算法第二次作业

2154312 郑博远

算法分析

2-7

$$P(x)=\prod_{i=1}^{d}(x-n_i)=\prod_{i=1}^{\lfloor d/2\rfloor}(x-n_i)\cdot\prod_{i=\lfloor d/2\rfloor+1}^{d}(x-n_i)=P_1(x)P_2(x)$$

每次将 d 次多项式二分为两个 $d/2$ 次多项式相乘，时间复杂度递归如下：

$$T(d)=\begin{cases}O(1) & d=1\\ 2T(d/2)+O(d\log d) & d\geq 1\end{cases}$$

最顶层（记为第0层）用于合并各子问题的时间复杂度为$O(d\log d)$, 问题被划分为规模为$\frac{d}{2}$的2个子问题;

对于第1层，问题的总规模为 $O(\frac{d}{2})$, 用于合并子问题的时间复杂度为 $O(\frac{d}{2}\log\frac{d}{2})$, 问题被继续划分为规模为$\lfloor\frac{d}{4}\rfloor$的$a$个子问题;
同理可得, 对于第i层, 将会以 $O(\frac{d}{2^i}\log\frac{d}{2^i})$ 的时间复杂度合并子问题;
这样的递归层数共计 $\log d$ 层.
故总的时间复杂度为:

$$T(d)=\Sigma_{i=0}^{\log d}2^i\cdot\frac{d}{2^i}\log\frac{d}{2^i}=O(d\log^2 d)$$

2-15

```python
# 偶数情况直接复制
def copyMatrix(M, ax, ay, bx, by, scope):
    for i in range(scope):
        for j in range(scope):
            M[ax + i][ay + j] = M[bx + i][by + j]
```

```
# 奇数情况有特殊填充规则
def patchMatrix(M, x, y, scope):
    # 计算出的辅助矩阵
    N1 = [[ (i + j) % scope + scope + x for i in range(scope)] for j in range(scope)]
    N2 = [[ (j + scope - i) % scope + x for i in range(scope)] for j in range(scope)]
    # 填入空缺
    for i in range(scope):
        j1 = 0
        j2 = 0
        while(j2 < scope):
            if(M[x + i][y + j1] == 0):
                M[x + i][y + j1]         = N1[i][j2]
                M[x + i + scope][y + j1] = N2[i][j2]
                j2 = j2 + 1
            j1 = j1 + 1


# 分治递归解决问题
def fillMatrix(M, x, y, scope):
    # 递归结束
    if(scope == 2):
        M[x][y]     = x
        M[x+1][y]   = x + 1
        M[x][y+1]   = x + 1
        M[x+1][y+1] = x
        return

    # 若为奇数则虚拟增加一位选手
    subscope  = scope // 2 + scope % 2
    fillMatrix(M, x, y, subscope)
    fillMatrix(M, x + subscope, y, subscope)

    # 两个偶数子矩阵，直接复制
    if(subscope % 2 == 0):
        copyMatrix(M, x + subscope, y + subscope, x, y, subscope)
        copyMatrix(M, x, y + subscope, x + subscope, y, subscope)
    # 两个奇数子矩阵，对子矩阵边长奇数情况进行修正
    else:
        patchMatrix(M, x, y, subscope)

    # 奇数个则移除虚拟增加的选手
    if(scope % 2):
        for i in range(scope + 1):
            for j in range(scope + 1):
                if(i == scope or M[x + i][y + j] ≥ x + scope):
                    M[x + i][y + j] = 0


# 打印矩阵
def printMatrix(M, n):
    for i in range(n):
        for j in range(n + n % 2):
            print(M[i + 1][j + 1], end = ' ')
        print()

n = input()
n = int(n)
M = [[0 for i in range(100)] for j in range(100)]
fillMatrix(M, 1, 1, n)
printMatrix(M, n)
```

算法实现

2-6

```
def getFac(n):
    if(n == 0):
        return 1
    else:
        return n * getFac(n - 1)

def getOrder(list):
    if(len(list) == 0):
        return 0
    else:
        newlist = [(num if num < list[0] else num - 1) for num in list[1:]]
    return getOrder(newlist) + getFac(len(list) - 1) * (list[0] - 1)

def getSeq(sequence):
    i = len(sequence) - 2

    # 倒退找到第一个下降的数
    while(i ≥ 0):
        if(sequence[i] < sequence[i + 1]):
            break
        i = i - 1

    # 找到后续比sequencep[i]小的最大数
    jmin = len(sequence)
    for j in range(i, len(sequence)):
        if(sequence[j] ≤ jmin and sequence[j] > sequence[i]):
            jindex = j
            jmin = sequence[j]

    # 交换
    sequence[i], sequence[jindex] = sequence[jindex], sequence[i]

    # 后半部分转置
    sequence = sequence[:i + 1] + list(reversed(sequence[i + 1:]))
    return sequence

filein = open("input.txt", 'r')
filein.readline()   # 空读一行
filein = filein.read()
sequence = filein.split(' ')
sequence = [int(num) for num in sequence]

fileout = open("output.txt", 'w')
outstr = str(getOrder(sequence)) + '\n'
outstr += ' '.join([str(s) for s in getSeq(sequence)])
fileout.write(outstr)
```

2-9

普通版本的汉诺塔便不会出现异色盘子相叠的情况，因此实现普通汉诺塔即可。代码如下：

```
def Hanoi(n, src, dst, tmp, fileout):
    if(n == 1):
        print(n, src, dst, file = fileout)
        return

    Hanoi(n - 1, src, tmp, dst, fileout)
    print(n, src, dst, file = fileout)
    Hanoi(n - 1, tmp, dst, src, fileout)


filein  = open("input.txt", 'r')
fileout = open("output.txt", 'w')
n = int(filein.read())
Hanoi(n, 'A', 'B', 'C', fileout)
```