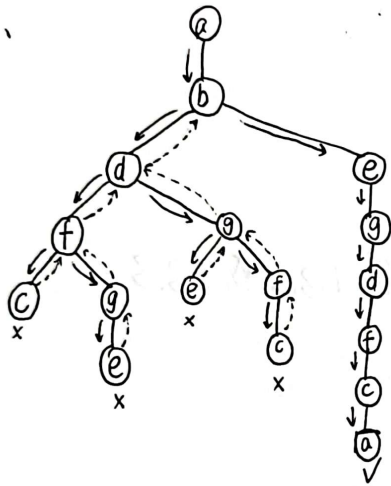


2 计算题

1.



可行解: abegdfca

2. 伪代码: $\text{ChangeMaking}(D[1..m], n)$
 // 应用动态规划算法求解找零问题, 找出使硬币加起来等于 n 时所需最少硬币数
 // 其中币值 $d_1 < d_2 < \dots < d_m, d_1 = 1$
 // $D[1..m]$ 为币值递增整数数组, $F[n]$ 为币值 n 所需最少硬币数

$F[0] \leftarrow 0$

for $i \leftarrow 1$ to n do

temp $\leftarrow \infty$; $j \leftarrow 1$

while $j \leq m$ and $i \geq D[j]$ do

temp $\leftarrow \min[F[i - D[j]], \text{temp}]$

$j \leftarrow j + 1$

$F[i] \leftarrow \text{temp} + 1$;

find-solution(temp, $D[1..m]$, $F[1..n]$, n) // 回溯找硬币组合

2(续) find-solution(temp, $D[1..m]$, $F[1..n]$, i)

// 函数实现

if temp ≤ 0
 print("n")
 return;

$j \leftarrow 1$;

while $j \leq m$ and $D[j] \leq i$

if $F[i - D[j]] == \text{temp}$
 print($D[j] + "$ ")

temp = $F[i - D[j]] - 1$;

find-solution(temp, $D[1..m]$, $F[1..n]$, $i - D[j]$)

对于 $n=9$, 币值为 1, 3, 5 的硬币应用上述算法的运行过程.

$$F[0] = 0$$

$$F[1] = \min\{F[1-1]\} + 1 = 1$$

$$F[2] = \min\{F[2-1]\} + 1 = 2$$

$$F[3] = \min\{F[3-1], F[3-3]\} + 1 = 1$$

$$F[4] = \min\{F[4-1], F[4-3]\} + 1 = 2$$

$$F[5] = \min\{F[5-1], F[5-3], F[5-5]\} + 1 = 1$$

$$F[6] = \min\{F[6-1], F[6-3], F[6-5]\} + 1 = 2$$

$$F[7] = \min\{F[7-1], F[7-3], F[7-5]\} + 1 = 3$$

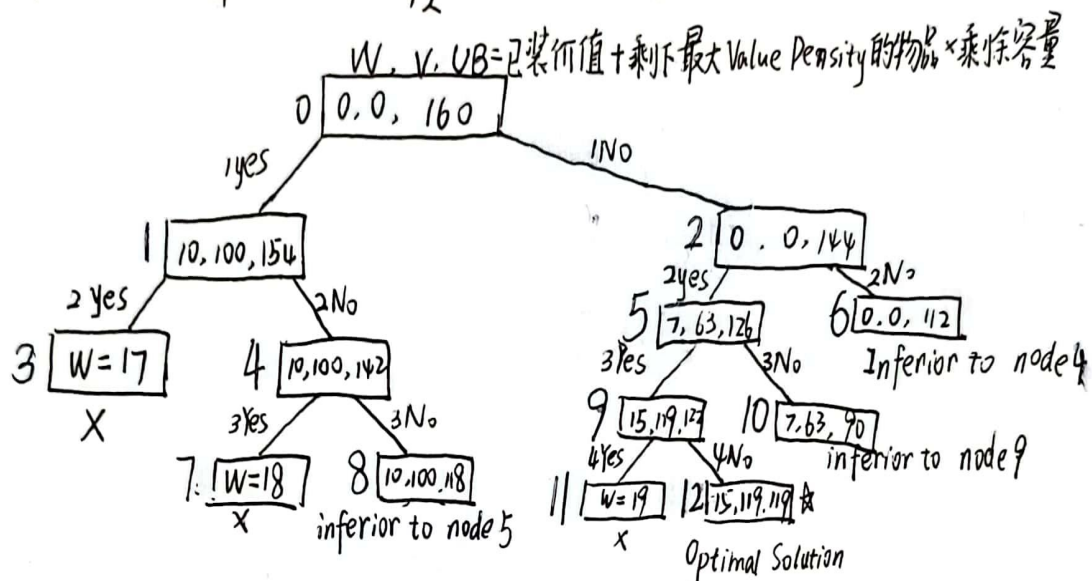
$$F[8] = \min\{F[8-1], F[8-3], F[8-5]\} + 1 = 2$$

$$F[9] = \min\{F[9-1], F[9-3], F[9-5]\} + 1 = 3$$

∴ 硬币最少个数是 3. 应用回溯法找产生该值的解可得: 1, 3, 5 或 3, 3, 3

3. $W=16$, $n=4$. 按 Value Density 降序排列:

Item	Weight	Value	Value Density
1	10	100	10
2	7	63	9
3	8	56	7
4	4	12	3



∴ 最优解: (2, 3) 项加入背包.

$B \quad \underline{C^2} \quad D \quad E$
$$D \quad C^2 \quad P^5 \quad E^7 \quad F^{10} \quad G^5 \quad H^{13} \quad I^6$$
$$B \quad C^2 \quad D^5 \quad E^7 \quad F^{10} \quad G^5 \quad H^{13} \quad I^6 \quad \frac{M''}{10} \quad O''$$
$$\begin{array}{cccccccccccc} B & C & D & E & F & G & H & I & J & K & L & M \\ B & C^2 & D^5 & E^7 & F^{10} & G^5 & H^{13} & I^6 & J^{10} & K^8 & L^4 & M^{11} \end{array}$$
$$B \quad d^2 \quad A^5 \quad E^7 \quad F^{10} \quad G^5 \quad H^{13} \quad Y^6 \quad J^{10} \quad K^8 \quad \underline{M^{11}} \quad O^{11} \quad Q^{18}$$
$$B^1 A^2 D^5 E^7 F^{10} G^5 H^{13} I^6 J^{10} K^8 \quad \underline{M^{11}} \quad O^{11} \quad P^{18} \quad L^{18}$$
$$A^1 A^2 A^3 A^4 A^5 A^6 A^7 A^8 A^9 A^{10} A^{11} A^{12} A^{13} A^{14} A^{15} A^{16} A^{17} A^{18} A^{19} A^{20} A^{21} A^{22} A^{23}$$

$\therefore a b d c a$ ~~$a c d b a$~~ $\min = 11$

5.

主要思路：每次按顺序遍历 board，有棒子就从该位置依次向左上、右上、左、右、左下、右下 6 个方向走，当走到越界/有棒的位置就返回。若走到合法位置，判断中间是否跳过了棒子，将被跳过的棒子删去，步数加 1，当盘上只剩一根棒子时，是为一个成功路径，将 step 更新，记录最短总步数。对于 b 版本，成功条件还要加上最后一个棒子在 (4, 2) 处。

伪代码：

```

min_path;
step = INT_MAX
stick_num = 14
board[5][5] = { 1, 0, 0, 0, 0;
                 1, 1, 0, 0, 0;
                 1, 1, 1, 0, 0;
                 1, 1, 1, 1, 0;
                 1, 1, 1, 1, 1; }
dir[6][2] = { -2, -2; -2, 0; 0, -2; 0, 2; 2, 0; 2, -2; } // 表示移动方向
Solution(x=0, y=0, temp=0, which_dir=-1, path)
if which_dir != -1 and (x < 0 or x >= 5 or y < 0 or y >= 5 or board[x][y] or y > x or temp >= step
// 不是起始步，但位置越界/非空/当前步数已 > step
return;
temp++; // 当前步数 + 1
board[x][y] = 1; remove = 0; // 是否删了棒子
if board[x-dir[which_dir][0]/2][y-dir[which_dir][1]/2] and which_dir != -1
board[x-dir[which_dir][0]/2][y-dir[which_dir][1]/2] = 0; // 被跳过棒子从 board 上移去
stick_num--; remove = 1;
path.append(move); // 将这次棒子移动加入路径 (x-dir, y-dir) -> (x, y)
if stick_num == 1 /* and board[4][2] == 1 */ // b 版本把括号去掉
step = min(step, temp); min_path = path;
return
for i <- 0 to 4 do
for j <- 0 to i do
if board[i][j] == 1
board[i][j] = 0
for k <- 0 to 5 do // 遍历 6 个方向，将相邻位置有棒子的方向加入 q1.
if board[i+dir[k][0]/2][j+dir[k][1]/2] == 1 // 否则加入 q2，即优先考
q1.push(k); // 虑可以减棒子的方向，
else // 尽量避免盲目空跳
q2.push(k)

while q1 do
k = q1.top(); q1.pop()
solution(i+dir[k][0], j+dir[k][1], temp, path)
while q2 do
k = q2.top(); q2.pop()
solution(i+dir[k][0], j+dir[k][1], temp, path)
board[i][j] = 1
board[x][y] = 0 // 回溯前先复原本层修改
if remove // 恢复删去的棒子
board[x-dir[which_dir][0]/2][y-dir[which_dir][1]/2] = 1
stick_num++

```


3、编程题

思路：遍历整个矩阵grid，以每一个值不为0且连通度不为2的位置为起点进行深搜（连通度指与之相邻的可走的位置数），在深搜时用到回溯法和递归函数，每走到一个结点，依次将它上下左右相邻位置作为新的扩展结点进行深搜。为了防止重复遍历同一位置，在回溯前将经过的位置值置0，将要回溯时再将该结点恢复原来的值。每次从一个新起点开始搜索时NowGold=0，作为当前gold总值被传给下一层dfs累加，每经过一个有矿位置NowGold加上该位置的值，用Max变量记录历史最大NowGold值，每次NowGold变化时更新Max。最后输出结果即为Max。

在这里做了点优化：选择起点时跳过连通度为2的位置。因为连通度为2的位置只有一个入口一个出口，以该位置出发的路径必然包括其联通的两个位置，所以只要考虑其联通的两个位置为起点出发的路径即可，即只要以连通度为1、3、4的位置为起点。

时间复杂度： $O(a \cdot 3^a)$ 。其中a表示包含黄金的单元格数。枚举起点需要 $O(mn)$ 的时间，可能的起点有a个。对于每一个起点，第一步最多有4个可行的方向，后面的每一步最多有3个可行的方向，因此可以粗略估计出一次搜索需要的时间为 $O(4 \cdot 3^a - 2) = O(3^a)$

空间复杂度： $O(\min(mn, a))$ 。取决于数组大小和递归栈的大小。

主要函数截图：

```
// 判断是否越界 或 走到了无金子位置
bool ifNotEnd(int x, int y) {
    if (x < 0 || x >= m || y < 0 || y >= n || grid[x][y] == 0) {
        return false;
    }
    return true;
}

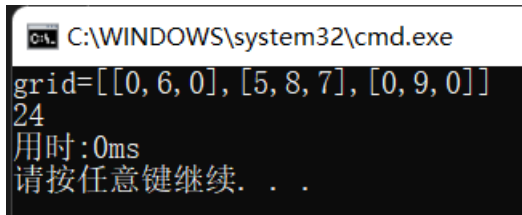
//判断某位置上下左右有几个方向邻接了有矿位置
int OpenDerection(int x, int y) {
    int ans = 0;
    ans += ifNotEnd(x + 1, y);
    ans += ifNotEnd(x - 1, y);
    ans += ifNotEnd(x, y + 1);
    ans += ifNotEnd(x, y - 1);
    return ans;
}

void dfs(int NowGold, int x, int y) {
    if (!ifNotEnd(x, y)) { // 判断是否越界或到无金子位置，结束
        return;
    }
    int temp = grid[x][y];
    NowGold += grid[x][y];
    Max = max(Max, NowGold); // 更新最大值
    grid[x][y] = 0; // 防止出现重复遍历
    dfs(NowGold, x + 1, y); // 向上
    dfs(NowGold, x - 1, y); // 向下
    dfs(NowGold, x, y + 1); // 向左
    dfs(NowGold, x, y - 1); // 向右
    grid[x][y] = temp; // 回溯
}

//求最大受益
void getMaximumGold() {
    m = grid.size(), n = grid[0].size();
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            if (grid[i][j] && OpenDerection(i, j) != 2) { // 从每个非0且通路方向不为2的位置开始dfs
                dfs(0, i, j);
            }
}
```

示例运行结果：

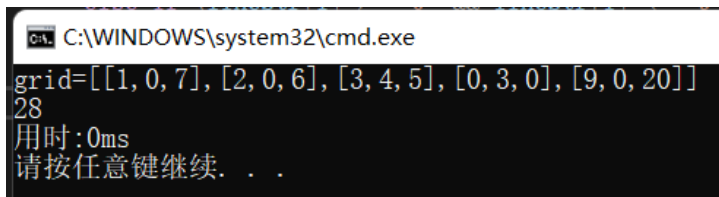
示例1：



```
C:\WINDOWS\system32\cmd.exe
grid=[[0, 6, 0], [5, 8, 7], [0, 9, 0]]
24
用时:0ms
请按任意键继续. . .
```

示例运行结果：

示例2：



```
C:\WINDOWS\system32\cmd.exe
grid=[[1, 0, 7], [2, 0, 6], [3, 4, 5], [0, 3, 0], [9, 0, 20]]
28
用时:0ms
请按任意键继续. . .
```