第6章　分支限界法

授课教师：陈香凝

学习要点

- 理解分支限界法的剪枝搜索策略。
- 掌握分支限界法的算法框架
  - （1）队列式(FIFO)分支限界法
  - （2）优先队列式分支限界法
- 通过应用范例学习分支限界法的设计策略。
  - （1） 0-1背包问题；
  - （2）最优装载问题
  - （3）旅行商问题

概述

- 一种求解离散最优化问题的计算分析方法，又称分枝定界法。
- 这种方法通常仅需计算和分析部分允许解，即可求得最优解。
- 同回溯法一样适合解决组合优化问题。

6.1 分支限界法的基本思想

分支限界法与回溯法

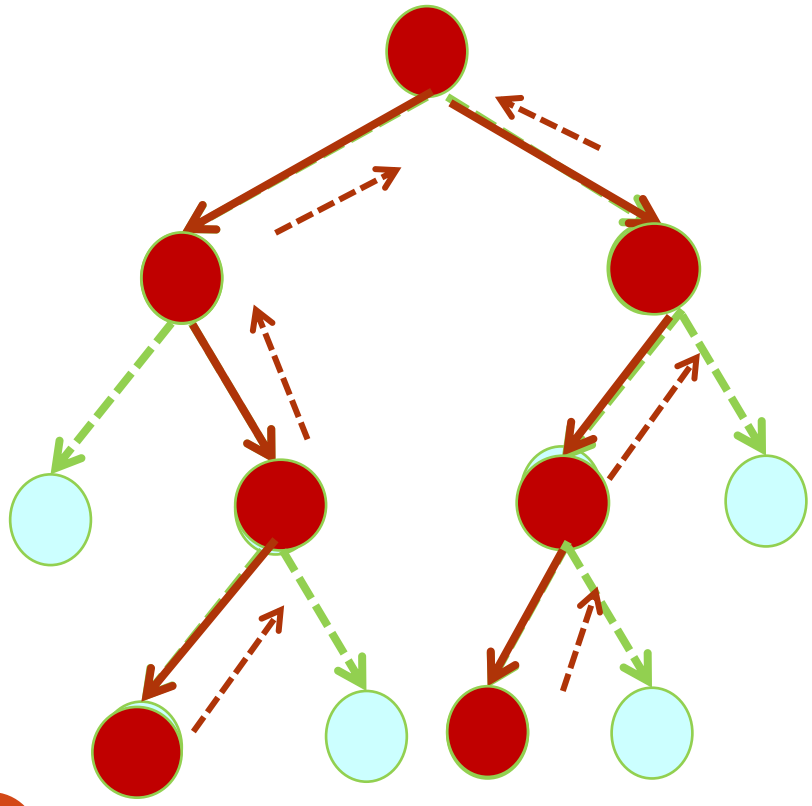
- **求解目标不同：**
  - 回溯法的求解目标是找出解空间树中满足约束条件的**所有解**，而分支限界法的求解目标则是找出满足约束条件的**一个解**，或是在满足约束条件的解中找出在某种意义下的**最优解**。

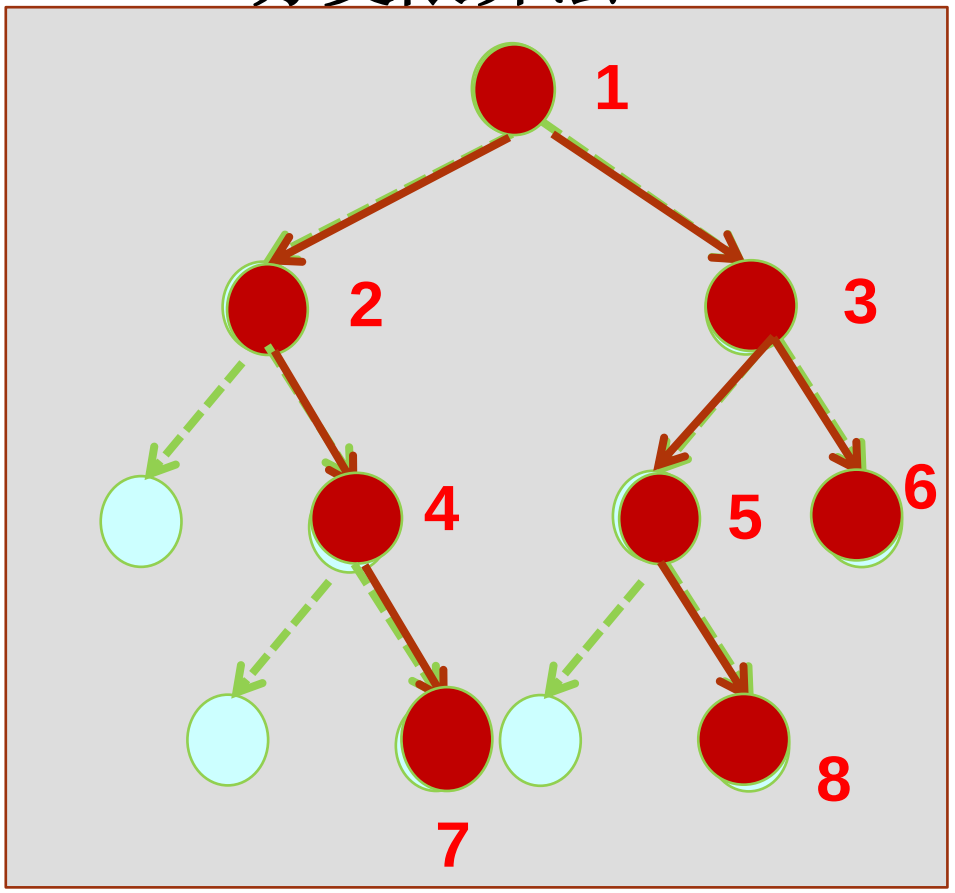
问题：

深度优先搜索和广度优先搜索在处理活结点的方式上有何不同？

0-1背包问题搜索解空间树的模拟过程

回溯法　　活结点表　　分支限界法





6.1 分支限界法的基本思想

- 分支限界法常以广度优先或以最小耗费（最大效益）优先的方式搜索问题的解空间树。
- 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。
- 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。
- 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。

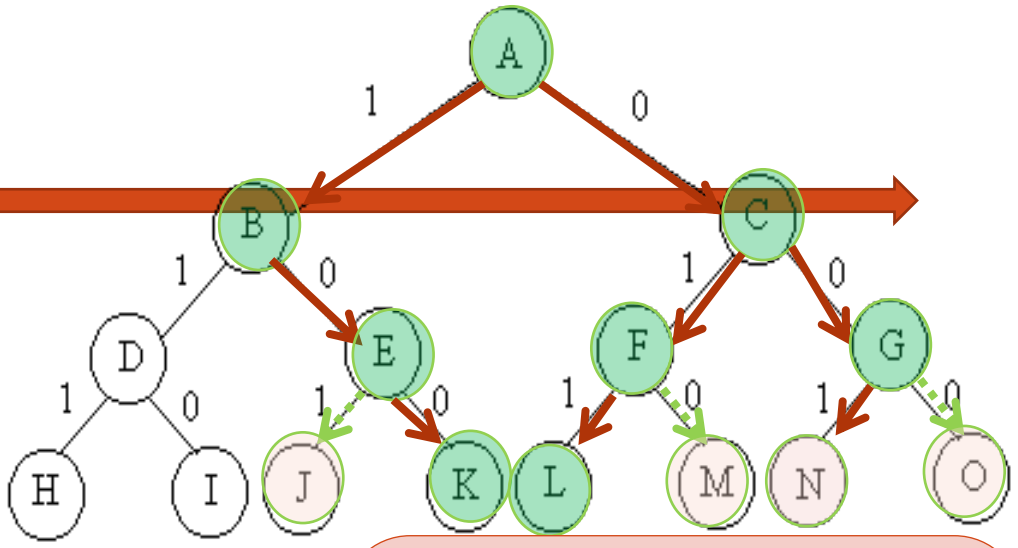
6.1 分支限界法的基本思想

3. 常见的两种分支限界法

- 队列式(FIFO)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。

- 优先队列式分支限界法：
  - 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。

实例



0/1背包问题的两种分支定界方法

W=[16,15,15],

p=[45,25,25], c=30

课堂习题：

N=4, C=7,

P=(15, 11, 7, 2),

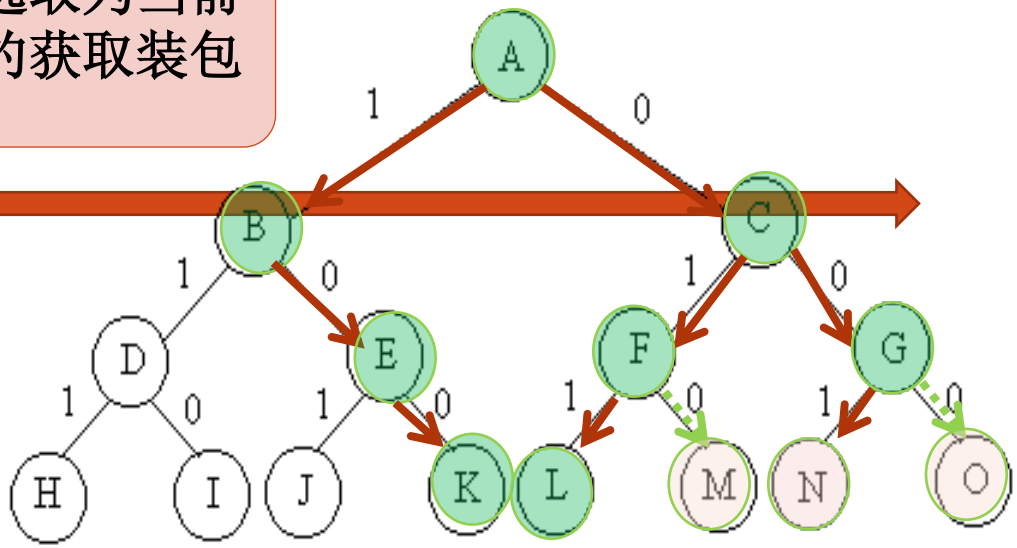
W=(5, 4, 3, 2)

队列式分支限界法：

- [A] B, C => B, C
- [B, C] D, E => E
- [C, E] F, G => F, G
- [E, F, G] J, K => K(45) [1,0,0]
- [F, G] L, M =>L(50) [0, 1, 1] M(25)
- [G] N, O =>N(25), O(0)

实例

若优先级选取为当前节点表示的获取装包价值



0/1背包问题的两种分支界方法

W=[16,15,15],

p=[45,25,25], c=30

优先队列式分支限界法：

- [A] B, C => B(45), C(0)
- [B, C] D, E => E(45)
- [E, C] J, K => K(45) [1, 0, 0]
- [C] F, G => F(25), G(0)
- [F, G] L, M => L(50), [0, 1, 1] M(25)
- [G] N, O => N(25), O(0)

课堂习题：

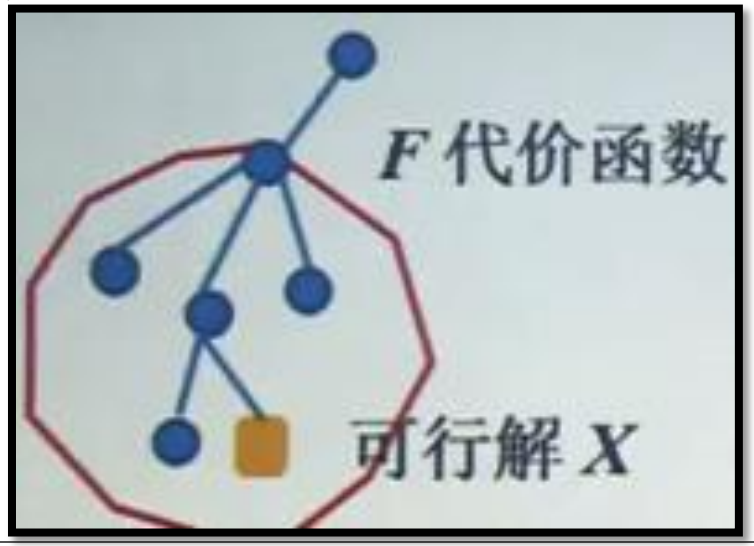
N=4, C=7,

P=(15, 11, 7, 2),

W=(5, 4, 3, 2)

进一步讨论：关于“界”的理解

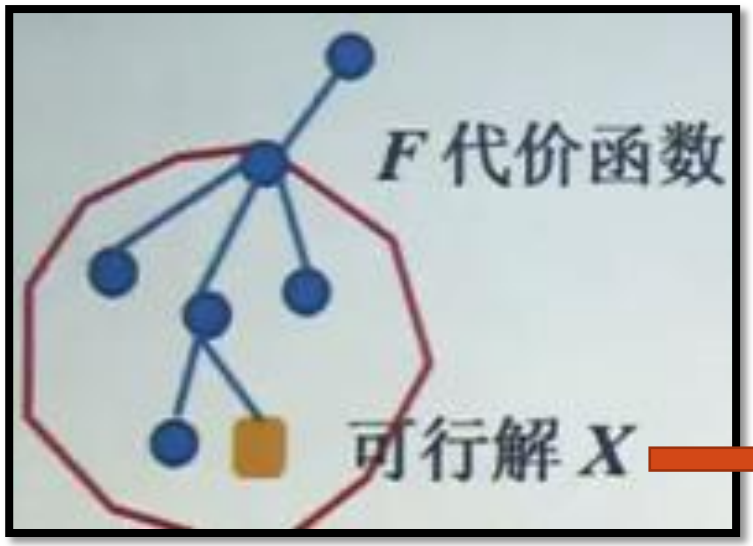
- 代价函数（即限界函数）
  - **计算位置**：搜索树的结点处
  - **值**：最大化问题是以该点为根的子树所有可行解的值得上界（最小化问题这是下界）
  - **性质**：对极大化问题父结点代价不小于子结点的代价（最小化问题相反）



- F代价函数>=v(X)
- 极大化

进一步讨论：关于“界”的理解

- 界的含义：当前得到的可行解的目标函数的最大值（最小化问题想法）
- 界的初值：最大化问题初值为0（最小化初值为最大值）
- 界的更新：得到更好的可行解时。



界B=v(X)

拓展思考：优先级队列的几个问题

- 优先级队列中，通常处理优先级任务最高的任务，处理完后删除该任务，要尽快找到剩余任务中优先级最高的任务，应如何组织这些任务？
  - 最大堆比无序线性结构有何优点？
  - 如何插入一个新任务到一个最大堆？
  - 如何将一组具有优先级的任务组织成最大堆？
  - 在最大堆的根结点（优先级最高的结点）被删除后如何保证最大堆性质？

6.2 分支限界的应用： 0-1背包问题

算法的思想

- 首先，要对输入数据进行预处理，将各物品依其单位重量价值从**大到小**进行**排列**。
- 在**优先队列分支限界法**中，结点的**优先级**由**已装袋的物品价值加上剩下的最大单位重量价值的物品装满剩余容量的价值和**。
- 算法首先检查当前扩展结点的左儿子结点的可行性。如果该**左儿子结点是可行结点**，则将它加入到子集树和活结点优先队列中。
- 当前扩展结点的右儿子结点一定是可行结点，仅当右儿子结点**满足上界约束**时才将它加入子集树和活结点优先队列。当扩展到叶节点时为问题的最优值。

6.2 分支限界的应用： 0-1背包问题

- C——背包容量
- w[i]——第i个物品重量
- p[i]——第i个物品价值
- x[i]]——第i个物品是否装包
- bestp——当前最优价值
- cp——当前价值
- cw ——当前重量
- up——价值上界(已装袋的物品价值加上剩下的最大单位重量价值的物品装满剩余容量的价值和)

算法实现

```
MaxKnapsack()
{  i=1;
   cw=cp=0;
   up=bound(1);   //价值上界
   while (i != n + 1)
   { wt = cw + w[i];      // 检查
      if (wt <= c)        // 左儿子结点为可行结点
      {if (cp + p[i] > bestp)
            bestp = cp + p[i];
         addLiveNode(up,cp + p[i],cw + w[i], true, i + 1);
      }
      up = bound(i + 1); //检查右儿子节点，是否满足上界约束
      if (up >= bestp)                          // 右子树可能含最优解
         addLiveNode(up,cp,cw, false, i + 1 );
                                                // 取下一个扩展节点（略）
      …… }
```

习题：
N=4, C=8,
P=(15, 10, 6, 2),
W=(5, 4, 3, 2)画出采用分支限界法的优先队列搜索过程

0-1背包问题:上界函数 Bound

```
Bound(int i)
{// 计算上界
  cleft = c - cw;  // 剩余容量
  b = cp;
  // 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];
    b += p[i];
    i++;
    }
  // 装满背包
  if (i <= n)
      b += p[i]/w[i] * cleft;
  return b;
  }
```

cw当前装入包裹重量

cp当前装入包裹价值

课堂练习

习题：
N=4, C=7,
P=(15, 11, 7, 2),
W=(5, 4, 3, 2)

```
MaxKnapsack()
{
  while (i != n + 1)                    // 非叶结点
  { wt = cw + w[i];      // 检查当前扩展结点的左儿子结点
      if (wt <= c)       // 左儿子结点为可行结点
      {if (cp + p[i] > bestp)
          bestp = cp + p[i];
        addLiveNode(up,cp + p[i],cw + w[i], true, i + 1);
          }
      up = bound(i + 1); //检查右儿子节点，是否满足上界约束
     if (up >= bestp)                          // 右子树可能含最优解
       addLiveNode(up,cp,cw, false, i + 1 ); //  取下一个扩展节点（略）
    …… }
```