算法设计与分析

# 第10讲 回溯与分支限界

汪小林

北京大学 信息科学技术学院

# 主要内容

- 回溯算法的基本思想和适用条件
- 回溯算法的设计步骤
- 估计回溯算法的效率
- 改进回溯算法的途径
- 分支估界
- 应用实例

# 基本思想和适用条件

- 三个实例
  - 四皇后、0-1背包、巡回售货员
- 基本思想

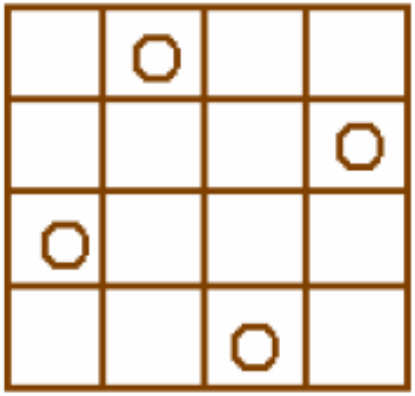
  搜索问题、搜索空间、搜索策略

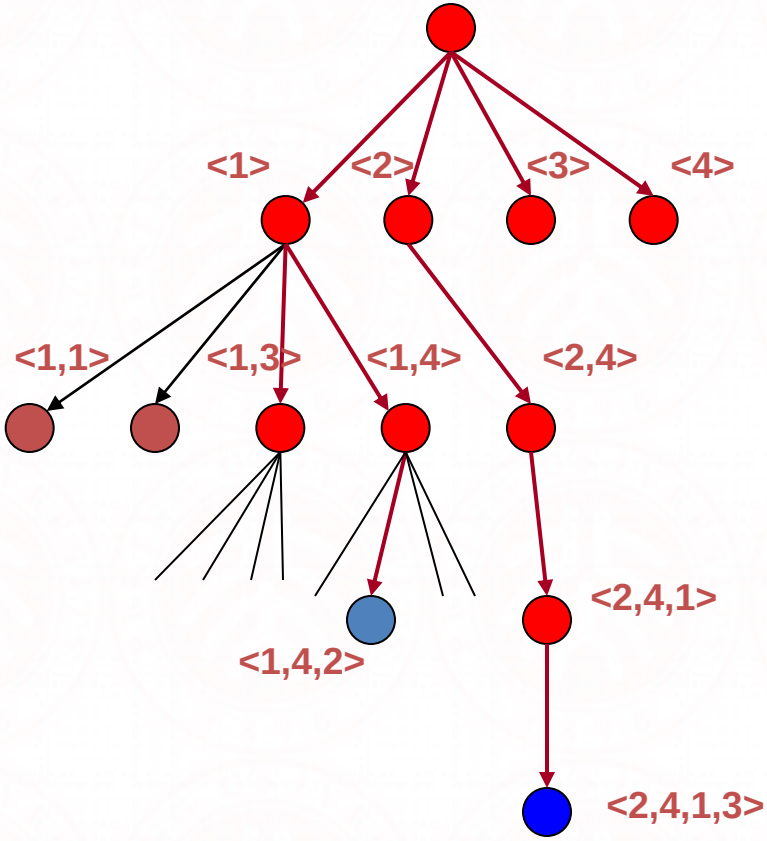
  判定条件、结点状态、存储结构
- 必要条件
  - 多米诺性质

# 例1　四后问题

解表示成一个4维向量，$<x_1, x_2, x_3, x_4>$（放置列号）

搜索空间：**4叉树**



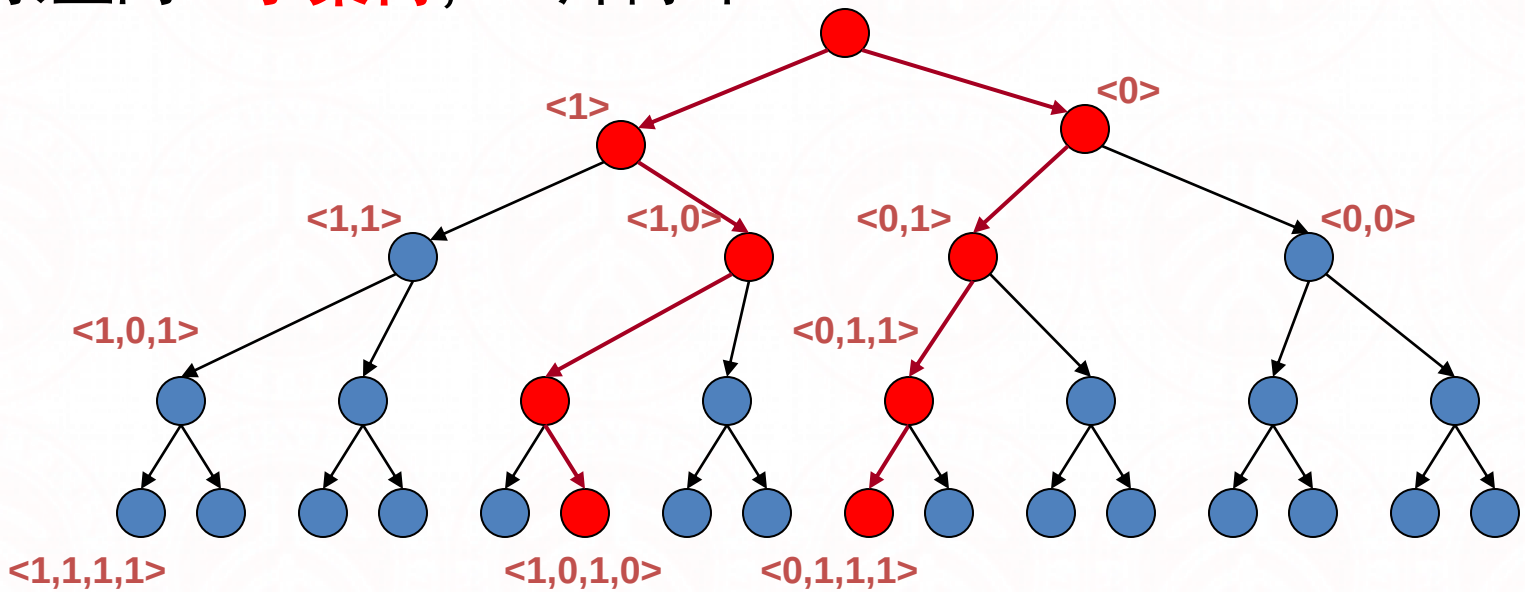


# 例2　0-1背包问题

$V$={12,11,9,8}, $W$={8,6,4,3}, $B$=13

结点：向量<$x_1$, $x_2$, $x_3$, $x_4$>（子集的部分特征向量）

搜索空间：**子集树**，$2^n$片树叶




对应于可行解

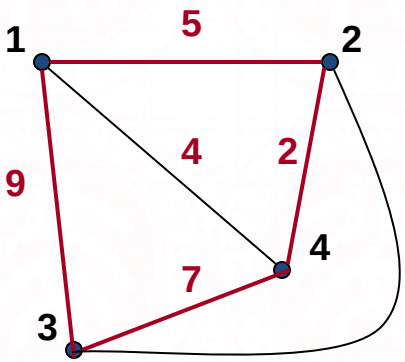
$x_1$=1, $x_2$=0, $x_3$=1, $x_4$=0.
重量：12，价值：21

$x_1$=0, $x_2$=1, $x_3$=1, $x_4$=1.
重量：13, 价值：28

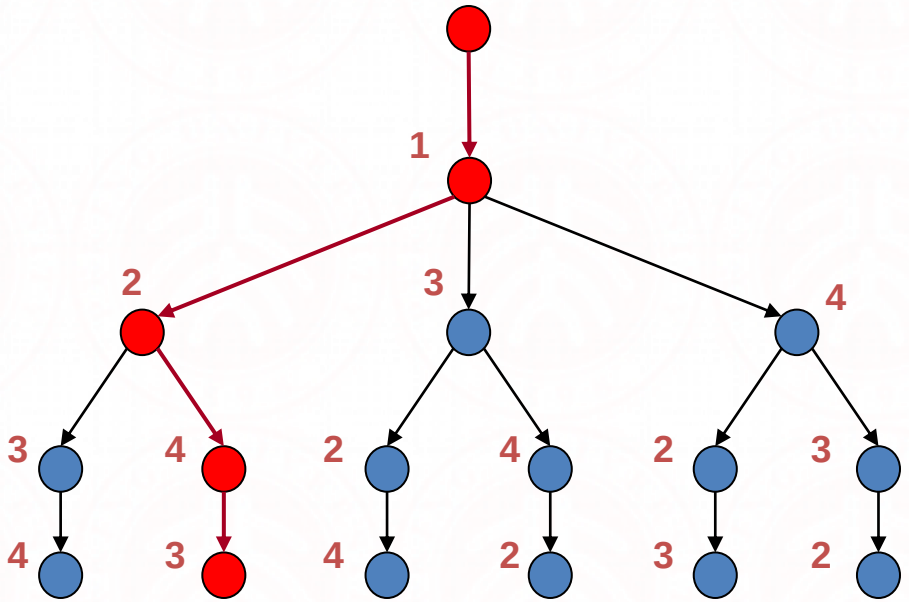
# 例3　巡回售货员问题

结点：向量$<x_1, x_2, x_3, x_4>$（部分巡回路线）

搜索空间：**排列树**




13




**<1,2,4,3>**

对应于巡回路线：

**1→2 →4 →3 →1**

长度：**5+2+7+9=23**

# 回溯法的基本思想

- 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，
  - 可达的树叶对应了可行解
- 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

# 回溯法的基本思想

- 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
    - 新结点（尚未访问）；
    - 活结点（正在访问该结点为根的子树）
    - 死结点（该结点为根的子树遍历完成）
- 存储：
  - 当前路径

# 必要条件——多米诺性质

在$n$皇后问题中

设$P(x_1, x_2, \ldots, x_i)$为真表示向量$<x_1, x_2, \ldots, x_i>$中$i$个皇后放置在彼此不能攻击的位置，则

$$P(x_1, x_2, \ldots, x_{k+1}) \rightarrow P(x_1, x_2, \ldots, x_k) \qquad 0 < k < n$$

即部分解向量一定满足于解向量的约束；

反之，如果部分向量不满足解向量约束，则无法从此部分向量扩充为一个解向量（可剪枝）。

# 回溯算法的设计步骤

- 定义搜索问题的解向量和每个分量的取值范围
  - 解向量为$<x_1,x_2,\ldots,x_n>$
  - $x_i$的可能取值的集合为$X_i$, $i = 1, 2, \ldots, n$.
  - $x_1, x_2, \ldots, x_{k-1}$确定以后$x_k$的取值集合为$S_k$, $S_k \subseteq X_k$
- 确定结点儿子的排列规则
- 判断是否满足多米诺性质
- 搜索策略——深度优先
- 确定每个结点能够分支的约束条件
- 确定存储搜索路径的数据结构

# 递归回溯

算法**ReBack(*k*)**

**1. if** $k > n$ **then** $< x_1, x_2, \ldots, x_n >$是解;

**2. else while** $S_k \neq \emptyset$ **do**

**3.** $x_k \leftarrow S_k$中最小值

**4.** $S_k \leftarrow S_k - \{x_k\}$

**5.** 计算$S_{k+1}$

**6.** **ReBack(*k*+1)**

算法**ReBacktrack(*n*)**

**1. for** $i \leftarrow 1$ **to** $n$ 计算$X_k$

**2. ReBack(1)**

# 迭代回溯

**迭代算法Backtrack**

**1. 对于$i = 1, 2, \ldots, n$ 确定$X_i$**

**2. $k = 1$**

**3. 计算$S_k$**

**4. while $S_k \neq \emptyset$ do**

**5. $x_k \leftarrow S_k$中最小值; $S_k \leftarrow S_k - \{x_k\}$**

**6. if $k<n$ then**

**7. $k \leftarrow k+1$; 计算$S_k$**

**8. else $<x_1, x_2, \ldots, x_n>$是解**

**9. if $k > 1$ then $k \leftarrow k-1$; goto 4**

# 估计回溯算法的平均效率

计数搜索树中平均遍历的结点，Monte Carlo方法

Monte Carlo方法

1. 从根开始，随机选择一条路经，直到不能分支为止，即从$x_1, x_2, \ldots$依次对$x_i$赋值，每个$x_i$的值是从当时的$S_i$中随机选取，直到向量不能扩张为止
2. 假定搜索树的其他$|S_i|-1$个分支与以上随机选出的路径一样，计数搜索树的点数。
3. 重复步骤1和2，将结点数进行概率平均。

# 平均效率估计的算法

算法Monte Carlo

1. *sum*←0　　//*sum*为 *t* 次结点平均数
2. for *i*←1 to *t* do　　//取样次数 *t*
3. 　　*m*←Estimate(*n*)　　//*m*为本次结点总数
4. 　　*sum*←*sum*+*m*
5. *sum*←*sum*/*t*

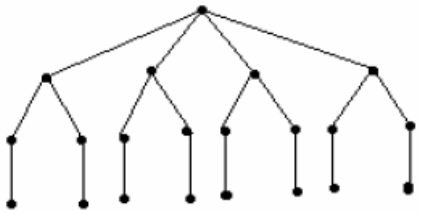
# 一次取样的结点数计算

$m$为本次取样结点总数，$k$ 为层数，$r_1$为本层分支数，$r_2$为上层分支数，$n$为树的层数。算法Estimate($n$)

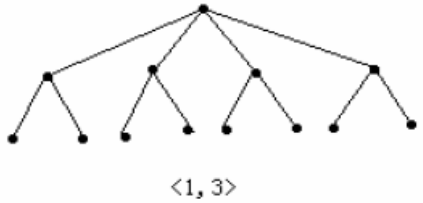
1. $m \leftarrow 1$; $r_2 \leftarrow 1$; $k \leftarrow 1$ //$m$为结点总数
2. While $k \le n$ do
3. if $S_k = \varnothing$ then return $m$
4. $r_1 \leftarrow |S_k| * r_2$ //$r_1$为扩张后结点总数
5. $m \leftarrow m + r_1$ // $r_2$为扩张前结点总数
6. $x_k \leftarrow$ 随机选择 $S_k$ 的元素
7. $r_2 \leftarrow r_1$
8. $k \leftarrow k+1$

# Monte Carlo方法估计四后问题的效率

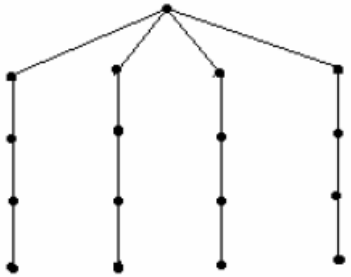
- **case1．<1,4,2>：1+4+4×2+4×2 = 21**
- **case2．<2,4,1,3>::：4×4+1=17**
- **case3．<1,3>：1+4×1+4×2=13**
- 假设4次抽样测试：**case1** 1次，**case2** 1次，**case3** 2次，平均为16
- 解空间的结点数为17



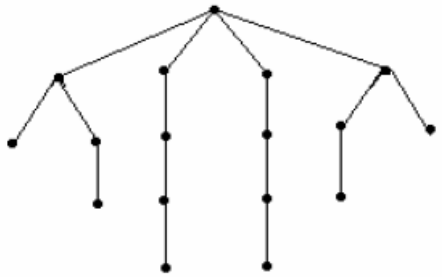
<1, 4, 2>



<1, 3>



<2, 4, 1, 3>



解空间

# 影响算法效率的因素

- 最坏情况下的时间$W(n)=(p(n)f(n))$
  - 其中$p(n)$每个结点时间，$f(n)$结点个数
- 影响回朔算法效率的因素
  - 搜索树的结构
    - 分支情况：分支均匀否
    - 树的深度
    - 对称程度：对称适合裁减
  - 解的分布
    - 在不同子树中分布多少是否均匀
    - 分布深度
  - 约束条件的判断：计算简单

# 改进途径

- 根据树分支设计优先策略：
  - 结点少的分支优先
  - 解多的分支优先
- 利用搜索树的对称性剪裁子树
- 分解为子问题:
  - 问题求解时间$f(n)=c2^n$，子问题组合时间$T=O(n)$
  - 如果分解为$k$个子问题，每个子问题大小为$n/k$
  - 则求解时间为$kc2^{n/k}+T$
  - 例如：TSP问题，包含两条边的最小割划分图。

# 组合优化问题的描述

- 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
- 例5 背包问题：

$$\max x_1 + 3x_2 + 5x_3 + 9x_4$$

$$2x_1 + 3x_2 + 4x_3 + 7x_4 \leq 10$$

$$x_i \in N, i = 1,2,3,4$$

# 分支估界技术

- 设立代价函数（极大化）
  - 函数值是以该结点为根的搜索树中的所有可行解的目标函数值的上界
  - 父结点的代价大于等于子结点的代价
- 设立界
  - 代表当时已经得到的可行解的目标函数的最大值
- 搜索中停止分支的依据
  - 不满足约束条件或者其代价函数小于当时的界
- 界的设定与更新（目标函数值为正数）
  - 初值可以设为**0**
  - 可行解的目标函数值大于当时的界，进行更新

# 例5 背包问题

背包问题：

$$\max x_1 + 3x_2 + 5x_3 + 9x_4$$
$$2x_1 + 3x_2 + 4x_3 + 7x_4 \le 10$$
$$x_i \in N, i = 1,2,3,4$$

对变元重新排序：

$$\frac{v_i}{w_i} \ge \frac{v_{i+1}}{w_{i+1}}$$

$$\max 9x_1 + 5x_2 + 3x_3 + x_4$$
$$7x_1 + 4x_2 + 3x_3 + 2x_4 \le 10$$
$$x_i \in N, i = 1,2,3,4$$

# 代价函数与分支策略的设定

$$F(<x_1, x_2, \ldots, x_k>) =$$

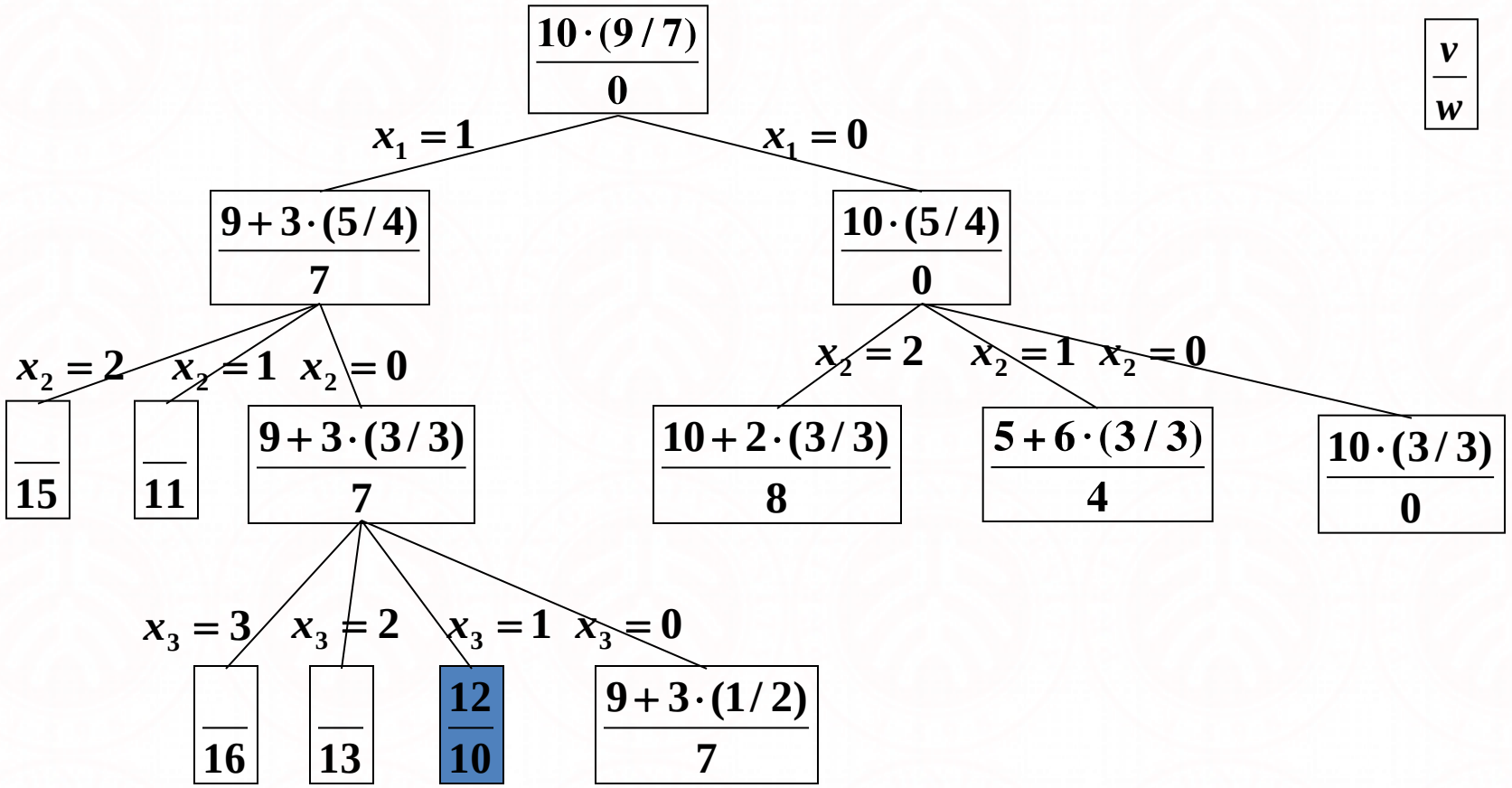
$$\begin{cases} \sum_{i=1}^{k} v_i x_i + \left(b - \sum_{i=1}^{k} w_i x_i\right) \dfrac{v_{k+1}}{w_{k+1}}, & \text{if } \exists j > k \text{ st } \left(b - \sum_{i=1}^{k} w_i x_i\right) \geq w_j \\ \sum_{i=1}^{k} v_i x_i, & \text{else} \end{cases}$$

分支策略——深度优先

$$\max 9x_1 + 5x_2 + 3x_3 + x_4$$

$$7x_1 + 4x_2 + 3x_3 + 2x_4 \le 10$$

$$x_i \in N, i = 1,2,3,4$$



# 分支限界法基本思想

- 对问题的解有费用函数：$f(x)$
- 对问题的解空间树，分支限界法以广度优先（或以最小费用优先）的方式搜索。
- 分支：每一个活结点只有一次机会成为扩展结点。活结点一旦成为扩展结点，就一次性产生其所有儿子结点。
- 首先舍弃所有导致不可行解的儿子结点。对最其他儿子结点，计算结点费用函数$f(x)$的上界$u(x)$和下界$l(x)$，加入当前扩展结点表。
- 限界：扩展结点表中，如果一个结点$x$的下界$l(x)$ 大于或等于其它某个结点$y$的上界$u(y)$，则基于结点$x$不可能扩展出最优解，可舍弃之。
- 此后，从活结点表中取下一结点成为当前扩展结点，并重复上述结点扩展过程。直到当前扩展结点表只剩下一个结点或当前扩展结点集合的上界与下界相等时（则集合中任意结点扩展出的任意解均为最优解）。

# 分支限界法的最佳实现方式

- 优先队列式分支限界法 - LCBB
  - 按照规定的结点费用最小（LC: $l(x)$**最小**）优先的原则选取下一个结点为扩展结点（常采用优先队列实现）。
  - 当扩展结点为叶结点时（$f(x)=u(x)=l(x)$），队列中所有结点的下界$l(y)\geq l(x)=f(x)$，所以此结点即为最优解。
    也就是说：第一个被扩展的叶结点即最优解。
  - 对于优先队列中的任意结点$x$，如果存在结点$y$使得$l(y)\geq u(x)$，则$y$不再可能被选中为扩展结点。相当于$y$已经被从扩展结点表中删除。

# 分支限界策略

- 两种机制:
  - 产生分支的机制
  - 产生界的机制
    - 目标：使更多的分支通过界的限制终止
- 平均情形下都很高效，很多分支可以较早剪去
- 虽然通常高效，但最坏情形下搜索空间非常大
- 对NPC问题
  - 很多NPC问题可以用分支限界方法高效的求解（平均情形下），但最坏情形下的复杂性仍然是指数级的

# 例6 个人作业分配问题

一组线性序的人的集合$P=\{P_1, P_2, \ldots, P_n\}$，其中$P_1<P_2<\ldots<P_n$

一组偏序的作业集合$J=\{J_1, J_2, \ldots, J_n\}$

- 设$P_i$和$P_j$被分配到任务$f(P_i)$和$f(P_j)$，
  如果$f(P_i) \le f(P_j)$，则$P_i \le P_j$。
- 设$C_{ij}$ 是把作业$J_j$分配给$P_i$的费用。
- 我们要找一个使费用最小的可行的作业安排。
- 即求矩阵$X$，其中：

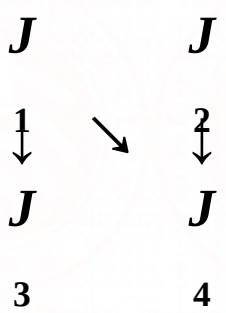
  $X_{ij} = 1$ 表示$P_i$被分配到任务$J_j$

  $X_{ij} = 0$ 否则

  使得$\sum_{i,j} C_{ij} X_{ij}$最小。

# 拓扑排序

- 例：一组偏序的任务




- 拓扑排序后，会产生下列中的一组序列：

$J_1, J_2, J_3, J_4$

$J_1, J_2, J_4, J_3$

$J_1, J_3, J_2, J_4$

$J_2, J_1, J_3, J_4$

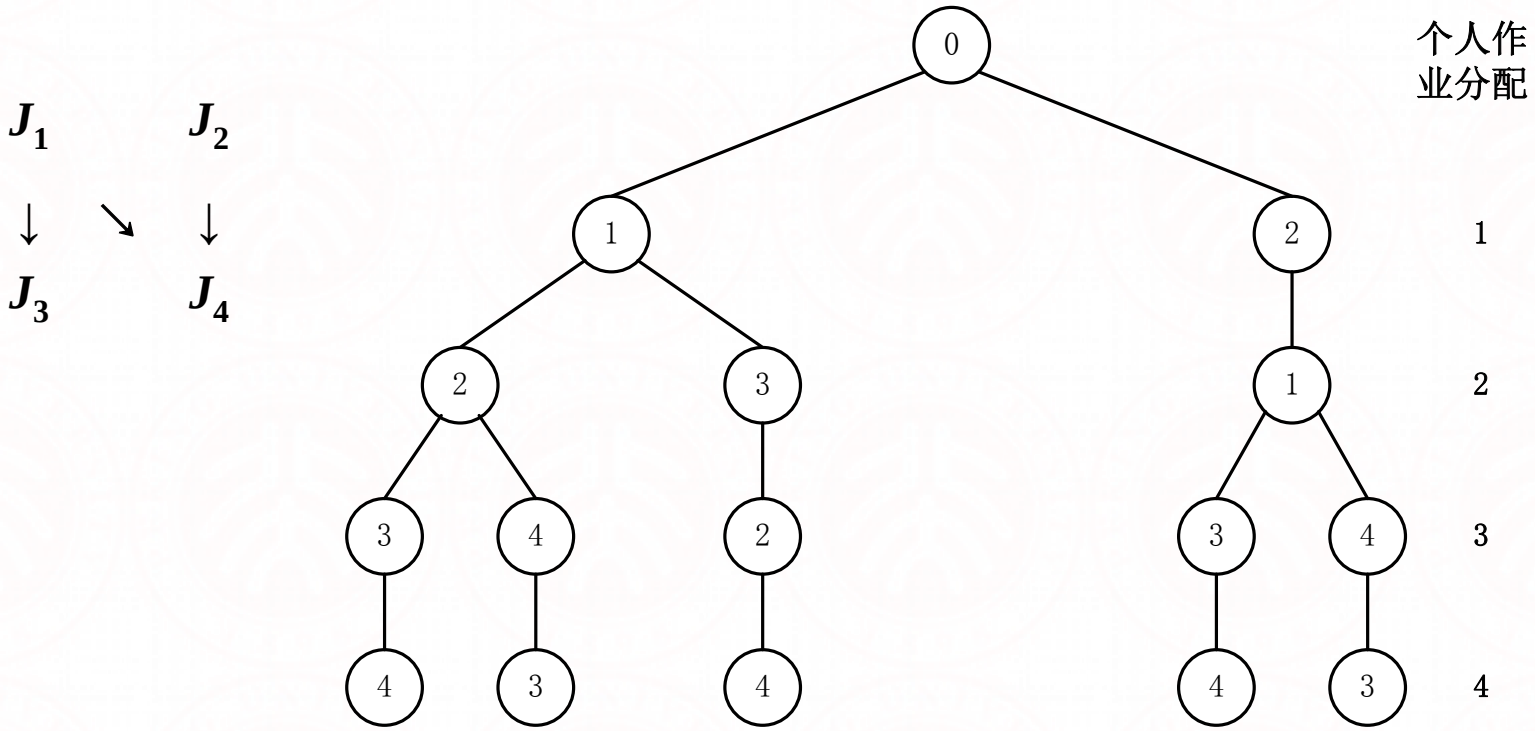
$J_2, J_1, J_4, J_3$

- 一种可行的任务安排：

$P_1 \rightarrow J_1, P_2 \rightarrow J_2, P_3 \rightarrow J_3, P_4 \rightarrow J_4$

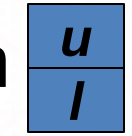
# 解空间树

- 所有可能的解都可通过一个遍历解空间树给出



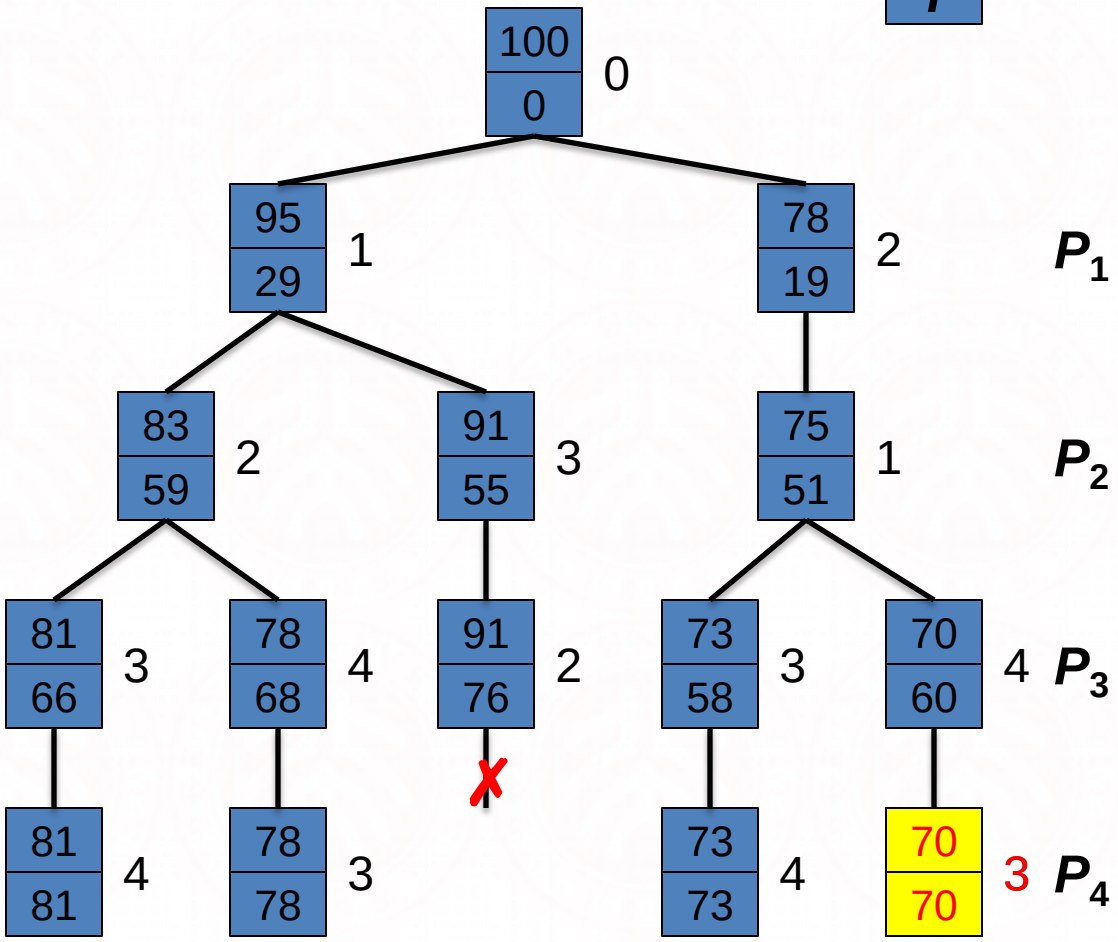
# 费用矩阵

采用最佳优先搜索方式：Best-First-Search




费用矩阵

| 作业 / 人 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 29 | 19 | 17 | 12 |
| 2 | 32 | 30 | 26 | 28 |
| 3 | 3 | 21 | 7 | 9 |
| 4 | 18 | 13 | 10 | 15 |




仅一个分支被剪除

# 费用矩阵化简

## 费用矩阵

| 作业<br>人 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 29 | 19 | 17 | 12 |
| 2 | 32 | 30 | 26 | 28 |
| 3 | 3 | 21 | 7 | 9 |
| 4 | 18 | 13 | 10 | 15 |

## 化简的费用矩阵

| 作业<br>人 | 1 | 2 | 3 | 4 | |
|---|---|---|---|---|---|
| 1 | 17 | 4 | 5 | 0 | (-12) |
| 2 | 6 | 1 | 0 | 2 | (-26) |
| 3 | 0 | 15 | 4 | 6 | (-3) |
| 4 | 8 | 0 | 0 | 5 | (-10) |
| | | (-3) | | | |

# 费用矩阵化简

- 可以获得一个化简的费用矩阵：

  从每行/每列减去相应的常数，使得每行/每列都至少包含一个**0**项。

- 总的减去的费用为：

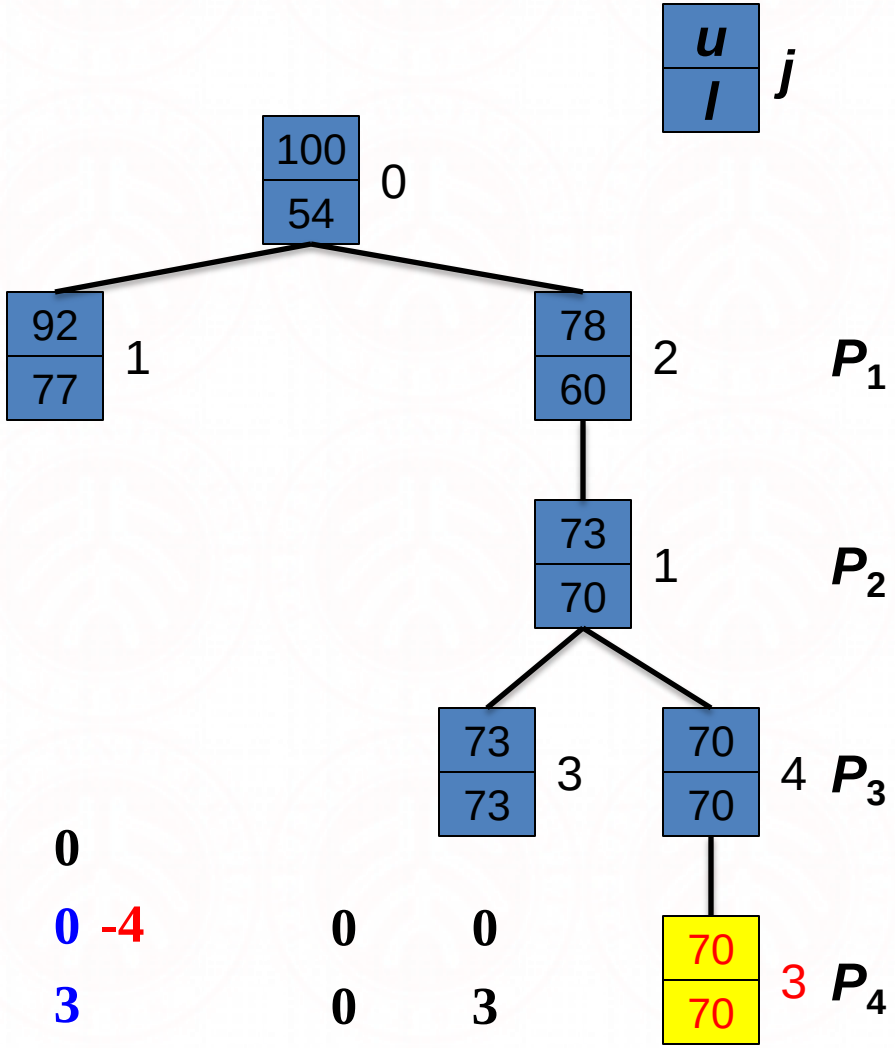
  **12+26+3+10+3 = 54**

- 这个数值是所有解的下界。

# 在化简的费用矩阵上采用分支限界法

| 人 \ 任务 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 17 | 4 | 5 | 0 |
| 2 | 6 | 1 | 0 | 2 |
| 3 | 0 | 15 | 4 | 6 |
| 4 | 8 | 0 | 0 | 5 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | | | | | | |
| 11 | 0 | 0 | -4 | | | | | |
| 0 | 0 | 3 | | | | | | |
| | | -2 | | | | | | |
| 6 | 0 | 0 | | 6 | 0 | 0 | | |
| 0 | 4 | 4 | | 0 | 0 | 0 | -4 | 0 | 0 |
| 8 | 0 | 3 | | 8 | 0 | 3 | | 0 | 3 |
| | | -2 | | | | | | |





33

# 例7 巡回售货员问题（TSP）

- 给定一个图, TSP最优化问题为找出一个遍历路线，从任意点出发，访问到每个点后回到出发点，费用最小。
- 这是一个NP完全的问题。
- 我们将通过分支限界法避免高达$n!$的运行时间。 (我们知道：$O(n!)>O(2^n)$)

# “回溯法”求解TSP

- 问题：给定$n$个城市集合$C=\{c_1,c_2,\ldots,c_n\}$，从一个城市到另一个城市的距离$d_{ij}$为正整数，求一条最短且每个城市恰好经过一次的巡回路线。
- 巡回售货员问题的类型：有向图、无向图。

- 设巡回路线从1开始，解向量为$<i_1,i_2,\ldots,i_{n-1}>$，其中$i_1,i_2,\ldots,i_{n-1}$为$\{2,3,\ldots,n\}$的排列。
- 搜索空间为排列树，结点$<i_1,i_2,\ldots,i_k>$表示得到$k$步巡回路线。

# 分析

- 约束条件
  - 令$B=\{i_1,i_2,\ldots,i_k\}$, 则$i_{k+1}\in\{2,\ldots,n\}-B$
- 界：当前得到的最短巡回路线长度
- 代价函数
  - 设顶点$c_i$出发的最短边长度为$l_i$，$d_j$为选定的部分巡回路线中第$j$ 段的长度，则

$$L=\sum_{j=1}^{k}d_j+\sum_{i_j\notin B}l_{i_j}$$

为部分巡回路线扩张成全程巡回路线的长度下界。

- 时间$O(n!)$：计算$O((n-1)!)$次，代价函数计算$O(n)$

# TSP问题的费用矩阵

| j / i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | ∞ | 3 | 93 | 13 | 33 | 9 | 57 |
| 2 | 4 | ∞ | 77 | 42 | 21 | 16 | 34 |
| 3 | 45 | 17 | ∞ | 36 | 16 | 28 | 25 |
| 4 | 39 | 90 | 80 | ∞ | 56 | 7 | 91 |
| 5 | 28 | 46 | 88 | 33 | ∞ | 25 | 57 |
| 6 | 3 | 88 | 18 | 46 | 92 | ∞ | 7 |
| 7 | 44 | 26 | 33 | 27 | 84 | 39 | ∞ |

# 基本思路

- 存在划分解空间的方法（分支）
- 存在预测一类解的下限的方法。
- 也存在求解一个最优解的上界的方法。
- 如果一个解的下限超出了这个上界，则这个解不可能是最优解，此时这个分支就可以被剪去。


38

# 划分

- 把解划分为两组：
  - 一组包含某个边
  - 另一组不包含这条边
- 每次划分，都会产生一个更大的下限，我们基于这个更大的下限遍历搜索树。


39

# TSP最优化问题

- 化简费用矩阵

| j / i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|
| 1 | ∞ | 0 | 90 | 10 | 30 | 6 | 54 | (-3) |
| 2 | 0 | ∞ | 73 | 38 | 17 | 12 | 30 | (-4) |
| 3 | 29 | 1 | ∞ | 20 | 0 | 12 | 9 | (-16) |
| 4 | 32 | 83 | 73 | ∞ | 49 | 0 | 84 | (-7) |
| 5 | 3 | 21 | 63 | 8 | ∞ | 0 | 32 | (-25) |
| 6 | 0 | 85 | 15 | 43 | 89 | ∞ | 4 | (-3) |
| 7 | 18 | 0 | 7 | 1 | 58 | 13 | ∞ | (-26) |
| | | | | | | | | reduced:84 |

化简的费用矩阵



40

# TSP最优化问题

| j / i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | ∞ | 0 | 83 | 9 | 30 | 6 | 50 |
| 2 | 0 | ∞ | 66 | 37 | 17 | 12 | 26 |
| 3 | 29 | 1 | ∞ | 19 | 0 | 12 | 5 |
| 4 | 32 | 83 | 66 | ∞ | 49 | 0 | 80 |
| 5 | 3 | 21 | 56 | 7 | ∞ | 0 | 28 |
| 6 | 0 | 85 | 8 | 42 | 89 | ∞ | 0 |
| 7 | 18 | 0 | 0 | 0 | 58 | 13 | ∞ |
| | | | (-7) | (-1) | | | (-4) |

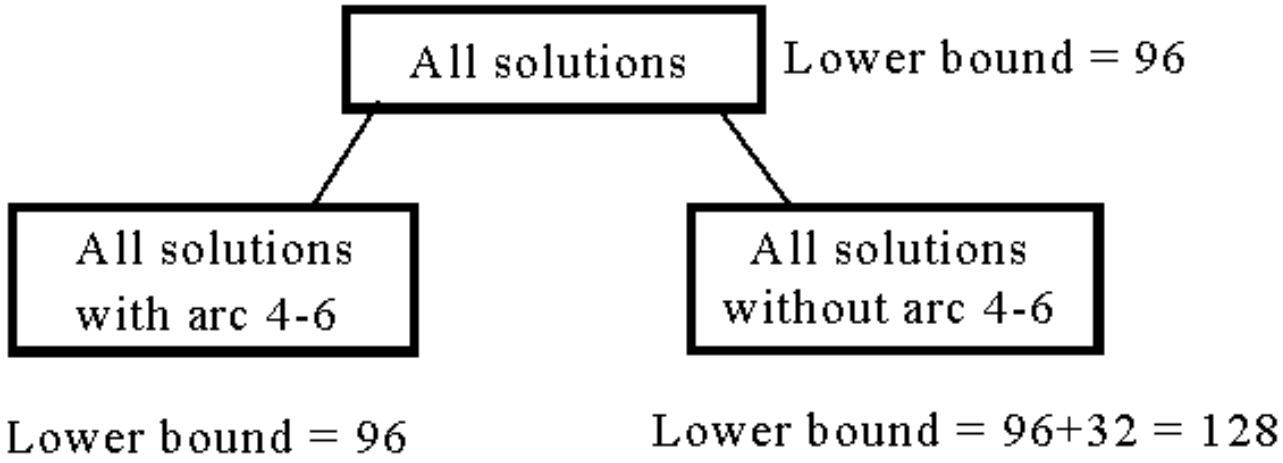
进一步化简费用矩阵



41

# TSP最优化问题

- 总的减去的费用为**84+12=96** (下限)
  决策树:




决策树的最高层

- 如果选**3-5**这条边划分，下限的变化为**17+1 = 18**.


42

# 对于右子树

| j<br>i | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | ∞ | 0 | 83 | 9 | 30 | 6 | 50 |
| 2 | 0 | ∞ | 66 | 37 | 17 | 12 | 26 |
| 3 | 29 | 1 | ∞ | 19 | 0 | 12 | 5 |
| 4 | 32 | 83 | 66 | ∞ | 49 | ∞ | 80 |
| 5 | 3 | 21 | 56 | 7 | ∞ | 0 | 28 |
| 6 | 0 | 85 | 8 | 42 | 89 | ∞ | 0 |
| 7 | 18 | 0 | 0 | 0 | 58 | 13 | ∞ |

如果不包括**4-6**这条边，需要把**4-6** 置为∞



43

# 对于左子树

- 包括了**4-6**边的解的费用矩阵:

| j / i | 1 | 2 | 3 | 4 | 5 | 7 | |
|---|---|---|---|---|---|---|---|
| 1 | ∞ | 0 | 83 | 9 | 30 | 50 | |
| 2 | 0 | ∞ | 66 | 37 | 17 | 26 | |
| 3 | 29 | 1 | ∞ | 19 | 0 | 5 | |
| 5 | 0 | 18 | 53 | 4 | ∞ | 25 | (-3) |
| 6 | 0 | 85 | 8 | ∞ | 89 | 0 | |
| 7 | 18 | 0 | 0 | 0 | 58 | ∞ | |

- 总的减去费用为: **96+3 = 99** (新的下限)

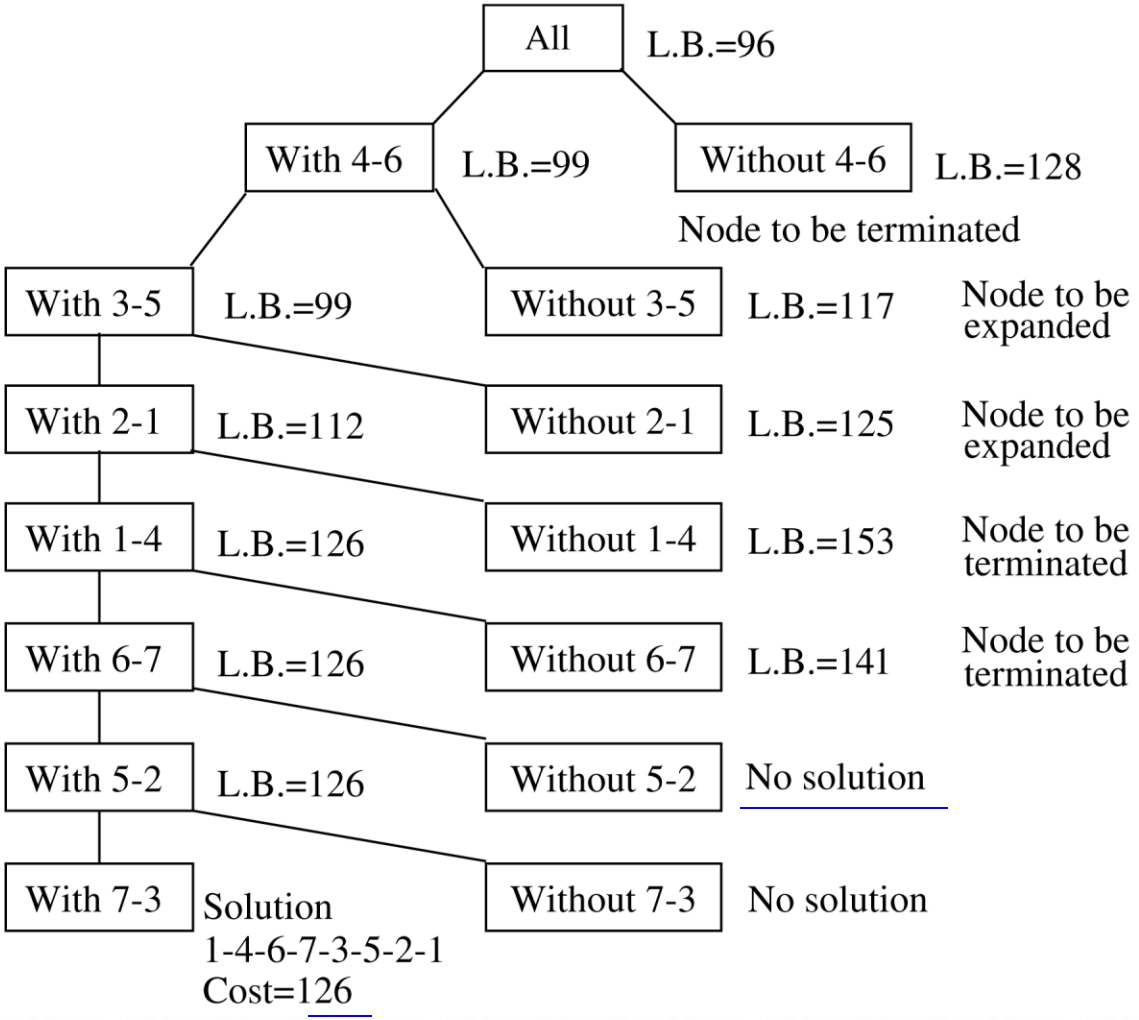

44

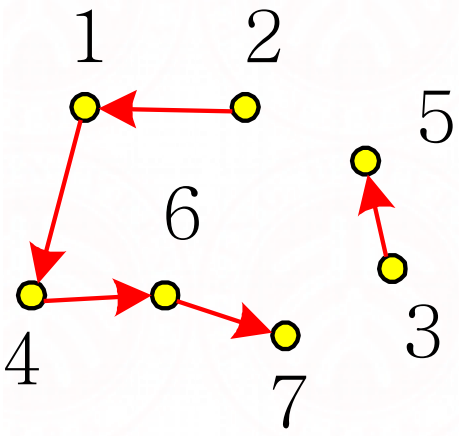
# 上界

- 采用最佳优先搜索（**best-first search**）策略可获得一个可行解（费用为$C$）
- $C$可作为上界，很多分支会因为它们的下限超过上界而被剪枝


45

# TSP最优化问题


All
L.B.=96
With 4-6
L.B.=99
Without 4-6
L.B.=128
Node to be terminated
With 3-5
L.B.=99
Without 3-5
L.B.=117
Node to be expanded
With 2-1
L.B.=112
Without 2-1
L.B.=125
Node to be expanded
With 1-4
L.B.=126
Without 1-4
L.B.=153
Node to be terminated
With 6-7
L.B.=126
Without 6-7
L.B.=141
Node to be terminated
With 5-2
L.B.=126
Without 5-2
No solution
With 7-3
Solution
1-4-6-7-3-5-2-1
Cost=126
Without 7-3
No solution


1
2
5
6
4
7
3



46

# 阻止边

- 一般来说，对于路线$i_1$-$i_2$-…-$i_m$ 和$j_1$-$j_2$-…-$j_n$ ，如果已经确定要把一条从$i_m$到$j_1$ 的边加入，则需要阻止再把从$j_n$ 到$i_1$ 的边加入。
  (通过把从$j_n$ 到 $i_1$ 的费用设置为∞实现)
  (除非$j_n$到$i_1$ 是最后一条待加入的边)


47

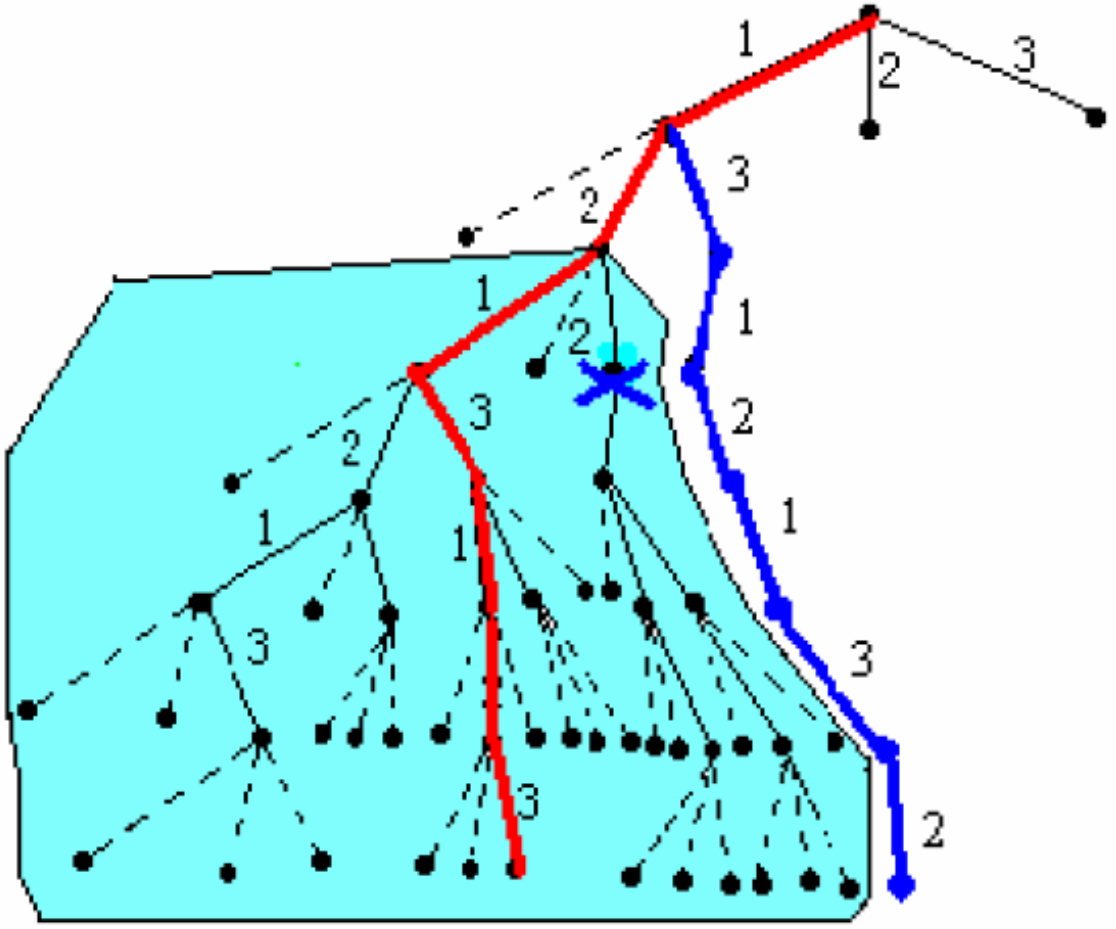
# 思考

- 这种方法和在普通回溯方法有何区别？有何联系？
- 如何评估分支限界法的运行时间？

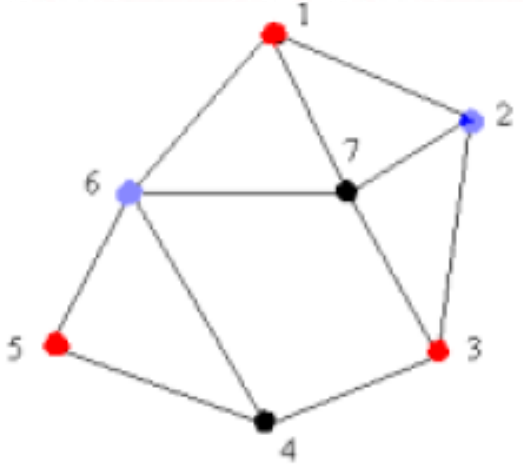
# 例9　图的$m$着色

- 给定无向连通图$G$和$m$种颜色，给图的顶点着色，每个顶点一种颜色，且每条边的两个顶点不同色。给出所有可能的着色方案（如果不存在，则回答“No”）
- 搜索空间
  - 为$m$叉完全树。将颜色编号为$1, 2, \dots, m$。
- 结点$<x_1, x_2, \dots, x_k>$ 表示
  - 顶点1着色$x_1$,顶点2着色$x_2$, …，顶点$k$着色$x_k$
- 约束条件
  - 该顶点邻接表中已着色的顶点没有同色的
- 代价函数：没有（不是最优化问题）
- 时间：$O(nm^n)$

# 回溯算法的图示



<1>
<1,2>
<1,2,1>
<1,2,1,3>
<1,2,1,3,1>
<1,2,1,3,1,2>
<1,2,1,3,1,2,3>



# 提高效率的途径

- 根据对称性
  - 只需搜索1/3的解空间即可。当1和2确定,即<1,2>以后，只有1个解，因此在<1,3>为根的子树中也只有1个解。由于3个子树的对称性，总共有6个解.
- 提前回溯
  - 进一步分析，在取定<1,2>以后，不可以扩张成<1,2,3>，因为可以检查是否有和1,2,3都相邻的顶点。
  - 如果存在（例子中的点7），则没有解。
  - 所以可以从打叉的结点回溯，而不必搜索它的子树。

# 例8　最大团问题

给定无向图$G=<V,E>$，求$G$中的最大团。

- 搜索空间
  - 子集树
  - 结点：$<x_1, x_2, \ldots, x_k>$，表示已检索$k$个顶点
  - 含义：$x_i=1$表示对应的顶点在当前团内
- 约束条件
  - 顶点与当前团内每个顶点都有边相连
- 代价函数（已检索到的极大团的顶点数）
  - 当前团扩张为极大团的顶点数上界
  - $F=c_k+n-k$
- 算法运行时间为$O(n2^n)$
- **子树的搜索顺序？**
  - **度数小在前 vs.度数大在前**

# 例8　最大团问题

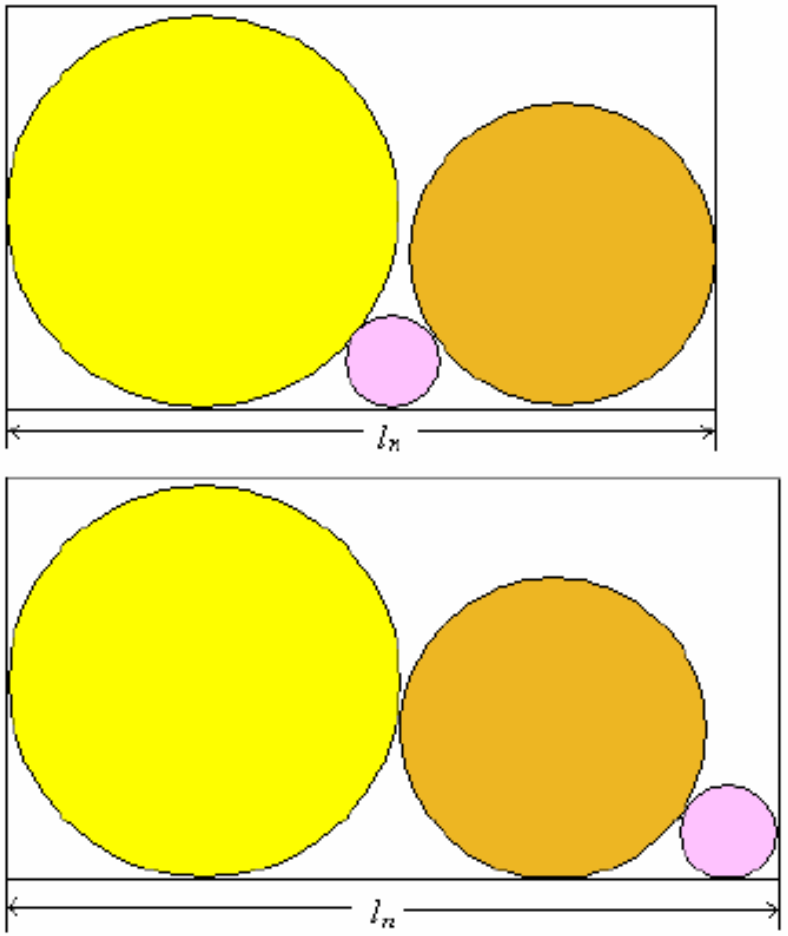
- 搜索空间
  - 组合树$C_m$= <$v_1$,$v_2$,…,$v_m$>
  - 部分解向量$C_k$，当前找到的待扩展的团
  - $X_{k+1} = V - C_k$ , $S_{k+1} = \{v_i | ((v_i, v_k) \in E) \text{ and } (v_i \in X_{k+1})\}$
  - 界$maxC = \max\{|C_k|, maxC\}$
  - 代价函数 $f^k(v) = colour^k(v) : v_i, v_j \in S_k, i < j$

$$colour^k(v_j) = \begin{cases} 1 & j = 0 \\ \min\{c\} & \forall v_i, colour(v_i) = c \rightarrow (v_i, v_j) \notin E \\ \max\{colour^k(v_i) + 1\} & (v_i, v_j) \in E \end{cases}$$

  - 剪枝条件$C_k + colour^k(v) \leq maxC$
  - $S_k$中节点遍历顺序： $colour^k(v)$降序遍历
  - 求$colour^k(v)$的节点顺序？

# 例10　圆排列问题

给定$n$个圆的半径序列$R=\{r_1,r_2,\ldots,r_n\}$，将圆放到矩形框中，各圆与矩形底边相切，求具有最小长度$l_n$的圆的排列顺序。



# 算法设计

- 搜索空间
  - 解为$<i_1, i_2, \ldots, i_n>$：其中$i_1, i_2, \ldots, i_n$为$1, 2, \ldots, n$ 的排列
  - 解空间为排列树
  - 部分解向量$<i_1,i_2,\ldots,i_k>$：表示前$k$个圆已经排好。
- 约束条件
  - 令$B=\{i_1,i_2,\ldots,i_k\}$，下一个圆选择$i_{k+1}$，
  - 约束条件：$i_{k+1}\in\{1,2,\ldots,n\}-B$
- 界：当前得到的最小园排列长度
- 代价函数
  - ？

# 代价函数计算的符号说明

$k$：　算法完成第$k$步，已经选择了第1～第$k$个圆，

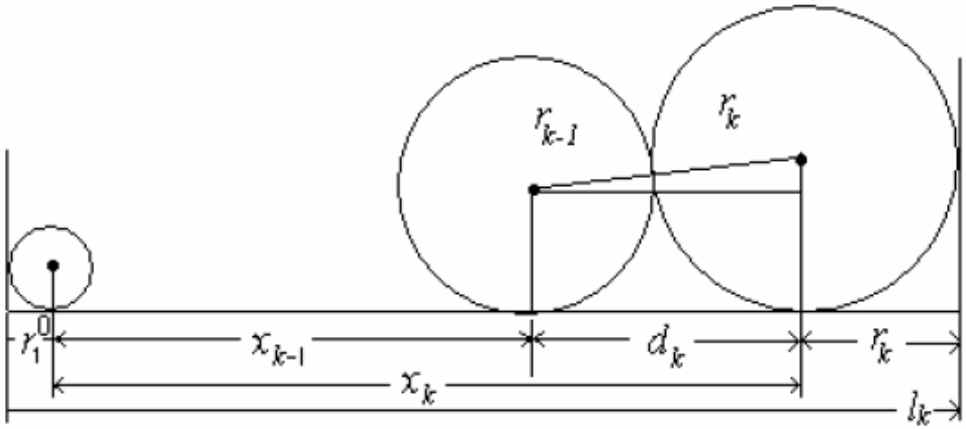
$r_k$：　第$k$个圆的半径

$d_k$：　第$k$-1个圆到第$k$个圆的圆心距离

$x_k$：　第$k$个圆的圆心坐标，规定$x_1=0$,

$l_k$：　第1至第$k$个圆的排列长度

$L_k$：　放好1至第$k$个圆后，对应结点的代价函数值$L_k \leq l_n$

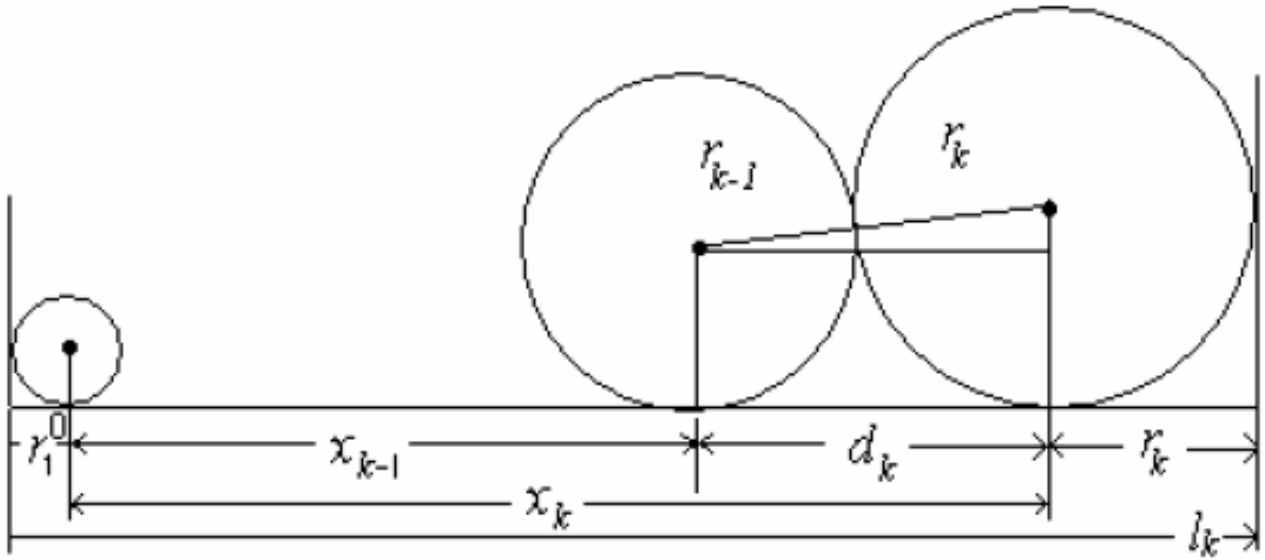


# 相关公式

$$d_k \geq \sqrt{(r_{k-1}+r_k)^2-(r_{k-1}-r_k)^2} = 2\sqrt{r_{k-1}r_k}$$

$$x_k = x_{k-1} + d_k$$

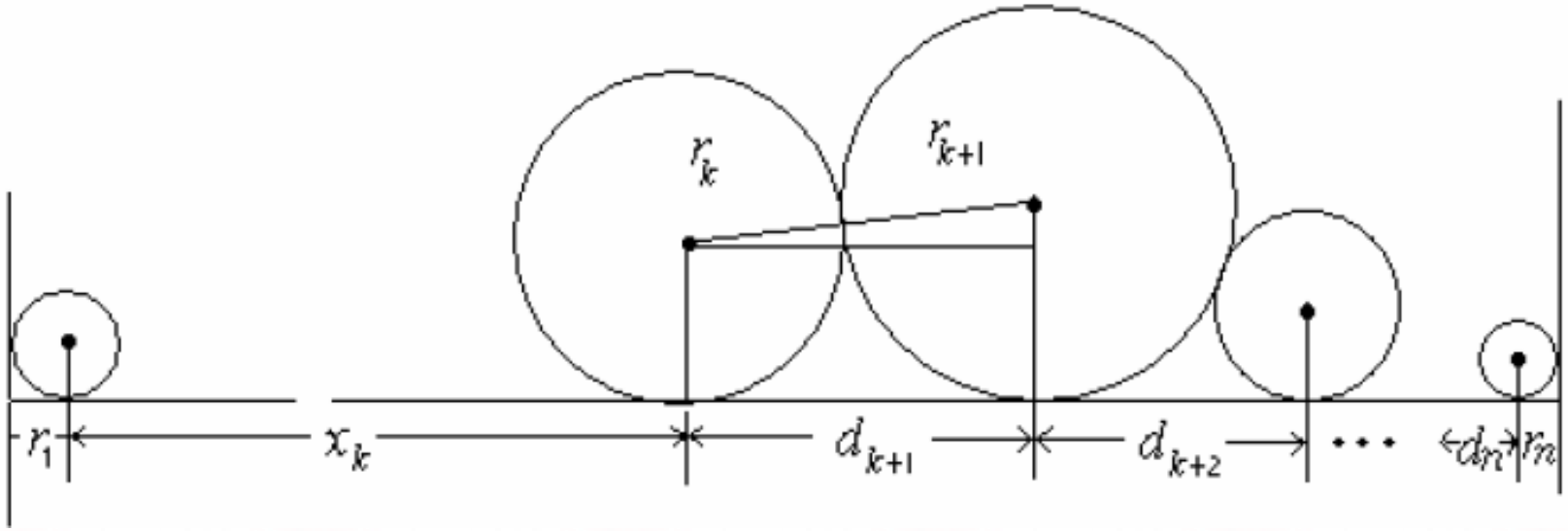
$$l_k = x_k + r_1 + r_k$$



# 相关公式（续）

$$l_n \geq x_k + d_{k+1} + d_{k+2} + \ldots + d_n + r_n + r_1$$

$$= x_k + 2\sqrt{r_k r_{k+1}} + 2\sqrt{r_{k+1} r_{k+2}} + 2\sqrt{r_{n-1} r_n} + r_n + r_1$$



# 代价函数

$$l_n \geq x_k + 2\sqrt{r_k r_{k+1}} + 2\sqrt{r_{k+1} r_{k+2}} + 2\sqrt{r_{n-1} r_n} + r_n + r_1$$

$$\geq x_k + 2(n-k)r + r + r_1$$

$$= x_k + (2n-2k+1)r + r_1 = L_k$$

其中 $r = \min(r_{i_j}, r_k)$

$$i_j \in \{1,2,...,n\} - B$$

$$B = \{i_1, i_2, ..., i_k\}$$

时间：$O(nn!) = O((n+1)!)$

# 实例

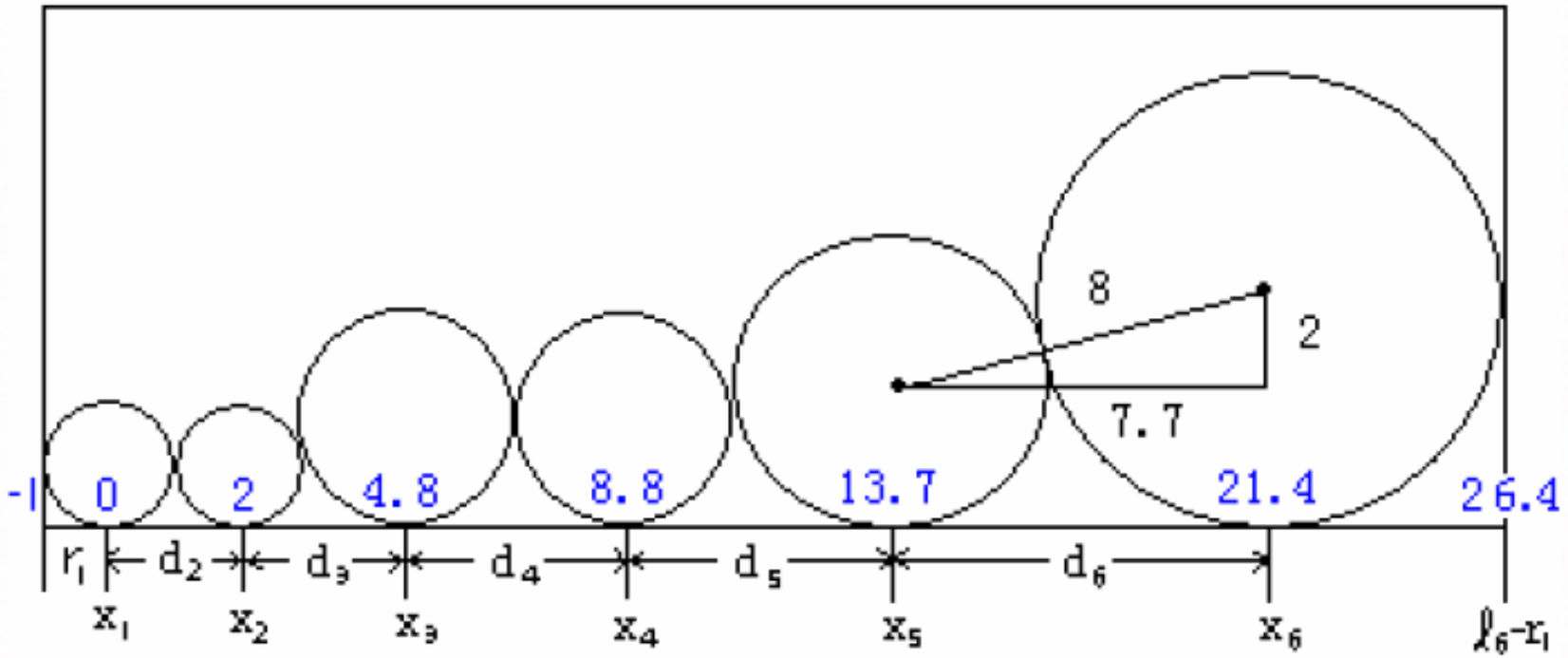
$R$={1,1,2,2,3,5}

取排列<1, 2, 3, 4, 5, 6>,

半径排列为：1, 1, 2, 2, 3, 5，结果见下表和下图

| $k$ | $r_k$ | $d_k$ | $x_k$ | $l_k$ | $L_k$ |
|---|---|---|---|---|---|
| 6 | 5 | 7.7 | 21.4 | 27.4 | 27.4 |
| 5 | 3 | 4.9 | 13.7 | 17.7 | 23.7 |
| 4 | 2 | 4 | 8.8 | 11.8 | 19.8 |
| 3 | 2 | 2.8 | 4.8 | 7.8 | 19.8 |
| 2 | 1 | 2 | 2 | 4 | 12 |
| 1 | 1 | 0 | 0 | 2 | 12 |

- $R$={1, 1, 2, 2, 3, 5}
- 取排列<1, 2, 3, 4, 5, 6>, 半径排列为：1, 1, 2, 2, 3, 5
- 最短长度$L_6$=27.4
- 取排列<1, 3, 5, 6, 4, 2>, 半径排列为：1, 2, 3, 5, 2, 1
- 最短长度$L_6$=26.5



# 分支限界法与回溯法对比

- 求解目标不同：
  - 回溯法：找一个可行解、找所有可行解、找最优解
  - 分支限界法：找最优解、找一个可行解
- 搜索方式不同：
  - 回溯法：深度优先（剪枝：约束条件、代价函数）
  - 分支限界法：广度优先（剪枝：上/下界、约束条件）
- 主要特点：
  - 回溯法：空间效率高
  - 分支限界法：往往更“快”
- See: http://en.wikipedia.org/wiki/Branch_and_bound

# 小结

- **适应于求解组合搜索问题**（含组合优化问题）
- **求解条件：**满足多米诺性质
- **解的表示：**解向量，求解是不断扩充解向量的过程
- **回溯条件：**
  - 搜索问题——约束条件
  - 优化问题——约束条件 + 代价函数
- **算法复杂性：**
  - 遍历搜索树的时间，最坏情况为指数
  - 空间代价小
- **平均时间复杂性的估计：**Monte Carlo方法
- **降低时间复杂性的主要途径：**
  - 利用对称性裁减子树
  - 划分成子问题
- **分支策略**（深度优先、宽度优先、宽深结合、优先函数）