**集美大学实验报告**

**班级 计算2114 学号 202121331104 姓名 庄佳强 成绩**

**课程 《算法设计与分析》 日期 2024 年 4 月 23 日**

|  |  |  |  |
| --- | --- | --- | --- |
| **实验项目** | **实验三 动态规划法的应用** | **指导老师** | **杨艳华** |
| **实验目的** | 掌握动态规划法的使用 | | |
| **一、实验内容**  1.最大子段和问题。给定由n个整数组成的序列，求序列中子段的最大和，若所有整数均为负整数时定义最大子段和为0。  2.给定n个矩阵｛A1,A2,…,An｝（n<=40），其中Ai与Ai+1是可乘的，i=1，2…，n。第i个矩阵的维数用pi−1​，pi​来表示。如何确  定计算矩阵连乘积的计算次序，使得依此次序计算矩阵连乘积需要的数乘次数最少。例如，给定三个连乘矩阵{A1，A2，A3}的维数 数组p为：10,100,5,50，即分别是10 ×100，100×5和5×50，采用（A1A2）A3，乘法次数为10×100×5+10×5×50=7500次，而采用A1（A2A3），乘法次数为100×5×50+10×100×50=75000次乘法，显然，最好的次序是（A1A2)A3，乘法次数为7500次。  3.一种双核CPU的两个核能够同时处理任务，现在有n个已知数据量的任务需要交给CPU处理，假设已知CPU的每个核1秒可以处理1KB，每个核同时只能处理一项任务，n个任务可以按照任意顺序放入CPU进行处理。编写一个程序求出一个设计方案让CPU处理完这批任务所需的时间最少，求这个最少的时间。 | | | |
| **二、问题分析**  1.最大字段和问题，可以换为从下标为i到j的序列的最大值。其中i<=j  写为动态规划方程为  dp[i]=dp[i-1]+A[i] dp[i-1]>0  dp[i]=A[i] dp[i-1]<0  这样就可以得出从0-n中i当前的最大值字段和，可以解决问题。   1. 矩阵连乘的条件：第一个矩阵的列等于第二个矩阵的行，此时两个矩阵是可乘的；多个矩阵连乘的结果矩阵，其行列等于第一个矩阵的行和最后一个矩阵的列；两个矩阵相乘的计算量矩阵Am\*n和Bn\*k的乘法运算次数为：m\*n\*k。   假设先求i到j矩阵的连乘最小值dp[i][j]，假设在i到j中的k位置找到最小值，问题转化为dp[i][k]+dp[k][j]+A[i-1]\*A[k]\*A[j]，其中A表示矩阵边数，A[i-1]\*A[k]\*A[j]表示把两个矩阵连接起来。  所以动态转化方程为：  dp[i][k]=Min(dp[i][k]+m[k+1][j]+A[i-1]\*A[k]\*A[j]，dp[i][j]) i<j i<k<j   1. 双核心问题可以优化为单核心问题，通过贪心算法，我们想要两个核心都可以把任务时间对半吃，这样就可以让时间最短。这样一个核心的时间就是任务总时间的一半。优化为了0/1背包问题。   可以轻易写出方程：  dp[j]=max(dp[j],dp[j-A[i]]+A[i]) A[i]<=j | | | |
| **三、数据结构定义**  1.定义了一个dp数组用来存放值，定义了一个arr数组存放输入值。  2.定义了一个二维数组dp用来存放i到j的最大值，一个arr数组用来存储输入值。  3.定义了一个dp数组用来存放值。 | | | |
| **四、算法伪代码描述**  1.  function process(arr: array of int, n: int) -> int {  dp = new array of int with size n + 1  for i from 0 to n:  dp[i] = 0  max = 0  for i from 1 to n:  if dp[i - 1] >= 0:  dp[i] = dp[i - 1] + arr[i - 1]  else:  dp[i] = arr[i - 1]  flag = i    if dp[i] > max:  max = dp[i]  start = flag//获得最大值更新  end = i    return max  }  2.  function process(arr: array of int, n: int) -> int {  dp = new 2D array of int with size (n + 1) x (n + 1)  for r from 2 to n:  for i from 1 to (n - r + 1):  j = i + r - 1  dp[i][j] = dp[i][i] + dp[i + 1][j] + arr[i - 1] \* arr[i] \* arr[j]  for k from i + 1 to j - 1:  dp[i][j] = min(  dp[i][k] + dp[k + 1][j] + arr[i - 1] \* arr[k] \* arr[j],  dp[i][j]  )  return dp[1][n]  }  3.  函数 process(arr, n):  sum = 计算 arr 中所有任务的数据量之和  avc = sum / 2 // 平均每个核心需要处理的数据量  temp = avc >> 10  avc = temp << 10  avc\_ = ((sum - avc) >> 10) << 10  初始化 dp 数组，大小为 avc + 1，全部初始化为 0  对于 i 从 0 到 n-1：  对于 j 从 avc 到 arr[i]，递减：  dp[j] = max(dp[j - arr[i]] + arr[i], dp[j])  返回 max(dp[avc], avc\_) | | | |
| **五、算法时间和空间复杂度分析（要求写出详细过程）**   1. 函数值只使用了一个循环遍历数组，所以时间复杂度为O(n)。   函数中的dp和arr都是一维数组，大小为n+1,所以空间复杂度为O(n)。   1. 第一次循环执行了n-1次，第二次循环进行了n-r+1次，第三次循环进行了n/2次   所以总的时间复杂度为O(n^3)。  函数中的dp二维数组的大小为(n+1)^2，所以空间复杂度为O（n^2）。   1. 第一次循环执行了n次，第二次循环进行要看avc的值。所以时间复杂度为O(n\*avc)。   空间  函数中的dp和arr都是一维数组，大小为n+1,所以空间复杂度为O(n)。 | | | |
| **六、错误记录和解决**  2.一开始不知道矩阵乘法怎么做，一时间不怎么怎么下手？  解决方法：先复习了一下矩阵乘法后再做就好做多了。  2.因为是二个数字为一个矩阵，一开始没有注意，导致数组下标一直报错。后面同学提醒后就解决了。   1. 一开始想到了0/1背包问题，但是有两个核心不知道怎么进行分配。   解决方法：后来想到的要最大值是不是应该要均分，然后想到了贪心算法，就把任务总量对半分，然后就转化为了0/1背包问题。 | | | |
| **七、心得体会**  总的来说动态规划是用少量的空间来代替递归过程中存值。使得不需要使用递归来解决问题。  动态规划常常通过自底向上的方式求解，即先解决最小规模的子问题的最值，然后逐步求解规模更大的问题最值。这种方法有效地避免了递归过程中的重复计算，提高了算法的效率。  在实际应用中，可以通过状态压缩或其他手段进行空间优化，减少存储空间的使用。这对于处理大规模数据时尤其重要。 | | | |