《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4.2 分治算法搜索目标值

1. 实验目的

|  |
| --- |
| 1. 理解分治算法的求解过程。 2. 通过范例学习分治策略设计技巧，学会分析分治算法的时间复杂度。 3. 掌握用分治算法求解具体问题，了解其面临的瓶颈。 |

1. 实验任务

|  |
| --- |
| 编写一个分治算法来搜索mxn矩阵matrix中的一个目标值 target，该矩阵具有以下特性：每行的元素从左到右升序排列。每列的元素从上到下升序排列。 |

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采用二分搜索策略实现问题求解程序，验证输入输出结果，并对下述三种设计算法的时间复杂度进行对比分析  蛮力法：蛮力法就是一个一个地查找，时间复杂度和矩阵规模有关，所以对于n\*n矩阵来说，时间复杂度为O(n^2）；  二分法搜索策略：二分法从左下角开始，对于左下角，向右走数字变大，向上走数字变小。当target比当前值大时，向右查找；当target比当前值小时，向上查找，直到找到target，否则返回false。由分析可知，程序最多执行2\*n次，也即时间复杂度为O(n);  搜索空间缩减策略：   1. 步骤1：mid=中间列的索引，计算row使得matrix[row-1][mid]<=target<=matrix[row][mid]，如果target=matrix[row-1][mid]或者target=matrix[row][mid]，则返回true；否则，进行步骤2； 2. 步骤2：通过步骤1可知，matrix[row-1][mid]<target<matrix[row][mid]严格成立，据此将矩阵划分成四个小矩阵。左上角和右下角矩阵是不可能的，target只可能落在左下角和右上角矩阵内； 3. 步骤3：对左下角和右上角矩阵做递归   由分析可知时间复杂度为O(nlgn)  4.1.2程序代码  #include<stdio.h>  #include<Windows.h>  bool BruteForce(int *matrix*[5][5], int *tar*) *//蛮力法*  {      int target = *tar*;  *//矩阵为空的情形*  *//if (len(matrix) == 0)*  *//return False;*  *//矩阵非空的情形*  *//行列数*  *//两层遍历*      for (int i = 0; i < 5; i++)          for (int j = 0; j < 5; j++)              if (*matrix*[i][j] == target)                  return true;      return false;  }  bool Bisection(int *matrix*[5][5], int *tar*)*//二分法搜索策略*  {      int target = *tar*;  *//行索引*      int i = 4;  *//列索引*      int j = 0;      while(1){          if (target== *matrix*[i][j])          {              return true;              break;          }          else          {              if(target > *matrix*[i][j])                  j += 1;              else              {                  i -= 1;                  if (i < 0 || j > 5)                  {                      return false;                      break;                  }              }          }      }  }  bool SearchRec(int *tar*,int *left*, int *right*, int *up*, int *down*) *//搜索空间缩减策略*  {  *// if (left > right || up > down)*  *//   return false;*      int target = *tar*;      if (target < matrix[*up*][*left*] || target > matrix[*down*][*right*])          return false;      else      {          int row = *up*; *//行*          int mid = (*left* + *right*) / 2; *//中间列的索引*          while (row <= *down* && target >= matrix[row][mid])          {              if (target == matrix[row][mid])                  return true;              row++;          }          return SearchRec(*tar*, *left*, mid - 1, row, *down*) || SearchRec(*tar*, mid + 1, *right*, *up*, row - 1);      }  }  bool Search(int *matrix*[5][5], int *tar*)  {      return SearchRec(*tar*,0, 4, 0, 4);  }  int main()  {      int matrix[5][5];      int target, s;      bool t = false;      printf("请输入5\*5矩阵,matrix=\n");      for (int i = 0; i < 5; i++)          for (int j = 0; j < 5; j++)              scanf("%d", &matrix[i][j]);      printf("请输入目标,target=");      scanf("%d", &target);      printf("请选择搜索策略(1->蛮力法,2->二分法,3->搜索空间缩减策略)\n");      scanf("%d", &s);      if (s == 1)          t = BruteForce(matrix, target); *//蛮力法*      if (s == 2)          t = Bisection(matrix, target); *//二分法搜索策略*      if (s == 3)          t = Search(matrix, target); *//搜索空间缩减策略*      if (t == true)          printf("true\n");      else          printf("false\n");      system("pause");      return 0;  }  测试结果如图：      4.2上机实验  4.2.1算法测试  4.2.2测试结果及其分析  5.实验总结 |