# 计算理论与算法分析设计

林永钢

教材:

[王] 王晓东,计算机算法设计与分析(第4版),电子工业.

参考资料:

[C] 潘金贵等译, Cormen等著, 算法导论, 机械工业.

[M] 黄林鹏等译, Manber著, 算法引论-一种创造性方法, 电子.

[刘] 刘汝佳等, 算法艺术与信息学竞赛, 清华大学.

# 第5章 回溯法

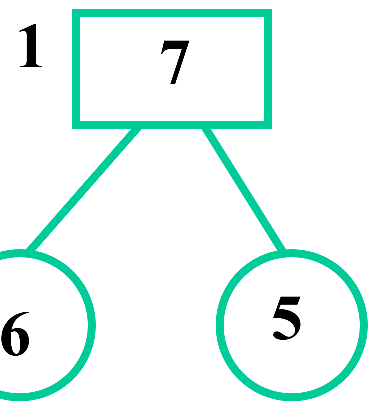
1. 装载问题与01背包
2. 回溯算法设计步骤
3. 旅行售货员问题(TSP)
4. n皇后问题
5. 最大团问题
6. 符号三角形
7. 回溯算法的效率

# 搜索算法

- 穷举搜索(brute-force Search)
- 图遍历(Graph traversal)
  - 深度优先搜索(DFS)
  - 广度优先搜索(BFS)
- 树遍历(Tree traversal)
  - 回溯(Backtracking)
  - 分枝限界法(Branch and Bound);
  - 博弈树搜索( α - β Search)等
- 启发式搜索(Heuristic Search)

# 博弈树搜索( α - β Search)

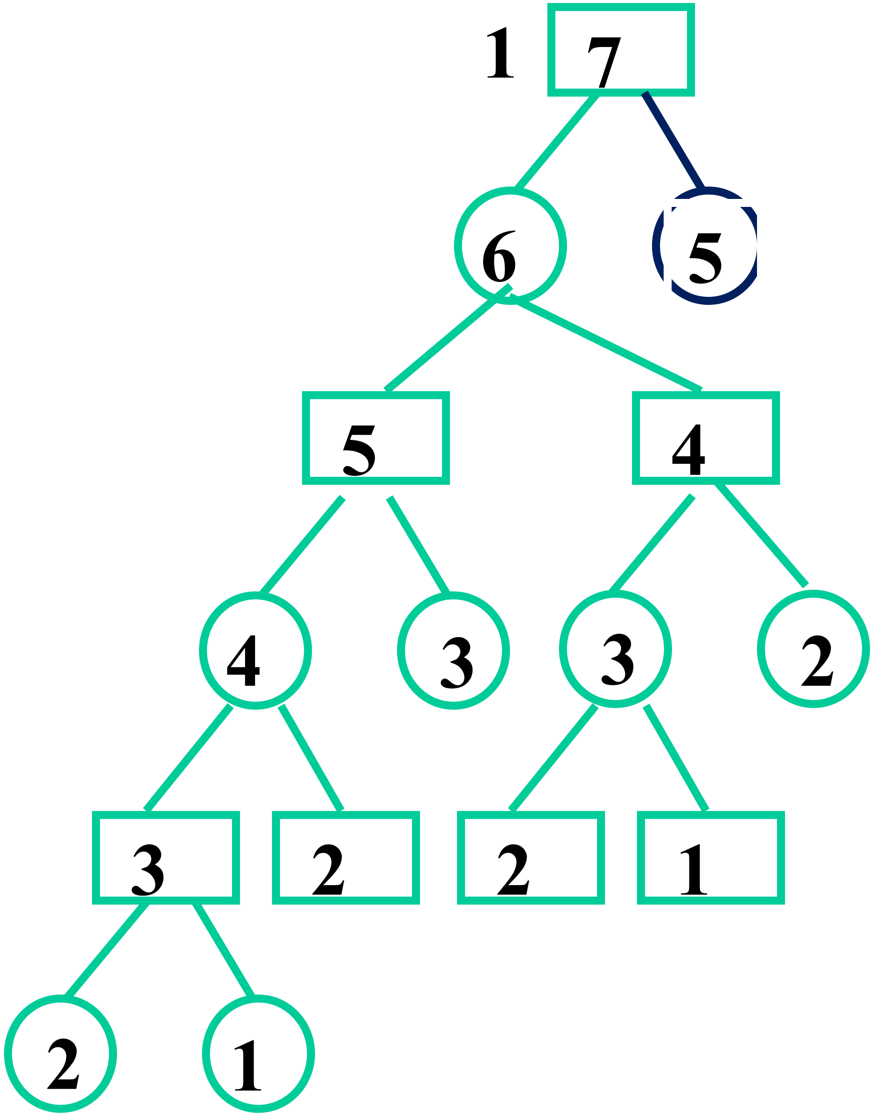
有7根火柴，A,B两人依次从中取出1根或2根，不能不取也不能取多于2根，最后一个将火柴取尽的就是胜利者。用符号 $\boxed{m_1}$ 表示轮到A时有$m_1$根火柴的状态,用 $m_2$ 表示轮到B时有$m_2$根火柴的状态，双方对弈假定从A开始。用1表示A取胜，用-1表示B取胜。



有7根火柴，A,B两人依次取出1根或2根，最后一个将火柴取尽是胜利者。用$\boxed{m_1}$表示轮到A时有$m_1$根火柴的状态,用$m_2$表示轮到B时有$m_2$根火柴的状态，双方对弈从A开始。

1表示A取胜，

-1表示B取胜。



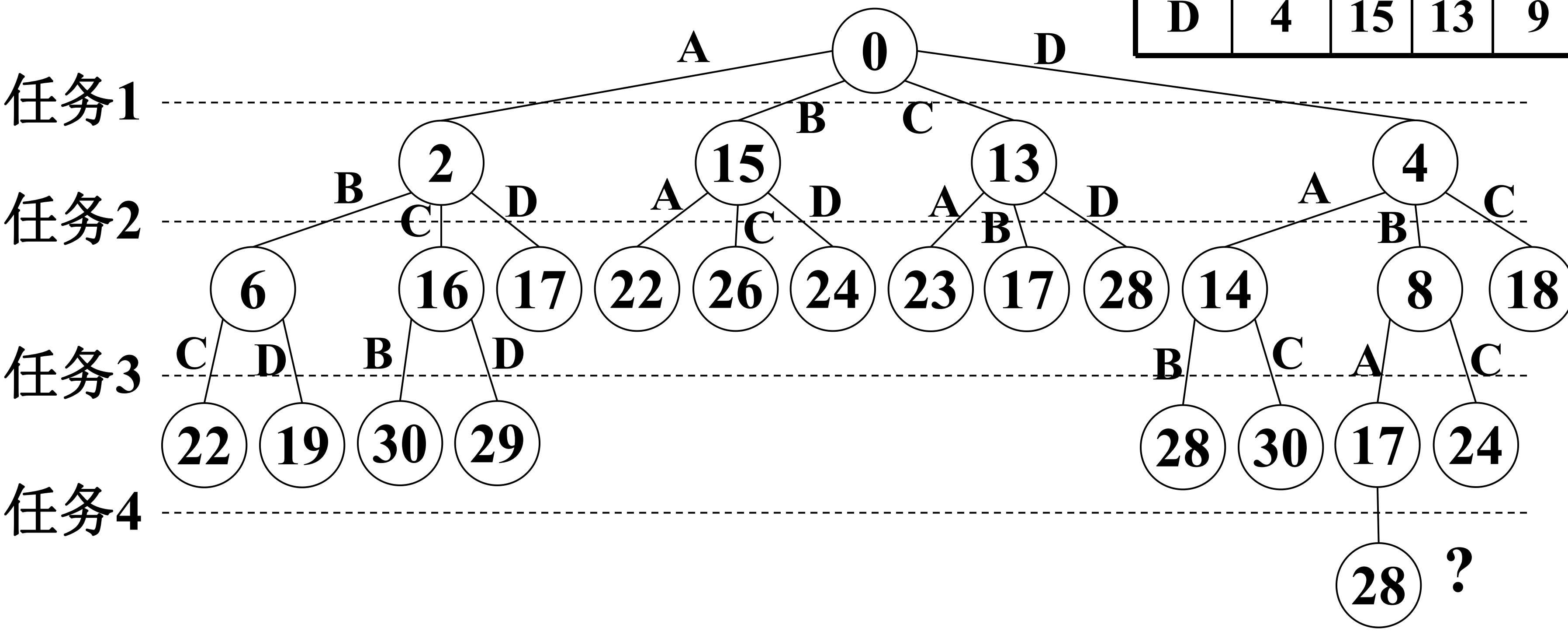
# 分支限界-观察: 任务分配问题

右表是不同人完成不同任务所需时间

找出总时间最少的分配方案

| 任务 / 人 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| A | 2 | 10 | 9 | 7 |
| B | 15 | 4 | 14 | 8 |
| C | 13 | 14 | 16 | 11 |
| D | 4 | 15 | 13 | 9 |

任务1:4最少时间: 2, 4, 9, 7



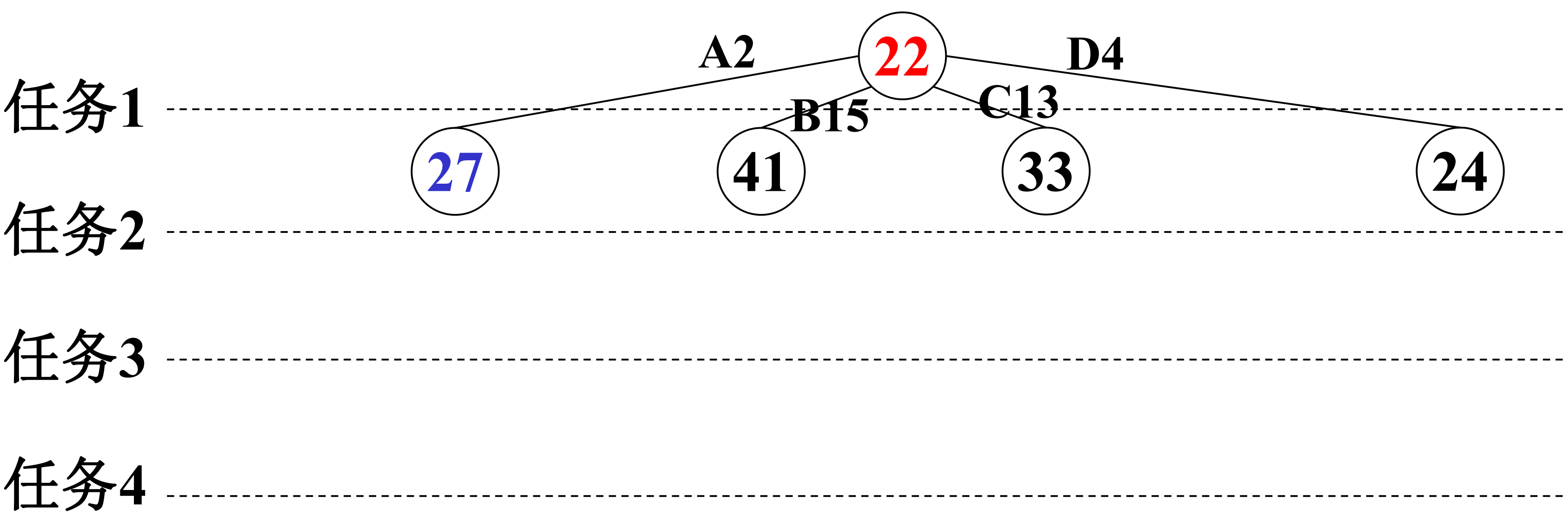
# 分支限界-任务分配: 时间下界

以(当前时间+剩余最少时间)为关键值扩展新的节点

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| A | 2 | 10 | 9 | 7 |
| B | 15 | 4 | 14 | 8 |
| C | 13 | 14 | 16 | 11 |
| D | 4 | 15 | 13 | 9 |

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| A | 2 | 10 | 9 | 7 |
| B | 15 | 4 | 14 | 8 |
| C | 13 | 14 | 16 | 11 |
| D | 4 | 15 | 13 | 9 |

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| A | 2 | 10 | 9 | 7 |
| B | 15 | 4 | 14 | 8 |
| C | 13 | 14 | 16 | 11 |
| D | 4 | 15 | 13 | 9 |



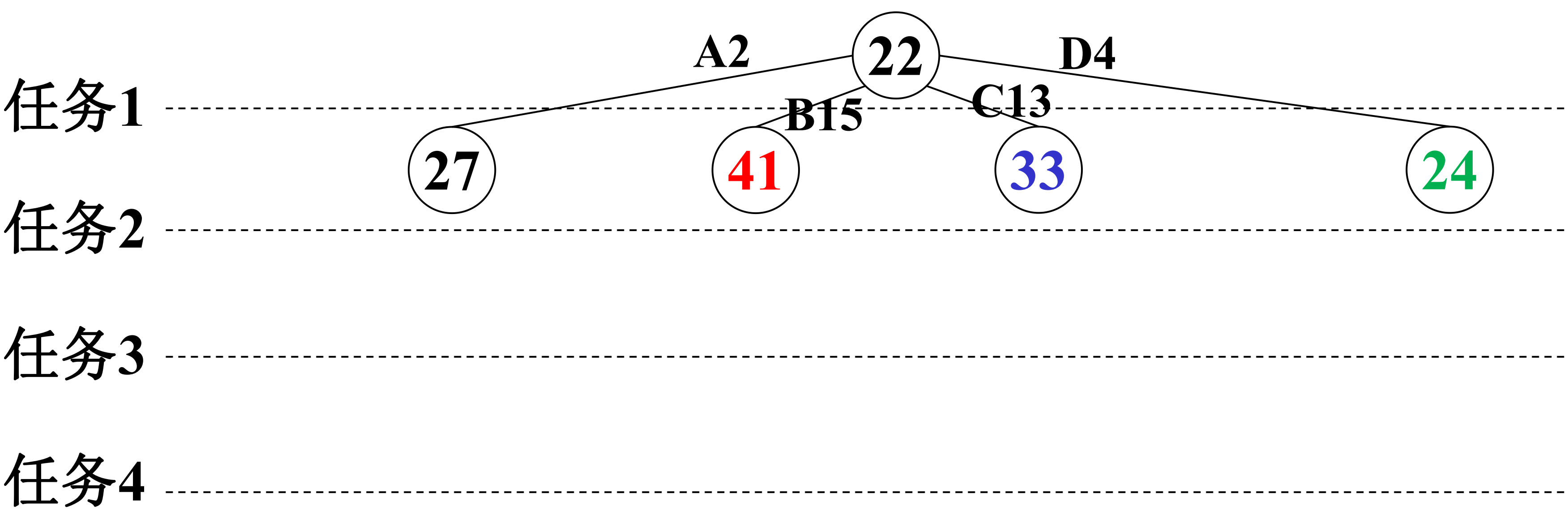
# 分支限界-任务分配: 时间下界

以(当前时间+剩余最少时间)为关键值扩展新的节点

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| A | 2 | 10 | 9 | 7 |
| B | 15 | 4 | 14 | 8 |
| C | 13 | 14 | 16 | 11 |
| D | 4 | 15 | 13 | 9 |

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| A | 2 | 10 | 9 | 7 |
| B | 15 | 4 | 14 | 8 |
| C | 13 | 14 | 16 | 11 |
| D | 4 | 15 | 13 | 9 |

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| A | 2 | 10 | 9 | 7 |
| B | 15 | 4 | 14 | 8 |
| C | 13 | 14 | 16 | 11 |
| D | 4 | 15 | 13 | 9 |



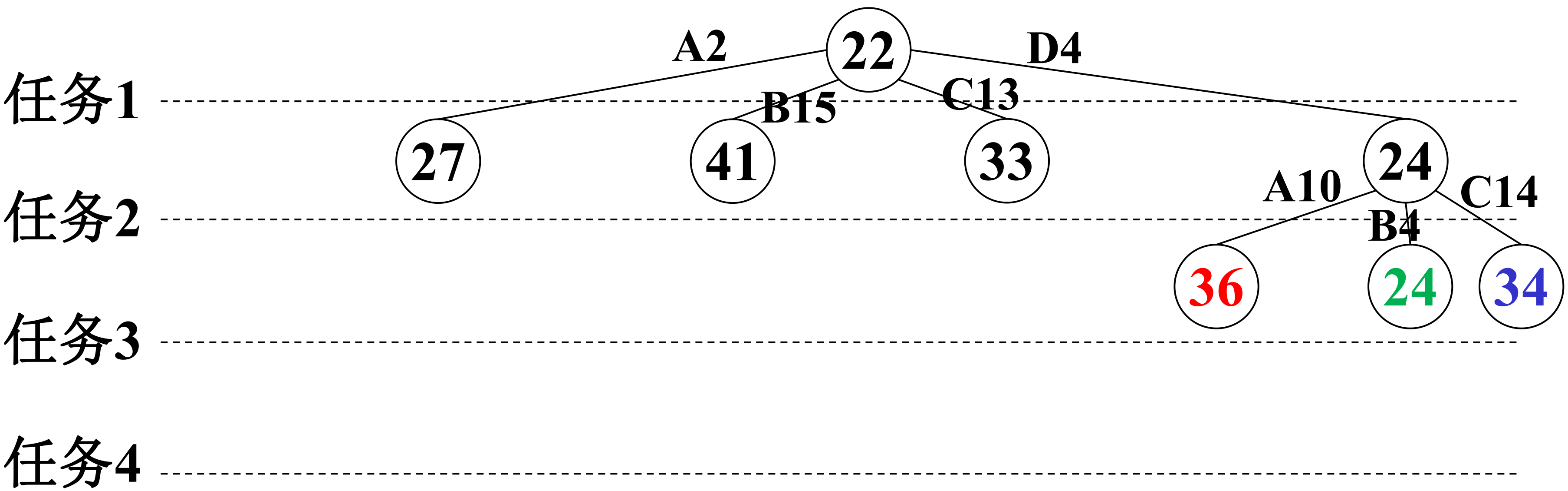
# 分支限界-任务分配: 时间下界

以(当前时间+剩余最少时间)为关键值扩展新的节点

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| A | 2 | 10 | 9 | 7 |
| B | 15 | 4 | 14 | 8 |
| C | 13 | 14 | 16 | 11 |
| D | 4 | 15 | 13 | 9 |

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| A | 2 | 10 | 9 | 7 |
| B | 15 | 4 | 14 | 8 |
| C | 13 | 14 | 16 | 11 |
| D | 4 | 15 | 13 | 9 |

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| A | 2 | 10 | 9 | 7 |
| B | 15 | 4 | 14 | 8 |
| C | 13 | 14 | 16 | 11 |
| D | 4 | 15 | 13 | 9 |

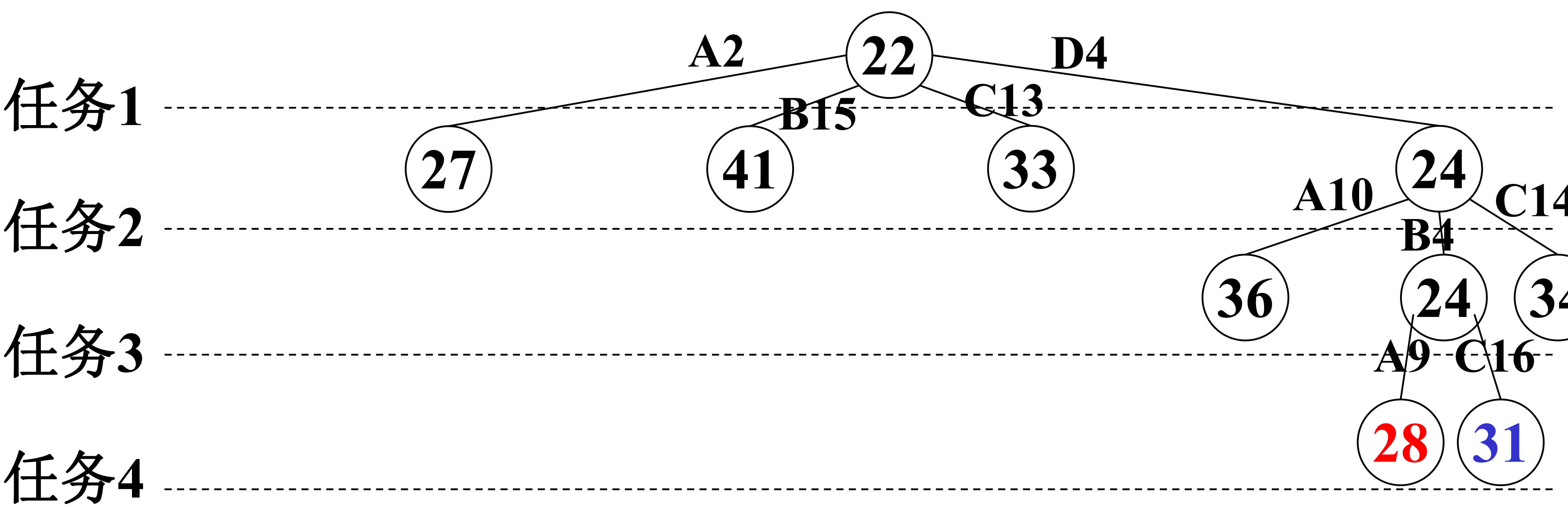


# 分支限界-任务分配: 时间下界

以(当前时间+剩余最少时间)为关键值扩展新的节点

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| A | 2 | 10 | 9 | 7 |
| B | 15 | 4 | 14 | 8 |
| C | 13 | 14 | 16 | 11 |
| D | 4 | 15 | 13 | 9 |

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| A | 2 | 10 | 9 | 7 |
| B | 15 | 4 | 14 | 8 |
| C | 13 | 14 | 16 | 11 |
| D | 4 | 15 | 13 | 9 |



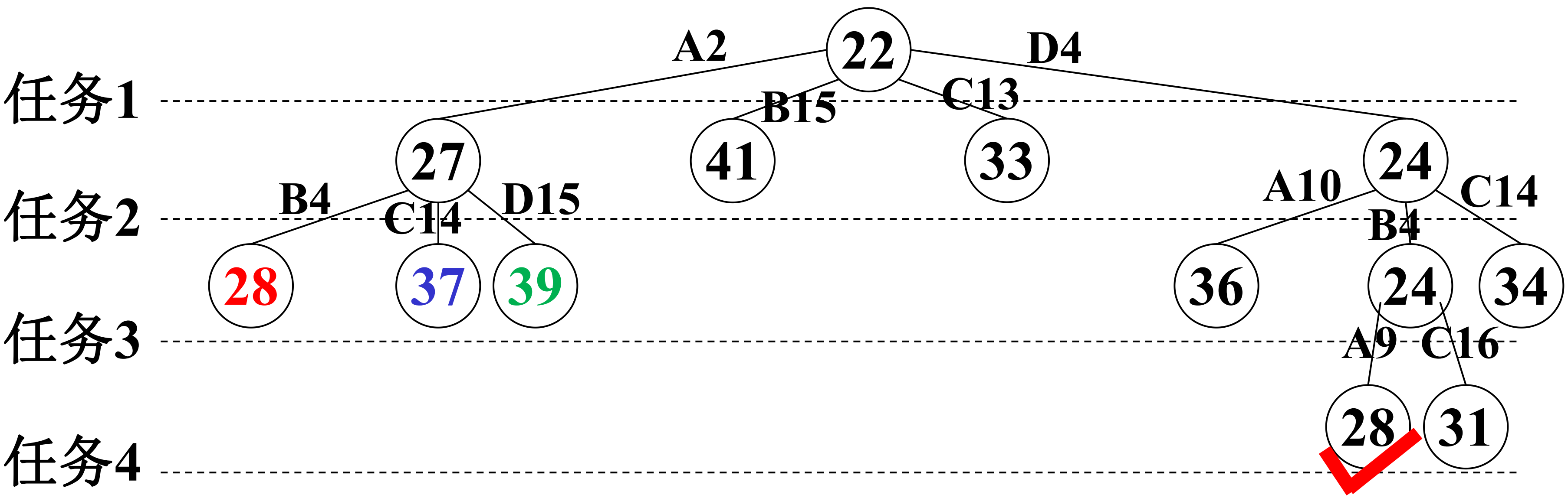
# 分支限界-任务分配: 时间下界

以(当前时间+剩余最少时间)为关键值扩展新的节点

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| A | 2 | 10 | 9 | 7 |
| B | 15 | 4 | 14 | 8 |
| C | 13 | 14 | 16 | 11 |
| D | 4 | 15 | 13 | 9 |

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| A | 2 | 10 | 9 | 7 |
| B | 15 | 4 | 14 | 8 |
| C | 13 | 14 | 16 | 11 |
| D | 4 | 15 | 13 | 9 |

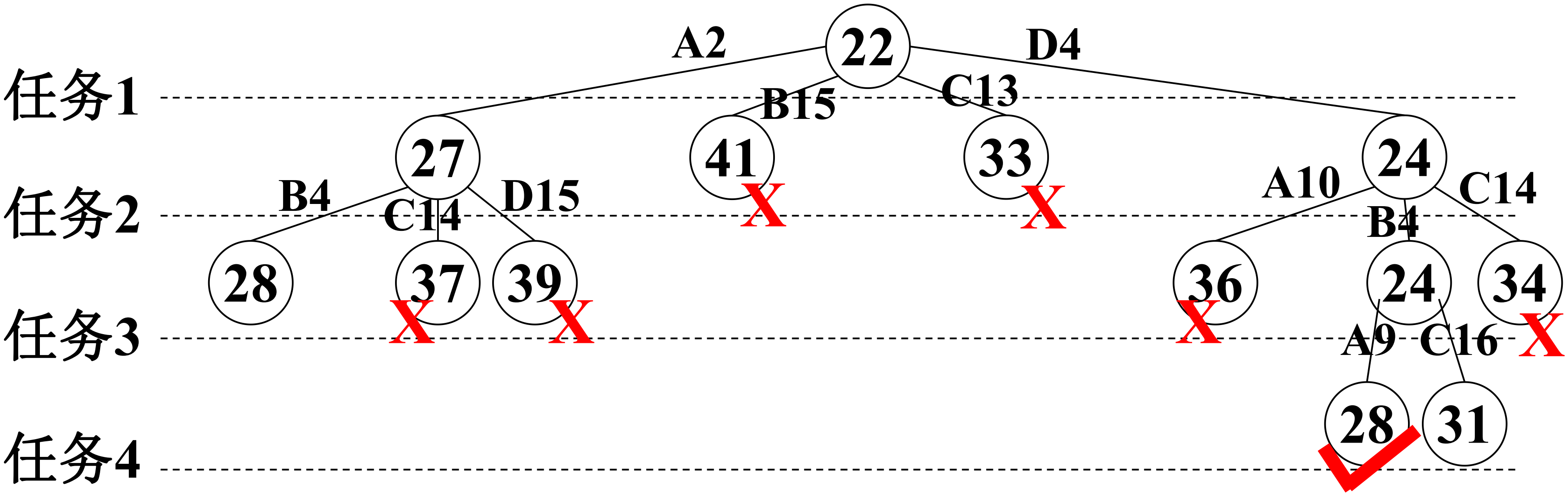
| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| A | 2 | 10 | 9 | 7 |
| B | 15 | 4 | 14 | 8 |
| C | 13 | 14 | 16 | 11 |
| D | 4 | 15 | 13 | 9 |



# 分支限界-任务分配: 时间下界

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| A | 2 | 10 | 9 | 7 |
| B | 15 | 4 | 14 | 8 |
| C | 13 | 14 | 16 | 11 |
| D | 4 | 15 | 13 | 9 |

下界: 当前时间+剩余最少时间



# 搜索空间的三种表示

- 穷举搜索(brute-force Search)
- 表序表示：搜索对象用线性表数据结构表示；
- 显式图表示：搜索对象在搜索前就用图(树)的
  数据结构表示；
- 隐式图表示：除了初始结点，其他结点在搜索
  过程中动态生成．缘于搜索空间大，难以全部存储．
- 灌水问题：7升和3升，量出5升.

# 装载问题

- **n**件货物(重**w[1:n]**)装两艘船(载重量$c_1, c_2$), $\Sigma_{i=1}^{n}$ w[i] ≤ $c_1+c_2$, 是否有装载方案.
- 讨论过类似问题: **0-1**背包, 分数背包, 最优装载
- 装载方案: 尽可能装满第**1**艘, 剩余的装第**2**艘
- 尽可能装满第**1**艘等价于下面变形的**0-1**背包

$$\max \sum_{i=1}^{n} w[i]x[i]$$

$$\text{s.t.} \sum_{i=1}^{n} w[i]x[i] \le c$$

$$x[i] \in \{0,1\}, 1 \le i \le n$$

本章以此为装载问题
讨论回溯算法
样例: w=[16,15,15], c=30

# 简单回溯: 装载问题w[1:n],c

- 在树上进行深度优先搜索, 通常**同层结构相同**.

**backtrack(t)**

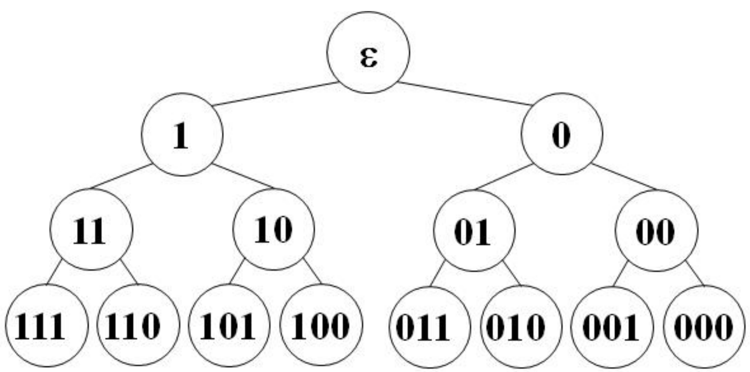
**// t:层号, cw:当前重量, bestw:最优重量**

**1. 若t>n, (若cw≤c且cw>bestw, bestw=cw.) 返回 ---//记录更新**

**2. cw+=w[t], backtrack(t+1), cw-=w[t],------------//左分支**

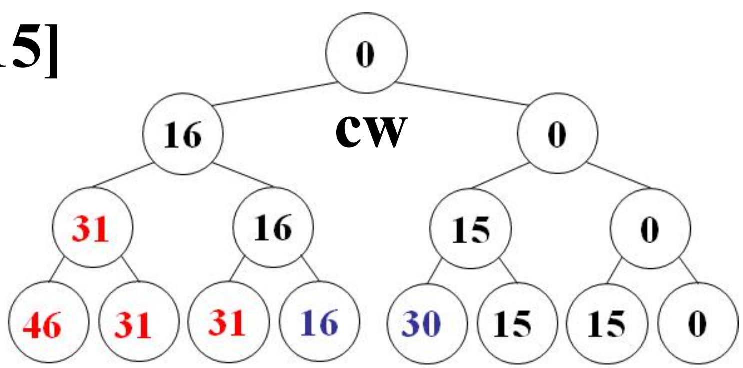
**3. backtrack(t+1)---------------------------------------//右分支**

- 初始: **bestw=cw=0,** 执行 **backtrack(1),** 简单, 优美

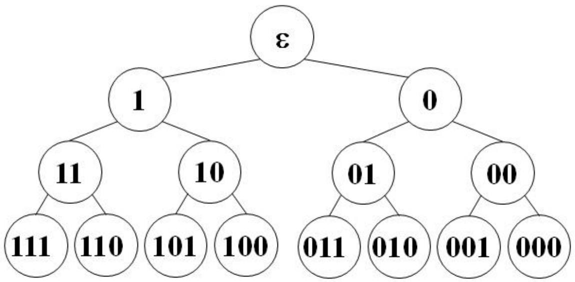



**w=[16,15,15]**

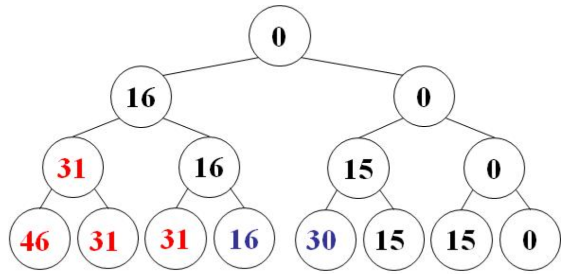
**c=30**



# w=[16,15,15], c=30, backtrack(1)




程序隐含的解空间结构




各节点的cw值

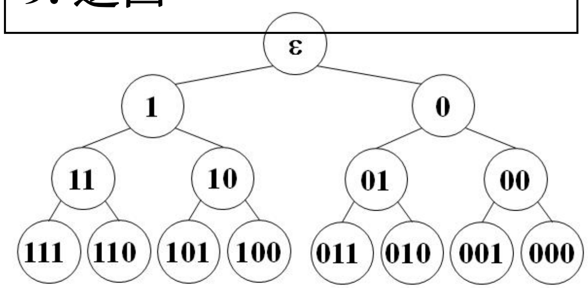
k-b-(t,bestw,cw)
//进入第k行前的数据.
//b: 实际对应节点标号

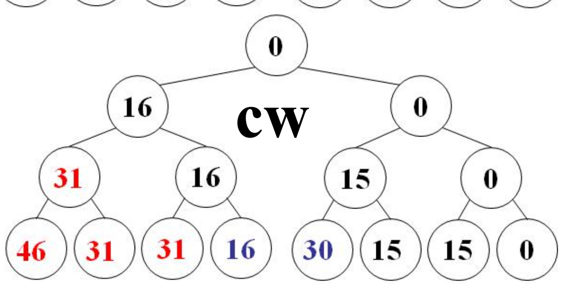
```
backtrack(t)
1. 若t>n,
2. |  若cw≤c且cw>bestw,
3. |  |  bestw=cw
4. |  返回
5. cw+=w[t]
6. backtrack(t+1)//进左分支
7. cw-=w[t]
8. backtrack(t+1)//进右分支
9. 返回
```

| | |
|---|---|
| 1-ε-(1,0,0) | 1-111-(4,0,46) |
| 5-ε-(1,0,0) | 2-111-(4,0,46) |
| 6-ε-(1,0,16) | 4-111-(4,0,46) |
| 1-1-(2,0,16) | 7-11-(3,0,46) |
| 5-1-(2,0,16) | 8-11-(3,0,31) |
| 6-1-(2,0,31) | 1-110-(4,0,31) |
| 1-11-(3,0,31) | 2-110-(4,0,31) |
| 5-11-(3,0,31) | 4-110-(4,0,31) |
| 6-11-(3,0,46) | 9-11-(3,0,31) |

# w=[16,15,15], c=30, backtrack(1)

```
backtrack(t)
1. 若t>n,
2. |  若cw≤c且cw>bestw,
3. |  |  bestw=cw
4. |  返回
5. cw+=w[t]
6. backtrack(t+1)//左分支
7. cw-=w[t]
8. backtrack(t+1)//右分支
9. 返回
```




k-b-(t,bestw,cw) //进入第k步前的数据. 剪枝?

| | | | |
|---|---|---|---|
| 1-ε-(1,0,0) | 9-11-(3,0,31) | 7-ε-(1,16,0) | 9-01-(3,30,15) |
| 5-ε-(1,0,0) | 7-1-(2,0,31) | 8-ε-(1,16,0) | 7-0-(2,30,15) |
| 6-ε-(1,0,16) | 8-1-(2,0,16) | 1-0-(2,16,0) | 8-0-(2,30,0) |
| 1-1-(2,0,16) | 1-10-(3,0,16) | 5-0-(2,16,0) | 1-00-(3,30,0) |
| 5-1-(2,0,16) | 5-10-(3,0,16) | 6-0-(2,16,15) | 5-00-(3,30,0) |
| 6-1-(2,0,31) | 6-10-(3,0,31) | 1-01-(3,16,15) | 6-00-(3,30,15) |
| 1-11-(3,0,31) | 1-101-(4,0,31) | 5-01-(3,16,15) | 1-001-(4,30,15) |
| 5-11-(3,0,31) | 2-101-(4,0,31) | 6-01-(3,16,30) | 2-001-(4,30,15) |
| 6-11-(3,0,46) | 4-101-(4,0,31) | 1-011-(4,16,30) | 4-001-(4,30,15) |
| 1-111-(4,0,46) | 7-10-(3,0,31) | 2-011-(4,16,30) | 7-00-(3,30,15) |
| 2-111-(4,0,46) | 8-10-(3,0,16) | 3-011-(4,16,30) | 8-00-(3,30,0) |
| 4-111-(4,0,46) | 1-100-(4,0,16) | 4-011-(4,30,30) | 1-000-(4,30,0) |
| 7-11-(3,0,46) | 2-100-(4,0,16) | 7-01-(3,30,30) | 2-000-(4,30,0) |
| 8-11-(3,0,31) | 3-100-(4,0,16) | 8-01-(3,30,15) | 4-000-(4,30,0) |
| 1-110-(4,0,31) | 4-100-(4,16,16) | 1-010-(4,30,15) | 9-00-(3,30,0) |
| 2-110-(4,0,31) | 9-10-(3,16,16) | 2-010-(4,30,15) | 9-0-(2,30,0) |
| 4-110-(4,0,31) | 9-1-(2,16,16) | 4-010-(4,30,15) | 9-ε-(1,30,0) |

# 约束条件: 装载问题w[1:n],c

backtrack(t)

// t:层号, cw:当前重量, bestw:最优重量

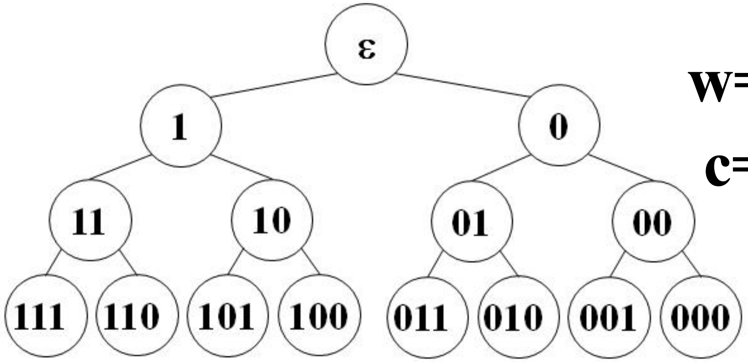
1. 若t>n, (若cw>bestw,bestw=cw.) 返回

2. 若 cw + w[t] ≤ c, 则 --------------------//约束条件(剪枝)

3. | cw+=w[t], backtrack(t+1), cw-=w[t],------//左分支

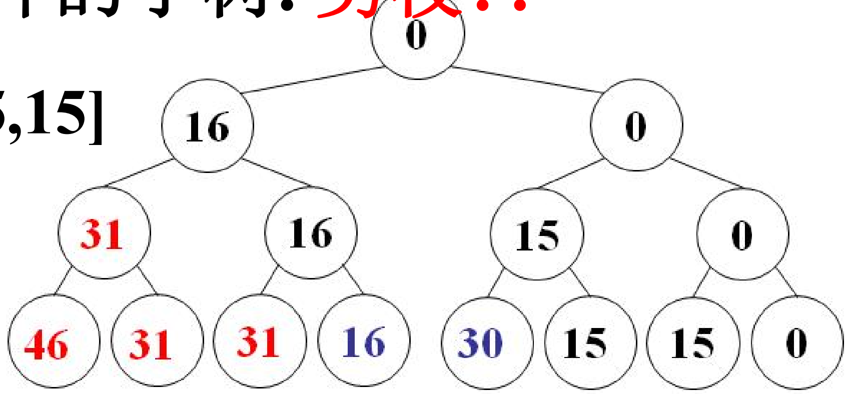
4. backtrack(t+1)-----------------------------------//右分支

约束条件: 剪去不满足约束条件的子树. 剪枝??




w=[16,15,15]

c=30

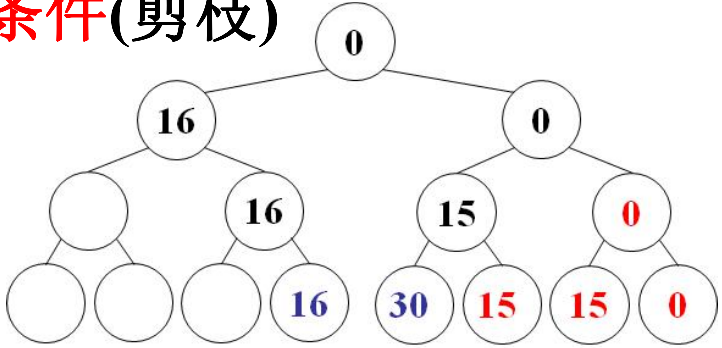


# 限界条件: 装载问题w[1:n],c

- 初始: bestw=cw=0. r=$\text{sum}_{t=1}^{n}$ w[t] //当前剩余重量

backtrack(t) //t层号, cw, bestw

1. 若t>n, (若cw>bestw, bestw=cw, bestx=x.) 返回
2. r-=w[t]
3. 若 cw + w[t] ≤ c, 则----------------//约束条件(剪枝)
4. | cw+=w[t], x[t]=1, backtrack(t+1), cw-=w[t], //左分支
5. 若 cw + r > bestw, 则--------------//限界条件(剪枝)
6. | x[t]=0, backtrack(t+1) //右分支
7. r+=w[t] //还原




限界条件: 剪去得不到最优解子树

约束条件与限界条件统称为剪枝函数. 执行哪些节点?

# 提前更新最优值

backtrack(t)

1. 若t>n, 则返回

2. r-=w[t]

3. 若 cw+w[t] ≤ c, 则

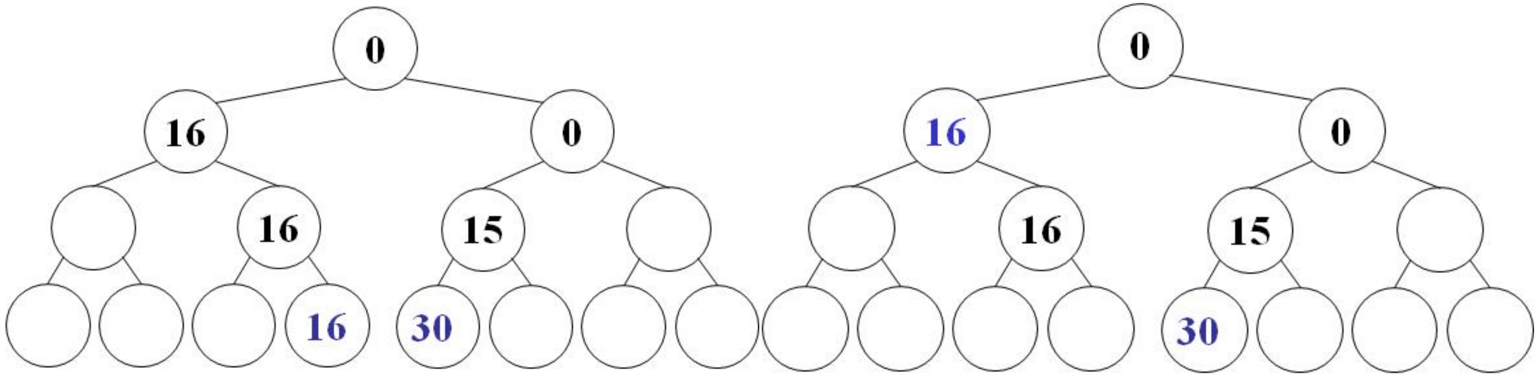
4. |　若cw+w[t]>bestw, 则bestw=cw+w[t]

5. |　cw+=w[t], backtrack(t+1), cw-=w[t]  -----//x[t]=1

6. 若 cw + r > bestw, 则

7. |　backtrack(t+1) --------------------------------//x[t]=0

8.  r+=w[t]



# 递归回溯(backtracking)

backtrack(t)

1. 若 t>n, 则判断 记录 返回 //记录更新

2. 对 i = f(n,t) : g(n,t) //第t层扩展节点的起止编号

3. | x[t]=h(i), //第t层的第i个可选值

4. | 若C(t)且B(t), 则backtrack(t+1) //剪枝

- C(t): 约束函数 B(t): 限界函数
- 执行backtrack(1)完成搜索
- 判断记录也可能提前.
- 可以迭代回溯.

```
backtrack(t)
1. 若 t>n, 则判断 记录 返回
2. 对 i = 1 : 0
3. |  x[t]=i, cw+=x[t]*w[t]
4. |  若C(t)且B(t), 则backtrack(t+1)
5. |   cw-=x[t]*w[t]
```

# 0-1背包回溯 $O(2^n)$

- 输入: n物品重w[1:n], 价值v[1:n], 背包容量C
- 输出: 装包使得价值最大.
- DP: 物品重量为整数, O(nC) // 输入规模max{n, $\log_2 C$}

初始: bestv=cv=cw=0, r=$\text{sum}_{t=1}^{n}$ v[t], 执行backtrack(1),

backtrack(t)

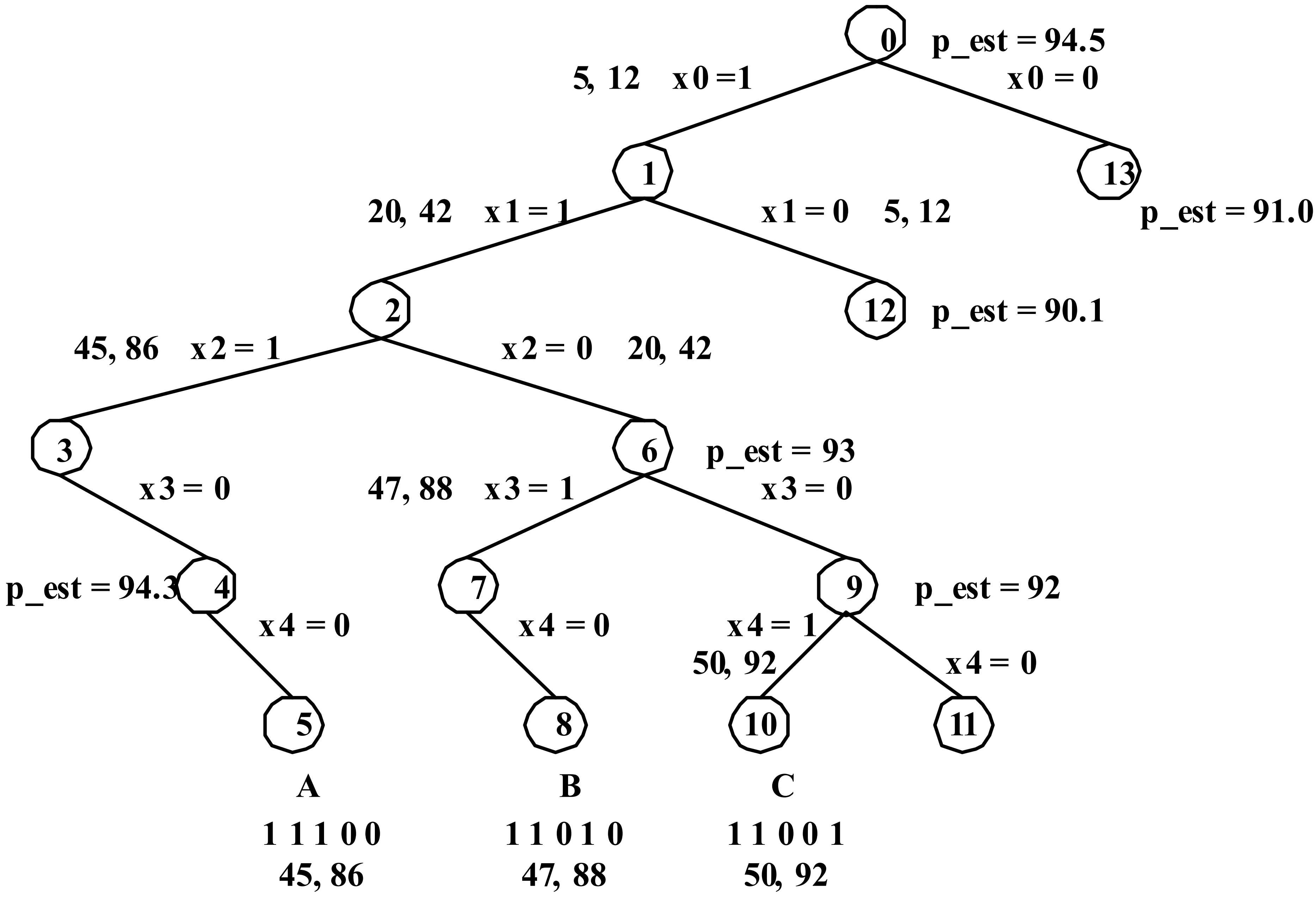
1. 若t>n, (若cv>bestv, bestv=cv), 返回
2. r -= v[t]
3. 若 cw+w[t] ≤ c, 则 ----------------------------------//x[t]=1
4. | cw+=w[t], cv+=v[t], backtrack(t+1), cv-=v[t], cw-=w[t]
5. 若 cv + r > bestv, 则
6. | backtrack(t+1) ------------------------------------//x[t]=0
7. r += v[t] //可添加提前更新最优值和找最优解, 优于备忘录

# 0-1背包回溯 $O(2^n)$

$$\sum_{i=0}^{k} x_i w_i + \sum_{i=k+1}^{k+m-1} w_i \le C \text{且} \sum_{i=0}^{k} x_i w_i + \sum_{i=k+1}^{k+m-1} w_i + w_{k+m} > C$$

$$\sum_{i=0}^{k} x_i v_i + \sum_{i=k+1}^{k+m-1} x_i v_i + (C - \sum_{i=0}^{k-1} x_i w_i - \sum_{i=k+1}^{k+m-1} x_i w_i) \times v_{k+m} / w_{k+m}$$

## C=50，5, 15, 25, 27, 30，价值12, 30, 44, 46, 50



# 第5章 回溯法

1. 装载问题与01背包
2. 回溯算法设计步骤
3. 旅行售货员问题(TSP)
4. n皇后问题
5. 最大团问题
6. 符号三角形
7. 回溯算法的效率

# 回溯算法设计步骤

1. 定义问题的解空间
2. 确定易于搜索的解空间结构
3. 设计约束和限界函数
4. 以深度优先方式搜索解空间

```
backtrack(t)
1. 若 t>n, 则判断 记录 返回  //记录更新
2. 对 i = f(n,t) : g(n,t)     //第t层扩展节点的起止编号
3. |   x[t]=h(i),             //第t层的第i个可选值
4. |   若C(t)且B(t), 则backtrack(t+1)   //剪枝
```

# 装载问题w[1:n],c

解空间: { x[i]∈{0,1}, 1 ≤ i ≤ n }//01串

解空间结构: 子集树, 1左0右

bestw, cw: 最佳质量, 当前质量

r: 当前剩余质量

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

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

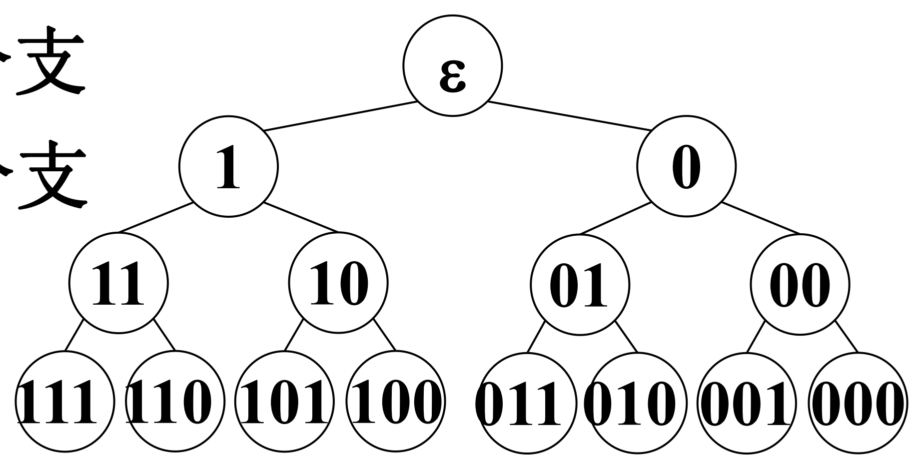
$$x_i \in \{0,1\}, 1 \le i \le n$$

cw ≤ c //约束条件

cw + r > bestw //限界条件

cw + w[i] ≤ c //用于左分支

cw + r > bestw //用于右分支



# 子集树回溯模型

**backtrack(int t)   //搜索到树的第t层**
**1. 若 t>n, 判断 记录 返回**
**2. 对 i = 0 : 1**
**3. |   若 满足 Constraint(t) 和 Bound(t)**
**4. |   则 backtrack(t+1);**

**Constraint(t) 约束条件**
**Bound(t) 限界条件**
**有必要时, 注意还原**

**backtrack(t)**

**1. 若t>n, 判断 记录 返回**

**2. r-=w[t]**

**3. 若 cw + w[t] ≤ c, 则**

**4. ｜ cw+=w[t], backtrack(t+1), cw-=w[t],**

**5. 若 cw + r > bestw, 则 backtrack(t+1)**

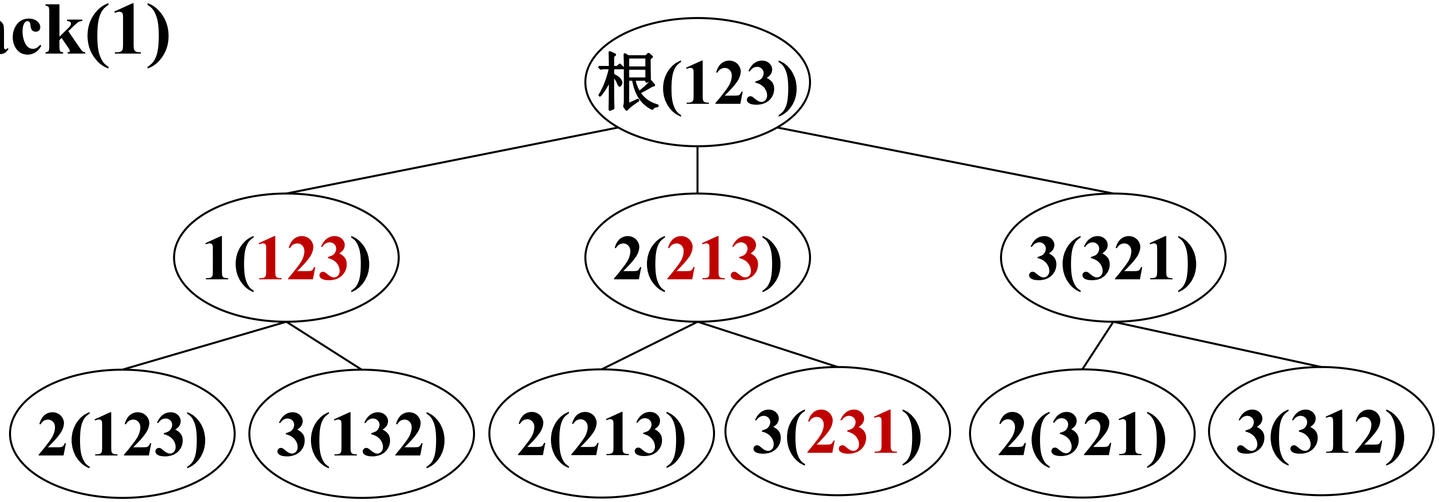
**6. r+=w[t]**

# 排列树回溯模型

backtrack(int t) //搜索树的第t层
1. 若 t>n, 判断 记录 返回
2. 对 i = t : n
3. | 交换 x[t] 和 x[i]
4. | 若 满足 Constraint(t) 且 Bound(t) //约束和限界条件
5. | 则 backtrack(t+1);
6. | 交换 x[t] 和 x[i]

执行backtrack(1)

初始x[i]=i



# 第5章 回溯法

1. 装载问题与01背包
2. 回溯算法设计步骤
3. 旅行售货员问题(TSP)
4. n皇后问题
5. 最大团问题
6. 符号三角形
7. 回溯算法的效率

# 旅行售货员问题(TSP)

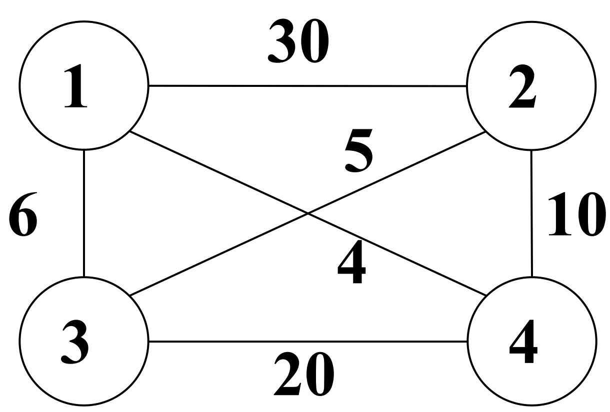
某售货员要到若干城市推销商品

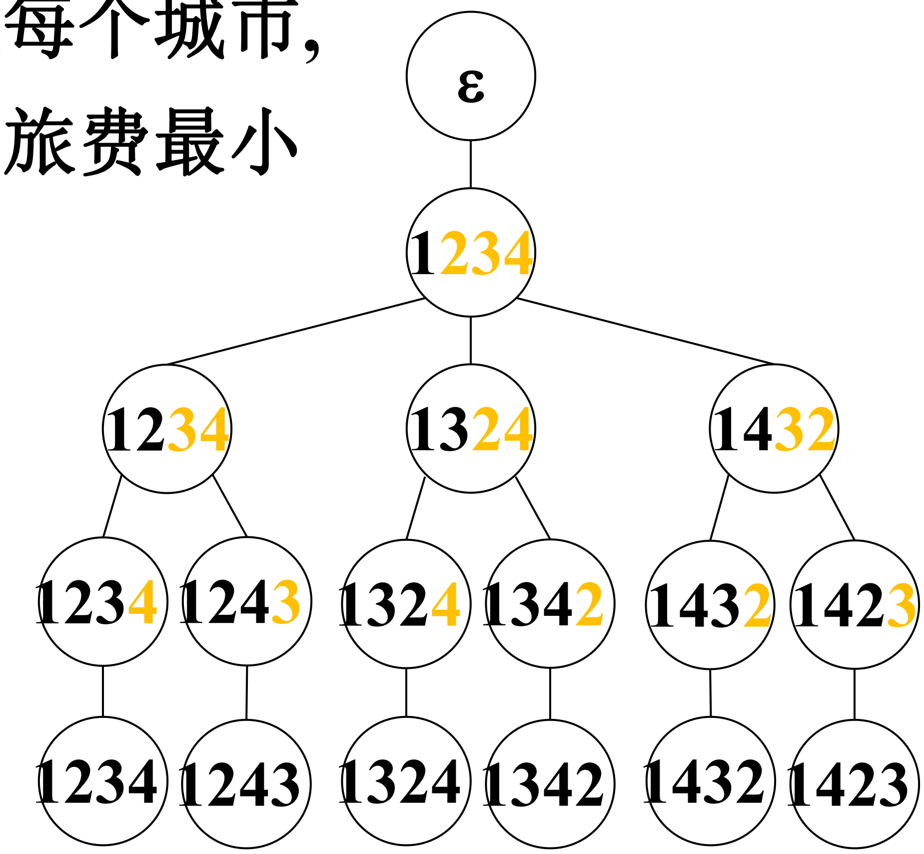
已知各城市间的旅费

选择一条从驻地出发, 经过每个城市, 最后回到驻地的路线, 使总旅费最小

解空间: 全体排列

解空间结构: 排列树



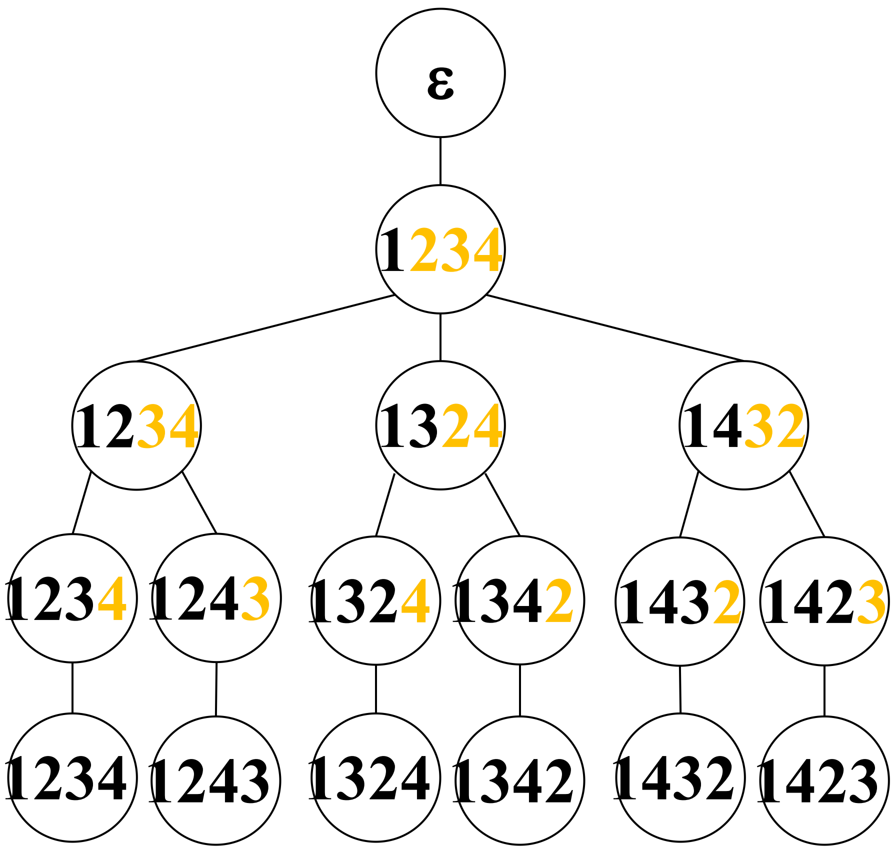



# TSP

backtrack(t)

1. 若 t > n, 则判断记录bestc, bestx, 返回
2. 对 j = t : n
3. |　交换x[t],x[j],
4. |　若x[1:t]费用<bestc, 则
5. |　|　　backtrace(i+1)
6. |　交换x[i],x[j]

- 初始x[i]=i, bestc=INF,
- backtrack(2)
- 分支数O((n-1)!), 时间O(n!)

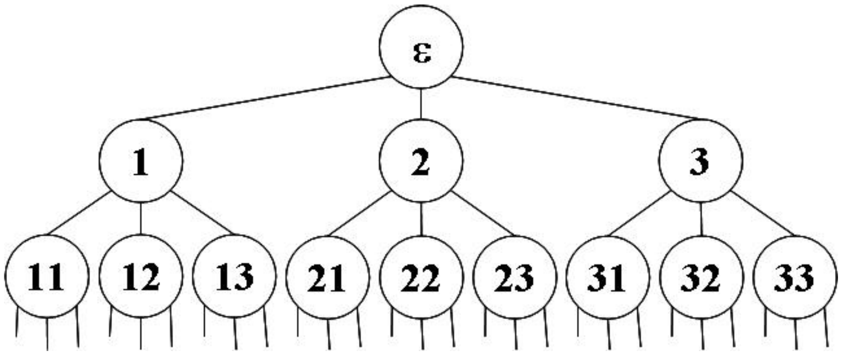


# 第5章 回溯法

1. 装载问题与01背包
2. 回溯算法设计步骤
3. 旅行售货员问题(TSP)
4. n皇后问题
5. 最大团问题
6. 符号三角形
7. 回溯算法的效率

# n皇后问题

- n×n棋盘放n个皇后, 要求彼此不能攻击
- 同行,同列或同一斜线上不能放两个皇后
- (i,j)与(i’,j’)有冲突 ⇔ i=i’ 或 j=j’ 或 |i-i’|=|j-j’|




backtrack(t)

1. 若 t > n, 则记录, 返回
2. 对 i = 1 : n
3. |　x[t]=i,　　//在第t行第x[t]列放一个皇后
4. |　若place(t), 则backtrace(t+1)

- place(t): 确定(t,x[t])与(i,x[i])无冲突(1≤i≤t-1),
- backtrack(1)

# n皇后问题-排列树版本

- n×n棋盘放n个皇后, x[1:n]排列
- (i,x[i])与(i',x[i'])有冲突 ⇔ |i-i'|=|x[i]-x[i']|

backtrack(t)

1. 若 t > n, 则记录, 返回
2. 对 i = t : n
3. | 交换x[i]和x[t]
4. | 若place(t), 则backtrace(t+1)
5. | 交换x[i]和x[t]

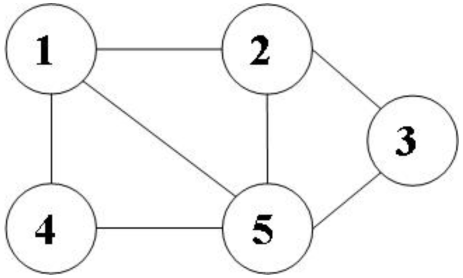
- place(t): 确定(t,x[t])与(i,x[i])无冲突(1≤i≤t-1),
- 初始x[i]=i, backtrack(1)

# 第5章 回溯法

1. 装载问题与01背包
2. 回溯算法设计步骤
3. 旅行售货员问题(TSP)
4. n皇后问题
5. 最大团问题
6. 符号三角形
7. 回溯算法的效率

# 最大团问题

- 无向图G=(V,E). G的完全子图称为团:
  即U⊆V满足∀u,v∈U, 都有(u,v)∈E
- 最大团: 顶点数最多的团(子集).
- 解空间结构: 子集树
- x[t]=1或0表示取或不取$v_t$.




bestn: 目前最大团顶点数
cn: 当前团顶点数

backtrack(t)  // bestx当前最优解

1. 若 t > n, (若cn>bestn, 更新bestn,bestx), 返回
2. 若$v_t$与x[1:t-1]中已有的cn个点都相连, 则
3. |  x[t]=1, cn++, backtrack(t+1), cn--, x[t]=0
4. 若cn+n-t>bestn, 则
5. |  x[t]=0, backtrace(t+1)

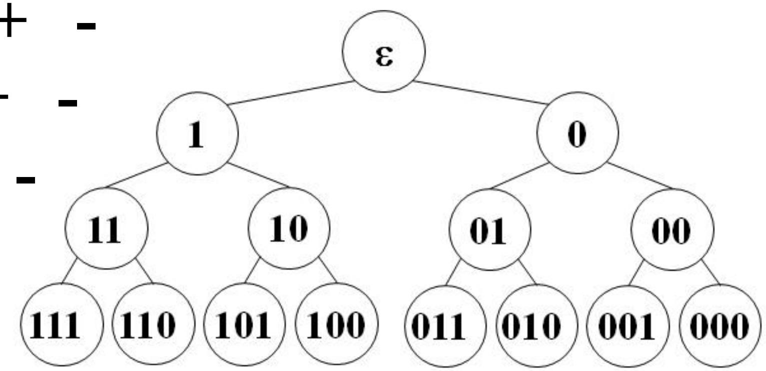
# 第5章 回溯法

1. 装载问题与01背包
2. 回溯算法设计步骤
3. 旅行售货员问题(TSP)
4. n皇后问题
5. 最大团问题
6. 符号三角形
7. 回溯算法的效率

# 符号三角形

- 由符号“+”, “-”组成
- 第一行有n个符号
- 两个同号下面是“+”
- 两个异号下面是“-”
- 右图有14“+”, 14“-”

```
+ + - + - + +
 + - - - - +
  - + + + -
   - + + -
    - + -
     - -
      +
```




- 问题: 给定n, 求“+”“-”个数相同的符号三角形个数
- 如何遍历所有符号三角形? 约束条件? 限界条件?
- 第一行取遍所有长为n的+-符号串:子集树
- 限界条件: 0,1的个数都不能超过n(n+1)/4

# 符号三角形

- sum累计+-个数相同的符号三角形个数
- 若n(n+1)/2是奇数则sum=0, 返回.
- half = n(n+1)/4
- p维护当前符号三角形
- p[1][t] = x[t]
- 对j=2:t, 依次计算 p[j][t-j+1]

```
+ + - + - + +
 + - - - - +
  - + + + -
   - + + -
    - + -
     - -
      +
```

# 符号三角形

```
//sum记+-个数相同的符号三角形个数
//half=n(n+1)/4, p维护当前符号三角形
backtrack (t)
1. 若count>half 或 t(t-1)/2-count>half, 返回
2. 若t>n, sum++,返回
3. 对 i = 0 : 1,
4. |   p[1][t]=i; count+=i;
5. |   对 j = 2 : t,
6. |   |  p[j][t-j+1]=p[j-1][t-j+1]^p[j-1][t-j+2];
7. |   |  count+=p[j][t-j+1];
8. |   backtrack(t+1);
9. |   对 j = 2 : t,  count-=p[j][t-j+1];
10.|  count-=i;  //可与9合并
```

```
0 0 0
0 0
0
-----
0 0 1
0 1
1
-----
0 1 0
1 1
0
-----
0 1 1
1 0
1
```

# 符号三角形

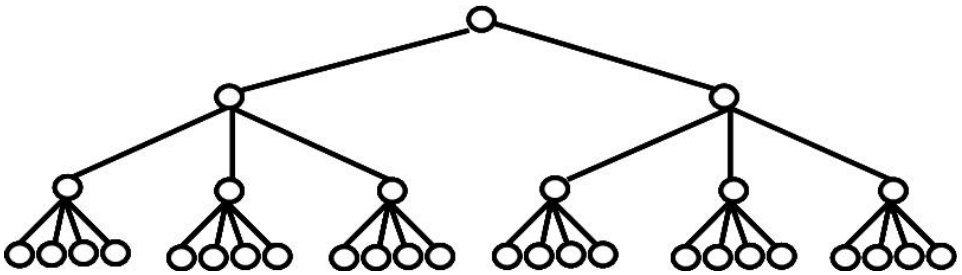
```
//sum记+-个数相同的符号三角形个数
//half=n(n+1)/4, p维护当前符号三角形
backtrack (t)
1.   若t>n, sum++,  返回
2.   对 i = 0 : 1,
3.   |   p[1][t]=i; count+=i;
4.   |   对 j = 2 : t,
5.   |    |   p[j][t-j+1]=p[j-1][t-j+1]^p[j-1][t-j+2];
6.   |    |   count+=p[j][t-j+1];
7.   |    若 count≤half 且 (t+1)t/2-count≤half
8.   |    |   backtrack(t+1);
9.   |   对 j = 1 : t,  count-=p[j][t-j+1];
```

```
0 0 0
0 0
0
---
0 0 1
0 1
1
---
0 1 0
1 1
0
---
0 1 1
1 0
1
```

# 第5章 回溯法

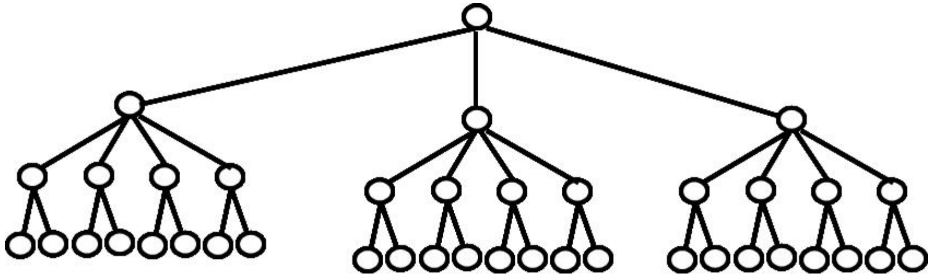
1. 装载问题与01背包
2. 回溯算法设计步骤
3. 旅行售货员问题(TSP)
4. n皇后问题
5. 最大团问题
6. 符号三角形
7. 回溯算法的效率

# 回溯法的效率分析



影响回溯算法效率的因素

1. 每个顶点的产生时间
2. 计算剪枝函数的时间
3. 剪枝后剩余顶点个数



剪枝函数的设计与平衡? 更好的剪枝会增加计算时间

# 回溯法的效率举例

教材[王]中对n皇后问题的回溯效率进行了概率估计

对于n=8的情形

估计搜索节点数/总节点数 ≈ 1.55%

总节点数 $= \sum_{i=1}^{n} n!/(n-i)! = 109601$ (n=8)

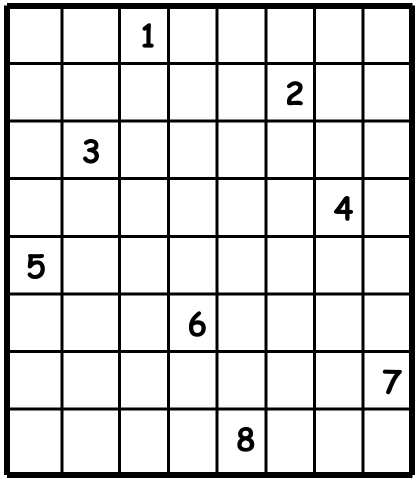
搜索节点数估计:

记第i层x[i]的可选列数为$m_i$,

随机选一可选列进入下一层

得节点数$1+m_1+m_1m_2+m_1m_2m_3+\ldots$

多次取平均得1702




2329

$m_1=8$; $m_2=5$: 因为2可选4-8列; $m_3=4$: 因为3可选1,6,7,8;

$m_4=3$: 因为4可选3,7,8; $m_5=2$: 因为5可选5,8;

# 回溯法的效率举例

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1 | | | | | |
| | X | X | X | | 2 | | |
| X | 3 | X | | X | X | X | |
| X | X | X | X | | X | 4 | X |
| 5 | X | X | X | | X | X | X |
| X | X | X | 6 | X | X | X | X |
| X | X | X | X | X | X | X | 7 |
| X | X | X | X | 8 | X | X | X |

1+8+8*5+8*5*3

+8*5*3*2

+8*5*3*2*2

+8*5*3*2*2*1

+8*5*3*2*2*1*1

+8*5*3*2*2*1*1*1
=2329