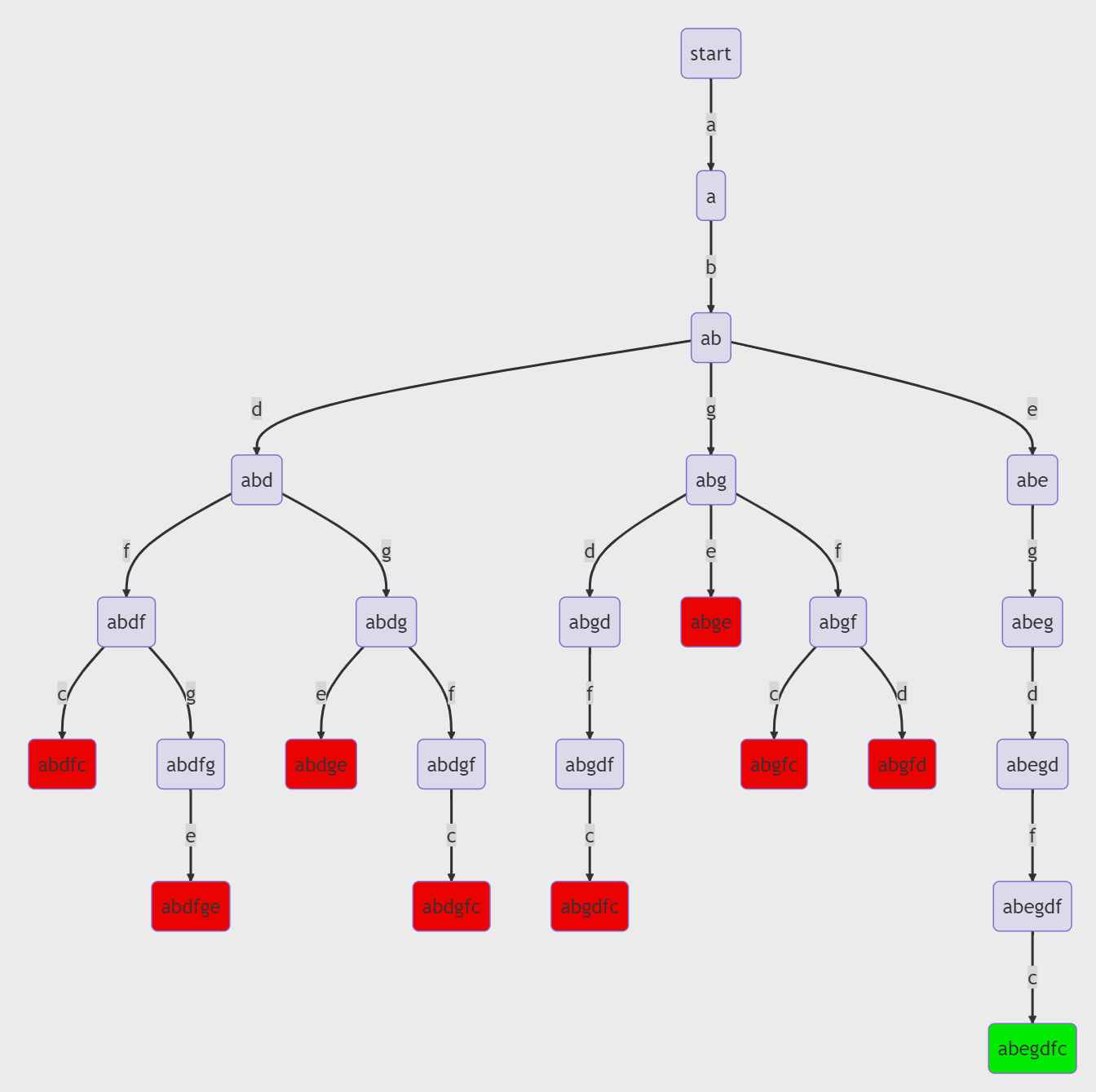
**1.**



**解空间树：**如上（仅画出了查找到第一条路径为止）。

**搜索过程：**从当前节点出发，遇到可以去到且当前路径上还未经过的节点，则将其作为解空间树中当前节点的子节点，同时添加到当前路径中。若无路可走，则删除当前路径上的当前节点，回溯到父节点再找一个合适的子节点继续搜索。

**搜索结果：**

①a-b-e-g-d-f-c-a

②......

**2.**

设需要找m元的零钱，最少硬币数为n。

①对于一般的零钱总额，只需考虑m-1元、m-3元、m-5元的最少硬币数，因为m元的最少硬币数都是这三个总额+1，是增长硬币数最少的方式。因此m元的最少硬币数等于三者硬币数分别+1的最小值。

②对于3<m<5，不可能会用到5元找零，只需考虑m-1、m-3元

③对于1<m<3，不可能会用到3元找零，只需考虑m-1元

④m=1、m=3、m=5的最优解显然是n=1，因此得到递推公式：



其中n(1) = n(3) = n(5) = 1

但由于最优解可能不止一种情况，状态表设置如下：它是一维的，记录了总额从1-m的最优组合。第i个位置记录的是找i元零钱硬币数最少的所有组合。

coin是可使用硬币的所有面值集合，money是找零总额。伪代码如下：

**FindSolution**(coin, money) {

result = [] \* (money + 1) //建立长度为result + 1的数组，每个元素是空集合

//长度为result + 1是为了应对m=0的情况

result[0].add([0, 0, 0]) //初始化：0元

**for** (c **in** coin) { //初始化，与硬币本身面值相等的最优解就是只找一个该面值的硬币

com ← []

**for** (i=1 **to** coin.**length**())

com.**add**(0)

com[coin.**index**(c)] ← 1

result[c].**add**(com)

}

**for** (i=2 **to** target) { //从2元开始填状态表

minSum ← **INT\_MAX** //记录i元时的最小硬币数，先赋一个很大的值

//先计算i元时所需的最小硬币数

**for** (c **in** coin) {

**if** (c > i) //硬币面值c比i还大，肯定不用c元硬币找零，不考虑

**pass**

**else** {

**for** (com **in** result[i - c]) {

tempSum ← **sum**(com)

minSum ← **min**(tempSum, minSum)

}

}

}

//再计算满足最小硬币数的所有组合

**for** (c **in** coin) {

**if** (c > i) //硬币面值c比i还大，肯定不用c元硬币找零，不考虑

**pass**

**else** {

**for** (com **in** result[i - c]) {

**if** (**sum**(com) == minSum) {

combination ← com

combination[coin.**index**(c)]++

result[i].**add**(combination)

}

}

}

}

}

return result[target]

}

求解本题时，coin = {1, 3, 5}，target = 9

状态表初始为：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| [0, 0, 0] | [1, 0, 0] |  | [0, 1, 0] |  | [0, 0, 1] |  |  |  |  |

①2元时，考虑1元的最少硬币数（1），因此硬币数最少只能为2。且计算所有情况后，得出都是2个1元

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| [0, 0, 0] | [1, 0, 0] | [2, 0, 0] | [0, 1, 0] |  | [0, 0, 1] |  |  |  |  |

②4元时，考虑1元和3元的最少硬币数（1、1），因此硬币数最少只能为2。且计算所有情况后，得出都是1个1元和1个3元

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| [0, 0, 0] | [1, 0, 0] | [2, 0, 0] | [0, 1, 0] | [1, 1, 0] | [0, 0, 1] |  |  |  |  |

③6元时，考虑1、3、5元的最少硬币数（1、1、1），因此硬币数最少只能为2。且计算所有情况后，得出1个1元和1个5元、2个3元

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| [0, 0, 0] | [1, 0, 0] | [2, 0, 0] | [0, 1, 0] | [1, 1, 0] | [0, 0, 1] | [1, 0, 1]  [0, 2, 0] |  |  |  |

④7元时，考虑2、4、6元的最少硬币数（2、2、2），因此硬币数最少只能为3。

且计算所有情况后，得出2个1元和1个5元、1个1元和2个3元

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| [0, 0, 0] | [1, 0, 0] | [2, 0, 0] | [0, 1, 0] | [1, 1, 0] | [0, 0, 1] | [1, 0, 1]  [0, 2, 0] | [2, 0, 1]  [1, 2, 0] |  |  |

⑤8元时，考虑3、5、7元的最少硬币数（1、1、3），因此硬币数最少为2。

且计算所有情况后，得出都是1个3元和1个5元

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| [0, 0, 0] | [1, 0, 0] | [2, 0, 0] | [0, 1, 0] | [1, 1, 0] | [0, 0, 1] | [1, 0, 1]  [0, 2, 0] | [2, 0, 1]  [1, 2, 0] | [0, 1, 1] |  |

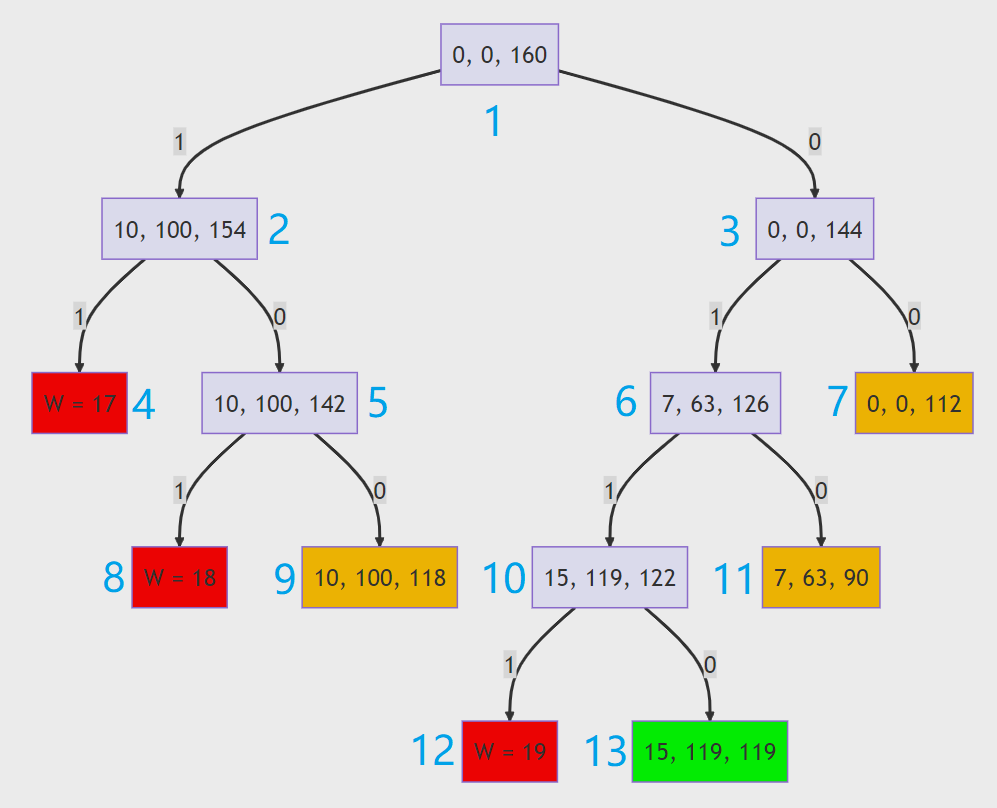
⑥9元时，考虑4、6、8元的最少硬币数（2、2、2），因此硬币数最少只能为3。

且计算所有情况后，得出1个1元和1个3元和1个5元、3个3元

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| [0, 0, 0] | [1, 0, 0] | [2, 0, 0] | [0, 1, 0] | [1, 1, 0] | [0, 0, 1] | [1, 0, 1]  [0, 2, 0] | [2, 0, 1]  [1, 2, 0] | [0, 1, 1] | [1, 1, 1]  [0, 3, 0] |

⑦得出结论：用1、3、5元硬币找9元的零钱，最少需要3个硬币，且所有组合为：[1, 1, 1]、[0, 3, 0]

**3.**



**解空间树：**如上

**搜索过程：**

**0.**价值上界计算方法（设当前包内总价值为v，总重量为w，剩余价值为r）：

**1.**从根节点出发，计算左右孩子的价值上界

Node1：w = 0, v = 0, dbest = 10, ub = 0 + 10 \* 16 = 160

Node2：w = 10, v = 100, dbest = 9, ub = 100 + 9 \* 6 = 154

Node3：w = 0, v = 0, dbest = 9, ub = 0 + 9 \* 16 = 144

**2.**当前叶结点的ub中，结点2（154）最大，从其开始计算

Node4：w = 17 > 16

Node5：w = 10, v = 100, dbest = 7, ub = 100 + 7 \* 6 = 142

**3.**当前叶结点的ub中，结点3（144）最大，从其开始计算

Node6：w = 7, v = 63, dbest = 7, ub = 63 + 7 \* 9 = 126

Node7：w = 0, v = 0, dbest = 10, ub = 0 + 7 \* 16 = 112

**4.**当前叶结点的ub中，结点5（142）最大，从其开始计算

Node8：w = 18 > 16

Node9：w = 10, v = 100, dbest = 3, ub = 100 + 3 \* 6 = 118

**5.**当前叶结点的ub中，结点6（126）最大，从其开始计算

Node10：w = 15, v = 119, dbest = 3, ub = 119 + 3 \* 1 = 122

Node11：w = 7, v = 63, dbest = 3, ub = 63 + 3 \* 9 = 90

**6.**当前叶结点的ub中，结点10（122）最大，从其开始计算

Node12：w = 19 > 16

Node13：w = 15, v = 119, dbest = 0, ub = 119 + 0 \* 1 = 119

**7.**找到一种可能解，检查其他叶结点：

ub7 = 112 < 119

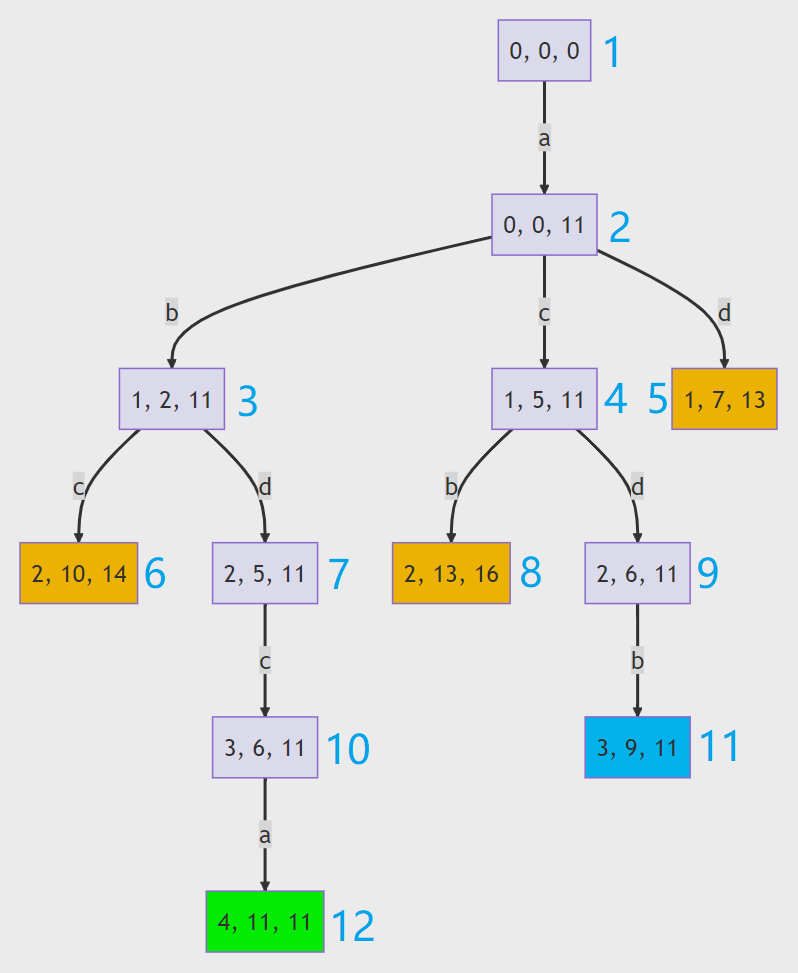
ub9 = 118 < 119

ub11 = 90 < 119

其他叶结点的ub都比119小，故不用再搜索

**搜索结果：**最优解为装入物品2和3

**4.**



**解空间树：**如上

**搜索过程：**

**0.**路径长度下界计算方法（设当前路径总长度为l，总边数为n，剩余路径长度为r）：

**1.**从根节点出发，计算所有孩子的路径长度下界

Node2：l = 0, n = 0, lenbest = (1, 2, 3, 5), lb = 0 + (1 + 2 + 3 + 5) = 11

Node3：l = 2, n = 1, lenbest = (1, 3, 5), lb = 2 + (1 + 3 + 5) = 11

Node4：l = 5, n = 1, lenbest = (1, 2, 3), lb = 5 + (1 + 2 + 3) = 11

Node5：l = 7, n = 1, lenbest = (1, 2, 3), lb = 7 + (1 + 2 + 3) = 13

**2.**当前叶结点的lb中，结点3（11）最小（假设lb相等时，取结点编号最小的），从其开始计算

Node6：l = 10, n = 2, lenbest = (1, 3), lb = 10 + (1 + 3) = 14

Node7：l = 5, n = 2, lenbest = (1, 5), lb = 5 + (1 + 5) = 11

**3.**当前叶结点的lb中，结点4（11）最小，从其开始计算

Node8：l = 13, n = 2, lenbest = (1, 2), lb = 13 + (1 + 2) = 16

Node9：l = 6, n = 2, lenbest = (2, 3), lb = 6 + (2 + 3) = 11

**4.**当前叶结点的lb中，结点7（11）最小，从其开始计算

Node10：l = 6, n = 3, lenbest = (5), lb = 6 + (5) = 11

**5.**当前叶结点的lb中，结点9（11）最小，从其开始计算

Node11：l = 9, n = 3, lenbest = (2), lb = 9 + (2) = 11

**6.**当前叶结点的lb中，结点10（11）最小，从其开始计算

Node12：l = 11, n = 4, lenbest = (0), lb = 11 + (0) = 11

**7.**找到一种可能解，检查其他叶结点：

lb5 = 13 > 11

lb6 = 14 > 11

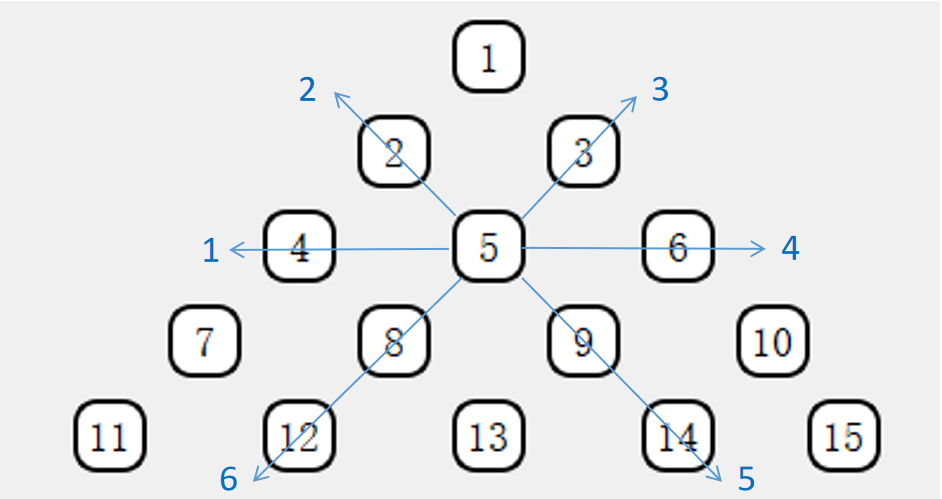
lb8 = 16 > 11

lb11 = 11 ≥ 11（lb与可能解相等，也不需要再查找）

**搜索结果：**最优解为a-b-d-c-a

**5.**

由题意，目标是要消去13个棒，而符合条件的移动每次可以消去一个棒，所以理论上最短步骤仅需13步就可达到目标。为求解该最短步骤，先做以下说明：



如图，每个插棒位置编号如下，每个空位可以接受来自这6个方向的跳跃（这六个方向均与图中最小等边三角形的边平行）。

例如：若5号为空，9、14号有棒，根据规则14号上的棒可以越过其邻居9而落到5上，同时消去9上的棒，我们称这是5号位接受了方向5的跳跃。

因此，可以利用回溯法：

①用数组记录棋盘状态，栈记录移动步骤，集合记录当前棋盘上的空位置

②先遍历13号位的6个方向，若能接受某一方向的跳跃，则改变棋盘状态

③遍历跳跃过后棋盘中的每一个空位，对其进行递归查询

④为了不产生重复的解，②中一旦找到一个解，便停止循环

⑤若想找出最后棒回到原来位置的算法，则在找到一个解时判断最后接受跳跃的位置是否为原位即可

**a.**

①board是一个结构体数组，记录棋盘每个位置的编号、是否为空等信息

②openlist是一个集合，记录当前棋盘上的空位置

③solution是一个栈，记录移动步骤

④Find函数判断num号位能否接受dir方向的跳跃

⑤Jump函数通过改变board状态位、openlist内容实现跳跃，同时将这一步记录在solution中

⑥Resume函数实现某一次Jump操作的撤销

伪代码如下：

**FindSolution**(board, openlist, solution, num) {

**if** (openlist.size == **14**)

**print**(solution)

**return true**

**for** (dir=**0** **to** **5**) {

**if** (**Find**(board, num, dir) == **true**) {

**Jump**(board, openlist, solution, num, dir)

**for** (pos **in** openlist) {

prime ← **FindSolution**(board, openlist, solution, pos)

**if** (prime == **true**)

**break**

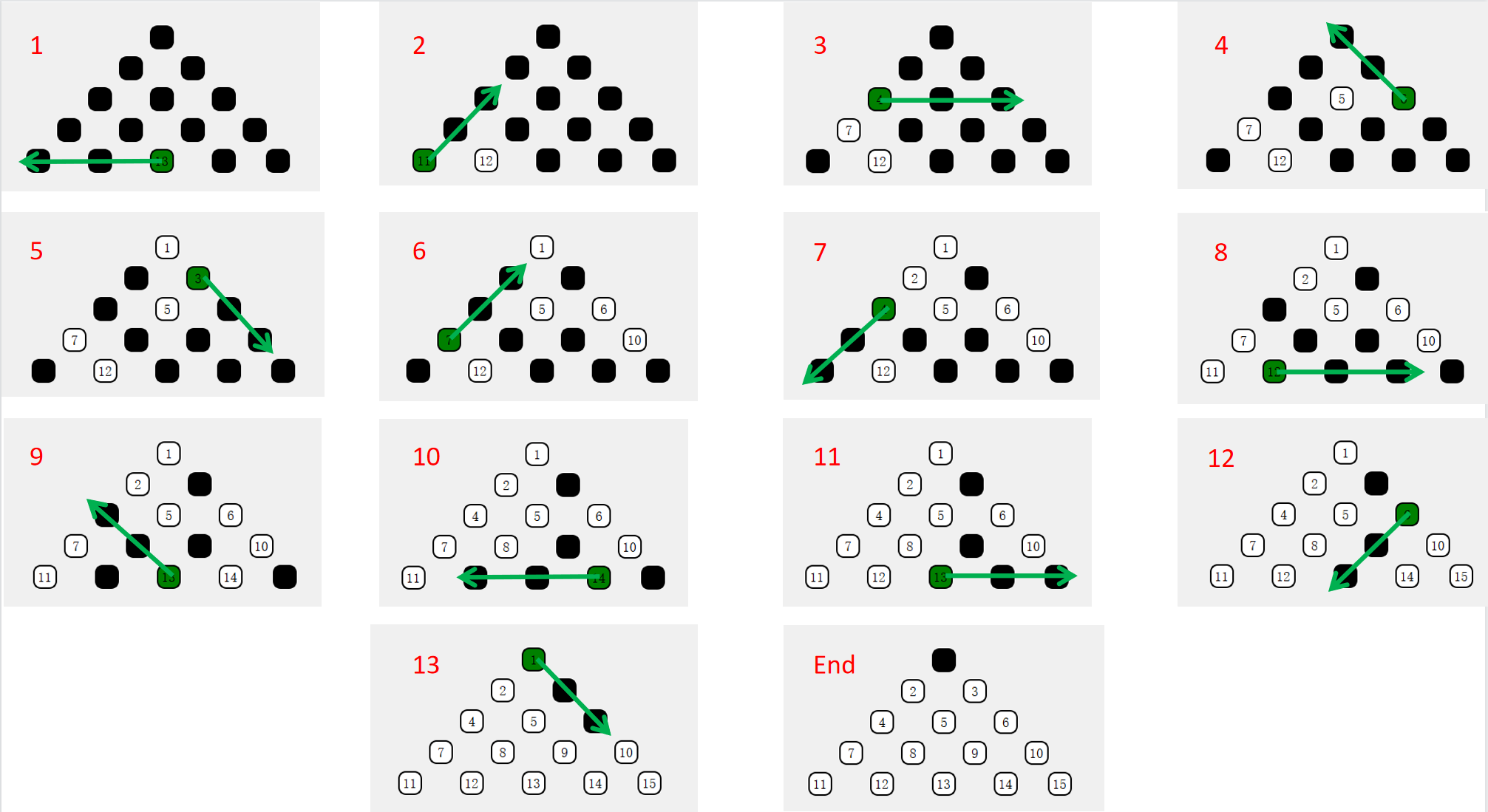
}

**Resume**(board, openlist, solution, num, dir)

}

}

}

求解时先对board进行初始化、将起点加入openlist、把solution置为空，再调用**FindSolution**(board, openlist, solution, **13**)即可，其中一个解如下：

**b.**

只需在找到解的时候判断最后一步是否由起点位置接受跳跃即可，其余与a一致，伪代码如下：

**FindSolution**(board, openlist, solution, num) {

**if** (openlist.size == **14 and** solution.**getTop**().num == **13**) //最后一步回到起点

**print**(solution)

**return true**

**for** (dir=**0** **to** **5**) {

**if** (**Find**(board, num, dir) == **true**) {

**Jump**(board, openlist, solution, num, dir)

**for** (pos **in** openlist) {

prime ← **FindSolution**(board, openlist, solution, pos)

**if** (prime == **true**)

**break**

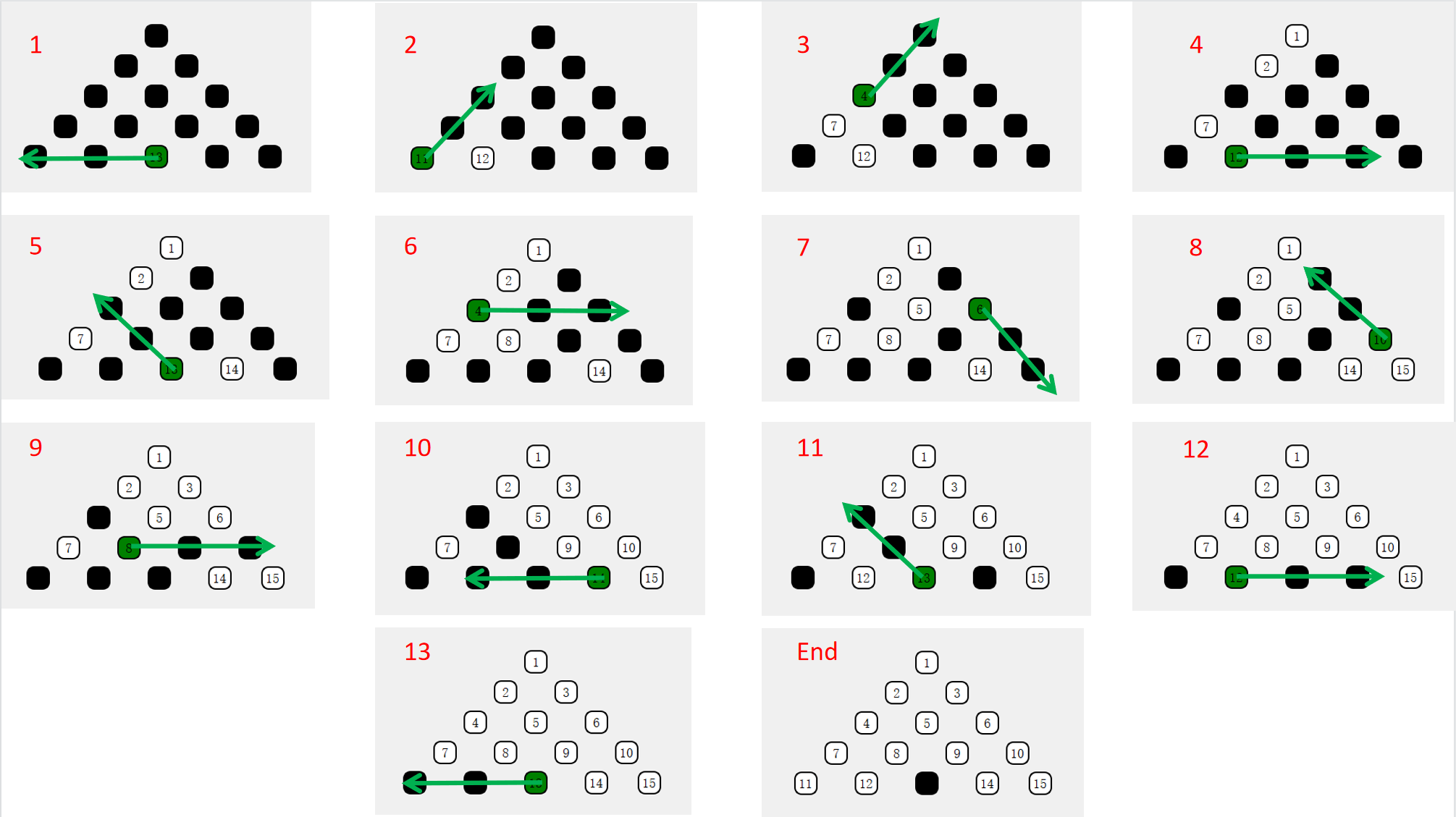
}

**Resume**(board, openlist, solution, num, dir)

}

}

}

求解时先对board进行初始化、将起点加入openlist、把solution置为空，再调用**FindSolution**(board, openlist, solution, **13**)即可，其中一个解如下：

**编程题：**

由题意，需要找到一条路径使得黄金数量最大。对于某个特定的起点，寻找黄金数量最大的路线可采用回溯法查找。整个过程中有一个栈currrent记录当前路径、两个变量currentValue和maxValue记录当前黄金数量和当前最大黄金数量：

①统计当前位置上可行进的方向，若没有可行进方向，则看currentValue是否大于maxValue，若是则更新maxValue的值

②若有可行进方向，则遍历每个可行进方向进行递归查询

③每一次递归查询完成后撤销当前这一步以完成回溯

这样可以通过回溯法，找到对于某个特定起点的最优解。然而起点并不是固定的，因此实际上需遍历grid中所有非0位置，进行①-③的过程，在得到的结果中取黄金数量最大的解，才能满足题目需要。

开销上，设grid大小为m\*n。由于要遍历所有非0位置（最多mn个），除了第一个位置外，其余位置都有三个可递归查询的方向（上下左右减去自己走过的方向），解空间可表示为三叉树，故**时间复杂度**为。空间上栈current最大只有mn，而grid、wayMark固定为mn，故**空间复杂度**为

代码如下（仅展示主体部分，其余函数的具体实现在cpp文件中）：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

  函数名称：getPointMaximumGold

  功    能：查找对于某一特定起点的最优解

  输入参数：vector<vector<maps>>& grid：表示金矿的网格

            vector<vector<bool>>& wayMark：标记当前路径走过的位置的二维数组

            stack<point>& current：记录当前路径的栈

            point& miner：矿工的当前位置

            int& currentValue：当前黄金数目

            int& maxValue：当前最优解黄金数目

  返 回 值：int：对于某一特定起点，最优解的黄金数目

  说    明：无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int getPointMaximumGold(vector<vector<int>>& grid,

    vector<vector<bool>>& wayMark,

    stack<point>& current,

    point& miner,

    int& currentValue,

    int& maxValue)

{

    vector<point> next;  /\*存放可行进位置的向量\*/

    direction\_count(grid, wayMark, next, miner);  /\*统计可行进位置\*/

    if (next.empty()) {  /\*若无可行进位置，则找到一条路径，根据当前黄金数量更新最优解\*/

        if (currentValue > maxValue)

            maxValue = currentValue;

        return maxValue;

    }

    for (auto iter = next.begin(); iter != next.end(); iter++)  /\*若有可行进位置，则对每一个可行进方向进行递归查询\*/

    {

        walk(grid, wayMark, current, miner, \*iter, currentValue);  /\*前进\*/

        getPointMaximumGold(grid, wayMark, current, miner, currentValue, maxValue);  /\*递归查询\*/

        back(grid, wayMark, current, miner, currentValue);  /\*恢复\*/

    }

    return maxValue;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

  函数名称：getMaximumGold

  功    能：查找黄金数量最大的路径

  输入参数：vector<vector<maps>>& grid：表示金矿的网格

  返 回 值：int：最优解的黄金数目

  说    明：无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int getMaximumGold(vector<vector<int>>& grid)

{

    int maxValue = 0;  /\*当前最大黄金数量\*/

    stack<point> current;  /\*当前路径\*/

    vector<vector<bool>> wayMark(grid.size(), vector<bool>(grid[0].size(), false));  /\*记录路径上经过位置的二维数组\*/

    for (int i = 0; i < grid.size(); i++)

    {

        for (int j = 0; j < grid[0].size(); j++)

        {

            if (grid[i][j]) {  /\*遍历每个不为0的位置，让其作为起点\*/

                point miner(i, j);              /\*矿工在起点位置\*/

                int currentValue = grid[i][j];  /\*当前黄金数量，初始为起点的黄金数量\*/

                wayMark[i][j] = true;  /\*初始位置已走过，做上标记\*/

                current.push(miner);   /\*当前路径记录下起点\*/

                getPointMaximumGold(grid, wayMark, current, miner, currentValue, maxValue);  /\*找出这个起点的最优解\*/

                wayMark[i][j] = false;  /\*查找完成，标记取消\*/

                current.pop();  /\*查找完成，当前路径弹出起点\*/

            }

        }

    }

    return maxValue;

}

运行结果：

