数据结构课后习题

Chapter1

1. Write a recursive method that returns the number of 1 s in the binary representation of N. Use the fact that is equal to the number of 1 s in the representation of N/2, plus 1, if N is odd.

分析：设N的二进制表示有n位，即求n位二进制数中1的个数，即求前n - 1位二进制数中1的个数与最后一位1的个数的总和。

代码：

```
public int count(int n) {
   if(n==0) {
      return 0;
   }
   else {
      return n%2 +count(n/2);
   }
}
```

2. Write the routines wise the following declarations:

```
public void permute( String str );
private void permute( char [ ] str, int low, int high )
```

The first routine is a driver that calls the second and prints all the permutations of the characters in String str. If str is “abc”, then the strings that are output are abc, acb, bac, bca, cab,and cba. Use recursion for the second routine.

分析：字符串中只有一个字符时，全排列就是该字符本身；

依次将每个字符置出，递归求剩余字符的全排列，即可得到整个字符串的全排列。

代码：

```
public void permute(String str){
   int len = str.length();
   permute(str.toCharArray, 0, len);
}
private void permute(char[] str, int low, int high){
   if(low == high)     system.out.println(str);
   else{
     for(int i = low; i < high; i++){
        char temp = str[low];
        str[low] = str[i];
        str[i] = temp;      //将 i 位置的字符置出，存放在第一位，
        permute(str,low+1,high);   //求剩余字符串的全排列
        temp = str[low];
        str[low] = str[i];
        str[i] = temp;     //将 i 位置的字符恢复原位置
     }
   }
}
```

3. 已知a[n]为整型数组，试写出实现下列运算的递归算法。

1）求数组a中的最大整数。

2）求n个整数的平均值。

1）分析：数组只有一个元素时，直接返回为最大值；

判断第n个元素与前n-1个元素的最大值的大小，返回较大值；前n-1个元素的最大值的求解就是递归

代码：

```
public int findMax(int[] a, int n) {
   //n表示第n个元素，对应数组a[n-1]
   if(n==1){
      return a [0];
   } else {
      int temp = findMax(a, n-1);
      return temp > a[n-1] ? temp : a[n-1];
   }
}
```

2）分析：数组只有一个元素时，该元素即为平均值

n个元素的平均值=[前n-1个元素的平均值 *（n-1）+第n个元素的值] / 个数 n

代码：

```
public double getAverage(int[] a, int n) {
```

```
        // n表示数组的第n个元素
        if (n==1)
            return a[0];
        else
            return (getAverage(a, n-1)*(n-1) + a[n-1]) / n;
    }
```

4. Write a recursive method that calculates and returns the length of a linked list.

分析： 链表节点总数=当前节点的下一个节点的长度+1

代码：

```
    public int length(ListNode a)
    {
        if(a==null)    return 0;
        else    return 1 + length(a.next);
    }
```

5. Check recursively if the following objects are palindromes:

a. A word

b. a sentence (ignoring blanks, lower- and uppercase differences, and punctuation marks so that "Madam, I m Adam" is accepted as a palindrome)

a） 分析：比较头尾两个字符，相同则向中间聚拢比较，不相同则返回false。

代码：

```
    public static boolean palindrome0(String word, int low, int high) {
        if (low > high)
            return true;
        if (word.charAt(low) == word.charAt(high)
        || Math.abs(word.charAt(low) - word.charAt(high)) == 32)
            return palindrome(word, low+1, high-1);
        else
            return false;
        }
```

b）分析：方法1：首先去除不符合要求的字符，将问题转化为a）的问题

方法2：在递归的过程中判断是否符合要求

代码：

```
public static boolean palindrome(String word, int low, int high) {
        if (low > high)
            return true;
        while (!Character.isLetter(word.charAt(low)))
            low++;
        while (!Character.isLetter(word.charAt(high)))
            high--;
        if (word.charAt(low) == word.charAt(high)
            || Math.abs(word.charAt(low) - word.charAt(high)) == 32)
            return palindrome(word, low+1, high-1);
        else
            return false;
    }
```

Chapter2

1. Find the complexity of the function used to find the kth smallest integer in an unordered array of integers

```
int selectkth ( int a[], int k, int n){
    int i, j, mini, temp;
    for ( i = 0; i < k; i++){
        mini = i;
        for ( j = i+1; j < n; j++)
            if ( a[j] < a[mini])
                mini = j;
        tmp = a[i];
        a[i] = a[mini];
        a[mini] = tmp;
    }
    return a[k-1];
}
```

分析：

$$\sum_{i=0}^{k-1}\left(1+\sum_{j=i+1}^{n-1}1+3\right)=kn-\frac{k^2}{2}+\frac{7}{2}k$$

2. Find the computational complexity for the following four loops:

```
c. for (cnt3=0, i =1; i<=n; i*=2)
       for (j=1; j<=n; j++)
           cnt3++;
d. for (cnt4=0, i=1; i<=n; i*=2)
       for (j=1; j<=i; j++)
           cnt4++;
```

分析：

c)

$$(Math.floor(\log_2 n)+1)*\sum_{1}^{n}1=n*\log_2 n+n=O(n*\ln n)$$

d)

$$t=Math.floor(\log_2 n)$$
$$2^0+2^1+\ldots+2^t=2^{t+1}-1=O(n)$$

3. For each of the following two program fragments:Give an analysis of the running time(Big-Oh will do)

```
1)  sum = 0;
    for( i = 0; i < n; i++ )
        for( j = 0; j <i*i; j++ )
            for( k = 0; k <j; k++ )
                sum++;
2)  sum = 0;
    for( i = 1; i < n; i++ )
        for( j = 0; j < i*i; j++ )
            if( j % i == 0 )
                for( k = 0; k < j; k++ )
                    sum++;
```

分析：

1)

$$\sum_{i=0}^{n-1}\sum_{j=0}^{i^2-1}\sum_{k=0}^{j-1}1=\sum_{i=0}^{n-1}\sum_{j=0}^{i^2-1}j=\sum_{i=0}^{n-1}\frac{i^4-i^2}{2}=O(i^5)$$

2)

$$\sum_{i=1}^{n-1}\left(\sum_{k=0}^{i-1}1+\sum_{k=0}^{2i-1}1+\ldots+\sum_{k=0}^{i(i-1)-1}1\right)=\sum_{i=1}^{n-1}(i+2i+\cdots+i(i-1))=\sum_{i=1}^{n-1}\frac{i^3-i^2}{2}=O(i^4)$$

4. 设n为正整数，分析下列各程序段中加下划线的语句的执行次数。

1)

```
for (int i = 1; i <= n; i++)
    for (int j = 1; j<=n; j++)
    {   c[i][j] = 0.0;
        for ( int k = 1; k <= n; k++)
            c[i][j] = c[i][j]+a[i][k]*b[k][j];
    }
```

分析：

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}1=n^3$$

2)

```
x = 0; y = 0;
for (int i = 1; i <= n; i++)
    for (int j = 1; j <= i; j++)
        for (int k = 1; k <= j; k++)
            x = x+y;
```

分析：

$$\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{j}1=\frac{n(n+1)(n+2)}{6}$$

$1^2+2^2+3^2+....+n^2=n(n+1)(2n+1)/6$

3) int x = 91; int y = 100;
while(y>0)
{ if(x>100) {
x -= 10; y--; //a
}
else
x++; //b
}

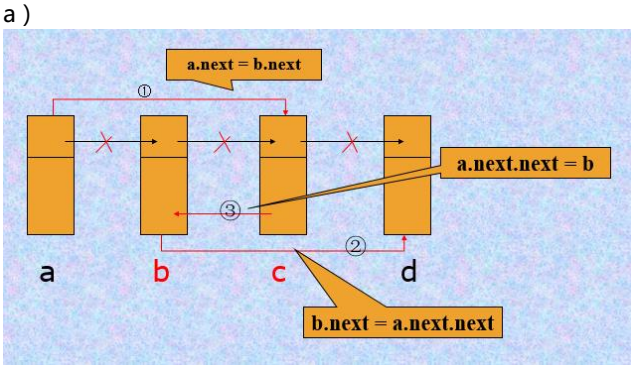
分析：b每执行10次，a执行1次
y初值为100，故 a执行100次，故b执行100*10=1000次

Chapter3.0

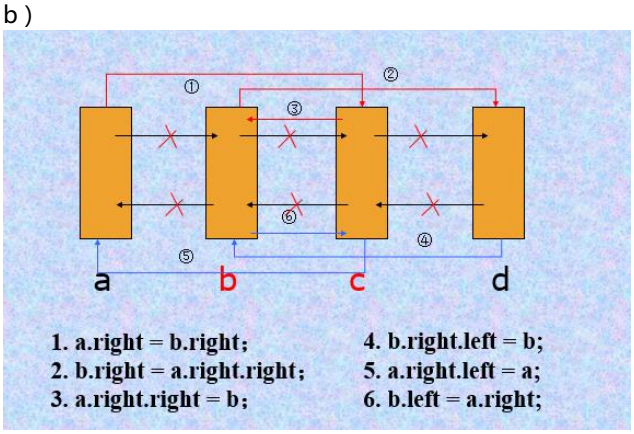
1. Swap two adjacent elements by adjusting only the links(and not the data) using:
a. Singly linked lists.
b. Doubly linked lists.

分析：

a)



b)



2. Given two sorted lists L 1 and L 2 ,write a procedure to compute L 1 ∩ L 2 using only the basic list operations.

分析：

1. 假设链表按照节点值递增排序
2. 新建一个结果链表，存放 L 1 ∩ L 2的结果
3. 循环遍历L1和L2，比较当前节点的值

- L1和L2节点中的值大小相等时，将该节点加入结果链表，L1和L2同时指向下一个节点；
- L1<L2节点中的值时，L1指向下一节点；
- L1>L2节点中的值时，L2指向下一节点；

4. L1或L2为空时，停止遍历，返回结果链表

代码：

```
public LinkedList intersection (LinkedList L1,LinkedList L2){
        LinkedListItr itr1 = L1.first();
        LinkedListItr itr2 = L2.first();
        LinkedList L = new LinkedList();
        LinkedListItr itr = L.zeroth();          //返回header

        while(!itr1.isPastEnd() && !itr2.isPastEnd()) {        //当前节点是否为空
         if(itr1.retrive() == itr2.retrive()) {   //若两个链表当前节点值相等
         L.insert(itr1.current,itr);
         itr1.advance();
         itr2.advance();
            }
         else if(itr1.retrive() < itr2.retrive())
                    itr1.advance();
         else
           itr2.advance();
        }
        return L;
}
```

3. Given two sorted lists, L 1 and L 2 , write a procedure to compute L 1 ∪ L 2 using only the basic list operations.

分析：

1. 假设链表按照节点值递增排序
2. 新建一个结果链表，存放 L 1 ∪ L 2的结果
3. 循环遍历L1和L2，比较当前节点的值
   - L1和L2节点中的值大小相等时，将该节点加入结果链表，L1和L2同时指向下一个节点；
   - L1<L2节点中的值时，将L1该节点加入结果链表，L1指向下一节点；
   - L1>L2节点中的值时，将L2该节点加入结果链表，L2指向下一节点；
4. L1为空时，将L2剩余节点加入结果链表

   L2为空时，将L1剩余节点加入结果链表
5. 返回结果链表

代码：

```
public LinkedList intersection (LinkedList L1,LinkedList L2){
      LinkedListItr itr1 = L1.first();
      LinkedListItr itr2 = L2.first();
      LinkedList L = new LinkedList();
      LinkedListItr itr = L.zeroth();   //返回header

      while(!itr1.isPastEnd() && !itr2.isPastEnd()){   //当前节点是否为空
            if(itr1.retrive() == itr2.retrive()){  //若两个链表当前节点值相等
                L.insert(itr1.current,itr);
                itr1.advance();
                itr2.advance();
           }
 else if(itr1.retrive() < itr2.retrive()) (){
                    L.insert(itr1.current,itr);
itr1.advance();
}
 else{
L.insert(itr2.current,itr);
itr2.advance();
     }
 }
      //如果L1没有结束，将L1剩余元素添加到L中
  for(;!itr1.isPastEnd();itr1.advance())
L.insert(itr1.current,itr);

     //如果L2没有结束，将L2剩余元素添加到L中
     for(;!itr2.isPastEnd();itr2.advance())
    L.insert(itr2.current,itr);
```

```
        return L;
    }
```

4. Write a nonrecursive method to reverse a singly linked List in O(N) time.

分析：

1. 从第一个节点开始，依次保存三个节点p1, p2, p3
2. 修改p2的下一个节点为p1
3. p1，p2，p3三个节点同时指向原链表的下一个节点
4. 循环至链表尾部
5. 头指针指向最后一个节点

代码：

```
public void reverse() {
        ListNode p1 = header.next;
        if (p1 == null)
                return ;
        ListNode p2 = p1.next;
        if (p2 == null)
                return ;
        ListNode p3 = p2.next;
        p1.next = null;

        while (p3 != null) {
                p2.next = p1;
                p1 = p2;
                p2 = p3;
                p3 = p3.next;
        }
        p2.next = p1;    //最后一个元素指向倒数第二个元素
        header.next = p2;   //将头指针指向最后一个元素
 }
```

Chapter3.1

1.2009年考研统考题:

1) 为解决计算机主机与打印机之间速度不匹配问题, 通常设置一个打印数据缓冲区,主机将要输出的数据依次写入该缓冲区, 而打印机则依次从该缓冲区中取出数据. 该缓冲区的逻辑结构应该是（B）

A. 栈 B. 队列 C. 树 D. 图

分析：

依次的意思就是先进先出，后进后出，所以选队列。

2) 设栈S和队列Q 的初始状态为空, 元素 a,b,c,d,e,f,g 依次进入栈S. 若每个元素出栈后立即进入队列Q, 且7个元素出队的顺序是b,d,c,f,e,a,g , 则栈S的容量至少是（C）

A. 1 B. 2 C. 3 D. 4

分析：

分析进栈出栈的过程即可得到结果。

2. Suppose that a singly list is implemented with both a header and tail node. Describe contant-time algorithms to

a. Insert item x before position p (given by an iterator).

b. Remove the item stored at position p (given by an iterator)

a）（***本题重点在于常数时间***）

分析：题目要求在给定p节点前插入新节点，考虑到单链表不能直接获取前一节点，从头遍历又不符合常数时间的要求。

因此，先将节点插入在p节点之后，再修改新节点和p的值，达到要求的效果

代码：

```
public void insert(LinkedListItr itr, Object x){
        ListNode p= itr.current;
        if(p!=head) {
            ListNode addnode=new ListNode(p.element, p.next);  //在p之后插入一新节点
            p.next=addnode;
            p.element=x;     //p指向节点的元素值用x替换
        }
    }
```

b)
分析：

- 若p为最后一个节点，直接将p赋值给tail节点，释放原tail节点
- p为中间节点，修改p节点的值为p.next的值，再删除p.next节点，达到删除p的效果

代码：

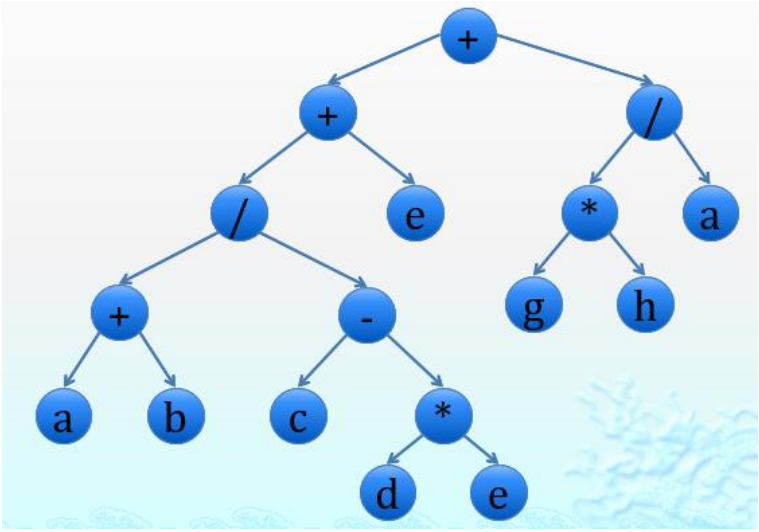
```
Public void remove(LinkedListItr itr){
        ListNode p=itr.current;
        if(p!=head && p!=tail){
            if(p.next==tail){  //如果p是最后一个节点
                tail=p;
                tail.next=null;
            }
            else{
              p.element=p.next.element;
               p.next=p.next.next;
            }
        }
 }
```

3. 假设以数组Q[m]存放循环队列中的元素，同时以rear和length 分别指示环形队列中的队尾位置和队列中所含元素的个数：
1 ）求队列中第一个元素的实际位置。
2 ）给出该循环队列的队空条件和队满条件，并写出相应的插入(enqueue)和删除(dlqueue)元素的操作算法。
1) front=(rear-length+1+m)%m
2 ）队空 : length==0
队满 : length==m

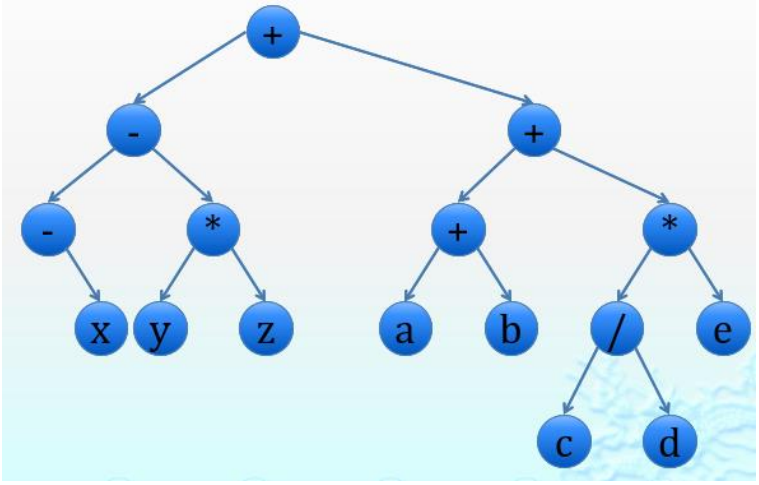
```
public void enqueue(Object x) throws Overflow{  /*入队*/
        if(isFull())
          throws new Overflow();
        else{
            length++;
            rear = (rear+1)%m;
            Q[rear] = x;
        }
}
public Object dequeue() throws Underflow{  /*出队*/
        if(isEmpty())
         throws new Underflow();
        else{
            length--;  //length--和front++是等价的，因为front满足1 ）中的表达式
            return Q[(rear-length+1+m)%m];
        }
}
```

Chapter4.0

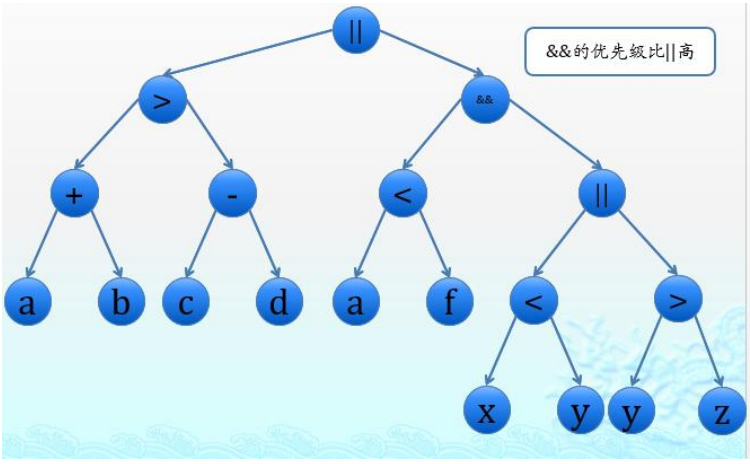
1. 给出如下各表达式的二叉树：
1) (a+b)/(c-d*e)+e+g*h/a
2) -x-y*z+(a+b+c/d*e)
3) ((a+b)>(c-d))|| a<f &&(x<y || y>z)
1)



2)



3)



2. 如果一棵树有n 1 个度为1 的结点，有n 2 个度为2 的结点，…… ，n m 个度为m 的结点，试问有多少个度为0 的结点？写出推导过程。

分析：

节点数=边数+1

$$n_0 + n_1 + n_2 + n_3 + \ ... + n_m = n_1 + 2n_2 + 3n_3 + \ ... + mn_m + 1$$

$$n_0 = n_2 + 2n_3 + 3n_4 + \ ... + (m-1)n_m + 1$$

3. 分别找出满足以下条件的所有二叉树：

1 ）二叉树的前序序列与中序序列相同

2 ）二叉树的中序序列与后序序列相同

3 ）二叉树的前序序列与后序序列相同

分析：

1 ）前序遍历 NLR 和中序遍历 LNR 恒等

=>所有节点都没有左节点、只有根节点、空树

2）中序遍历 LNR 和后序遍历 LRN 恒等

=>所有节点都没有右节点、只有根节点、空树

3）前序遍历 NLR 和后序遍历 LRN 恒等

=>只有根节点、空树

4. 若用二叉链表作为二叉树的存储表示，试对以下问题编写递归算法。

1 ）统计二叉树中叶结点的个数。

2 ）以二叉树为参数，交换每个结点的左子女和右子女

1）分析：二叉树的叶节点的个数=左子树的叶节点的个数+右子树叶节点的个数

代码：

```
public static int leafNum(BinaryNode root) {
    if (root == null)
        return 0;
    if (root.left == null && root.right == null)
        return 1;
    return leafNum(root.left) + leafNum(root.right);
}
```

2）分析：

- 交换根节点的左右子女
- 交换根节点的左子树的每个节点的左右子女
- 交换根节点的右子树的每个节点的左右子女

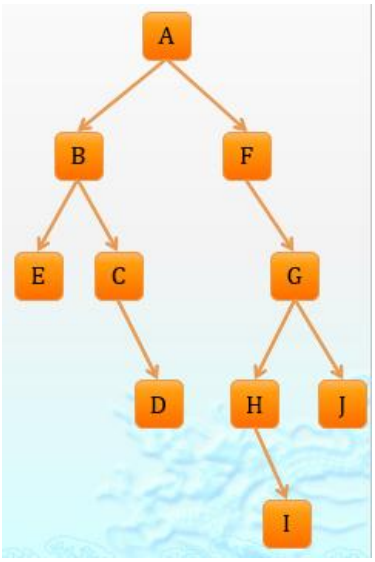
代码：

```
public static void switchLR(BinaryNode root) {
    if (root == null)
        return;
    BinaryNode tmp = root.left;
    root.left = root.right;
    root.right = tmp;
    switchLR(root.left);
    switchLR(root.right);
}
```

5. 是 已知一棵二叉树的先序遍列结果是 ABECDFGHIJ ，中序遍列结果是 EBCDAFHIGJ，试画出这棵二叉树。

分析：根据先序遍历得到根节点，结合中序遍历，得到节点关于根节点的左右分布情况；

对根节点的左右子树重复以上过程。



6. 编写一个Java 函数， 输入后缀表达式， 构造其二叉树表示。设每个操作符有一个或两个操作数。

分析： 遍历后缀表达式：

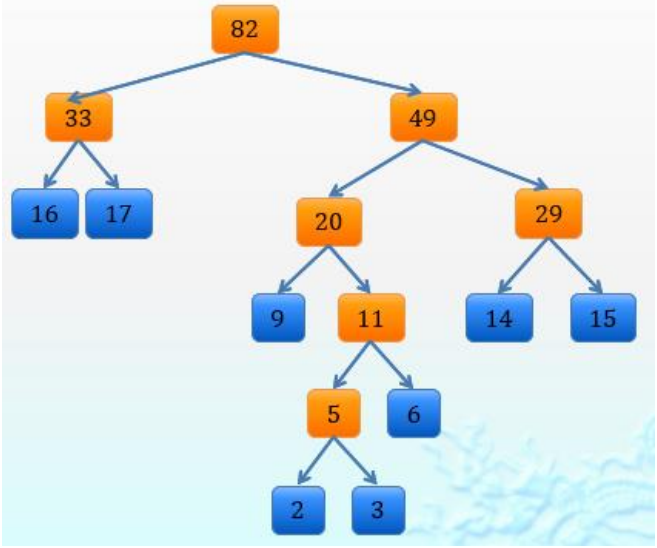
- 若为操作数，将其构造成节点后入栈；
- 若为一元操作符，弹出一个节点作为操作符的**右**子女构造新节点，将新节点入栈
- 若为二元操作数，弹出两个节点分别作为操作符的**右左**子女构造新的节点，将新节点入栈

代码：

```
public static BinaryNode makeTreeFromPostfixExpression(String expression) {
    Stack stack = new Stack ();
    expression = expression.replaceAll(" ", "");
    for (int i = 0; i < expression.length(); ++i) {
```

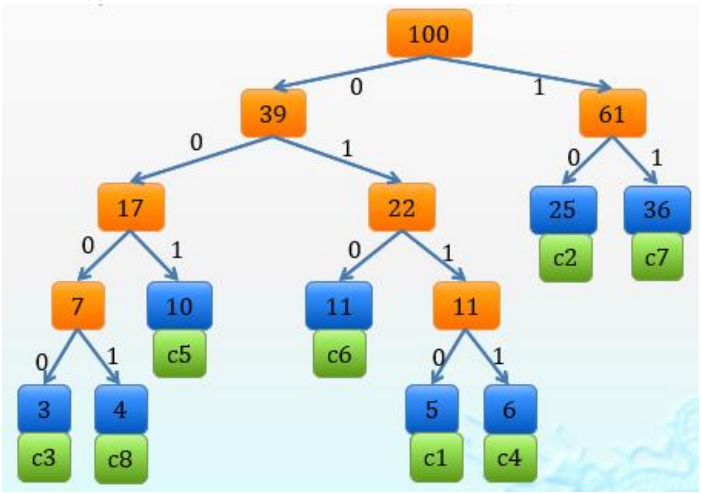
```
        char ch = expression.charAt(i);
        switch (ch) {
        case '+': case '-': case '*': case '/': case '^':
         BinaryNode right = (BinaryNode) stack.pop();
         BinaryNode left = (BinaryNode) stack.pop();
         stack.push(new BinaryNode(ch, left, right));
         break;
        case '~':
         BinaryNode node = (BinaryNode) stack.pop();
         stack.push(new BinaryNode(ch, null, node));
         break;
        default:
         stack.push(new BinaryNode(ch, null, null));
         break;
        }
       }
       return (BinaryNode) stack.pop();
      }
```

7. 给定权值{ 15, 03, 14, 02, 06, 09, 16, 17 }，构造相应的霍夫曼树，并计算它的带权外路径长度。
分析：从集合中取出最小的两个数作为结点，并将两者之和作为其父节点，同时在集合中用父节点代替这两个节点，重复之前的操作。



带权外路径长度：2 * 5 + 3 * 5 + 6 * 4 + 9 * 3 + 14 * 3 + 15 * 3 + 16 * 2 + 17 * 2 = 229

8. c1, c2, c3, c4, c5, c6, c7, c8 这八个字母的出现频率分别{ 5,25,3,6,10,11,36,4,} 为这八个字母设计不等长的Huffman 编码，并给出该电文的总码数.

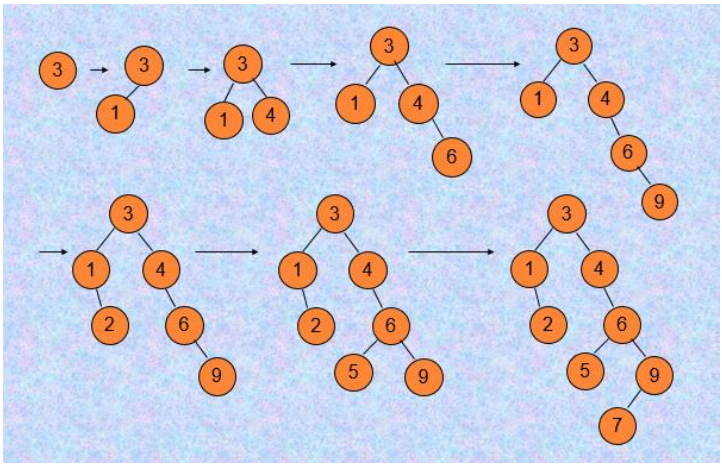


c1:0110 c2:10 c3:0010 c4:0111
c5:000 c6:010 c7:11 c8:0011
总码数即带权外路径长度：4*5 + 2*25 + 4*3 + 4*6 + 3*10 + 3*11 + 2*36 + 4*4 = 257

Chapter4.1

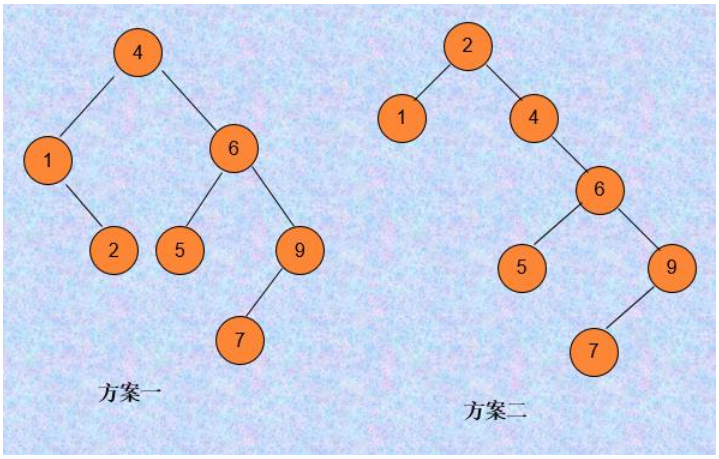
1. a. Show the result of inserting 3, 1, 4, 6, 9, 2, 5, 7 into an initially empty binary search tree.
   b. Show the result of deleting the root.

a)



b) 分析 :

- 方案一 : 用根节点的右子树中的最小值4替换根节点3，再删除原本的4节点
- 方案二 : 用根节点的左子树中的最大值2替换根节点3，再删除原本的2节点



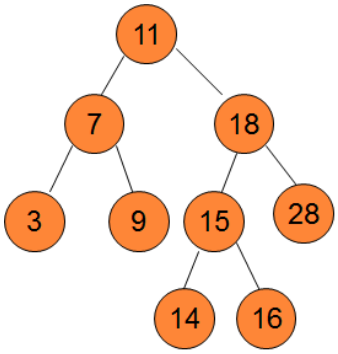
2. 写一递归函数实现在带索引的二叉搜索树 (IndexBST)中查找第k个小的元素。
分析 : 带索引的二叉搜索树的节点中的leftsize指定了左子树中的节点个数+1

- k==leftsize : 当前结点即为目标节点
- k<leftsize : 目标节点在当前结点的左子树中
- k>leftsize : 目标节点在当前结点的右子树中

代码 : public BinaryNode findkth(BinaryNode t,int k) {

```
 if(k<=0 || t==null)
                return null;
       if(k<t.leftsize)
           return findkth(t.left ,k);
       else if(k>t.leftsize)
           return findkth(t.right ,k- t.leftsize);
       else
        return t;
    }
```

3. 对一棵空的AVL树，分别画出插入关键码为{ 16，3，7，11，9，28，18，14，15}后的AVL树。
分析 : 依次插入节点，一旦发生不平衡，立刻调整为AVL树，直至全部节点插入完成。



4.设计算法检测一个二叉树是不是一个二叉搜索树.
分析：

1. 方法一：
    a. 对于二叉搜索树的每个节点来说，左子树中的节点均小于该节点，右子树中的节点均大于该节点
    b. 左子树也满足a
    c. 右子树满足a
2. 方法二：
    a. 中序遍历二叉搜索树；
    b. 遍历结果是递增排序的；

代码：
方法一：

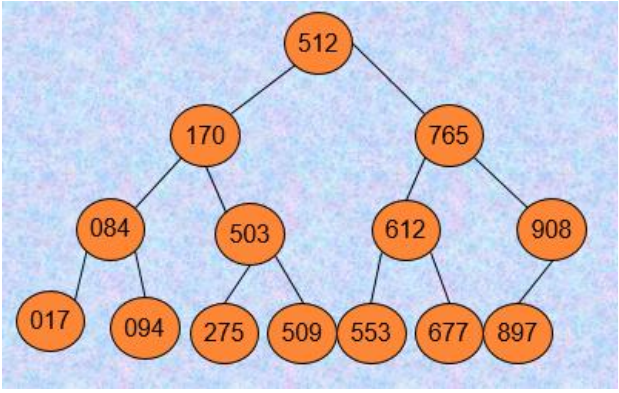
```
public boolean isBST(Node n) {
 if (n==null) return true;
 if (n.left!=null && max(n.left).element>n.element)
       return false;
 if (n.right!=null && min(n.right).element<n.element)
  return false;
 return isBST(n.left) && isBST(n.right);
}
public Node max (Node n) {
 if (n==null) return null;
 while (n.right!=null) { n=n.right; }
 return n;
}
public Node min (Node n) {
 if (n==null) return null;
 while (n.left!=null) { n=n.left; }
 return n;
}
```

方法二：

```
public boolean isBST(Node n) {
 If(n==null) return true;
 List l = inOrderList(n);
 int i,j;
 for(i=0,j=1;j<l.length();i++,j++) {
  if(l.get(i)>=l.get(j)) return false;
 }
 return true;
}
public List inOrderList(Node n) {
 List l = new ArrayList();
 inOrderList(n, l);
 return l;
}
public void inOrderList(Node n, List l) {
 If(n==null) return;
 inOrderList(n.left, l);
 l.add(n.element);
 inOrderList(n.right, l);
}
```

5. 设有序顺序表中的元素依次为 017,094,154,170,275,503,509,512,553,612,677,765,897,908. 试画出对其进行二分法搜索时的判定树,

并计算搜索成功的平均搜索长度。
分析：从根节点起，每次取中间位置的数作为节点内容，若中间有两个则取后一个。

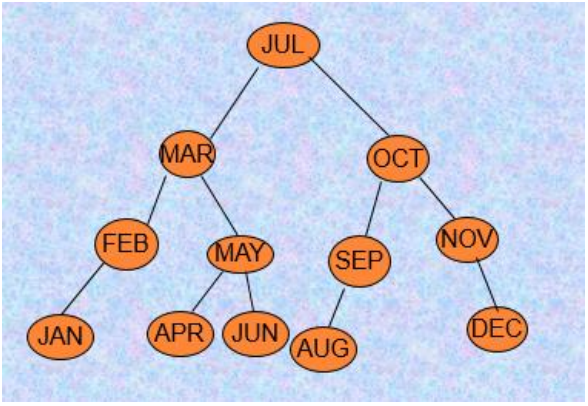


平均搜索长度为：(1+2*2+3*4+4*7)/14=45/14

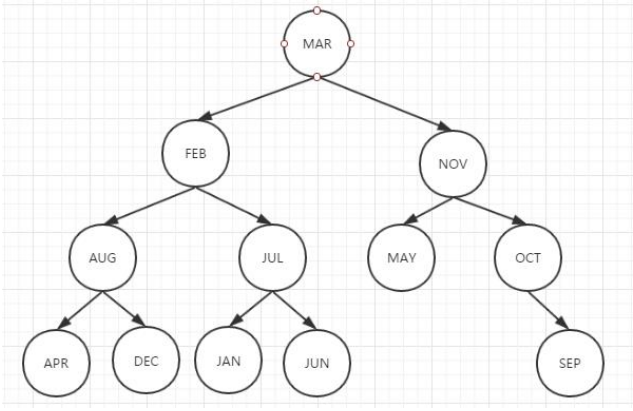
6.在一棵表示有序集S 的二叉搜索树中, 任意一条从根到叶结点的路径将S分为三部分: 在该结点左边结点中的元素组成集合S1; 在该路径上的结点中的元素组成集合S2; 在该路径右边结点中的元素组成集合S3, S=S1US2US3. 若对于任意的a S1, b S2, c S3, 是否总有a<=b<=c? 为什么?
分析：不一定。如上图，取路径512->765->612->677，取a=553，b=512。

7. 将关键码DEC, FEB, NOV, OCT, JUL, SEP, AUG, APR, MAR, MAY, JUN, JAN 依次插入到一棵初始为空的AVL 树中, 画出每插入一个关键码后的AVL 树, 并标明平衡旋转的类型.
分析：根据同学们的做题结果，有两种答案：

- 将12月份按照数字大小进行比较



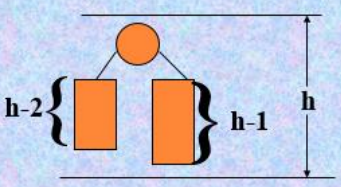
- 将12月份按照英文字典的顺序进行比较

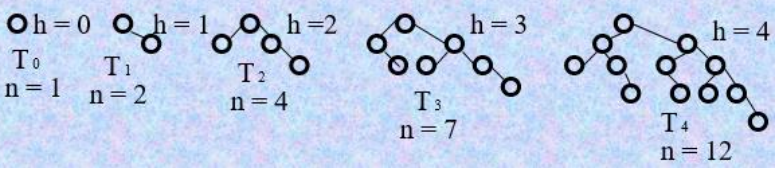


*8. 对于一个高度为h 的AVL 树, 其最少结点数是多少? 反之, 对于一个有n 个结点的AVL 树, 其最大高度是多少? 最小高度是多少?

设T_h为一棵高度为h，且结点个数最少的平衡二叉树。

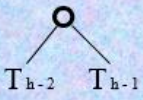
假设右子树高度为h-1，因结点个数最少，
∴左子树高度只能是h-2
这两棵左子树,右子树高度分别为h-2, h-1,也一定是结点数最少的:





以上五棵平衡二叉树，又称为Fibonacci树。

也可以这样说一棵高度为h的树，其右子树高度为h-1的Fibonacci树，左子树是高度为h-2的Fibonacci树，即



假设N_h表示一棵高度为h的Fibonacci树的结点个数，则

$N_h = N_{h-1} + N_{h-2} + 1$

$N_0 = 1, N_1 = 2, N_2 = 4, N_3 = 7, N_4 = 12, \ldots$

$N_0 + 1 = 2, N_1 + 1 = 3, N_2 + 1 = 5, N_3 + 1 = 8, N_4 + 1 = 13, \ldots$

∴ $N_h + 1$满足费波那契数的定义，并且$N_h + 1 = F_{h+3}$

f_0	f_1	f_2	f_3	f_4	f_5	f_6	...
0	1	1	2	3	5	8	...

费波那契数F_i满足下列公式

$$F_i = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^i - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^i$$

∵ $\left|\frac{1-\sqrt{5}}{2}\right| < 1$, ∴ $\frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^i$ 相当小

$$N_h + 1 = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{h+3} + O(1)$$

∵费波那契数树是具有相同高度的所有平衡二叉树中结点个数最少的

$$n + 1 \geq N_h + 1 = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{h+3} + 0(1)$$

对于一个有N 个结点的AVL 树，其最大高度是多少？ 最小高度是多少？

∵费波那契数树是具有相同高度的所有平衡二叉树中结点个数最少的，

$$n+1 \geqslant N_h+1=\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{h+3}+0(1)$$

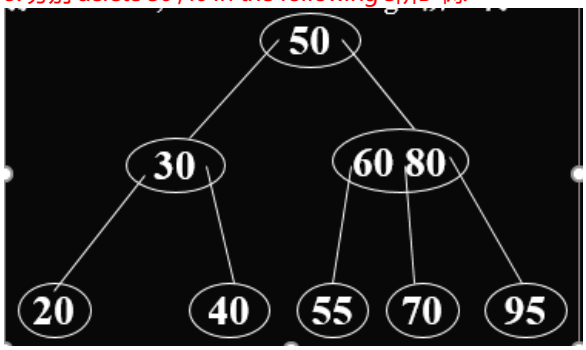
$$\therefore h \leqslant \frac{1}{\log_2 \frac{1+\sqrt{5}}{2}}\log_2(n+1)+0(1) \approx \frac{3}{2}\log_2(n+1)$$

这就是h的最大值；

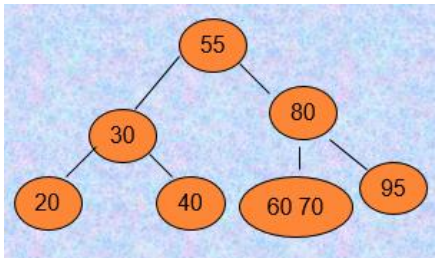
至于h的最小值，是一颗完全二叉树的时候，高度最小，有

$2^{h+1}-1=n$，所以$h=\log_2(n+1)-1$.

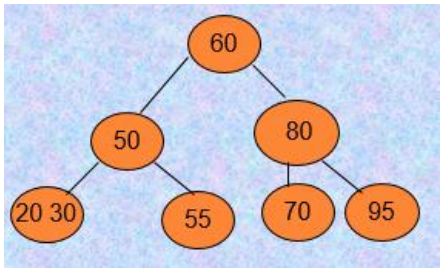
9. 分别 delete 50 ,40 in the following 3阶B-树.



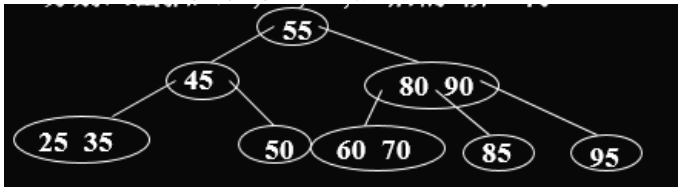
1. delete 50



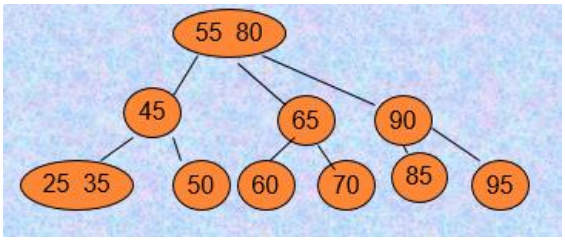
2. delete 40



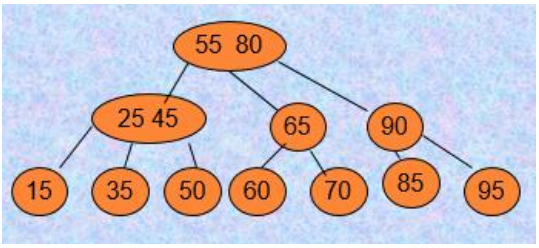
10. 分别画出插入65, 15, 40, 30后的3阶B-树。



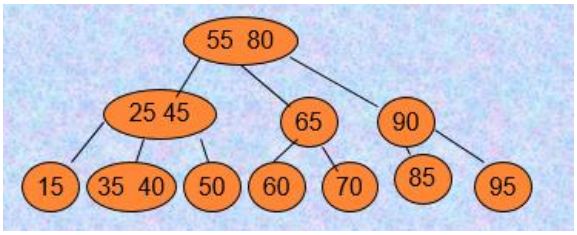
1. 插入65



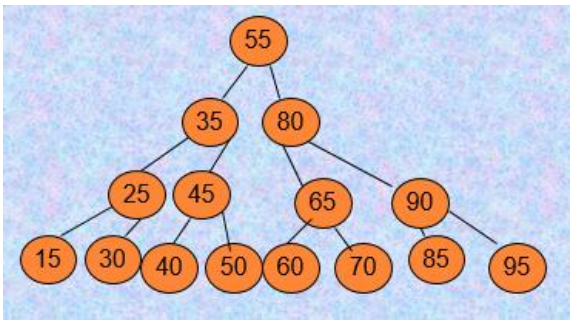
2. 插入15



3. 插入40



4. 插入30



Chapter 5

exercises:

1. Given input { 4371, 1323, 6173, 4199, 4344, 9679, 1989 }and a hash function h(x) = x (mod 10), show the resulting:
   a. Separate chaining hash table.
   b. Hash table using linear probing.
   c. Hash table using quadratic probing.
   d. Hash table with second hash function h2 (x) = 7- (x mod 7).

a) H(4371)=1, H(1323)=3, H(6173)=3, H(4199)=9, H(4344)=4, H(9679)=9, H(1989)=9

0	1	2	3	4	5	6	7	8	9
	4371		1323 6173	4344					4199 9679 1989

b)

0	1	2	3	4	5	6	7	8	9
9679	4371	1989	1323	6173	4344				4199

c)

0	1	2	3	4	5	6	7	8	9
9679	4371		1323	6173	4344			1989	4199

d） H1(6173)=3,H2(6173)=1,3+1=4;
H1(4344)=4,H2(4344)=3,4+3=7;
H1(9679)=9,H2(9679)=2,(9+2+2+2)%10=5;
H1(1989)=9,H2(1989)=6,(9+6)%10=5,(9+6*2)%10=1…(9+6*9)%10=3,
需要扩容，取比原表长大两倍的最小质数23，新的hash函数为H(x)=(x)mod23

0	1	2	3	4	5	6	7
	4173						
8	9	10	11	12	13	14	15
	6173		1989	1323	4199		
16	17	18	19	20	21	22	
			9679	4344			

2. 设散列表为HT[13]，散列函数为 H(key) = key % 13 。用线性开地址法解决冲突，对下列关键码序列12,23,45,57,20,03,78,31,15,36 ：

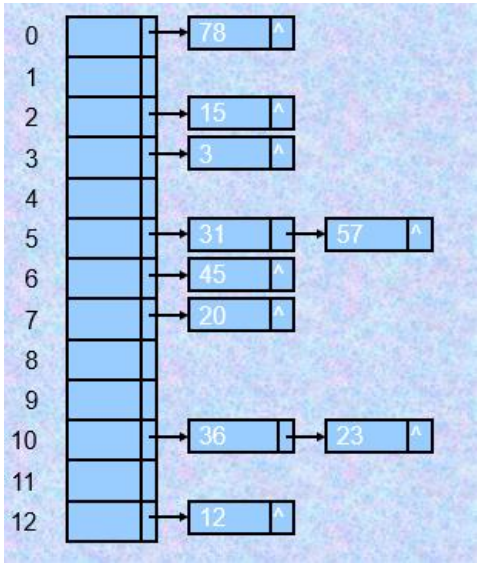
1) 画出其散列表。
2) 计算等概率下搜索成功的平均搜索长度。
3) 如果采用链表散列解决冲突，画出该链表。

1)

0	1	2	3	4	5	6
78		15	03		57	45
7	8	9	10	11	12	
20	31		23	36	12	

2) 平均搜索长度：（1+1+1+1+1+1+4+1+2+1）/ 10=1.4

3)



Chapter6

2009年统考题:

8. 已知关键字序列 5,8,12,19,28,20,15,22 是最小根堆(最小堆)，插入关键字3,调整后得到的小根堆是（A）

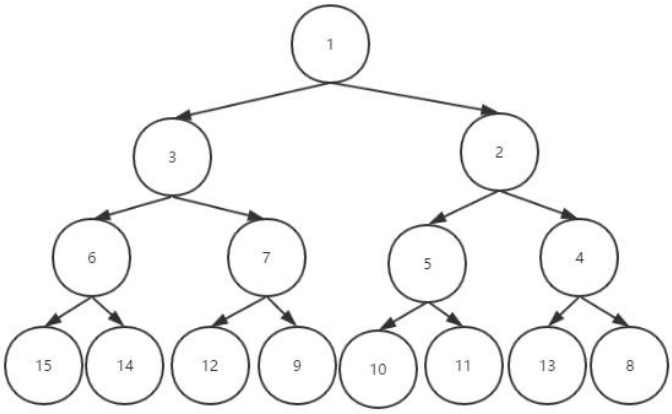
A. 3, 5, 12, 8, 28, 20, 15, 22, 19　　B. 3, 5, 12, 19, 20, 15, 22, 8, 28
C. 3, 8, 12, 5, 20, 15, 22, 28, 19　　D. 3, 12, 5, 8, 28, 20, 15, 22, 19

exercises:

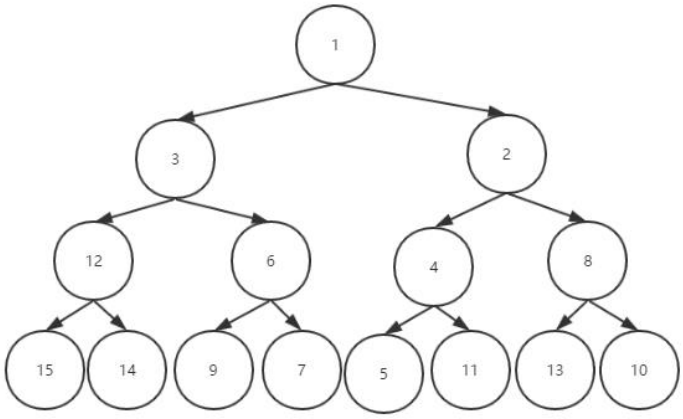
1. a. Show the result of inserting 10, 12, 1, 14, 6, 5, 8, 15, 3, 9, 7, 4, 11, 13, and 2,one at a time, into an initially empty binary heap.

b. Show the result of using the linear-time algorithm to build a binary heap using the same input.

a)



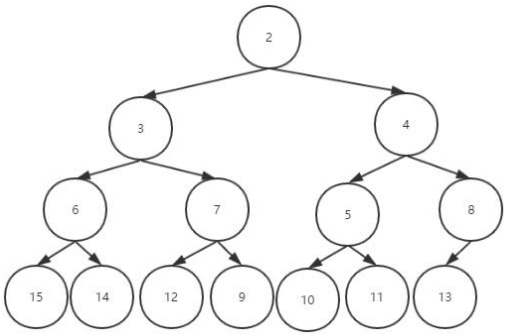
b)



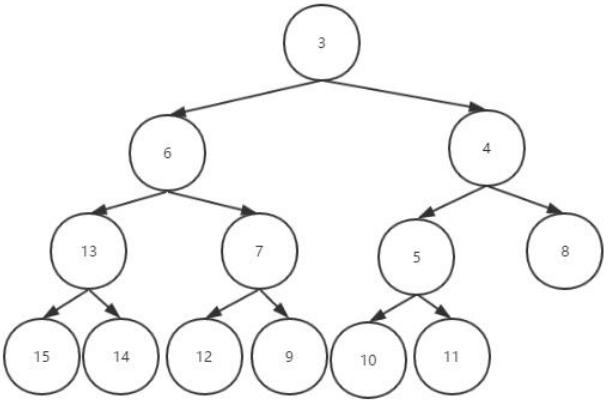
2. Show the result of performing three deleteMin operations in the heap of the previous exercise.

对于a)

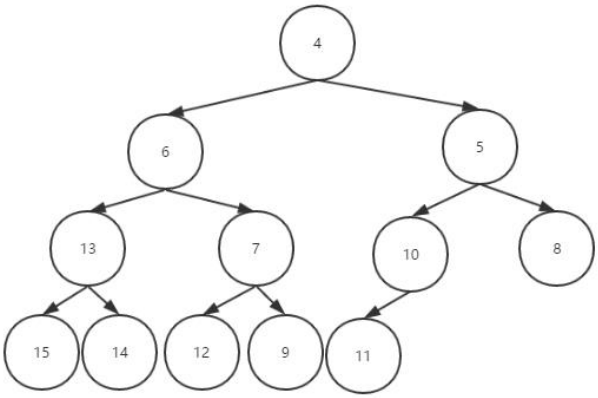
1. 删除1



2. 删除2

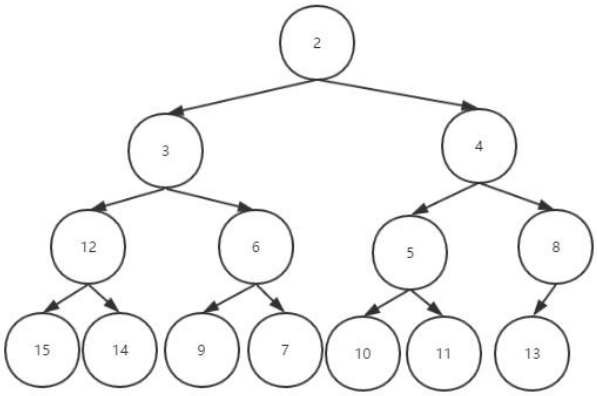


3. 删除3

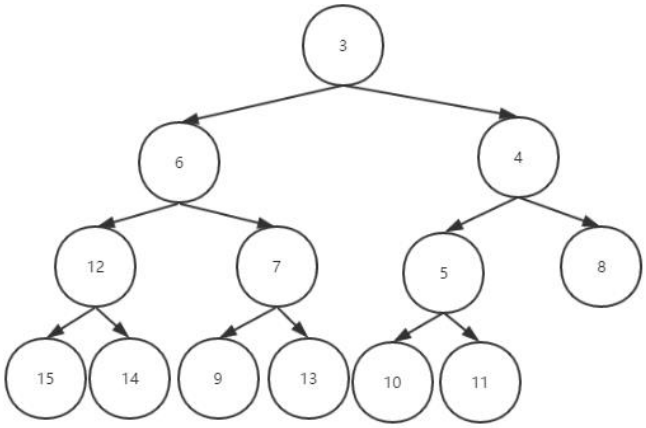


对于b）

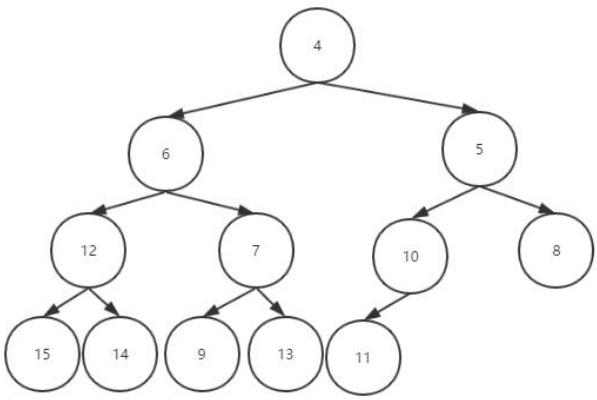
1. 删除1



2. 删除2



3. 删除3



3.判别以下序列是否是堆？如果不是，将它调整为堆。

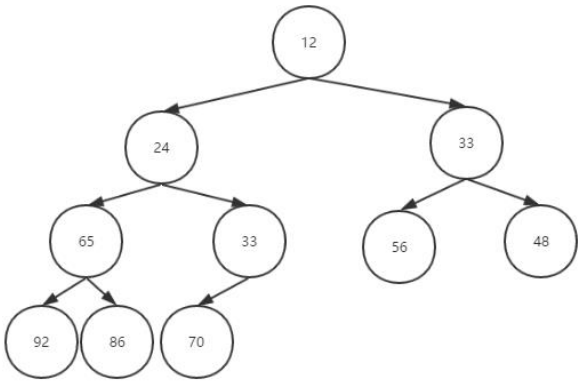
1) { 100, 86, 48, 73, 35, 39, 42, 57, 66, 21 }
2) { 12, 70, 33, 65, 24, 56, 48, 92, 86, 33 }

3) { 103, 97, 56, 38, 66, 23, 42, 12, 30, 52, 06, 20 }
4) { 05, 56, 20, 23, 40, 38, 29, 61, 35, 76, 28, 100 }

分析：

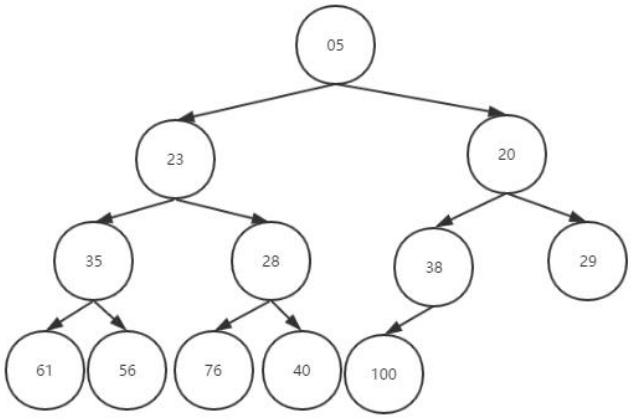
1) 是最大堆

2) 不是堆，调整为最小堆



3) 是最大堆

4) 不是堆，调整为最小堆



4.设待排序的关键码序列为{ 12, 2, 16, 30, 28, 10, 16*, 20, 6, 18 }, 使用堆排序方法进行排序。写出建立的初始堆，以及调整的每一步。

最后的输出排序为：2，6，10，12，16*，16，18，20，28，30

过程略。

Chapter 7

2009年统考题:

9. 若数据元素序列 11, 12, 13, 7, 8, 9, 23, 4, 5 是采用下列排序方法之一得到的第二趟排序后的结果, 则该排序算法只能是（B）

A. 起泡排序　　B. 插入排序　　C. 选择排序　　D. 二路归并排序

1. Sort the sequence 3, 1, 4, 1, 5, 9, 2, 6, 5 using insertion sort.

分析：每趟排序结果如下：

[1 3] 4 1 5 9 2 6 5
[1 3 4] 1 5 9 2 6 5
[1 1* 3 4]5 9 2 6 5
[1 1* 3 4 5] 9 2 6 5
[1 1* 3 4 5 9] 2 6 5
[1 1* 2 3 4 5 9] 6 5
[1 1* 2 3 4 5 6 9] 5
[1 1* 2 3 4 5 5* 6 9]

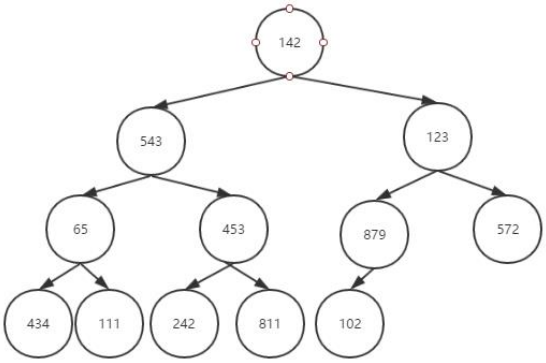
2. Show the result of running Shellsort on the input 9, 8, 7, 6, 5, 4, 3, 2, 1 using the increments { 1, 3, 7 }.

9 8 7 6 5 4 3 2 1
Gap=7 => 2 1 7 6 5 4 3 9 8
Gap=3 => 2 1 4 3 5 7 6 9 8
Gap=1 => 1 2 3 4 5 6 7 8 9

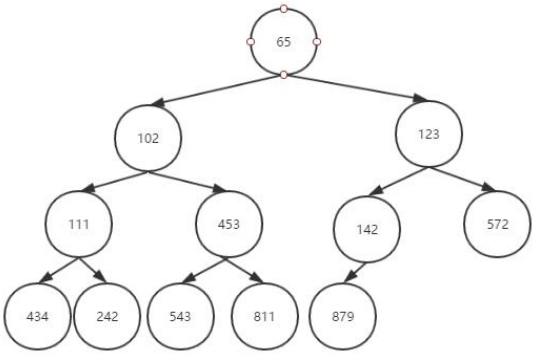
3. Show how heapsort processes the input 142, 543, 123, 65, 453, 879, 572, 434, 111, 242, 811, 102.

分析：构建初始堆，调整为最小堆（最大堆），输出最小值（最大值）后，重新调整为最小堆（最大堆），重复该过程。

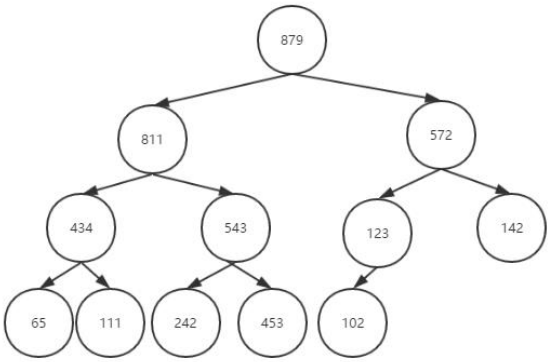
这里给出初始堆：



调整为最小堆：



调整为最大堆：



4. Rewrite heapsort so that it sorts only iterms that are in the range low to high which are passed as additional parameters.

分析：给定范围内的堆排序

根据初始数据，在指定范围内循环下沉，构造初始的最大堆；

依次将堆顶元素置出，重新调整，直至排序完成。

代码：

```
public void HeapSort_New(int[] a, int low, int high){
 for(int i = low+(high-low+1)/2; i >= low;i --)
  perDown(a, i, low, high);
 for(int i = high; i > low; i--){
  swap(a, low, i);
  perDown(a, low, low, i-1);
 }
}
private int leftChild(int i, int low){
 return 2*i+1 - low;
}
public void perDown(int[] a, int i, int low, int high){
 int child = leftChild(i, low);
 int temp = a[i];
 for(temp; leftChild(i, low)<high+1; i=child){
  child = leftChild(i, low);
  if(child != high && a[child] < a[child+1])
   child ++;
  if(a[child] > temp)
```

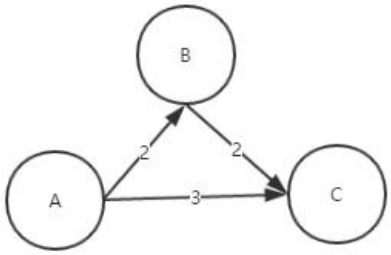
```
        a[i] = a[child];
       else
        break;
      }
      a[i] = temp;
     }
```

5. Sort 3, 1, 4, 1, 5, 9, 2, 6 using mergesort.
原始： 3 1 4 1* 5 9 2 6
P=1: [1 3] [1* 4] [5 9] [2 6]
P=2: [1 1* 3 4] [2 5 6 9]
P=3: [1 1* 2 3 4 5 6 9]

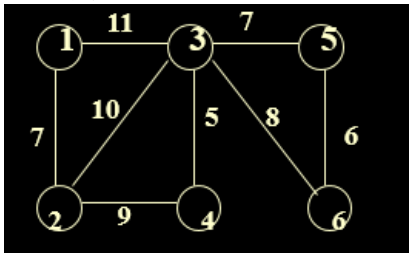
Chapter9

2009年统考题(单选题):
10. 下列关于无向连通图特性的叙述中, 正确的是（A）
a. 所有顶点的度之和为偶数 b. 边数大于顶点个数减1
c. 至少有一个顶点的度为 1
A. 只有a B. 只有b C. a和b D. a和c
分析：每条边必然属于两个顶点的入度和出度，所以入度和出度之和为边数的2倍，为偶数，所以a正确。

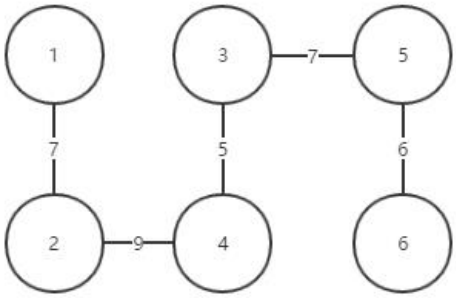
2009年统考题(综合应用题):
带权图（权值非负，表示边连接的两顶点间的距离）的最短路径问题是找出从初始顶点到目标顶点之间的一条最短路径. 假设从初始顶点到目标顶点之间存在路径，现有一种解决该问题的方法:
1) 设最短路径初始时仅包含初始顶点，令当前顶点u为初始顶点;
2) 选择离u 最近且尚未在最短路径中的一个顶点v，加入到最短路径中，修改当前顶点 u = v；
3) 重复步骤2)，直到u 是目标顶点时为止.
请问上述方法能否求得最短路径？ 若该方法可行，请证明之；否则，请举例说明.
分析：不一定。反例: 如下图，A到C的最短路径为3，按照上述方法得到路径ABC，长度为4。



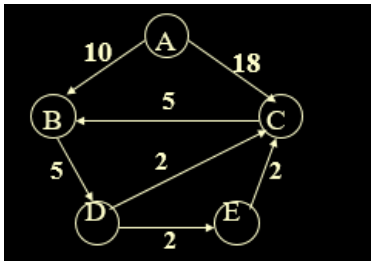
1. 对下列无向图：



分别用Prim算法与Kruscal算法，求出最小代价生成树（要求写出构造生成树的每一步）。
结果：两种算法结果是一样的。



2. 对下列有向图：



用Dijkstra算法求从顶点A到其它各顶点的最短路径。
分析：

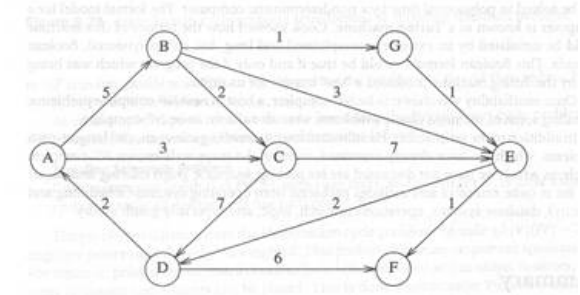
A	0		
B	10 A->B		
C	18 A->C	18 A->C	17 A->B->D->C
D	∞ A->D	15 A->B->D	
E	∞ A->E	∞ A->E	17 A->B->D->E

路径数组：

A				
B	A	0	0	0
C	A	A	D	D
D	-1	B	0	0
E	-1	-1	D	0

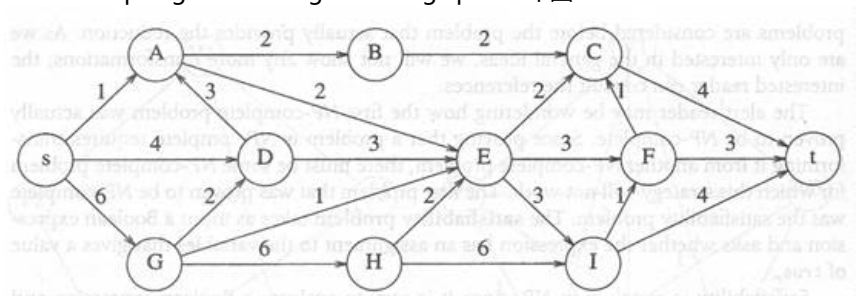
最短路径：
A -> B　10
A -> B -> D -> C　17
A -> B -> D　15
A -> B -> D -> E　17

3. a. Find the shorest path from A to all other vertices for the grath in 下图
b. Find the shortest unweighted path from B to all other vertices for the graph in 下图



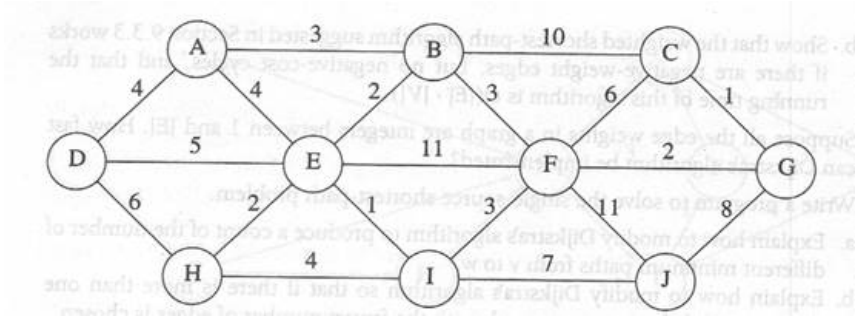
分析：
a) A到各点路径及路径长度：
A->B 5
A->C 3
A->B->G 6
A->B->G->E->D 9
A->B->G->E 7
A->B->G->E->F 8
b) B到各点的路径以及路径长度为：
B->C->D->A(或B->E->D->A)
B->C
B->C->D (或B->E->D)
B->E
B->E->F
B->G

4. Find a topological ordering for the graph in 下图

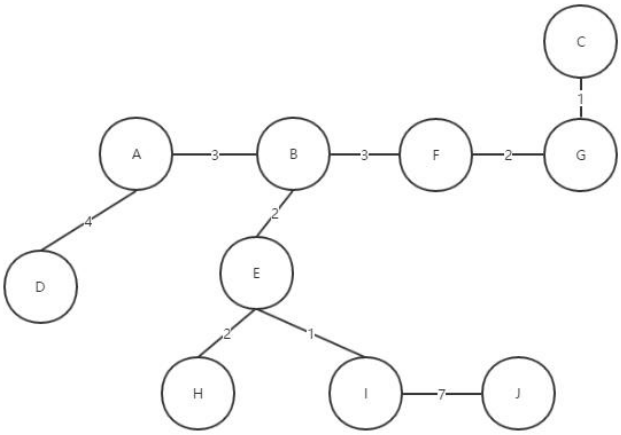


分析：排序结果不唯一
S->G->D->A->B->H->E->I->F->C->T
S->G->D->H->A->B->E->I->F->C->T

*5. a. Find a minimum spanning tree for the graph in 下图 using both Prim’s and Kruskal’s algorithms.
b. Is this minimum spanning tree unique? Why?



分析：
a) 两种算法得到的都可以是下图所示的结果



b) 不是唯一的，因为存在多条权值相同的路径。

注意：
1) 看清题目要求

2) 有些题目需要有过程，不能直接给出答案

3) 字迹清晰

4) 以上答案仅供参考