《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7.4 搭积木

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

|  |
| --- |
| 一个小孩手中有N块正方形的积木，他总是想不同的方法来搭建各种 不同的楼梯。他搭建的楼梯必须满足如下条件：  （1） 楼梯每个台阶的砖块数不能相同，且严格递减。  （2） 每个楼梯至少包含两个台阶  （3） 必须用完所有的积木  请编写程序计算该小孩最多有多少种满足上述条件的不同的搭建方法。  输入要求：  输入1行，积木的个数N, N大于等于3。  输出要求：输出1行，有多少种不同的楼梯搭建方法。  输入样例：  10  输出样例:  9  **实验预习:**  （1） 写出采用回溯法求解上述问题的目标函数，约束条件以及限界函数。  （2） 画出采用回溯法在求解样例输入时的解空间树。  （3） 画出采用回溯法求解样例输入时的搜索空间树。  （4） 采用回溯法求解上述问题，写出算法的实现过程，简要分析算法的时间复杂度。  （5） 编写相应的程序。  **上机实验：**  （6） 上机验证程序在样例输入时的执行过程是否与步骤（3）所得到的搜索空 间树一致，写出程序调试过程中搜索空间树上每个结点的值，并与前面的结果进行比对。  （7） 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写出采用回溯法求解上述问题的目标函数，约束条件以及限界函数  void backtracking(int *targetsum*, int *sum*, int *k*, int *startIndex*) *//总积木数，当前积木数，最大深度，从第一块开始*  目标函数：              if (*sum* == *targetsum*) *//积木用完，此情况成立*  约束条件：              for (int i = *startIndex*; i < *targetsum*; i++) *//每阶尝试不同个数的积木数，但需小于总积木数*  限界函数：              if (*k* < 0 || *sum* > *targetsum*) *//到达最底层，和是否大于总积木数*  4.1.2画出采用回溯法在求解样例输入时的解空间树    4.1.3画出采用回溯法求解样例输入时的搜索空间树    4.1.4采用回溯法求解上述问题，写出算法的实现过程，简要分析算法的时间复杂度  class Solution  {      public:          int ans = 0;          void backtracking(int *targetsum*, int *sum*, int *k*, int *startIndex*) *//总积木数，当前积木数，最大深度，从第一块开始*          {              if (*k* < 0 || *sum* > *targetsum*) *//到达最底层，和是否大于总积木数*                  return;              if (*sum* == *targetsum*) *//积木用完，此情况成立*              {                  ans++;                  return;              }              for (int i = *startIndex*; i < *targetsum*; i++) *//每阶尝试不同个数的积木数，但需小于总积木数*              {  *sum* = *sum* + i; *//当前积木总和*                  backtracking(*targetsum*, *sum*, *k* - 1, i + 1);  *sum* = *sum* - i; *//回溯*              }          }          void combine(int *n*)          {              int k = (*n*+1) / 2; *//最大深度，样例中k=5，即有5层*              backtracking(*n*, 0, k, 1);          }  };  算法时间复杂度：  输入规模为1；  基本操作为ans的自加，时间复杂度为n，每次n自减1；  操作次数与输入规模有关，层数k=（n+1）/2；  所以时间复杂度为1/2（n+1）\*n！，即O（n）=（n+1）！  4.1.5编写相应的程序  #include <iostream>  using namespace std;  class Solution  {      public:          int ans = 0;          void backtracking(int *targetsum*, int *sum*, int *k*, int *startIndex*) *//总积木数，当前积木数，最大深度，从第一块开始*          {              if (*k* < 0 || *sum* > *targetsum*) *//到达最底层，和是否大于总积木数*                  return;              if (*sum* == *targetsum*) *//积木用完，此情况成立*              {                  ans++;                  return;              }              for (int i = *startIndex*; i < *targetsum*; i++)              {  *sum* = *sum* + i; *//当前积木总和*                  if (*sum* > *targetsum*)                      return;                  backtracking(*targetsum*, *sum*, *k* - 1, i + 1);  *sum* = *sum* - i; *//回溯*              }          }          void combine(int *n*)          {              int k = (*n*+1) / 2; *//最大深度，样例中k=5，即有5层*              backtracking(*n*, 0, k, 1);          }  };  int main()  {      int n;      cin >> n;      Solution A;      A.combine(n);      cout << A.ans << endl;      system("pause");      return 0;  }  4.2上机实验  4.2.1算法测试  输入样例：  10  输出样例：  9  4.2.2测试结果及其分析  搭建第一种楼梯时的各节点变化，由中间变量显示如下：    由截图可知，第一种楼梯的每个节点分别为1，2，3，4，总数为10，且符合严格递减要求  第二种楼梯每个节点分别为2，3，5，结果如图所示：    第三种楼梯每个节点分别为1，4，5，结果如图所示：    第四种楼梯每个节点分别为1，3，6，结果如图所示：    第五种楼梯每个节点分别为4，6， 结果如图所示：    第六种楼梯每个节点分别为1，2，7，结果如图所示：    第七种楼梯每个节点分别为3，7，结果如图所示：    第八种楼梯每个节点分别为2，8，结果如图所示：    第九种楼梯每个节点分别为1，9. |

5.实验总结

|  |
| --- |
| 通过具体实例学习了回溯法的分析求解过程，时间复杂度分析方法，与其他算法的效率差异，以及如何设计回溯法解决实际问题。理解了回溯法的特点以及在实际应用中如何正确设计并使用回溯算法。 |