ACM程序设计培训

计算机与信息学院 刘必雄

递归与回溯专题

内 容 提 要

R E C U R S I O H & B A C K T R A C K T O P I C

1、递归

2、回溯法

3、分支限界

R E C U R S I O H & B A C K T R A C K T O P I C

1、递归

递归

一个函数在它的函数体内调用它自身称为递归（Recursion）调用 。即一个过程或函数在其定义或说明中直接或间接调用自身的一种方法 。

- 递归策略只需少量的程序就可描述出解题过程所需要的多次重复计算，大大地减少了程序的代码量。
- 递归的能力在于用有限的语句来定义对象的无限集合。
- 用递归思想写出的程序往往十分简洁易懂。

1、递归

使用递归的注意事项

- 递归就是在过程或函数里调用自身 。
- 在使用递增归策略时，必须有一个明确的递归结束条件，称为递归出口 。

```
int r(int a)
{
  b=r(a-1);
  return b;
}
```

这个函数是一个递归函数，但是运行该函数将无休止地调用其自身，这显然是不正确的。

2、回溯法

排队购票问题

- 问题描述

一场球赛开始前，售票工作正在紧张的进行中。每张球票为50元，现有30个人排队等待购票，其中有20 个人手持50元的钞票，另外10个人手持100元的钞票。假设开始售票时售票处没有零钱，求出这30个人排队购票，使售票处不至出现找不开钱的局面的不同排队种数。（约定：拿同样面值钞票的人对换位置后为同一种排队。）

输入	输出
15 12	4345965

2、回溯法

排队购票问题

- 问题分析

  - n=0意味着排队购票的所有人手中拿的都是50元的钱币，注意到拿同样面值钞票的人对换位置后为同一种排队，那么这m个人的排队总数为1，即f(m,0)=1。
  - 当m<n时，即排队购票的人中持50元的人数小于持100元的钞票，即使把m张50元的钞票都找出去，仍会出现找不开钱的局面，所以这时排队总数为0，即f(m,n)=0。

2、回溯法

排队购票问题

- 问题分析

  - 思考m+n个人排队购票，第m+n个人站在第m+n-1个人的后面，则第m+n个人的排队方式可由下列两种情况获得：
  - 第m+n个人手持100元的钞票，则在他之前的m+n-1个人中有m个人手持50元的钞票，有n-1个人手持100元的钞票，此种情况共有f(m,n-1)。
  - 第m+n个人手持50元的钞票，则在他之前的m+n-1个人中有m-1个人手持50元的钞票，有n个人手持100元的钞票，此种情况共有f(m-1,n)。

2、回溯法

排队购票问题

● 问题描述

一场球赛开始前，售票工作正在紧张的进行中。每张球票为50元，现有30个人排队等待购票，其中有20 个人手持50元的钞票，另外10个人手持100元的钞票。假设开始售票时售票处没有零钱，求出这30个人排队购票，使售票处不至出现找不开钱的局面的不同排队种数。（约定：拿同样面值钞票的人对换位置后为同一种排队。）

递推关系	初始条件
$f(m,n)=f(m,n-1)+f(m-1,n)$	当$m<n$时，$f(m,n)=0$ 当$n=0$时，$f(m,n)=1$

2、回溯法

概述

- 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结点时，**先判断该结点是否包含问题的解**。若肯定不包含，则跳过对以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，继续按深度优先搜索策略搜索。
- 回溯法求问题的**所有解**时，要回溯到根，且根结点的所有子树都被搜索遍才结束。
- 回溯法求问题的**一个解**时，只要搜索到问题的一个解就结束。

2、回溯法

问题的解空间

问题的解空间是由复杂问题的所有的可能解构成的，是进行穷举的搜索空间。

确定正确的解空间很重要，如果没有确定正确的解空间就开始搜索，可能会增加很多重复解，或者根本就搜索不到正确的解。

【例】桌子上有6根火柴棒，要求以这6根火柴棒为边搭建4个等边三角形。在二维搜索空间无解，而在三维搜索空间有解。

2、回溯法

问题的解空间

有n个物品的0/1背包问题

- 可能解的表示方式
  - 可能解由一个**不等长向量**组成，当物品i(1≤i≤n)装入背包时，解向量中包含分量i，否则，解向量中不包含分量i。

{ ( ), (1), (2), (3), (1, 2), (1, 3), (2, 3), (1, 2, 3) }

  - 可能解由一个**等长向量**$\{x_1, x_2, \ldots, x_n\}$组成，其中$x_i=1(1\le i\le n)$表示物品i装入背包，$x_i=0$表示物品i没有装入背包

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

2、回溯法

问题的解空间

● 解向量

对于具有n个输入问题，可以将其可能解表示为满足某个约束条件的**等长向量**$X=(x_1, x_2, \ldots, x_n)$，其中分量$x_i$ $(1\leq i\leq n)$的取值范围是某个有限集合$S_i=\{a_{i1}, a_{i2}, \ldots, a_{ir_i}\}$，所有可能的解向量构成了**问题的解空间**。

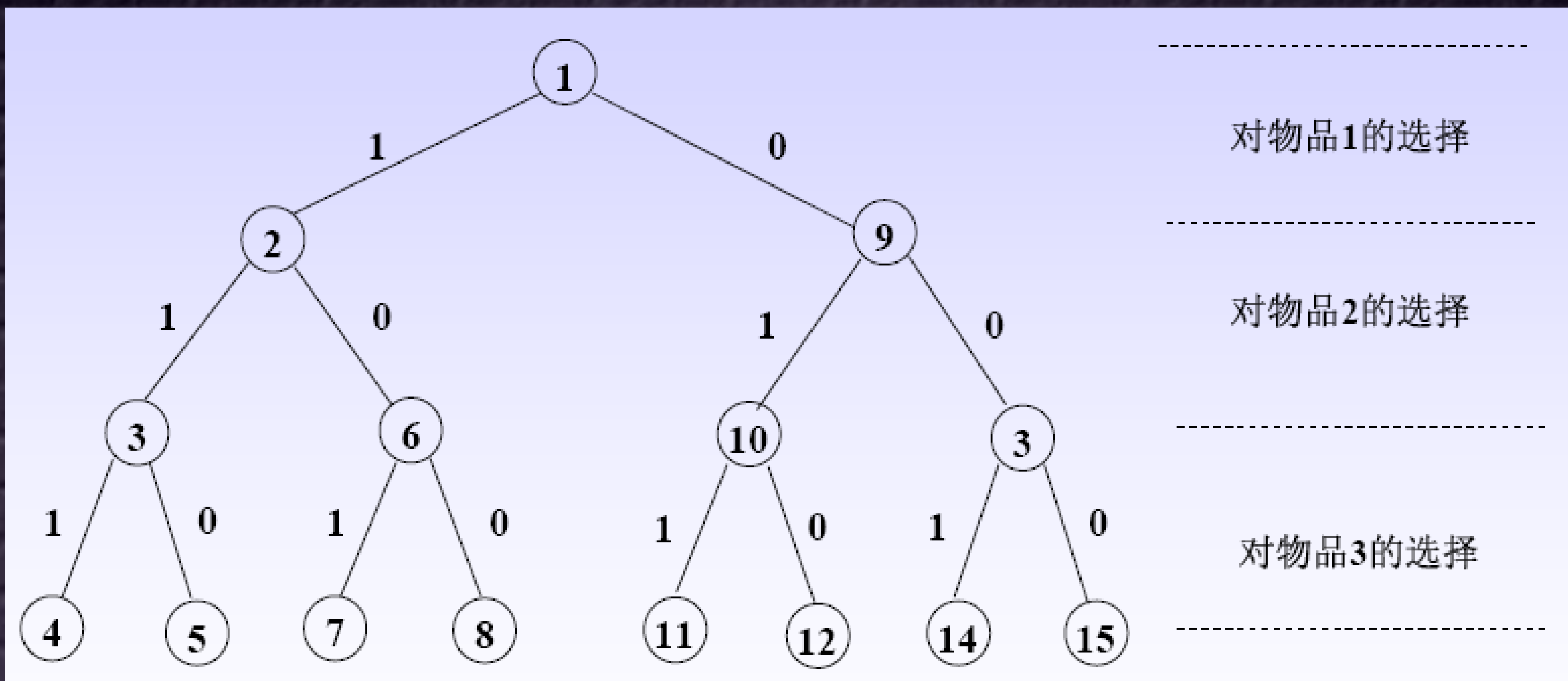
● 解空间树

问题的解空间一般用**解空间树**的方式组织，树的根结点位于第1层，表示搜索的**初始状态**，第2层的结点表示对解向量的第一个分量做出选择后到达的状态，第1层到第2层的边上标出对第一个分量选择的结果，依此类推，从树的根结点到叶子结点的路径就构成了解空间的一个可能解。

2、回溯法

问题的解空间

对于n=3的0/1背包问题的解空间树

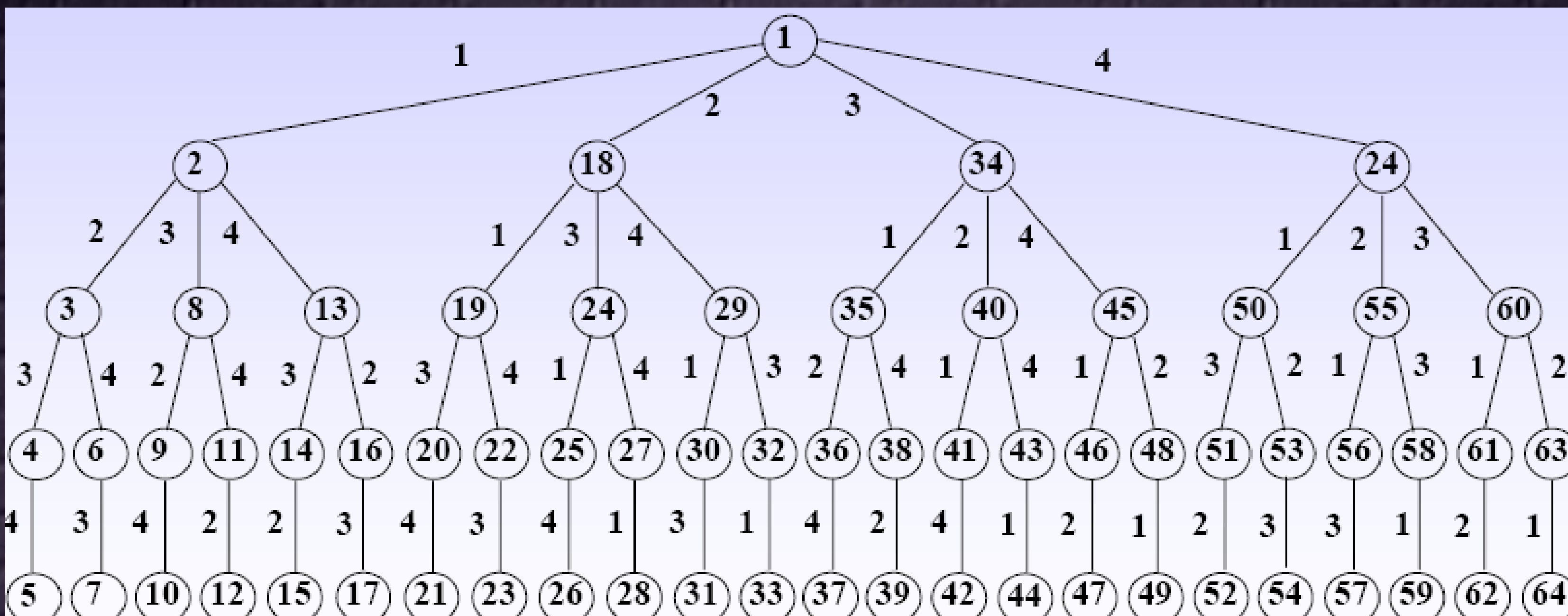


树中的8个叶子结点分别代表该问题的**8个可能解**。

2、回溯法

问题的解空间

对于n=4的全排列问题的解空间树



树中的24个叶子结点分别代表该问题的**24个可能解**，例如结点5代表一个可能解，路径为1→2→3→4。

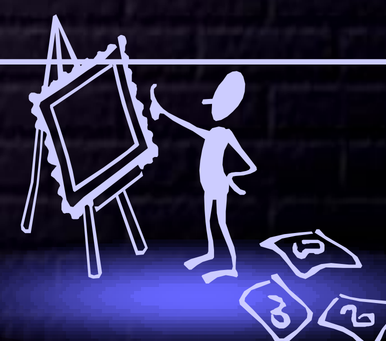
2、回溯法

基本思想

- **基本思想**

回溯法从根结点出发，按照**深度优先策略**遍历解空间树，搜索满足约束条件的解。在搜索至树中任一结点时，先判断该结点对应的部分解是否满足**约束条件**，或者是否超出**目标函数**的界，也就是判断该结点是否包含问题的（最优）解，如果肯定不包含，则跳过对以该结点为根的子树的搜索，即所谓**剪枝**（Pruning）；否则，进入以该结点为根的子树，继续按照深度优先策略搜索。

回溯法是**带优化**的穷举法。



2、回溯法

基本思想

● 构造过程

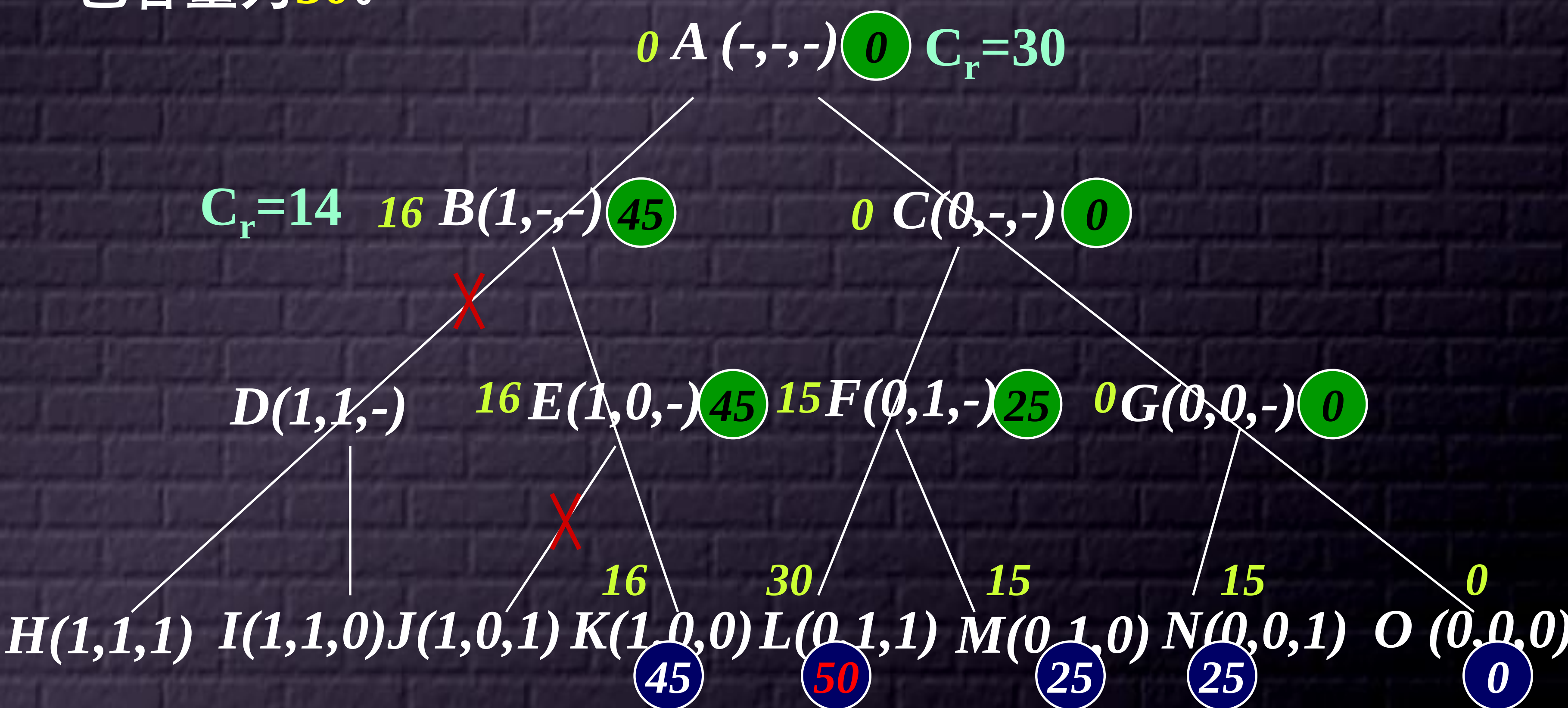
- 在回溯法中，并不是先构造出整棵状态空间树，再进行搜索，而是在搜索过程中逐步构造出状态空间树，即边搜索，边构造。
- 从开始结点（根结点）出发，以深度优先的方式搜索整个解空间。该这个开始结点是活结点，同时也成为当前的扩展结点。
- 在当前的扩展结点处，搜索向纵深方向移至一个新结点。这个新结点就成为一个新的活结点，并成为当前扩展结点。
- 如果在当前的扩展结点处不能再向纵深方向移动，则当前扩展结点就成为死结点。此时，应往回移动（回溯）至最近的一个活结点处，并使这个活结点成为当前的扩展结点。
- 回溯法即以这种工作方式递归地在解空间中搜索，直至找到所要求的解或解空间中已没有活结点时为止。

2、回溯法

基本思想

对于n=3的0/1背包问题的搜索示例

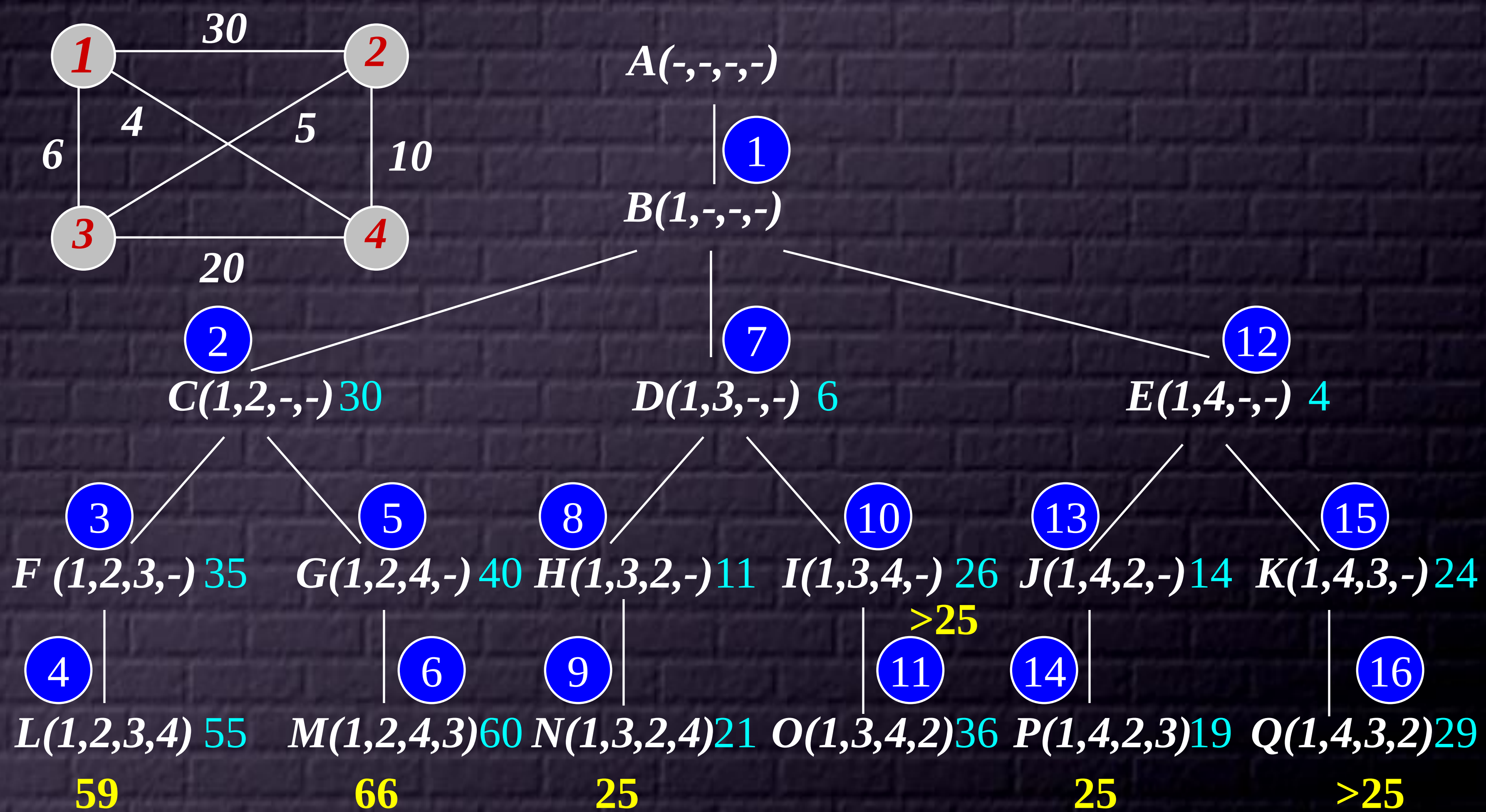
三个物品的重量w为{16, 15, 15}，价值p为{45, 25, 25}，背包容量为30。



2、回溯法

基本思想

对于n=4的TSP问题的搜索示例



2、回溯法

基本思想

- 剪枝函数

回溯法的搜索过程涉及的结点，只是整个解空间树的一部分，在搜索过程中，通常采用两种策略避免无效搜索

- 用约束条件剪去得不到可行解的子树；
- 用目标函数剪去得不到最优解的子树。

注意

问题的解空间树是虚拟的，并不需要在算法运行时构造一棵真正的树结构，只需要存储从根结点到当前结点的路径。

2、回溯法

求解过程

● 解向量

问题的**解向量**$X=(x_1, x_2, \ldots, x_n)$的分量$x_i$ $(1 \le i \le n)$都属于一个有限集合$S_i=\{a_{i1}, a_{i2}, \ldots, a_{iri}\}$，显然，回溯法可以按某种顺序依次考察**笛卡尔积**$S_1 \times S_2 \times \ldots \times S_n$中的元素。

● 求解过程

首先置解向量X为空，然后选择S_1的**第一个元素**作为解向量X的第1个分量，即$x_1=a_{11}$；如果$X=(x_1)$是问题的**部分解**，则**继续扩展**解向量X，选择S_2的第一个元素作为解向量X的第2个分量；否则，选择S_1的下一个元素作为解向量X的第1个分量，即$x_1=a_{12}$。依此类推。

2、回溯法

求解过程

如果$X=(x_1, x_2, \ldots, x_i)$是问题的**部分解**，则选择$S_{i+1}$的第1个元素作为解向量X的**第i+1个分量**时，有以下3种情况

- $X=(x_1, x_2, \ldots, x_{i+1})$是问题的**最终解**，则输出这个解。如果问题只希望得到一个解，则**结束搜索**，否则**继续搜索**其他解。
- $X=(x_1, x_2, \ldots, x_{i+1})$是问题的**部分解**，则继续构造解向量的**下一个分量**。
- $X=(x_1, x_2, \ldots, x_{i+1})$既不是问题的部分解也不是问题的最终解。

2、回溯法

求解过程

$X=(x_1, x_2, \ldots, x_{i+1})$既不是问题的部分解也不是问题的最终解，则存在下面两种情况

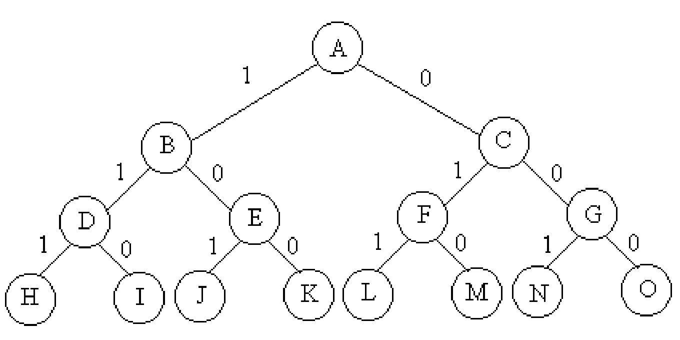
- 如果$x_{i+1}=a_{i+1,k}$不是集合S_{i+1}的最后一个元素，则令$x_{i+1}=a_{i+1,k+1}$，即选择S_{i+1}的下一个元素作为解向量X的第i+1个分量；
- 如果 $x_{i+1}=a_{i+1,k}$ 是集合 S_{i+1} 的最后一个元素，就回溯到$X=(x_1, x_2, \ldots, x_i)$，选择$S_i$的下一个元素作为解向量X的第i个分量，假设$x_i=a_{ik}$，如果$a_{ik}$不是集合$S_i$的最后一个元素，则令$x_i=a_{i,k+1}$；否则，就继续回溯到$X=(x_1, x_2, \ldots, x_{i-1})$。

2、回溯法

子集树和排列树

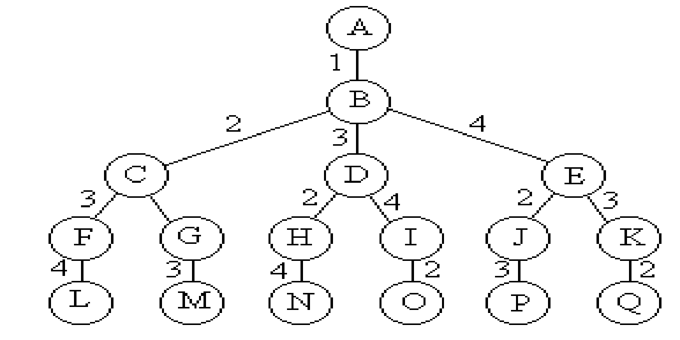
●子集树

当所给问题是从n个元素的集合S中找出S满足某种性质的子集时，相应的解空间称为子集树。



●排列树

当所给问题是确定n个元素满足某种性质的排列时，相应的解空间树称为排列树。



2、回溯法

子集树和排列树

子集树算法框架

```
void BackTrack(int t)  // t表示递归深度
{
   if (t>n) //当搜索到叶结点，输出可行解
      OutPut(x);
   else
      for (int i=0;i<=1;i++){//从下界 to 上界
         x[t]=i;
         if (Constraint(t)&&Bound(t))
            BackTrack(t+1);
      }
}
```

遍历子集树需$O(2^n)$计算时间

2、回溯法

子集树和排列树

排列树算法框架

```
void BackTrack (int t)
{
   if (t>n) OutPut(x);
   else
      for (int i=t;i<=n;i++){
         Swap(x[t],x[i]);
         if (Constraint(t)&&Bound(t))
            BackTrack(t+1);
         Swap(x[t],x[i]);
      }
}
```

遍历排列树需要O(n!)计算时间

2、回溯法

0-1背包问题

● 问题描述

给定n 种物品和一个背包。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	输出
5 10 6 3 5 4 6 2 2 6 5 4	15 1 1 0 0 1

2、回溯法

旅行商问题

● 问题描述

旅行售货员问题的提法是：某售货员要到若干城市去推销商品，已知各城市之间的路程（或旅费）。他要选定一条从驻地出发，经过每个城市一次，最后回到驻地的路线，使总的路程（或总旅费）最小。

输入	输出
7 8 1 4 5 4 2 8 2 6 3 6 5 1 5 3 3 3 7 2 7 1 9 1 5 10	31 1 4 2 6 5 3 7

2、回溯法

装载问题

● 问题描述

有一批共n个集装箱要装上2艘载重量分别为c1和c2的轮船，其中集装箱i的重量为w_i，且满足集装箱总重量小于等于c1和c2载重量。要求确定是否有一个合理的装载方案可将这些集装箱装上这2艘轮船。

输入	输出
5 15 20 25 45 50 90 70	1 0 1 0 1 0 1 0 1 0

2、回溯法

批处理作业调度

- 问题描述

给定n个作业的集合$\{J_1,J_2,\ldots,J_n\}$，每个作业必须先由机器1处理，然后由机器2处理。作业J_i需要机器j的处理时间为t_{ji}。对于一个确定的作业调度，设F_{ji}是作业i在机器j上完成处理的时间。所有作业在机器2上完成处理的时间和称为该作业调度的完成时间和。批处理作业调度问题要求对于给定的n个作业，制定最佳作业调度方案，使其完成时间和达到最小。

输入	输出
3 2 1 3 1 2 3	18 1 3 2

2、回溯法

批处理作业调度

● 问题分析

【例】三个作业{1, 2, 3}，这三个作业在机器1上所需的处理时间为(2, 3, 2)，在机器2上所需的处理时间为(1, 1, 3)，则最佳调度方案是(1, 3, 2)，其完成时间和为18。

(a) 调度方案(1, 2, 3)，完成时间和为19

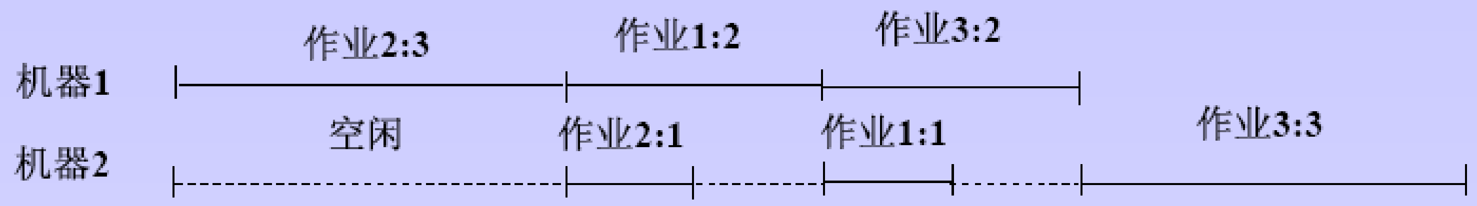
(b) 调度方案(1, 3, 2)，完成时间和为18

2、回溯法

批处理作业调度

- 问题分析

【例】三个作业{1, 2, 3}，这三个作业在机器1上所需的处理时间为(2, 3, 2)，在机器2上所需的处理时间为(1, 1, 3)，则最佳调度方案是(1, 3, 2)，其完成时间和为18。



(c) 调度方案(2, 1, 3)，完成时间和为20

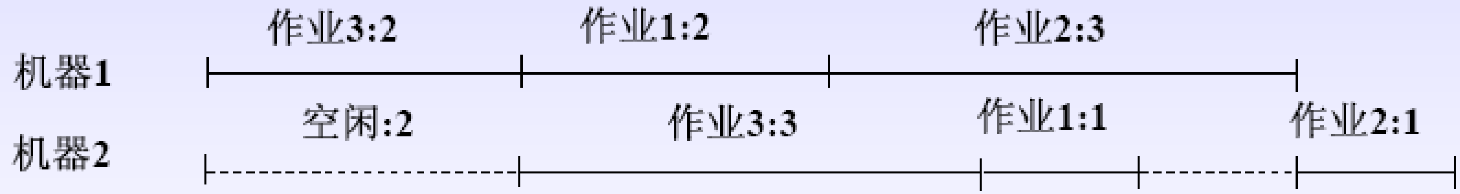
(d) 调度方案(2, 3, 1)，完成时间和为21

2、回溯法

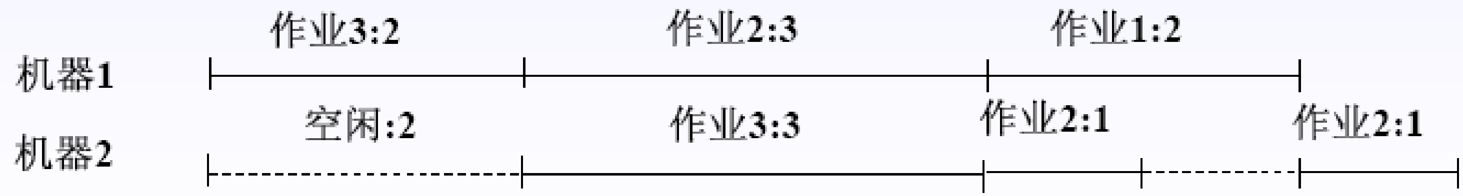
批处理作业调度

● 问题分析

【例】三个作业{1, 2, 3}，这三个作业在机器1上所需的处理时间为(2, 3, 2)，在机器2上所需的处理时间为(1, 1, 3)，则最佳调度方案是(1, 3, 2)，其完成时间和为18。



(e) 调度方案(3, 1, 2)，完成时间和为19



(f) 调度方案(3, 2, 1)，完成时间和为19

3、分支限界

● 基本思想

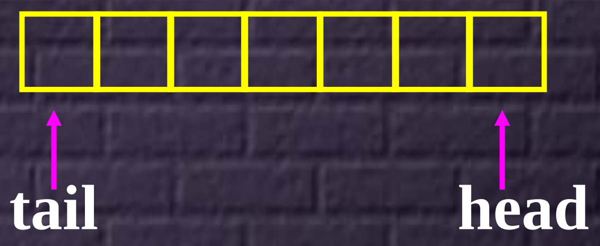
- 分支限界法在问题的解空间树中，按广度优先策略、或以最小耗费（最大效益）优先的方式搜索解空间树。
- 在分支限界法中，每一个活结点只有一次机会成为扩展结点。活结点一旦成为扩展结点，就一次性产生其所有儿子结点。在这些儿子结点中，导致不可行解或导致非最优解的儿子结点被舍弃，其余儿子结点被加入活结点表中。
- 从活结点表中取下一结点成为当前扩展结点，并重复上述结点扩展过程。这个过程一直持续到找到所需的解或活结点表为空时为止。

3、分支限界

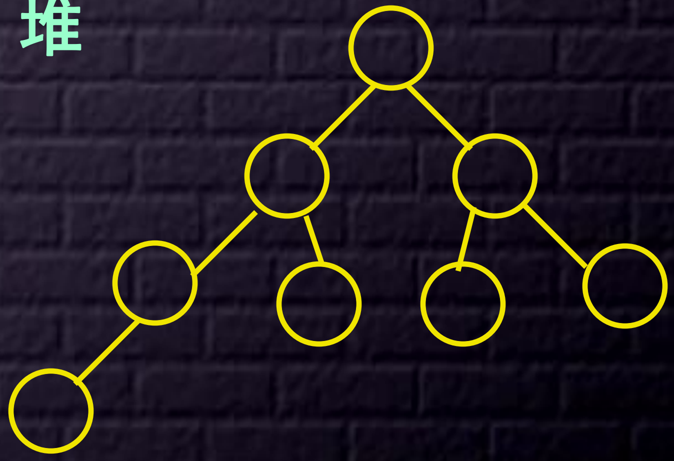
常见的两种分支限界法

队列式分支限界法	优先队列式分支限界法
按照队列先进先出FIFO原则选取下一个节点为扩展节点。	按照优先队列中规定的优先级选取优先级最高的节点成为当前扩展节点。

队列



堆



3、分支限界

队列式分支限界法

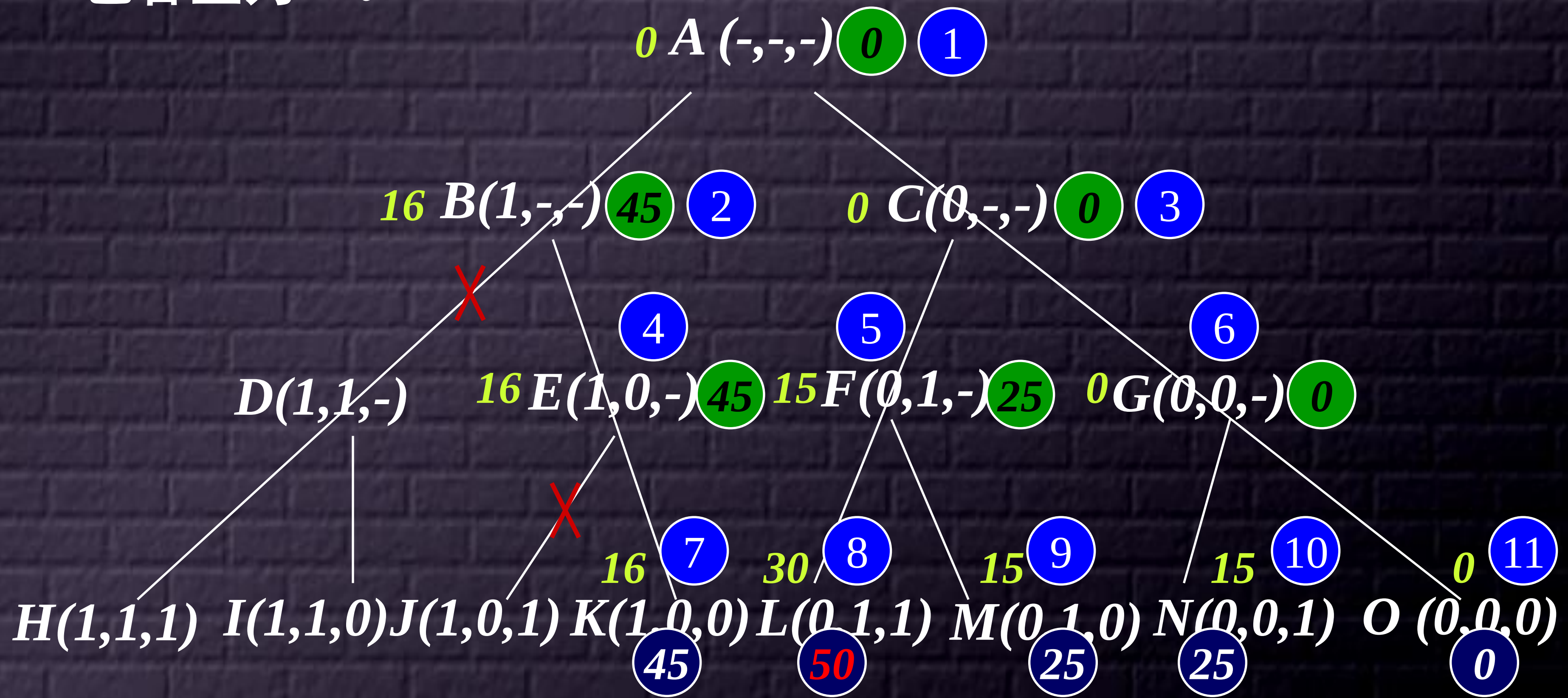
- **搜索策略**
  - 首先，确定一个合理的限界函数，并根据限界函数确定目标函数的界[*down*, *up*]。
  - 最初根结点是唯一的活结点，根结点入队。然后，从活结点队列中取出根结点后，作为当前扩展结点。
  - 对当前扩展结点，按照广度优先策略从左到右地产生它的所有儿子，然后分别估算儿子结点的目标函数值，把所有满足目标函数的儿子加入活结点队列中。
  - 再从活结点表中取出队首结点（队中最先进来的结点）为当前扩展结点，……，直到找到一个解或活结点队列为空为止。

3、分支限界

队列式分支限界法

对于n=3的0/1背包问题的队列式分支搜索示例

三个物品的重量w为{16, 15, 15}，价值p为{45, 25, 25}，背包容量为30。



3、分支限界

队列式分支限界法

对于n=3的0/1背包问题的队列式分支搜索示例

队列式分支法的搜索过程是对子集树进行盲目搜索，虽然不能将搜索算法改进为多项式复杂度，但若在在算法中加入了“限界”技巧，还是能降低算法的复杂度。

- 限界函数

- 设r是当前剩余物品价值总和、cp是当前价值、bestp是当前已获得的最优价值。 $cp + r \leq bestp$ 剪枝
- 将剩余物品依其单位重量价值排序，做剩余物品的背包问题来作为右子树中的最优解，即右子树中解的上界，设上界为bound。 $cp + bound \leq bestp$ 剪枝

3、分支限界

队列式分支限界法

对于n=4的0/1背包问题的队列式分支限界搜索示例

四个物品的重量w为{4, 7, 5,3}，价值p为{40,42, 25, 12}，背包容量为10。

将给定物品按单位重量价值从大到小排序

物品	重量(w)	价值(p)	价值/重量(p/w)
1	4	40	10
2	7	42	6
3	5	25	5
4	3	12	4

3、分支限界

队列式分支限界法

对于n=4的0/1背包问题的队列式分支限界搜索示例

四个物品的重量w为{4, 7, 5,3}，价值p为{40,42, 25, 12}，背包容量为10。

- 限界函数

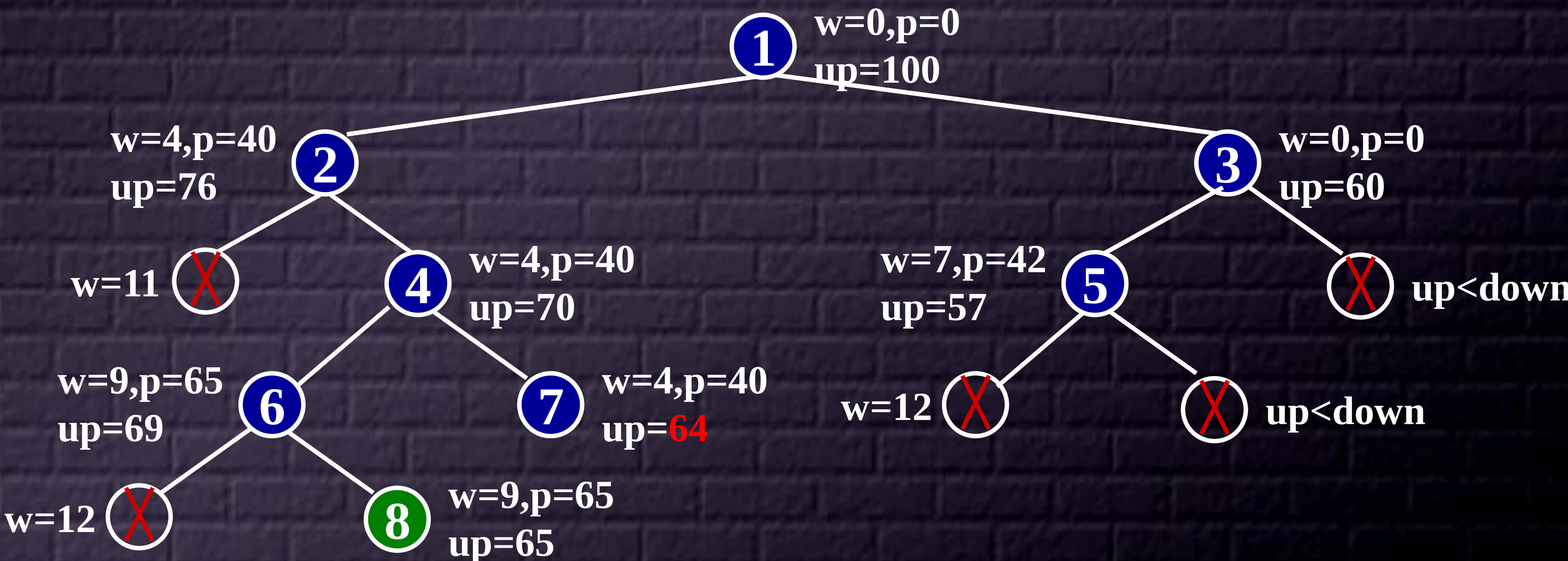
  - 应用贪心法求得近似解为(0, 1, 0, 1)，获得的价值为54，作为0/1背包问题的下界down。
  - 考虑最好情况，背包中装入的全部是第1个物品且可以将背包装满，则可以得到一个非常简单的上界的计算方法：up=w×(p1/w1)=10×10=100。

3、分支限界

队列式分支限界法

对于n=4的0/1背包问题的队列式分支限界搜索示例

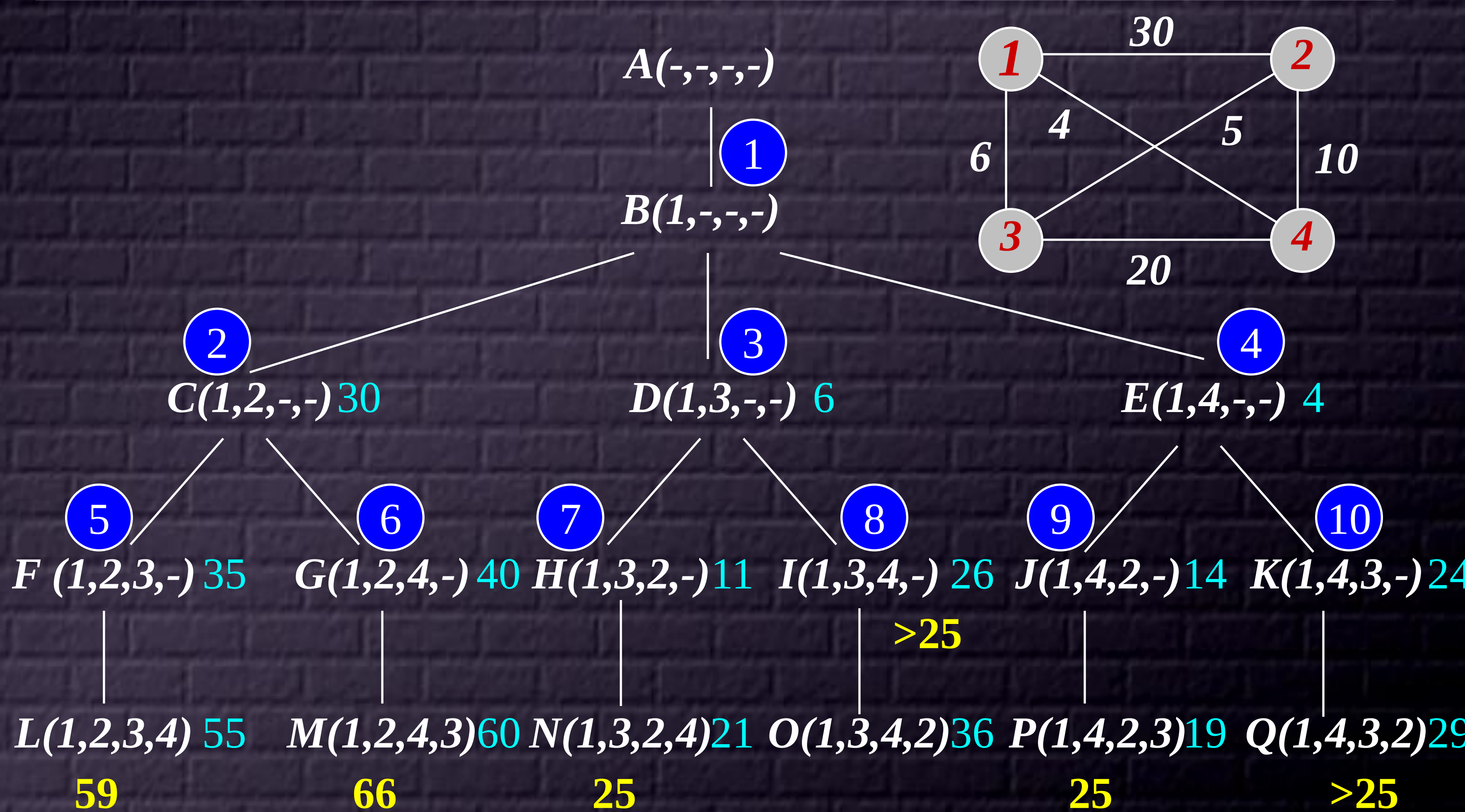
四个物品的重量w为{4, 7, 5,3}，价值p为{40,42, 25, 12}，背包容量为10。



3、分支限界

队列式分支限界法

对于n=4的TSP问题的队列式分支限界搜索示例



3、分支限界

队列式分支限界法

对于n=5的TSP问题的队列式分支限界搜索示例

$$C=\begin{pmatrix} \infty & 3 & 1 & 5 & 8 \\ 3 & \infty & 6 & 7 & 9 \\ 1 & 6 & \infty & 4 & 2 \\ 5 & 7 & 4 & \infty & 3 \\ 8 & 9 & 2 & 3 & \infty \end{pmatrix}$$

- 限界函数

- 应用贪心法求得近似解为1→3→5→4→2→1，其路径长度为1+2+3+7+3=16，作为TSP问题的up。
- 如果把矩阵中每一行最小的两个元素相加再除以2，如果图中所有的代价都是整数，再对这个结果向上取整，就得到了一个合理的down。

3、分支限界

优先队列式分支限界法

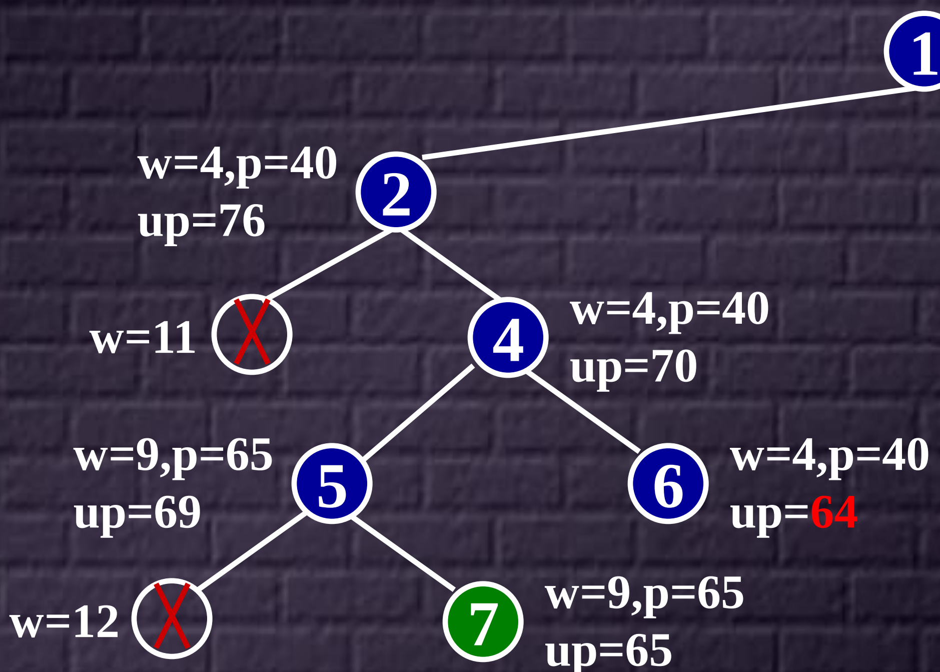
- **搜索策略**
  - 对每一活结点计算一个优先级（某些信息的函数值）。
  - 根据优先级从当前活结点表中选择一个优先级最高的结点作为扩展结点，使搜索朝着解空间树上有最优解的分支推进，以便尽快地找出一个最优解。
  - 再从活结点表中下一个优先级别最高的结点为当前扩展结点，……，直到找到一个解或活结点队列为空为止。

3、分支限界

优先队列式分支限界法

对于n=4的0/1背包问题的优先队列式分支限界搜索示例

四个物品的重量w为{4, 7, 5,3}，价值p为{40,42, 25, 12}，背包容量为10。



3、分支限界

优先队列式分支限界法

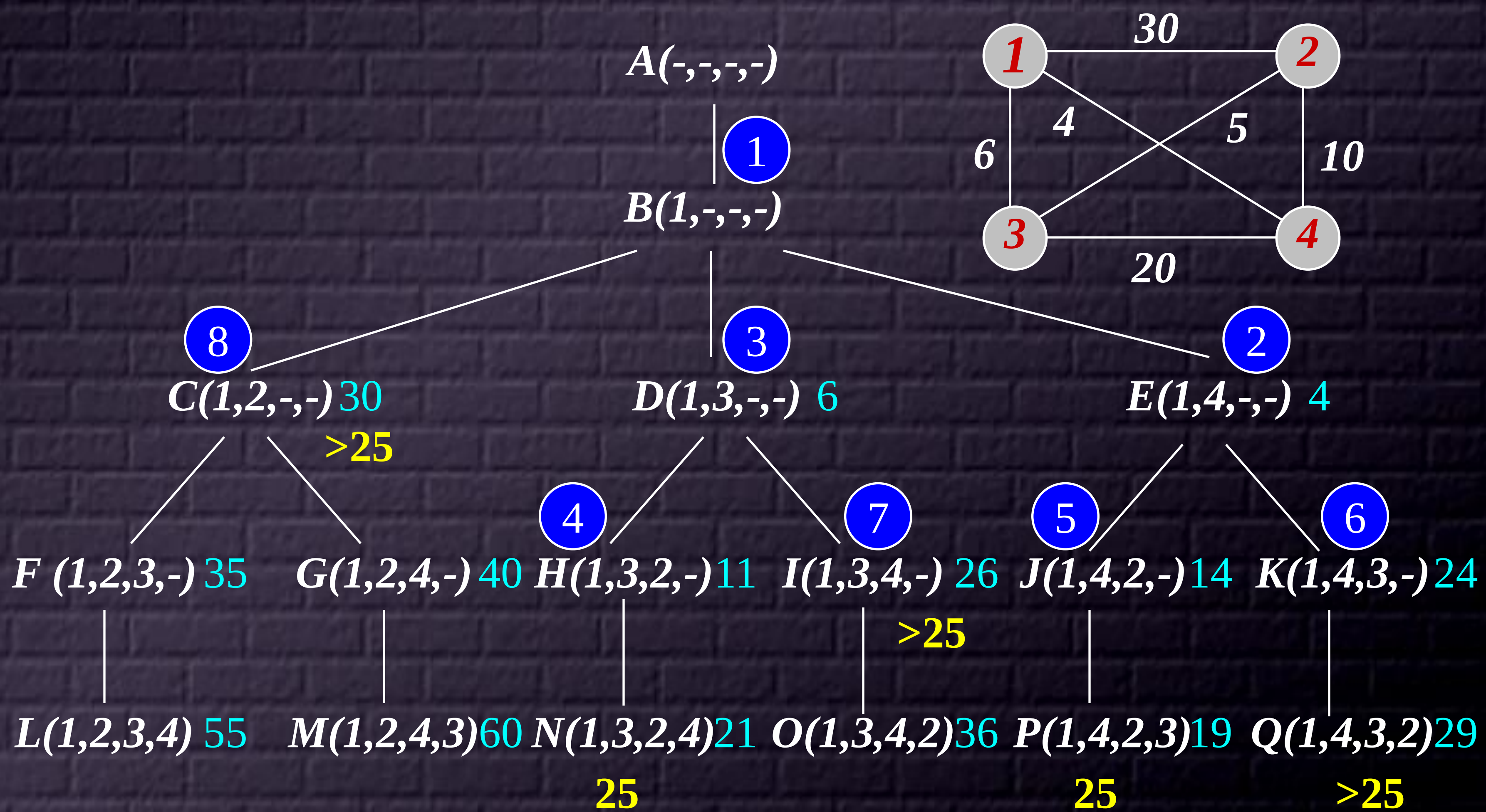
分支限界算法框架

1. 根据限界函数确定目标函数的界[down, up]；
2. 将待处理结点表PT初始化为空；
3. 对根结点的每个孩子结点x执行下列操作
   3.1 估算结点x的目标函数值value;
   3.2 若(value>=down)，则将结点x加入表PT中；
4. 循环直到某个叶子结点的目标函数值在表PT中最大
   4.1 i=表PT中值最大的结点；
   4.2 对结点i的每个孩子结点x执行下列操作
      4.2.1 估算结点x的目标函数值value;
      4.2.2 若(value>=down)，则将结点x加入表PT中；
      4.2.3 若(结点x是叶子结点且结点x的value值在表PT中最大)，则将结点x对应的解输出，算法结束；
      4.2.4 若(结点x是叶子结点但结点x的value值在表PT中不是最大)，则令down=value，并且将表PT中所有小于value的结点删除；

3、分支限界

优先队列式分支限界法

对于n=4的TSP问题的优先队列式分支限界搜索示例



3、分支限界

0-1背包问题

● 问题描述

给定n 种物品和一个背包。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	输出
5 10 6 3 5 4 6 2 2 6 5 4	15 1 1 0 0 1

3、分支限界

旅行商问题

- 问题描述

旅行售货员问题的提法是：某售货员要到若干城市去推销商品，已知各城市之间的路程（或旅费）。他要选定一条从驻地出发，经过每个城市一次，最后回到驻地的路线，使**总的路程**（或总旅费）**最小**。

输入	输出
7 8 1 4 5 4 2 8 2 6 3 6 5 1 5 3 3 3 7 2 7 1 9 1 5 10	31 1 4 2 6 5 3 7

3、分支限界

装载问题

问题描述

有一批共n个集装箱要装上2艘载重量分别为c1和c2的轮船，其中集装箱i的重量为w_i，且满足集装箱总重量小于等于c1和c2载重量。要求确定是否有一个合理的装载方案可将这些集装箱装上这2艘轮船。

输入	输出
5 15 20 25 45 50 90 70	1 0 1 0 1 0 1 0 1 0

5、分支限界法

批处理作业调度

● 问题描述

给定n个作业的集合$\{J_1,J_2,\ldots,J_n\}$，每个作业必须先由机器1处理，然后由机器2处理。作业J_i需要机器j的处理时间为t_{ji}。对于一个确定的作业调度，设F_{ji}是作业i在机器j上完成处理的时间。所有作业在机器2上完成处理的时间和称为该作业调度的完成时间和。批处理作业调度问题要求对于给定的n个作业，制定最佳作业调度方案，使其完成时间和达到最小。

输入	输出
3 2 1 3 1 2 3	18 1 3 2

谢 谢

欢迎批评指正