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：202200400053 | 姓名： 王宇涵 | | 班级： 22级2班 |
| 实验题目：递归练习 | | | |
| 实验学时：2 | | 实验日期： 2023-9-11 | |
| 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：  Vscode | | | |
| 1. 实验内容      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）   第一题:  1. 用户首先输入一个整数 n，表示数组的大小。  2. 用户随后输入 n 个整数，这些整数存储在数组 arr 中。  3. 程序调用 `work\_subSet` 函数来生成数组 arr 的所有子集的价值，并存储在 `sub\_value` 数组中。`sup` 数组用于表示当前子集中的元素是否包含在子集中，其中 `sup[i]` 为 1 表示第 i 个元素包含在子集中，为 0 表示不包含。  4. `work\_subSet` 函数使用递归来生成子集。它从第一个元素开始，对每个元素有两种选择：包含在子集中（sup[i] = 1）或不包含在子集中（sup[i] = 0）。当递归到达数组的末尾（index == n-1）时，它计算当前子集的价值，并将其存储在 `sub\_value` 数组中。  5. `main` 函数调用 `work\_subSet` 函数来生成所有子集的价值。  6. 最后，`main` 函数使用异或操作来计算所有子集价值的异或结果，并将结果输出。  第二题:  1. 用户首先输入一个整数 n，表示数组的大小。  2. 用户随后输入 n 个整数，这些整数存储在数组 arr 中。  3. 程序定义了一个函数 `factorial` 用于计算阶乘，这个函数用来计算有多少种不同的排列方式，因为对于 n 个元素的排列，共有 n! 种排列方式。  4. 程序调用 `dfs` 函数来生成数组 arr 的所有排列的价值，并存储在 `value` 数组中。`sup` 数组用于暂时存储当前排列中的元素，`st` 数组用于记录元素是否被使用过。  5. `dfs` 函数使用深度优先搜索 (DFS) 来生成排列。它从第一个位置开始，尝试将未使用的元素加入排列中，然后递归处理下一个位置，直到排列中包含了所有的元素。在每个递归步骤中，它计算当前排列的价值，并将其存储在 `value` 数组中。  6. `main` 函数调用 `dfs` 函数来生成所有排列的价值。  7. 最后，`main` 函数使用按位或操作来计算所有排列价值的按位或结果，并将结果输出。   1. 测试结果（测试输入，测试输出）   第一题:  输入:2 1 2输出6  第二题:  输入:3 1 2 3 输出6   1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   注意:第一题不需要对递归返回的数组元素进行复位操作,因为不需要再次相同的0或1元素, 而第二题使用dfs深搜需要进行元素的复位,以便递归返回的时候可以继续搜索到该元素  st[arr[i]] = true;  sup[u] = arr[i];  dfs(u + 1, arr, st, value, sup);  // 注意此时需要复位  st[arr[i]] = false;   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）   第一题  #include <iostream>  #include <math.h>  using namespace std;  int n, value\_index;  const int N = 20;  int work\_subSet(int arr[], int sup[], int sub\_value[], int index)  {  for (int i = 0; i <= 1; i++)  {  // 表示将sup的位置依此赋为0/1  sup[index] = i;  // 如果递归已到头  if (index == n - 1)  {  int tmpsum = 0;  int tmp[n];  int sub\_index = 0;  // 将标号为1的元素放入tmp数组中  for (int j = 0; j < n; j++)  {  if (sup[j] == 1)  {  tmp[sub\_index++] = arr[j];  }  }  // 求出tmp数组的价值  for (int j = 0; j < sub\_index; j++)  {  tmpsum += (tmp[j] \* (j + 1));  }  // 将这个价值存入sub\_value中  sub\_value[value\_index++] = tmpsum;  }  // 如果递归还没有到头  else  {  work\_subSet(arr, sup, sub\_value, index + 1);  }  // 此时不需要重新赋值,因为每次都会赋值为0-1  }  }  int main()  {  cin >> n;  int arr[n], sub\_value[int(pow(2, n))], sup[n];  for (int i = 0; i < n; i++)  cin >> arr[i];  // 求出各个子集和子集价值  work\_subSet(arr, sup, sub\_value, 0);  // 处理最终结果  int ans = sub\_value[0];  for (int i = 1; i < int(pow(2, n)); i++)  {  ans ^= sub\_value[i];  }  cout << ans;  return 0;  }  第二题:  #include <iostream>  using namespace std;  int n;  int v\_index = 0;  const int N = 100010;  // 求n的阶乘  int factorial(int n)  {  if (n == 1)  return 1;  return (n \* factorial(n - 1));  }  void dfs(int u, int arr[], bool st[], int value[], int sup[])  {  if (u == n)  {  // 求出其价值  int t\_sum = 0;  for (int i = 0; i < u; i++)  {  t\_sum += (sup[i]) ^ (i + 1);  }  value[v\_index++] = t\_sum;  }  else  {  for (int i = 0; i < n; i++)  {  // 遍历每一个数组中的元素  if (!st[arr[i]])  {  st[arr[i]] = true;  sup[u] = arr[i];  dfs(u + 1, arr, st, value, sup);  // 注意此时需要进行元素的复位,以便递归返回的时候可以继续搜索到该元素  st[arr[i]] = false;  }  }  }  }  int main()  {  cin >> n;  int arr[n], sup[n];  bool st[N];  int value[factorial(n)];  for (int i = 0; i < n; i++)  cin >> arr[i];  // 求出各个排列价值  dfs(0, arr, st, value, sup);  // 求出最终结果  int ans = value[0];  for (int i = 1; i < v\_index; i++)  {  ans |= (value[i]);  }  cout << ans;  return 0;  } | | | |
|  | | | |