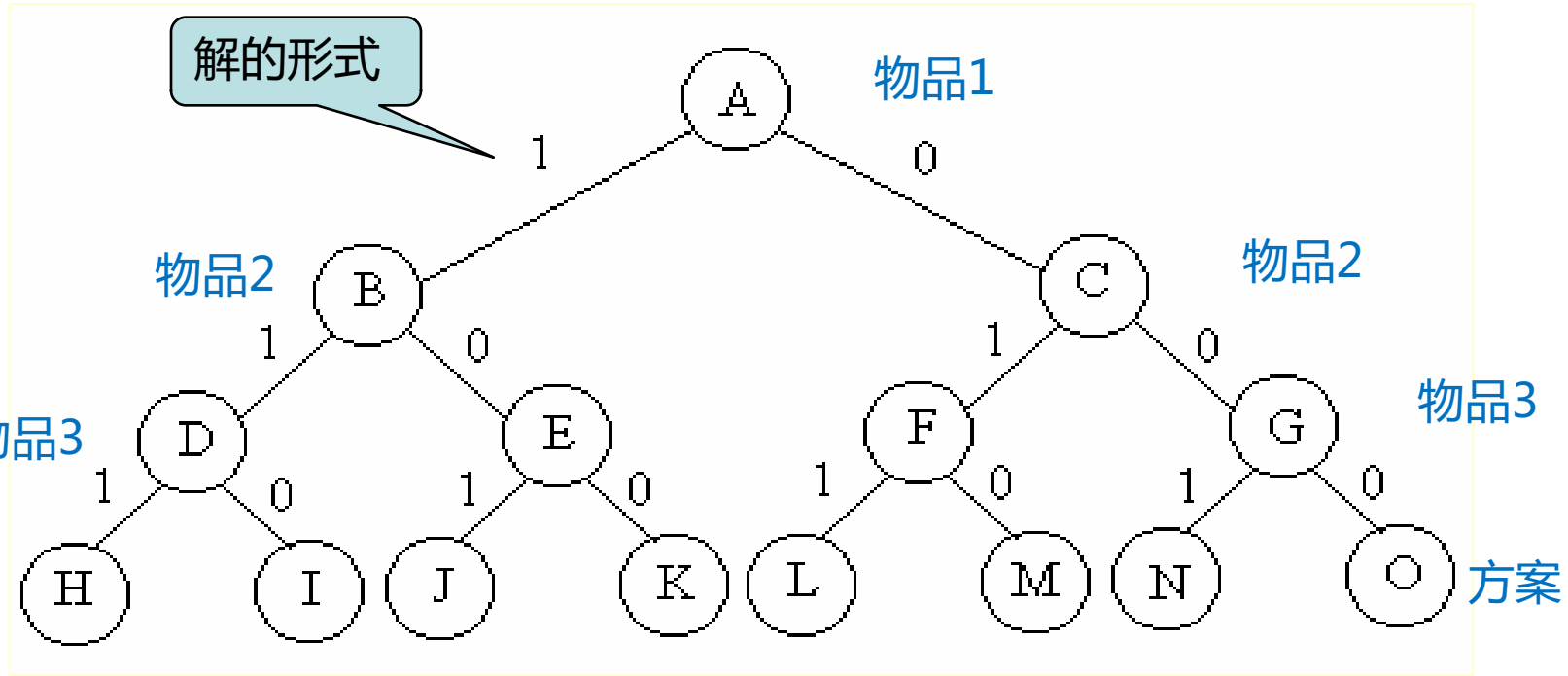
解的内容

0-1背包问题

✓ 步骤1:定义问题解空间： （ x_1,x_2,x_3 ）

$\{(0,0,0),(0,1,0),(0,0,1),(1,0,0),(0,1,1),(1,0,1),(1,1,0),(1,1,1)\}$

✓ 步骤2:确定解空间树结构



- 解空间：子集树
- 可行性约束函数：
  $\sum w_i x_i \le C$
- 上界函数：
  Bound()
- 算法：略

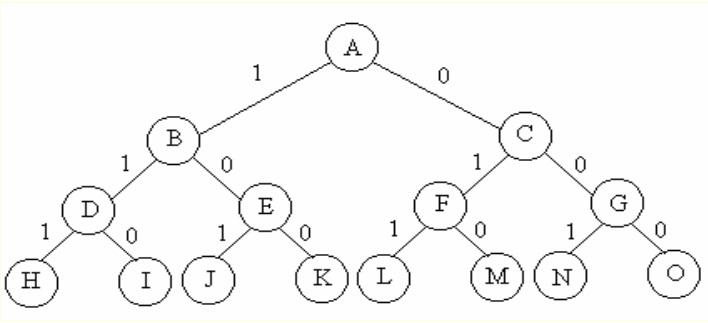
0-1背包问题

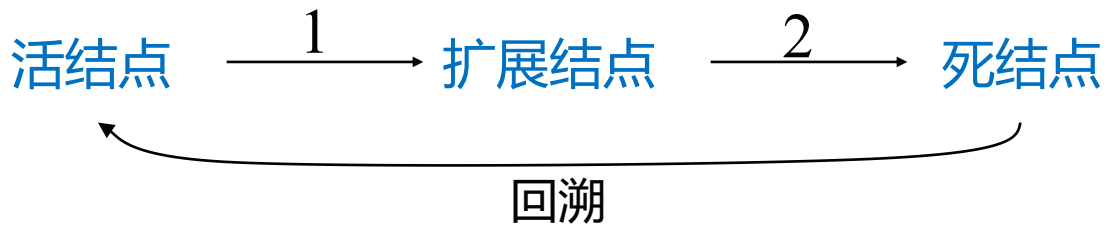
- 定义（子集树）
  - 当所给的问题是从n个元素的集合S中找出满足某种性质的**子集**时，相应的解空间树称为子集树
  - 0-1背包问题子集树通常有 2^n 个叶结点，结点总数为 $2^{n+1}-1$
  - 遍历解空间树需要 $\Omega(2^n)$的计算时间
- 例如
  - 0-1背包问题：3个元素（x_1,x_2,x_3），每个元素的集合为{0，1}，搜索空间为：{(0,0,0), (0,0,1), (0,1,0), (1,0,0), (0,1,1),(1,0,1),(1,1,0),(1,1,1)},状态有$2^3=8$种

部分解

回溯法 - 相关概念

- 活结点
  - 指一个自身已生成(访问)但其儿子还没有全部生成(访问)的节点
- 扩展结点
  - 一个正在产生儿子的节点（正在访问）
- 死结点
  - 指一个所有儿子已经产生的结点. 死节点都是无法扩展的节点
- 深度优先的问题状态生成法（每个节点对应一个问题状态）
  - 如果对一个扩展结点R，一旦产生了它的一个儿子C，就把C当做新的扩展结点。在完成对子树C（以C为根的子树）的穷尽搜索之后，将R重新变成扩展结点，继续生成R的下一个儿子（如果存在）



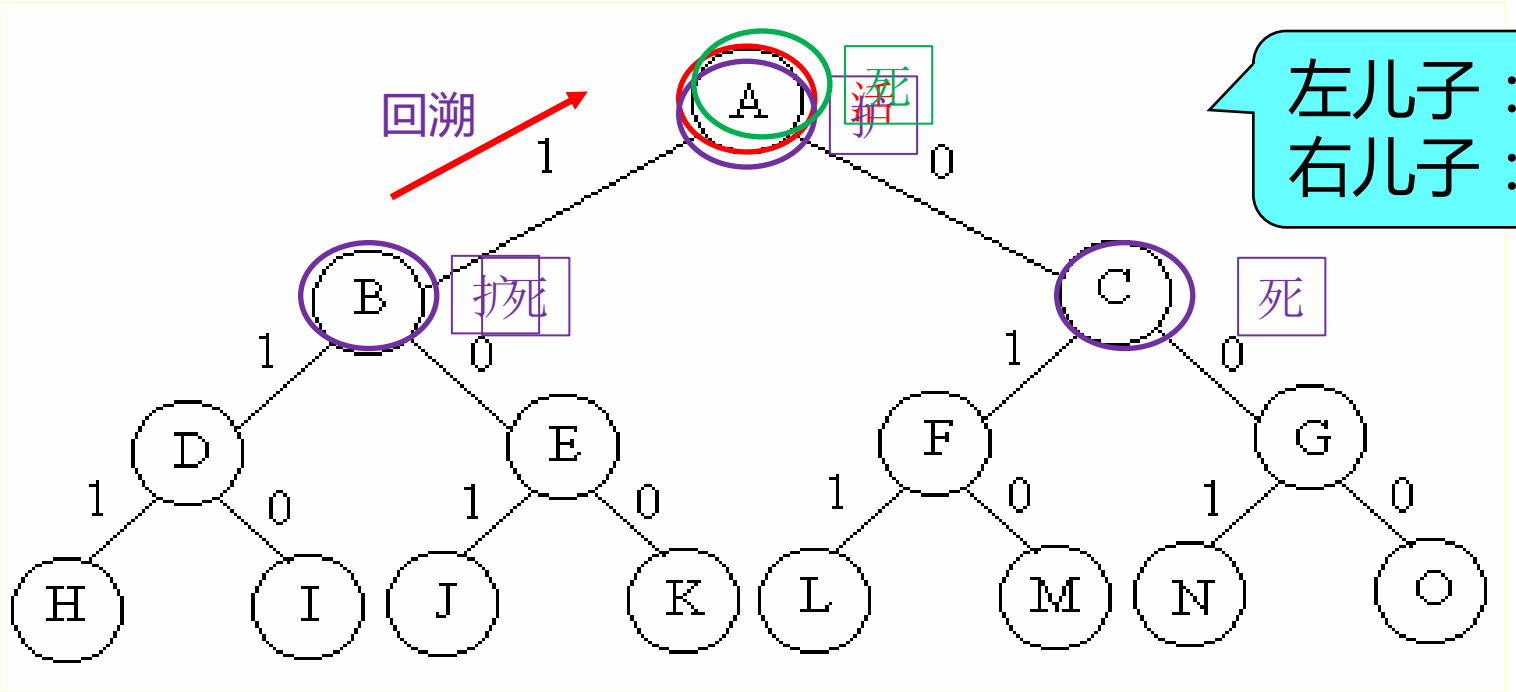


结束条件:
1)找到解
2)解空间无活结点

0-1背包问题

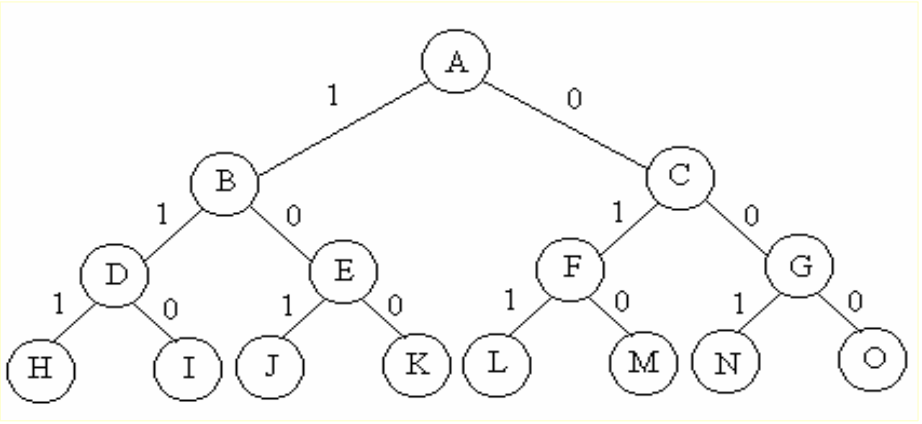
- 问题
  - n=3，C=30，w={16，15，15}，p={45，25，25}
    可行性约束函数： $\sum_{i=1}^{n} w_i x_i \leq C$

✓ 解空间树——扩展结点，活结点，死结点



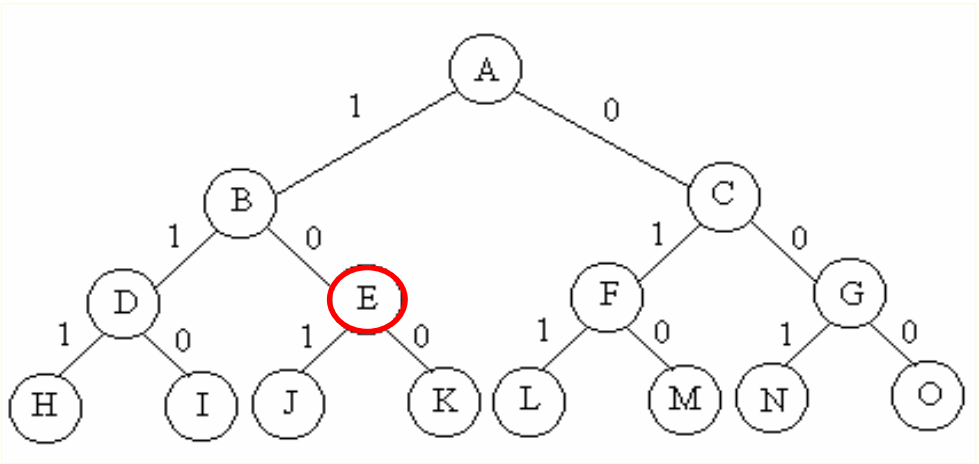
0-1背包问题

- 步骤3. 深度优先搜索解空间，剪枝避免无效搜索
  - 深度优先
    - 活结点->扩展结点->死结点的扩展过程
  - 剪枝
    - 是解吗？可行性约束constraint() $\sum w_i x_i \leq C$ 左子树
    - 是最优解吗？上界函数bound() 右子树



0-1背包问题

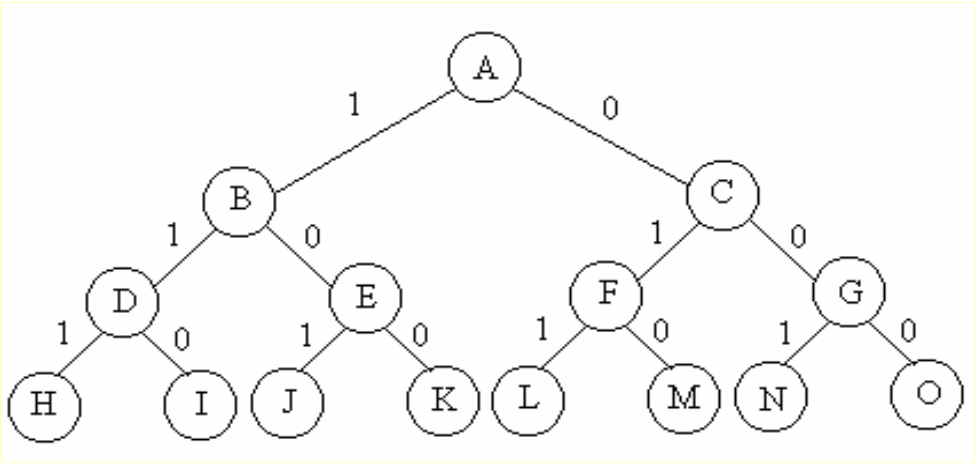
- 限界函数Bound()
  - 当前价值cp+剩余价值r ≤当前最优值bestp？
    - 当cp+r<=bestp时,可剪去右子树



- 贪心思想的上界计算方法
  - 将剩余物品按单位重量价值排序
  - 依次装入物品
  - 直至装不下时，装入该物品的一部分而装满背包

0-1背包问题

- 例子: 贪心思想的上界计算方法
  - n=3，C=30，w={16，15，15}，p={45，25，25}
  - 分析：p/w={45/16, 25/15, 25/15}={2.81, 1.67, 1.67}
    - Bound(A)=45+14*25/15=68.33
    - Bound(B)=45+14*25/15=68.33
    - Bound(C)=25+25=50
    - Bound(D)=68.33
    - Bound(E)=45+14*25/15=68.33
    - Bound(F)=25+25=50, bound(G)=25
    - Bound(H)=68.33, bound(I)=68.33, bound(J)=68.33, bound(K)=45, bound(L)=50, bound(M)=25, bound(N)=25, bound(O)=0



子集树的递归回溯实现框架

- 回溯法对解空间作深度优先搜索，因此，在一般情况下用递归方法实现回溯法

```
void backtrack (int t) //t扩展结点深度
{
    if (t>n) output(x); //n树高度
    else
      for (int i=f(n,t);i<=g(n,t);i++) {
        x[t]=h(i);
        if (constraint(t)&&bound(t))
              backtrack(t+1);
        }
}
```

f(n,t) /g(n, t):在当前扩展结点处未搜索过的子树的起始编号和终止编号

h(i): 当前扩展结点处x[t]的第i个可选值

是否可能为解？
constraint()=true 在当前扩展结点处x[1:t]取值满足约束条件，否则剪枝

是否可能为最优解？
bound()=true 在当前扩展结点处x[1:t]取值未使目标函数越界，否则剪枝

子集树的迭代回溯实现框架

- 采用树的非递归深度优先遍历算法，可将回溯法表示为一个非递归迭代过程

```
void iterativeBacktrack ()
{
  int t=1;
  while (t>0) {
    if (f(n,t)<=g(n,t))    //有1个以上孩子
      for (int i=f(n,t);i<=g(n,t);i++) {
        x[t]=h(i);
        if (constraint(t)&&bound(t)) {
          if (solution(t)) output(x);
          else t++;}
        }
    else t--;  //回溯
    }
}
```

f(n,t) /g(n, t):在当前扩展结点处未搜索过的子树的起始编号和终止编号

是部分解,继续纵向搜索

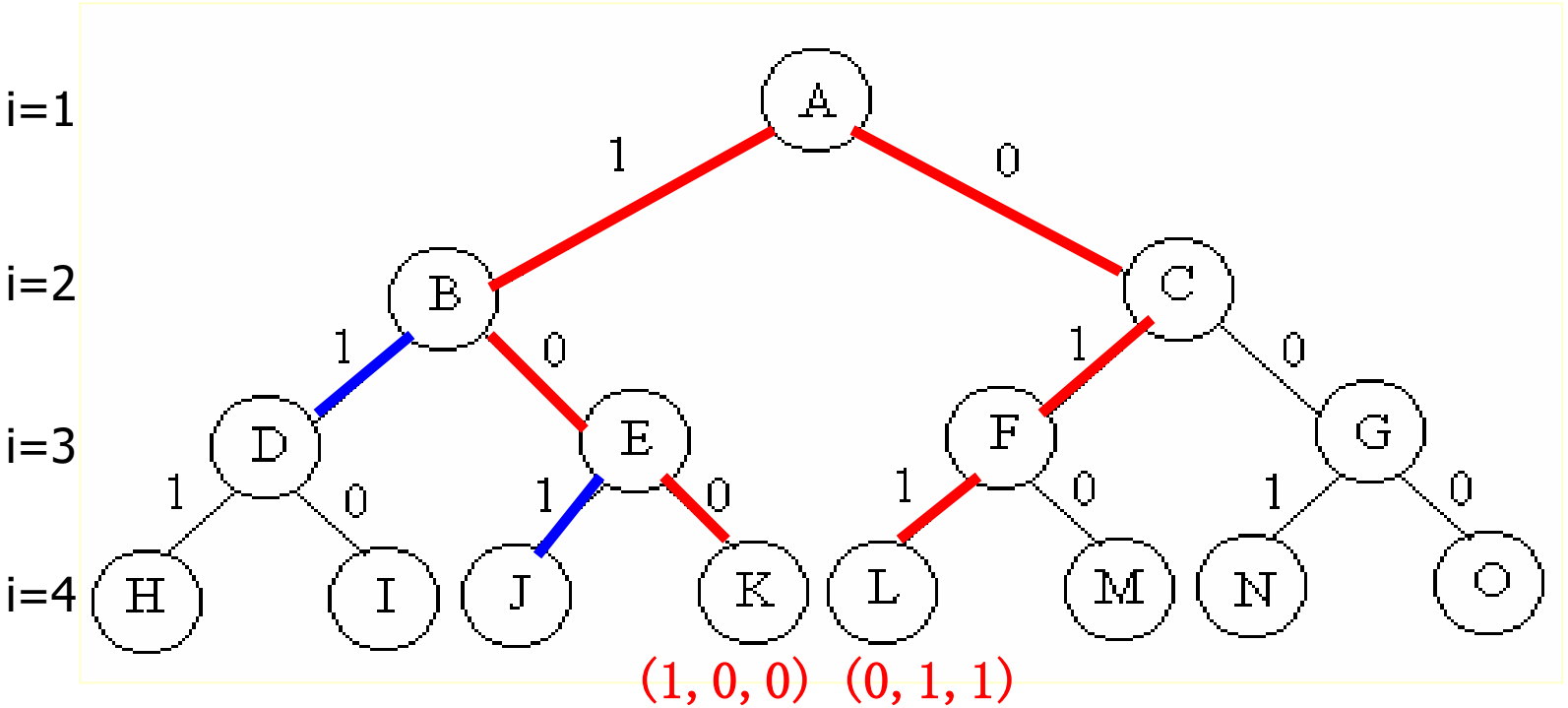
0-1背包问题

- 例子： n=3，C=30，w={16，15，15}，p={45，25，25}

✓ 问题解空间： （x_1,x_2,x_3）

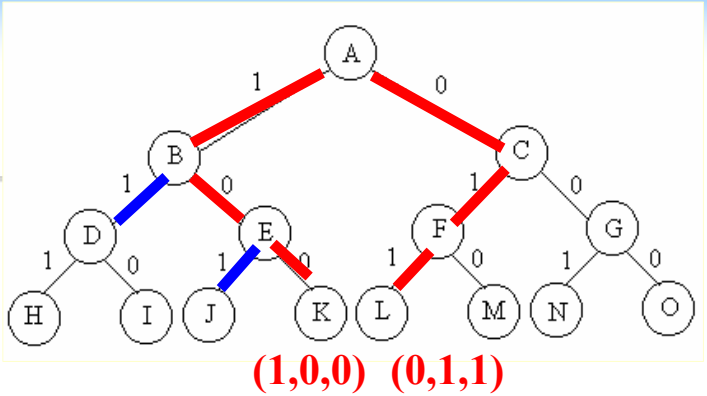
{(0,0,0),(0,1,0),(0,0,1),(1,0,0),(0,1,1),(1,0,1),(1,1,0),(1,1,1)}

✓ 解空间树——扩展结点，活结点，死结点



0-1背包问题

例子： n=3，C=30，w={16，15，15}，p={45，25，25}



i=1
i=2
i=3
i=4

	活结点表	扩展结点	死结点	处理过程	当前价值cp	
t=1	A	A		A→B,C	cp=45	Con(B)=16<30 ✓ bestp=0
t=2	AB	B		B→D,E		Constraint(D)=31 X,bound(E)=68.33
t=3	ABE	E		E→J,K		Constraint(J)=31X, bound(K)=45 ✓
t=4	ABEK	K	K	K为叶子结点,可行解(1,0,0)	Bestp=45	
	ABE	(bt)E	E	E所有孩子已处理		
	AB	(bt)B	B			
	A	(bt)A		A→C		bound(C)=50>bestp=45 ✓
	AC	C		C→F,G		Constraint(F)=15<30 ✓
	ACF	F		F→L,M		Constraint(L)=30<=30 ✓
	ACFL	L	L	L为叶子结点,可行解(0,1,1)	bestp=50	
	ACF	(bt)F	F	F→M		bound(M)=25<50 X
	AC	(bt)C	C	C→G		bound(G)=25<50 X
	A	(bt)A	A			

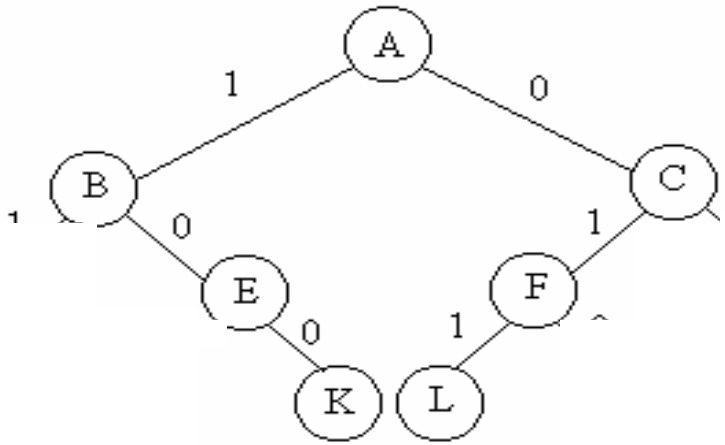
0-1背包问题

- 例子：
  - n=3，C=30，w={16，15，15}，p={45，25，25}

注意：物品要按照P/W排序

- 回溯法搜索过程中的扩展结点表顺序为
  - ABEKEBACFLFCA

- 实际生成的子集树为：



回溯法的基本思想

- 回溯法求解步骤
  - 1、针对所给问题，定义问题的解空间　　内容　枚举
  - 2、确定易于搜索的解空间结构　　形式　边，结点
  - 3、以深度优先方式搜索解空间，在搜索过程中用剪枝函数避免无效搜索

约束,限界

- 常用剪枝函数
  - ✓ 用约束函数在扩展结点处剪去不满足约束的子树
  - ✓ 用限界函数剪去得不到最优解的子树

0-1背包问题

- 实例
  (1)物品个数为 n=4
  (2)背包的容量为 C=7
  (3)物品的重量分别为 w={3，5，2，1}
  (4)物品的价值分别为 p={9，10，7，4}
- 可行性约束函数
- 限界函数（上界函数）： Bound(i)
- 请画出解空间树，并计算各个结点的上界值，写出扩展结点表顺序

0-1背包问题

- 贪心策略上界计算方法
  - 以物品单位重量价值的递减序排序

C=7

id	4	3	1	2
p	4	7	9	10
w	1	2	3	5
d=p/w	4	3.5	3	2
sorted id	[1]	[2]	[3]	[4]

- **物品按照d的顺序装入—— 4，3，1，2**
- **产生一个非可行解(上界)— x=[1,0.2,1,1], 价值22（=4+7+9+1*10/5）**

0-1背包问题

- 贪心思想计算上界

```
Bound(int i){        //计算上界
      Typew cleft=c-cw;      //剩余容量
      Typep b=cp;            //当前价值
       //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];              //w[i]为物品重量
          b+=p[i];                  //p[i]为物品价值
          i++;
       }
       if (i<=n) b+=p[i]/w[i]*cleft;
       return b;
}
```

回溯法

- 剪枝策略
  - Constraint(t):
    - True:当前扩展结点处的取值满足问题的约束条件
    - False:当前扩展结点处的取值不满足问题的约束条件, 可剪去子树
  - Bound(t):
    - True:当前扩展结点处的取值未使目标函数越界
    - False:当前扩展结点处的取值已使目标函数越界,可剪去子树

0-1背包问题-回溯算法

- 回溯算法

```
Backtrack(int i){
    if (i>n){ bestp=cp; return;}  //得到一个可行解
    if (cw+w[i]<=c){          //x[i]=1，左子树
        cw+=w[i];             //cw是当前重量
        cp+=p[i];             //cp是当前价值
        Backtrack(i+1);       //左子树深入
        cw-=w[i];
        cp-=p[i];
    }
    if (Bound(i+1)>bestp)  //x[i]=0; 右子树
        Backtrack(i+1);       //右子树深入
}
```

回溯

可行性约束
Constraint(t)

限界约束
bound()

0-1背包问题-回溯算法

```
void backtrack (int t) { //扩展结点深度
    if (t>n) output(x); //n树高度
    else
      for (int i=f(n,t);i<=g(n,t);i++) {
        x[t]=h(i);
        if (constraint(t)&&bound(t))
                backtrack(t+1);
      }
}
```

```
Backtrack(int i){
   if (i>n){ bestp=cp; return;}
   if (cw+w[i]<=c){ //x[i]=1，左子树
      cw+=w[i];        //cw是当前重量
      cp+=p[i];        //cp是当前价值
      Backtrack(i+1); //左子树深入
      cw-=w[i];
      cp-=p[i];
   }
   if (Bound(i+1)>bestp)  //x[i]=0; 右子树
      Backtrack(i+1);     //右子树深入
}
```

0-1背包问题-回溯算法

- 回溯算法分析
  - 计算上界bound()：O(n)
  - 最坏情况下有$O(2^n)$个右孩子需要计算上界
  - $Time=O(n*2^n))$

回溯法的基本思想

- 回溯法解题的一个显著特征是在搜索过程中动态产生问题的解空间。
- 算法只保存从根结点到当前扩展结点的路径。
  - 如果解空间树中从根结点到叶结点的最长路径的长度为h(n)，则回溯法所需的计算空间通常为O(h(n))。
  - 显式地存储整个解空间则需要$O(2^{h(n)})$或O(h(n)!)内存空间。

子集树　　排列树

小结

- 回溯法
  - 主要用于以下困难的组合问题：这些问题可能存在精确解,但无法用高效的算法求解
  - 回溯法不同于穷举查找法，回溯法提供了一种有价值的解题方法
- 回溯法解题步骤
  - 定义解空间　：内容
  - 确定解空间结构：　形式
  - 深度优先搜索，剪枝函数去掉无用搜索：约束+限界
    - 扩展结点顺序
    - 选择孩子结点，进行遍历
    - 处理步骤：叶子、内部结点

小结

- 重点和难点
  - 回溯法的深度优先搜索策略
  - 回溯法解题的算法框架
    （1）递归回溯
    （2）迭代回溯
    （3）子集树算法框架：0/1背包问题案例
  - 剪枝函数
    - 约束函数
    - 限界函数