

硬币收集问题算法设计

1. 符号

- 设：有一个 n 行 m 列的棋盘 $M(n, m)$, 并用 $M(i, j)$ 表示 M 的第 i 行第 j 列位置上放有几枚硬币（在本题中取值为0或1）
- 设：有一个机器人 R 位于 M 的最左上角，初始坐标为 $R(1, 1)$
- 令： $F(i, j)$ 为机器人走到 (i, j) 位置时能收集到的硬币的最大值
- 令： $R(i + 1, j)$ 为机器人向下移动一个单位
- 令： $R(i, j + 1)$ 为机器人向右移动一个单位
- 令： $OPT(n, m)$ 为机器人在 n 行 m 列棋盘移动所能收集到的最大硬币数量，即全局最优解

2. 精确解

解法一：Brute force

- 因为机器人在移动到除了最右边或最下边的格子外的位置时，都需要进行一次选择——向右或向下，所以可以生成一棵二叉树来表示所有可能的解
- 因为机器人从 $R(1, 1)$ 到 $R(n, m)$ 需要移动 $n + m - 2$ 步，又因为机器人只能向右或者向下移动，所以深度优先遍历二叉树后有 2^{n+m-2} 条路径，因此可以采用Brute force遍历所有路径，等效于对 $['DOWN'] \times (n - 1) + ['RIGHT'] \times (m - 1)$ 做全排列，然后遍历所有路径找到最优解

伪代码一

```
Algorithm 1 Exact Solution: Brute force
1: procedure BRUTEFORCE( $M[n, m], Path_{temp}, i, j$ )
2:    $Path_{temp} = ['DOWN'] \times (n - 1) + ['RIGHT'] \times (m - 1)$ 
3:   procedure PERMUTATIONS( $arr, position, end$ )
4:     if  $position = end$  then
5:       if not  $arr \in Path_{aval}$  then
6:          $Path_{aval}.append(arr.copy())$ 
7:       end if
8:     else
9:       for  $index = position$  to  $end$  do
10:        exchange
11:        exchange  $arr[index]$  with  $arr[position]$ 
12:        PERMUTATIONS( $arr, position + 1, end$ )
13:        exchange  $arr[index]$  with  $arr[position]$ 
14:      end for
15:    end if
16:  end procedure
17: PERMUTATIONS( $Path_{temp}, 0, len(Path_{temp})$ )
18: for  $Path_{each} \in Path_{aval}$  do
19:    $i, j = 0, 0$ 
20:    $F = M[0, 0]$ 
21:   for  $step \in Path_{each}$  do
22:     if  $step = "DOWN"$  then
23:        $i += 1$ 
24:     else if  $step = "RIGHT"$  then
25:        $j += 1$ 
26:     end if
27:      $F += M[i, j]$ 
28:   end for
29:   if  $F > Ans$  then
30:      $Ans = F$ 
31:      $Path = Path_{each}$ 
32:   end if
33: end for
34: return  $Ans, Path$ 
35: end procedure
```

解法二：迭代遍历

- 因为机器人只能从左向右，从上向下移动，且机器人会收集经过的路径上的所有硬币，所以 $F(i, j)$ 的值为 $F(i - 1, j)$ 和 $F(i, j - 1)$ 中较大的值加上 c_{ij}

2. 因为机器人在 $i = 1$ 时无法向上移动, 前一推理结果, 等效为上一行的硬币数量为0, 即 $P(0, j) = 0$, 同理可得 $P(i, 0) = 0$

$$F(i, j) = \max\{F(i - 1, j), F(i, j - 1)\} + M(i, j) \quad \text{for} \quad 1 \leq i \leq n, \quad 1 \leq j \leq m \tag{1-1}$$

$$F(i, 0) = 0 \quad \text{for} \quad 1 \leq i \leq n \quad \text{and} \quad F(0, j) = 0 \quad \text{for} \quad 1 \leq j \leq m \tag{1-2}$$

伪代码二

Algorithm 2 Exact Solution: Iterative Traversal

```
1: procedure ITERATIVE TRAVERSAL( $M[n, m]$ )
2:    $F[1, 1] = M[1, 1]$ 
3:   for  $j = 2$  to  $m$  do
4:      $F[1, j] = F[1, j - 1] + M[1, j]$ 
5:   end for
6:   for  $i = 2$  to  $n$  do
7:      $F[i, 1] = F[i - 1, 1] + M[i, 1]$ 
8:     for  $j = 2$  to  $m$  do
9:        $F[i, j] = \max(F[i - 1, j], F[i, j - 1]) + M[i, j]$ 
10:    end for
11:  end for
12:   $x, y = n, m$ 
13:  while  $x > 1$  and  $y > 1$  do
14:    if  $F[x - 1, y] > F[x, y - 1]$  then
15:       $y - = 1$ 
16:       $path + = "UP - > "$ 
17:    else if  $F[x - 1, y] < F[x, y - 1]$  then
18:       $x - = 1$ 
19:       $path + = "LEFT - > "$ 
20:    else
21:       $x - = 1, y - = 1$ 
22:       $path + = "LEFT/UP - > UP/LEFT - > "$ 
23:    end if
24:  end while
25:  return  $F[n, m], \quad path$ 
26: end procedure
```

3. 近似解

解法三：贪心算法

1. 因为机器人每移动一次, 若机器人没有移动到最右边或最左边 $(i, j) \quad 1 \leq i < n - 1, 1 \leq j < m - 1$, 就需要判断下一步要移动到右边相邻的位置或者下边相邻的位置, 此时可以遇到两种情况: *A*. 移动到两个位置后硬币数量相同 (对应两个邻居都右硬币或都没有硬币的情况), 以及 *B*. 移动到两个位置后硬币数量不同 (对应其中一个位置有硬币, 另一个位置没有硬币的情况); 所以可以设置指导规则, 判断最优路线
2. 指导规则为: 情况A下, 始终选择右边的相邻位置; 情况B下, 始终选择有硬币的相邻位置;

$$F(i, j) = F(i, j - 1) + M(i, j) \quad \text{if} \quad M(i - 1, j) \leq M(i, j - 1) \tag{2-1}$$

$$F(i, j) = F(i - 1, j) + M(i, j) \quad \text{if} \quad M(i - 1, j) > M(i, j - 1) \tag{2-2}$$

伪代码三

Algorithm 3 Approximate Solution 1: Greedy Algorithm

```
1:  $path[n + m - 1]$ 
2: procedure GREEDY ALGORITHM( $M[n, m], i, j$ )
3:   if  $i < n$  and  $j < m$  then
4:     if  $M(i + 1, j) = M(i, j + 1)$  then
5:        $path.append("RIGHT")$ 
6:        $ans = GREEDY ALGORITHM(i, j + 1)$ 
7:     else if  $M(i + 1, j) > M(i, j + 1)$  then
8:        $path.append("DOWN")$ 
9:        $ans = GREEDY ALGORITHM(i + 1, j)$ 
10:    else if  $M(i + 1, j) < M(i, j + 1)$  then
11:       $path.append("RIGHT")$ 
12:       $ans = GREEDY ALGORITHM(i, j + 1)$ 
13:    end if
14:  else if  $i = n$  then
15:     $ans = \sum_j^m M_{i, j}$ 
16:     $path.append(["RIGHT"] \times (m - j))$ 
17:  else if  $j = m$  then
```

解法四：模拟退火算法

1. 因为贪心算法在硬币密度过低或过高或分布不均匀的情况下很难找到全局最优解，所以设计退火算法引入随机扰动并多次迭代来避免陷入局部最优解
2. 优化解法三的指导规则为：情况A下，完全随机选择两个位置中的一个；情况B下，设置可调节的概率参数 $p \in (0, 1)$ ，使得机器人以 p 的概率移动到
有硬币的位置；
3. 利用初始化温度 $Temperature_{init}$ 和温度变化步长 $Temperature_{change}$ 以及退火停止温度 $Temperature_{min}$ 控制退火循环次数
4. 利用 $Energy = (Temperature_{init} - Temperature_{now})/Temperature_{init}$ 和 $EnergyFix$ 为初始高温时决策的随机性提供较大的动能
(可能性)，动能随着温度降低而降低
5. 记录退火过程中每一次搜索到的最优解和路径，取最大值作为全局最优解的近似解

$$F(i, j) = F(i - 1, j) + M(i, j) \quad \text{if} \quad M(i - 1, j) > M(i, j - 1) \quad \text{and} \quad q - Energy > random(0, 1)$$

(3-1)

$$F(i, j) = F(i - 1, j) + M(i, j) \quad \text{if} \quad M(i - 1, j) = M(i, j - 1) \quad \text{and} \quad random(0, 1) > 0.5$$

(3-2)

伪代码四

Algorithm 4 Approximate Solution 1: Simulated Annealing Algorithm

```
1: procedure SIMULATEDANNEALING( $M[n, m], i, j$ )
2:    $Temperature_{now} = Temperature_{init}$ 
3:   while  $Temperature_{now} > Temperature_{min}$  do
4:      $Energy = (Temperature_{init} - Temperature_{now})/Temperature_{init}$ 
5:     if  $i < n$  and  $j < m$  then
6:       if  $M(i + 1, j) = M(i, j + 1)$  then
7:         if  $random(0, 1) > 0.5$  then
8:            $Path_{temp}.append("RIGHT")$ 
9:            $Ans_{temp} = \text{SIMULATEDANNEALING}(i, j + 1)$ 
10:        else
11:           $Path_{temp}.append("DOWN")$ 
12:           $Ans_{temp} = \text{SIMULATEDANNEALING}(i + 1, j)$ 
13:        end if
14:      else if  $M(i + 1, j) > M(i, j + 1)$  then
15:        if  $p + Energy \times EnergyFix > random(0, 1)$  then
16:           $Path_{temp}.append("DOWN")$ 
17:           $Ans_{temp} = \text{SIMULATEDANNEALING}(i + 1, j)$ 
18:        else
19:           $Path_{temp}.append("RIGHT")$ 
20:           $Ans_{temp} = \text{SIMULATEDANNEALING}(i, j + 1)$ 
21:        end if
22:      else if  $M(i + 1, j) < M(i, j + 1)$  then
23:        if  $p + Energy \times EnergyFix > random(0, 1)$  then
24:           $Path_{temp}.append("RIGHT")$ 
25:           $Ans_{temp} = \text{SIMULATEDANNEALING}(i, j + 1)$ 
26:        else
27:           $Path_{temp}.append("DOWN")$ 
28:           $Ans_{temp} = \text{SIMULATEDANNEALING}(i + 1, j)$ 
29:        end if
30:      end if
31:    else if  $i = n$  then
32:       $Ans_{temp} = \sum_j^m M_{i,j}$ 
33:       $Path_{temp}.append(["RIGHT"] \times (m - j))$ 
34:    else if  $j = m$  then
35:       $Ans_{temp} = \sum_i^n M_{i,j}$ 
36:       $Path_{temp}.append(["DOWN"] \times (m - j))$ 
37:    else
38:      if  $Ans_{temp} > Ans_{max}$  then
39:         $Ans_{max} = Ans_{temp}$ 
40:         $Path_{prop} = Path_{temp}$ 
41:      end if
42:    end if
43:     $Temperature_{now} = Temperature_{now} - Temperature_{chang}$ 
44:  end while
45:  return  $Ans_{max}, Path_{prop}$ 
46: end procedure
```