


# 第5章 回溯法



有这样几类问题：

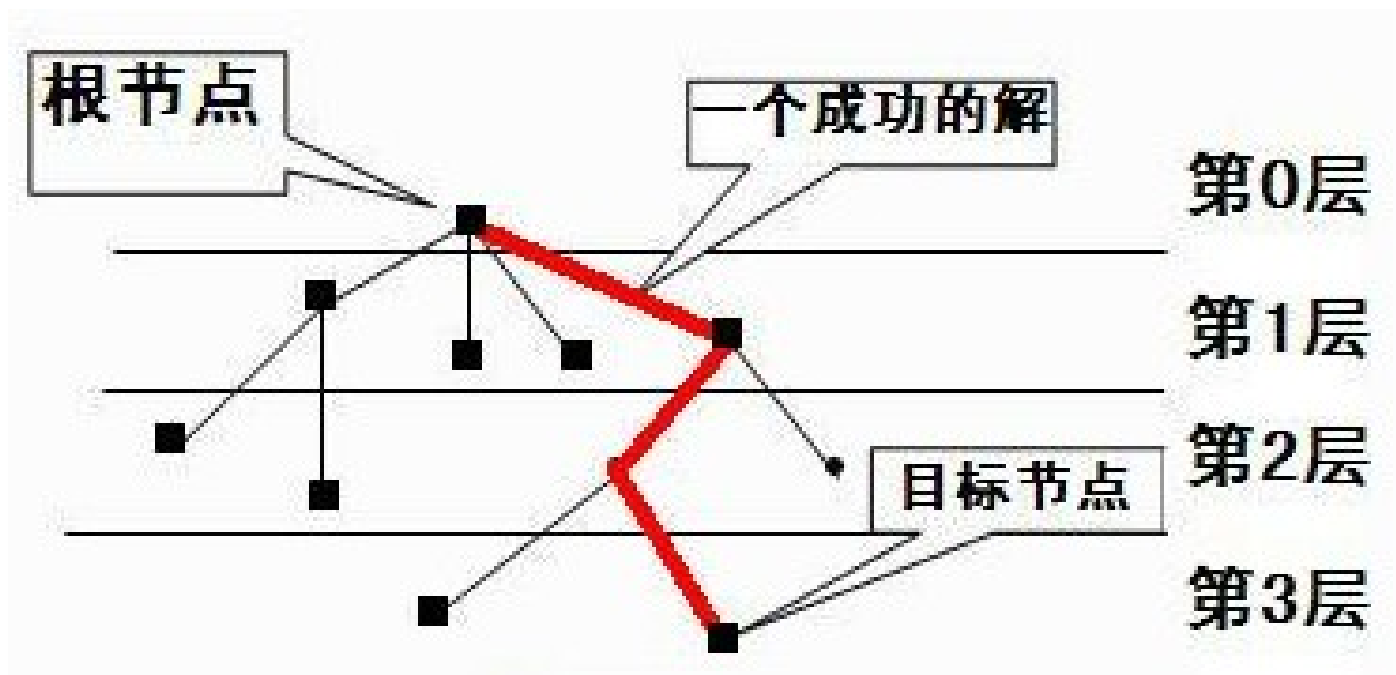
- 建立数学模型，在有限时间内，用解析的方法求解
- 建立数学模型，但在有限时间内，用数学解析的方法求解困难，只好用搜索或模拟来求解，常见方法有：
  - 穷举
  - 深度优先搜索方法
  - 广度优先搜索方法
  - 启发式算法

# 关于搜索算法

- 搜索是平常最常用的一种算法，同时应该也是最难掌握的一种算法。搜索类似于枚举、穷举。顾名思义就是将所有情况都试一下(这个“试”是有逻辑的顺序的)，当存在某种状态符合问题要求的时候，那个这个状态就是问题一个解。
- 搜索算法的效率通常是非常低的，时间复杂度一般都是级数增长的。所以在搜索算法的同时我们通常要加上优化，例如：剪枝，分枝定界，对称剪裁，记忆化搜索等，用这些优化来降低时间复杂度。

# 搜索算法:

根据初始条件和扩展规则，构造解空间树，并在解空间树中寻找符合目标状态。



## 1) 穷举，适用于：

可预见确定解元素个数，且问题规模不是特别大

对于每个解变量 $a_1, \dots, a_n$ 的可能值为一个连续的值域

## 2) 深度优先搜索，适用于：


求解初始结点到目标结点的所有方案

求解初始结点到目标结点的一种方案

## 3) 广度优先搜索，适用于：

求解初始结点所能达到的所有结点

求某一结点到某目标结点的最短路径



## 关于搜索算法

- 纯随机搜索 (**Random Walk**)
- 广度优先搜索 (**BFS**)
- 深度优先搜索 (**DFS**)
- 启发式搜索：启发式搜索就是：在状态空间中，对每一个搜索的位置进行评估，得到最好的位置，再从这个位置进行搜索直到目标（即：通过启发式函数，选择代价最少的结点作为下一步搜索结点而跳转其上），使搜索过程沿着被认为最有希望的前沿区段发展。

## 关于搜索算法

- 纯随机搜索 (**Random Walk**)
- 广度优先搜索 (**BFS**)
- 深度优先搜索 (**DFS**)
- 启发式搜索: 启发式搜索就是对每个搜索的位置进行评估, 得进行搜索直到目标 (即: 通过的结点作为下一步搜索结点, 沿着被认为最有希望的前沿区

### 盲目式搜索:

它的下一个搜索节点有着固定的“扩展控制方式”, 不会因在下一次搜索中哪个更优而去跳转到那个结点去搜索。

在最坏情况下, 均需要试探完整个解集空间, 显然, 只能适用于问题规模不大且搜索规则清晰的搜索问题中。

置少沿



## 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
- （1）递归回溯
- （2）迭代回溯
- （3）子集树算法框架
- （4）排列树算法框架





通过应用范例学习回溯法的设计策略。

- (1) **5.2** 装载问题;
- (2) **5.3** 批处理作业调度;
- (3) **5.5**  $n$ 后问题;
- (4) **5.6** 0-1背包问题;
- (5) **5.8** 图的 $m$ 着色问题;
- (6) **5.9** 旅行售货员问题。
- 其他 ..... (自阅)

# 搜索算法的基本思想

**适用问题：** 求解搜索问题

**搜索空间：** 一棵树

每个结点对应了部分解向量

树叶对应了可行解

**搜索过程：**

采用系统的方法隐含遍历搜索树

**搜索策略：**

深度优先，宽度优先，函数优先，宽深结合等

# 搜索算法的基本思想（续）

## 结点分支判定条件：

满足约束条件——分支扩张解向量

不满足约束条件，回溯到该结点的父结点

## 结点状态：

动态生成

结点状态：白结点（尚未访问）；

灰结点（正在访问该结点为根的子树）

黑结点（该结点为根的子树遍历完成）

存储：当前路径

# 一类组合优化问题

**目标函数**（极大化或极小化）

**约束条件**

搜索空间中满足约束条件的解称为**可行解**

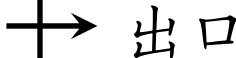
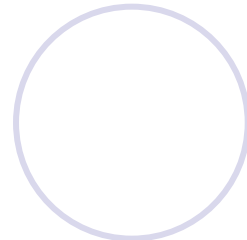
使得目标函数达到极大(或极小)的解称为**最优解**

如，0-1背包问题， $N=4$ ,  $P=\{1,3,5,9\}$ ,  $W=\{2,3,4,7\}$ ,  $C=10$ :

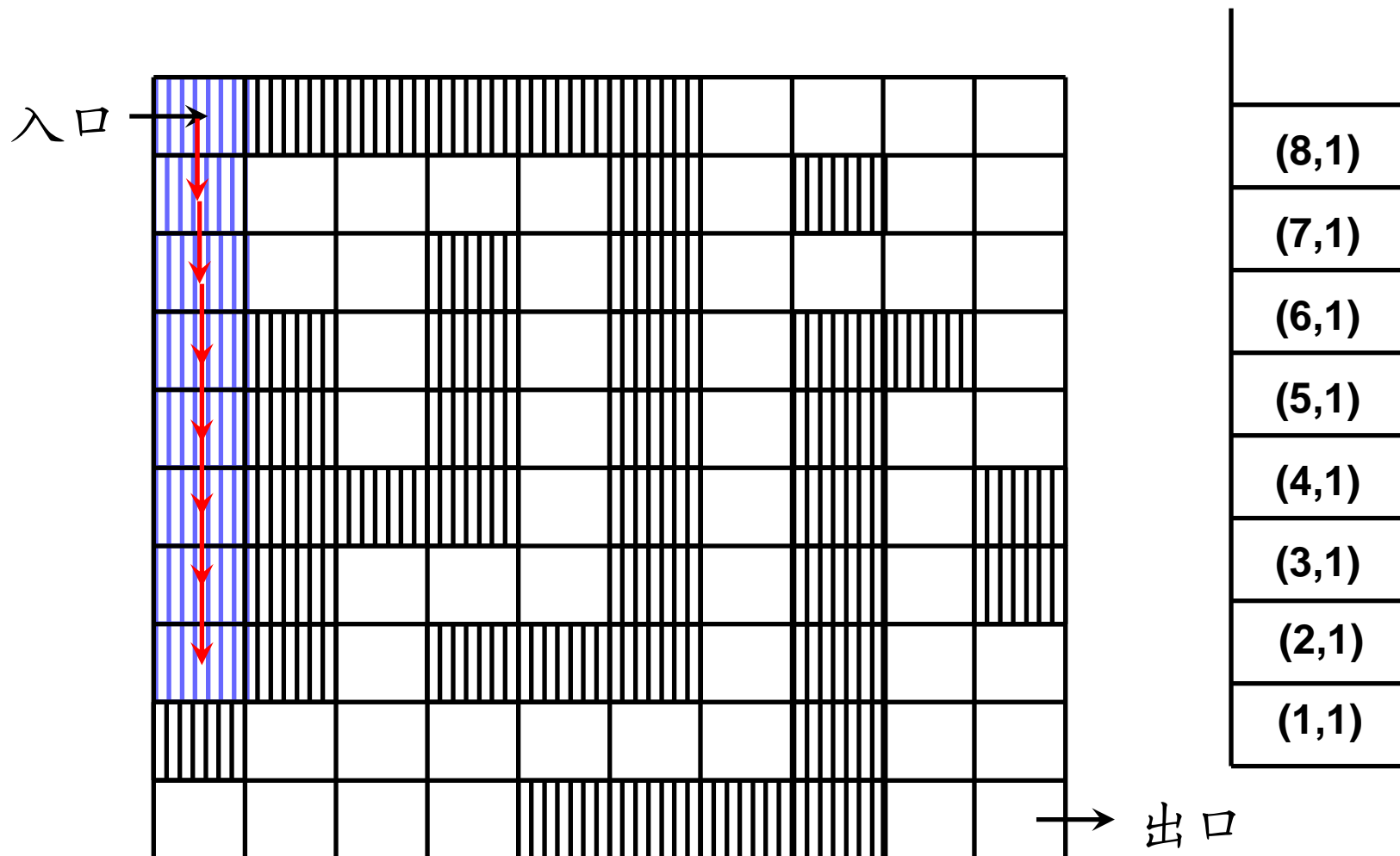
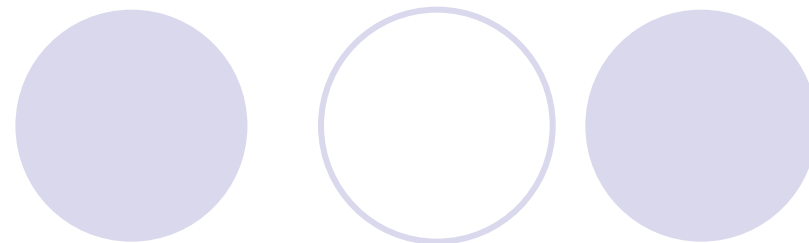
**目标函数:**  $\max x_1 + 3x_2 + 5x_3 + 9x_4$

**约束条件:**  $2x_1 + 3x_2 + 4x_3 + 7x_4 \leq 10$

$x_i \in \{0,1\}$ ,  $i=1,2,3,4$ :



# 迷宫老鼠问题



# 回溯法

- 有许多问题，当需要找出它的解集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。
- 回溯法的基本做法是搜索，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。这种方法适用于解一些组合数相当大的问题。
- 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。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。
- 这种以“深度优先”的方式系统地搜索问题地解地算法称为回溯法。

# 回溯法

- 有许多问题，当需要找出它的解集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。
- 回溯法有“通用的解题法”之称。是一种既带有“系统性”又带有“跳跃性”的搜索算法。

能避免不必要的组合数
- 回溯法在搜索过程中，一旦发现当前结点不能成为所求解的一部分，即跳出发出搜索的结点，而将搜索回溯到该结点的父结点，并继续从该父结点出发搜索。这种搜索过程，既不会“遗漏”可行的，又不会有“确实没必要”的搜索！

吉点，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。
- 这种以“深度优先”的方式系统地搜索问题的解地算法称为回溯法。



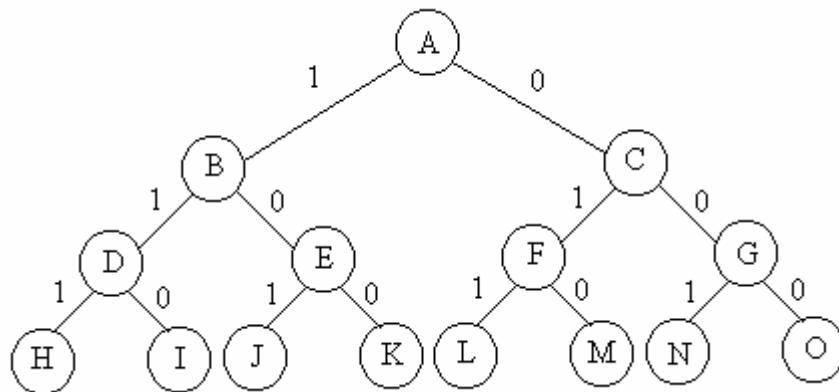
# 回溯法

- 采用回溯法，必须要要弄清如下问题：
  - 1) 问题的解空间树或解空间图是什么？是子集树、或排列树，还是 $m$ 叉树，或更复杂的树或图？
  - 2) 搜索的方向是：深度优先
  - 3) 用剪枝函数来避免不可能产生最优解的分支的搜索，越好的剪枝和限界函数可以砍去更多的无用分支搜索。

# 问题的解空间

- 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 $(x_1, x_2, \dots, x_n)$ 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。

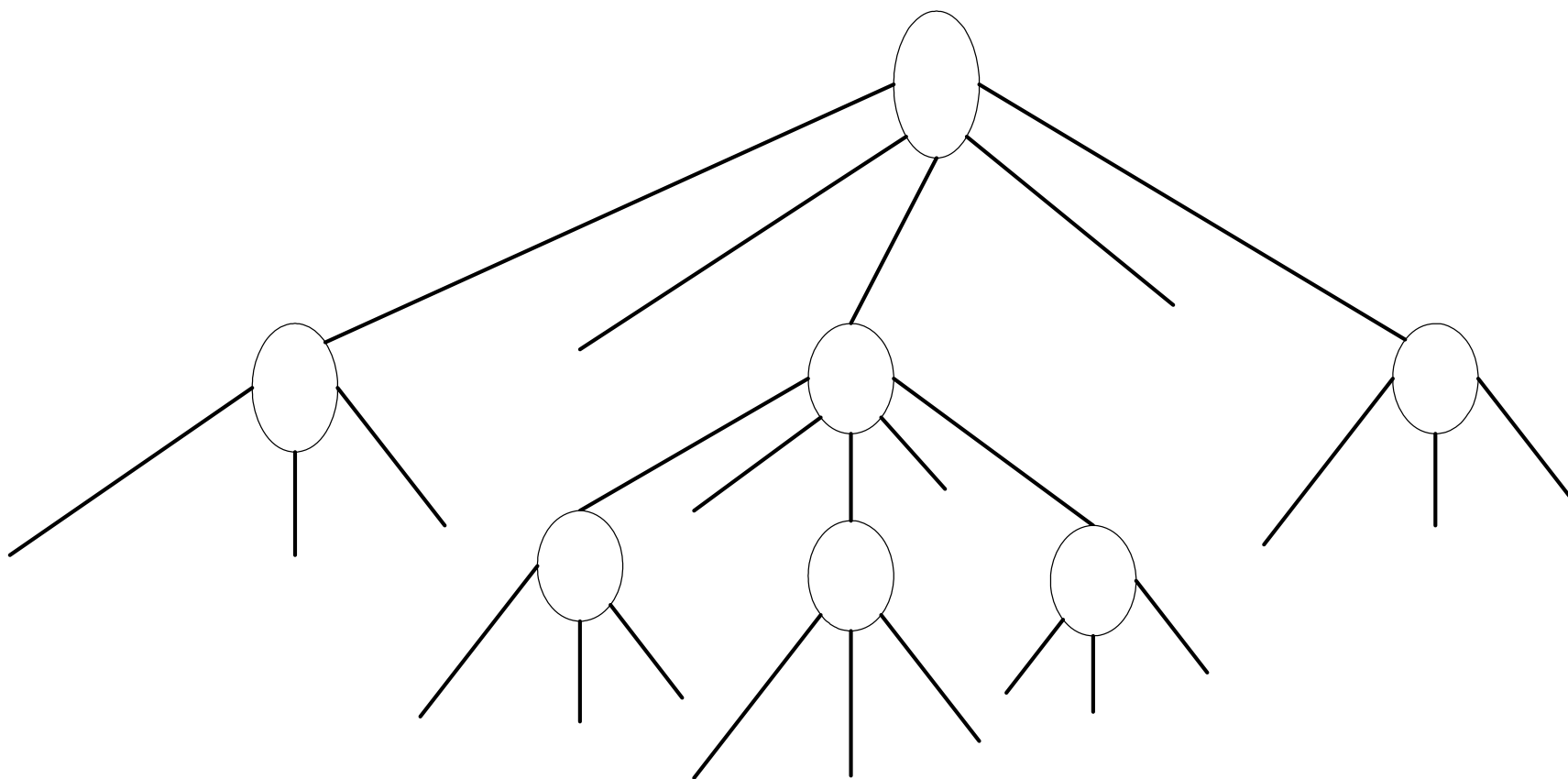
注意：有时同一个问题可以有多种解空间表示，有些表示更简单，所需表示的状态空间也更小。（存储量少，搜索方法简单）



$n=3$ 时的0-1背包问题用完全二叉树表示的解空间

# 解空间树的表现形式

常用的是解空间树，它表现为如下形式：



# 生成问题状态的基本方法

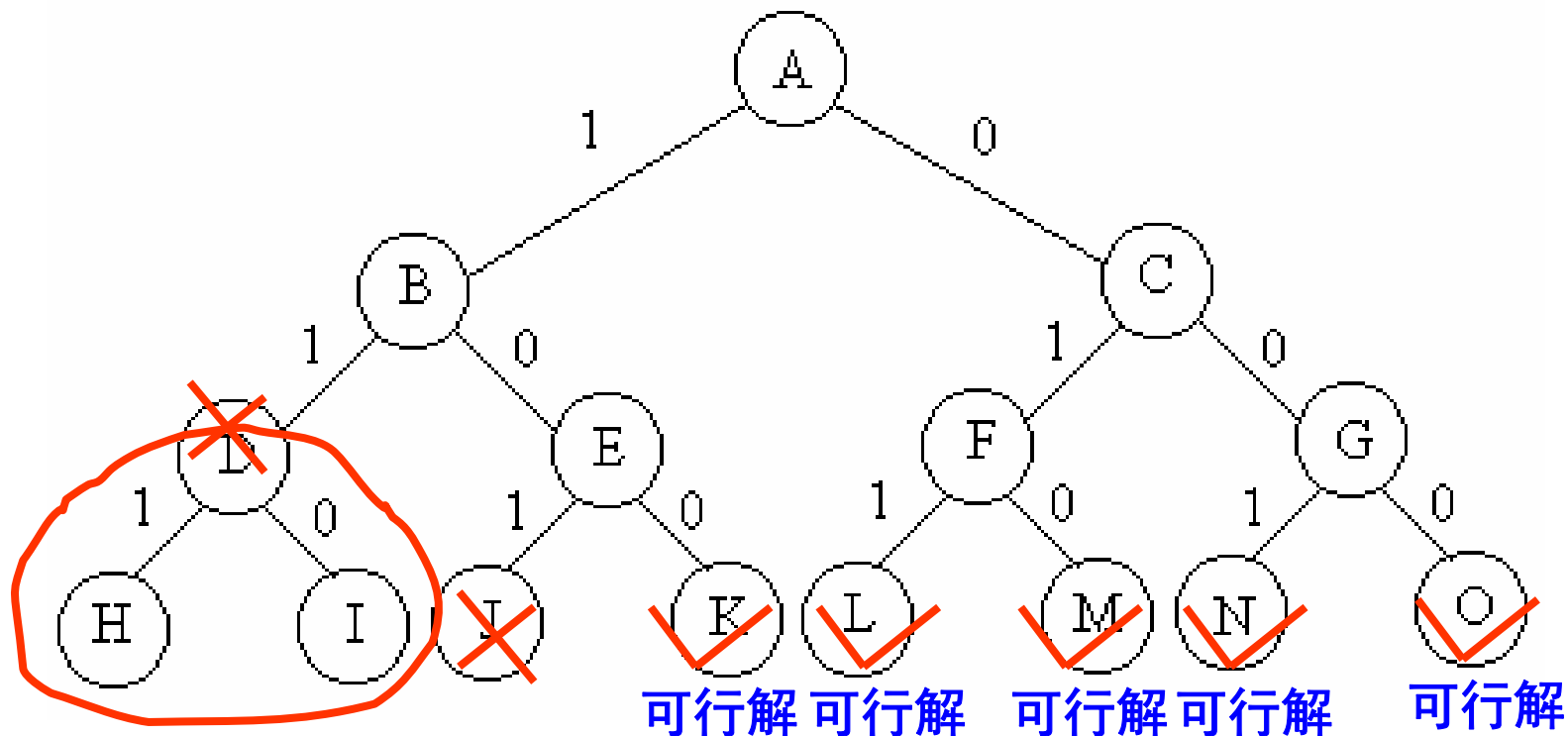
- 扩展结点：一个正在产生儿子的结点称为扩展结点
- 活结点：一个自身已生成但其儿子还没有全部生成的节点称做活结点
- 死结点：一个所有儿子已经产生的结点称做死结点
- 深度优先的问题状态生成法：如果对一个扩展结点R，一旦产生了它的一个儿子C，就把C当做新的扩展结点。在完成对子树C（以C为根的子树）的穷尽搜索之后，将R重新变成扩展结点，继续生成R的下一个儿子（如果存在）
- 广度优先的问题状态生成法：在一个扩展结点变成死结点之前，它一直是扩展结点
- 回溯法：为了避免生成那些不可能产生最佳解的问题状态，要不断地利用剪枝函数来处死那些实际上不可能产生所需最优解的活结点，以减少问题的计算量。具有剪枝函数的深度优先生成法称为回溯法

# 回溯法的基本思想

## 0-1背包问题

$n=3$ 时，0-1背包问题解空间树

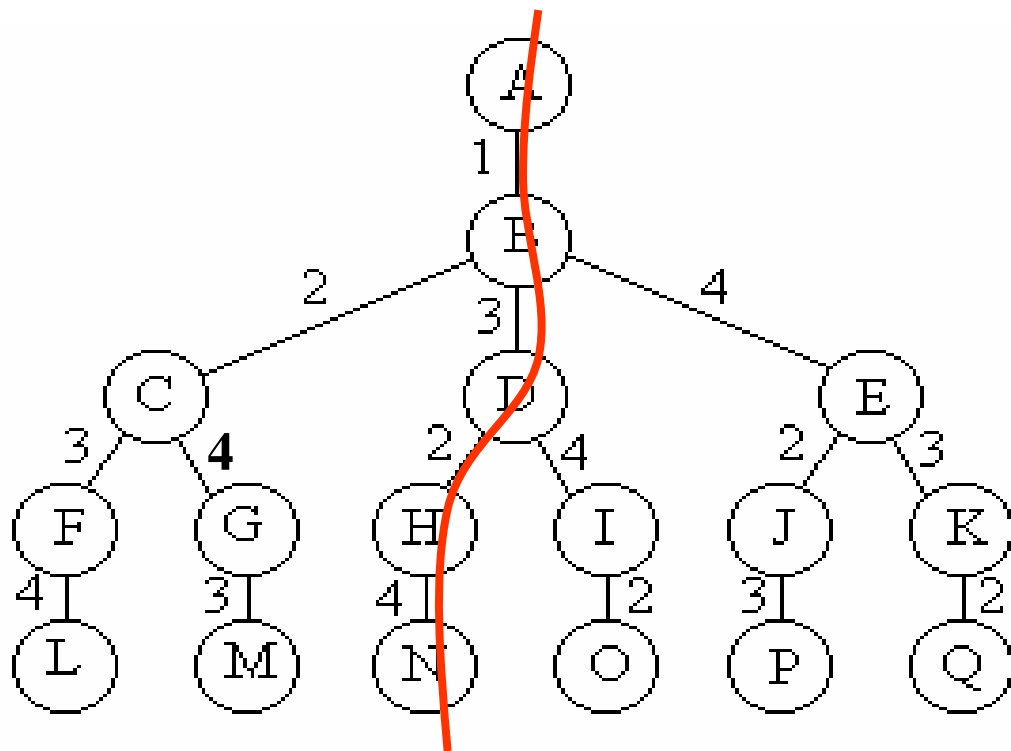
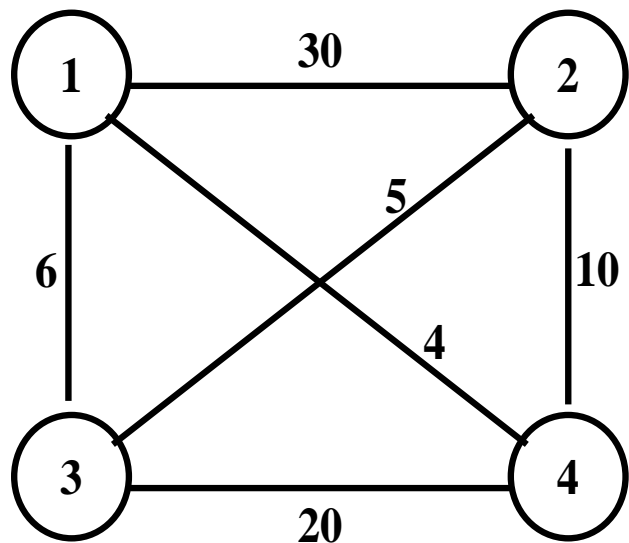
搜索空间：子集树， $2^n$ 片树叶



$w=[16,15,15]$ ,  $p=[45,25,25]$ , 背包容量 $C=30$

# 回溯法的基本思想

旅行售货员问题 搜索空间：排列树



# 回溯法的基本思想

- (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，并构造结果判断函数；
- (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。

**空间上：** 用回溯法解题时，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 $O(h(n))$ 。而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。

# 回溯法的基本思想

回溯算法框架＝

问题的解空间＋深度优先遍历＋判断结果的函数＋剪枝函数

这个剪枝函数：

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。

“约束函数”和“限界函数”含义不同，在后面装载问题和0-1背包问题的搜索中我们都可以看到这两类剪枝函数不同之处。



# 递归回溯

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。

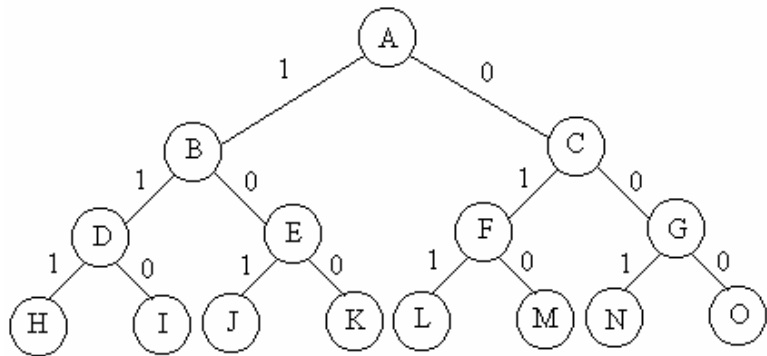
```
void backtrack (int t)
{
    if (t>n) output(x);
    else
        for (int i=f(n,t);i<=g(n,t);i++) {
            x[t]=h(i);
            if (constraint(t)&&bound(t)) backtrack(t+1);
        }
}
```

# 迭代回溯

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。

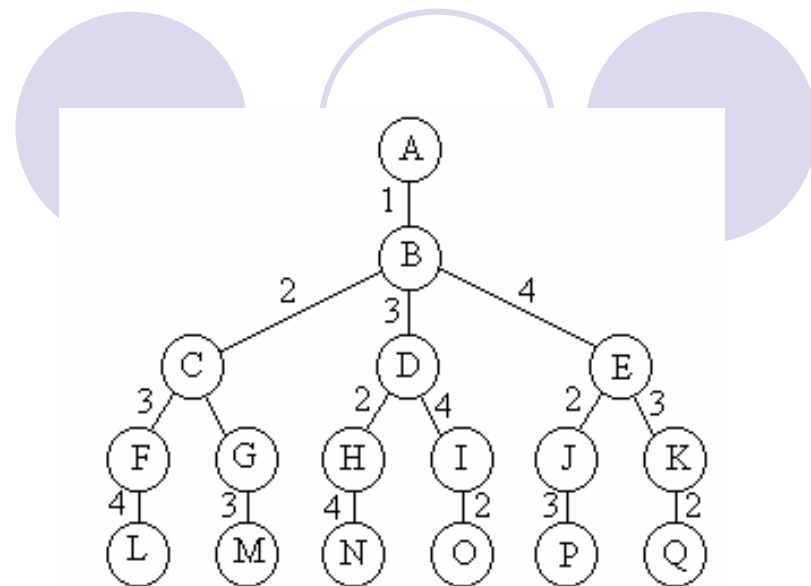
```
void iterativeBacktrack ()
{
    int t=1;
    while (t>0) {
        if (f(n,t)<=g(n,t))
            for (int i=f(n,t);i<=g(n,t);i++) {
                x[t]=h(i);
                if (constraint(t)&&bound(t)) {
                    if (solution(t)) output(x);
                    else t++;
         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

# 子集树与排列树



遍历子集树需 $O(2^n)$ 计算时间

```
void backtrack (int t)
{
    if (t>n) output(x);
    else
        for (int i=0;i<=1;i++) {
            x[t]=i;
            if (legal(t)) backtrack(t+1);
        }
}
```



遍历排列树需要 $O(n!)$ 计算时间

```
void backtrack (int t)
{
    if (t>n) output(x);
    else
        for (int i=t;i<=n;i++) {
            swap(x[t], x[i]);
            if (legal(t)) backtrack(t+1);
            swap(x[t], x[i]);
        }
}
```