2006年《数据结构》期终考试试卷（A）

班级 学号 姓名

一、简答题（每小题6分，共30分）

(1) 假设一个线性链表的类名为linkedList，链表结点的类名为ListNode，它包含两个数据成员data和link。data存储该结点的数据，link是链接指针。下面给定一段递归打印一个链表中所有结点中数据的算法：

**void** PrintList (ListNode \*L) **{**

**if** ( L != NULL ) **{**

**cout** << L->data << **endl;**

PrintList ( L->link )**;**

**}**

**}**

试问此程序在什么情况下不实用？给出具体修改后的可实用的程序？

(1) **此程序在内存容量不足时不适用。因为需要一个递归工作栈。**当链表越长，递归工作栈的深度越深，需要的存储越多。可采用非递归算法节省存储。

**void** PrintList (ListNode \*L) **{**

**while** ( L != NULL ) **{**

**cout** << L->data << **endl;**

L = L->link**;**

**}**

**}**

(2) 如果每个结点占用2个磁盘块因而需要2次磁盘访问才能实现读写，那么在一棵有n个关键码的2m阶B树中，每次搜索需要的最大磁盘访问次数是多少？

(2) 在2m阶B树中关键码个数n与B树高度h之间的关系为 h≤log m ((n+1)/2)+1，那么**每次搜索最大磁盘访问次数为2hmax = 2log m ((n+1)/2)+2。**

(3) 给定一棵保存有n个关键码的m阶B树。从某一非叶结点中删除一个关键码需要的最大磁盘访问次数是多少？

(3) 在m阶B树中关键码个数n与B树最大高度h的关系为h = log⎡m/2⎤((n+1)/2)+1。若设寻找被删关键码所在非叶结点读盘次数为h’，被删关键码是结点中的ki，则从该结点的pi出发沿最左链到叶结点的读盘次数为h-h’。当把问题转化为删除叶结点的k0时，可能会引起结点的调整或合并。极端情况是从叶结点到根结点的路径上所有结点都要调整，除根结点外每一层读入1个兄弟结点，写出2个结点，根结点写出1个结点，假设内存有足够空间，搜索时读入的盘块仍然保存在内存，则结点调整时共读写盘3(h-1)+1。总共的磁盘访问次数为

**h’+(h-h’)+3(h-1)+1 = 4h-2 = 4(log⎡m/2⎤((n+1)/2)+1)-2 =**

**= 4log⎡m/2⎤((n+1)/2)+2**

(4) 给定一个有n个数据元素的序列，各元素的值随机分布。若要将该序列的数据调整成为一个堆，那么需要执行的数据比较次数最多是多少？

(4) 设堆的高度为h = ⎡log2(n+1)⎤，当每次调用siftDown算法时都要从子树的根结点调整到叶结点，假设某子树的根在第i层（1≤i≤h-1），第h层的叶结点不参加比较。从子树根结点到叶结点需要比较h-i层，每层需要2次比较：横向在两个子女里选一个，再纵向做父子结点的比较。因此，**在堆中总的比较次数为**



因为 2h-1≤n≤2h-1，且，则

(5) 设有两个分别有n个数据元素的有序表，现要对它们进行两路归并，生成一个有2n个数据元素的有序表。试问最大数据比较次数是多少？最少数据比较次数是多少？

(5) 两个长度为n的有序表，当其中一个有序表的数据全部都小于另一个有序表的数据时，**关键码的比较次数达到最小（= n）**。而当两个有序表的数据交错排列时，**关键码的比较次数达到最大（= 2n-1）**。

二、简作题（每小题5分，共15分）

针对如下的带权无向图

E

F

G

A

B

C

H

I

J

D

e2**:** 4

e3**:** 4

e8**:**4

e4**:**5

e6**:** 6

e12**:** 6

e17**:** 7

e19**:** 8

e10**:** 10

e14**:** 11

e18**:** 11

e1**:** 3

e11**:** 3

e16**:**3

e7**:** 2

e15**:** 2

e5**:** 2

e9**:** 1

e13**:** 1

其中，每条边上所注的*ei*为该边的编号，冒号后面是该边所对应的权值。

(1) 使用Prim算法，从顶点A出发求出上图的最小生成树。要求给出生成树构造过程中依次选择出来的边的序列（用边的编号表示），权值相等时编号小的边优先。（不必画图）

(2) 使用Kruskal算法求出上图的最小生成树。要求给出生成树构造过程中依次选择出来的边的序列（用边的编号表示），权值相等时编号小的边优先。（不必画图）

(3) 上面求出的最小生成树是唯一的吗？试举理由说明。

1. 使用Prim算法

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| e1 | e5 | e9 | e7 | e11 | e15 | e13 | e2 | e17 |
| 3 | 2 | 1 | 2 | 3 | 2 | 1 | 4 | 7 |

E

F

G

A

B

C

H

I

J

D

e4**:**5

e2**:** 4

e3**:** 4

e8**:**4

e6**:** 6

e12**:** 6

e17**:** 7

e19**:** 8

e10**:** 10

e14**:** 11

e18**:** 11

e1**:** 3

e11**:** 3

e16**:**3

e7**:** 2

e15**:** 2

e5**:** 2

e9**:** 1

e13**:** 1

(2) 使用Kruskal算法

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| e9 | e13 | e5 | e7 | e15 | e1 | e11 | e2 | e17 |
| 1 | 1 | 2 | 2 | 2 | 3 | 3 | 4 | 7 |

E

F

G

A

B

C

H

I

J

D

e4**:**5

e2**:** 4

e3**:** 4

e8**:**4

e6**:** 6

e12**:** 6

e17**:** 7

e19**:** 8

e10**:** 10

e14**:** 11

e18**:** 11

e1**:** 3

e11**:** 3

e16**:**3

e7**:** 2

e15**:** 2

e5**:** 2

e9**:** 1

e13**:** 1

(3) 这样**选取的最小生成树是唯一的**。因为在边上的权值相等时先选编号小的，限定了选择的机会。假如不限定在具有相等权值的边中的选择次序，结果可能就可能不唯一了。

三、简作题（共10分）

假设一个散列表中已装入100个表项并采用线性探查法解决冲突，要求搜索到表中已有表项时的平均搜索次数不超过4，插入表中没有的表项时找到插入位置的平均探查次数不超过50.5。请根据上述要求确定散列表的容量，并用除留余数法设计相应的散列函数。



三、简作题（共10分）



由前一式得到，由后一式得到，综合得

因n = 100，有，,

可取m = 117。用除留余数法设计散列函数：

Hash(key) = key % 113 (注：117不是质数，117 = 9 \* 13)

四、算法设计题（每小题5分，共15分）

设中序线索化二叉树的类声明如下：

**template <class Type>**

**struct** ThreadNode **{** //中序线索化二叉树的结点类

**int** leftThread, rightThread**;** //线索标志

ThreadNode**<Type>** \*leftChild, \*rightChild**;** //线索或子女指针

**Type** data**;** //结点中所包含的数据

**};**

**template <class Type>**

**class** inOrderThreadTree **{** //中序线索化二叉树类

**public:**

ThreadNode**<Type>** \* getRoot ( ) **{ return** root**; }**

//其他公共成员函数

**……**

**private:**

ThreadNode**<Type>** \*root**;** //树的根指针

**};**

试依据上述类声明，分别编写下面的函数。

(1) ThreadNode**<Type>** \* getPreorderFirst (ThreadNode**<Type>** \*p)**;**

//寻找以p为根指针的中序线索化二叉树在前序下的第一个结点。

(2) ThreadNode**<Type>** \* getPreorderNext (ThreadNode**<Type>** \*p)

//寻找结点\*p的在中序线索化二叉树中前序下的后继结点。

(3) **void** preorder (inOrderThreadTree**<Type>&** T)**;**

//应用以上两个操作，在中序线索化二叉树上做前序遍历。

四、算法设计题（每小题5分，共15分）

**(1) tamplate <class Type>**

ThreadNode**<Type>** \* getPreorderFirst (ThreadNode**<Type>** \*p) **{**

**return** p**;**

**}**

**(2) template <class Type>**

ThreadNode**<Type>** \* getPreorderNext (ThreadNode**<Type>** \*p) **{**

**if** ( p->leftThread *==* 0 ) **return** p->leftChild**;**

**if** (p->rightThread *==* 0 ) **return** p->rightChild**;**

**while** ( p->rightThread != 0 **&&** p->rightChild != NULL )

p = p->rightChild**;**

**return** p->rightChild**;**

**}**

1. **template <class Type>**

**void** preorder ( inOrderThreadTree**<Type>&** T ) {

ThreadNode**<Type>** \*p **=** getRoot()**;**

p = getPreorderFirst ( p )**;**

**while** ( p != NULL ) **{**

**cout** << p->data << **endl;**

p =getPreorderNext ( p )**;**

**}**

**}**

五、算法分析题（每小题5分，共15分）

下面给出一个排序算法，其中n是数组A[ ]中元素总数。

**template<class Type>**

**void** unknown (**Type** a[ ]**, int** n) **{**

**int** d = 1, j**;**

**while** ( d < n /3 ) d = 3\*d+1**;**

**while** ( d > 0 ) **{**

**for** ( **int** i = d**;** i < n**;** i++ ) **{**

**Type** temp = a[i]**;**

j = i**;**

**while** ( j >= d && a[j-d] > temp ) **{** a[j] = a[j-d]**;** j -= d**; }**

a[j] = temp**;**

**}**

d /= 3**;**

**}**

**}**

(1) 阅读此算法，说明它的功能。

(2) 对于下面给出的整数数组，追踪第一趟**while** ( d > 0 ) 内的每次**for**循环结束时数组中数据的变化。（为清楚起见，本次循环未涉及的不移动的数据可以不写出，每行仅写出一个**for**循环的变化）

(3) 以上各次循环的数据移动次数分别是多少。

五、算法分析题（每小题5分，共15分）

(1) **希尔排序**

(2) 第一趟while循环内各for循环结束时数组中数据的变化：

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 步 | a[0] | a[1] | a[2] | a[3] | a[4] | a[5] | a[6] | a[7] | a[8] | a[9] | 移动次数 |
|  | 77 | 44 | 99 | 66 | 33 | 55 | 88 | 22 | 44 | 11 |  |
| 1 | 33 |  |  |  | 77 |  |  |  |  |  | 3 |
| 2 |  | 44 |  |  |  | 55 |  |  |  |  | 2 |
| 3 |  |  | 88 |  |  |  | 99 |  |  |  | 3 |
| 4 |  |  |  | 22 |  |  |  | 66 |  |  | 3 |
| 5 |  |  |  |  | 44 |  |  |  | 77 |  | 3 |
| 6 |  | 11 |  |  |  | 44 |  |  |  | 55 | 4 |

(3) 各趟数据移动次数见表的最右一栏。

六、算法设计题（每小题5分，共15分）

下面是队列和栈的类声明：

**template <class** **Type**> **class** queue **{**

**public:**

queue ( )**;** //队列的构造函数

queue (**const** queue**&** qu)**;** //队列的复制构造函数

queue**& operator**= (**const** queue**&** qu)**;** //赋值操作

**bool** isEmpty ( )**;** //判断队列空否，=1为空, =0不空

**Type&** getFront ( )**;**  //返回队头元素的值

**void** push (**const** **Type&** item)**;** //将新元素插入到队列的队尾

**void** pop ( )**;** //从队列的队头删除元素

//…… //其他成员函数

**}**

**template <class** **Type**> **class** stack **{**

**public:**

stack ( )**;** //栈的构造函数

bool isEmpty ( )**;** //判断栈空否。=1栈空，=0不空

**void** push ( **const** stack**&** item )**;** //将新元素进栈

**void** pop ( )**;** //栈顶元素退栈

**Type&** getTop ( )**;** //返回栈顶元素的值

**}**

试利用栈和队列的成员函数，编写以下针对队列的函数的实现代码（要求非递归实现）。

(1) “逆转”函数 **template <class** **Type**> **void** reverse (queue<**Type**>**&** Q)**;** （5分）

(2) “判等”函数 **bool** queue**::operator***==* (**const** queue**&** Q)**;**（5分）

(3) “清空”函数 **void** queue**::**clear ( )**;** （5分）

六、算法设计题（每小题5分，共15分）

1. **#include** “stack”

**template <class** **Type**>

**void** reverse (queue<**Type**>**&** Q) **{** //普通函数

stack <**Type**> S**; Type** tmp**;**

**while** ( !Q.isEmpty() )

**{** tmp = Q.getFront()**;** Q.Pop()**;** S.Push (tmp)**; }**

**while** ( !S.isEmpty() )

**{** tmp = S.getTop()**;**  S.Pop()**;** Q.EnQueue()**; }**

**};**

(2) **bool** queue**::operator***==* (**const** queue**&** Q) **{** //成员函数

queue<**Type**> Q1, Q2**; Type** t1**,** t2**; bool** finished = true**;**

**while** ( !is.Empty() ) **{**

t1 = getFront()**;** Pop()**;** Q1.Push(t1)**; //**从左队列退出, 进临时队列

t2 = Q.getFront()**;** Q.Pop()**;** Q2.Push(t2)**; //**从右队列退出, 进临时队列

**if** ( t1 != t2 ) **{** finished = false**; break; }**

**}**

**while** ( !Q1.isEmpty() ) **{** t1 = Q1.getFront()**;** Q1.Pop()**;** Push(t1)**; }**

**while** ( !Q.isEmpty() ) **{** t2 = Q.getFront()**;** Q.Pop()**;** Q2.Push(t2)**; }**

**while** ( !Q2.isEmpty() ) **{** t2 = Q2.getFront()**;** Q2.Pop()**;** Q.Push(t2)**; }**

**}**

(3) **void** queue**::**clear ( ) **{** //成员函数

**while** ( !isEmpty() ) Pop()**;**

**};**