部分课后习题参考答案：

**第一章习题**

**2．判断题：**

|  |  |  |
| --- | --- | --- |
| **（1）** | **（2）** | **（3）** |
| **×** | **×** | **√** |

**3．填空题：**

**（1）包含改变量定义的最小范围**

**（2）数据抽象、信息隐蔽**

**（3）数据对象、对象间的关系、一组处理数据的操作**

**（4）指针类型**

**（5）集合结构、线性结构、树形结构、图状结构**

**（6）顺序存储、非顺序存储**

**（7）一对一、一对多、多对多**

**（8）一系列的操作**

**（9）有限性、输入、可行性**

**4．选择题：**

|  |  |  |
| --- | --- | --- |
| **（1）** | **（2）** | **（3）** |
| **A** | **C** | **D** |

**5．语句频度为：**

**1+（1+2）+（1+2+3）+…+（1+2+3+…+n）**

存在问题：

（１）语句频度！＝算法的时间复杂度

**=1（1+1）/2+2（2+1）/2+3（3+1）/2+…+n（n+1）/2**

**=1/2[（1+2+3+…+n）+（12+22+32+…+n2）]**

**=1/2[n（n+1）/2+ n（n+1）（2n+1）/6]**

**=n（n+1）（n+2）/6**

**6．编写算法以计算在给定各系数和变量x的值时的多项式fn(x)的值，要求时间尽可能少。**

**利用秦九韶式算法计算多项式：**

存在问题：

（１）算法！＝程序

（２）算法的前提条件（参数）

（３）算法的处理结果（返回值）

（４）只写算法的主要代码，不完整

****

**n为多项式最高次幂；x为变量；a[n]为按系数组成的数组；**

**double MultiC(int n, double x, int a[]){**

**double result=a[n];**

**for(i=n-1;i>=0;i--) result= a[i] + result\*x;**

**return result;  
} /\* 算法的时间复杂度：T(n)=O(n) \*/**

**对比以下代码：**

**for(i=1,t=1,result=a[0];i<=n;i++) { t=t\*x; result= result+a[i]\*t; }**

**通常算法的输入和输出可采用下列两种方式之一：**

**（1）通过参数表中的参数显示传递。**

**（2）通过全局变量隐式传递。**

**各自优缺点如下：**

**（1）通过参数表中的参数显示传递：算法表达清晰，不调用时不占内存，函数通用性强；形参、实参须一一对应，返回值最多只有一个。**

**（2）通过全局变量隐式传递：全局只用一个变量表示某个数据，减少了形参实参数据传递的时间消耗；但长期占用内存，函数通用性低。**

**7．参数设置规则：在算法实现时，只引用数据不改变其值的量应设为传值方式，而需要通过算法改变其值的量应作为传地址方式。**

**8．函数结果带出方式种类：**

**（1）函数返回值**

**（2）变参（指针类型）**

**（3）全局变量**

**问题：**

* **有同学描述的算法，就若干行代码，并没有以函数的形式来表示这个算法。**
* **算法处理的结果没有明确的输出或返回**
* **有部分同学上交的是两个多项式相加。**

第二章 习题

1．填空题：

（1）一半，插入、删除的位置

（2）显示，隐式

（3）一定，不一定

（4）头指针，头结点的指针域，其前驱的指针域

2．选择题：

（1）A

（2） ① D A

② G H D A

③ E L

④ I A F

（3）D

（4）D

（5）C

3．头指针：指向整个链表首地址的指针，标示着整个单链表的开始。

头结点：为了操作方便，可以在单链表的第一个结点之前附设一个结点，该结点的数据域可以存储一些关于线性表长度的附加信息，也可以什么都不存。

首元素结点：线性表中的第一个元素结点成为首元素结点。

4．已知顺序表L递增有序。试写一算法，将X插入到L的适当位置上，以保持顺序表L的有序性。

分析:表满异常的处理，寻找插入的位置

int Insert(SeqList \*L, ElemType X) {

int i=0,k;

if(L->last>=MAXSIZE-1) { printf(“表已满无法插入”)； return(0); }

while(i<=L->last&&L->elem[i]<X) i++;

for(k=L->last;k>=i;k--) L->elem[k+1]=L->elem[k];

L->elem[i]=X;

L->last++;

return(1);

}

5．写一算法，从顺序表中删除自第i个元素开始的k个元素。

分析：i要满足i>=1, i<=L->last+1

k要满足 k>=1, i+k-1<= L->last+1

int Del(Seqlist \*L, int i, int k) {

int j;

if(i<1||(i+k)>(L->last+2))

{ printf(“输入的i，k值不合法”); return 0; }

if((i+k)==(L->last+2)) { L->last=i-2; ruturn 1; }

else {

for(j=i+k-1;j<=L->last;j++) elem[j-k]=elem[j];

L->last=L->last-k;

return 1;

}

}

6．已知长度为n的线性表A采用顺序存储结构，请写一时间复杂度为O(n)，空间复杂度为O(1)的算法，删除线性表中所有值为item的数据元素。

分析：要求时间复杂度为O(n)，空间复杂度为O(1)

只能循环一次

int Delet1 (Seqlist \*A, ElemType item) {

int i,j=0;

for(i=0;i<=L-last;i++)

if(L->elem[i]!=item)) { L->elem[j]= L->elem[i]; j++;}

L->last=j-1;

}

或

从第一个元素开始向后寻找值等于item的元素

找到一个就用表尾的一个值不等于item的元素覆盖

int Delet2 (Seqlist \*L, ElemType item) {

int i,j;

for(i=0,j=L->last;i<=j;i++)

if(L->elem[i]==item)) {

while(L->elem[j]==item&&j>=0) { j--; L->last--; }

if(i<j) { L->elem[i]= L->elem[j]; j--; L->last--; }

}

}

7. 编写算法，在一顺序表L中，删除所有值相等的多余元素。要求时间复杂度为O(n)，空间复杂度为O(1)。

分析：

顺序表 or 有序的顺序表？

8. 已知线性表中的元素（整数）以值递增有序排列，并以单链表作存储结构。试写一高效算法，删除表中所有大于mink且小于maxk的元素（若表中存在这样的元素），分析你的算法的时间复杂度（注意：mink和maxk是给定的两个参变量，它们的值为任意的整数）。

分析：

mink和maxk的合法性：mink < maxk；

不要一个一个的删除（多次修改next域）。

int delete(LinkList L, int mink, int maxk){

if(mink>=maxk) { printf(“参数出错”); return 0; }

pre=L; p=L->next;

while(p!=NULL&&p->data<=mink) { pre=p; p=p->next; }

s=p;

while(s!=NULL&&s->data<maxk) { t =s; s=s->next; free(t); }

pre->next=s;

return 1;

}

9. 就地逆置

（1）顺序存储

int inverse1(SeqList \*L) {

int i,j;

if(L->last==-1) { printf(“Empty.”); return 0; }

for(i=0,j=L->last;i<j;i++,j--)

{ temp=L->elem[i]; L->elem[i]=L->elem[j]; L->elem[j]=temp; }

return 1;

}

（2）链式存储

int inverse2(LinkList L) {

if(L->next==NULL|| L->next->next==NULL||) return 0;

p=L->next->next;

L->next ->next=NULL;

while(p) { temp=p->next; p=L->next; L->next=p; p=temp; }

return 1;

}

10. 算法分析：

首先找到链表中的第1个元素结点和第k个元素结点；

如果第k个元素结点不存在，return 0;

如果第k个元素结点存在，指向第1个元素结点和第k个元素结点的指针同时向后扫描，知道其中一个指针指向表尾。

算法的时间复杂度：O(n)

int Locate(LinkList list, int k) {

int j=0;

pre=p=list;

while(p&&j<k) { p=p->link; j++; }

if(!p) return 0;

pre=pre->link;

while(p->next) { pre=pre->next; p=p->next; }

printf(pre->data);

return 1;

}

**第三章 习题**

**1．B**

**2．C**

**3．C**

**4．（1） 如进站的车厢序列为123，则可能得到的出站车厢序列为：123,132,213,231,321。**

**（2）如进站的车厢序列为123456，则：**

**不能得到435612序列，1入，2入，3入， 4入，4出，3出，5入，5出，6入，6出，1无法先于2出站。**

**能够得到135426序列，1入，1出，2入，3入，3出，4入，5入，5出，4出，2出，6入，6出。**

**5．（1）**

**6．（1）顺序栈：**

**判空：S.top==-1**

**判满：S.top==MaxSize-1**

**（2）链栈：**

**判空：top->next==NULL**

**判满：通常不存在栈满的情况**

**7. 运算符的优先顺序： ↑ → \*,/ → +,-**

**运算数栈OPND, 运算符栈OPTR，下图中的^表示幂运算（↑）**

**栈中的表达式表示完成一次运算后，运算的结果入栈**

**（1）A-B\*C/D+E↑F**



**（2）A/B↑C+D\*E**



**8.设计算法，将一个书写形式正确的中缀表达式转换成逆波兰表达式。**

**分析：**

**（1）栈中数据元素为字符串；**

**（2）函数In(char c,char \*OPSet)：判断字符c是否在集合OPSet中，返回值为1或0；**

**（3）运算符的优先级别:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **+** | **-** | **\*** | **/** | **（** | **）** |
| **+** | **>** | **>** | **<** | **<** | **<** | **>** |
| **-** | **>** | **>** | **<** | **<** | **<** | **>** |
| **\*** | **>** | **>** | **>** | **>** | **<** | **>** |
| **/** | **>** | **>** | **>** | **>** | **<** | **>** |
| **（** | **<** | **<** | **<** | **<** | **<** | **==** |
| **）** | **>** | **>** | **>** | **>** |  | **>** |

**（4）表达式GetTop(OPTR1)<\*s1表示栈顶运算符的优先级别低于\*s1运算符；表达式GetTop(OPTR1)>\*s1表示栈顶运算符的优先级别高于\*s1运算符；表达式GetTop(OPTR1)==\*s1表示栈顶运算符的优先级别等于于\*s1运算符（比如“（”与“）”）。**

**void TransExpression(char \*s1,char \*s2) {**

**Stack OPND, OPTR1, OPTR2;**

**char temp[M],sum[M];**

**InitStack(&OPND); InitStack(&OPTR1); InitStack(&OPTR2);**

**while(\*s1 ||!Empty(OPTR1)) {**

**if(!In(\*s1,OPSet))**

**{ temp[0]=\*s1; temp[1]=’\0’; Push(&OPND,temp); s1++; }**

**else if(Empty(OPTR1)||GetTop(OPTR1)<\*s1)**

**{ temp[0]=\*s1; temp[1]=’\0’; Push(&OPTR1,temp); s1++; }**

**else if(\*s1==’\0’||GetTop(OPTR1)>\*s1) {**

**Pop(&OPTR1,temp); Push(&OPTR2,temp);**

**for(i=1;i<=2;i++)**

**{ Pop(&OPND,temp); Push(&OPTR2, temp); }**

**for(i=1;i<=3;i++)**

**{ Pop(&OPTR2,temp); strcat(sum,temp); }**

**Push(&OPND,sum);**

**}**

**else if(GetTop(OPTR1)==\*s1) /\*当括号相遇时\*/**

**{ Pop(&OPTR1,temp); s1++; }**

**}**

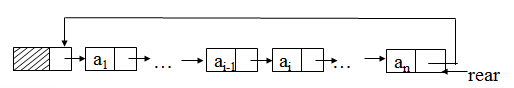
**Pop(&OPND,temp);**

**strcpy(s2,temp);**

**}**

**9.假设以带头结点的循环链表表示队列，并且只设一个指针指向队尾元素结点（注意不设头指针），试编写相应的队列初始化、入队列和出队列的算法。**

**分析：队列的结构示意图如下：**



**typedef struct QueueNode{**

**ElemType data;**

**struct QueueNode \* next;**

**} QueueNode, \* LinkQueue;**

1. **初始化**

**int InitQueue(LinkQueue \* rear){**

**\*rear=( QueueNode \*)malloc(sizeof(QueueNode));**

**if(!(\*rear)) return ERROR;**

**(\*rear)->next=\*rear;**

**return OK;**

**}**

1. **入队列**

**int InSQueue(LinkQueue rear, ElemType e){**

**QueueNode \* p;**

**p=( QueueNode \*)malloc(sizeof(QueueNode));**

**if(!(\*p)) return ERROR;**

**p->data=e;**

**p->next=rear->next;**

**rear->next=p;**

**rear=p;**

**return OK;**

**}**

1. **出队列**

**int DelQueue(LinkQueue rear, ElemType \* e){**

**QueueNode \* p;**

**if(rear->next==rear) { printf(“空队列”); return ERROR; }**

**p=rear->next->next;**

**\*e=p->data;**

**rear->next->next=p->next;**

**if(p==rear) /\*若删除的是队列中唯一的结点，修改rear指针的值\*/**

**rear=rear->next;**

**free(p);**

**return OK;**

**}**

**第四章 习题**

**9．编写一个函数将顺序串S1中的第i个字符到第j个字符之间的字符用s2替换。**

**int Replace(SString \*s1, SString \* s2, int i, int j){**

**dis= s2->len-(j-i+1);**

**if(s1->len+s2->len-(j-i+1)>MaxSize)**

**{ printf(“串长溢出”); return ERROR; }**

**if(dis>0)**

**for(k=s1->len-1;k>=j;k--) s1->ch[k+dis]=s1->ch[k];)**

**else if(dis<0)**

**for(k=j;k<=s1->len-1;k++) s1->ch[k+dis]=s1->ch[k];**

**for(k=0;k<=s2->len-1;k++) s->ch[k+i-1]=s2->ch[k];**

**return OK;**

**}**

**11．编写算法，实现顺序串的基本操作StrReplace(&s,t,v)。**

**int StrReplace1(SString \*s, SString \* t, SString \*v){**

**int i=0,j,k,cnt=0;**

**while(i<s->len){**

**k=Index(s,i,t);**

**if(k){**

**cnt++;**

**dis= v->len-t->len;**

**if(s->len+dis>MaxSize)**

**{ printf(“串长溢出”); return ERROR; }**

**if(dis>0)**

**for(j=s->len-1;j>=k+t->len-1;j--) s->ch[j+dis]=s->ch[j];**

**else if(dis<0)**

**for(j=k+t->len-1;j<=s->len-1;j++) s->ch[j+dis]=s->ch[j];**

**for(j=0;j<=v->len-1;j++) s->ch[j+k-1]=v->ch[j];**

**i=k+v->len-1;**

**}**

**else break;**

**}**

**if(cnt==0) printf(“主串中不存在要找的子串”);**

**return OK;**

**}**

**int StrReplacec(SString \*s, SString \* t, SString \*v){**

**int i=0,j=0;**

**int pos[M];**

**while(i<s->len){**

**k=Index(s,i,t);**

**if(k){ pos[j]=k; j++; i=pos[j]+t->len-1;}**

**else break;**

**}**

**if(j==0) { printf(“主串中不存在要找的子串”); return OK; }**

**else if(s->len+(v->len-t->len)\*j>MaxSize)**

**{ printf(“串长溢出”); return ERROR; }**

**else {**

**……**

**}**

**}**

**12．主串为aaabbbababaabb，子串为abaa，分别用Bruce-Force算法和KMP算法给出其匹配过程，并讨论模式匹配的匹配效率。**

**（1）Bruce-Force算法**

**第一趟：**

**a a**

**a b**

**第二趟：**

**a a a**

**a a b**

**第三趟：**

**a a a b b**

**a a a b a**

**第四趟：**

**a a a b**

**a a a a**

**第五趟：**

**a a a b b**

**a a a b a**

**第六趟：**

**a a a b b b**

**a a a b b a**

**第七趟：**

**a a a b b b a b a b**

**a a a b b b a b a a**

**第八趟：**

**a a a b b b a b**

**a a a b b b a a**

**第九趟：**

**a a a b b b a b a b a a**

**a a a b b b a b a b a b**

**匹配效率:为了找到子串（模式串），比较了19次。**

**（2）KMP算法**

**模式串的next[j]值：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **j** | **0** | **1** | **2** | **3** |
| **串** | **a** | **b** | **a** | **a** |
| **next[j]** | **-1** | **0** | **0** | **1** |

**第一趟：**

**a a**

**a b（j=next[1]=0）**

**第二趟：**

**a a a**

**a a b（j=next[1]=0）**

**第三趟：**

**a a a b b**

**a a a b a（j=next[2]=0）**

**第四趟：**

**a a a b b**

**a a b a a（j=next[1]=-1，i++，j++）**

**第五趟：**

**a a a b b b**

**a a bb b a（j=next[1]=-1，i++，j++）**

**第六趟：**

**a a a b b b a b a b**

**a a a b b b a b a a（j=next[3]=1）**

**第七趟：**

**a a a b b b a b a b a a**

**a a a b b b a b a b a a**

**匹配效率:为了找到子串（模式串），比较了16次。**

**第五章 习题**

**2．**

**void shift(ElemType A[],int len,int k){**

**for(n=1;n<=k;n++) {**

**temp=A[len];**

**for(i=len-1;i>=1;i--) A[i+1]=A[i];**

**A[1]=temp;**

**}**

**}**

**时间复杂度：O(k\*len)**

**往年考题1：**

1. 单选题（每题2分，共30分）

1. 被计算机加工的数据元素不是孤立无关的，它们彼此之间一般存在着某种联系。通常将数据元素之间的这种联系称为（ ）。

A． 规则 B． 集合 C． 结构 D．运算

2. 执行下面程序段时，语句S的执行次数为（ ）。

for(i=0;i<=n;i++)

for(j=0;j<=i;j++)

S;

A．n2 B．n2/2 C．n(n+1) D．(n+1)(n+2)/2

3. 设单链表中指针p指向结点A之前的结点，若要把指针S指向的结点插在结点A之前，则需修改指针的操作为（ ）。

A．S->next=p->next; p->next=S; B．p->next=S; S->next=p->next;

C．p=p->next->next D．S->next=p->next;

4. 在一个长度为n的顺序表中，在第i个元素（1≤i≤n+1）之前插入一个新元素时须向后移动（ ）个元素。

A．n-i B．n-i+1 C．n-i-1 D．i

5. 若在链表中最常用的操作是删除表中最后一个结点和在最后一个结点之后插入元素，则采用（ ）最节省时间。

A．带头指针的单向循环链表 B．双向链表

C．带尾指针的单向循环链表 D．带头指针的双向循环链表

6. 若已知一个栈的入栈序列是1，2，3，…，n，其输出序列为p1，p2，p3，…，pn，若p1=n，则pi为（ ）。

A．n-i+1 B．n-i C．i D．不确定

7. 已知一棵完全二叉树的第6层有8个叶子结点，则完全二叉树的结点个数最多有（ ）个。

A． 39 B． 52 C． 111 D．119

8. 给定二叉树如右图所示。设N代表二叉树的根，L代表根结点的左子树，R代表根结点的右子树。若遍历后的结点序列为3,1,7,5,6,2,4,则其遍历方式是（　）。

A． LRN B．NRL C． RLN D．RNL

9. 若一棵二叉树的前序遍历序列和后序遍历序列分别为1,2,3,4和4,3,2,1，则该二叉树的中序遍历序列不会是（ ）。

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

10. 下列线索二叉树中（用虚线表示线索）符合后序线索树定义的是（ ）。



11. 若无向连通图G=(V,E)中有7个顶点，则图G中的边的个数最少是（ ）。

 A．6 B．15 C．16 D．21

12. 对右图进行拓扑排序，可以得到不同的拓扑序列的个数是（ ）。

A．4 B．3 C．2 D．1



13. 在右图所示的平衡二叉树中插入关键字48后得到一棵新的平衡二叉树，在新平衡二叉树中，关键字37所在结点的左右结点中保存的关键字分别是（ ）。

A．13，48 B．24，48 C．24，53 D．24，90

14. 已知一个长度为16的顺序表L，其元素按照关键字有序排列，若用折半查找法查找一个L中不存在的元素，则关键字的比较次数最多是（ ）。

A．4 B．5 C．6 D．7

15.一组记录的关键字为{ 46，79，56，38，40，84 }，利用快速排序的方法，以第一个记录为枢轴得到的一次划分结果为（ ）。

A．38，40，46，56，79，84 B．40，38，46，79，56，84

C．40，38，46，56，79，84 D．40，38，46，84，56，79

1. 填空题（每空2分，共20分）

1. 下面程序段的时间复杂度为 （1） 。

i=1;

while(i<=n)

i=i\*3;

2. 假设有6行8列的二维数组A（下标从1开始），每个元素占用6个字节，存储器按字节编址，数组A共占用 （2） 字节。已知A的基地址为1000，按列存储时元素A[3][6]的地址 （3） 。

3. 广义表A=(a,b,(c,d),(e,(f,g)))，则Head(Tail(Head(Tail(Tail(A)))))的值为 （4） 。

4. 若某二叉树有20个叶子结点，有30个结点仅有一个孩子，则该二叉树的总的结点数是 （5） 。

5. Prim算法适合求 （6） 的最小代价生成树MST。

6.下图所示的是带权图的有向图G的邻接表存储结构示意图。从结点v1出发，深度遍历图G所得结点序列为 （7） ；从v1到v8的关键路径为 （8） 。



7. 若一个具有N个顶点K条边的无向图是一森林（N>K)，则该森林中必有 （9） 棵树。

8. 含有9个叶子结点的3阶B-树中至少有 （10） 个非叶子结点。

1. 算法应用（每题5分，共30分）

1. 令主串为“aaabbbababaabb”，子串为“abaa”，请给出KMP算法的匹配过程。

2. 假设用于通讯的电文仅由5个字母a,b,c,d,e组成，字母在电文中出现的频率分别为： 0.14，0.1，0.12，0.46，0.18。请为这5个字母设计哈夫曼编码，并求平均编码长度（平均编码长度为字符出现的概率乘以字符编码长度，再求和）。

3. 已知如右图所示的有向网，试利用Dijkstra算法求顶点v1到其余顶点的最短路径，并给出算法执行过程中各步的状态。

4. 从空树开始，画出按以下次序向3阶B-树中插入关键码的建树过程：20，30，50，52，60，68，70。若删除了50和68，结果又如何？

5. 设哈希函数H（K）=3 K mod 11，哈希地址空间为0～10，对关键字序列（22，13，49，24，38，21，4，12），按线性探测再散列法解决冲突，请分析并构造哈希表，再求出等概率下查找成功时平均查找长度ASLsucc。

6. 关键字序列为（503，087，512，061，908，170，897，275，653，426），执行增量序列为5，3，1的希尔排序算法，写出每一趟排序结束时的关键字状态。

1. 算法设计（每题10分，共20分）

1. 已知线性表中的元素（整数）以值递增有序排列，并以单链表作存储结构。编写算法，删除表中所有大于mink且小于maxk的元素（若表中存在这样的元素）（注意：mink和maxk是给定的两个参变量，它们的值为任意的整数）。

2. 采用邻接表方式存储图（存储结构如下），借助队列编写算法实现图的广度优先搜索算法。说明：队列类型Queue，队列的基本运算有初始化空队列InitQueue(Queue &Q)，入队EnQueue(Queue &Q，ElemType e)，出队DeQueue(Queue &Q，ElemType &e)，判断队列是否为空QueueEmpty(Queue Q)。

#define Maxsize 100

typedef struct ArcNode {

int adjvex; //邻接点域，存放与Vi邻接的点在表头数组中的位置

struct ArcNode \*nextarc; //链域，指示下一条边或弧

}ArcNode ;

typedef struct VNode{

VertexType data; //存放顶点信息

ArcNode \*firstarc; //指示链表中第一个结点

} AdjList[ MaxSize ];

typedef struct{

AdjList vertices; // 顶点表列(邻接表)

int vexnum, arcnum; // 图的当前顶点数和弧数

}ALGraph;

**答案1**

1. 单选题（2分每题，共30分）

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| C | D | A | B | C | A | C | D | C | D | A | B | C | B | C |

1. 填空题（2分每空，共20分）

(1) O（log3n）

(2) 288

(3) 1192

(4) d

(5) 69

(6) 稠密网

(7) V1 v2 v3 v8 v5 v7 v4 v6或V1 v2 v3 v8 v4 v5 v7 v6或V1 v2 v4 v5 v3 v8 v6 v7

(8) V1 v6 v5 v3 v8

(9) N-K

(10) 4

1. 算法应用（5分每题，共30分）

1. 模式串的next[j]值：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| j | 0 | 1 | 2 | 3 |
| 串 | a | b | a | a |
| next[j] | -1 | 0 | 0 | 1 |

第一趟：

a a

a b（j=next[1]=0）

第二趟：

a a a

a a b（j=next[1]=0）

第三趟：

a a a b b

a a a b a（j=next[2]=0）

第四趟：

a a a b b

a a b a a（j=next[1]=-1，i++，j++）

第五趟：

a a a b b b

a a bb b a（j=next[1]=-1，i++，j++）

第六趟：

a a a b b b a b a b

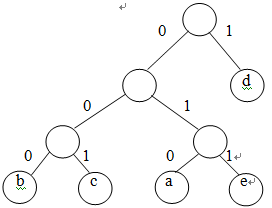
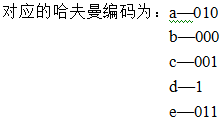
a a a b b b a b a a（j=next[3]=1）

第七趟：

a a a b b b a b a b a a

a a a b b b a b a b a a

2. 字符a，b，c，d，e出现的概率分别为：14%，10%，12%，46%，18%，构造的哈夫曼树如下。平均编码长度=0.14\*3+0.1\*3+0.12\*3+0.46\*1+0.18\*3=2.08

3. 邻接矩阵为：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 终点 | 从V1到各终点的最短路径及其长度 | | | | |
| v2 | 20  <v1,v2> | 19  <v1,v3,v2> | --- | --- | --- |
| v3 | 15  <v1,v3> | --- | --- | --- | --- |
| v4 |  |  | 29  <v1,v3,v2,v4> | 29  <v1,v3,v2,v4> | --- |
| v5 |  | 25  <v1,v3,v5> | 25  <v1,v3,v5> | --- | --- |
| v6 |  |  |  | 45  <v1,v3,v5,v6> | 44  <v1,v3,v2,v4,v6> |
| Vj | V3:20  <v1,v3> | V2:19  <v1,v3,v2> | V5:25  <v1,v3,v5> | V4:29  <v1,v3,v2,v4> | 44  <v1,v3,v2,v4,v6> |

4.





删除了50之后：



删除了68之后：



5. ASLsucc =（6\*1+2\*2）/8=10/8=1.25

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 散列地址 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 关键字 | 22 | 4 |  | 12 | 49 | 38 | 13 | 24 | 21 |  |  |
| 比较次数 | 1 | 1 |  | 1 | 1 | 2 | 1 | 2 | 1 |  |  |

6. 第一趟排序结果：170，087，275，061，426，503，897，512，653，908

第二趟排序结果：061，087，275，170，426，503，897，512，653，908

第三趟排序结果：061，087，170，275，426，503，512，653，897，908

1. 算法设计参考答案（10分每题，共20分）

1. 分析：mink和maxk的合法性：mink < maxk；

不要一个一个的删除（多次修改next域）。

int delete(LinkList L, int mink, int maxk){

if(mink>=maxk) { printf(“参数出错”); return 0; }

pre=L; p=L->next;

while(p!=NULL&&p->data<=mink) { pre=p; p=p->next; }

s=p;

while(s!=NULL&&s->data<maxk) { t =s; s=s->next; free(t); }

pre->next=s;

return 1;

}

2.

void BFSTraverse ( ALGraph G ){

int i, visited[M];

Queue Q；

for( i=0; i<G.vexnum; i++ ) visited[i]=0;

InitQueue(&Q);

for(i=0;i<G.vexnum;i++)

if(!visited[i]){

visited[i]=1; visit(i); EnQueue(&Q,i);

while(!QueueEmpty(Q)){

DeQueue(&Q,&u);

for(p=G.vertices[u].firstarc;p;p=p->nextarc){

w=p->adjvex;

if(!visited[w]){

visited[w]=1; visit(G.vertices[w].data);

EnQueue(&Q,w);

}

}

}

}

**往年考题2：**

1. 单选题（每题2分，共30分）

1. 算法的时间复杂度取决于（ ）。

A．问题的规模 B．待处理数据的初态 C．A与B都对 D．算法的易读性

2. 设n是描述问题规模的非负整数，下面的程序片段的时间复杂度是（ ）。

x=2;

while(x<n/2) x=2\*x;

A．O(log2n) B．O(n) C．O(nlog2n) D．O(n2)

3. 在数据结构中，从逻辑上可以把数据结构分成（ ）。

A．动态结构和静态结构 B．紧凑结构和非紧凑结构

C．线性结构和非线性结构 D．内部结构和外部结构

4. 设单链表中指针p指向结点A，若要删除A之后的结点（若存在），则需修改指针的操作为（ ）。

A．p->next=p->next->next B．p=p->next

C．p=p->next->next D．p->next=p

5. 若线性表中最常用的操作是存取序号为i的元素和在最后进行插入和删除运算，则采用（ ）存储方式时间性能最好。

A．双向链表 B．双向循环链表 C．单向循环链表 D．顺序表

6. 下列选项中，（ ）是链表不具有的特点。

A．插入和删除运算不需要移动元素

B．所需要的存储空间与线性表的长度成正比

C．不必事先估计存储空间大小

D．可以随机访问表中的任意元素

7. 对n(n>=2)个权值均不相同的字符构成哈夫曼树，下列关于该哈夫曼树的叙述中，错误的是（ ）。

A．该树一定是一棵完全二叉树

B．树中一定没有度为1的结点

C．树中两个权值最小的结点一定是兄弟结点

D．树中任一非叶结点的权值一定不小于下一层任一结点的权值

8. 已知一棵完全二叉树有768个结点，则树中叶子结点的个数是（ ）个。

A．257 B．258 C．384 D．385

9. 将森林转换为对应的二叉树，若在二叉树中，结点u是结点v的父结点的父结点，则在原来的森林中，u和v可能具有的关系是（ ）。

Ⅰ.父子关系

Ⅱ.兄弟关系

Ⅲ.u的父结点与v的父结点是兄弟关系

A．只有Ⅱ B．Ⅰ和Ⅱ C．Ⅰ和Ⅲ D．Ⅰ、Ⅱ和Ⅲ

10. 下列关于无向连通图的叙述中，正确的是（ ）。

Ⅰ.所有顶点的度之和为偶数

Ⅱ.边数大于顶点个数减1

Ⅲ.至少有一个顶点的度为1

A．Ⅰ B．Ⅱ C．Ⅰ和Ⅱ D．Ⅰ和Ⅲ

11. 为解决计算机主机与打印机间速度不匹配问题，通常设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（ ）。

A．队列 B．栈 C． 线性表 D．有序表

12. 下列关于图的叙述中，正确的是（　）。

1. 回路是简单路径
2. 存储稀疏图，用邻接矩阵比邻接表更省空间
3. 若有向图中存在拓扑序列，则该图不存在回路

A． （1） B．（1）（2） C．（3） D．（1）（3）

13. 下列二叉排序树中，满足平衡二叉树定义的是（　）。



14. 对于下列关键字序列，不可能构成某二叉排序树中一条查找路径的序列是（　）。

A．95,22,91,24,94,71 B．92,20,91,34,88,35

C．21,89,77,29,36,38 D．12,25,71,68,33,34

15. 下列算法中，（ ）算法可能会出现：当初始数据有序时，花费的时间反而最多。

A．堆排序 B．冒泡排序 C．快速排序 D．希尔排序

1. 填空题（每空2分，共20分）

1. 在顺序表中插入或删除一个元素，需要平均移动 （1） 元素，具体移动的元

素个数与 （2） 有关。

2. 假设有6行8列的二维数组A（下标从1开始），每个元素占用6个字节，存储器按字节编址。已知A的基地址为1000，数组A的最后一个元素的地址 （3） ，按行存储时元素A[3][6]的地址 （4） 。

3. 广义表A=((a,b),(c,d))，则 Tail(A)的值为 （5） 。

4. 已知二叉树有50个叶子结点，则该二叉树的总结点数至少应该有 （6） 个。

5. Kruskal算法适合求 （7） 的最小代价生成树MST。

6. 下图所示的是带权图的有向图G的邻接表存储结构示意图。从结点v1出发，深度遍历图G所得结点序列为 （8） ；从结点v1到结点v8的最短路径为 （9） 。

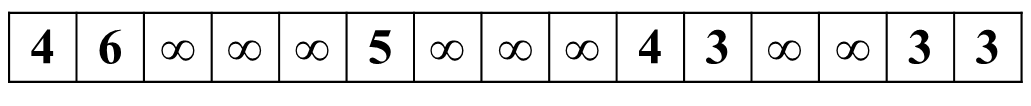


7. 含有10个叶子结点的3阶B-树中至少有 （10） 个非叶子结点。

1. 算法应用（每题5分，共30分）

1. 设一棵度为k的树中有n1个度为1的结点，n2个度为2的结点，…… ，nk个度为k的结点。求该树上长了多少片叶子？

2. 设一棵二叉树的先序序列为ABDEHCFGI，中序序列为DBEHAFCIG，请画出该树。

3. 已知有一个6个顶点（顶点编号0-5）的有向带权图G，其邻接矩阵A为上三角矩阵，它的压缩存储如右图。要求：

（1）写出图G的邻接矩阵A；

（2）画出有向带权图G；

（3）求图G的关键路径，并计算关键路径的长度。

4. 画出对长度为10的有序表进行折半查找的判定树，并求其等概率时查找成功的平均查找长度ASL。

5. 设哈希函数H（K）=3 K mod 11，哈希地址空间为0～10，对关键字序列（22，13，49，24，38，21，4，12），按链地址法解决冲突，请分析并构造哈希表，再分别求出等概率下查找成功时和查找失败时的平均查找长度ASLsucc和ASLunsucc。

6. 已知一组关键字：（40，27，28，12，15，50，7），要求采用快速排序法从小到大排序。请写出每趟排序后的划分结果。

1. 算法设计（每题10分，共20分）

1. 设计算法，将一维数组A种的元素循环右移k位，要求只用一个元素大小的附加存储空间。请给出所设计算法的时间复杂度。

2. 采用邻接矩阵方式存储图（存储结构如下），借助队列编写算法实现图的广度优先搜索算法。说明：队列类型Queue，队列的基本运算有初始化空队列InitQueue(Queue &Q)，入队EnQueue(Queue &Q，ElemType e)，出队DeQueue(Queue &Q，ElemType &e) ，判断队列是否为空QueueEmpty(Queue Q)。

#define Maxsize 100

typedef int AdjMatrix[Maxsize][ Maxsize ];

// AdjMatrix为邻接矩阵，无权图用1或0表示相邻否; 带权图则为权值类型

typedef struct {

VertexType vexs[Maxsize]; //顶点向量

AdjMatrix arcs; //邻接矩阵

int vexnum,arcnum; //图的当前顶点数和弧数

} MGraph;

参考答案2：

1. 单选题（2分每题，共30分）

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| C | A | C | A | D | D | A | C | B | A | A | C | B | A | C |

1. 填空题（2分每题，共20分）

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 一半 | 6 | 99 |
| 2 | 插入或删除元素的位置 | 7 | 稀疏网 |
| 3 | 1282 | 8 | V1 v2 v3 v8 v5 v7 v4 v6 |
| 4 | 1126 | 9 | V1 v2 v5 v7 v8 |
| 5 | ((c,d)) | 10 | 7 |

1. 算法应用（5分每题，共30分）
2. 分析：

总结点数：n=n0+n1+n2+……+nk

总分支数：n-1=n1+2n2+……+knk

所以，叶子结点 n0=n2+2n3+……+(k-1)nk+1

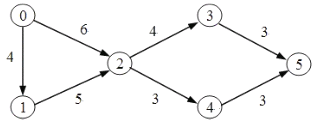
2.

3.

（1）图G的邻接矩阵A为：



（2）有向带权图G为：



（3）

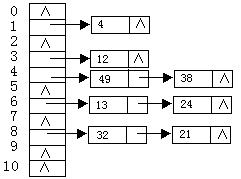
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 顶点 | 0 | 1 | 2 | 3 | 4 | 5 |
| 最早开始时间 | 0 | 4 | 9 | 13 | 12 | 16 |
| 最晚开始时间 | 0 | 4 | 9 | 13 | 13 | 16 |

图G的关键路径为：0,1,2,3,5； 关键路径的长度为：16。

4.



ASL =（1+2\*2+3\*4+4\*3）/10 = 29/10 = 2.9

5. 

ASLsucc =（1\*5+2\*3）/8=11/8

ASLunsucc=（1+2+1+2+3+1+3+1+3+1+1）/11=19/11

6. 初始状态：40 27 28 12 15 50 7

一次划分：{ 7 27 28 12 15 } 40 { 50 }

依次划分：7 { 27 28 12 15 } 40 50

7 { 15 12 } 27 { 28 } 40 50

7 12 15 27 28 40 50

1. 算法设计（10分每题，共10分）参考答案，不唯一。

1.

void shift(ElemType A[],int n,int k){

int temp,j,i;

for(j=1;j<=k;j++){

temp=A[n];

for(i=n-1;i>=1;i--) A[i+1]=A[i];

A[1]=temp;

}

算法的时间复杂度：O(k\*n)。

2.

void BFSTraverse ( MGraph G ){

int i, visited[M]; Queue Q；

for( i=0; i<G.vexnum; i++ ) visited[i]=0;

InitQueue(&Q);

for(i=0;i<G.vexnum;i++)

if(!visited[i]){

visited[i]=1; visit(i); EnQueue(&Q,i);

while(!QueueEmpty(Q)){

DeQueue(&Q,&u);

for(j=0;j<G.vexnum;j++){

if( G.arcs[u][j] ==1)

if(!visited[j]) { visited[j]=1;visit(G.vexs[j]);EnQueue(&Q,j); }

}

}

}

}