《算法分析与设计》

实 验 报 告

学 号

姓 名

年 级 2020

专 业 软件工程

院 系 计算机与人工智能学院

二0二二年三月

目 录

实验一 求两个圆相交部分的面积······································1

实验二 算法效率分析与比较·········································11

实验三 穷举法设计与实验···········································11

实验四 分治法设计与实验···········································11

实验五 动态规划算法设计与实验·····································11

# 实验5.4动态规划在实际中的应用

1. 实验目的

|  |
| --- |
| 1. 理解动态规划算法的求解过程。 2. 分析动态规划算法的时间复杂度，比较动态规划算法与其他算法的效率差异 3. 学会如何利用动态规划算法求解具体问题，了解动态规划算法的局限性。 |

1. 实验任务

|  |
| --- |
| 有一个二维矩阵，矩阵中元素有正也有负。定义子矩阵的和为其所有元素之和，最大子矩阵为子矩阵和值最大的子矩阵。  输入：输入的第一行为N，表示矩阵的行数和列数。其后的N行每行包含N个元素，表示矩阵每行元素的值，元素之间用空格隔开。  输出：输出一行，包含一个整数，表示其最大子矩阵的和。  样例输入：4  0 -2 -7 0  9 2 -6 2  -4 1 -4 1  -1 8 0 -2  要求：   1. 将矩阵的某一行看成一个序列，设计算法求该序列的最大子序列。 2. 将矩阵相邻两行对应列的元素相加形成一个新的序列。试分析该子序列中每一个元素的含义，其最大子序列的含义。 3. 设计算法求解该问题，分析样例输入情况下算法的执行过程，主要变量的变化情况，并分析算法的时间复杂度。 4. 编写程序实现（3）中的算法。 5. 上机实验，验证程序执行过程与（3）中的执行过程是否一致。   撰写相应的实验报告，实验报告内容包括：实验目的、实验任务、实 验环境、实验步骤、实验结果及其分析以及实验总结等部分内容。 |

1. 实验环境

|  |
| --- |
| * 1. 硬件环境  1. 计算机：拯救者R7000P 2020H 2. CPU: AMD Ryzen 7 4800H with Radeon Graphics 2.90 GHz 3. RAM：16GB    1. 软件环境 4. 操作系统：Windows11家庭中文版 5. 开发工具：Visual Studio Code |

1. 实验步骤及结果

|  |
| --- |
| * 1. 实验预习   4.1.1求最大子列和  只用了两层循环，相当于是第一层用来找到每个可能的子序列的起点，而第二个循环直接从第一层确定的起点往数组结尾遍历，在遍历的过程中同时也计算序列值。把所有可能的子序列都找出来，然后计算值再和最大值比较。  算法代码  int MaxSubsequence(int *m*[10], int *N*)*//最大子列和*  {      int s = 0, max = 0;      int i, j;      for (i = 0; i < *N*; i++)      {          s = 0;          for (j = i; j < *N*; j++)              s += *m*[j];          max = s > max ? s : max;      }      if (max <= 0)          return 0;      else          return max;  }  4.1.2最大子序列含义  子序列每个元素含义：2\*n矩阵中每列元素之和；  最大子序列含义：上下两行合并后对应序列中任意子序列和的最大值。  4.1.3最大子矩阵的和  样例输入下的执行情况：  *4*  0 -2 -7 0  9 2 -6 2  -4 1 -4 1  -1 8 0 -2  根据算法，程序执行情况和主要变量变化为  (1)matrix[1]=0,matrix[2]=0,matrix[3]=0,matrix[4]=0  (2)matrix[1]=0,matrix[2]=-2,matrix[3]=-7,matrix[4]=0,maximal=0  (3)matrix[1]=9,matrix[2]=0,matrix[3]=-13,matrix[4]=2,maximal=9  (4)matrix[1]=5,matrix[2]=1,matrix[3]=-17,matrix[4]=3,maximal=9  (5)matrix[1]=4,matrix[2]=9,matrix[3]=-17,matrix[4]=1,maximal=13  (6)matrix[1]=0,matrix[2]=0,matrix[3]=0,matrix[4]=0,maximal=13  (7)matrix[1]=9,matrix[2]=2,matrix[3]=-6,matrix[4]=2,maximal=13  (8)matrix[1]=5,matrix[2]=3,matrix[3]=-10,matrix[4]=3,maximal=13  (9)matrix[1]=4,matrix[2]=11,matrix[3]=-10,matrix[4]=1,maximal=15  (10)matrix[1]=0,matrix[2]=0,matrix[3]=0,matrix[4]=0,maximal=15  (11)matrix[1]=-4,matrix[2]=1,matrix[3]=-4,matrix[4]=1,maximal=15  (12)matrix[1]=-5,matrix[2]=9,matrix[3]=-4,matrix[4]=-1,maximal=15  (13)matrix[1]=0,matrix[2]=0,matrix[3]=0,matrix[4]=0,maximal=15  (14)matrix[1]=-1,matrix[2]=8,matrix[3]=0,matrix[4]=-2,maximal=15  4.1.5时间复杂度分析  SubMaxMatrix中首先有三个嵌套for循环，进行不同组合排序的循环次数为n+(n-1)+(n-2)+...+1=n\*(n+1)/2，时间复杂度认为O（n^2）,由f变量控制的循环中的循环，由于直接更改matrix，忽略该层循环的影响，而MaxSubsequence的时间复杂度为O（n）,因此SubMaxMatrix的总时间复杂度为O（n^3）  4.1.4程序代码  #include<stdio.h>  #include<Windows.h>  int matrix[10][10] = {0};  int N;  int MaxSubsequence(int *m*[10], int *N*)*//最大子列和*  {      int s = 0, max = 0;      int i, j;      for (i = 0; i < *N*; i++)      {          s = 0;          for (j = i; j < *N*; j++)              s += *m*[j];          max = s > max ? s : max;      }      if (max <= 0)          return 0;      else          return max;  }  int SubMaxMatrix(int *matrix*[10][10], int *N*)  {      int total[10][10] = {0};  *//total[10][10] = matrix[10][10];*      for (int i = 0; i < *N*; i++)          for (int j = 0; j < *N*; j++)              total[i][j] = *matrix*[i][j];      for (int i = 1; i < *N*; i++)          for (int j = 0; j < *N*; j++)              total[i][j] += total[i - 1][j];      int maximum = 0;      for (int i = 0; i < *N*; i++)      {          for (int j = i; j < *N*; j++)          {              int result[10] = {0}; *//result保存从i行到第j行所对应的矩阵上下值的和*              for (int f = 0; f < *N*; f++)              {                  if (i == 0)                      result[f] = total[j][f];                  else                      result[f] = total[j][f] - total[i - 1][f];              }              int maximal = MaxSubsequence(result,*N*);              if (maximal > maximum)                  maximum = maximal;          }      }      return maximum;  }  int main()  {      scanf("%d", &N);      for (int i = 0; i < N; i++)          scanf("%d %d %d %d", &matrix[i][0], &matrix[i][1], &matrix[i][2], &matrix[i][3]);      int max = SubMaxMatrix(matrix, N);      printf("%d\n", max);      system("pause");      return 0;  }  4.2上机实验  4.2.1算法测试  样例输入：  4  0 -2 -7 0  9 2 -6 2  -4 1 -4 1  -1 8 0 -2  部分运行和结果如图所示：        4.2.2测试结果及其分析  运行结果如图所示：    结果与实验分析吻合，算法正确 |

5.实验总结

|  |
| --- |
| 理解动态规划算法的求解过程，分析动态规划算法的时间复杂度，比较动态规划算法与其他算法的效率差异，学会如何利用动态规划算法求解具体问题。 |