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列棋盘中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：迭代遍历

1. 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 $P(i,0)=0$

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

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

伪代码一

Algorithm 1 Exact Solution: Iterative Traversal

```
procedure IterativeTraversal(M[n,m])
    F = [n,m]
    F[1,1] = M[1,1]
    for j = 2 to m do
        F[1,j] = F[1,j-1] + M[1,j]
    end for
    for i = 2 to n do
        F[i,1] = F[i-1,1] + M[i,1]
        for j = 2 to m do
            F[i,j] = max(F[i-1,j], F[i,j-1]) + M[i,j]
        end for
    end for
    path = [n+m-1]
    x = n
    y = m
    step = m+n-1
    while x > 1 and y > 1 do
        if F[x-1,y] > F[x,y-1] then
            step = step - 1
            y = y - 1
            path[step] = "UP"
        else if F[x-1,y] < F[x,y-1] then
            step = step - 1
            x = x - 1
            path[step] = "LEFT"
        else
            step = step - 1
            x = x - 1
            y = y - 1
            path[step] = "LEFT/UP"
            step = step - 1
            path[step] = "UP/LEFT"
        end if
    end while
    return F[n,m],   path[n+m]
end procedure
```

3. 近似解

解法二：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；

伪代码二

Algorithm 2 Approximate Solution 1: Greedy Algorithm

$path[n+m-1]$
procedure GreedyAlgorithm($M[n,m],i,j$)
  if $i<n$ **and** $j<m$ **then**
    if $M(i+1,j)=M(i,j+1)$ **then**
      $ans=GreedyAlgorithm(i,j+1)$
      $path.append("RIGHT")$
    else if $M(i+1,j)>M(i,j+1)$ **then**
      $ans=GreedyAlgorithm(i+1,j)$
      $path.append("DOWN")$
    else if $M(i+1,j)<M(i,j+1)$ **then**
      $ans=GreedyAlgorithm(i,j+1)$
      $path.append("RIGHT")$
    end if
  else if $i=n$ **then**
    $ans=\sum_j^m M_{i,j}$
    $path.append(["RIGHT"]\times(m-j))$
  else if $j=m$ **then**
    $ans=\sum_i^n M_{i,j}$
    $path.append(["DOWN"]\times(m-j))$
  else
    return $path,ans$
  end if
end procedure

解法三：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；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
4. 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

伪代码三

Algorithm 3 Approximate Solution 1: Simulated Annealing Algorithm

$Path_{temp}[n+m-1]$
$Path_{prop}[n+m-1]$
$Ans_{max}=0$
$p\in(0,1)$
$Temperature_{init}$
$Temperature_{change}$
$Temperature_{min}$
$EnergyFix$
procedure SimulatedAnnealing($M[n,m],i,j$)
  $Temperature_{now}=Temperature_{init}$
  while $Temperature_{now}>Temperature_{min}$ **do**
    $Energy=(Temperature_{init}-Temperature_{now})/Temperature_{init}$
    if $i<n$ **and** $j<m$ **then**
      if $M(i+1,j)=M(i,j+1)$ **then**
        if $random(0,1)>0.5$ **then**
          $Ans_{temp}=SimulatedAnnealing(i,j+1)$
          $Path_{temp}.append("RIGHT")$
        else
          $Ans_{temp}=SimulatedAnnealing(i+1,j)$
          $Path_{temp}.append("DOWN")$
        end if
      else if $M(i+1,j)>M(i,j+1)$ **then**
        if $p+Energy\times EnergyFix>random(0,1)$ **then**

$Ans_{temp} = SimulatedAnnealing(i+1, j)$
$Path_{temp}.append("DOWN")$
else
$Ans_{temp} = SimulatedAnnealing(i, j+1)$
$Path_{temp}.append("RIGHT")$
end if
else if $M(i+1, j) < M(i, j+1)$ **then**
if $p + Energy \times EnergyFix > random(0, 1)$ **then**
$Ans_{temp} = SimulatedAnnealing(i, j+1)$
$Path_{temp}.append("RIGHT")$
else
$Ans_{temp} = SimulatedAnnealing(i+1, j)$
$Path_{temp}.append("DOWN")$
end if
end if
else if $i = n$ **then**
$Ans_{temp} = \sum_{j}^{m} M_{i,j}$
$Path_{temp}.append(["RIGHT"] \times (m-j))$
else if $j = m$ **then**
$Ans_{temp} = \sum_{i}^{n} M_{i,j}$
$Path_{temp}.append(["DOWN"] \times (m-j))$
else
if $Ans_{temp} > Ans_{max}$ **then**
$Ans_{max} = Ans_{temp}$
$Path_{prop} = Path_{temp}$
end if
end if
$Temperature_{now} = Temperature_{now} - Temperature_{change}$
end while
return $Ans_{max}, Path_{prop}$
end procedure