《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

实验六 贪心算法设计与实验·········································11

实验七 回溯法设计与实验···········································11

# 实验7.2 回溯法时间复杂度分析比较

1. 实验目的

|  |
| --- |
| (1)理解回溯法的求解过程。  (2)分析回溯法的时间复杂度，比较回溯法算法与其他算法的时间效率差异。  (3)学会如何利用回溯法求解具体问题，了解动回溯法的应用范围及在实际应用中的局限性。 |

1. 实验任务

|  |
| --- |
| (1)分析实验7.1中算法的时间复杂度。  (2)采用动态规划算法求解实验7.1中的问题，分析其算法时间复杂度。  (3)分析比较(1)和(2)两种算法的特点及适用范围。  (4)实验比较回溯法及动态规划算法程序的运行时间与城市数量之间的关系，并与前面的分析结果进行比较。  (5)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实验7.1时间复杂度分析  Backtrack函数，最坏情况，所有点直接都联通，其递归时类似全排列，其时间复杂度为：  T(n)=(n-1)T(n-1)=(n-1)(n-2)T(n-2)=(n-1)(n-2)(n-3)…2T(1) （1）  又T(1)=O(n)，故Backtrack()的时间复杂度为  T(n)=(n-1)!O(n)=O(n!) （2）  该算法的时间复杂度为  T(n)=O(n!)+O(n)+O(1)=O(n!) （3）  4.1.2程序代码  #include <iostream>  #include <math.h>  #include <cstring>  using namespace std;  #define MAX 10  #define MAX\_s 1023  int n = 0;  int G[MAX + 1][MAX + 1] = { 0 };  int dp[MAX][MAX\_s] = { 0 };  int TSP()  {      for (int i = 1; i <= n; i++)          dp[i][0] = G[i][1];      for (int j = 1; j < pow(2, n - 1); j++)      {          for (int i = 1; i <= n; i++)          {              dp[i][j] = 9999;              if (((j >> (i - 2)) & 1) != 1)                  for (int k = 2; k <= n; k++)                      if (((j >> (k - 2)) & 1) != 0)                          dp[i][j] = min(G[i][k] + dp[k][j ^ (1 << (k - 2))], dp[i][j]);          }      }      return 0;  }  int main()  {      cin >> n;      int c = 0;      memset(G, 0, sizeof(G));      for (int i = 1; i <= n; i++)      {          for (int j = i + 1; j <= n; j++)          {              cin >> c;              G[i][j] = c;              G[j][i] = c;          }      }      TSP();      cout << dp[1][(int)pow(2, n - 1) - 1];      system("pause");      return 0;  }  时间复杂度分析：  函数dp中采用三重for循环，最内层语句的执行次数为：  C(n)=n\*\*n= （4）  故函数dp的时间复杂度为O()，  算法的时间复杂度为  T(n)=O()+O(N)+O(1)=O( （5）  从时间复杂度来看动态规划优于回溯法。  动态规划的特点以及适用范围：  特点：  1．每个子问题只求解一次，可以使用前面已经求解出的结果；  2．难于寻找最优解的结构特征。  适用范围：需要满足最优化原理。  回溯算法的特点以及适用范围：  特点：  1.与穷举法类似，存在大量重复计算，效率较低；  2.程序实现简单。  适用范围：几乎适用于所有问题，可以通过约束函数或者限界函数去除不必要的分支，从而提高搜索速度。  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

|  |
| --- |
|  |