

第6章 分支限界法

- 分支限界法的基本思想
 - 什么是分支限界法
 - 分支限界的设计思想
- 典型应用
 - 加1乘2平方 (4-1) ★
 - 电子老鼠闯迷宫 (4-2) ★
 - 八数码问题 (4-3) ★
 - 0-1背包问题 ★
 - 独轮车
 - 分油问题
- 总结

分枝限界法采用广度优先搜索，并在生成当前结点的孩子结点时，通过一些限界条件即限界函数，帮助避免生成不包含解结点子树的状态空间的检索方法。

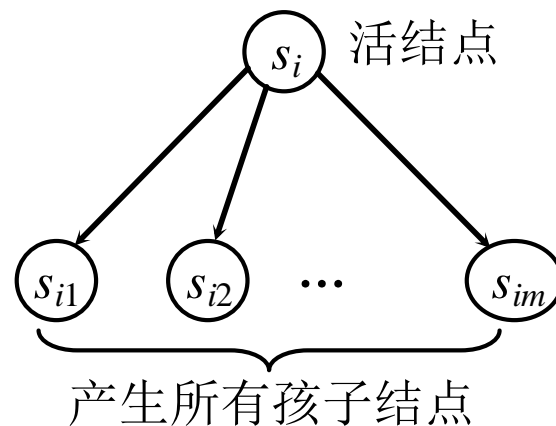
分枝限界法提高了求解问题的效率。

6.1 分枝限界法的基本思想

6.1.1 什么是分枝限界法

所谓“分枝”就是采用**广度优先**的策略，依次搜索活结点的所有分枝，也就是所有相邻结点，如图所示。

为了有效的选择下一扩展结点，以加速搜索的进程，在每一活结点处，计算一个函数值（限界函数），并根据这些已计算出的函数值，从当前活结点表中选择一个最有利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枝推进，以便尽快地找出一个最优解。



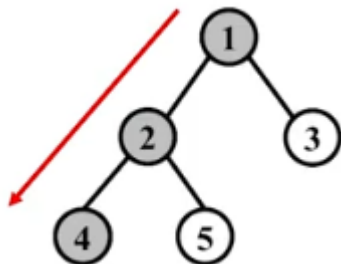
分支限界法与回溯法的不同：★

● 求解目标：

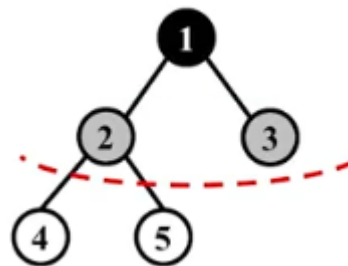
- 回溯法：找出解空间树中满足约束条件的**所有解**；
- 分枝限界法：找出满足约束条件的**一个解**，或是在满足约束条件的解中找出在某种意义下的**最优解**。

● 搜索方式的不同：

- 回溯法：**深度优先**的方式搜索解空间树；
- 分枝限界法：以**广度优先**或以最小耗费优先的方式搜索解空间树。



深度优先



广度优先

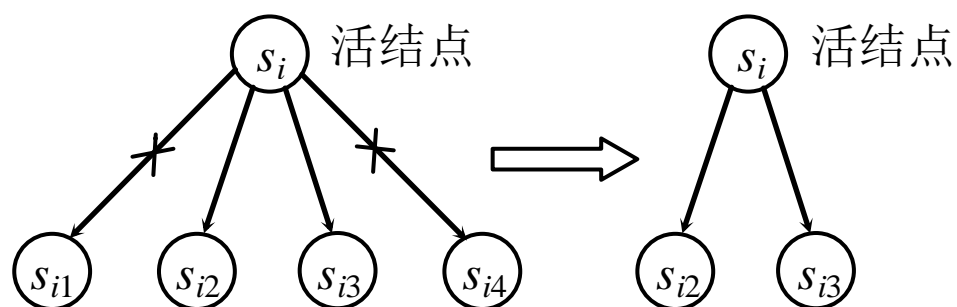
6.1.2 分枝限界法的设计思想

1. 设计合适的限界函数

在搜索解空间树时，每个活结点可能有很多孩子结点，其中有些孩子结点搜索下去是不可能产生问题解或最优解的。

可以设计好的**限界函数**在扩展时**删除这些不必要的孩子结点**，从而提高搜索效率。

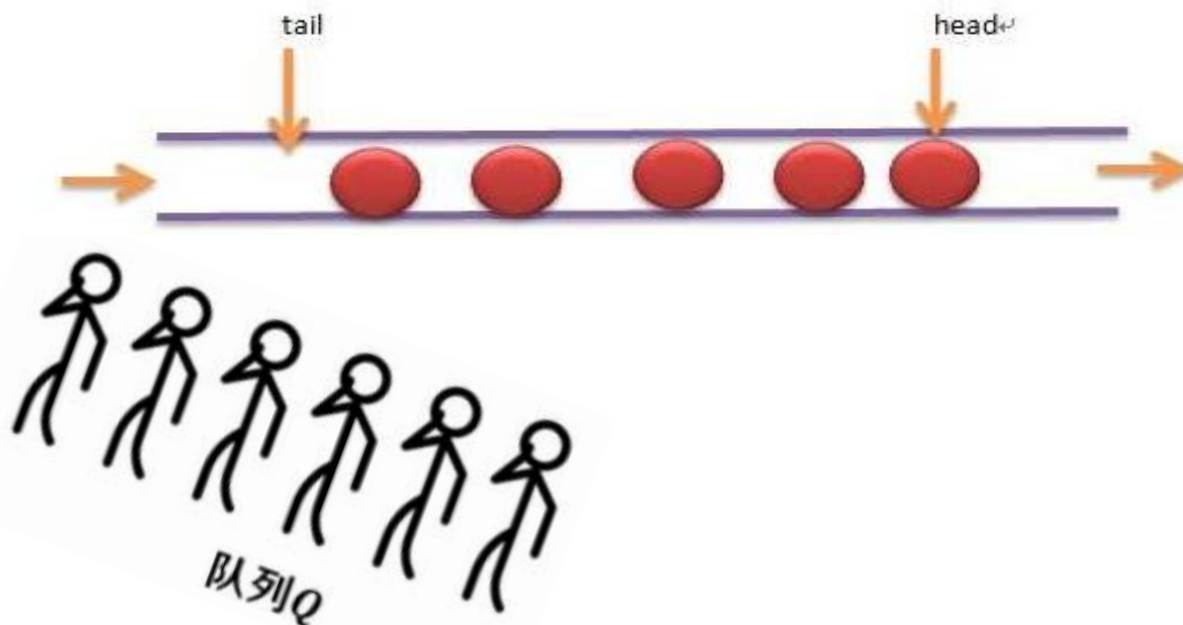
如图所示，假设活结点 s_i 有4个孩子结点，而满足限界函数的孩子结点只有2个，可以删除这2个不满足限界函数的孩子结点，使得从 s_i 出发的搜索效率提高一倍。



分枝限界法 = 广搜 + 剪枝

2.常见的两种分支限界法

根据选择下一个扩展结点的方式来组织活结点表，不同的活结点表对应不同的分枝搜索方式。常见的有**队列式**分枝限界法和**优先队列式**分枝限界法两种。



(1) 队列式分枝限界法

将活结点表组织成一个队列，按照先进先出（**FIFO**）原则选取下一个结点为扩展结点：

- ① 将根结点加入活结点队列。
- ② 从活结点队中取出队**头结点**，作为当前扩展结点。
- ③ 对当前扩展结点，先从左到右地产生它的所有孩子结点，**用约束条件检查**，把所有满足约束条件的孩子结点加入活结点队列。
- ④ 重复步骤②和③，直到找到一个解或活结点队列为空为止。

队列式分支限界法与广度优先的遍历算法极为相似，唯一不同在于：分支限界法**不搜索**不可行节点子树。

(2) 优先队列式分枝限界法

在许多应用中需要另一种队列，每次从队列中取出的应是具有最高优先权的元素，这种队列就是优先级队列。在相同优先级的情况下则遵照先来先服务的原则。

优先队列式分枝限界法的主要特点是将活结点表组组成一个优先队列，并选取优先级最高的活结点成为当前扩展结点。步骤如下：

- ① 计算起始结点（根结点）的优先级并加入优先队列（与特定问题相关的信息的函数值决定优先级）。
- ② 从优先队列中取出优先级最高的结点作为当前扩展结点，使搜索朝着解空间树上可能有最优解的分枝推进，以便尽快地找出一个最优解。
- ③ 对当前扩展结点，先从左到右地产生它的所有孩子结点，然后用约束条件检查，对所有满足约束条件的孩子结点计算优先级并加入优先队列。
- ④ 重复步骤②和③，直到找到一个解或优先队列为空为止。

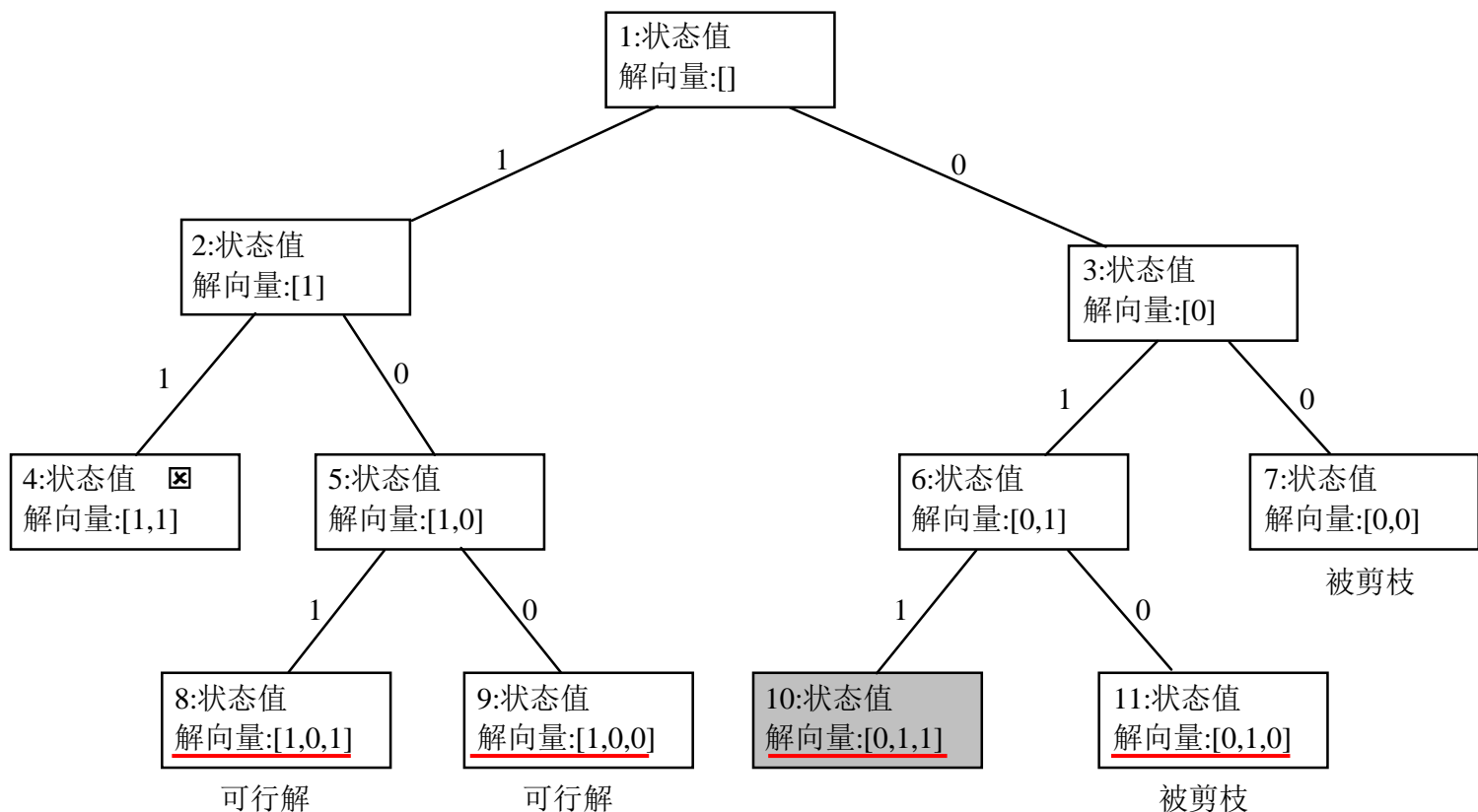
3. 确定最优解的解向量

分枝限界法在搜索解空间树时，结点的处理是跳跃式的，回溯也不是单纯地沿着双亲结点一层一层地向上回溯，因此当搜索到某个叶子结点且该结点对应一个可行解时，**如何得到对应的解向量呢？**

主要有两种方法。

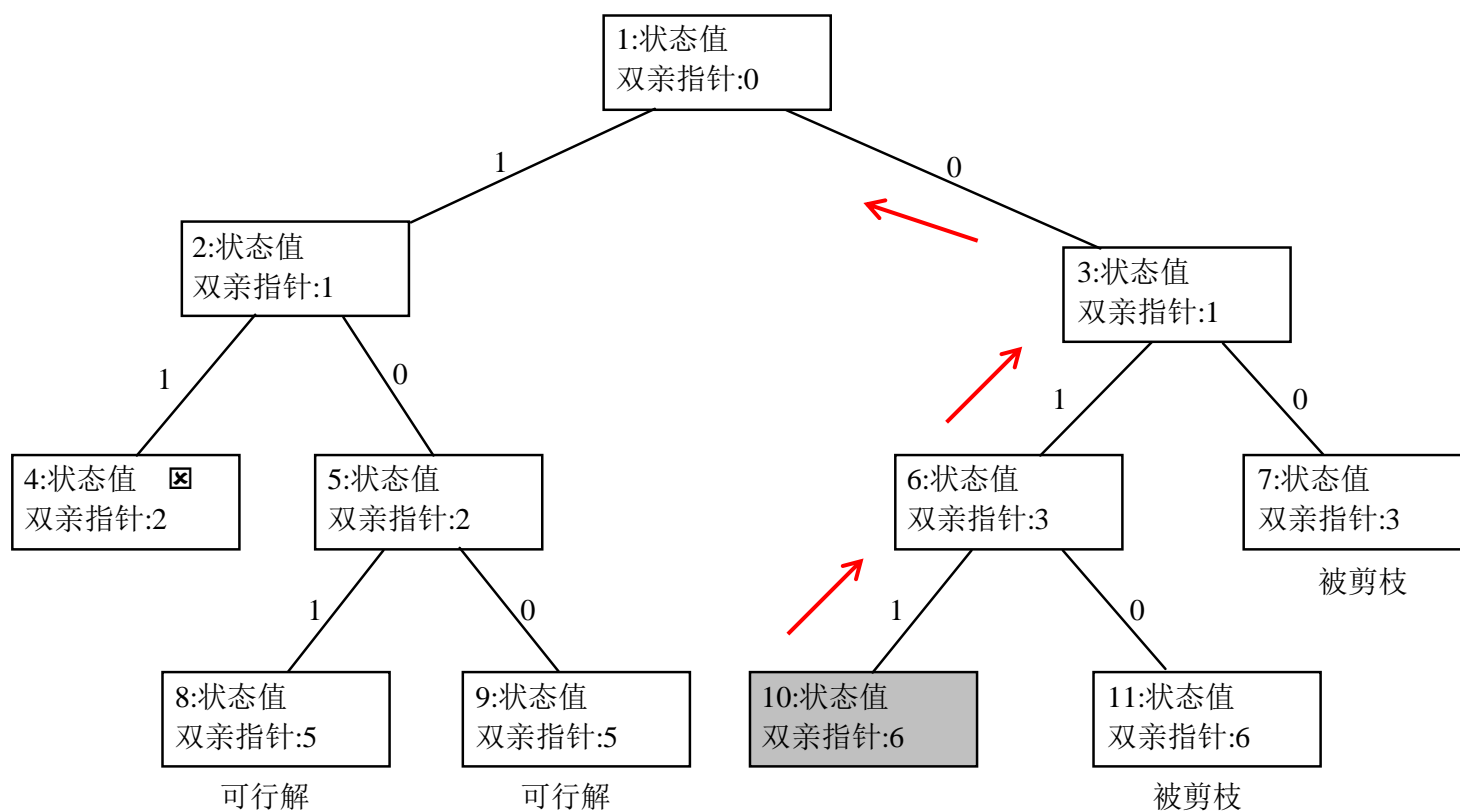
① 对每个扩展结点保存从根结点到该结点的路径。

如图所示，结点编号为搜索顺序，每个结点带有一个可能的解向量，带阴影的结点为最优解结点，对应的最优解向量为 $[0,1,1]$ 。这种做法比较浪费空间，但实现起来简单。



② 在搜索过程中构建搜索经过的树结构。

如图所示，结点编号为搜索顺序，每个结点带有一个双亲结点指针（根结点的双亲结点指针为0或-1），带阴影的结点为最优解结点，当找到最优解时，通过双亲指针找到对应的最优解向量为 $[0,1,1]$ 。这种做法需保存搜索经过的树结构，每个结点增加一个指向双亲结点的指针。



6.1.3 分枝限界法的时间性能

一般情况下，在问题的解向量 $X = (x_1, x_2, \dots, x_n)$ 中，分量 x_i ($1 \leq i \leq n$) 的取值范围为某个有限集合 $S_i = (s_{i1}, s_{i2}, \dots, s_{ir})$ 。

因此，问题的解空间由笛卡尔积 $S_1 \times S_2 \times \dots \times S_n$ 构成，第1层根结点有 $|S_1|$ 棵子树，第2层有 $|S_1|$ 个结点，第2层的每个结点有 $|S_2|$ 棵子树，则第3层有 $|S_1| \times |S_2|$ 个结点，依此类推，第 $n+1$ 层有 $|S_1| \times |S_2| \times \dots \times |S_n|$ 个结点，它们都是叶子结点，代表问题的所有可能解。

在最坏情况下，分支限界法的时间复杂度是指数级的，同时还需要维护活节点队列，其空间复杂度最坏情况也是指数级的。

加1乘2平方

问题描述：

给定两个正整数 m 、 n ，问只能做加1、乘2和平方这三种变化，从 m 变化到 n 最少需要几次。

输入：

两个10000以内的正整数 m 和 n 。

输出：

从 m 变化到 n 的最少次数。

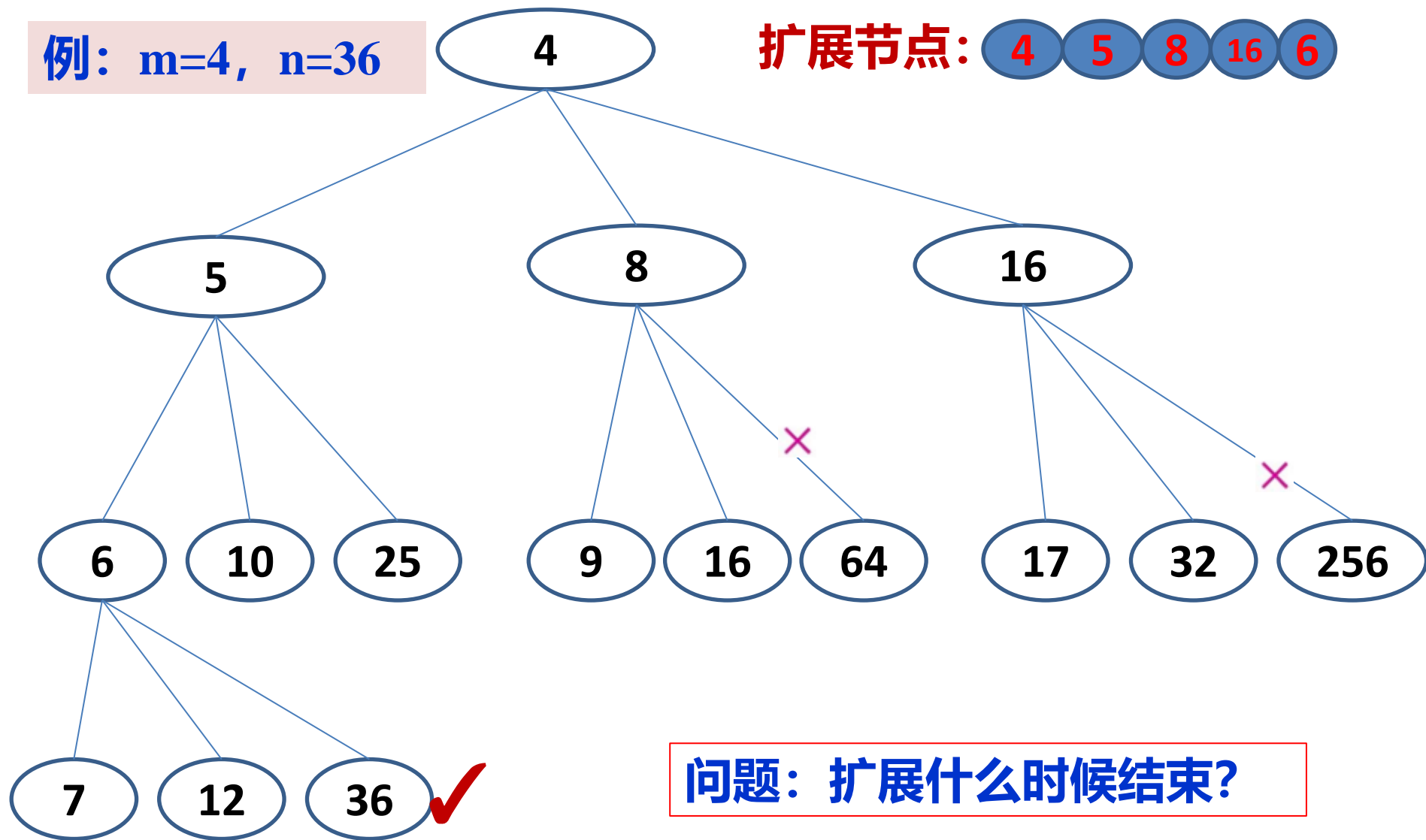
活节点 (队列)

4 5 8 16 6 10 25 9 16 17 32 7 12 36



例: $m=4$, $n=36$

扩展节点: 4 5 8 16 6



```

// 是否m能够经过运算，到达n
// return: ture, 可以; false, 不可;
bool bfs(int m, int n)
{
    int pop_num;
    q.push(m);
    visited[m] = 0;          //已入队数据要标记，便于查重；第一个数的父节点数标记为0!
    while(!q.empty())       // 只要队列不空
    {
        pop_num = q.front(); q.pop();    // 弹出头记录
        // 探索3个方向的可能路径
        int new_num[3];
        for (int i=0; i<3; i++) {
            new_num[i] = make_new_place(pop_num, i); //生成新数，通过 +1、*2、^2
            if (check_OK(new_num[i]))                //如果产生的新节点是可行的（包括查重）
            {
                q.push(new_num[i]);                  // 将它加入队列尾部
                visited[new_num[i]] = pop_num;        // 记住它的父亲数据
                //如果这个新位置就是终点，则搜索结束
                if (new_num[i] == n)
                {
                    return true;
                }
            }
        }
    }
    //while
    return false;
}

```



```
int make_new_num(int root, int type)
{
    switch(type)
    {
        case 0:
            return root+1;
            break;
        case 1:
            return root*2;
            break;
        case 2:
            return root*root;
            break;
    }
    return 0;
}
```

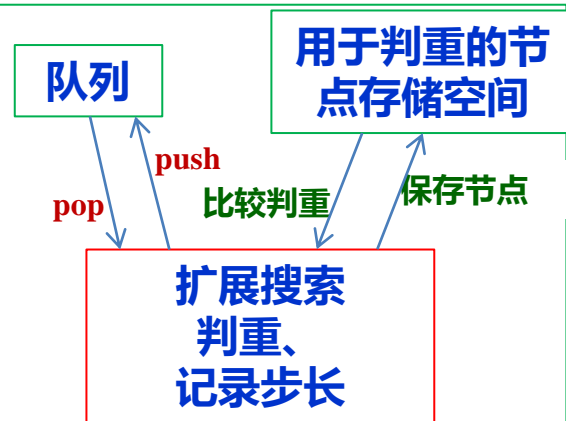
```
bool check_OK(int num)
{
    // 越界 || 之前出现过
    if (num>n || visited[num]>=0)
    {
        return false;
    }

    return true;
}
```

BFS算法的一般模式



```
bool bfs ()
{
    队列初始化为空;
    起点加入到队列;
    while(队列非空)
    {
        从队列中出队一个节点 x (获取并删除) ; // step1
        for (x的所有可扩展节点y) // step2
        {
            if (检查可达性通过 (包括判重) OK) //step3
            {
                将新扩展节点排入队列 //step4
                记录y的父节点为x;
                记录到达y的步数
            }
            // 找到目标, 结束搜索
            if (新扩展节点y == 目标节点) return true; //step5
        }
    }
}
```



BSF算法简介的基本框架（另外一种详细写法）

```
bool bfs ()
{
    queue <类型> q;           // 队列
    q.push(start);           // 向队列加入起始扩展节点
    记录步数d[start] = 0;
    while (!q.empty())
    {
        从队列q中出队一个节点 x;
        for (x的所有可扩展节点y)
        {
            if (检查可达性通过 (包括判重) )
            {
                q.push(y);           // 将新扩展节点排入队列
                记录y的父节点为x;
                d(y) = d(x)+1;       // 记录到达y的步数
            }
            if (新扩展节点y == 目标节点) return true; // 找到目标，结束搜索
        }
    }
}
```

BSF算法的一般模式（扩展后直接查看是否达到目标节点）

```
void search()
{
    open表初始化为空;
    起点加入到open表;
    while( open表非空 )
    {
        取open表中的一个结点u;
        从open表中删除u;
        for( 对扩展结点u得到的每个新结点vi )
        {
            if(vi是目标结点)
                输出结果并返回;
            If(notused(vi)) // 判重
                vi进入open表;
        }
    }
}
```

STL (Standard Template Library) 中队列的用法:

- ❑ 头文件: `#include <queue.h>`
- ❑ 使用命名空间: `using namespace std;`
- ❑ 建立队列: `queue<元素类型>队列名`
- ❑ 基本函数:
 - ❑ 向队列中 (队尾) 添加元素: `队列名.push(元素名)`
 - ❑ 获取队列头元素: `队列名.front()`
 - ❑ 删除队列头元素: `队列名.pop()`
 - ❑ 判断队列是否为空: `队列名.empty()`
 - ❑ 获取队列元素个数: `队列名.size()`

<http://www.cplusplus.com/reference/queue/queue/push/>

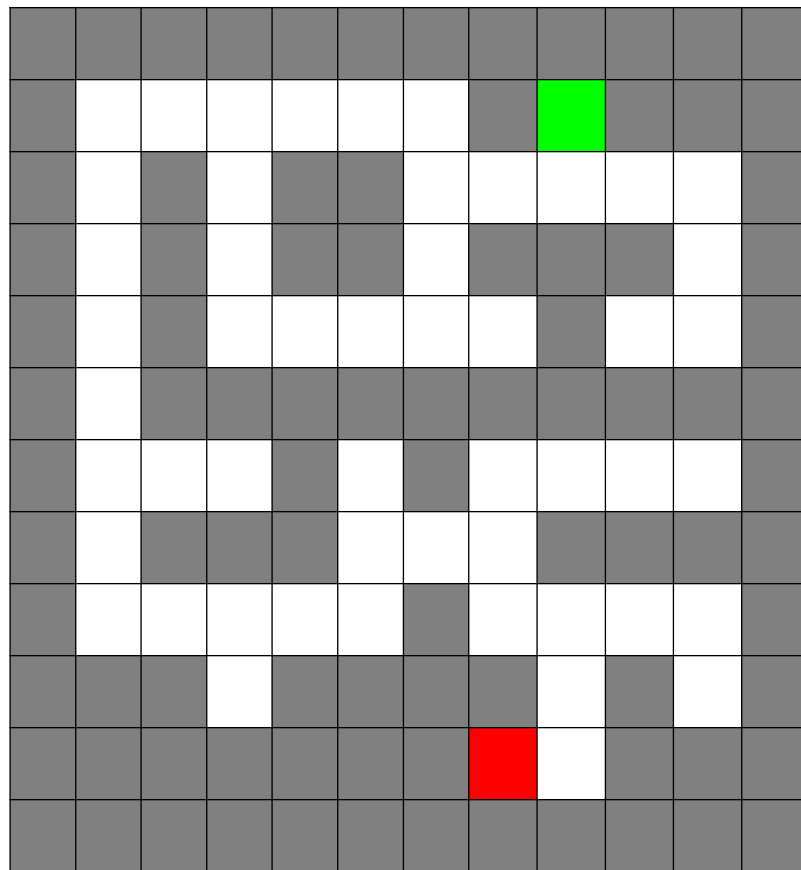
STL教程: C++ STL快速入门<http://c.biancheng.net/stl/>

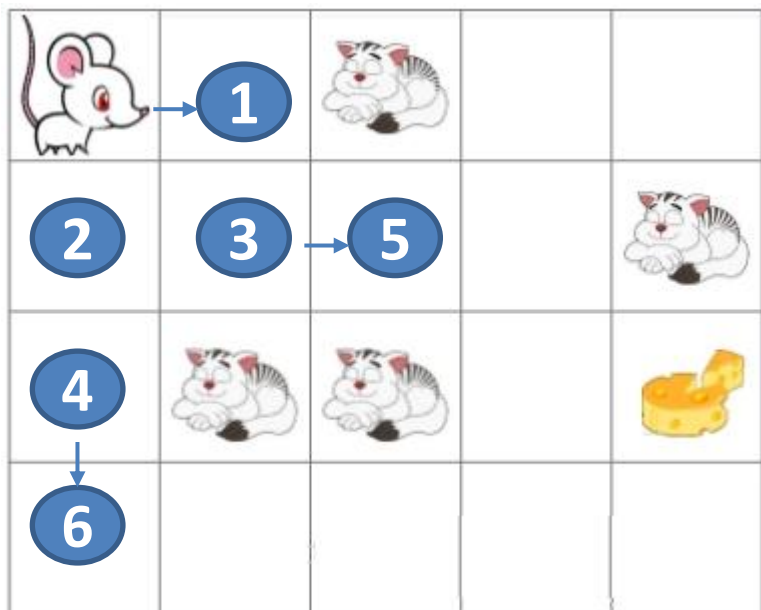
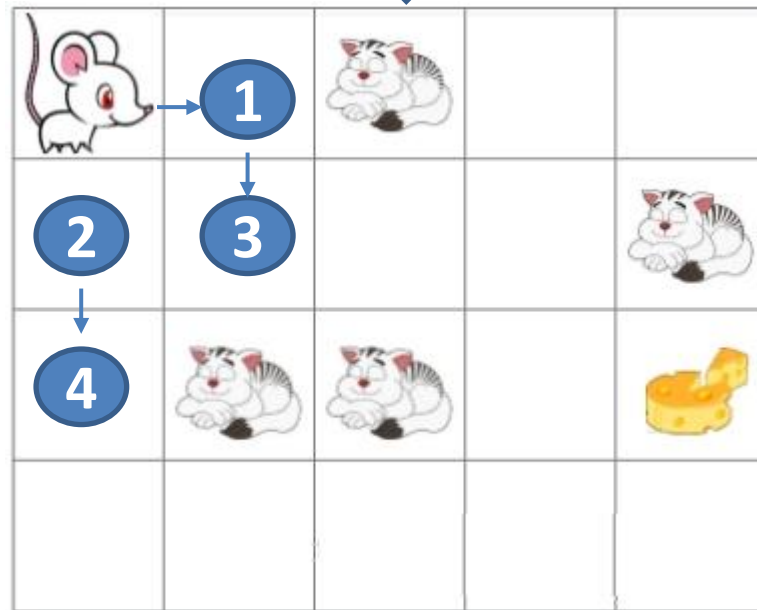
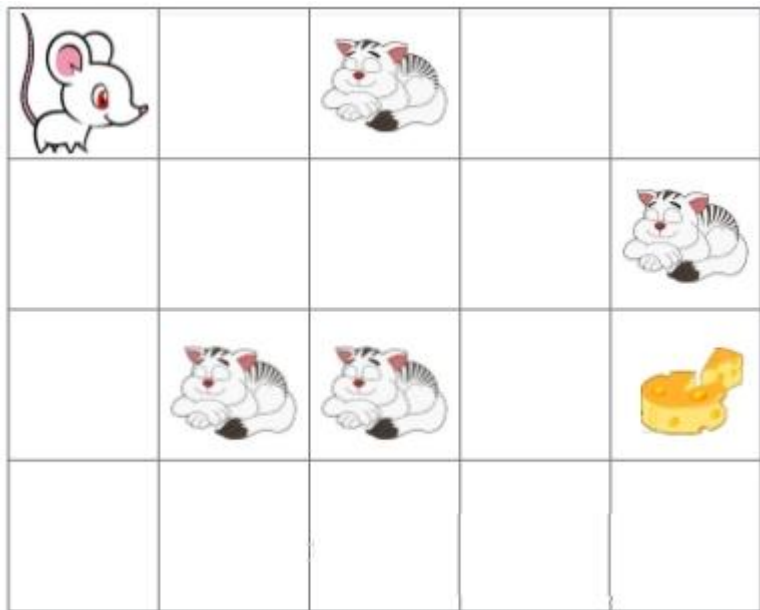
电子老鼠闯迷宫

问题描述:

如下图12×12方格图，找出一条自入口（1，8）到出口（10，7）的最短路径。

- *黑色部分表示建筑物，白色部分是路。
- *电子老鼠可以在路上向上、下、左、右行走。
- *每一步走一个格子。





```

bool bfs(queue<int> q, int end_r, int end_c)
{
    while(!q.empty())                // 只要队列不空
    {
        pop_num = q.front();q.pop();    // 弹出头记录
        //获得头记录对应的位置坐标
        place_r = pop_num/COLUMN;    place_c = pop_num%COLUMN;
        // 以这个记录为起始点, 探索4个方向的可能路径
        int new_num[4];
        for (int i=0; i<4; i++){
            new_num[i] = make_new_place(place_r, place_c, i);

            if (new_num[i] != -1)        // 如果产生的新位置是可行的
            {
                q.push(new_num[i]);    // 将它加入队列尾部
                record_its_father(new_num[i], pop_num); // 记住它的父亲位置
                //如果这个新位置就是终点, 则搜索结束
                if (new_num[i] == end_r*COLUMN + end_c)
                {
                    return true;
                }
            }
        }
    }
    return false;
}

```



```
//根据director方向, 产生位置 (place_r, place_c的新路径
int make_new_place(int place_r, int place_c, int director)
{
    int ret_place = -1; // 记录新产生的位置, 初始值为-1, 标示不能走
    int new_r=place_r, new_c=place_c; // 用来记录新产生位置的坐标
    switch (director)
    {
        case 0:           //right
                           new_c++;
                           break;
        case 1:           // down;
                           new_r++;
                           break;
        case 2:           //left
                           new_c--;
                           break;
        case 3:           //up
                           new_r--;
                           break;
    }
}
```

```
// 检查新产生的位置是否越过了界限
if ((new_r <=0) || (new_r >= ROW) || (new_c <=0) || (new_c >= COLUMN))
{
    ret_place = -1;
}
else
{
    // 对于没有越过界限的位置，只有它内部填充的是0，才能走
    if (maze[new_r][new_c] == 0)
    {
        ret_place = new_r*COLUMN + new_c;
    }
}

return ret_place;

}
```

求解0/1背包问题

分枝限界法和回溯法一样，也不是万能的，主要适合于将求解过程转化为解空间树搜索一类的问题，特别适合于图搜索。

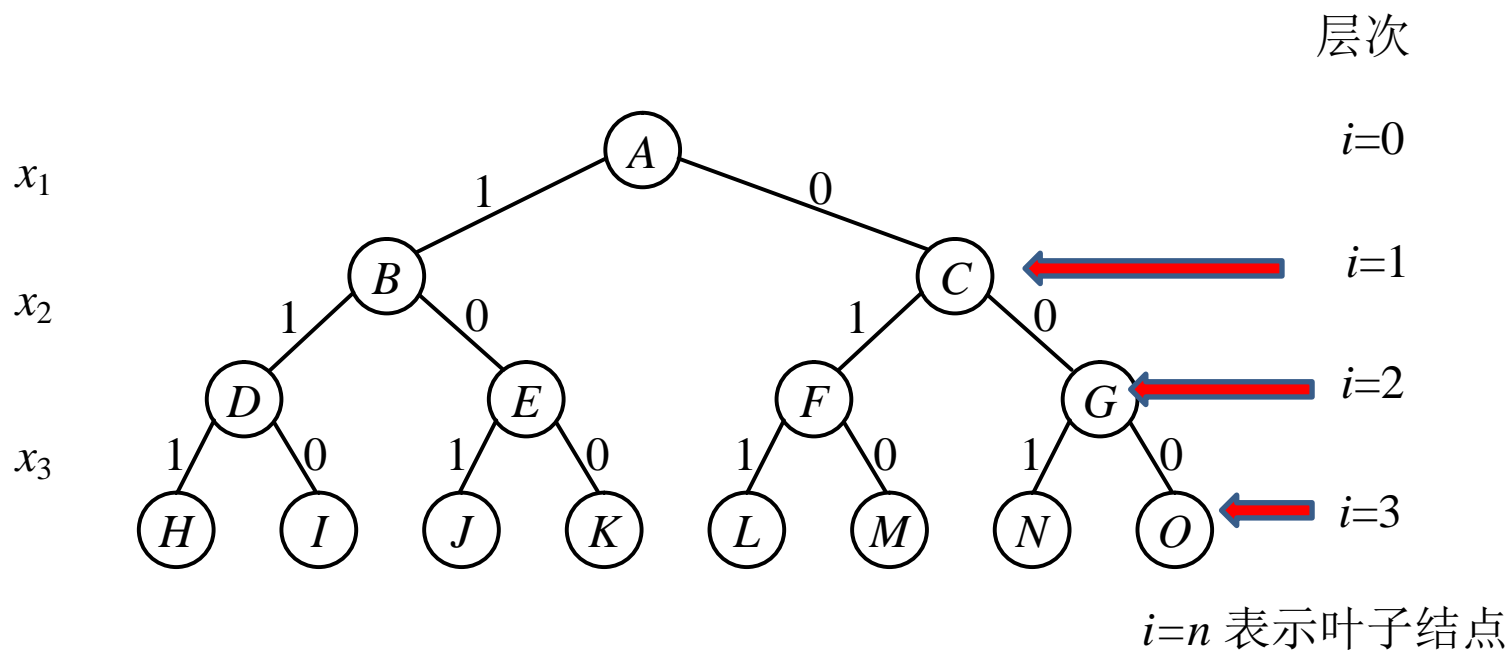
问题描述:

有 n 个重量分别为 $\{w_1, w_2, \dots, w_n\}$ 的物品，它们的价值分别为 $\{v_1, v_2, \dots, v_n\}$ ，给定一个容量为 W 的背包。设计从这些物品中选取一部分物品放入该背包的方案，每个物品要么选中要么不选中，要求选中的物品不仅能够放到背包中，而且具有最大的价值。

例：对下表所示的3个物品求出背包限重 $C=30$ 时的所有解和最佳解。

物品编号	重量 (w)	价值 (v)	单位价值比 (v/w)
1	16	45	3
2	15	25	2
3	15	25	2

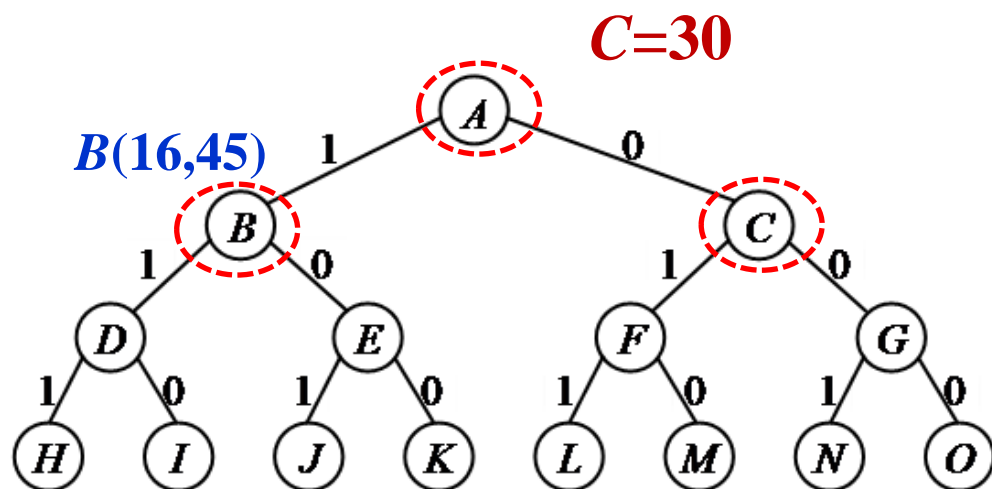
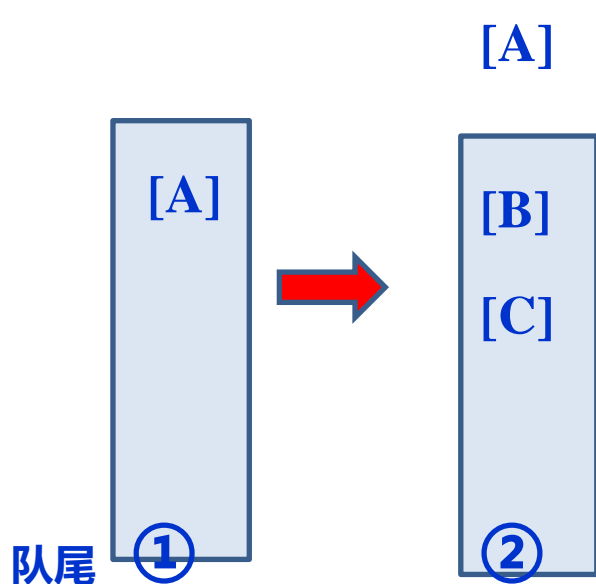
解向量为 $x=(x_1, x_2, x_3)$ ，其解空间树如下图所示。
通过**队列式**和**优先队列式**两种分枝限界法求解该问题。



6.2.1 采用队列式分枝限界法求解

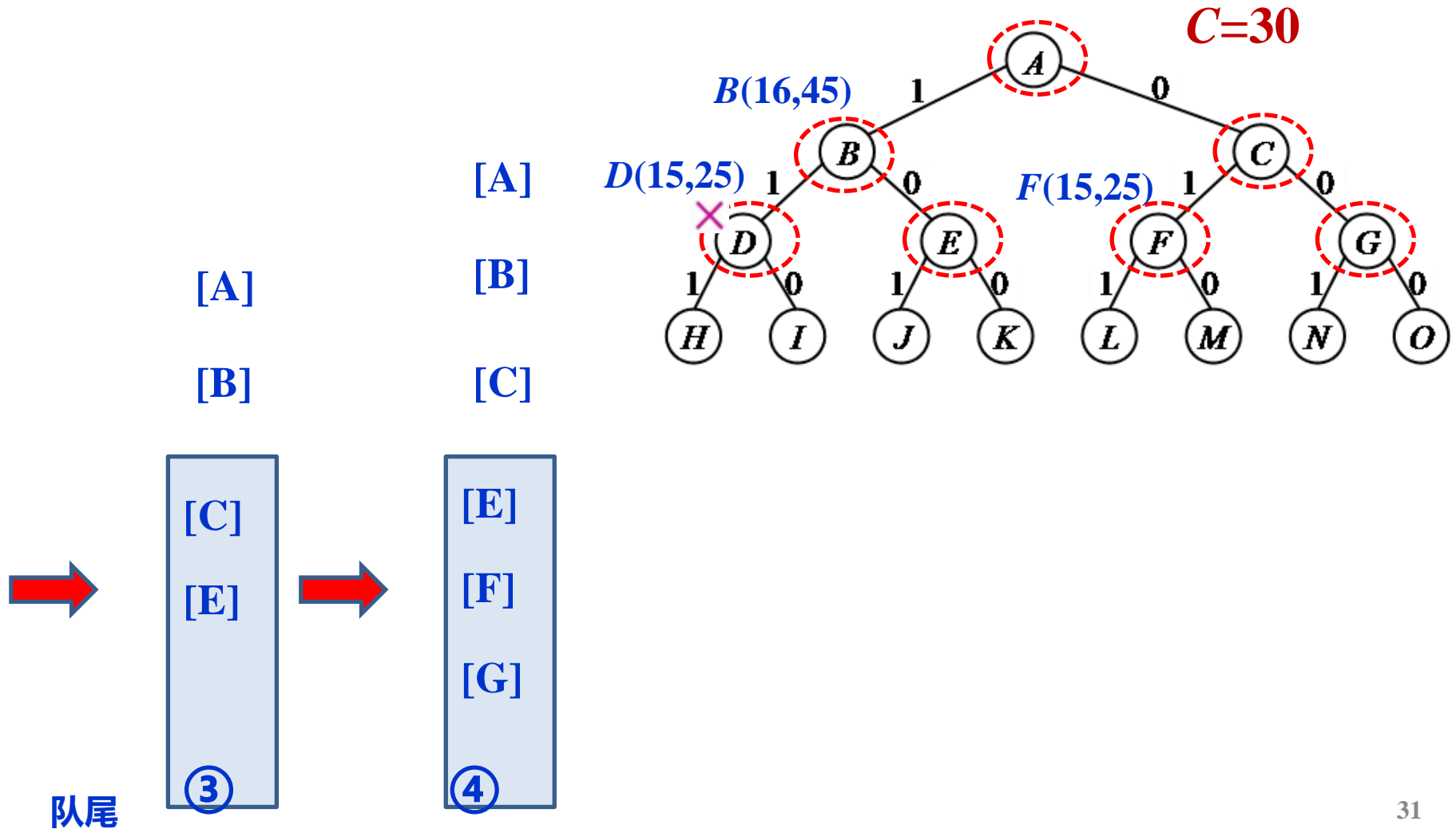
首先不考虑限界问题，用**FIFO**表示队列，初始时，**FIFO=[]**，其求解过程如下：

- ① 根结点 $A(0,0)$ 进队（括号内两个数分别表示此状态下装入背包的重量和价值，初始时均为0），**FIFO=[A]**。
- ② 出队 A ，其孩子结点 $B(16,45)$ 、 $C(0,0)$ 进队，**FIFO=[B,C]**。

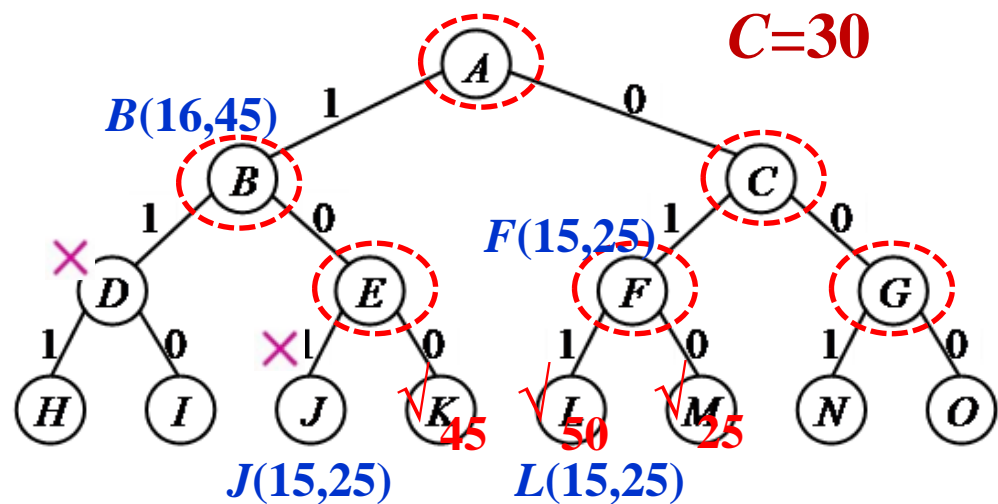


③ 出队 B ，其孩子 $D(15,25)$ 变为死结点、只有孩子 E 进队， $FIFO=[C,E]$ 。

④ 出队 C ，其孩子 $F(15,25)$ 和 $G(0,0)$ 进队， $FIFO=[E,F,G]$ 。



- ⑤ 出队 E ，其孩子 $J(15,25)$ 变为死结点（超重），孩子 K 是叶子结点，总重量 $<W(30)$ ，为一个可行解，总价值为45，对应的解向量为 $(1,0,0)$ ， $FIFO=[F,G]$ 。
- ⑥ 出队 F ，其孩子 $L(15,25)$ 为叶子结点，构成一个可行解，总价值为50，解向量 $= (0,1,1)$ ；其孩子 $M(15,25)$ 为叶子结点，构成一个可行解，总价值为25，解向量 $= (0,1,0)$ 。 $FIFO=[G]$ 。



[A]
[B]
[C]
[E]

[A]
[B]
[C]
[E]
[F]

[F]
[G]

[G]

队尾

⑤

⑥

- ⑦ 出队G，其孩子N(15,25)为叶子结点，构成一个可行解，**总价值为25**，解向量= (0,0,1)；其孩子O(0,0)为叶子结点，构成一个可行解，总价值为0，解向量= (0,0,0)。FIFO=[]。
- ⑧ 因为FIFO=[]，算法结束。

[A]

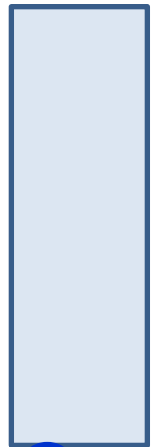
[B]

[C]

[E]

[F]

[G]

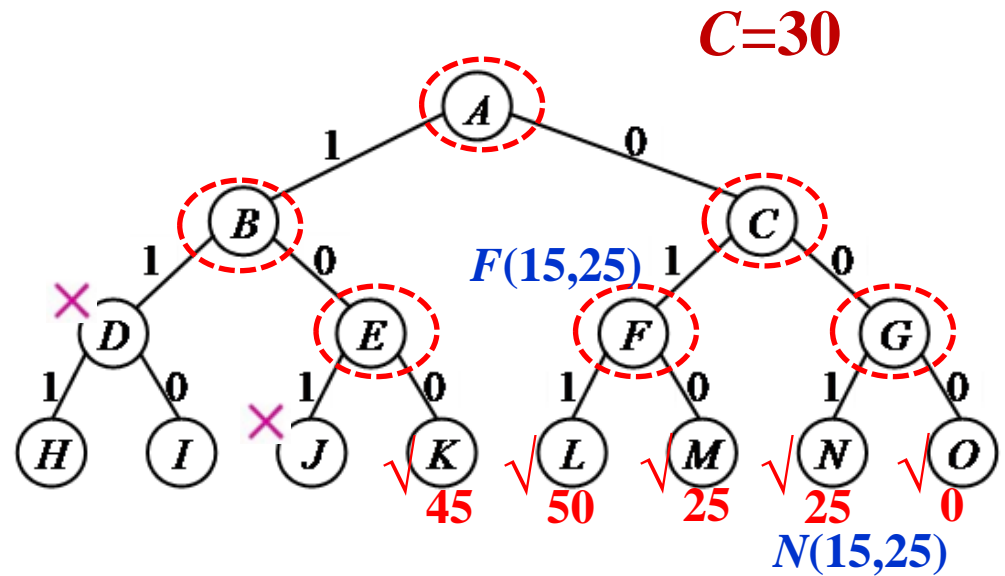


队尾

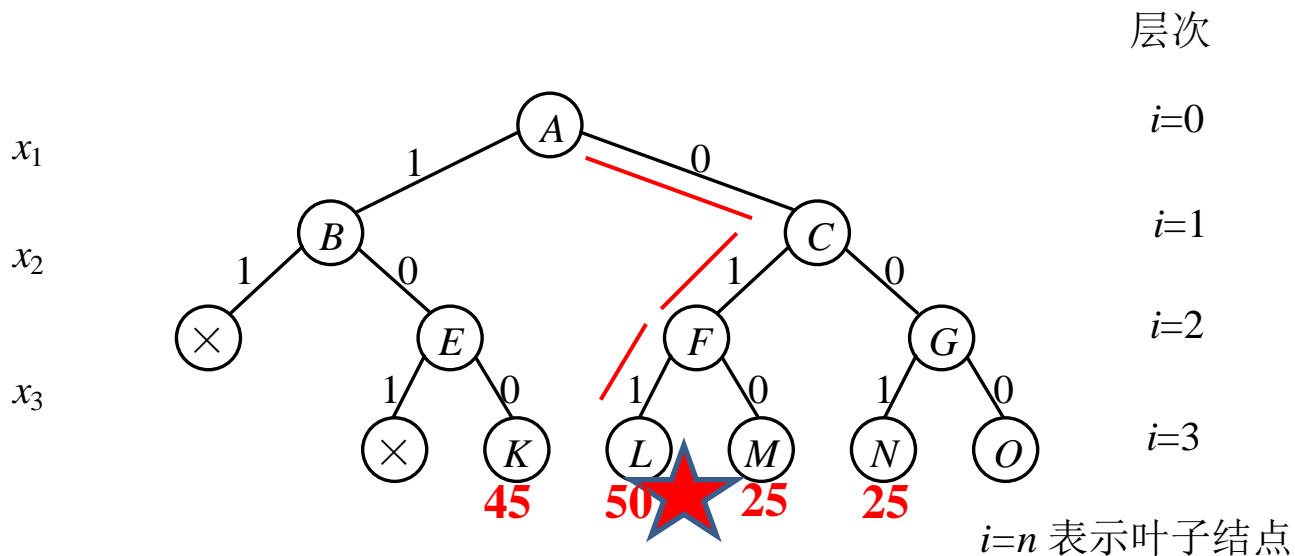
⑦



⑧




对应的搜索空间如图所示，通过所有可行解的总价值比较，得到最优解为 $(0,1,1)$ ，总价值为50。



6.2.2 采用优先队列式分枝限界法求解

采用优先队列式分枝限界法求解必须设计限界函数，因为优先级是以限界函数值为基础的。用大根堆表示活结点表，取优先级为：活结点所获得的价值。

其求解过程与FIFO相似，只是将队列操作改为优先队列的操作。



本部分内容只
需要了解

限界函数设计：由于该问题是求装入背包的最大价值，属于求最大值问题，采用上界。

因为要尽可能地装入最有价值的物品，为此将物品按单位重量的价值递减排序，在重量不超重的条件下先选取单位价值高的物品。

限界函数为：

$$ub = v + (W - w) \times (v_{i+1} / w_{i+1})$$

含义：上界 = 目前价值 + 剩余物品可能的最大价值

例：对下表所示的4个物品（单位价值比按照降序排列），求出背包限重**C=10**时的最佳解。

物品编号	重量 (w)	价值 (v)	单位价值比 (v/w)
1	4	40	10
2	7	42	6
3	5	25	5
4	3	12	4



$w_i = (4, 7, 5, 3)$
 $v_i = (40, 42, 25, 12)$
 $v_i/w_i = (10, 6, 5, 4)$

ub=已装入价值+剩余重量*剩余物品的最高单价

$C=10$

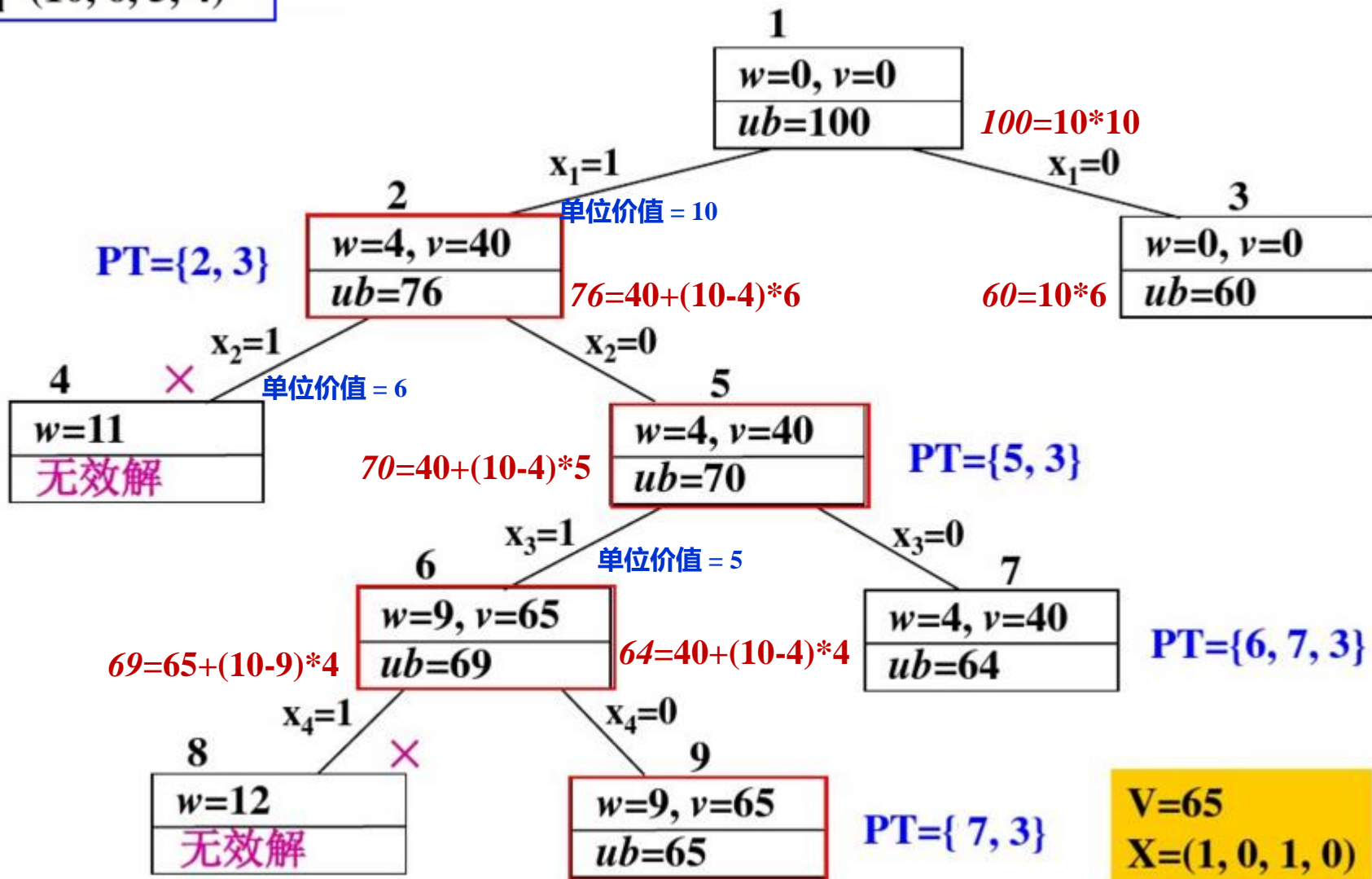
层次

x1

x1

x3

x4



具体的搜索过程如下:

(1) 在根结点1, 没有将任何物品装入背包, 因此, 背包的重量和获得的价值均为0, 根据限界函数计算结点1的目标函数值为 $10 \times 10 = 100$;

(2) 在结点2, 将物品1装入背包, 因此, 背包的重量为4, 获得的价值为40, 目标函数值为 $40 + (10-4) \times 6 = 76$, 将结点2加入待处理结点表PT中; 在结点3, 没有将物品1装入背包, 因此, 背包的重量和获得的价值仍为0, 目标函数值为 $10 \times 6 = 60$, 将结点3加入表PT中;

(3) 在表PT中选取目标函数值取得极大的结点2优先进行搜索;

(4) 在结点4, 将物品2装入背包, 因此, 背包的重量为11, 不满足约束条件, 将结点4丢弃; 在结点5, 没有将物品2装入背包, 因此, 背包的重量和获得的价值与结点2相同, 目标函数值为 $40 + (10-4) \times 5 = 70$, 将结点5加入表PT中;

(5) 在表PT中选取目标函数值取得极大的结点5优先进行搜索;

(6) 在结点6, 将物品3装入背包, 因此, 背包的重量为9, 获得的价值为65, 目标函数值为 $65 + (10-9) \times 4 = 69$, 将结点6加入表PT中; 在结点7, 没有将物品3装入背包, 因此, 背包的重量和获得的价值与结点5相同, 目标函数值为 $40 + (10-4) \times 4 = 64$, 将结点7加入表PT中;

(7) 在表PT中选取目标函数值取得极大的结点6优先进行搜索;

(8) 在结点8, 将物品4装入背包, 因此, 背包的重量为12, 不满足约束条件, 将结点8丢弃; 在结点9, 没有将物品4装入背包, 因此, 背包的重量和获得的价值与结点6相同, 目标函数值为65;

(9) 由于结点9是叶子结点, 同时结点9的目标函数值是表PT中的极大值, 所以, 结点9对应的解即是问题的最优解, 搜索结束。

物品 编号	重量w	价值v	单位价值比	
			(v/w)	值
1	16	45	=45/16	2.8
2	15	25	=25/15	1.7
3	15	25	=25/15	1.7

ub=已装入价值+剩余可以装入物品的
价值：包含可完全装入的各个物品+不
可完全装入的物品进行折算价值

C=30

层次

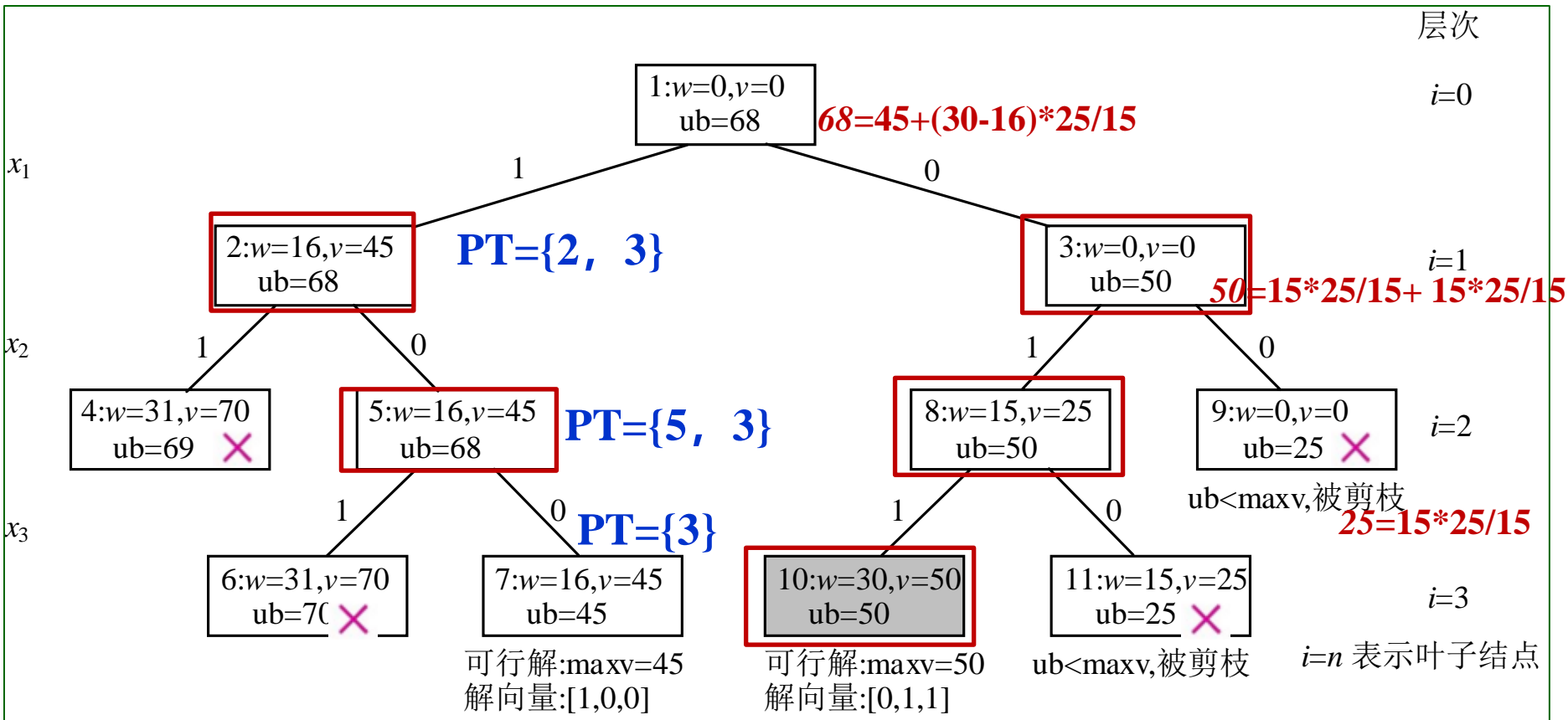
i=0

i=1

i=2

i=3

i=n 表示叶子结点



因为节点3的ub=50>节点7的解45，所以节点3继续扩展查找。

优先队列求解最优解的过程：

(1) 把物体按照价值重量比递减顺序排序；

(2) 上界的根结点进队，将其孩子节点加入优先队列（活节点表）；

(3) 在队不空时循环：出队一个最优结点 e ，检查其左孩子结点并求出其上界，若满足约束条件 ($e.w + w[e.i+1] \leq W$)，将其计入解节点，否则该左孩子结点变成死结点；

(4) 再检查其右孩子结点并求出其上界，若它是可行的（即其上界大于当前已找到可行解的最大总价值 $\max v$ ，否则沿该结点搜索下去不可能找到一个更优的解），则将该右孩子结点进队，否则该右孩子结点被剪枝。

(5) 以上过程循环直到某个叶子节点的目标函数在优先队列中最大。

(6) 算法最后输出最优解向量和最大总价值。

采用**优先队列**，结点的扩展不再是一层一层顺序展开的，而是**按限界函数值的大小选取**的。当物品数较多时，这种效率的提高会更为明显。

算法分析：无论采用队列式分枝限界法还是优先队列式分枝限界法求解0/1背包问题，最坏情况下要搜索整个解空间树，所以**最坏时间和空间复杂度均为 $O(2^n)$** ，其中 n 为物品个数。

八数码问题 (8-puzzle problem)

问题描述:

3×3九宫棋盘，放置数码为1 -8的8个棋牌，剩下一个空格，只能**通过棋牌向空格的移动**来改变棋盘的布局。

要求: 根据给定初始布局（即初始状态）和目标布局（即目标状态），如何移动棋牌才能从初始布局到达目标布局，找到合法的走步序列。

2	8	3
1	6	4
7		5

(初始状态)



1	2	3
8		4
7	6	5

(目标状态)

问题求解:

- 从初始状态到达到目标状态是否有解
- 广度优先搜索算法
- 记录查找路径

从初始状态到达到目标状态是否有解

按照游戏规则，在游戏过程中，棋局的棋子数列的逆序数列的奇偶性不会发生改变，所以有如下结论：

1 初始状态棋局的棋子数列的逆序与目标初始状态棋局的棋子数列的逆序奇偶性不相同，八数码问题无解。

2 初始状态棋局的棋子数列的逆序与目标初始状态棋局的棋子数列的逆序奇偶性相同时，八数码问题有解。

例子：从初始状态到达到目标状态是否有解

在图 (a) 中，从上至下、从左到右排列棋盘里的数字可得出结果为**81567342**。用函数 $F(n)$ 表示数字 n 前面比它大的数字的个数。

① $F(8) = 0$

② $F(1) = 1$

③ $F(5) = 1$

④ $F(6) = 1$

⑤ $F(7) = 1$

⑥ $F(3) = 4$

⑦ $F(4) = 4$

⑧ $F(2) = 6$

8	1	5
6	7	3
4	2	

(a) 逆序状态为偶数

将结果相加，可得图 (a) 图的逆序状态为18，那么它的逆序状态为偶数。

图 (b) 的数字序列为 **76842531**.

- ① F (7) = 0
- ② F (6) = 1
- ③ F (8) = 0
- ④ F (4) = 3
- ⑤ F (2) = 4
- ⑥ F (5) = 3
- ⑦ F (3) = 5
- ⑧ F (1) = 7

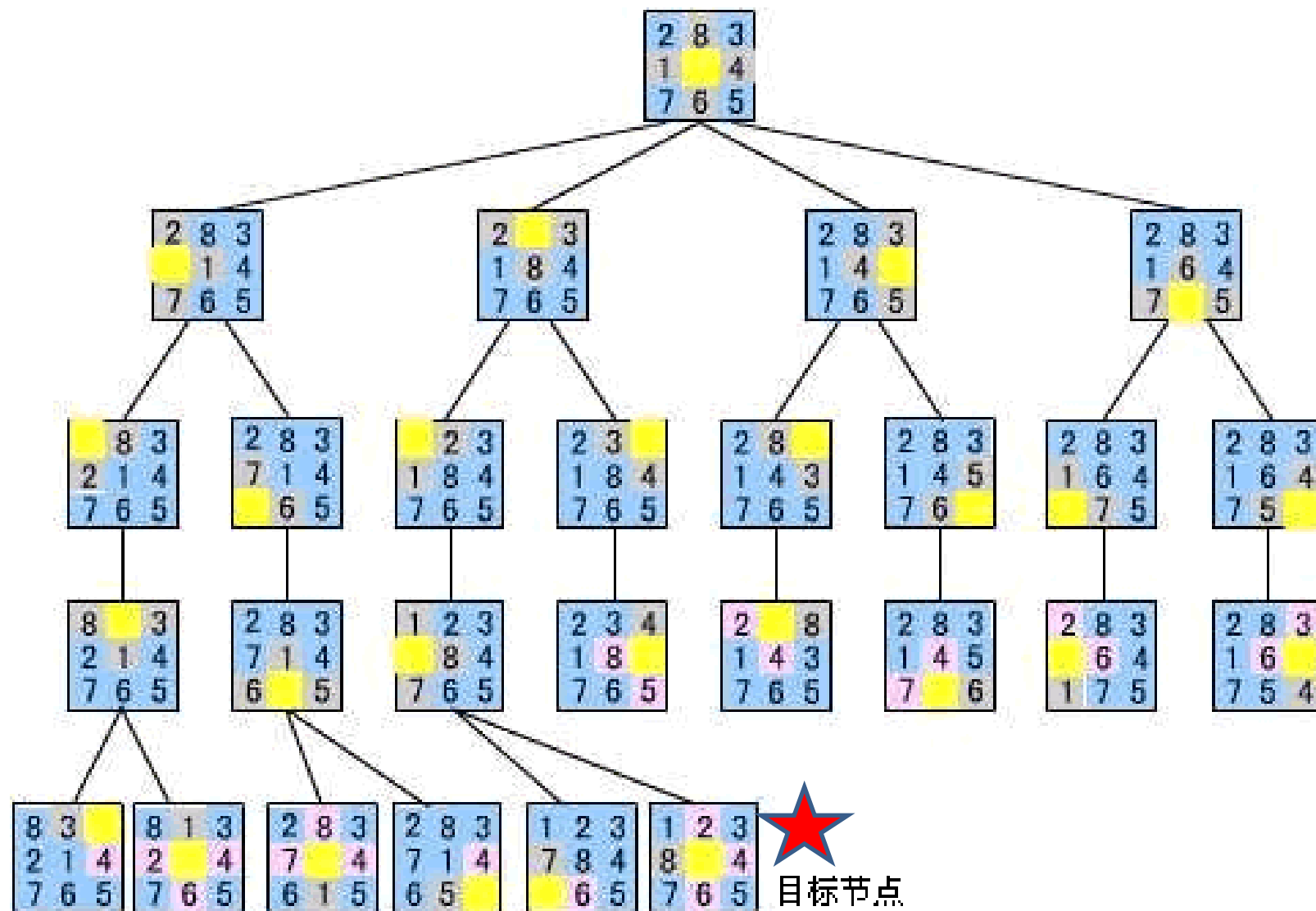
7	6	8
4		2
5	3	1

(b) 逆序状态为奇数

将结果相加，可得图 (b) 的逆序状态值为23，它的逆序状态为奇数。

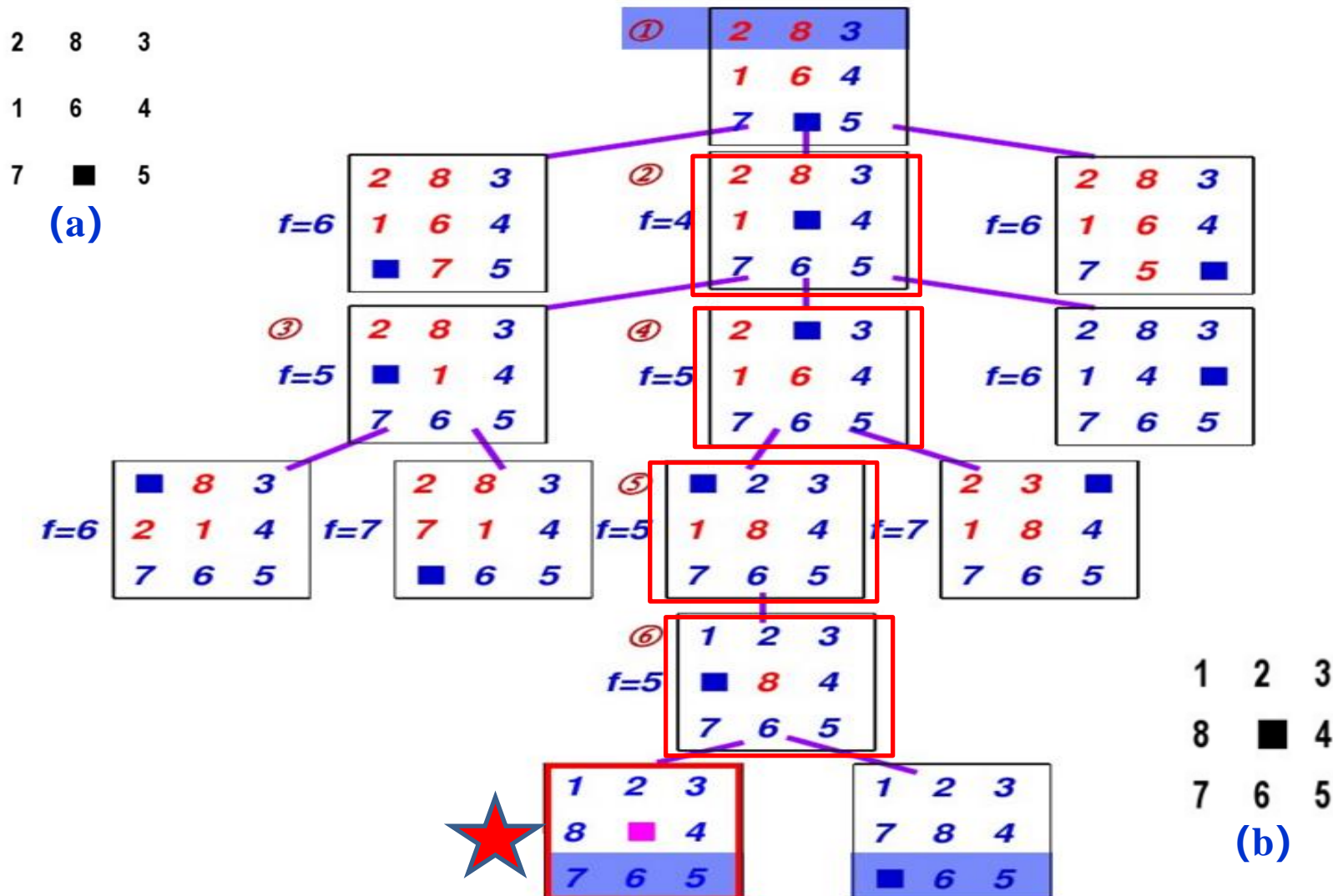
也就是说图中的 (a)、(b) 两种状态互不可达。

使用广度优先的搜索方法（BFS）



优化: 启发式函数设计

$f(n)$ = 数出每个状态与目标状态相比错位的牌数。



数据结构：

- 使用数据队列<queue>保存八数码对应的数字串；
- 使用<map>记录每一步扩展获得的数字串以及其对应的父节点，用于判重，以及记录查找的路径！

map是STL的一个关联容器，它提供一对一的数据处理能力。

map对象是模板类，需要关键字和存储对象两个模板参数，其中第一个参数称为关键字，每个关键字只能在map中出现一次；第二个参数称为该关键字的值，可理解为“{键，值}对”。

例如，学生的“学号”与“姓名”就可以用map进行描述，“学号”对应“关键字”，“姓名”对应“值”，具体的map描述如下：

```
std::map<int, string> mapStudent;
```

<http://www.cplusplus.com/reference/map/map/>

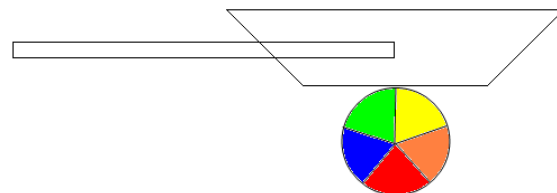
独轮车问题

问题描述:

独轮车的轮子上有5种颜色，每走一格颜色变化一次，独轮车只能往前推，也可以在原地旋转，每走一格，需要一个单位的时间，每转90度需要一个单位的时间，转180度需要两个单位的时间。现给定一个 20×20 的迷宫、一个起点、一个终点和到达终点的颜色，问独轮车最少需要多少时间到达。

说明1: 独轮车只能沿着原来方向前推;

说明2: 起始点包含位置、方向、颜色（某一个颜色），走一格会变化颜色（5种颜色循环变化）；终点包含位置、以及某一个目标颜色。



数据结构:

```
struct colornode
{
    int row;           //该状态的行
    int col;           //该状态的列
    int color;          //该状态的颜色(五种颜色, 0-R, 1-Y, 2-B, 3-W, 4-G)
    int direction;      //该状态的方向(四个方向, 0-W, 1-S, 2-E, 3-N, 西南东北)
    int num;            //该状态的最小步数
};

struct colornode s,t;           //起点, 终点
struct colornode open[2000];    //队列
int head,tail,openlen=2000;
int direct[4][2]={0,-1},{1,0},{0,1},{-1,0}}; //向 左下右上 四个方向转时, 行列的增加值
int a[20][20]={0},n=20;         //a数组表示迷宫, n为迷宫边长
int b[20][20][5][4] = {0};      //b数组表示搜索时的所有状态, 0为未访问, 1为已访问
```

完整代码

<https://www.cnblogs.com/IThaitian/archive/2012/07/16/2593310.html>

//核心代码

```
int search()
{
    struct colornode u, v;
    while(head != tail)
    {
        u = takeoutofopen(); //从队列头取数据
        /*** ----- 向前走可能到目标 ----- ***/
        v = moveahead(u);    //向前走
        if(islegal(v))
        {
            if(isaim(v))
                return(v.num);
            else if(!used(v))
                addtoopen(v);
        }
        /*** ----- 向左，向右转都不会到目标点 ----- ***/
        v=turntoleft(u);    //左转
        if(!used(v))
            addtoopen(v);
        ///////////////////////////////////
        v=turntoright(u);   //右转
        if(!used(v))
            addtoopen(v);
    }
    return 0;              //没有通路
}
```

分油问题

问题描述：

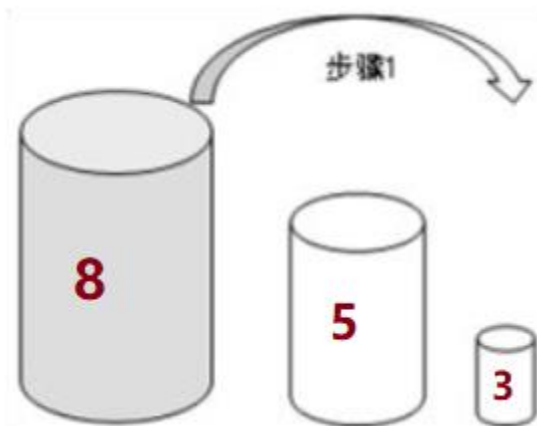
版本1：日本分油问题。有一个装满油的8公升容器，另有一个5公升及3公升的空容器各一个，且三个容器都没有刻度，试将此8公升油分成4公升。

版本2：法国著名数学家泊松年轻时研究过的一道题：

某人有12品脱美酒，想把一半赠人，但没有6品脱的容器，而只有一个8品脱和一个5品脱的容器，问怎样才能把6品脱的酒倒入8品脱的容器中。

版本3：我国的韩信分油问题：韩信遇到两个路人争执不下，原因是两人有装满10斤的油篓和两个3斤、7斤的空油篓，无法平均分出两份，每份5斤油。韩信是如何解决这个难题的？

算法问题：有一酒瓶装有8斤酒，没有量器，只有分别装5斤和3斤的空酒瓶。设计一程序将8斤酒分成两个4斤，并以最少的步骤给出答案。



数据结构:

选择合适的数据结构表示问题状态，用队列和链表的结点来放置瓶内油量有关数据。

如struct node{
 int x //container1;
 int y //container2;
 int z //container3;
 int father; //父节点;
 int steps; //变换到本状态的步数;
};

分析问题时可以把3个油瓶里的油量作为一个向量考虑:
问题的起始状态: (8, 0, 0)。 问题的目标状态: (4, 4, 0)。

注意: 分油过程中，由于油桶上没有刻度，只能将油桶倒满或者倒空。三个油桶盛油的总量始终等于初始时第一个油桶盛满的油量。

广度搜索所有情况：

可操作 情况数	操作	条件		油量的改变(结果)		
				大瓶(x)	中瓶(y)	小瓶(z)
1	x→y	h. x+h. y≤y	不会溢出，全倒过去	0	h. x+h. y	h. z
		否则	x→y会溢出，y倒满	h. x+h. y-y	Y	h. z
2	x→z	h. x+h. z≤z	不会溢出，全倒过去	0	h. y	h. x+h. z
		否则	x→z会溢出，z倒满	h. x+h. z-z	h. y	Z
3	y→x	h. x+h. y≤x	不会溢出，全倒过去	h. x+h. y	0	h. z
		否则	y→x会溢出，x倒满	X	h. x+h. y-y	h. z
4	y→z	h. y+h. z≤z	不会溢出，全倒过去	h. x	0	h. y+h. z
		否则	y→z会溢出，z倒满	h. x	h. y+h. z-z	Z
5	z→x	h. z+h. x≤x	不会溢出，全倒过去	h. x+h. z	h. y	0
		否则	z→x会溢出，x倒满	X	h. y	h. z+h. x-x
6	z→y	h. y+h. z≤z	不会溢出，全倒过去	h. x	h. y+h. z	0
		否则	z→y会溢出，y倒满	h. x	Y	h. z+h. y-y

完整代码参考：

<https://www.cnblogs.com/taozi1115402474/p/8687154.html> (只判断是否可分)

<http://www.mamicode.com/info-detail-2818653.html> (详细记录路径)

搜索过程：

		x->y	x->z	y->x	y->z	z->x	z->y	x->y	x->z	y->x	y->z	z->x	z->y	x->y	x->z	y->x	y->z	z->x	z->y
x8	8	3	5						0		3								5
y5	0	5	0	重	重	重	重	重	5	重	2	重	重	重	重	重	重	重	3
z3	0	0	3						3		3								0
father	-1	0	0	0	0	0	0	1	1	1	1	1	1	2	2	2	2	2	2
steps	0	1	1						2		2								2
下标	0	1	2						3		4								5

head/tail → tail → tail

head → head

tail → tail

tail → tail

tail → tail

节点存放数组：

x8	8	3	5	0	3	5	6	2	6	2	1	7	1	7	4				
y5	0	5	0	5	2	3	2	3	0	5	5	0	4	1	4				
z3	0	0	3	3	3	0	0	3	2	1	2	1	3	0	0				
father	-1	0	0	1	1	2	4	5	6	7	8	9	10	11	12				
steps	0	1	1	2	2	2	3	3	4	4	5	5	6	6	7				
下标	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14				

head: 负责pop节点 //从队列头取数据takeoutofopen();

tail: 负责push节点 //向队列追加数据

思考1: 如何优化减少节点存放数据的空间?

思考2: 如何按照“起点->终点”的变换顺序（正序）进行打印?

总结

- 下表列出了回溯法和分支限界法的一些区别：

方法	对解空间树的搜索方式	存储结点的常用数据结构	结点存储特性	常用应用
回溯法	深度优先搜索	堆栈	活结点的所有可行子结点被遍历后才被从栈中弹出	找出满足约束条件的所有解
分支限界法	广度优先或最小消耗优先搜索	队列、优先队列	每个结点只有一次成为活结点的机会	找出满足约束条件的一个解或特定意义下的最优解

BFS (Breadth First Search) 算法总结

- Breadth First Search (宽度优先、广度优先搜索)
- 确定初始状态;
- 从起点开始层层扩展 (状态转移)
 - 第一层是离起点距离为1的节点 (一个状态直接能达的状态有哪些)
 - 第二层是离起点距离为2的节点
 - 。 。 。
- 本质就是按照层“扩散”，无回退
- 注意点：
 - 节点判重
 - 路径记录
- 关键词：最少、最短

第三回 作业

- 编写加1乘2平方；
- 编写八数码问题，试着用队列(queue)和数组两种方法实现；
- 扩展：八数码问题不使用map，思考其他存储空间小的存放方式（提示： $9!=362880$ ，比876,543,210小）。
- 调试并比较迷宫问题的深搜和广搜解决方法；
- 完成跳马问题
（求出马最少需要多少跳才能从当前位置到达目标位置）；
- 总结分支限界法的算法基本框架！
- 本次作业不需要提交！

小练习1

分支限界法在问题的解空间树中按（____）策略搜索解空间树？

A

广度优先。

B

活结点优先。

C

扩展结点优先。

D

深度优先。

小练习2

常见的两种分支限界法为（____）

- A 广度优先分支限界法与深度优先分支限界法。
- B 队列式（**FIFO**）分支限界法与堆栈式分支限界法。
- C 排列树法与子集树法。
- D 队列式（**FIFO**）分支限界法与优先队列式分支限界法。

END!