1998年试题参考答案

数据结构试题

一、

栈是一种只能在一端进行插入或删除操作的线性表；

队列是一种只允许在表的一端插入在另一端删除的线性表。

二、

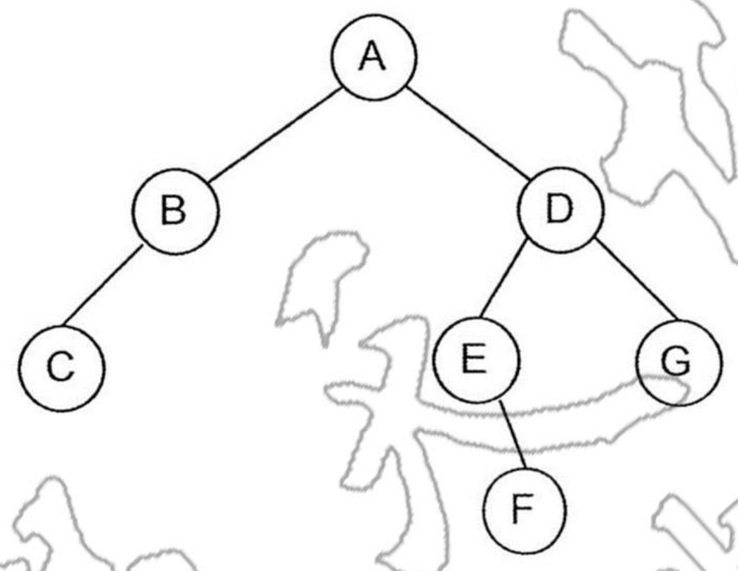
（1）

先序遍历：A B D E C F

中序遍历：D B E A C F

后序遍历：D E B F C A

（2）如图所示：



三、参照课本自行举例。

四、参照课本自行举例。（普利姆算法（Prim）或者克鲁斯卡尔算法（Kruskal）均可，写出一种就好）

五、

```
void MatrixToList(MGraph g,ALGraph *&G)
{//将邻接矩阵 g 转换成邻接表 G
   int i,j,n=g.vexnum;//n 为顶点数
   ArcNode *p;
```

```
G=(ALGraph *)malloc(sizeof(ALGraph));
for (i=0;i<n;i++)//给邻接表中所有头结点的指针域置初值
   G->vertices[i].first=NULL;
for (i=0;i<n;i++)//检查邻接矩阵中每个元素
   for (j=n-1;j>=0;j--)
      if (g.edge[i][j]!=0)//邻接矩阵的当前元素不为 0
      {
         p=(ArcNode *)malloc(sizeof(ArcNode));//创建一个结点*p
         p->adjvex=j;
         p->info=g.edge[i][j];
         p->next=G->vertices[i].first;//将*p 链到链表后
         G->vertices[i].first=p;
      }
   G->vexnum=n;
   G->edgenum=g.edgenum;
}
```

注：一般在图的算法中只需要写出算法，不需要写结点类型，除非题目要求。

程序设计试题

一、（1）

```
#include<iostream.h>
void main()
{
    int K,i,j,count=0;
    cin>>K;
    for(i=0,j=0;;)
    {
        if((2*i+1)<(3*j+1))
        {
            cout<<2*i+1<<" ";
            count++;
            i++;
        }
        else if((2*i+1)==(3*j+1))
        {
            cout<<2*i+1<<" ";
            count++;
            i++;
            j++;
        }
        else
        {
            cout<<3*j+1<<" ";
            count++;
            j++;
        }
```

```
            if(count==K)
                break;
        }
    }
```

（2）

```
#include<iostream.h>
void main()
{
    int K,a,b,i,j,count=0;
    cin>>K>>a>>b;
for(i=0,j=0;;)
{
        if((a*i+1)<(b*j+1))
        {
            cout<<a*i+1<<" ";
            count++;
            i++;
        }
        else if((a*i+1)==(b*j+1))
        {
            cout<<a*i+1<<" ";
            count++;
            i++;
            j++;
        }
        else
        {
            cout<<b*j+1<<" ";
            count++;
            j++;
        }
        if(count==K)
            break;
    }
}
```

二、

```
typedef struct LNode
{//结点定义
    ElemType data;
    struct LNode *next;
}LNode;
```

（1）

```
    void Insert(LNode *&L,ElemType x)
    {
        int *p=L;
```

```
while(p->next)
{
        if(p->next->data<x)
                p=p->next;
        else
        {
                LNode *q;
                q=(LNode *)maloc(sizeof(LNode));
                q->data=x;
                q->next=p->next;
                p->next=q;
                return;
            }
        }
        p->next->data=x;
        p->next->next=NULL;
    }
```

（2）

```
    void Reverse(LNode *&L)
    {
        LNode *p=L->next,*q;
        L->next=NULL;
        while(p)
        {
            q=p->next;
            p->next=L->next;
            //将 p 所指结点插入新的链表
            L->next=p;
            p=q;
        }
    }
```

设计题

五、

```
    void Convert(ALGraph *G,int arcs[M][N])
    {
        for(i=0;i<n;i++)
        {//依次遍历各顶点表结点为头的边链表
            p=(G->vertices[i]).first;
            while(p)
            {
                arcs[i][p->data]=1;
                p=p->next;
            }
        }
    }
```

六、

```
#include<iostream.h>
int PerNum(int n)
{//判断 n 是否为完全数，若是则返回 1，否则返回 0
    int i,sum=0;
    for(i=1;i<=n/2;i++)
            if(n%i==0)
                    sum+=i;//如果是因子则加到 sum 中
    if(sum==n)
            return 1;
    return 0;
}
void main()
{
    int i;
    for(i=1;i<=1000;i++)
            if(PerNum(i))
                    cout<<i<<endl;
}
```

七、

假设采用二叉链表存储结构：

```
typedef struct BTNode
        {
            ElemType data;
            struct BTNode *lchild;
            struct BTNode *rchild;
} BTNode;
```

法一：

```
int n;
void count(BTNode *p)
{
        if(p)
        {
                if(p->lchild==NULL&&p->rchild==NULL)
                        n++;
                count(p->lchild);
                count(p->rchild);
        }
}
```

法二：

```
int count(BTNode *p)
{
```

```
int n1,n2;
if(p==NULL)
        return 0;
else if(p->lchild==NULL&&p->rchild==NULL)
        return 1;
else
{
        n1=count(p->lchild);
        n2=count(p->rchild);
        return n1+n2;
        }
}
```

八、

单链表结点定义：

```
typedef struct LNode
{
        ElemType data;
        struct LNode *next;
}LNode;
```

算法：

```
void Search(LNode *&C, ElemType x)
{
        LNode *p.*q;
        p=C;
        while(p->next)
        {//寻找查找结点的前驱
                if(p->next->data==x)
                        break;
                p=p->next;
        }
        if(p->next==NULL)
                cout<<"Cannot search!";
        else
                {//删除操作
                        q=p->next;
                        p->next=p->next->next;
                        free(q);
                }
        }
```

九、

递归方式：

```
int Fib(int n)
{
        if(n==1||n==2)
                return 0;
        if(n==3)
                return 1;
        return Fib(n-1)+Fib(n-2)+Fib(n-3);
}
```

非递归方式：

```
int Fib(int n)
{
        if(n==1||n==2)
                return 0;
        if(n==3)
                return 1;
        int tmp1=0,tmp2=0,tmp3=1,tmp;
        for(int i=4;i<=n;i++)
        {
                tmp=tmp1+tmp2+tmp3;
                tmp1=tmp2;
                tmp2=tmp3;
                tmp3=tmp;
        }
        return tmp;
}
```

十、

冒泡排序算法：

```
void BubbleSort(ElemType A[],int n)
{
        for(i=n-1;i>=1;i--)
        {
                flag=0;
                for(j=1;j<=i;j++)
                        if(A[j-1]>A[j])
                        {
                                temp=A[j];
                                A[j]=A[j-1];
                                A[j-1]=temp;
                                flag=1;
                        }
                if(flag==0)
                        return;
        }
}
```

改进后，双向冒泡排序算法：

```
void BubbleSort(ElemType A[],int n)
{
        int low=0,high=n-1;
        int flag=1;
        while(low<high&&flag)
        {
flag=0;
for(i=low;i<high;i++)
            if(A[i]>A[i+1])
            {
                    temp=A[i];
                    A[i]=A[i+1];
                    A[i+1]=temp;
                    flag=1;
            }
high--;
for(j=high;j>low;j--)
             if(A[j]<A[j-1])
             {
                    temp=A[j];
                    A[j]=A[j-1];
                    A[j-1]=temp;
                    flag=1;
                }
        low++;
    }
}
```