《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5.2动态规划时间复杂度分析比较

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

|  |
| --- |
| 已知某问题采用动态规划算法求解的程序，要求：   1. 将该程序dp函数修改为采用递归调用实现。 2. 分析两种不同算法的时间复杂度。 3. 上机实验，绘制曲线比较两种算法在相同输入情况下的运行时间，并与(2)中的分析结果进行对比。   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时间复杂度分析  动态规划：  递归的顺序是依次查询当前位置的上下左右的四个位置，寻找较小的位置，递归并利用vis数组标记路径；对任意一个结点来说，最长递减路径=最长子结点递减路径+1，由于每个结点只能被搜索一次。所以时间复杂度与输入规模R和C有关，而且每个位置要进行一次遍历，所以时间复杂度为O（R\*C）；  递归：  算法的输入规模是：R\*C  最优情况：所有数据都相等，算法时间复杂度为O（R\*C）；  最差情况：如表所示：   |  |  |  | | --- | --- | --- | | 1 | 2 | 3 | | 8 | 9 | 4 | | 7 | 6 | 5 |       最长递减路径为9->8->7->6->5->4->3->2->1,一般情况时，假设从（i,j）开始，部分树形结构如下图所示：    递归的顺序是依次查询当前位置的上下左右的四个位置，寻找较小的位置，所以宽度为4,而每个位置都要遍历，可得算法时间复杂度为O（4^(R\*C)）。  4.1.2程序代码  #include <stdio.h>*//动态规划*  #include <string>  int vis[101][101];  int map[101][101];  int R, C;  int dp(int *i*, int *j*)  {      int max = 0;      if (vis[*i*][*j*] > 0) return vis[*i*][*j*];      if (*i* - 1 >= 0)          if (map[*i* - 1][*j*] < map[*i*][*j*])              if (max < dp(*i* - 1, *j*))                  max = dp(*i* - 1, *j*);      if (*i* + 1 < R)          if (map[*i* + 1][*j*] < map[*i*][*j*])              if (max < dp(*i* + 1, *j*))                  max = dp(*i* + 1, *j*);      if (*j* - 1 >= 0)          if (map[*i*][*j* - 1] < map[*i*][*j*])              if (max < dp(*i*, *j* - 1));                  max = dp(*i*, *j* - 1);      if (*j* + 1 < C)          if (map[*i*][*j* + 1] < map[*i*][*j*])              if (max < dp(*i*, *j* + 1))                  max = dp(*i*, *j* + 1);      return vis[*i*][*j*] = max + 1;  }  int main()  {      int i, j, ans, max;      scanf("%d%d", &R, &C);      for (i = 0; i < R; i++)          for (j = 0; j < C; j++)              scanf\_s("%d", &map[i][j]);      max = 0;      for (i = 0; i < R; i++)      {          memset(vis[i], -1, sizeof(vis[i]));          for (j = 0; i < C; j++)          {              ans = dp(i, j);              if (ans > max)                  max = ans;          }      }      printf("%d\n", max);      system("pause");      return 0;  }  #include <stdio.h>*//递归*  #include <string>  int map[101][101];  int R, C;  int dp(int *i*, int *j*)  {      int max = 0;      if (*i* - 1 >= 0)          if (map[*i* - 1][*j*] < map[*i*][*j*])              if (max < dp(*i* - 1, *j*))                  max = dp(*i* - 1, *j*);      if (*i* + 1 < R)          if (map[*i* + 1][*j*] < map[*i*][*j*])              if (max < dp(*i* + 1, *j*))                  max = dp(*i* + 1, *j*);      if (*j* - 1 >= 0)          if (map[*i*][*j* - 1] < map[*i*][*j*])              if (max < dp(*i*, *j* - 1));                  max = dp(*i*, *j* - 1);      if (*j* + 1 < C)          if (map[*i*][*j* + 1] < map[*i*][*j*])              if (max < dp(*i*, *j* + 1))                  max = dp(*i*, *j* + 1);      return max + 1;  }  int main()  {      int i, j, ans, max;      scanf("%d%d", &R, &C);      for (i = 0; i < R; i++)          for (j = 0; j < C; j++)              scanf\_s("%d", &map[i][j]);      max = 0;      for (i = 0; i < R; i++)      {          for (j = 0; i < C; j++)          {              ans = dp(i, j);              if (ans > max)                  max = ans;          }      }      printf("%d\n", max);      system("pause");      return 0;  }  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

|  |
| --- |
|  |