山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202000130198 | 姓名： 隋春雨 | | 班级： 20.4 |
| 实验题目：递归练习 | | | |
| 实验学时：2 | | 实验日期： 2021-09-30 | |
| 实验目的：  1、 熟悉开发工具的使用。  2、 掌握递归的实现思想。 | | | |
| 软件开发环境：  CLION2020 | | | |
| 1. 实验内容   ①题目描述：  现有一个有n个元素的序列a=[a1,a2,⋯,an]，定义这个序列的价值为 。空序列的价值为0。先给你一个长度为n的序列a，求a中所有子集价值的异或和，要求子集中元素的相对位置保持不变。  异或和: 位运算的一种。如果a、b两个值不相同，则异或结果为1；如果a、b两个值相同，异或结果为0。  输入输出格式：  输入：第一行，一个整数n  接下来一行有n个非负整数：a1,a2,⋯,an  输出：一个整数，表示所有子集价值的异或和。  ②现有一个有n个元素的序列a=[a1,a2,⋯,an]，定义其价值为  给出这样一个序列，求其所有排列的价值vi或  其中∣为位运算或操作，⊕为位运算异或操作。  输入输出格式：  输入：输入的第一行是一个整数n (2<=n<=10)，表示需排列的数的个数。  接下来一行是n个整数，数的范围是0到100000，每两个相邻数据间用一个空格分隔。  输出：  一个整数，代表所有排列价值的或。   1. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法） 2. A题思路、算法与数据结构   数据结构：数与集合  思路与算法：**dfs**,类似于树一样的数据结构（没有画出全部的节点），并且我们设置第一行的点为一个虚点。    我们结合**深度优先搜索**思想考虑每一层，发现，如果想要求出这个集合的所有子集，有一个思路就是求出元素数目为N-1个元素的所有子集，再分析剩下的元素是否加入。其中，递归终止条件为集合  没有元素（元素数目为0）。  (2) B题思路、算法与数据结构  数据结构：树与集合。  思路与算法：**dfs**,同样采用递归查询子集（或者结合树的性质运用dfs），最终递归的终止条件为递归次数等于数组的size。此时需要查询数组中元素是否被选中。值得注意的是，在计算价值的时候，需要开辟一片新的空间进行数组元素的记录，因为我们在扫描原来数组的时候，需要用cnt记录当前的id，方便记录价值。   1. 测试结果（测试输入，测试输出） 2. A题样例      1. A题自造数据      1. B题样例      1. B题自造数据（边界条件）     经过手工计算，该结果正确   1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   (1)有没有别的方法（比如说不用递归），求解这道题？  解决：从上面的那幅图可以看到，这是一个树的数据结构，我们可以考虑用dfs。结合需要列举出它的全部子集，我们可以用二进制来表示每一个元素是否出现。最终这道题的实质就变成了，给你一串0-1序列，枚举出它的一切可能取值。枚举二进制代码如下：    (2)通过递归调用和（2）中的枚举二进制哪个更快？  解决：结合《计算机系统基础》的知识，递归需要压栈，保存函数的相关信息，因此通常情况下，枚举循环是要比递归快的。  (3)对于递归调用中的复原现场操作，可不可以去掉？（代码如下）    答：不可以，因为我们如果不复原现场，在i增大之后，有可能进行的第一个swap会跟之前已经进行全排列的数字进行交换，最终这个数字重复进行了全排列。我们可以测试1,2,3这组数据，结果如下：    可以看出，排列出来的全排列有重复的，也就是说，对于每一次递归结束后的恢复现场都是必要的。  (4)异或的作用:我们知道，0^1=1,1^1=0,0^0=0。具体如下：    比如这道题，就用到了异或的性质。    解决：结合异或的性质，我们知道a^a必然为0，而0异或任何数都为这个数自身。所以扫描这个数组，我们可以用一个变量ans（初始化为0）分别异或这个数组的每一个数字，最终的ans就是那个单个的元素。  最终代码如下：    结果：    (5)每一次动态分配内存，基本都需要初始化为0，有没有方便快捷的初始化方法？  解决：首先我想到的就是用memset函数，如下：    后来发现，只有a数组的第一个元素为0，后面的元素都是随机的，于是又查阅了相关博客，找到一种适合的方法，就是：     1. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释） 2. A题 3. #include<iostream> 4. #include <cstring> 5. #include<cmath> 6. **using** **namespace** std; 8. **void** num\_to2(**int** val,**int** \*binary){//binary用来存储二进制 9. **int** cnt=1; 10. **while**(val){//binary数组从1开始存储，方便计算价值 11. binary[cnt++]=val%2; 12. val/=2; 13. } 14. } 16. **int** main(){ 17. **int** n;//长度 18. cin>>n; 20. **int** \*binary\_array=**new** **int**[n+1]();//存储二进制,并全部初始化为0 22. **int** \*a=**new** **int**[n+1]();//记录集合,并全部初始化为0 24. **for**(**int** i=1;i<=n;i++) 25. { 26. cin>>a[i]; 27. } 29. **int** ans=0;//记录结果,并且初始化为0的原因是0异或任何数都为0 30. **for**(**int** i=0;i<pow(2,n);i++) 31. { 32. **int** val=i;//临时记录，防止破坏i,造成死循环 34. **for**(**int** j=1;j<=n;j++) 35. { 36. binary\_array[j]=0;//初始化,不能用memset，因为是指针 37. } 39. num\_to2(val,binary\_array);//枚举二进制，挑选需要哪些元素 41. //计算价值 42. **int** temp\_array[n+1];//记录挑选出来了哪些元素，多开一个数组空间是为了方便计算机智 43. **int** cnt=1;//计数器 44. **for**(**int** j=1;j<=n;j++) 45. { 46. **if**(binary\_array[j])//如果选中 47. { 48. temp\_array[cnt++]=a[j]; 49. } 50. } 52. **int** temp\_res=0;//记录结果 53. **for**(**int** i=1;i<=cnt;i++) 54. { 55. temp\_res+=i\*temp\_array[i]; 56. } 58. ans=ans^temp\_res; 59. } 60. cout<<ans; 62. **return** 0; 63. }   (2)B题   1. #include <iostream> 2. **using** **namespace** std; 4. **int** range(**int** \*num,**int** n,**int** cnt,**bool** \*jud,**int** \*res) 5. {   //num是原始的数组，n是数组长度,cnt是排列到了第几位,jud是判断每一位用咩用过 6. **int** temp\_ans = 0; 7. **for** (**int** i = 1; i <= n; i++) 8. { 9. **if** (!jud[i]) 10. {//如果没用过 11. jud[i] = 1;//将其赋值为1 12. res[cnt] = num[i];//赋值 13. temp\_ans=temp\_ans|range(num,n,cnt+1,jud,res);//进入下一层进行全排列 14. jud[i] = 0;//复原现场 15. } 16. } 18. **if** (cnt == n) //递归终止条件 19. { 21. **for** (**int** i = 1; i <= n; i++) 22. { 23. temp\_ans += i ^ res[i]; 24. } 26. } 27. **return** temp\_ans; 28. }  31. //先全排列，然后记录ans|运算 32. **int** main() 33. { 34. **int** n;//几组数据 35. cin >> n; 36. **int** \*num=**new** **int**[n+1]();//读入原始数据并初始化 37. **bool** \*jud=**new** **bool**[n+1]();//记录某个数字用没用过 38. **int** \*res=**new** **int**[n+1]();//记录全排列对应的结果 40. **for** (**int** i = 1; i <= n; i++) 41. { 42. cin >> num[i]; 43. } 45. cout<<range(num,n,1,jud,res); 47. **return** 0; 48. } | | | |