精选微软等数据结构+算法面试100题

--答案修正V0.2版本

此份答案是针对，前期已公布的最初的那份答案的，初步校正与修正。
http://download.csdn.net/source/2796735（**V0.1版**）
相比第一份V0.1版答案，此份答案V0.2版更加准确，亦修正了不少题目的答案。
此份20题的答案，思路更加清晰易懂，简介明了。

帖子更新地址：

http://topic.csdn.net/u/20101023/20/5652ccd7-d510-4c10-9671-307a56006e6d.html
若对已公布的面试题有任何问题，请把意见发表在上述帖子上，探讨交流。
所有源码及答案，只做了初步校正，欢迎批评指正。

My Blog:

http://blog.csdn.net/v_JULY_v

My Sina Blog:

http://blog.sina.com.cn/shitou009

My E-mail:

zhoulei0907@yahoo.cn

//东华理工、July 整理编辑。2010/11/06。
谢谢。

我很享受思考的过程，个人思考的全部结果，都放在了这篇帖子上，
http://topic.csdn.net/u/20101023/20/5652ccd7-d510-4c10-9671-307a56006e6d.html
现在，我要，好好整理下，上篇帖子上，已公布出来的题目答案 了。
展示自己的思考结果，我觉得很骄傲。:)。

2010年 10月18日下午 July

1.把二元查找树转变成排序的双向链表
题目：
输入一棵二元查找树，将该二元查找树转换成一个排序的双向链表。
要求不能创建任何新的结点，只调整指针的指向。

```
   10
   / \
  6   14
/ \ / \
4  8 12 16
```

转换成双向链表
4=6=8=10=12=14=16。

首先我们定义的二元查找树 节点的数据结构如下：

```
struct BSTreeNode
{
  int m_nValue; // value of node
  BSTreeNode *m_pLeft; // left child of node
  BSTreeNode *m_pRight; // right child of node
};
```

```
//引用 245 楼 tree_star 的回复
#include <stdio.h>
#include <iostream.h>


struct BSTreeNode
{
    int m_nValue; // value of node
    BSTreeNode *m_pLeft; // left child of node
    BSTreeNode *m_pRight; // right child of node
};
typedef BSTreeNode DoubleList;
DoubleList * pHead;
DoubleList * pListIndex;

void convertToDoubleList(BSTreeNode * pCurrent);
// 创建二元查找树
void addBSTreeNode(BSTreeNode * & pCurrent, int value)
{
    if (NULL == pCurrent)
    {
        BSTreeNode * pBSTree = new BSTreeNode();
```

```
        pBSTree->m_pLeft = NULL;
        pBSTree->m_pRight = NULL;
        pBSTree->m_nValue = value;
        pCurrent = pBSTree;

    }
    else
    {
        if ((pCurrent->m_nValue) > value)
        {
            addBSTreeNode(pCurrent->m_pLeft, value);
        }
        else if ((pCurrent->m_nValue) < value)
        {
            addBSTreeNode(pCurrent->m_pRight, value);
        }
        else
        {
            //cout<<"重复加入节点"<<endl;
        }
    }
}

// 遍历二元查找树  中序
void ergodicBSTree(BSTreeNode * pCurrent)
{
    if (NULL == pCurrent)
    {
        return;
    }
    if (NULL != pCurrent->m_pLeft)
    {
        ergodicBSTree(pCurrent->m_pLeft);
    }

    // 节点接到链表尾部
    convertToDoubleList(pCurrent);
    // 右子树为空
    if (NULL != pCurrent->m_pRight)
    {
        ergodicBSTree(pCurrent->m_pRight);
    }
}
```

```
// 二叉树转换成 list
void  convertToDoubleList(BSTreeNode * pCurrent)
{
    pCurrent->m_pLeft = pListIndex;
    if (NULL != pListIndex)
    {
        pListIndex->m_pRight = pCurrent;
    }
    else
    {
        pHead = pCurrent;
    }
    pListIndex = pCurrent;
    cout<<pCurrent->m_nValue<<endl;
}

int main()
{
    BSTreeNode * pRoot = NULL;
    pListIndex = NULL;
    pHead = NULL;
    addBSTreeNode(pRoot, 10);
    addBSTreeNode(pRoot, 4);
    addBSTreeNode(pRoot, 6);
    addBSTreeNode(pRoot, 8);
    addBSTreeNode(pRoot, 12);
    addBSTreeNode(pRoot, 14);
    addBSTreeNode(pRoot, 15);
    addBSTreeNode(pRoot, 16);
    ergodicBSTree(pRoot);
    return 0;
}
//////////////////////////////////////////////
4
6
8
10
12
14
15
16
Press any key to continue
//////////////////////////////////////////////
```

2.设计包含 min 函数的栈。
定义栈的数据结构，要求添加一个 min 函数，能够得到栈的最小元素。
要求函数 min、push 以及 pop 的时间复杂度都是 O(1)。

结合链表一起做。
首先我做插入以下数字：10，7，3，3，8，5，2， 6
0: 10 -> NULL (MIN=10, POS=0)
1: 7 -> [0] (MIN=7, POS=1) 用数组表示堆栈，第0个元素表示栈底
2: 3 -> [1] (MIN=3, POS=2)
3: 3 -> [2] (MIN=3, POS=3)
4: 8 -> NULL (MIN=3, POS=3) 技巧在这里，因为8比当前的 MIN 大，所以弹出8不会对当前的 MIN 产生影响
5: 5 -> NULL (MIN=3, POS=3)
6: 2 -> [2] (MIN=2, POS=6) 如果2出栈了，那么3就是 MIN
7: 6 -> [6]

出栈的话采用类似方法修正。

所以，此题的第1小题，即是借助辅助栈，保存最小值，
且随时更新辅助栈中的元素。
如先后，push 2 6 4 1 5
stack A stack B（辅助栈）

```
4:  5       1      //push 5,min=p->[3]=1      ^
3:  1       1      //push 1,min=p->[3]=1      |   //此刻 push 进 A 的元素1小于 B 中栈顶元素2
2:  4       2      //push 4,min=p->[0]=2      |
1:  6       2      //push 6,min=p->[0]=2      |
0:  2       2      //push 2,min=p->[0]=2      |
```

push 第一个元素进 A，也把它 push 进 B，
当向 Apush 的元素比 B 中的元素小， 则也 push 进 B，即更新 B。否则，不动 B，保存原值。

向栈 A push 元素时，顺序由下至上。
辅助栈 B 中，始终保存着最小的元素。

然后，pop 栈 A 中元素，5 1 4 6 2

```
     A       B ->更新
4:   5       1    1     //pop 5,min=p->[3]=1      |
3:   1       1    2     //pop 1,min=p->[0]=2      |
2:   4       2    2     //pop 4,min=p->[0]=2      |
1:   6       2    2     //pop 6,min=p->[0]=2      |
0:   2       2    NULL  //pop 2,min=NULL          v
```

当 pop A 中的元素小于 B 中栈顶元素时，则也要 pop B 中栈顶元素。

3.求子数组的最大和

题目：

输入一个整形数组，数组里有正数也有负数。

数组中连续的一个或多个整数组成一个子数组，每个子数组都有一个和。

求所有子数组的和的最大值。要求时间复杂度为 O(n)。

例如输入的数组为1, -2, 3, 10, -4, 7, 2, -5，和最大的子数组为3, 10, -4, 7, 2，因此输出为该子数组的和18。

```
//July 2010/10/18

#include <iostream.h>

int maxSum(int* a, int n)
{
  int sum=0;
  int b=0;

  for(int i=0; i<n; i++)
  {
    if(b<0)
      b=a[i];
    else
      b+=a[i];
    if(sum<b)
      sum=b;
  }
```

```
  return sum;
}

int main()
{
    int a[10]={1,-8,6,3,-1,5,7,-2,0,1};
    cout<<maxSum(a,10)<<endl;
    return 0;
}
```

运行结果，如下：

```
20
Press any key to continue
```

```
int maxSum(int* a, int n)
{
  int sum=0;
  int b=0;

  for(int i=0; i<n; i++)
  {
    if(b<=0)            //此处修正下，把 b<0改为 b<=0
      b=a[i];
    else
      b+=a[i];
    if(sum<b)
      sum=b;
  }
  return sum;
}
```

//

解释下：

例如输入的数组为1, -2, 3, 10, -4, 7, 2, -5，

那么最大的子数组为3, 10, -4, 7, 2，

因此输出为该子数组的和18

所有的东西都在以下俩行，

即：

```
  b：0  1  -1  3  13   9  16  18  7
sum：0  1   1  3  13  13  16  18  18
```

其实算法很简单，当前面的几个数，加起来后，b<0后，

把 b 重新赋值，置为下一个元素，b=a[i]。
当 b>sum，则更新 sum=b;
若 b<sum，则 sum 保持原值，不更新。:)。July、10/31。
//

关于此第3题，有网友提出，说以上的解答，

没有给出如果数组中全部为负数的情况，的解决方案。

的确，我一直把，题目限定了是既有负数也有正数的，不考虑全部为负数的情况。

当然，若考虑如果数组，全部为负数，解决方案如下（网友给出的答案，未作测试）：

```
int maxSum(int* a, int n)
{
    int sum=0;
    int b=0;
    //针对数组全部为负数的判断。
    int nTemp=a[0];
    for(int j=1; j<n; j++)
    {
        if (nTemp<a[j])
            nTemp=a[j];
    }
    if (nTemp<0)
        return nTemp;
    for(int i=0; i<n; i++)
    {
        if(b<0)
            b=a[i];
        else
            b+=a[i];

        if(sum<b)
            sum=b;
```

```
    }
    return sum;
}

void main()
{
   //int a[]={1, 2, 3, 10, 4, 7, 2, 5};//
    int a[]={1, -2, 3, 10, -4, 7, 2, -5};
   //int a[]={-1, -2, -3, -10, -4, -7, -2, -5};
    printf("\n%d\n",maxSum(a,8));
    getchar();
}
```

//关于第4题，
当访问到某一结点时，把该结点添加到路径上，并累加当前结点的值。
如果当前结点为叶结点并且当前路径的和刚好等于输入的整数，则当前的路径符合要求，我们把它打印出来。

如果当前结点不是叶结点，则继续访问它的子结点。当前结点访问结束后，递归函数将自动回到父结点。
因此我们在函数退出之前要在路径上删除当前结点并减去当前结点的值，
以确保返回父结点时路径刚好是根结点到父结点的路径。

我们不难看出保存路径的数据结构实际上是一个栈结构，因为路径要与递归调用状态一致，

而递归调用本质就是一个压栈和出栈的过程。

```
void FindPath
(
      BinaryTreeNode*   pTreeNode,     // a node of binary tree
      int               expectedSum,   // the expected sum
      std::vector<int>& path,          // a path from root to current node
      int&              currentSum     // the sum of path
)
{
      if(!pTreeNode)
            return;

      currentSum += pTreeNode->m_nValue;
      path.push_back(pTreeNode->m_nValue);
```

```
        // if the node is a leaf, and the sum is same as pre-defined,
        // the path is what we want. print the path
        bool isLeaf = (!pTreeNode->m_pLeft && !pTreeNode->m_pRight);
        if(currentSum == expectedSum && isLeaf)
        {
            std::vector<int>::iterator iter = path.begin();
            for(; iter != path.end(); ++ iter)
                std::cout << *iter << '\t';
            std::cout << std::endl;
        }

        // if the node is not a leaf, goto its children
        if(pTreeNode->m_pLeft)
            FindPath(pTreeNode->m_pLeft, expectedSum, path, currentSum);
        if(pTreeNode->m_pRight)
            FindPath(pTreeNode->m_pRight, expectedSum, path, currentSum);

        // when we finish visiting a node and return to its parent node,
        // we should delete this node from the path and
        // minus the node's value from the current sum
        currentSum -= pTreeNode->m_nValue;
        path.pop_back();
}
```

5.查找最小的 k 个元素
题目：输入 n 个整数，输出其中最小的 k 个。
例如输入1，2，3，4，5，6，7和8这8个数字，
则最小的4个数字为1，2，3和4。
//July 2010/10/18
//引用自116 楼 wocaoqwer 的回复。

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

class MinK{
public:
    MinK(int *arr,int si):array(arr),size(si){}

    bool kmin(int k,int*& ret){
```

```
        if(k>size)
        {
            ret=NULL;
            return false;
        }
        else
        {
            ret=new int[k--];
            int i;
            for(i=0;i<=k;++i)
                ret[i]=array[i];
            for(int j=(k-1)/2;j>=0;--j)
                shiftDown(ret,j,k);
            for(;i<size;++i)
                if(array[i]<ret[0])
                {
                    ret[0]=array[i];
                    shiftDown(ret,0,k);
                }
                return true;
        }
    }

    void remove(int*& ret){
        delete[] ret;
        ret=NULL;
    }


private:
    void shiftDown(int *ret,int pos,int length){
        int t=ret[pos];
        for(int s=2*pos+1;s<=length;s=2*s+1){
            if(s<length&&ret[s]<ret[s+1])
                ++s;
            if(t<ret[s])
            {
                ret[pos]=ret[s];
                pos=s;
            }
            else break;
        }
        ret[pos]=t;
    }
```

```
    int *array;
    int size;
};


int main()
{
    int array[]={1,2,3,4,5,6,7,8};
    MinK mink(array,sizeof(array)/sizeof(array[0]));
    int *ret;
    int k=4;
    if(mink.kmin(k,ret))
    {
        for(int i=0;i<k;++i)
            cout<<ret[i]<<endl;
        mink.remove(ret);
    }
    return 0;
}


//////////////////////////////////////////
运行结果：
4
2
3
1
Press any key to continue
//////////////////////////////////////////
```

第6题

腾讯面试题：

给你10分钟时间，根据上排给出十个数，在其下排填出对应的十个数

要求下排每个数都是先前上排那十个数在下排出现的次数。

上排的十个数如下：

【0，1，2，3，4，5，6，7，8，9】

初看此题，貌似很难，10分钟过去了，可能有的人，题目都还没看懂。

举一个例子，
数值: 0,1,2,3,4,5,6,7,8,9
分配: 6,2,1,0,0,0,1,0,0,0
0在下排出现了6次，1在下排出现了2次，
2在下排出现了1次，3在下排出现了0次....
以此类推..
// 引用自 July 2010年10月18日。

```
//数值: 0,1,2,3,4,5,6,7,8,9
//分配: 6,2,1,0,0,0,1,0,0,0


#include <iostream.h>
#define len 10


class NumberTB
{
private:
    int top[len];
    int bottom[len];
    bool success;
public:
    NumberTB();
    int* getBottom();
    void setNextBottom();
    int getFrequecy(int num);
};

NumberTB::NumberTB()
{
    success = false;
    //format top
    for(int i=0;i<len;i++)
    {
        top[i] = i;
    }
}


int* NumberTB::getBottom()
{
    int i = 0;
    while(!success)
```

```
    {
        i++;
        setNextBottom();
    }
    return bottom;
}

//set next bottom
void NumberTB::setNextBottom()
{
    bool reB = true;

    for(int i=0;i<len;i++)
    {
        int frequecy = getFrequecy(i);

        if(bottom[i] != frequecy)
        {
            bottom[i] = frequecy;
            reB = false;
        }
    }
    success = reB;
}

//get frequency in bottom
int NumberTB::getFrequecy(int num)   //此处的num即指上排的数 i
{
    int count = 0;

    for(int i=0;i<len;i++)
    {
        if(bottom[i] == num)
            count++;
    }
    return count;    //cout即对应 frequecy
}

int main()
{
    NumberTB nTB;
    int* result= nTB.getBottom();

    for(int i=0;i<len;i++)
```

```
    {
        cout<<*result++<<endl;
    }
    return 0;
}
//////////////////////////////////////////
```
运行结果：
```
6
2
1
0
0
0
1
0
0
0
Press any key to continue
//////////////////////////////////////////
```

第7题

微软亚院之编程判断俩个链表是否相交

给出俩个单向链表的头指针，比如h1，h2，判断这俩个链表是否相交。

为了简化问题，我们假设俩个链表均不带环。

问题扩展：

1.如果链表可能有环列?

2.如果需要求出俩个链表相交的第一个节点列?

```
//这一题，自己也和不少人讨论过了，
//更详细的，请看这里：
//My sina Blog：
//http://blog.sina.com.cn/s/blog_5e3ab00c0100le4s.html
```

1.首先假定链表不带环

那么，我们只要判断俩个链表的尾指针是否相等。

相等，则链表相交；否则，链表不相交。

2.如果链表带环，

那判断一链表上俩指针相遇的那个节点，在不在另一条链表上。

如果在，则相交，如果不在，则不相交。

所以，事实上，这个问题就转化成了：
1.先判断带不带环
2.如果都不带环，就判断尾节点是否相等
3.如果都带环，判断一链表上俩指针相遇的那个节点，在不在另一条链表上。
如果在，则相交，如果不在，则不相交。

```
//用两个指针，一个指针步长为1，一个指针步长为2，判断链表是否有环
bool check(const node* head)
{
    if(head==NULL)
      return false;
    node *low=head, *fast=head->next;
    while(fast!=NULL && fast->next!=NULL)
    {
        low=low->next;
        fast=fast->next->next;
        if(low==fast) return true;
    }
    return false;
}

//如果链表可能有环，则如何判断两个链表是否相交
//思路：链表1 步长为1，链表2步长为2 ，如果有环且相交则肯定相遇，否则不相交
list1 head: p1
list2 head: p2
while( p1 != p2 && p1 != NULL && p2 != NULL )
[b]//但当链表有环但不相交时，此处是死循环。![/b]
{
      p1 = p1->next;
      if ( p2->next )
         p2 = p2->next->next;
      else
         p2 = p2->next;
}
if ( p1 == p2 && p1 && p2)
   //相交
else
  //不相交
[color=#FF0000][b]所以，判断带环的链表，相不相交，只能这样[/b]：[/color]
如果都带环，判断一链表上俩指针相遇的那个节点，在不在另一条链表上。
```

如果在，则相交，如果不在，则不相交。(未写代码实现，见谅。:)..

第9题

判断整数序列是不是二元查找树的后序遍历结果
题目：输入一个整数数组，判断该数组是不是某二元查找树的后序遍历的结果。
如果是返回 true，否则返回 false。
例如输入5、7、6、9、11、10、8，由于这一整数序列是如下树的后序遍历结果：

```
    8
   / \
  6   10
/ \ / \
5 7 9 11
```

因此返回 true。
如果输入7、4、6、5，没有哪棵树的后序遍历的结果是这个序列，因此返回 false。
//貌似，少有人关注此题。:).2010/10/18

```
bool verifySquenceOfBST(int squence[], int length)
{
	if(squence == NULL || length <= 0)
		return false;

	// root of a BST is at the end of post order traversal squence
	int root = squence[length - 1];

	// the nodes in left sub-tree are less than the root
	int i = 0;
	for(; i < length - 1; ++ i)
	{
		if(squence[i] > root)
			break;
	}

	// the nodes in the right sub-tree are greater than the root
	int j = i;
	for(; j < length - 1; ++ j)
	{
		if(squence[j] < root)
			return false;
```

```
        }

        // verify whether the left sub-tree is a BST
        bool left = true;
        if(i > 0)
              left = verifySquenceOfBST(squence, i);

        // verify whether the right sub-tree is a BST
        bool right = true;
        if(i < length - 1)
              right = verifySquenceOfBST(squence + i, length - i - 1);

        return (left && right);
}
```

第9题：

其实，就是一个后序遍历二叉树的算法。

关键点：

1.

```
        //确定根结点
        int root = squence[length - 1];
```

2.

```
        // the nodes in left sub-tree are less than the root
        int i = 0;
        for(; i < length - 1; ++ i)
        {
              if(squence[i] > root)
                    break;
        }

        // the nodes in the right sub-tree are greater than the root
        int j = i;
        for(; j < length - 1; ++ j)
        {
              if(squence[j] < root)
                    return false;
        }
```

3.

递归遍历，左右子树。

```
-------------------------------------

//第10题，单词翻转。
//单词翻转，引用自117 楼 wocaoqwer 的回复。


#include<iostream>
#include<string>
using namespace std;

class ReverseWords{
public:
    ReverseWords(string* wo):words(wo){}
    void reverse_()
    {
        int length=words->size();
        int begin=-1,end=-1;
        for(int i=0;i<length;++i){
            if(begin==-1&&words->at(i)==' ')
                continue;
            if(begin==-1)
            {
                begin=i;
                continue;
            }
            if(words->at(i)==' ')
                end=i-1;
            else if(i==length-1)
                end=i;
            else
        continue;
            reverse__(begin,end);     //1.字母翻转
            begin=-1,end=-1;
        }
        reverse__(0,length-1);        //2.单词翻转
    }

private:
    void reverse__(int begin,int end)   //
    {
        while(begin<end)
```

```
    {
            char t=words->at(begin);
            words->at(begin)=words->at(end);
            words->at(end)=t;
            ++begin;
            --end;
        }
    }
    string* words;
};


int main(){
    string s="I  am a student.";
    ReverseWords r(&s);
    r.reverse_();
    cout<<s<<endl;

    return 0;
}
```

运行结果：

```
student. a am  I
Press any key to continue
```

第11题

求二叉树中节点的最大距离...

如果我们把二叉树看成一个图，
父子节点之间的连线看成是双向的，
我们姑且定义"距离"为两节点之间边的个数。
写一个程序，
求一棵二叉树中相距最远的两个节点之间的距离。

```
//July  2010/10/19
//此题思路，tree_star and i 在257、258楼，讲的很明白了。



//定义一个结构体
```

```
struct NODE
{
    NODE* pLeft;
    NODE* pRight;
    int MaxLen;
    int MaxRgt;
};
NODE* pRoot;  //根节点
int MaxLength;

void traversal_MaxLen(NODE* pRoot)
{
    if(pRoot == NULL)
    {
        return 0;
    };

    if(pRoot->pLeft == NULL)
    {
        pRoot->MaxLeft = 0;
    }
    else                                  //若左子树不为空
    {
        int TempLen = 0;
        if(pRoot->pLeft->MaxLeft > pRoot->pLeft->MaxRight)
          //左子树上的，某一节点，往左边大，还是往右边大
        {
            TempLen+=pRoot->pLeft->MaxLeft;
        }
        else
        {
            TempLen+=pRoot->pLeft->MaxRight;
        }
        pRoot->nMaxLeft = TempLen + 1;
        traversal_MaxLen(NODE* pRoot->pLeft);
        //此处，加上递归
    }


    if(pRoot->pRigth == NULL)
    {
        pRoot->MaxRight = 0;
    }
    else                                  //若右子树不为空
```

```
    {
        int TempLen = 0;
        if(pRoot->pRight->MaxLeft > pRoot->pRight->MaxRight)
        //右子树上的，某一节点，往左边大，还是往右边大
        {
            TempLen+=pRoot->pRight->MaxLeft;
        }
        else
        {
            TempLen+=pRoot->pRight->MaxRight;
        }
        pRoot->MaxRight = TempLen + 1;
        traversal_MaxLen(NODE* pRoot->pRight);
        //此处，加上递归
    }

   if(pRoot->MaxLeft + pRoot->MaxRight > 0)
    {
        MaxLength=pRoot->nMaxLeft + pRoot->MaxRight;
    }
}



// 数据结构定义
    struct NODE
    {
         NODE* pLeft;             // 左子树
         NODE* pRight;           // 右子树
         int nMaxLeft;           // 左子树中的最长距离
         int nMaxRight;          // 右子树中的最长距离
         char chValue;         // 该节点的值
    };

    int nMaxLen = 0;

    // 寻找树中最长的两段距离
    void FindMaxLen(NODE* pRoot)
    {
         // 遍历到叶子节点，返回
         if(pRoot == NULL)
         {
              return;
         }
```

```
// 如果左子树为空，那么该节点的左边最长距离为0
if(pRoot -> pLeft == NULL)
{
     pRoot -> nMaxLeft = 0;
}

// 如果右子树为空，那么该节点的右边最长距离为0
if(pRoot -> pRight == NULL)
{
     pRoot -> nMaxRight = 0;
}

// 如果左子树不为空，递归寻找左子树最长距离
if(pRoot -> pLeft != NULL)
{
     FindMaxLen(pRoot -> pLeft);
}

// 如果右子树不为空，递归寻找右子树最长距离
if(pRoot -> pRight != NULL)
{
     FindMaxLen(pRoot -> pRight);
}

// 计算左子树最长节点距离
if(pRoot -> pLeft != NULL)
{
     int nTempMax = 0;
     if(pRoot -> pLeft -> nMaxLeft > pRoot -> pLeft -> nMaxRight)
     {
          nTempMax = pRoot -> pLeft -> nMaxLeft;
     }
     else
     {
          nTempMax = pRoot -> pLeft -> nMaxRight;
     }
     pRoot -> nMaxLeft = nTempMax + 1;
}

// 计算右子树最长节点距离
if(pRoot -> pRight != NULL)
{
     int nTempMax = 0;
     if(pRoot -> pRight -> nMaxLeft > pRoot -> pRight -> nMaxRight)
```

```
            {
                nTempMax = pRoot -> pRight -> nMaxLeft;
            }
            else
            {
                nTempMax = pRoot -> pRight -> nMaxRight;
            }
            pRoot -> nMaxRight = nTempMax + 1;
        }

        // 更新最长距离
        if(pRoot -> nMaxLeft + pRoot -> nMaxRight > nMaxLen)
        {
            nMaxLen = pRoot -> nMaxLeft + pRoot -> nMaxRight;
        }
    }
    //很明显，思路完全一样，但书上给的这段代码更规范！:)。
```

第12题

题目：求1+2+…+n，
要求不能使用乘除法、for、while、if、else、switch、case 等关键字
以及条件判断语句（A?B:C）。
//July、2010/10/19

循环只是让相同的代码执行 n 遍而已，我们完全可以不用 for 和 while 达到这个效果。
比如定义一个类，我们 new 一含有 n 个这种类型元素的数组，
那么该类的构造函数将确定会被调用 n 次。我们可以将需要执行的代码放到构造函数里。

```
#include <iostream.h>

class Temp
{
public:
      Temp()
      {
          ++N;
```

```
            Sum += N;
        }
        static void Reset() { N = 0; Sum = 0; }
        static int GetSum() { return Sum; }

private:
        static int N;
        static int Sum;
};

int Temp::N = 0;
int Temp::Sum = 0;

int solution1_Sum(int n)
{
        Temp::Reset();

        Temp *a = new Temp[n];   //就是这个意思，new出n个数组。
         delete []a;
        a = 0;

        return Temp::GetSum();
}

int main()
{
    cout<<solution1_Sum(100)<<endl;
    return 0;
}

//运行结果：
//5050
//Press any key to continue



//July、2010/10/19

//第二种思路：
----------------
既然不能判断是不是应该终止递归，我们不妨定义两个函数。
一个函数充当递归函数的角色，另一个函数处理终止递归的情况，
我们需要做的就是在两个函数里二选一。
```

从二选一我们很自然的想到布尔变量，
比如 ture/(1)的时候调用第一个函数，false/(0)的时候调用第二个函数。
那现在的问题是如和把数值变量 n 转换成布尔值。

如果对 n 连续做两次反运算，即!!n，那么非零的 n 转换为 true，0转换为 false。

```
#include <iostream.h>

class A;
A* Array[2];

class A
{
public:
    virtual int Sum (int n) { return 0; }
};

class B: public A
{
public:
    virtual int Sum (int n) { return Array[!!n]->Sum(n-1)+n; }
};


int solution2_Sum(int n)
{
    A a;
    B b;
    Array[0] = &a;
    Array[1] = &b;

    int value = Array[1]->Sum(n);
    //利用虚函数的特性，当 Array[1]为0时，即 Array[0] = &a; 执行 A::Sum，
    //当 Array[1]不为0时，                即 Array[1] = &b; 执行 B::Sum。

    return value;
}

int main()
{
    cout<<solution2_Sum(100)<<endl;
    return 0;
}
```

```
//5050
//Press any key to continue
```

第13题：
题目：
输入一个单向链表，输出该链表中倒数第 k 个结点，
链表的倒数第0个结点为链表的尾指针。

//此题一出，相信，稍微有点 经验的同志，都会说到：

设置两个指针 p1,p2
首先 p1和 p2都指向 head
然后 p2向前走 n 步，这样 p1和 p2之间就间隔 k 个节点
然后 p1和 p2同……

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

struct ListNode
{
    char data;
    ListNode* next;
};
ListNode* head,*p,*q;
ListNode *pone,*ptwo;

ListNode* fun(ListNode *head,int k)
{
    pone = ptwo = head;
    for(int i=0;i<=k-1;i++)
        ptwo=ptwo->next;
    while(ptwo!=NULL)
    {
        pone=pone->next;
        ptwo=ptwo->next;
    }
    return pone;
}
```

```
int main()
{
    char c;
    head = (ListNode*)malloc(sizeof(ListNode));
    head->next = NULL;
    p = head;
    while(c !='0')
    {
        q = (ListNode*)malloc(sizeof(ListNode));
        q->data = c;
        q->next = NULL;
        p->next = q;
        p = p->next;
        c = getchar();
    }
    cout<<"---------------"<<endl;
    cout<<fun(head,2)->data<<endl;

    return 0;
}

//////////////////////////
1254863210
---------------
2
Press any key to continue
//////////////////////////
```

第14题：
题目：输入一个已经按升序排序过的数组和一个数字，
在数组中查找两个数，使得它们的和正好是输入的那个数字。
要求时间复杂度是O(n)。
  如果有多对数字的和等于输入的数字，输出任意一对即可。
  例如输入数组1、2、4、7、11、15和数字15。由于4+11=15，因此输出4和11。
//由于数组已经过升序排列，所以，难度下降了不少。
//July、2010/10/19

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

```
bool FindTwoNumbersWithSum
(
int data[],           // 已经排序的 数组
unsigned int length,  // 数组长度
int sum,              //用户输入的 sum
int& num1,            // 输出符合和等于 sum 的第一个数
int& num2             // 第二个数
)
{
    bool found = false;
    if(length < 1)
        return found;

    int begin = 0;
    int end = length - 1;

    while(end > begin)
    {
        long curSum = data[begin] + data[end];

        if(curSum == sum)
        {
            num1 = data[begin];
            num2 = data[end];
            found = true;
            break;
        }
        else if(curSum > sum)
            end--;
        else
            begin++;
    }
    return found;
}


int main()
{
    int x,y;
    int a[6]={1,2,4,7,11,15};
    if(FindTwoNumbersWithSum(a,6,15,x,y) )
    {
        cout<<x<<endl<<y<<endl;
```

```
    }
    return 0;
}
4
11
Press any key to continue
```

//扩展：如果输入的数组是没有排序的，但知道里面数字的范围，其他条件不变，
//如何在O(n)时间里找到这两个数字？

关于第14题，
1.题目假定是，只要找出俩个数，的和等于给定的数，
其实是，当给定一排数，
4,5,7,10,12
然后给定一个数，22。
就有俩种可能了。因为22=10+12=10+5+7。
而恰恰与第4题，有关联了。望大家继续思考下。:)。

2.第14题，还有一种思路，如下俩个数组：
1、 2、 4、7、11、15 //用15减一下为
14、13、11、8、4、 0 //如果下面出现了和上面一样的数，稍加判断，就能找出这俩个数来了。

第一个数组向右扫描，第二个数组向左扫描。

第15题：
题目：输入一颗二元查找树，将该树转换为它的镜像，
即在转换后的二元查找树中，左子树的结点都大于右子树的结点。
用递归和循环两种方法完成树的镜像转换。
例如输入：

```
    8
  / \
6   10
/ \ / \
5 7 9 11
```

输出：

```
    8
   / \
  10  6
/ \ / \
11 9 7  5
```

定义二元查找树的结点为：

```
struct BSTreeNode // a node in the binary search tree (BST)
{
  int m_nValue; // value of node
  BSTreeNode *m_pLeft; // left child of node
  BSTreeNode *m_pRight; // right child of node
};
//就是递归翻转树，有子树则递归翻转子树。
//July、2010/10/19
```

```
void Revertsetree(list *root)
{
    if(!root)
       return;
    list *p;

    p=root->leftch;
    root->leftch=root->rightch;
    root->rightch=p;

    if(root->leftch)
      Revertsetree(root->leftch);
    if(root->rightch)
      Revertsetree(root->rightch);
}
```

由于递归的本质是编译器生成了一个函数调用的栈，
因此用循环来完成同样任务时最简单的办法就是用一个辅助栈来模拟递归。

首先我们把树的头结点放入栈中。
在循环中，只要栈不为空，弹出栈的栈顶结点，交换它的左右子树。

如果它有左子树，把它的左子树压入栈中；
如果它有右子树，把它的右子树压入栈中。

这样在下次循环中就能交换它儿子结点的左右子树了。
//再用辅助栈模拟递归，改成循环的（有误之处，望不吝指正）：

```
void Revertsetree(list *phead)
{
    if(!phead)
       return;

    stack<list*> stacklist;
    stacklist.push(phead);          //首先把树的头结点放入栈中。

    while(stacklist.size())
    //在循环中，只要栈不为空，弹出栈的栈顶结点，交换它的左右子树
    {
      list* pnode=stacklist.top();
      stacklist.pop();

      list *ptemp;
      ptemp=pnode->leftch;
      pnode->leftch=pnode->rightch;
      pnode->rightch=ptemp;

      if(pnode->leftch)
        stacklist.push(pnode->leftch);   //若有左子树，把它的左子树压入栈中
      if(pnode->rightch)
        stacklist.push(pnode->rightch);  //若有右子树，把它的右子树压入栈中
    }
}
```

第16题
题目：输入一颗二元树，从上往下按层打印树的每个结点，同一层中按照从左往右的顺序打印。
例如输入

```
      8
    /  \
   6    10
  /\     /\
5  7   9  11
```

输出8　6　10　5　7　9　11。
//题目不是我们所熟悉的，树的前序，中序，后序。即是树的层次遍历。

/*308 楼 panda_lin 的回复，说的已经很好了。:)
利用队列,每个单元对应二叉树的一个节点.
1：输出8，　队列内容：6，10
2：输出6，　6的2个子节点5,7入队列。队列的内容：10，　5，　7
3：输出10，10的2个子节点9，11入队列。队列的内容：5，7，9，11。
4：输出5 ，5没有子节点。队列的内容：7，9，11
5:。。。

由于 STL 已经为我们实现了一个很好的 deque（两端都可以进出的队列），
我们只需要拿过来用就可以了。

我们知道树是图的一种特殊退化形式。
同时如果对图的深度优先遍历和广度优先遍历有比较深刻的理解，

将不难看出这种遍历方式实际上是一种广度优先遍历。
因此这道题的本质是在二元树上实现广度优先遍历。
//July、2010/10/19/晚。

```
#include <deque>
#include <iostream>
using namespace std;

struct BTreeNode // a node in the binary tree
{
	int         m_nValue; // value of node
	BTreeNode  *m_pLeft;  // left child of node
	BTreeNode  *m_pRight; // right child of node
};
BTreeNode* pListIndex;
BTreeNode* pHead;

void PrintFromTopToBottom(BTreeNode *pTreeRoot)
{
	if(!pTreeRoot)
		return;

	// get a empty queue
	deque<BTreeNode *> dequeTreeNode;

	// insert the root at the tail of queue
```

```
    dequeTreeNode.push_back(pTreeRoot);

    while(dequeTreeNode.size())
    {
        // get a node from the head of queue
        BTreeNode *pNode = dequeTreeNode.front();
        dequeTreeNode.pop_front();

        // print the node
        cout << pNode->m_nValue << ' ';

        // print its left child sub-tree if it has
        if(pNode->m_pLeft)
            dequeTreeNode.push_back(pNode->m_pLeft);
        // print its right child sub-tree if it has
        if(pNode->m_pRight)
            dequeTreeNode.push_back(pNode->m_pRight);
    }
}

// 创建二元查找树
void addBTreeNode(BTreeNode * & pCurrent, int value)
{
    if (NULL == pCurrent)
    {
        BTreeNode * pBTree = new BTreeNode();
        pBTree->m_pLeft = NULL;
        pBTree->m_pRight = NULL;
        pBTree->m_nValue = value;
        pCurrent = pBTree;

    }
    else
    {
        if ((pCurrent->m_nValue) > value)
        {
            addBTreeNode(pCurrent->m_pLeft, value);
        }
        else if ((pCurrent->m_nValue) < value)
        {
            addBTreeNode(pCurrent->m_pRight, value);
        }
    }
}
```

```
int main()
{
    BTreeNode * pRoot = NULL;
    pListIndex = NULL;
    pHead = NULL;
    addBTreeNode(pRoot, 8);
    addBTreeNode(pRoot, 6);
    addBTreeNode(pRoot, 5);
    addBTreeNode(pRoot, 7);
    addBTreeNode(pRoot, 10);
    addBTreeNode(pRoot, 9);
    addBTreeNode(pRoot, 11);
    PrintFromTopToBottom(pRoot);
    return 0;
}

//输出结果：
//8 6 10 5 7 9 11 Press any key to continue
```

是的，由这道题，突然想到了，树的广度优先遍历，BFS 算法，
算法王帖：精选经典的24个算法 [3.BFS 和 DFS 优先搜索]
http://blog.sina.com.cn/s/blog_5e3ab00c0100lya2.html

第17题：
题目：在一个字符串中找到第一个只出现一次的字符。
如输入 abaccdeff，则输出 b。
这道题是2006年 google 的一道笔试题。

思路剖析：由于题目与字符出现的次数相关，我们可以统计每个字符在该字符串中出现的次数.
要达到这个目的，需要一个数据容器来存放每个字符的出现次数。

在这个数据容器中可以根据字符来查找它出现的次数，
也就是说这个容器的作用是把一个字符映射成一个数字。

在常用的数据容器中，哈希表正是这个用途。
由于本题的特殊性，我们只需要一个非常简单的哈希表就能满足要求。

由于字符（char)是一个长度为8的数据类型，因此总共有可能256 种可能。
于是我们创建一个长度为256的数组，每个字母根据其 ASCII 码值作为数组的下标对应数组的对应项，
而数组中存储的是每个字符对应的次数。

这样我们就创建了一个大小为256，以字符 ASCII 码为键值的哈希表。
我们第一遍扫描这个数组时，每碰到一个字符，在哈希表中找到对应的项并把出现的次数增加一次。
这样在进行第二次扫描时，就能直接从哈希表中得到每个字符出现的次数了。
//July、2010/10/20

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


char FirstNotRepeatingChar(char* pString)
{
	if(!pString)
		return 0;

	const int tableSize = 256;
	unsigned int hashTable[tableSize];
	for(unsigned int i = 0; i < tableSize; ++ i)
		hashTable[i] = 0;

	char* pHashKey = pString;
	while(*(pHashKey) != '\0')
		hashTable[*(pHashKey++)] ++;

	pHashKey = pString;
	while(*pHashKey != '\0')
	{
		if(hashTable[*pHashKey] == 1)
			return *pHashKey;

		pHashKey++;
	}

	return *pHashKey;
}

int main()
{
    cout<<"请输入一串字符："<<endl;
```

```
    char s[100];
    cin>>s;
    char* ps=s;
    cout<<FirstNotRepeatingChar(ps)<<endl;
    return 0;
}


////////////////////////////////////////
请输入一串字符：
abaccdeff
b
Press any key to continue
////////////////////////////////////////
```

第18题：
题目：n 个数字（0,1,…,n-1）形成一个圆圈，从数字0开始，
每次从这个圆圈中删除第 m 个数字（第一个为当前数字本身，第二个为当前数字的下一个数字）。
当一个数字删除后，从被删除数字的下一个继续删除第 m 个数字。
求出在这个圆圈中剩下的最后一个数字。
July：我想，这个题目，不少人已经 见识过了。

先看这个题目的简单变形。
n 个人围成一圈，顺序排号。从第一个人开始报数（从1到3报数），
凡报到3的人退出圈子，问最后留下的是原来第几号的那个人?

```
//July、2010/10/20
//我把这100题，当每日必须完成的作业，来做了。:)。


#include <stdio.h>

int main()
{
    int i,k,m,n,num[50],*p;
    printf("input number of person:n=");
    scanf("%d",&n);

    printf("input number of the quit:m=");   //留下->18题
    scanf("%d",&m);                          //留下->18题
```

```
        p=num;
    for(i=0;i<n;i++)
        *(p+i)=i+1;     //给每个人编号
    i=0;   //报数
    k=0;   //此处为3
//    m=0;   //m 为退出人数                          //去掉->18题
    while(m<n-1)
    {
        if(*(p+i)!=0)
            k++;
        if(k==3)
        {
            *(p+i)=0;     //退出，对应的数组元素置为0
            k=0;
            m++;
        }
        i++;
        if(i==n)
            i=0;
    }
    while(*p==0)
        p++;
    printf("The last one is NO.%d\n",*p);
}



////////////////////////////////////
int LastRemaining_Solution2(int n, unsigned int m)
{
      // invalid input
      if(n <= 0 || m < 0)
            return -1;

      // if there are only one integer in the circle initially,
      // of course the last remaining one is 0
      int lastinteger = 0;

      // find the last remaining one in the circle with n integers
      for (int i = 2; i <= n; i ++)
            lastinteger = (lastinteger + m) % i;

      return lastinteger;
}
////////////////////////////////////////////
```

第19题：
题目：定义 Fibonacci 数列如下：

```
  /   0                  n=0
f(n)= 1                  n=1,2
  \ f(n-1)+f(n-2)        n>2
```

输入 n，用最快的方法求该数列的第 n 项。
分析：在很多 C 语言教科书中讲到递归函数的时候，都会用 Fibonacci 作为例子。
因此很多程序员对这道题的递归解法非常熟悉，但....呵呵，你知道的。。

```
//0 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597.....
//注意，当求第100项，甚至更大的项时，请确保你用什么类型，长整型?or long long int
存储。
//不然，计算机，将 得不到结果。
```

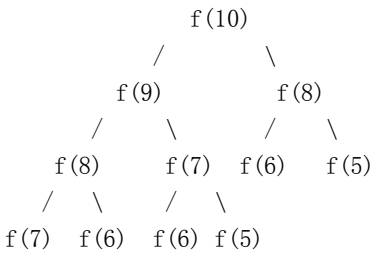
```
//若用递归方法，可写 下如下代码：
#include <iostream.h>

int Fibona(int n)
{
    int m;
    if(n==0)
        return 0;
    else if(n==1||n==2)
        return 1;
    else
    {
        m=Fibona(n-1)+Fibona(n-2);
        return m;
    }
}

int main()
{
```

```
    cout<<"-----------------"<<endl;
    cout<<Fibona(17)<<endl;
    return 0;
}
```

科书上反复用这个题目来讲解递归函数，并不能说明递归解法最适合这道题目。
我们以求解 f(10)作为例子来分析递归求解的过程。
要求得 f(10)，需要求得 f(9)和 f(8)。同样，要求得 f(9)，要先求得 f(8)和 f(7)……
我们用树形结构来表示这种依赖关系



更简单的办法是从下往上计算，首先根据 f(0)和 f(1)算出 f(2)，再根据 f(1)和 f(2)算出 f(3)……
依此类推就可以算出第 n 项了。很容易理解，这种思路的时间复杂度是 O(n)。

其实，就是转化为非递归程序，用递推。！

```
long long Fibonacci_Solution2(unsigned n)
{
      int result[2] = {0, 1};
      if(n < 2)
            return result[n];

      long long  fibNMinusOne = 1;
      long long  fibNMinusTwo = 0;
      long long  fibN = 0;
      for(unsigned int i = 2; i <= n; ++ i)
      {
            fibN = fibNMinusOne + fibNMinusTwo;

            fibNMinusTwo = fibNMinusOne;
            fibNMinusOne = fibN;
      }
```

```
        return fibN;
}
//很可惜，这还不是最快的方法。
//还有一种方法，可达到，时间复杂度为 0(lgn).
//.....
```

第20题：
题目：输入一个表示整数的字符串，把该字符串转换成整数并输出。
例如输入字符串"345"，则输出整数345。

此题一点也不简单。不信，你就先不看一下的代码，
你自己先写一份，然后再对比一下，便知道了。

1.转换的思路：每扫描到一个字符，我们把在之前得到的数字乘以10再加上当前字符表示的数字。
这个思路用循环不难实现。
2.由于整数可能不仅仅之含有数字，还有可能以'+'或者'-'开头，表示整数的正负。
如果第一个字符是'+'号，则不需要做任何操作；如果第一个字符是'-'号，
则表明这个整数是个负数，在最后的时候我们要把得到的数值变成负数。
3.接着我们试着处理非法输入。由于输入的是指针，在使用指针之前，
我们要做的第一件是判断这个指针是不是为空。
如果试着去访问空指针，将不可避免地导致程序崩溃。
4.输入的字符串中可能含有不是数字的字符。
每当碰到这些非法的字符，我们就没有必要再继续转换。

最后一个需要考虑的问题是溢出问题。由于输入的数字是以字符串的形式输入，
因此有可能输入一个很大的数字转换之后会超过能够表示的最大的整数而溢出。

```
//July、2010、10/22。
enum Status {kValid = 0, kInvalid};
int g_nStatus = kValid;

int StrToInt(const char* str)
{
      g_nStatus = kInvalid;
      long long num = 0;

      if(str != NULL)
```

```
{
	const char* digit = str;

	// the first char in the string maybe '+' or '-'
	bool minus = false;
	if(*digit == '+')
		digit ++;
	else if(*digit == '-')
	{
		digit ++;
		minus = true;
	}

	// the remaining chars in the string
	while(*digit != '\0')
	{
		if(*digit >= '0' && *digit <= '9')
		{
			num = num * 10 + (*digit - '0');

			// overflow
			if(num > std::numeric_limits<int>::max())
			{
				num = 0;
				break;
			}

			digit ++;
		}
		// if the char is not a digit, invalid input
		else
		{
			num = 0;
			break;
		}
	}

	if(*digit == '\0')
	{
		g_nStatus = kValid;
		if(minus)
			num = 0 - num;
	}
}
return static_cast<int>(num);
```

}

//在C语言提供的库函数中，函数atoi能够把字符串转换整数。
//它的声明是int atoi(const char *str)。该函数就是用一个全局变量来标志输入是否合法的。

此份答案是针对，前期已公布的最初的那份答案的，初步校正与修正。
http://download.csdn.net/source/2796735（**V0.1版**）
相比第一份V0.1版答案，此份答案V0.2版更加准确，亦修正了不少题目的答案。
此份20题的答案，思路更加清晰易懂，简介明了。

帖子更新地址：
http://topic.csdn.net/u/20101023/20/5652ccd7-d510-4c10-9671-307a56006e6d.html
若对已公布的面试题有任何问题，请把意见发表在上述帖子上，探讨交流。
所有源码及答案，只做了初步校正，欢迎批评指正。
My Blog:
http://blog.csdn.net/v_JULY_v
My Sina Blog:
http://blog.sina.com.cn/shitou009
My E-mail:
zhoulei0907@yahoo.cn
//东华理工、July 整理编辑。2010/11/06。
谢谢。

欢迎各位，对以上公布的任何一题的答案，提出质疑，
有不精准之处，望批评指正。多谢 各位了。:)。

July、2010/11/06。
Zhoulei0907@yahoo.cn