算法设计与分析实验

计算机85 张子诚

实验二

一.问题描述

钢条切割 设有一个长度为L的钢条,在钢条上标有n个位置点$(p_1, p_2, \cdots, p_n)$。现在需要按钢条上标注的位置将钢条切割成为$n+1$段,假定每次切割所需要的代价与所切割的钢条长度成正比。请编写一个算法，能够确定一个切割方案，使得总切割代价最小。

二.算法设计与分析

问题分析

首先这种分割问题具有子问题同时又是最值问题，所以很容易让人联想到动态规划。经过分析之后不难发现该问题是矩阵连乘问题的变形。

（1）最优子结构性质：

假设dp[0:n+1] 为整段钢材的包括首尾的每一个切点，假设最优解的切割次序的第一个切点，即整段的切割点为k，则整体最优解包含切割dp[0:k]和切割dp[k:n+1] 的最优解。事实上如果dp[0:k]有代价更少的切割方法，则可以得到dp[0:n+1]代价比最优解更小的切割方案，产生了矛盾。同理可知dp[0:n+1]的最优切割方案包含了dp[k:n+1]的最优切割方案 。由此继续向下分析，可以证明每一步切割后剩余的钢材的切割方案包含于整体最优解在内。

（2）重叠子问题：

在选择切割点求解最优方案的过程中，剩余钢材的切割方案不一定总是新的问题，可能会被反复计算多次。

算法设计

（1）dp table 定义

`dp[i][j]`:代表从第i个断点到第j个断点的最优代价

`P[i]` : 代表第i个断点的位置

（2）状态转移

```
dp[i][j]=0 if i==j
```

```
dp[i][j]=min{dp[i][k]+dp[k][j]+p[j]-p[i]} if i!=j
```

（3）最优解

```
dp[0][n+1]
```

（4）边界条件初值

```
dp[i][i]=0 i=0...n-1
```

代码实现

```python
import numpy as np
# 读取文件
def readfile(path,mode):
    table={
        'write':'w',
        'read':'r'
    }
    with open(path,table[mode],encoding='utf-8') as file:
        lines=file.readlines()
        test_data=[]
        for i in range(0,len(lines),2):
            L,n=list(map(int,lines[i].strip('\n').split(' ')))
            segment=sorted(list(map(int,lines[i+1].strip('\n').split(' '))))
            test_data.append(Data(L,n,segment))
        return test_data

class Data(object):
    def __init__(self,L,n,segment):
        self.L=L
        self.n=n
        self.segment=segment

# 测试结果的正确性
#
def check_correctness(ans):
    corret_ans=[]
    with open('./exp2_out.txt','r',encoding='utf-8') as file:
        for item in file.readlines():
            corret_ans.append(int(item.strip('\n')))
    print('Reference answer:',corret_ans)
    print('my answer:',ans)
    ans,corret_ans=np.array(ans),np.array(corret_ans)
    print((ans-corret_ans).any()==0)




class solution(object):
    """
    dp[i][j]代表从第i个断点到第j个断点的最优代价
    p[i] 表是第i个断点的位置
    递推关系
    dp[i][j]=0 if i==j
    dp[i][j]=min{dp[i][k]+dp[k][j]+p[j]-p[i]}
    最优解:dp[0][n+1]
```

```
    边界条件
    dp
     """
    def __init__(self,n,L,p):
        self.s=[[0]*(n+2) for i in range(n+2)]
        self.n=n
        self.L=L
        self.p=p
    def segmentation(self):
        n=self.n
        L=self.L
        p=self.p
        # 补全p数组
        p.append(L)
        p.insert(0,0)
        size=n+2
        # 初始化
        # dp[i][i]=0
        dp=[[0]*size for i in range(size)]
        for gap in range(2,size):
            for i in range(0,size-gap):
                j=i+gap
                temp=float("inf")
                for k in range(i+1,j):
                    if temp>dp[i][k]+dp[k][j]+p[j]-p[i]:
                        temp=dp[i][k]+dp[k][j]+p[j]-p[i]
                        self.s[i][j]=k
                dp[i][j]=temp
        return dp[0][n+1]

    def traceback(self,i,j):
        if (j-i==1):
            return
        print("{}~{}在{}分割".format(i,j,self.s[i][j]))
        self.traceback(i,self.s[i][j])
        self.traceback(self.s[i][j],j)




if __name__=='__main__':
    '''
    n,L=4,7
    p=[1,3,4,5]
    sol=solution(n,L,p)
    print("The number of segment:{}".format(n))
    print("Total Length:{}".format(L))
    print("Breaking point:{}".format(p))

    ans=sol.segmentation()
    print("Total Cost:{}".format(ans))
    sol.traceback(0,n+1)
    '''
    Dataset=readfile('./exp2_in.txt','read')
    ans=[]
    for item in Dataset:
        n,L,segment=item.n,item.L,item.segment
        sol=solution(n,L,segment)
        ans.append(sol.segmentation())

    check_correctness(ans)
```

```
108
```

测试结果

```
PS D:\study\algorithm\ex\ex2> & C:/ProgramData/Anaconda3/envs/python37/python.exe d:/study/algorithm/ex/ex2/ex2.py
Reference answer: [2, 14, 20, 421, 474, 7232, 5240, 67145, 724145, 626176]
my answer: [2, 14, 20, 421, 474, 7232, 5240, 67145, 724145, 626176]
True
PS D:\study\algorithm\ex\ex2>
```

参考结果

exp2_out.txt

```
2
14
20
421
474
7232
5240
67145
724145
626176

```