# 分支限界法

**笔记源文件：Markdown，长图，PDF，HTML**

# 1. 分支限界法和回溯法

1. 求解目标的区别
   1. 回溯法：求出解空间树所有满足约束的解
   2. 分支限界法：找出满足约束的一个解OR在满足约束的解中找出一个最优
2. 搜索方式的区别：回溯法以**深度优先**搜索解空间树，分支界限法以**广度优先或最小耗费优先**
3. 结点扩展模式
   1. 每个活结点只有一次机会成为扩展结点
   2. 成为扩展结点后，就会一次性产生所有子节点
   3. 舍弃导致不可行解OR非最优解的子节点，剩下的子节点加入活结点队列
   4. 该结点扩展完成后，再从列表中取下一活结点扩展，循环往复
4. 如何从活结点列表中选择活结点：FIFO，优先队列

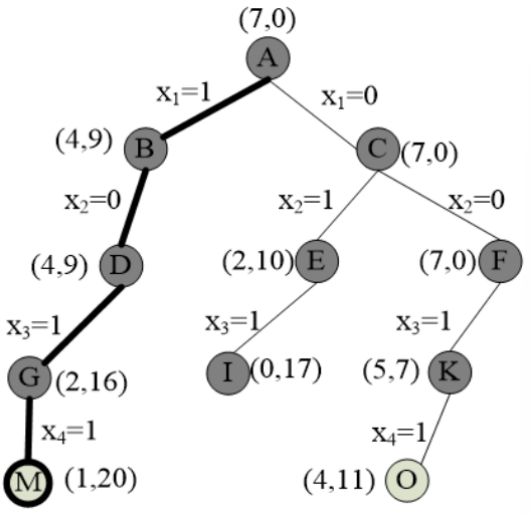
# 2. 示例: 0-1背包问题

## 2.1. 问题概述

1. 条件：四个物品，重量分别为$w=[3,5,2,1]$，价值分别为$v=[9,10,7,4]$，背包总重量为7
2. 解空间：$(x_1,x_2,x_3,x_4),x_i=0/1$表示物品选择与否，用四层的子集树表示
3. 限界条件：当前价值$cp$+剩余所有价值总和$rp$ <当前最优$bestp$，直接裁掉该子树

## 2.2. FIFO队列分支界限法

1. 初始化与更新：$cp=bsetp=0$，$rp=30$，当$cp>bestp$时更新$bestp=cp$
2. 解空间




3. 过程概述

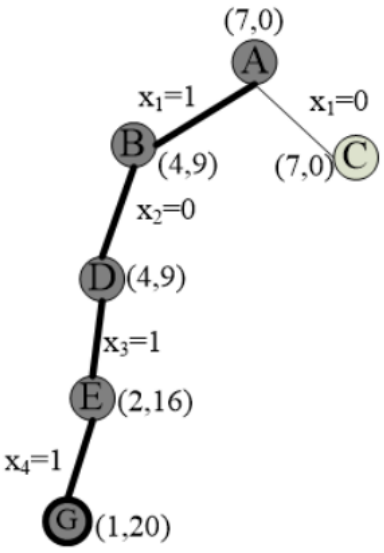
| 结点 | 先考虑$x_i=1$子节点 | 再考虑$x_i=0$子节点 | 活结点队列 |
| --- | --- | --- | --- |
| A | B(bestp=cp=9, cp+rp=9+21>bestp) | C(bestp=9, cp+rp=0+21>bestp) | →BC |
| B | 背包载重量不足舍 | D(bestp=9, cp+rp=9+11>bestp) | →CD |
| C | E(bestp=cp=10, cp+rp=10+11>bestp) | F(bestp=10, cp+rp=0+11>bestp) | →DEF |
| D | G(bestp=cp=16, cp+rp=16+4>bestp) | bestp=16, cp+rp=9+4<bestp舍 | →EFG |
| E | I(bestp=cp=17, cp+rp=17+4>bestp) | bestp=17, cp+rp=10+4<bestp舍 | →FGI |
| F | K(bestp=17, cp+rp=7+4<bestp) 舍 | bestp=17, cp+rp=0+4<bestp舍 | →GI |
| G | M(bestp=cp=20, cp+rp=20+0>bestp) | bestp=20, cp+rp=16+0<bestp舍 | →IM |
| I | 背包载重量不足舍 | 背包载重量不足舍 | →M |
| M | 物品全部选完，再无子节点 | 物品全部选完，再无子节点 | → |

## 2.3. 优先队列分治界限法

1️⃣优先队列法概述

1. 假设当前结点有$a_1, a_2, \ldots, a_n$子节点，估算通过$a_i$继续搜索时目标函数所能达到的上界/下界
2. 将符合要求的子结点加入活结点队列，按照上一步中的估值为他们排列
3. 从队列中选取估值最大/最小的结点，作为下一个要考察的结点
4. 一直到达叶节点，如果当前的目标函数值已经是当前已知最值，那么这个便是最优解

2️⃣解空间



3️⃣过程概述：以结点的价值上界作为优先级

| 结点 | 先考虑$x_i=1$子节点 | 再考虑$x_i=0$子节点 | 队列+估价 |
| --- | --- | --- | --- |
| A | B(bestp=cp=9, cp+rp=9+21>bestp) | C(bestp=9, cp+rp=0+21>bestp) | B(20)C(17) |
| B | 背包载重量不足舍 | D(bestp=9, cp+rp=9+11>bestp) | D(20)C(17) |
| D | E(bestp=cp=16, cp+rp=16+4>bestp) | bestp=16, cp+rp=9+4<bestp舍 | E(20)C(17) |
| E | G(bestp=cp=20, cp+rp=20+0=bestp) | bestp=20, cp+rp=16+0<bestp舍 | G(20)C(17) |

1. 重量$w=[3,5,2,1]$，价值$v=[9,10,7,4]$
2. 估价当作优先级

| 结点 | 已确定的装入选择 | 背包剩余空间 | 能使估价最优的操作 | 最优估价 |
| --- | --- | --- | --- | --- |
| B | 装1 | 4kg | 再装物品4+物品3 | 9+7+4=20 |
| C | 不装1 | 7kg | 再装物品2+物品3 | 10+7=17 |
| D | 装1不装2 | 4kg | 再装物品3+物品4 | 9+7+4=20 |
| E | 装1不装2装3 | 2kg | 再装物品4 | 9+7+4=20 |

3. 当来到了叶节点$G$时，队列为→G(20)C(17)，按照优先级应该接下来考虑$G$，但由于其实叶节点无法再分下去，所以就认为其时最优解

# 3. 单源最短路径问题

求有向图中两点间的最短距离，贪心策略采用Dijikstra算法

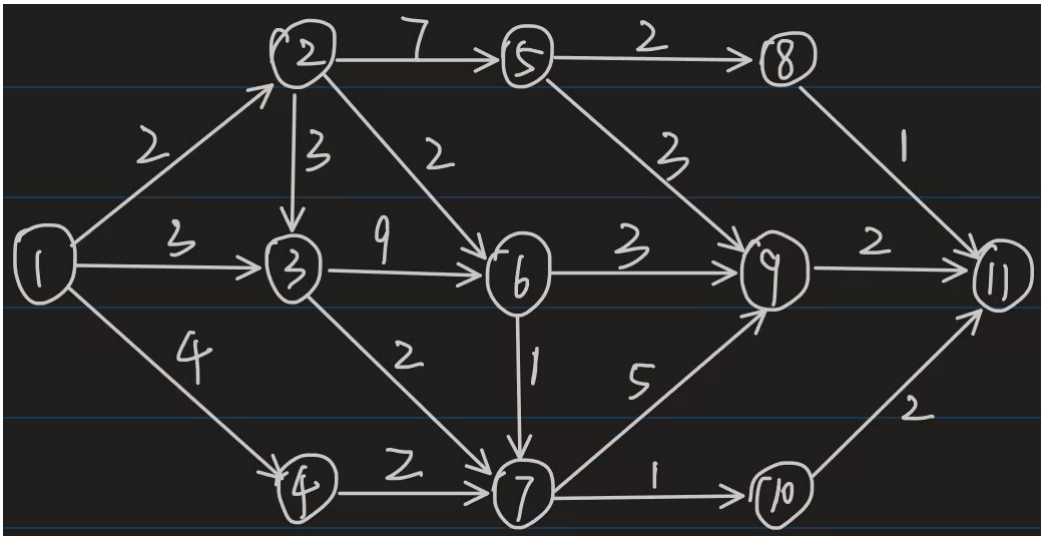
## 3.1. 优先队列式分支限界策略

1️⃣某点的优先级：基于原点到该点已知的最短距离，距离越短优先级越高

2️⃣算法操作：

1. 初始化：确定原点$s$和空的优先队列
2. 扩展结点：
   - 原点$s$：考察所有与$s$直接相连的顶点，计算他们间的距离，然后全部加入优先队列
   - 其他结点$v_i$：考察与之相连的顶点$v_j$，如果存在$s\xrightarrow{经过v_i}v_j$，则更新$v_j$的最短距离并加入队列
3. 优先队列使用：根据结点当前路径长度来排序，每次取出最小路径长的结点，进行下一轮扩展
4. 结束：一直到优先队列变空

## 3.2. 算法实例

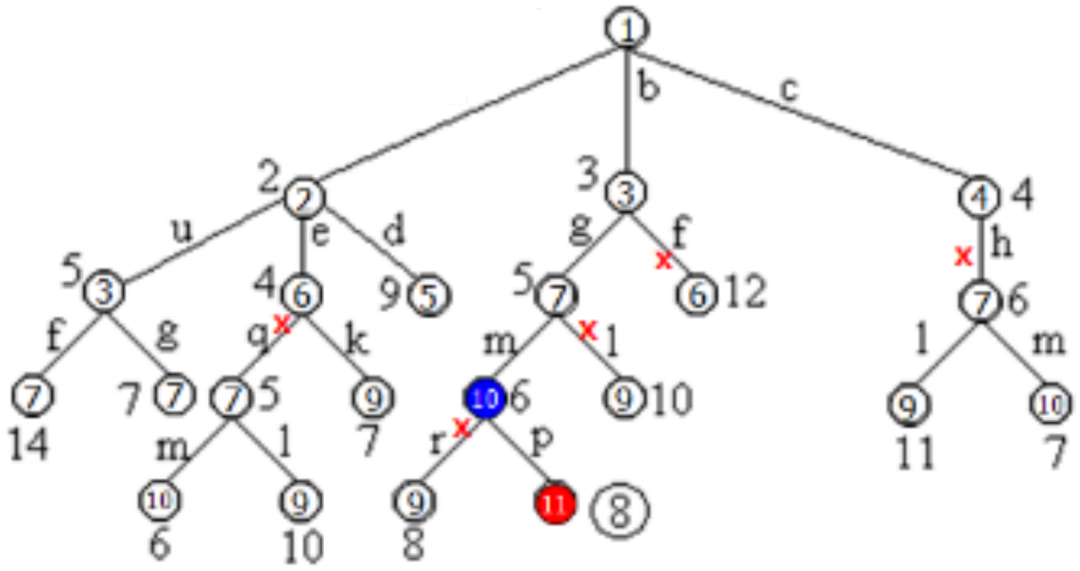



1️⃣分支限界的过程

| 扩展结点 | 源到各顶点的已知最短距离 | 优先队列 |
| --- | --- | --- |
| 1(邻接2,3,4) | 2(2)+3(3)+4(4) | 2(2)+3(3)+4(4) |
| 2(邻接3,5,6) | 2(2)+3(3)+4(4)+5(9)+6(4) | 3(3)+4(4)+6(4)+5(9) |
| 3(邻接6,7) | 2(2)+3(3)+4(4)+5(9)+6(4)+7(5) | 4(4)+6(4)+7(5)+5(9) |
| 4(邻接7) | 2(2)+3(3)+4(4)+5(9)+6(4)+7(5) | 6(4)+7(5)+5(9) |
| 6(邻接7,9) | 2(2)+3(3)+4(4)+5(9)+6(4)+7(5)+9(7) | 7(5)+9(7)+5(9) |
| 7(邻接9,10) | 2(2)+3(3)+4(4)+5(9)+6(4)+7(5)+9(7)+10(7) | 10(7)+9(7)+5(9) |
| 10(邻接11) | 2(2)+3(3)+4(4)+5(9)+6(4)+7(5)+9(7)+10(7)+11(9) | 9(7)+5(9)+11(9) |
| 9(邻接11) | 2(2)+3(3)+4(4)+5(9)+6(4)+7(5)+9(7)+10(7)+11(9) | 5(9)+11(9) |
| 5(邻接8,9) | 2(2)+3(3)+4(4)+5(9)+6(4)+7(5)+8(11)+9(7)+10(7)+11(9) | 11(9)+8(11) |
| 11(无邻接) | 2(2)+3(3)+4(4)+5(9)+6(4)+7(5)+8(11)+9(7)+10(7)+11(9) | 8(11) |
| 8(11) | 2(2)+3(3)+4(4)+5(9)+6(4)+7(5)+8(11)+9(7)+10(7)+11(9) | null |

所以1→11最短距离为9

2️⃣子集树表示以上过程



# 4. 作业分配问题

## 4.1. 问题描述

1. 有$n$个操作员，要完成$n$个不同作业
2. 数据结构
   1. 矩阵$c[i][j]$：操作员$i$完成任务$j$所耗的时间
   2. 向量$x[i]$：操作员$i$所分得的作业编号
3. 问题：让$n$个作业在最短时间内完成

## 4.2. 算法设计

### 4.2.1. 基本方法

1. 从根节点开始，每遇到一个扩展结点，就计算其子节点的下界，并登记再节点表中
2. 选取下界最小的子节点再扩展
3. 一直搜索到叶子结点
   1. 当叶结点的下界是节点表中最小的，那么这个就是最优解
   2. 否则就继续对下届最小的结点扩展

### 4.2.2. 下界的确认

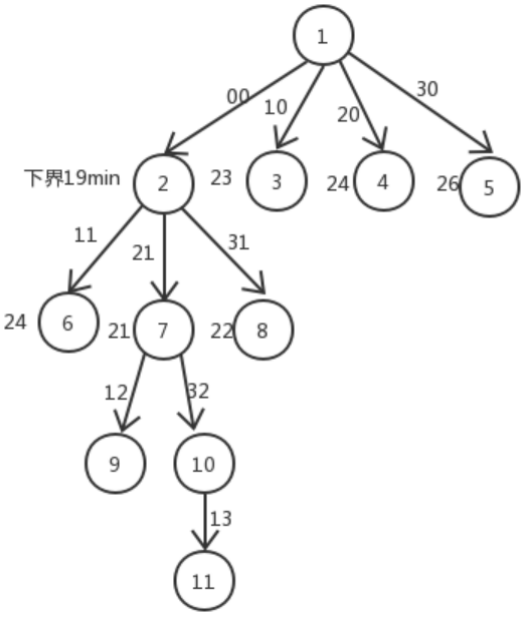
1. 初始阶段：搜索深度为0时的下界，$t = c_{i0} + \sum_{j=1}^{n-1}\left(\min_{l\neq i} c_{lj}\right)$
   1. 让操作员$i$完成作业0，耗时$c_{i0}$
   2. 其余$n-1$个作业由剩下操作员完成，最短耗时$\sum_{j=1}^{n-1}\left(\min_{l\neq i} c_{lj}\right)$，找到最小即可不在意操作员是否重复
   3. 示例：0作业分给0操作员后，1作业交给2或3，2作业交给6，3作业交给2

| 操作员\作业 | 0 | 1 | 2 | 3 |
| --- | --- | --- | --- | --- |
| 0 | 3 | 8 | 4 | 12 |
| 1 | 9 | 12 | 13 | 5 |
| 2 | 8 | 7 | 9 | 3 |
| 3 | 12 | 7 | 6 | 8 |

2️⃣搜索深度为k时的下界：$t=\sum_{l=0}^{k}c_{i_l l}+\sum_{l=k+1}^{n-1}\left(\min_{i\in S-m_k}c_{i,l}\right)$

1. $1\rightarrow k$编号的作业已分配，让操作员$i_l$完成作业$l$且$l\in[1,k]$，则耗时$\sum_{l=0}^{k}c_{i_l l}$
2. 对于未分配的作业，令操作员集合$S$减去已分配操作员集合$m_k$即$S-m_k$为未分配操作员集合，则剩余$n-k$个作业耗时最短为$\sum_{l=k+1}^{n-1}\left(\min_{i\in S-m_k}c_{il}\right)$

## 4.3. 算法实例




1️⃣第一层：啥都没有，让第0/1/2/3个操作员选择0四个作业，即00/10/20/30，扩展出第二层

2️⃣第二层：有四个结点，00结点的下界最小，所以选择00结点扩展

| 结点 | 作业0 | 作业1 | 作业2 | 作业3 | 下界 |
| --- | --- | --- | --- | --- | --- |
| 00 | 0操作员，耗时3 | 2操作员，耗时7 | 3操作员，耗时6 | 2操作员，耗时3 | 19 |
| 10 | 1操作员，耗时9 | 2操作员，耗时7 | 0操作员，耗时4 | 2操作员，耗时3 | 23 |
| 20 | 2操作员，耗时8 | 3操作员，耗时7 | 0操作员，耗时4 | 1操作员，耗时5 | 24 |

| 结点 | 作业0 | 作业1 | 作业2 | 作业3 | 下界 |
| --- | --- | --- | --- | --- | --- |
| 30 | 3操作员，耗时12 | 2操作员，耗时7 | 0操作员，耗时4 | 2操作员，耗时3 | 26 |

**优先队列→**

由于0操作员已经用过了，所以考虑1/2/3操作员完成任务1，即11/21/31，扩展出第三层

3️⃣第三层：有三个结点，21结点的下界最小，所以选择21结点扩展

| 结点 | 作业0 | 作业1 | 作业2 | 作业3 | 下界 |
| --- | --- | --- | --- | --- | --- |
| 11 | 0操作员，耗时3 | 1操作员，耗时12 | 3操作员，耗时6 | 2操作员，耗时3 | 24 |
| 21 | 0操作员，耗时3 | 2操作员，耗时7 | 3操作员，耗时6 | 1操作员，耗时5 | 21 |
| 31 | 0操作员，耗时3 | 3操作员，耗时7 | 2操作员，耗时9 | 2操作员，耗时3 | 22 |

```
优先队列→[21] [31] [11]
下界       21   22   24
```

由于0/2操作员已经用过了，所以考虑1/3操作员完成任务2，即12/32，扩展出第四层

4️⃣第四层：有两个节点，32下界最小，再扩展32结点

| 结点 | 作业0 | 作业1 | 作业2 | 作业3 | 下界 |
| --- | --- | --- | --- | --- | --- |
| 12 | 0操作员，耗时3 | 2操作员，耗时7 | 1操作员，耗时13 | 3操作员，耗时8 | 31 |
| 32 | 0操作员，耗时3 | 2操作员，耗时7 | 3操作员，耗时6 | 1操作员，耗时5 | 21 |

```
优先队列→[32] [31] [11] [12]
下界       21   22   24   31
```

由于0/2/3操作员已经用过了，所以考虑1操作员完成任务3，即13

5️⃣第五层：终于来到了叶节点

| 结点 | 作业0 | 作业1 | 作业2 | 作业3 | 下界 |
| --- | --- | --- | --- | --- | --- |
| 13 | 0操作员，耗时3 | 2操作员，耗时7 | 3操作员，耗时6 | 1操作员，耗时5 | 21 |

> 
> ```
> 优先队列→[13] [31] [11] [12]
> 下界      21   22   24   31
> ```
>
> 当一个结点达到优先队列的队首，且为叶节点时，这个结点就是最优值