真题代码

数组负值放在非负前面 -- 线性复杂度
不带头节点的链表（首节点指针为 f），都是整型数据，递归求平均值(2005 期末原题)
以数组Q[m]，存放循环队列中的元素，同时以rear和length分别只是队尾位置和队中所含元素个数，实现该队列的类声明，队空队满条件，插入删除

2017年

无

2018年

无

2013年备用卷

二叉搜索树实现优先队列: Insert()+Delete()

数据结构期末试题

期末试卷（一）

返回二叉树先序最后一个结点指针，非递归

```
BinaryTreeNode<T> * getFinalNode(BinaryTreeNode<T>*root){
    if(root==NULL)return NULL;
    BinaryTreeNode<T>* p = root;
    while(p!=NULL){
        if(p->rightChild!=NULL){
            p = p->rightChild;
        }else if(p->leftChild != NULL){
            p = p->leftChild;
        }
    }
    return p;
}
```

根据邻接表求逆邻接表 O(n+e)

```
void getReverse(BinaryTreeNode*V[]){
    for(int i = 0;i < numVertics;i++){//遍历所有结点
        Edge *p = NodeTable[i].adj;
        while(p!=NULL){
            Edge *e = new Edge(i,p->weight);
            e->link = NodeTable[p->dest].reverseAdj->link;//头插法
            NodeTable[p->dest].reverseAdj = e;
        }
    }
}
//i --- > [p,weight,dest] --- > []

//p->dest ---- > [e,weight,dest(i)] ---> []
```

2005 数据结构期末

不带头节点的链表（首节点指针为 f），都是整型数据，递归求平均值

```
// 方法一：  殷人昆书上答案
float Avg(LinkNode * L, int &n){// n返回节点数目
    if(L->link == NULL) {
        n = 1;
        return (float)(L->data);//链表只有一个节点时，其值就是平均值
    }else{
        float Sum = Avg(L->link, n) * n; 先递归求后续节点的平均值 记为Sum
        n++;
        return (L->data + Sum) / n; // 加上本节点再求平均值
    }
}
---------------------------------- 分割线 ----------------------------------

// 我写的
// 需要设置全局变量
int n;
float getAvg(Node f){
    if(f == NULL)return 0;
    if(f.link == NULL){
        n ++ ;
        return f.data;
    }
    float avg = (f.data + n* getAvg(f.link))/ (n+1);
    n++;
    return avg;
}
---------------------------------- 分割线 ----------------------------------

// 下面两个解法来源于网络

// 函数需要传入上一轮的结果，和当前节点数目
double getAverage_List(LinkList L, double sum, int i){
    if (L->next != NULL){
        sum = sum + L->data;
        return getAverage_List(L->next, sum, i+1);
    }else{
        double ave = (sum + L->data)/(i+1);
        return ave;
    }
}
---------------------------------- 分割线 ----------------------------------

// 先求出所有节点的数目
```

```
int count_LNode(LinkList L){     //统计节点个数，传入链表首地址
    if (L == NULL)return 0;     //递归结束条件，递归到尾节点的下一个节点，直接返回
    else return count_LNode(L->next) + 1;     //否则继续指向下一个节点，表长加1
}
int count_LNode(LinkList L){     //统计节点个数，传入链表首地址
    if (L == NULL)return 0;     //递归结束条件，递归到尾节点的下一个节点，直接返回
    else return count_LNode(L->next) + 1;     //否则继续指向下一个节点，表长加1
}
n = count_LNode(L)
double getAverage_List(LinkList L,int n){//递归计算平均值
    if (!L->next)return L->data;
    else{
        double ave = getAverage_List(L->next, n - 1);
        return (ave*(n - 1) + L->data) / (n+1);
    }
}
```

————————————————
版权声明：本文为CSDN博主「yz764127031」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。
原文链接：https://blog.csdn.net/yz764127031/article/details/81811461

还是殷人昆的方法比较巧妙

递归：从大到小顺序输出二叉搜索树中所有不小于k的关键字

```
void BinaryTree<T>::FindLargeThanK(BinaryTreeNode * root,int k){
    if(root == NULL)return;
    FindLargeThanK(root->rightChild,k);
    if(root->data >= k)print(root->data);
    FindLargeThanK(root->leftChild,k);
}
```

2006 数据结构期末

二叉搜索树中每个节点 Lsize(左子树节点个数加1),left,data right，求第k小的关键码结点

```
BianryTreeNode<T>* findK(BinaryTreeNode<T> *root,int k){
    if(root==NULL||k<=0){return NULL}
    if(k == root->Lsize){
        return root;
    }else if(k < root->Lsize){
        return findK(root->left,k);
    }else{
        return findK(root->right,k-Lsize);
    }
}
```

建立最小堆的SiftUP

```
private static void siftUp(int a[],int n){
    int child = n;
    int temp = a[child];
    int father = (child-1)/2;
    while(a[father]>temp && child!=0){
        a[child] = a[father];
        child = father;
        father = (child - 1)/2;
    }
    a[child] = temp;
}
```

2007数据结构

循环链表解决Josephus 问题？？？？？？

```
ListNode Josephus(int n,int m){
    int w = m;
    ListNode * head,*p;
    for (int i = 0;i<=n-1;i++){
        for( j = 0;j<=w-1;j++){
            rear = rear->link;
        }
        if(i==1){
            head = rear.link;
            p = head;
        }else{
            p.link = rear.link;
            p = rear.link;
        }
        rear.link = p.link;
    }
    p.link = rear;
    rear.link = NULL;
    return head;
}
```

判断是不是二叉搜索树

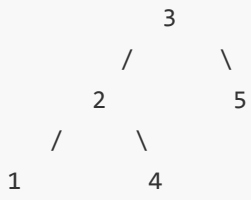
https://blog.csdn.net/fly_yr/article/details/52172839

1. 错误解法：对每一个节点，检测其值是否大于左子树节点，是否小于右子树节点。

```
bool isBST(TreeNode* root)
{
    if (root == NULL)
```

```
        return true;
    if (root->left != NULL && root->left->data> root->data)
        return false;
    if (root->right != NULL && root->right->data < root->data)
        return false;
    if (!isBST(root->left) || !isBST(root->right))
        return false;
    return true;
}
```

反例



2. 把中序遍历的结果存起来，然后判断是否递增
3. 但是上述方法需要额外线程空间保存遍历结果，在此可以省去该空间开销，只需一个变量保存访问当前节点时上一节点的值即可
4. 另一种方法，对于每一个子树：left<=current<right；

```
/*方法一，将中序遍历结果保存到数组 T(n)=O(n) S(n)=O(n)*/
    void inOrder(TreeNode *root,vector<int> &v)
    {
        if (root == NULL)
            return;
        inOrder(root->left, v);
        v.push_back(root->val);
        inOrder(root->right, v);
    }

    bool checkBST1(TreeNode* root)
    {
        vector<int> ret;
        inOrder(root, ret);
        for (auto i = ret.begin()+1; i != ret.end(); ++i)
        {
            if (*i < *(i - 1))
                return false;
        }
        return true;
    }
/*方法二、省掉线性空间，保存遍历的最后一个节点*/
    int lastVal = INT_MIN;
    bool checkBST2(TreeNode* root) {
        if (!root)
            return true;
```

```
        /*递归检查左子树*/
        if (!checkBST2(root->left))
            return false;

        /*比较当前节点，并更新已遍历节点最后的值*/
        if (root->val <= lastVal)
            return false;
        lastVal = root->val;

        /*递归检查右子树*/
        if (!checkBST2(root->right))
            return false;
        return true;
    }
/*方法三，最大最小值法*/
    bool checkBST3(TreeNode* root) {
        // write code here
        if (!root)
            return true;
        return checkBST3(root, INT_MAX, INT_MIN);
    }
    bool checkBST3(TreeNode *root, int maxVal, int minVal)
    {
        if (!root)
            return true;
        if (root->val < minVal || root->val >= maxVal)
            return false;
        if (!checkBST3(root->left, root->val, minVal) || !checkBST3(root->right,
maxVal, root->val))
            return false;
        return true;
```

————————————————
版权声明：本文为CSDN博主「逆風的薔薇」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。
原文链接：https://blog.csdn.net/fly_yr/article/details/52172839

```
//


BinaryNode min(BinaryNode n){
    if(n == null)return null;
    while(n.left != null)n = n.left;
    return n;
}
BinaryNode max(BinaryNode n){
    if(n==null)return null;
    while(n.right != null)n=n.right;
```

```
    return n;
}
int isBST(BinaryNode t){
    if(t == null)return 1;
    else if(t.left != null && max(t.left).data > t.data)return 0;
    else if(t.right != null && min(t.right).data < t.data)return 0;
    else return isBST(t.left)&&isBST(t.right);
}
```

个人觉得第二种最好理解。

数据结构期末A1

已知先序遍历和中序遍历，建立二叉树

```
BinaryTreeNode<T> * createTree(int preOrder[],int inOrder[],int preLeft,int
preRight,int inLeft,int inRight){
    int llen;
    int rlen;
    BinaryTreeNode<T> * t = new BinaryTree<int>(preOrder[preLeft]);
    for(int i = inLeft,inOrder[i]!=preOrder[preLeft];i++);
    llen = i - inLeft;
    rlen = inRight - i;
    if(llen!=0){
        t.leftChild = createTree(perOrder, inOrder,
preLeft+1,preLeft+llen,inLeft,inLeft+llen-1);
    }else{
        t.leftChild = NULL;
    }
    if(rlen!=0){
        t.rightChild = createTree(preOrder,inOrder,preRight-llen+1,preRight,inRight-
rlen+1,inRight);
    }else{
        t.rightChild = NULL;
    }
    return t;
}
```

循环链表的逆转

```
template<class Type>
void ClistNode<Type>::Inverse(){
    CListNode<Type> *p,*q,*r;
    if(first->link = first)return;
    p = first->link;
    q = p->link;
    p->link = first;//逆转第1，2个结点
    while(p!=first){//p不是最后一个
```

```
        r = p->link;
        q->link = p;
        p = q;
        q = r;
    }
    //p停留在最后一个节点
    first->link = p;
}
```

数据结构期末A2

2009数据结构期末

调整最小堆 siftUp（2006期末）

2003 年真题

给出二叉树类定义，并求二叉树层数

```
//二叉树定义
template<class T>
class BinaryTreeNode{
    friend class BinaryTree //设置友元，如果不设置，BinaryTree对象只能用getter/setter来访问孩子，设置了就可以直接访问了
private:
    T data;
    BinaryTree<T>* leftChild;
    BinaryTree<T>* rightChild;
public:
    BinaryTreeNode(const T & item,BinaryTreeNode<T>* lptr = NULL,BinaryTreeNode<T>*rptr = NULL):data(item),leftchild(lptr),rightchild(rptr){}//构造函数
    BinaryTreeNode<T>* getLeft(void) const{return leftChild}
    BinaryTreeNode<T>* getRight(void) const{return rightChild}
    void setLeft(BinaryTreeNode<T>*L){leftChild = L}
    void setRight(BinaryTreeNode<T>*R){rightChild = R}
}
template<class T>
class BinaryTree{
private:
    BinaryTreeNode<T>*root;
public:
    BinaryTree(){root = 0;}
    ~BinaryTree(){}
    int getDepth(BinaryTreeNode<T> *root);
```

```
}
// 递归求层数
int BinaryTree<T>::getDepth(BinaryTreeNode<T> *root){
    if(root==NULL)return 0;
    int leftDepth = root->getLeft().getDepth();
    int rightDepth = root->getRight().getDepth();
    return 1 + (leftDepth > rightDepth)?leftDepth:rightDepth;
}
//非递归求层数
//按照层序遍历的过程，每遍历一层，层数+1
int BinaryTree<T>::getDepth(BinaryTreeNode<T>*root){
    int depth = 1;
    Queue<T> queue;
    BinaryTreeNode *p = root;
    queue.push(p);
    int count = queue.size();
    while(!queue.isEmpey()){
        if(count == 0){
            count = queue.size();
            depth++;
        }else{
            count--;
            BinaryTreeNode * node = queue.front();
            queue.pop();
            queue.push(node->getLeft());
            queue.push(node->getRight());
        }
    }
}
```

直接插入排序（原地）

```
void insertSort(int a[],int n){
    for(int i = 1;i<n;i++){
        int j = i-1;
        int temp = a[i];
        while(a[j] > temp){
            a[j+1] = a[j];
        }
        a[j+1] = temp;
    }
}
```

2004 真题

二叉树类定义 & 交换左右孩子

```
void BinaryTree<T>::reflect(BinaryTreeNode<T> * root){
    if(root == NULL)return;
    reflect(root->leftChild);
    reflect(root->rightChild);
    BinaryTreeNode<T> temp = root->leftChild; // temp = root->getLeft();
    root->leftChild = root->rightChild; // root->setLeft(root->getRight());
    root->rightChild = temp;//root->setRight(temp);
}
```

Q[m]存放循环队列：rear + length, 写出类定义和队空队满条件&插入& 删除

```
template <class T>
class Queue{
private:
    T * array;
    int rear;
    int length;
    int maxSize;
public:
    Queue(int capcity = 10):maxSize(capcity){}
    bool IsEmpty()const;
    bool IsFull()const;
    bool Add(T &x);
    bool Delete();
}
bool Queue<T>::IsEmpty(){
    return length == 0;
}
bool Queue<T>::IsFull(){
    return length == maxSize;
}
bool Queue<T>::Add(T &x){
    if(IsFull()){return false}
    rear = (rear+1)%maxSize;
    array[rear] = x;
    length += 1;
    return ture;
}
bool Queue<T>::Delete(){
    if(IsEmpty()){return false;}
    T res = array[(rear-length+1+maxSize)%maxSize];
    length -= 1;
    return res;
}
```

⚠️：ppt上的答案

```
#include<iostream.h>
```

```
#include<assert.h>
template<class Type>
class Queue{
private:
    int rear;
    int length;
    int m;
    Type *elements;
public:
    Queue(int m = 10);
    ~Queue(){delete[]elements;}
    void EnQueue(const Type &x)
    Type DeQueue()
    int isFull(){return length == m;}
    int isEmpty(){return length ==0;}
}
template<class Type>
void Queue<Type>::Dequeue(const Type&x){
    assert(!isFull());
    rear = (rear+1)%m;
    element[rear] = x;
    length++;
}
Type Queue<Type>::Dequeue(){
    assert(!isEmpty());
    length--;
    return element[(rear-length+m+1)%m]
}
```

2005年真题

递归判断是否是回文串

```
bool isPalindrome(char a[],int i, int n){
    if(i == n/2)return true;
    return a[i]==a[n-i-1] && isPalindrome(a,i+1,n);
}
```

子女-兄弟-度 表示法，写出类定义+求各节点度

```
template <class T>
class BinaryTreeNode{
    friend class BinaryTree;//为了让BinaryTree可以访问节点的孩子节点
private:
    T data;
    BinaryTreeNode<T> * firstChild;
```

```
    BinaryTreeNode<T> * nextsibling;
    int degree;
public:
    BinaryTreeNode(const T & item,BinaryTreeNode<T> lptr,BinaryTreeNode<T>
rptr):data(item),firstChild(lptr),rightChild(rptr){}
    void setDegree(int d){degree = d}
    int getDegree(){return degree}
}
template<class T>
class BinaryTree{
private:
    BinaryTreeNode<T> * root;
public:
    BinaryTree(){root = 0;}
    void getDegrees(BinaryTreeNode<T> * root);
}
void BinaryTree<T>::getDegrees(BinaryTreeNode<T> * root){
    if(root == NULL)return;
    root->degree = 0;//root->setDegree(0);因为设置了友元，所以可以不用setter
    if(root->firstChild!=NULL){
        root->degree ++;
        BinaryTreeNode<T> p = root->firstChild;
        while(!p->nextsibling){
            root->degree++;
            p = p->nextsibling;
        }
    }
    getDegree(root->firstChild);
    getDegree(root->nextsibling);
}
```

2006年真题

不带头结点的单链表，就地逆转

```
void reverse(LinkedNode * f){
    LinkedNode pre = f;
    LinkedNode cur = pre->next;
    LinkedNode next;
    pre->next = NULL;
    while(cur!=NULL){
        next = cur->next;
        cur->next = pre;
        pre = cur;
        cur = next;
    }
    f = cur;
}
```

孩子-兄弟表示法 -- 类定义+递归计算高度

```
int BinaryTree<T>::getHeight(BinaryTreeNode<T> *root){
    if(root==NULL)return 0;
    if(root->firstChild == NULL)return 1;
    BinaryTreeNode<T> p = root->firstChild;
    int height = 0;
    int maxHeight = height;
    while(p!=NULL){
        height = getHeight(p);
        if(height>maxHeight)maxHeight = height;
        p = p->nextsibling;
    }
    return 1+maxHeight;
}
```

2007年

判断是不是二叉搜索树（2007期末）

2008年

递归：从大到小顺序输出二叉搜索树中所有不小于k的关键字(2005期末原题)

求最大子序列和

「特别说明」默认如果所有整数均为负数，则最大子序列之和为0

1. 递归

   最大子序和，要么出现在左半部分（递归左半部分），要么出现在右半部分（递归右半部分），要么就左右都有（从中间往左右延伸）

```
// O(NlogN)
#include<math>
int FindGreatestSumOfSubArray(int a[], int low ,int height){
    if(low = height)return a[low];
    int mid = (low + height)/2;
    int leftSum = FindGreatestSumOfSubArray(a,low,mid);//递归左半部分
    int rightSum = FindGreatestSumOfSubArray(a,mid+1,height);//递归右半部分
    //计算中间部分
    int left = 0;//因为当最大子序列和为负时，答案也是0，所以最小就是0/
    int right = 0;//如果题目中没有「特别说明」，则要初始化为 MINX_VALUE
    int leftTemp = 0;
    int rightTemp = 0;
    for(int i = mid;i>=low;i--){
        leftTemp+=a[i];
        if(leftTemp > left){
            left = leftTemp;
        }
    }
    for(int j = mid+1;j<=height;j++){
        rightTemp += a[j];
        if(rightTemp>right){
            right = rightTemp;
        }
    }
    return math.Max(math.Max(leftSum,rightSum),left+right)//计算三者最大值
}
```

1. 非递归(动态规划)
    - 最大子序列 的第一个元素肯定非负
    - 最大子序列也不会以「和为负的子序列」开头

```
int FindGreatestSumOfSubArray(int a[],int n){
    int maxSum = 0;//如果题目中没有「特别说明」，则要初始化为 MINX_VALUE
    int thisSum = 0;
    for(int i=0;i<n;i++){
        thisSum += a[i];
        if(thisSum > maxSum){
            maxSum = thisSum;
        }else if(thisSum < 0){
            thisSum = 0;
        }
    }
}
```

2013年

二叉搜索树 从大到小输出所有不小于k 的数（2008原题）

2014年

无

2015年

孩子-兄弟表示法： 类定义，递归求高度（2006年原题）

16年

数组负值放在非负前面 -- 线性复杂度

```
/*
    思路：类似于快排的想法，从前往后扫描第一个非负数，从后往前扫描第一个负数
                交换他们俩
*/
void sort(int a[] , int n){
    int i = 0;
    int j = n-1;
    while(i < j){
        while(a[i] >= 0 && i<j)i++;
        while(a[j] < 0 && i<j)j--;
        int temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
}
```

不带头节点的链表（首节点指针为 f ），都是整型数据，递归求平均值(2005 期末原题)

```
float getAvg(LinkedNode<T>*first int &n){
    int n;
    if(first==NULL){
        return 0;
    }
    if(first->link==NULL){
        n=1;
        return (float)first->data;
    }else{
        float sum = getAvg(first->link,n);
        n++;
        return (first->data+sum)/n;
    }
}
```

以数组Q[m]，存放循环队列中的元素，同时以rear和length分别只是队尾位置和队中所含元素个数，实现该队列的类声明，队空队满条件，插入删除

2004原题

```
#include <iostream>
#include <assert.h>
using namespace std;

template <class Type>
class Queue                        //循环队列的类定义
{
private:
    int rear, length;                                    //队尾指针和队列长度
    Type *elements;                                      //存放队列元素的数组
    int maxSize;
public:
    Queue(int = 10);
    ~Queue() { delete[] elements; }
    void EnQueue(Type &item);
    Type DeQueue();
    Type GetFront();
    void MakeEmpty() { length = 0; }                     //置空队列
    int IsEmpty() const { return length == 0; }          //判队列是否为空
    int IsFull() const { return length == maxSize; }//判队列是否为满
                //队列最大可容纳元素个数
};

template <class Type>
Queue<Type>::Queue(int sz) :rear(maxSize - 1), length(0), maxSize(sz)   //建立一个最大具有maxSize个元素的空队列。
{
    elements = new Type[maxSize];                                                  //创建队列空间
    assert(elements != 0);                                                         //断言：动态存储分配成功与否
}

template <class Type>
void Queue<Type>::EnQueue(Type &item)
{
    assert(!IsFull());                    //判队列是否不满，满则出错处理
    length++;                             //长度加1
    rear = (rear + 1) % maxSize;          //队尾位置进1
    elements[rear] = item;                //进队列
}

template <class Type>
```

```
Type Queue<Type>::DeQueue()
{
    assert(!IsEmpty());                                    //判断队列是否不空，空则
出错处理
    length--;                                              //队列长度减1
    return elements[(rear - length + maxSize) % maxSize];  //返回原队头元素值
}

template <class Type>
Type Queue<Type>::GetFront()
{
    assert(!IsEmpty());
    return elements[(rear - length + maxSize) % maxSize];  //返回原队头元素值
}
//来自博客 https://www.cnblogs.com/si-lei/p/9426221.html
```

注：这里面的 「assert」就是软件工程里面的契约式编程 ✋

2017年

无

2018年

无

2013年备用卷

二叉搜索树实现优先队列: Insert()+Delete()

```
void BinaryTree<T>::Insert(BinaryTreeNode * root,T x){
    BinaryTreeNode<T> node = new BinaryTreeNode<T>(x);
    if(root==NULL)root = temp;
    if(root->data > x){
        if(root->leftChild!=NULL){
            Insert(root->leftChild,x);
        }else{
            root->leftChild = temp;
        }
    }else{
        if(root->right != NULL){
            Insert(root->right,x);
        }else{
            root->rightChild = temp;
        }
    }
}
bool BinaryTree<T>::Delete(BinaryTreeNode *root,T x){
```

```
    if(root==NULL)return false;
    if(root->data>x){return Delete(root->leftChild ,x );}
    if(root->data<x){return Delete(root->rightChild ,x);}
    //找到左子树的最右结点p，把它的值给当前结点，然后删除p
    BinaryTreeNode<T> *p = x.left;
    while(!p){p = p->right};
    root->data = p->data;
    delete p;
}
```