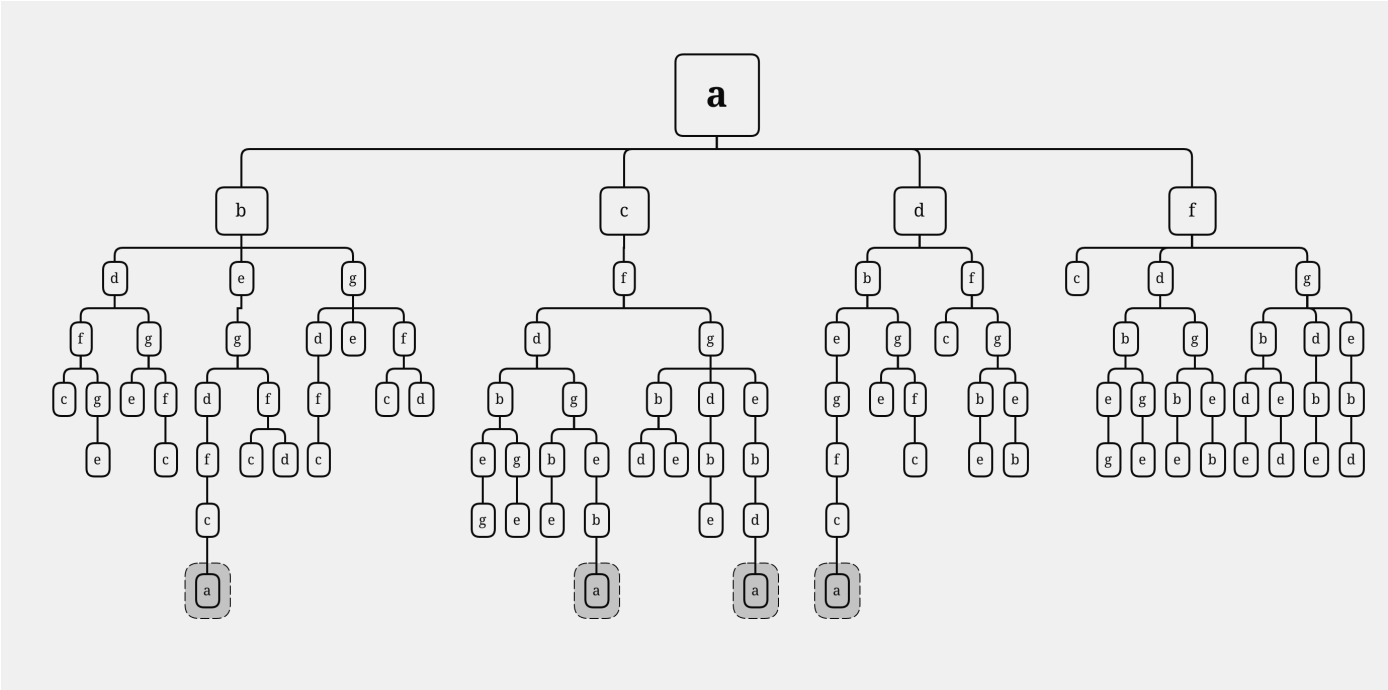
计算题

1.

解空间树：



搜索过程：

- a-b-d-f-c-(回溯到f)
- f-g-e(回溯到g-f-d)
- d-g-e(回溯到g)
- g-f-c(回溯到f-g-d-b)
- b-e-g-d-f-c-a(成功求到一个解)

结果：a-b-e-g-d-f-c-a

2.

定义状态转移方程：

设$f(n)$表示组成面额为n时所需最少硬币总数量，初始$f(0)=0$。

$g(n)$表示组成面额为n时3种硬币各需要几个，初始$g(0)=(0,0,0)$

则：$f(n)=min\{f(n-1),f(n-3),f(n-5)\}+1$

若$f(n-1)$最小，则$g(n)=g(n-1)+(1,0,0)$;

若$f(n-3)$最小，则$g(n)=g(n-3)+(0,1,0)$;

若$f(n-5)$最小，则$g(n)=g(n-5)+(0,0,1)$。

$g(n)$可以有多个值。

根据上面的初始化和递推规则，填完下面的表：

n	0	1	2	3	4	5	6	7	8	9
$f(n)$	0	1	2	1	2	1	2	3	2	3
$g(n)$	$(0,0,0)$	$(1,0,0)$	$(2,0,0)$	$(0,1,0)$	$(1,1,0)$	$(0,0,1)$	$(1,0,1),(0,2,0)$	$(2,0,1),(1,2,0)$	$(0,1,1)$	$(1,1,1),(0,3,0)$

得到最少硬币数为3，可能的组合为：

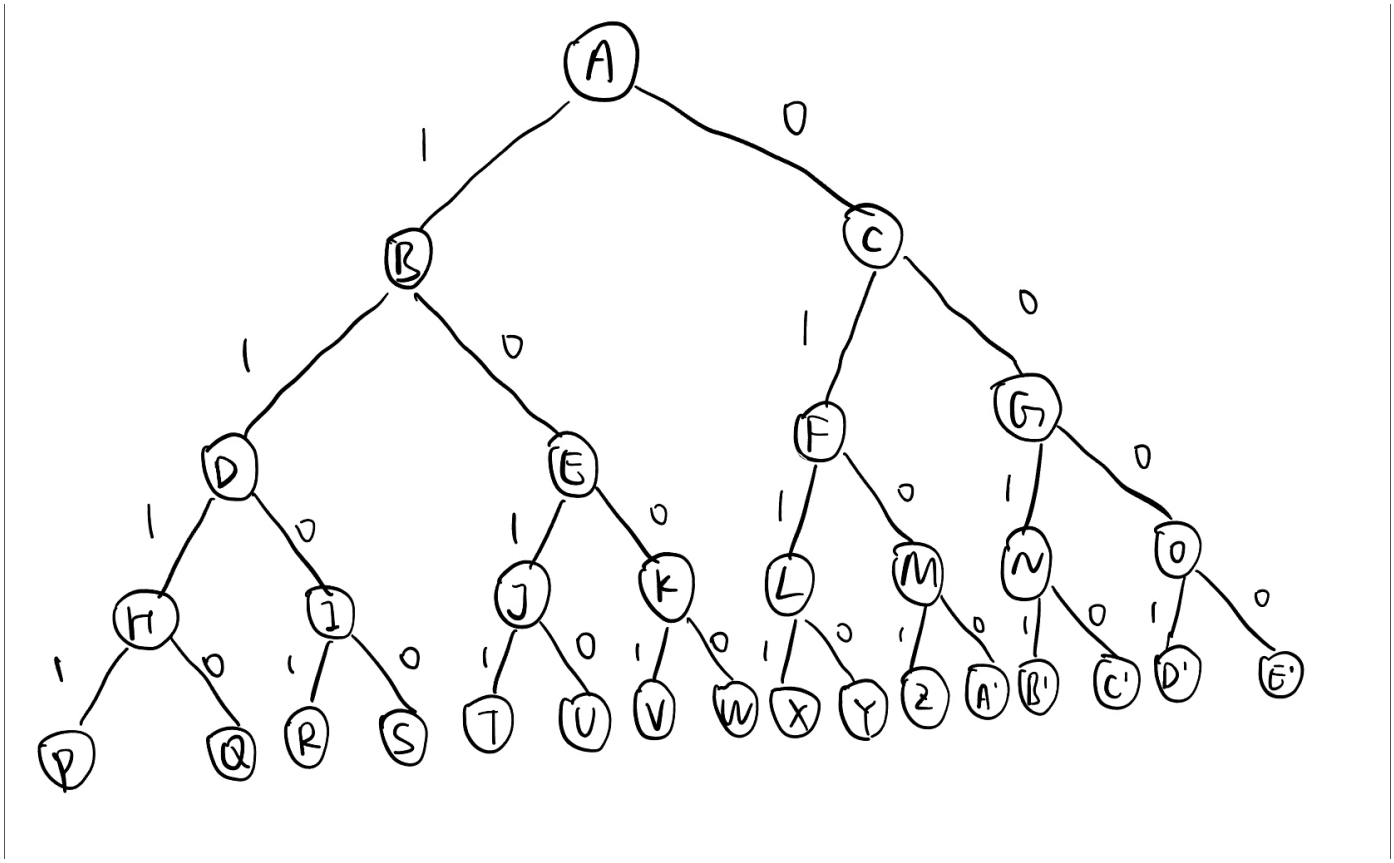
- $1\times 1, 3\times 1, 5\times 1$
- 3×3

算法伪代码：

```
def findCoinNum(n):
  # 初始化表格
  table[0][0]=0
  table[1][0]=(0,0,0)

  for i in range(1, n + 1):

    找出所有具有最小f值f_min的n值, min_f_n = [n_0, n_1, n_2, ...]

    # 更新f(n)
    table[0][i] = f_min + 1

    # 更新g(n)
    for m in min_f_n:
      if m == 1:
        table[1][i].append(table[1][m] + (1,0,0))
      elif m == 3:
        table[1][i].append(table[1][m] + (0,1,0))
      elif m == 5:
        table[1][i].append(table[1][m] + (0,0,1))

  # 返回结果
  return table[0][n],table[1][n]
```

3.

解空间树：



搜索过程：

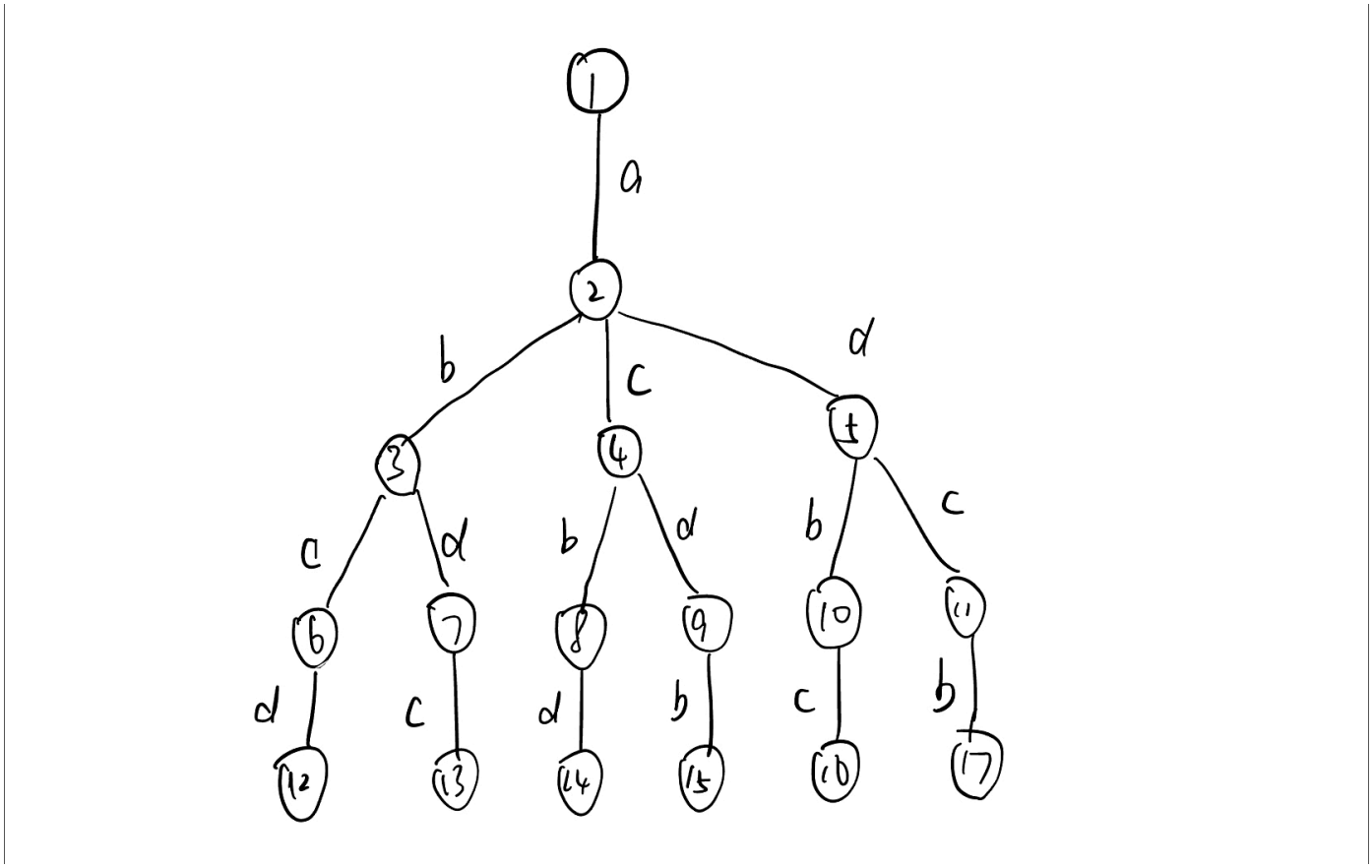
扩展结点	活结点	队列	可行解	解值
A	B, C	BC		
B	D, E(D死)	CE		
C	F, G	EFG		
E	J, K(J死)	FGK		
F	L, M	GKLM		
G	N, O	KLMNO		
K	V, W	LMNO	V, W	112, 100
L	X, Y (X死)	MNO	Y	119
M	Z, A'	NO	Z, A'	75, 63
N	B', C'	O	B', C'	68, 56
O	D', E'	ϕ	D', E'	12, 0

结果：

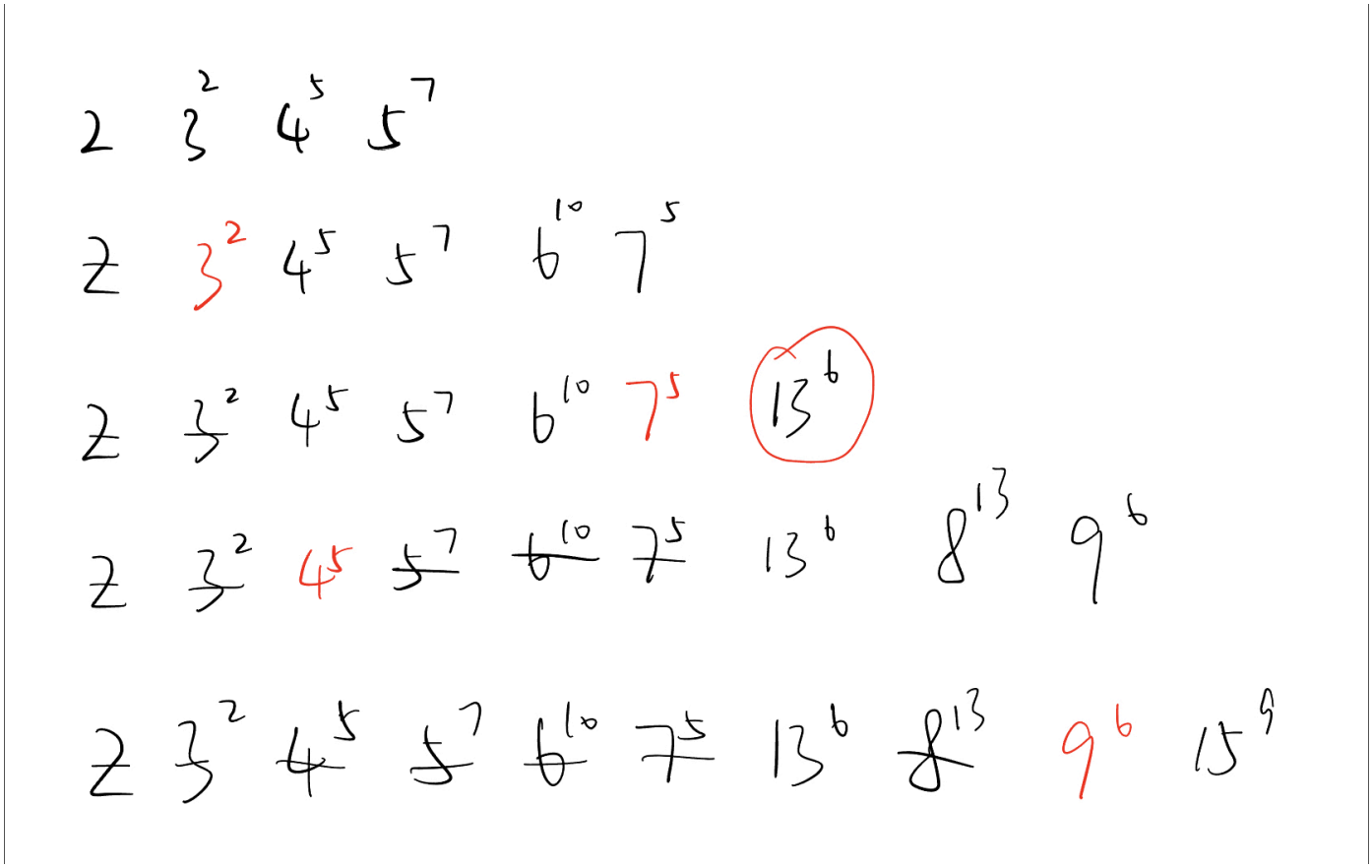
选择物体2、3，总价值119。

4.

解空间树：



优先队列式分支线接法搜索过程：（利用了极小堆剪枝）



结果：

选择物体2、3，总价值119。

5.

思路：

1. 根据等边三角形旋转对称性，状态空间中很多可以通过旋转、镜像相互转换，这些同构状态下要么都有解、要么都无解
2. 剪枝函数为判断当前状态是否有解。因初始为14个棒，若有解必然13步后结束，因此不能用最短步数剪枝。通过查询当前状态是否与之前记录过的一个无解状态同构，判断是否剪枝。

伪代码：

```
# 数据结构定义
棋盘状态s：二维数组
最初空孔位置pos：二元组
无解状态集合terminal：链表
步骤steps：栈
结果results：链表

# 初始化
初始化上述数据结构
将初始状态s_0放入栈steps

```

```
while steps不为空：
  取出栈顶元素s

  # 得到一个解
  if 状态s只剩一个棒：
    # 对于问题a，直接得到结果
    将整个栈steps复制并链入结果链表results

    # 对于问题b，需判断最后的棒是否在最初空孔上
    if 最后的棒的位置 == 最初空孔位置pos：
      将整个栈steps复制并链入结果链表results

  else:
    # 得到一个无解状态
    if 状态s已经无法进行跳跃：
      将整个栈steps复制并链入无解状态集合链表terminal

    # 当前状态既非无解、也非结果，则记录其子节点
    else:
      # 通过遍历状态里的每个棒子，判定其是否可以跳跃来得到子节点
      for elem in s:
        if 棒子elem可以跳跃：
          s_ = elem跳跃后得到的状态
          children.append(s_)

          # 若子节点s_不是无解状态，则加入栈
          if s_不与无解状态集合terminal中的任何状态同构：
            steps.append(s_)
```

编程题

算法思路

1. 遍历所有格子，对于有黄金的格子，都作为起点进行一次探索，计算每次探索得到的最大黄金数，取其中最大的那个作为最终结果
2. 每次探索在maxGold函数中完成，过程如下：
    - 维护一个最大黄金数变量
    - 获取当前格子上的黄金（grid对应元素置0）
    - 遍历当前格子的4个邻居格子，若邻居格子位置合法，则递归调用maxGold函数，计算以邻居格子为起点时探索获得的最大黄金数，与当前最大黄金数比较、更新
    - 递归探索结束即可得到以该格子为起点时得到的最大黄金数

复杂度计算

令m = grid.size(), n = grid[0].size(), k = 有黄金单元格的数量

1. 时间复杂度$O(mn + k * 3^k)$，mn项为计算起点的时间，$k * 3^k$中k表示有k种可能的起点，3^k表示开采黄金的路径上，每个单元格最多3个分岔。
2. 空间复杂度$O(k)$，即递归栈的最大尺寸。

主要函数

SharingGardenProject - main.cpp

```cpp
int maxGold(vector<vector<int>> &grid, int i, int j)
{
    int max_gold = 0;                                       // 最大黄金数
    int dir[4][2] = {{-1, 0}, {0, -1}, {1, 0}, {0, 1}}; // 方向数组
    int temp = grid[i][j];                                  // 保存当前位置的黄金数
    grid[i][j] = 0;                                         // 将当前位置的黄金数置为0，表示已经访问过

    // 遍历当前位置的四个方向
    for (int k = 0; k < 4; ++k)
    {
        // 计算下一步坐标
        int x = i + dir[k][0], y = j + dir[k][1];

        // 如果新坐标合法，则递归计算该路径上最大黄金数
        if (x >= 0 && x < grid.size() && y >= 0 && y < grid[i].size() && grid[x][y] != 0)
        {
            max_gold = max(max_gold, maxGold(grid, x, y));
        }
    }

    // 将当前位置的黄金数还原
    grid[i][j] = temp;

    return max_gold + grid[i][j];
}

int getMaximumGold(vector<vector<int>> &grid)
{
    // 最大黄金数
    int max_gold = 0;

    // 遍历每个位置
    for (int i = 0; i < grid.size(); ++i)
    {
        for (int j = 0; j < grid[i].size(); ++j)
        {
            // 如果当前位置有黄金，就以当前位置为起点计算可以获得的最大黄金数
            // 并更新最大黄金数
            if (grid[i][j] != 0)
            {
                max_gold = max(max_gold, maxGold(grid, i, j));
            }
        }
    }
    return max_gold;
}
```

运行结果

```
gorithm/Assignment3/"main
[[0,6,0],[5,8,7],[0,9,0]]
24
guolianglu@GuoLiangdeMacBook-Air Assignment3 %

[[1,0,7],[2,0,6],[3,4,5],[0,3,0],[9,0,20]]
28
guolianglu@GuoLiangdeMacBook-Air Assignment3 %
```