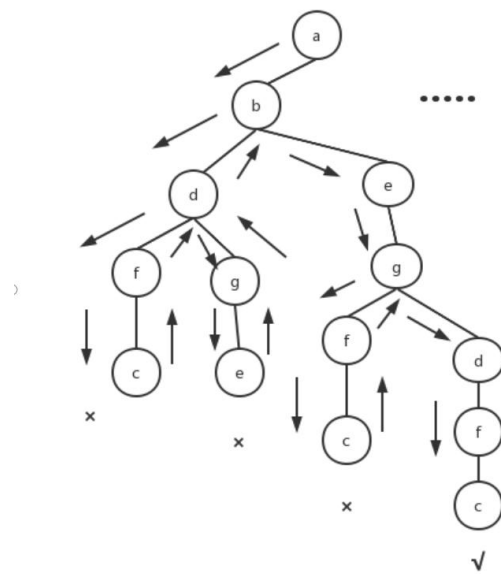
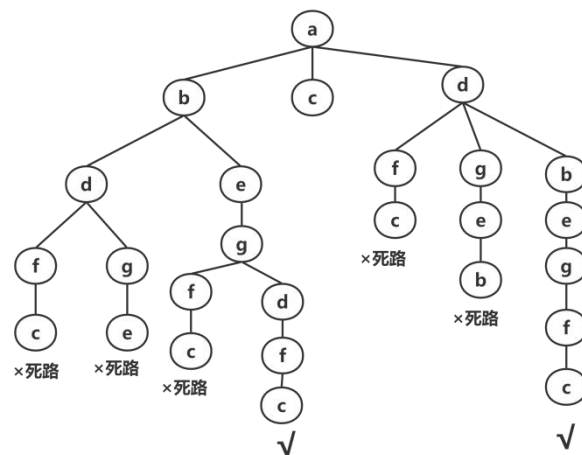


1. (1) 解空间树



(2) 搜索过程



由于对于哈密顿路径 $x_1x_2x_3...x_nx_1$ 来说, $x_1x_nx_{n-1}...x_2x_1$ 也是哈密顿回路, 所以为了减少搜索路径, 可以假设 **b** 在 **c** 之前。

最终得到的合理的路径为:

Abegdfca **adbegfca** 以及他们的相反的路径。

2.

n =	
$F(0) = 0$	
$F(1) = \min\{F(1 - c_1)\} + 1 = 0 + 1 = 1$	
$F(2) = \min\{F(2 - c_1)\} + 1 = 1 + 1 = 2$	
$F(3) = \min\{F(3 - c_1), F(3 - c_2)\} + 1 = 0 + 1 = 1$	
$F(4) = \min\{F(4 - c_1), F(4 - c_2)\} + 1 = 1 + 1 = 2$	+
$F(5) = \min\{F(5 - c_1), F(5 - c_2), F(5 - c_3)\} + 1 = 0 + 1 = 1$	
$F(6) = \min\{F(6 - c_1), F(6 - c_2), F(6 - c_3)\} + 1 = 1 + 1 = 2$	
$F(7) = \min\{F(7 - c_1), F(7 - c_2), F(7 - c_3)\} + 1 = 2 + 1 = 3$	
$F(8) = \min\{F(8 - c_1), F(8 - c_2), F(8 - c_3)\} + 1 = 1 + 1 = 2$	
$F(9) = \min\{F(9 - c_1), F(9 - c_2), F(9 - c_3)\} + 1 = 2 + 1 = 3$	

0	1	2	3	4	5	6	7	8	9
0									
0	1								
0	1	2							
0	1	2	1						
0	1	2	1	2					
0	1	2	1	2	1				
0	1	2	1	2	1	2			
0	1	2	1	2	1	2	3		
0	1	2	1	2	1	2	3	2	
0	1	2	1	2	1	2	3	2	3

$F(9)$ 的最优来源为 $F(9-5)$ 即 $F(4)$ ，而 $F(4)$ 的最优来源为 $F(4-3)$ 即 $F(1)$ ，而 $F(1)$ 来源为 1 块 1 元硬币。所以 9 元硬币应该由 1,3,5 各一块组成，且数量最少为 3 块。

伪代码：

MinCoin(n)

$F(0) \leftarrow 0;$

for $i \leftarrow 1$ **to** n **do** //选出 $F(i-c)$ 中最小的

temp $\leftarrow +\infty$

if $i \geq 1$ **then**

temp = min($F(i-1)$)+1,temp);

if $i \geq 3$ **then**

temp = min($F(i-3)$)+1,temp);

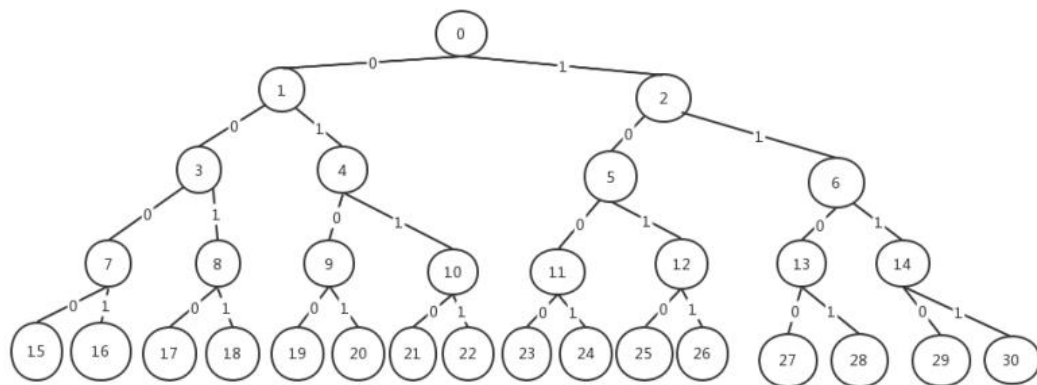
if $i \geq 5$ **then**

temp = min($F(i-5)$)+1,temp);

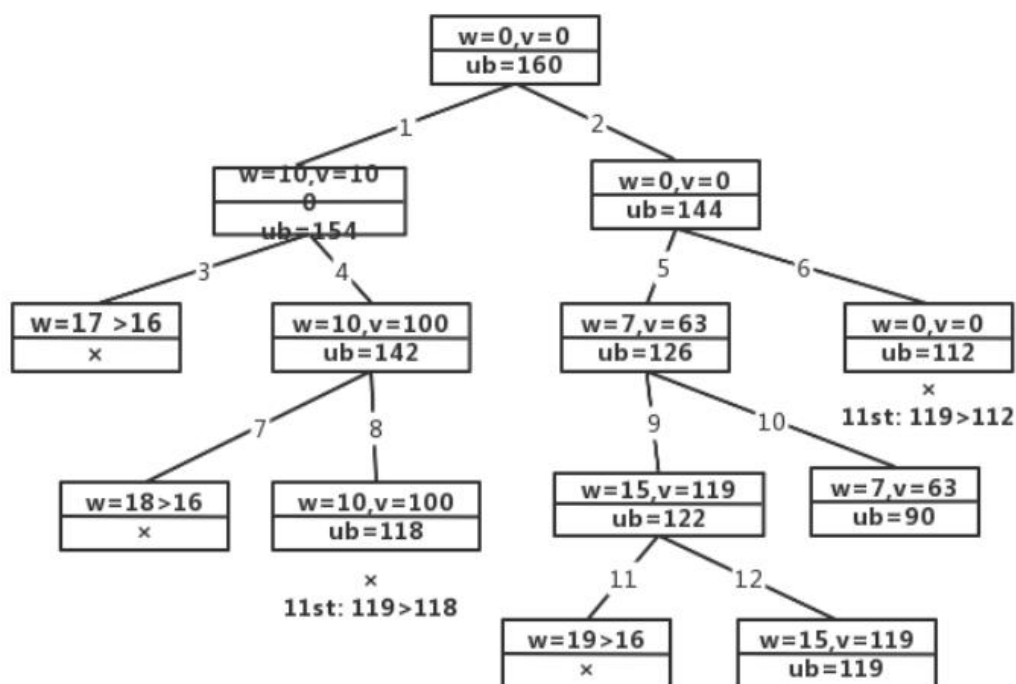
$F(i) = \text{temp};$

return $F(n);$

3. (1) 解空间树



(2) 搜索过程



搜索的时候按照优先级队列对队列中的元素进行优先级查找。用来决定优先级的是每个状态时的最终上界。上界的计算公式为： $ub = v + (W - w)(v_{i+1}/v_i)$

首先从 0 节点开始，计算 1,2 节点：

$$ub_1 = 100 + 6 \times 9 = 154$$

$$ub_2 = 0 + 16 \times 9 = 144$$

这时候队列中为 1、2，因为 1 节点上界更大，所以从 1 开始出列。计算 3、4 节点：

$$w_3 = 17 > 16$$

$$ub_4 = 100 + 6 \times 7 = 142$$

3 节点不满足限界函数，4 节点入队列。因为 $142 < 144$ ，所以 2 节点出队列：

$$ub_5 = 63 + 7 \times 9 = 126$$

$$ub_6 = 0 + 16 \times 9 = 144$$

5、6 满足限界条件，入队列，由于 4 队列最大，所以出队列：

$$w_7 = 18 > 16$$

$$ub_8 = 100 + 6 \times 3 = 118$$

7 不满足限界条件，8 入队列。此时 5 最大，出队列：

$$ub_9 = 122$$

$$ub_{10} = 90$$

此时 9,10 入队列，9 的上界最大：

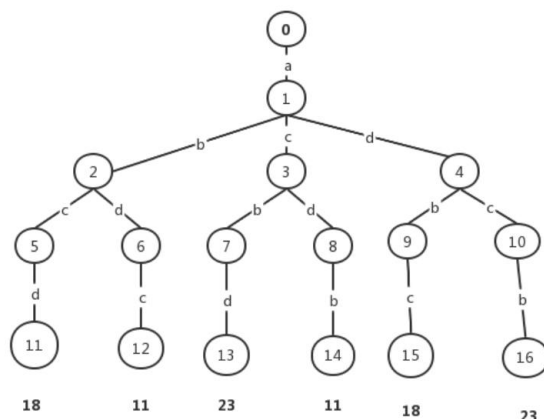
$$w_{11} = 19 > 16$$

$$ub_{12} = 119$$

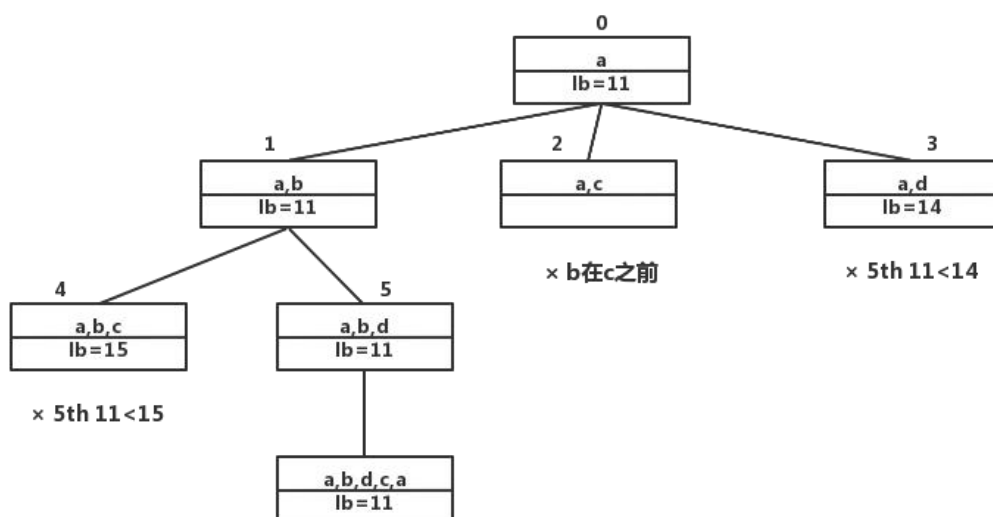
这时的结果已经是最优的了，大于了其他支路的最大上界。

所以选择第 2、3 个物品可以达到最大价值。

4. (1) 解空间树



(2) 搜索过程



在搜索时，由于对于最佳路径 $x_1x_2x_3...x_nx_1$ 来说， $x_1x_nx_{n-1}...x_2x_1$ 也是最佳路径，所以为了减少搜索路径，可以假设 **b** 在 **c** 之前。

搜索的时候按照优先级队列对队列中的元素进行优先级查找。用来决定优先级的是每个状态时的最终下界。下界的计算公式为：

$$lb = (2 \sum_{i=1}^{k-1} c[r_i][r_{i+1}] + \sum_{r_i \in U} r_i \text{ 行不在路径上的最小元素} + \sum_{r_j \notin U} r_j \text{ 行最小的两个元素}) / 2$$

对于 0 号起始： $lb_0 = [(2 + 5) + (2 + 3) + (1 + 3) + (1 + 5)] / 2 = 11$ 入队列

从 a,b,c 中继续选择，由于定义 **b** 在 **c** 之前，所以 **c** 可以忽略。

$$lb_1 = [(2 + 5) + (2 + 3) + (1 + 3) + (1 + 5)] / 2 = 11$$

$$lb_3 = [(2 + 7) + (2 + 3) + (1 + 7) + (1 + 5)] / 2 = 14$$

按照下界的最小值，让 1 号出队列：

$$lb_4 = [(2 + 5) + (2 + 8) + (1 + 3) + (1 + 8)]/2 = 15$$

$$lb_5 = [(2 + 5) + (2 + 3) + (1 + 3) + (1 + 5)]/2 = 11$$

由于已经选择了 n-1 个节点，所以可以直接计算出 abd 的路径 abdca 的值为 11，小于任何其他可能路径的下界。所以搜索结束，最小值为 11 路径为 abdca 或 acdba。

5. 采取回溯的算法，遍历各种情况。将插孔的位置利用数组来表示。因为总的插孔有 15 个，所以可以使用一个 5×5 的矩阵中的下三角来代表插孔。则这个矩阵可以用一个二维数组 C 来表示。

在这个数组中，对任意一个位置，若 $i > j$ 则 $C[i, j] = -1$ ，因为上三角是无效的。对于每一个插孔（ $i \leq j$ ），如果某一个插孔 $C[i, j]$ 里面插上了插棒，则 $C[i, j] = 1$ 否则 $C[i, j] = 0$ 。初始化数组：

$$\begin{bmatrix} 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & -1 \\ 1 & 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 & -1 \\ 1 & 1 & 0 & 1 & 1 \end{bmatrix}$$

插棒在棋盘上可以向左右，向左上左下，向右上右下进行移动，所以在数组中，则是在满足条件的情况下向上下方，左右方向，右下左上方六个方向进行移动。

对于每一个插棒的移动，一定能消去一根插棒，所以不论是第一问还是第二问，若该问题有解则一定会移动 13 次。而每一次移动后都会有不同的插棒对象可以获得移动机会。所以我们可以直接采取直接遍历加上条件判断的方法。如果一个矩阵具有可以移动的插棒，则让所有可以移动的插棒进行所有可能情况的移动，并且每一个插棒移动后都要恢复现场，方便后续可能发生的移动。

设置一个全局变量 $C[5,5]$

伪代码如下：

Test($C[5,5]$)

```

for  $i \leftarrow 1$  to 5 do
    for  $j \leftarrow 1$  to  $i$  do
        if  $C[i, j] = 1$ 
            counter++;
if counter = 1
    return True;
else
    return False;

```

FindSolution($C[5,5]$)

```

if Test( $C[5,5]$ ) = True then
    return True;
else
    for  $i \leftarrow 1$  to 5 do
         $j \leftarrow 1$ ;
        while  $i \geq j$  and  $C[i, j] = 1$ 
            if  $i - 2 > 0$  and  $C[i - 1, j] = 1$  and  $C[i - 2, j] = 0$  then //可以向左走
                 $C[i, j] \leftarrow 0$ ;  $C[i - 1, j] \leftarrow 0$ ;  $C[i - 2, j] \leftarrow 1$ ;
                if FindSolution( $C[5,5]$ ) = True
                    Print( $C[5,5]$ );
                    return True;

```

```

    C[i,j] ← 1; C[i-1,j] ← 1; C[i-2,j] ← 0;
else if i+2>0 and C[i+1,j] = 1 and C[i+2,j] = 0 then    //可以向右边走
    C[i,j] ← 0; C[i-1,j] ← 0; C[i-2,j] ← 1;
    if FindSolution(C[5,5]) = True
        Print(C[5,5]);
        return True;
    C[i,j] ← 1; C[i-1,j] ← 1; C[i-2,j] ← 0;
else if j-2>0 and C[i,j-1] = 1 and C[i,j-2] = 0 then    //可以向上边走
    C[i,j] ← 0; C[i,j-1] ← 0; C[i,j-2] ← 1;
    if FindSolution(C[5,5]) = True
        Print(C[5,5]);
        return True;
    C[i,j] ← 1; C[i,j-1] ← 1; C[i,j-2] ← 0;
else if j-2>0 and C[i,j-1] = 1 and C[i,j-2] = 0 then    //可以向下边走
    C[i,j] ← 0; C[i,j-1] ← 0; C[i,j-2] ← 1;
    if FindSolution(C[5,5]) = True
        Print(C[5,5]);
        return True;
    C[i,j] ← 1; C[i,j-1] ← 1; C[i,j-2] ← 0;
else if j-1>0 and i-1>0 and C[i-1,j-1] = 1 and C[i-2,j-2] = 0 then
                                                //可以向左上走
    C[i,j] ← 0; C[i-1,j-1] ← 0; C[i-2,j-2] ← 1;
    if FindSolution(C[5,5]) = True
        Print(C[5,5]);
        return True;
    C[i,j] ← 1; C[i-1,j-1] ← 1; C[i-1,j-2] ← 0;
else if j+1>0 and i+1>0 and C[i+1,j+1] = 1 and C[i+2,j+2] = 0 then
                                                //可以向右下走
    C[i,j] ← 0; C[i+1,j+1] ← 0; C[i+2,j+2] ← 1;
    if FindSolution(C[5,5]) = True
        Print(C[5,5]);
        return True;
    C[i,j] ← 1; C[i+1,j+1] ← 1; C[i+1,j+2] ← 0;
j++;

```

(1) 经过以上代码运行后，最终能够打印出一条历经 13 个步骤消去 13 个插棒且位置不限的转移方法。

(2) 若是想要插棒落在最初的插孔上，只需修改

```

if Test(C[5,5]) = True then
    return True;

```

为

```

if Test(C[5,5]) = True and C[5,3] = 1 then
    return True;

```

这样最后输出的路径即为剩下的插棒最终要落在最初的空孔上的消去 13 个插棒的路径。

编程题:

思路: 采用回溯法, 从每一个非 0 节点开始, 对其旁边的节点进行遍历。具体来说, 就是采用深度优先的方法, 从非 0 一个节点开始, 记录当前节点值, 并通过递归得到上下左右各个方向的所能采取到的黄金值, 最后将当前节点值与上下左右方向中最大的值相加并返回给上一层。直到最顶层后, 返回从该节点能走通的路径中黄金最多的数量值。

在对每一个非 0 节点进行遍历的时候, 会用一个 max 来表示目前走通的路径中最多的黄金总额, 并且会与接下来所生成的最大黄金总额比较。直到运行结束取到最大值。


```
int dfs(vector<vector<int>> &grid, int i, int j)
{
    int up = 0;
    int down = 0;
    int left = 0;
    int right = 0;
    if (grid[i][j] != 0) //四个方向行走
    {
        int t = grid[i][j];
        grid[i][j] = 0;
        if (i - 1 >= 0 && grid[i - 1][j] != 0) //向上走
            up = dfs(grid, i - 1, j);
        if (j - 1 >= 0 && grid[i][j - 1] != 0) //向左走
            left = dfs(grid, i, j - 1);
        if (i + 1 < grid[0].size() && grid[i + 1][j] != 0) //向右走
            right = dfs(grid, i + 1, j);
        if (j + 1 < grid.size() && grid[i][j + 1] != 0) //向下走
            down = dfs(grid, i, j + 1);
        grid[i][j] = t;
        return t + max(up, max(down, max(left, right)));
    }
    return 0;
}
```

```
int getMaximumGold(vector<vector<int>>& grid)
{
    int col = grid[0].size();
    int row = grid.size();
    int max = 0;
    int maxtemp = 0;
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            maxtemp = dfs(grid, i, j);
            if (maxtemp > max)
                max = maxtemp;
        }
    }
    return max;
}
```


时间复杂度分析: 对于每一个节点, 都需要将地图遍历一次, 共需 $O(mn)$, 而最坏情况下共有 mn 个节点, 所以总的时间复杂度为: $O(m^2n^2)$ 。

空间复杂度分析: 对于每一次递归, 都需要 5 个局部变量, 而每一次都需要将地图遍历一遍, 所以最坏情况下共需 $O(mn)$ 的空间。

示例：

 Microsoft Visual Studio 调试控制台

```
1 0 7
2 0 6
3 4 5
0 3 0
28
```

 Microsoft Visual Studio 调试控制台

```
0 6 0
5 8 7
0 9 0
24
```