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：

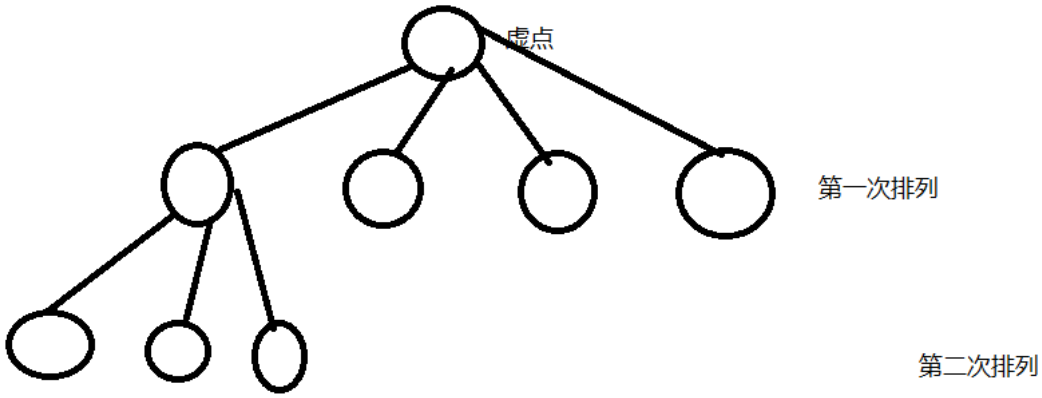
一个整数，代表所有排列价值的或。

2. 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）

(1) 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。

3. 测试结果（测试输入，测试输出）

(1) A 题样例

```
2
1 2
6
进程已结束，退出代码为 0
```

(2) A 题自造数据

```
3
5 7 1
8
```

(3) B 题样例

```
3
1 2 3
6
进程已结束，退出代码为 0
```

(4) B 题自造数据（边界条件）

```
5
0 0 0 0 0
15
```

经过手工计算，该结果正确

4. 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：

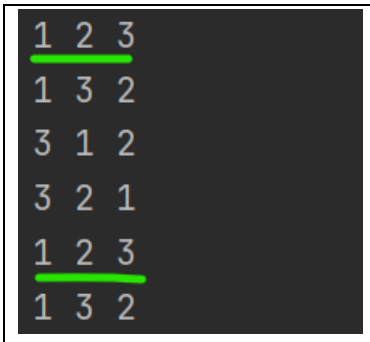
```
void num_to2(int val,int *binary){//binary用来存储二进制
    int cnt=1;
    while(val!=0){
        binary[cnt++]=val%2;
        val/=2;
    }
}
```

(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）

```
for (int i = k; i < m; i++)
{
    swap( &: a[k],  &: a[i]);
    permutation(a,  k: k + 1, m);
    swap( &: a[k],  &: a[i]);
}
```

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：

左侧	右侧	异或（^）
0（假）	0（假）	0（假）
0（假）	1（真）	1（真）
1（真）	0（假）	1（真）
1（真）	1（真）	0（假）

比如这道题，就用到了异或的性质。

给定一个**非空**整数数组，除了某个元素只出现一次以外，其余每个元素均出现两次。找出那个只出现了一次的元素。

说明：

你的算法应该具有线性时间复杂度。 你可以不使用额外空间来实现吗？

示例 1:

输入： [2,2,1]
输出： 1

示例 2:

输入： [4,1,2,1,2]
输出： 4

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：

```
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int ans = 0;
        for(int i = 0; i < nums.size(); i++){
            ans ^= nums[i];
        }
        return ans;
    }
};
```

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：

```
int* a = new int[n] ;

memset(a,0,sizeof(a));
```

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：

```
int* a = new int[n] ();
```

5. 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）

(1) A 题

```
#include<iostream>
#include <cstring>
#include<cmath>
using namespace std;

void num_to2(int val,int *binary){//binary 用来存储二进制
    int cnt=1;
    while(val){//binary 数组从 1 开始存储，方便计算价值
        binary[cnt++]=val%2;
        val/=2;
    }
}

int main(){
    int n;//长度
    cin>>n;

    int *binary_array=new int[n+1]();//存储二进制,并全部初始化为 0

    int *a=new int[n+1]();//记录集合,并全部初始化为 0
```

```
21.
22.     for(int i=1;i<=n;i++)
23.     {
24.         cin>>a[i];
25.     }
26.
27.     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
28.     for(int i=0;i<pow(2,n);i++)
29.     {
30.         int val=i;//临时记录，防止破坏 i,造成死循环
31.
32.         for(int j=1;j<=n;j++)
33.         {
34.             binary_array[j]=0;//初始化,不能用 memset，因为是指针
35.         }
36.
37.         num_to2(val,binary_array);//枚举二进制，挑选需要哪些元素
38.
39.         //计算价值
40.         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
41.         int cnt=1;//计数器
42.         for(int j=1;j<=n;j++)
43.         {
44.             if(binary_array[j])//如果选中
45.             {
46.                 temp_array[cnt++]=a[j];
47.             }
48.         }
49.
50.         int temp_res=0;//记录结果
51.         for(int i=1;i<=cnt;i++)
52.         {
53.             temp_res+=i*temp_array[i];
54.         }
55.
56.         ans=ans^temp_res;
57.     }
58.     cout<<ans;
59.
60.     return 0;
61. }
```

(2)B 题

```
1.  #include <iostream>
```

```
using namespace std;

int range(int *num,int n,int cnt,bool *jud,int *res)
{   //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
    int temp_ans = 0;
    for (int i = 1; i <= n; i++)
    {
        if (!jud[i])
        {//如果没用过
            jud[i] = 1;//将其赋值为 1
            res[cnt] = num[i];//赋值
            temp_ans=temp_ans|range(num,n,cnt+1,jud,res);//进入下一层进行全排列
            jud[i] = 0;//复原现场
        }
    }

    if (cnt == n) //递归终止条件
    {

        for (int i = 1; i <= n; i++)
        {
            temp_ans += i ^ res[i];
        }

    }
    return temp_ans;
}


//先全排列，然后记录 ans|运算
int main()
{
    int n;//几组数据
    cin >> n;
    int *num=new int[n+1]();//读入原始数据并初始化
    bool *jud=new bool[n+1]();//记录某个数字用没用过
    int *res=new int[n+1]();//记录全排列对应的结果

    for (int i = 1; i <= n; i++)
    {
        cin >> num[i];
    }

    cout<<range(num,n,1,jud,res);

    return 0;
```

```
48. }
```