408代码汇总

1. 2009年

   已知一个带有表头结点的单链表，结点结构为：

   假设该链表只给出了头指针 list。在不改变链表的前□下，请设计一个尽可能高效的算法，查找链表中倒数第 k 个位置上的结点（k 为正整数）。若查找成功，算法输出该结点的 data 域的值，并返回 1；否则，只返回 0。

   算法思想：使用p，q两个指针，p指针先移动扫描k个指针，之后q再与p同步移动，当p指向最后一个节点时，q正好指向倒数第k个节点

```
int SearchRearK(LNode *L, int k)
{
    int count=0;//用来计数
    LNode *q=L->link;
    while(p!=NULL)
    {
        if(count<k)
            count++;
        else
            q=q->link;//当count等于开始，q和p同步向后移动
        p=p->link;
    }
    if(count<k)
        return 0;//如果链表节点个数小于k
    else
    {
        printf("%d",q->data);
        return 1;
    }
}
```

---

2. 2010年

   设将 n（n>1）个整数存放到一维数组 R 中。试设计一个在时间和空间两方面都尽可能高效的算法。将 R 中保存的序列循环左移 p（0<p<n）个位置，即将 R 中的数据由（X0, X1, , Xn-1）变换为（Xp, Xp+1, , Xn-1, X0, X1, , Xp-1）。

   算法思想：先将R中前p个元素逆置，再将剩下的元素逆置，最后将R中所有的元素再整体做一次逆置即可

```
void Reverse(int R[],int l,int r)
{
   int i,j;
   int temp;
   for(i=l,j=r;i<j;++i,--j)
   {
       temp=R[i];
       R[i]=R[j];
```

```
        R[j]=temp
    }
}
void RCR(int R[],int n,int p)
{
    if(p<=0||p>=n)
        cout<<"ERROR"<<endl;
    else
    {
        Reverse(R,0,p-1);
        Reverse(R,p,n-1);
        Reverse(R,0,n-1);
    }
}
```

3. 2011年

一个长度为L（L≥1）的升序序列S，处在第¬L/2°个位置的数称为S 的中位数。例如，若序列S1=（11，13，15，17，19），则S1 的中位数是15，两个序列的中位数是含它们所有元素的升序序列的中位数。例如，若S2=（2，4，6，8，20），则S1 和S2 的中位数是11。现在有两个等长升序序列A 和B，试设计一个在时间和空间两方面都尽可能高效的算法，找出两个序列A 和B 的中位数

算法思想：（较容易想到）使用二路归并思想，顺序比较S1与S2，当S1小于S2中的值，S1的下标向后移动一位，同理S2小时，S2下标向后移动一位，当比较次数达到n（序列长度）时，返回对应的值，就是中位数了

```
int Search_M(int S1[],int S2[],int n)
{
    int i=j=k=0;
    while(i<n&&j<n)
    {
        k++;
        if(S1[i]<S2[j])
        {
            i++;
            if(k==n)
                return S1[i-1];
        }
        else
        {
            j++;
            if(k==n)
                return S2[j-1];
        }
    }
}
```

4. 2012年

假定采用带头结点的单链表保存单词，当两个单词有相同的后缀时，则可共享相同的后缀存储空间，设 str1 和 str2 分别指向两个单词所在单链表的头结点，链表结点结构为 ，请设计一个时间上尽可能高效 的算法，找出由 str1 和 str2 所指向两个链表共同后缀的起始位置。

算法思想：先将较长的链表先向后移动k个位置，让两个链表之后的扫描长度是一样长的，之后两个链表的扫描指针同步向后移动，返回第一个公共节点，即是两个链表公共后缀的起始位置

```
void FindSameRear(LNode *L1,LNode *L2)
{
    int len1=Length(L1);
    int len2=Length(L2);
    LNode *p,*q;
    for(p=L1;Len1>len2;len1--)
        p=p->next;
    for(q=L2;len2>len1;len2--)
        q=q->next;
    while(p->next!=NULL&&p->next!=q->next)//判断是否访问到同一节点
    {
        p=p->next;
        q=q->next;
    }
    return p-next;//返回共同后缀的起点
}
```

---

5. 2013年

已知一个整数序列 A=（a0, a1, ..., an+1），其中0≤ai < n（0≤i < n）。若存在ap1=ap2=...= apm=x 且 m> n/2 (0≤pk < n，1≤k≤m)，则称 x 为 A 的主元素。例如 A=（0, 5, 5, 3, 5, 7, 5, 5)，则 5 为主元素；又如 A=（0, 5, 5, 3, 5, 1, 5, 7），则 A 中没有主元素假设 A 中的 n 个元素保存在一个一维数组中，请设计一个尽可能高效的算法，找出 A 的主元素。若存在主元素，则输出该元素；否则输出-1。

算法思想：（较容易理解）计数排序思想，用一个数组记录，每个值出现次数，用的是使用牺牲空间换取时间的做法

```
int Majority(int A[],int n)
{
    int k,max=0;
    int *p;
    p=(int*)malloc(sizeof(int)*n);//为数组申请内存空间
    for(k=0;k<n;k++)
        p[k]=0;//将数组初始化都为0
    for(k=0;k<n;k++)
    {
        p[A[k]]++;
        if(p[A[k]]>p[max])
            max=a[k];//记录出现次数最多的元素
    }
    if(p[max]>n/2)
        return max;//如果个数大于n/2，表示找到了
    else
        return -1;
}
```

---

6. 2014年

二叉树的带权路径长度（WPL）是二叉树中所有叶结点的带权路径长度之和。给定一棵二叉树 T，采用 二叉链表存储，二叉树的带权路径长度（WPL）是二叉树中所有叶结点的带权路径长度之和。给定一棵二叉树 T，采用二叉链表存储

算法思想：1.先序遍历：用一个 static 变量记录 wpl，把每个结点的深度作为递归函数的一个参数传递，若该结点是叶子结点，那么变量 wpl 加上该结点的深度与权值之积；若该结点非叶子结点，那么若左子树不为空，对左子树调用递归算法，若右子树不为空，对右子树调用递归算法，深度参数均为本结点的深度参数加 1；最后返回计算出的 wpl 即可。

2.层次遍历使用队列进行层次遍历，并记录当前的层数，当遍历到叶子结点时，累计 wpl；当遍历到非叶子结点时对该结点的把该结点的子树加入队列；当某结点为该层的最后一个结点时，层数自增 1；队列空时遍历结束，返回 wpl。

```
//先序遍历
int WPL(BTNode *root){
    return PreOrder(root, 0);
}
int PreOrder(BTNode *root, int deep)
{
    static int wpl = 0; //定义一个 static 变量存储 wpl
    if(root->lchild == NULL && root->rchild == NULL) //若为叶子结点，累积 wpl
        wpl += deep*root->weight;
    if(root->lchild != NULL) //若左子树不空，对左子树递归遍历
        PreOrder(root->lchild, deep+1);
    if(root->rchild != NULL) //若右子树不空，对右子树递归遍历
        PreOrder(root->rchild, deep+1);
    return wpl;
}

//层次遍历
int wpl_LevelOrder(BTNode *root)
{
    BTNode *q[MaxSize]; //声明队列
    front = rear = 0; //头指针指向队头元素，尾指针指向队尾的后一个元素
    int wpl = 0, deep = 0; //初始化 wpl 和深度
    BTNode *lastNode; //lastNode 用来记录当前层的最后一个结点
    BTNode *newlastNode; //newlastNode 用来记录下一层的最后一个结点
    lastNode = root; //lastNode 初始化为根节点
    newlastNode = NULL; //newlastNode 初始化为空
    rear=(rear+1)%Maxsize;
    q[rear] = root; //根节点入队
    BTNode *p;
    while(front != rear)//层次遍历，若队列不空则循环
    {
        p = q[end1++]; //拿出队列中的头一个元素
        if(p->lchild == NULL & p->rchild == NULL)
            wpl += deep*p->weight; //若为叶子结点，统计 wpl
        if(p->lchild != NULL)   //若非叶子结点把左结点入队
        {
            rear=(rear+1)%Maxsize;
            q[rear] = p->lchild;
            newlastNode = p->lchild;
        } //并设下一层的最后一个结点为该结点的左结点
```

```
        if(p->rchild != NULL)//处理叶节点
        {
             rear=(rear+1)%Maxsize;
            q[rear] = p->rchild;
            newlastNode = p->rchild;
        }
        if(p == lastNode)//若该结点为本层最后一个结点，更新 lastNode
        {
            lastNode = newlastNode;
            deep += 1; //层数加 1
        }
    }
    return wpl; //返回 wpl
}
```

7. 2015年

用单链表保存 m 个整数，结点的结构为：data，且|data|≤n（n 为正整数）。现要求设计一个时 间复杂度尽可能高效的算法，对于链表中 data 的绝对值相等的结点，仅保留第一次出现的结点而删除其余绝对值相等的结点。

算法思想：类似前面的，也是计数排序思想，借助一个数组，具体思想参考前面

```
void DeleteABS(LNode *L,int n)
{
    LNode *p=L->link,*q;//p指向L下一节点，q用来指向被删节点
    int *a,m;
    a=(int*)malloc(sizeof(int)*n);//为数组申请内存空间
    for(int k=0;k<n;k++)
        a[k]=0;//将数组初始化都为0
    while(p!=NULL)
    {
        m=p->data>0?p->data:-p->data;
        if(a[m]==0)
        {
            a[m]=1;//标记，前面以及有这个数了，
            p=p->link;
        }
        else
        {
            q=p;
            p=p->link;
            free(q);
        }
    }
    free(a);//释放掉a的空间
}
```

8. 2016年

已知由 n（n≥2）个正整数构成的集合 A={ak|0≤k < n}，将其划分为两个不相交的子集 A1和 A2，元素个数分别是 n1和 n2，A1和 A2中元素之和分别为 S1和 S2。设计一个尽可能高效的划分算法，满足|n1-n2|最小且|S1-S2|最大。(全抄答案的，没有自己写)

算法思想：将最小的□n/2)□个元素放在 A1中，其余的元素放在 A2中，分组结果即可满足题目要求。仿照快速排序的思想，基于枢轴将 n 个整数划分为两个子集。根据划分后枢轴所处的位置 i 分别处理： ①若 i=□n/2)□，则分组完成，算法结束； ②若 i < □n/2)□，则枢轴及之前的所有元素均属于 A1，继续对 i 之后的元素进行划分； ③若 i > □n/2)□，则枢轴及之后的所有元素均属于 A2，继续对 i 之前的元素进行划分；

```
int setPartition(int a[], int n)
{
    int pivotkey, low=0, low0=0, high=n-1, high0=n-1, flag=1, k=n/2, i;
    int s1=0, s2=0;
    while(flag)
    {
        piovtkey=a[low]; //选择枢轴
        while(low<high)//基于枢轴对数据进行划分
        {
            while(low<high && a[high]>=pivotkey) -high;
            if(low!=high) a[low]=a[high];
            while(low<high && a[low]<=pivotkey) ++low;
            if(low!=high) a[high]=a[low];
        } //end of while(low<high)
        a[low]=pivotkey;
        if(low==k-1) //如果枢轴是第 n/2 小元素，划分成功
            flag=0;
        else//是否继续划分
        {
            if(low<k-1)
            {
                low0=++low;
                high=high0;
            }
            else
            {
                high0=--high;
                low=low0;
            }
        }
    }
    for(i=0;i<k;i++) s1+=a[i];
    for(i=k;i<n;i++) s2+=a[i];
    return s2-s1;
}
```

9. 2017年

请设计一个算法，将给定的表达式树（二叉树）转换成等价的中缀表达式（通过括号反映次序），并输出

算法思想：基于二叉树的中缀遍历，添加适当括号，显然，表达式的最外层（对于根节点）及操作数（对应叶节点）不需要添加括号（这句是答案说的，其实不太懂）

```
void B2E(BTNode *root)
{
    B2E(root,1);
}
void B2E(BTNode *root,int deep)
{
    if(root==NULL)
        printf("NULL");
    else if(root->left==NULL&&root->right==NULL)
        printf("%s",root->data);//输出操作数
    else
    {
        if(deep>1) printf("(");
        B2E(root->left,deep+1);
        printf("%s",root->data);//输出操作符
        B2E(root->right,deep+1);
        if(deep>1) printf(")");
    }
}
```

---

10. 2018年

给定一个含n（n>=1）个整数的数组，请设计一个在时间上尽可能高效的算法，找出数组中未出现的最小正整数。例如，数组{-5，3，2，3}中未出现的最小正整数是1，数组{1，2，3}中未出现的最小正整数是4。

算法思想：又是计数排序思想，就不再论述了，用一个数组计数，也就是用空间换时间

```
int FindNoMin(int A[],int n)
{
    int *B,i;
    B=(int*)malloc(sizeof(int)*n);//为数组申请内存空间
    for(i=0;i<n;i++)
        B[i]=0;//将数组初始化都为0
    for(i=0;i<n;i++)
        if(A[i]>0&&A[i]<=n)//若A属于0~n之间，标记B
            B[A[i]-1]=1;
    for(i=0;i<n;i++)//扫描计数数组，找到目标值
        if(B[i]==0)
            break;
    return i+1;
}
```