# DS

**第三章 栈和队列**

**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;**

**}**

**第三章 栈和队列**

**方法1:检验左子树的最大值和右子树的最小值**

**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;**

**}**

**第三章 栈和队列**

**class BinaryNode{**

**int data, degree;**

**BinaryNode firstchild;**

**BinaryNode nextsibling;**

**}**

**public void degree(BinaryNode t){**

**if(t==null) return;**

**t.degree=0;**

**if(t.firstchild!=null){**

**t.degree++;**

**BinaryNode p=t.firstchild;**

**while(p.nextsibling!=null){**

**t.degree++;**

**p=p.nextsibling;**

**}**

**degree(t.firstchild);**

**degree(t.nextsibling);**

**}**

**第三章 栈和队列**

**Private static void percUp(Comparable[ ] a , int start) {**

**int j=start , i=j/2;**

**comparable temp = a[j];**

**while(j>1) {**

**if(a[i]<=temp) break;**

**else {**

a[j]=**a[i];**

**j=i;**

**i=i/2;**

**}**

**}**

**a[j] = temp;**

**}**

**除了03——07年的重点算法外：**

**13年的备用卷：利用二叉搜索树实现优先队列。**

**Class BinaryNode {**

**BinaryNode left;**

**BinaryNode right;**

**Comparable data;**

**}**

**Public class BST {**

**private BinaryNode root;**

**public BST( ) { root=null;}**

**public BinaryNode insert(BinaryNode t , comparable x) {**

**if(t==null)**

**t=new BinaryNode(x, null, null);**

**else if(x.compareTo(t.data)<0)**

**t.left = insert(t.left , x);**

**else if(x.compareTo(t.data)>0)**

**t.right = insert(t.right , x);**

**return t;**

**}**

**private BinaryNode findMax( BinaryNode t )**

**{ if( t != null )**

**while( t.right != null )**

**t = t.right;**

**return t;**

**}**

**private BinaryNode remove( Comparable x, BinaryNode t )**

**{ if( t = = null )**

**return t;**

**if( x.compareTo( t.element ) < 0 )**

**t.left = remove( x, t.left );**

**else if( x.compareTo( t.element ) > 0 )**

**t.right = remove( x, t.right );**

**else if( t.left != null && t.right != null )**

**{ t.element = findMin( t.right ).element;**

**t.right = remove( t.element , t.right );**

**}**

**else**

**t = ( t.left != null ) ? t.left : t.right;**

**}**

**public BinaryNode delete() {**

**BinaryNode temp = findMax(root);**

**remove(temp.data , root);**

**}**

**已知先序序列和中序序列建立二叉树算法**

**void CreateBT(String pres, ins ; BinaryNode <Type>\* & t)**

**{ int inpos;**

**String prestemp, instemp ;**

**if (pres.length( )= =0) t=NULL;**

**else { t=new BinaryNode;**

**t->element=pres.ch[0]; inpos=0;**

**while (ins.ch[inpos]!=t->element) inpos++;**

**prestemp=pres(1,inpos);**

**instemp=ins(0,inpos-1);**

**CreateBT(prestemp, instemp, t->left);**

**prestemp=pres(inpos+1, pres.length( )-1);**

**instemp=ins(inpos+1, pres.length( )-1);**

**CreateBT(prestemp, instemp, t->right);**

**}**

**}**

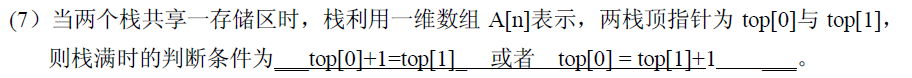
**此外还应该有二叉树先序，中序非递归算法**

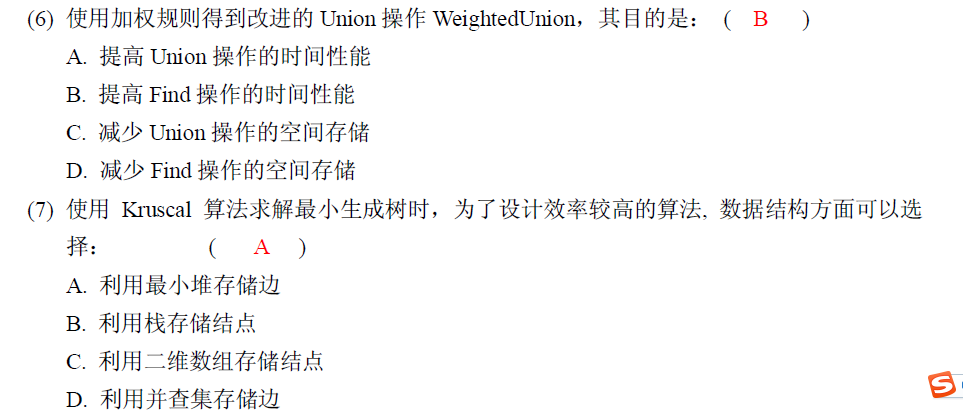
4)设W为一个二维数组，其每个数据元素占用6个字节，行下标i从0到8, 列下标j从0到3, 则二维数组W的数据元素共占用\_\_\_\_\_\_个字节。W中第6行的元素和第4列的元素共占用\_\_\_\_\_\_个字节。若按行主顺序存放二维数组W, 其起始地址为100, 则二维数组W的最后一个数据元素的起始地址为\_\_\_\_\_\_。

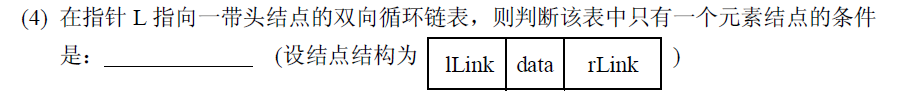
答案：216 72 310

(2) 有4个数据依次入栈，有 14 种出栈序列。

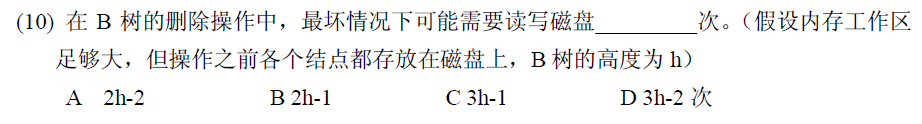
C_n = \frac{1}{n+1}{2n \choose n} = \frac{(2n)!}{(n+1)!n!}







L->rLink->rLink= =L && L->rLink!=L;



D

# 操作系统

1. 静态重定位的时机是\_\_\_\_.

程序链接时

程序编译时

程序装入时

程序运行时

13. 在请求分页存储管理中，当访问的页面不在内存时，便产生缺页中断，缺页中断是属于\_\_\_\_。

程序中断

访管中断

I/O中断

外中断

15. LRU置换算法所基于的思想是\_\_\_\_。

在最近的过去用得少的在最近的将来也用得少

在最近的过去很久未使用的在最近的将来也不会使用

在最近的过去很久未使用的在最近的将来会使用

在最近的过去用得多的在最近的将来也用得多

15. 进程资源静态分配方式是指一个进程在建立时就分配了它需要的全部资源，只有该进程所要资源都得到满足的条件下，进程才开始运行。这样可以预防进程死锁。静态分配方式破坏死锁的\_\_\_\_必要条件。

非剥夺式等待条件

循环等待条件

互斥条件

占有且等待

16. 银行家算法通过破坏\_\_\_\_来避免死锁。

不可抢占条件

部分分配条件

互斥条件

循环等待条件

20. 资源的按序分配策略可以破坏\_\_\_\_\_条件。

占有且等待资源

循环等待资源

互斥使用资源

非剥夺资源

1. Linux系统中的slab分配器，采用\_\_\_\_内存管理方式。

A．固定分区 B.分页式 C.伙伴系统 D.分段式

1. 实模式下16位CPU使用段偏移方式的寻址能力为\_\_\_\_\_\_\_\_\_\_。

A. 64kb B. 1M C. 16M D. 4G

1. 下面哪条指令不是从实模式进入保护模式需要的指令\_\_\_\_\_\_\_\_\_\_。

A. lgdt [GdtPtr] B. out 92h, al C. jmp $ D. mov cr0, eax

1. FAT12文件系统里，FAT表的数量和每张FAT表占用的扇区数量为\_\_\_\_\_\_\_\_\_。

A. 2, 9 B. 2, 10 C. 3, 9 D. 3, 10

1. 操作系统里没有下面哪种描述符表\_\_\_\_\_\_\_\_\_。

A. GDT B. LDT C. IDT D. KDT

（Global Descriptor Table） (Local Descriptor Table) （Interupt Descriptor Table）

1. C语言里面调用汇编的函数方法为\_\_\_\_\_\_\_\_\_。

A. C代码中使用extern声明，汇编中使用global导出

B. C代码中使用global声明，汇编中使用extern导出

C. C代码中使用extern声明，汇编中使用extern导出

D. C代码中使用global声明，汇编中使用global导出

1. 在Orange’S系统里，loader的作用不包括\_\_\_\_\_\_\_\_\_\_。

A. 引导扇区的启动 B. 跳入保护模式

C. 加载kernel并将执行转交kernel D. 启动分页机制

8. 请画出经典的五状态进程模型及其状态转换图。

8. 请画出经典的七状态进程模型及其状态转换图。

1. 叙述操作系统三个最基础的抽象，为什么要引入它们？