

搜索过程如下所示，结果树和解有：(acfgedbd)(acfdgebd)(adbeqfcd)(abegdfcd)

搜索过程：① $a \rightarrow c \rightarrow f \rightarrow d \rightarrow b \rightarrow g \rightarrow e$ 回溯到 ④ $b \rightarrow e \rightarrow g$ 回溯到 ⑦ $d \rightarrow g \rightarrow b \rightarrow e$ 回溯到 ⑩
 $g \rightarrow e \rightarrow b$ 回溯到 ⑦ $fg \rightarrow d \rightarrow b \rightarrow e$ 回溯到 ⑩ $g \rightarrow b \rightarrow d$ 回溯到 ⑩ $b \rightarrow e$ 回溯到 ⑩
 ⑩ $g \rightarrow e \rightarrow b \rightarrow d$ 求得第一个解 (acfgedbd)

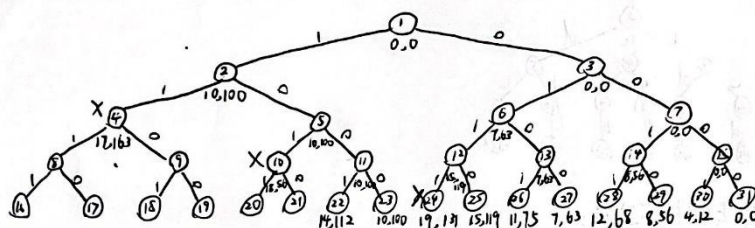
2. 定义数组 $coin[n]$ 其中 $coin[i]$ 表示组成面值为 i 所需要的最少硬币数

定义二维动态数组 $sum[n][1]$ 其中 $sum[i][1]$ 内存组成面值 i 所需最少硬币数的硬币组合，内容为一个数组，则代码如下：

```
// 初始化
coin[1] = 1; sum[1][0] = (1, 0, 0);
coin[3] = 1; sum[3][0] = (0, 1, 0);
coin[5] = 1; sum[5][0] = (0, 0, 1);
// 动态计算 2 和 4 的
coin[2] = 2; sum[2][0] = (2, 0, 0);
coin[4] = 2; sum[4][0] = (1, 1, 0);
// DP
for (int i = 6; i < n; ++i) {
    // 找最小值
    int coinMin = min(coin[i-1], coin[i-3], coin[i-5]);
    // 更新 coin[i]
    coin[i] = coinMin + 1;
    // 找解法：
    if (coinMin == coin[i-1]) { for (j = 0; j < sum[i-1].size(); ++j) sum[i].insert(sum[i-1][j] + (1, 0, 0)); }
    if (coinMin == coin[i-3]) { for (j = 0; j < sum[i-3].size(); ++j) sum[i].insert(sum[i-3][j] + (0, 1, 0)); }
    if (coinMin == coin[i-5]) { for (j = 0; j < sum[i-5].size(); ++j) sum[i].insert(sum[i-5][j] + (0, 0, 1)); }
}
return coin[n], sum[n]; // 返回所求值，硬币的解为值 1, 3, 5 分硬币：(1, 1, 1)
```

算法运行过程：对于每一个面值为 i 的 $coin[i]$ ，在 $i-1, i-3, i-5$ 中找最小值，分别代表在这些解中加和值为 1 的币、3 的币、5 的币，共 5 个硬币。
 若 $coin[i-1]$ 最小，则从 $coin[i-1]$ 所有解中加上 (1, 0, 0) 表示加入 1 元硬币并添入 $sum[i]$ 中。
 若 $coin[i-3]$ 最小， $coin[i-5]$ 最小，同理， $sum[i]$ 可以有多个解值。

3. 解空间树:

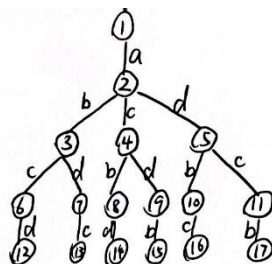


用FIFO队列分支限界法求解过程:

扩展结点	活结点	队列(可行结点)	可行解(叶结点)	解值
1	2, 3	2, 3		
2	4, 5 (4死结点)	3, 5		
3	6, 7	5, 6, 7		
5	10, 11 (10死结点)	6, 7, 11		
6	12, 13	7, 11, 12, 13		
7	14, 15	11, 12, 13, 14, 15		
11	22, 23	12, 13, 14, 15, 22, 23	22, 23	112, 100
12	24, 25 (24死结点)	13, 14, 15, 22, 23, 24, 25	25	119
13	26, 27	14, 15	26, 27	75, 63
14	28, 29	15	28, 29	68, 56
15	30, 31	ϕ	30, 31	12, 0

最优解为25, 即(0, 1, 1, 0) 解值为119

4. 解空间树为:



利用优先队列式分支限界法的搜索过程:

$$\frac{2}{9} \quad 3^2 \quad 4^5 \quad 5^7$$
$$2 \quad 3^2 \quad 4^5 \quad 5^7 \quad 6^{10} \quad 7^5$$
$$2^0 \quad 3^2 \quad 4^5 \quad 5^7 \quad 6^{10} \quad 7^5 \quad 8^{13} \quad 9^6$$
$$2 \quad 3^2 \quad 4^5 \quad 5^7 \quad 6^{10} \quad 7^5 \quad 8^{13} \quad 9^6 \quad 13^{11}$$
$$2 \quad 3^2 \quad 4^5 \quad 5^7 \quad 6^{10} \quad 7^5 \quad 8^{13} \quad 9^6 \quad 13^{11} \quad (15^{11})$$
$$2 \quad 3^2 \quad 4^5 \quad 5^7 \quad 6^{10} \quad 7^5 \quad 8^{13} \quad 9^6 \quad \underline{13^{11}} \quad \underline{15^{11}} \quad 10^{10} \quad 11^8$$
$$2 \quad 3^2 \quad 4^5 \quad 5^7 \quad 6^{10} \quad 7^5 \quad 8^3 \quad 9^6 \quad \underline{13^{11}} \quad \underline{15^{11}} \quad 10^{10} \quad 11^8 \quad \cancel{17}$$
$$\begin{array}{ccccccccccccccc} 2 & 3^2 & 4^5 & 5^7 & 6^{10} & 7^5 & 8^{13} & 9^6 & 13^{11} & 15^{11} & 10^{10} & 11^8 & 12 \\ 2 & 3^2 & 4^5 & 5^7 & 6^{10} & 7^5 & 8^{13} & 9^6 & 13^{11} & 15^{11} & 10^{10} & 11^8 & 16 \end{array}$$

∴ 最优解为 $a \rightarrow b \rightarrow d \rightarrow c \rightarrow a$ 最小花费为 11
和 $a \rightarrow c \rightarrow d \rightarrow b \rightarrow a$

5.

主要思路: 由于该图是旋转对称的, 所以位于同一三角区起始点的解法具有共同之处, 观察可得共有4个不同的起始状态, 并且每次移动会消除一个棒, 故最终解法必有13步, 在用回溯法求解过程中可以通过判断当前状态与之前记录过的某一无解状态在翻转旋转后能否从而进行剪枝.

伪代码如下:

```
// 初始化
Point initialPos(x, y); // 最初空间
vector<Stepresult> result[1]; // 结果链表
vector<Step> step; // 步骤栈
vector<State> falseState[1]; // 无解状态集合
bool qB = false; // 判断结果集合状态B: 最终点与起始点重合.
step.push(state_0);
while (step) {
    vector<State> board = step.pop(); // 取出栈顶状态
    if (board.num == 0) { // 状态只有一个棒
        if (board.lastMove == initialPos) qB = true; // 符合问题B.
        result.add(step, qB); // 将结果添加入result.
    } else {
        if (board.try() == false) { // 当 board 状态无法到达下一步
            falseState.push(board); // 无解.
        } else {
            for (int i = 0; i < board.num; ++i) { // 遍历每一个棒
                vector<stick> t = board.stick[i];
                if (canJump(board, t)) { // t 可以行动
                    vector<State> board_next = Jump(board, t); // 行动后的状态
                    if (board judge(board_next, falseState)) continue; // 符合解则剪枝
                    else result.add step.push(board_next); // 加入step.
                }
            }
        }
    }
}
return result; // 返回结果.
```


编程题

算法思路：使用深度优先搜索（DFS）解决。每当矿工进入一个单元，就会收集该单元格中的所有黄金，并从该位置继续向上下左右四个方向遍历。我们可以递归地进行深度优先搜索，并在每个递归步骤中更新已经遍历过的单元格。具体地，我们从每个有黄金的单元格出发，将其视为起点，并在每个单元格中寻找可以遍历的下一个单元格。在搜索过程中，我们需要将已经遍历过的单元格标记为已访问，可以将其置为 0 表示已访问。同时，我们需要维护当前遍历的黄金数量的和，以及在递归回溯时要将已访问的单元格标记为未访问。

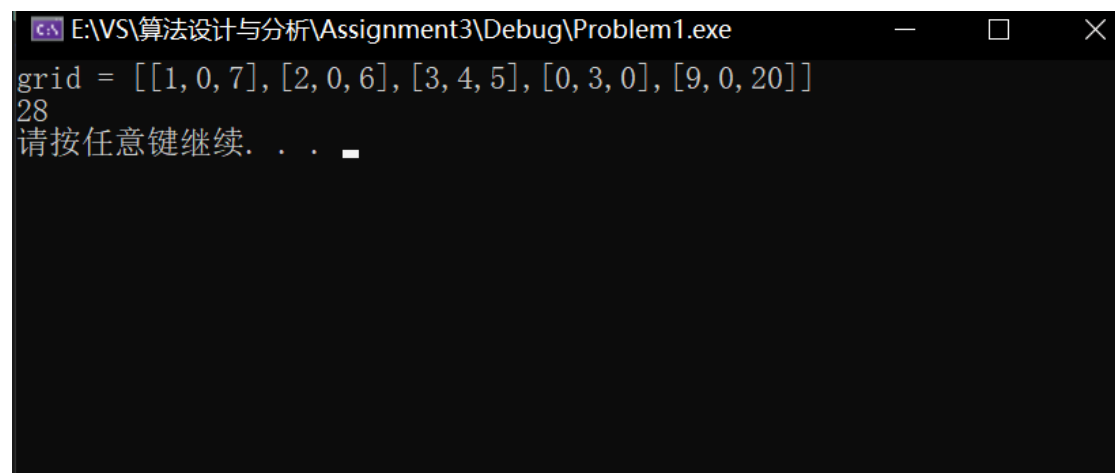
时间复杂度：每个单元格最多只被访问一次，所以总时间复杂度为 $O(mn(4^{(m*n)}))$ ，其中 m 和 n 是网格的行数和列数。

空间复杂度：递归树的深度最多为 mn ，所以空间复杂度为 $O(mn)$ 。

```
int getMaximumGold(vector<vector<int>> &grid) {
    int ans = 0;
    for (int i = 0; i < grid.size(); ++i) {
        for (int j = 0; j < grid[i].size(); ++j) {
            if (grid[i][j] != 0) {
                ans = max(ans, findGold(grid, i, j));
            }
        }
    }
    return ans;
}
```

```
int findGold(vector<vector<int>> &grid, int x, int y) {
    int sum = 0;
    int dir[4][2] = { {-1,0},{0,-1},{1,0},{0,1} };
    int temp = grid[x][y];
    grid[x][y] = 0;
    for (int i = 0; i < 4; ++i) {
        int u = x + dir[i][0], v = y + dir[i][1];
        if (u >= 0 && u < grid.size() && v >= 0 && v < grid[x].size()) {
            if (grid[u][v] != 0) {
                sum = max(sum, findGold(grid, u, v));
            }
        }
    }
    grid[x][y] = temp;
    return sum + grid[x][y];
}
```

运行结果



```
E:\VS\算法设计与分析\Assignment3\Debug\Problem1.exe
grid = [[1, 0, 7], [2, 0, 6], [3, 4, 5], [0, 3, 0], [9, 0, 20]]
28
请按任意键继续. . .
```