电子科技大学信息与软件工程学院

课程作业

点名序号 47

学　　号 2016220304022

姓　　名 罗悦

课程名称 数据结构与算法

理论教师 陈安龙

开课时间 2016-2017-2

电 子 科 技 大 学

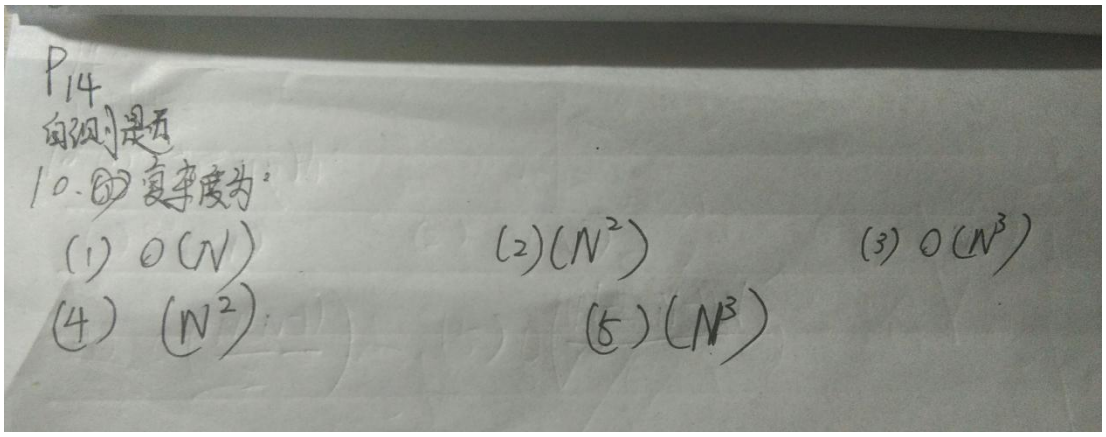
作业 1

本章作业

1. 教材14-15页的自测题10
2. 编程完成教材17页的编程项目2和3题，并分析时间复杂度

答案：

1.



2.

（2）

```
#include<stdio.h>
int main(){
	int i,j,m,n;
	float k,s;
	  s=0;
	for(i=1;i<=n;i++){printf("aa");
		m=1;
		for(j=1;i<=j;j++){
			m=m*j;
		  }
		  k=1/m;
		s=s+k;
	}
	return 0;
}
```

//时间复杂度为 O(n*n)。

（3）

```
#include<stdio.h>
int main(){
	int i;
	int j;		//公差
	  int a[20];
	  j=(330-300)/10;
	a[0]=(630-190*3)/20;
	for(i=0;i<19;i++){
		a[i+1]=a[i]+j;
	}
	printf("数列为：\n");
	for(i=0;i<20;i++){
	printf("%d\t",a[i]);
	}
	  return 0;
}
```

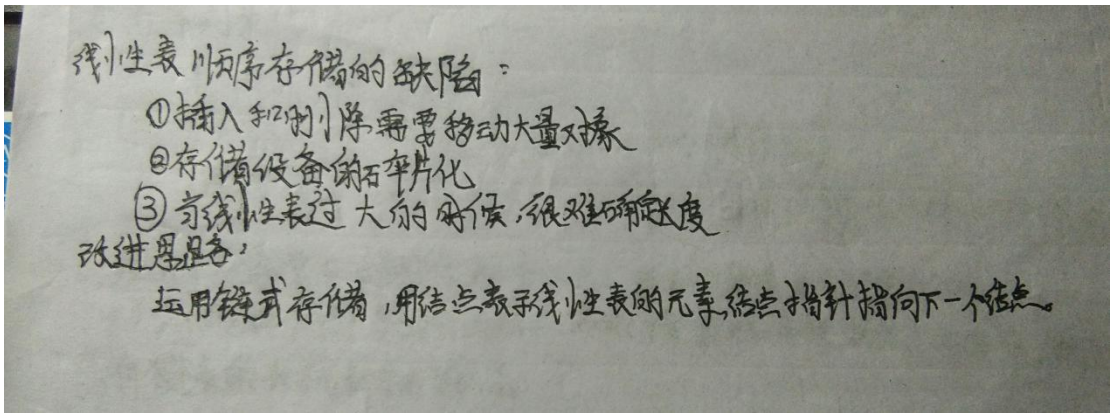
作业 2

顺序表部分作业

1. 分析总结线性表顺序存储的缺陷，并给出改进思路。
2. 在VS2010或VS2013环境下，用C语言编写实现线性表顺序存储结构的10种基本操作代码，并编写测试这些操作的main()程序代码。

答案：

1.



2.

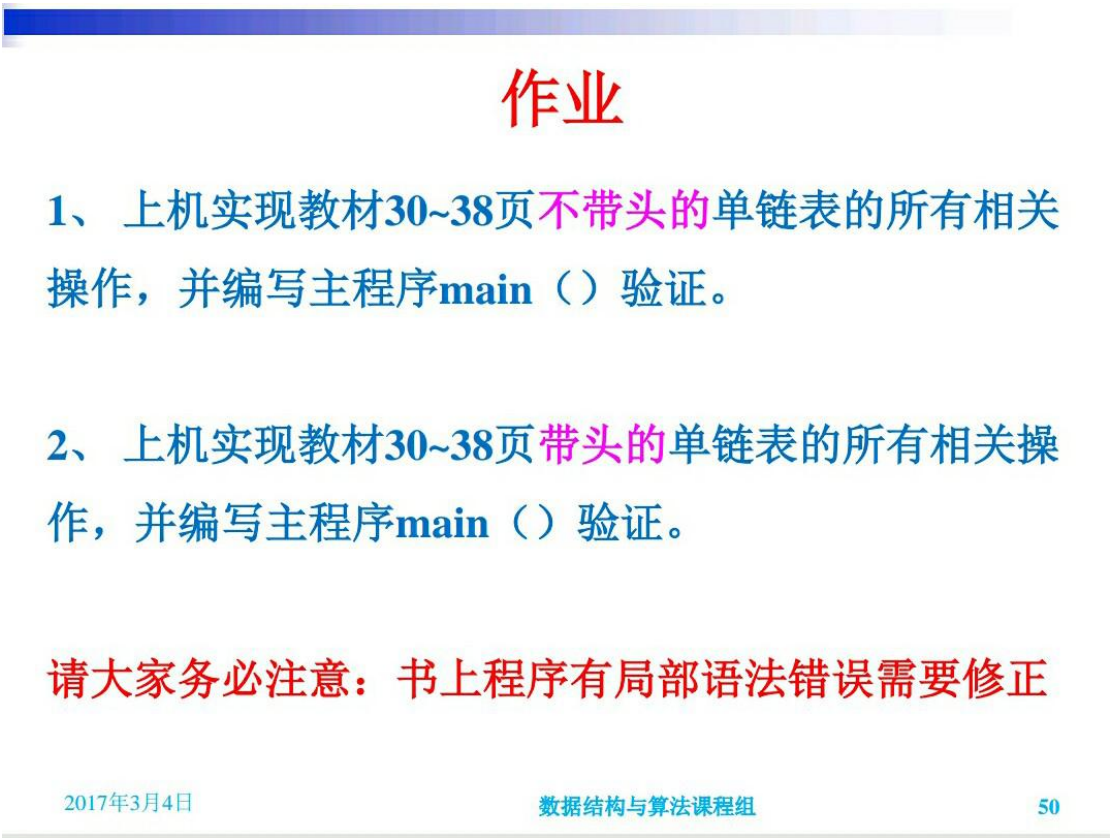
```
#include<stdio.h>
#include<stdlib.h>
#define maxSize 100                //最大允许长度

typedef int Elemtype;              //元素的数据类型
struct    node
```

```
{
        Elemtype data[maxSize];                    //存储数组
      int length;                                  //当前表元素个数
};
typedef struct node SeqList;

void SeqListInit (SeqList &L){                     //初始化操作
        L.length = 0;
}



int SeqListLength (SeqList& L){                    //求线性表长度
        return L.length;
}

Elemtype SeqListGet (SeqList& L,int i){     //取元素
        if ((i>=1)&&(i<=L.length))
            return L.data[i-1];
        else{
                printf("i  值不合法");
                exit(0);
        }
}

int SeqListLocate (SeqList& L, Elemtype e){        //定位函数
        int i；
        while ((i<=L.length)&&(e!=L.data[i-1]))
                i++;
          if (i<=L.length)
          return i;
          else{
                        printf("次元素在顺序表中不存在");
            return 0;
                }
}

Elemtype SeqListPrior (SeqList& L, Elemtype e){    //求前驱函数
        int i = SeqListLocate(L,e);
      if (1==i){
                printf("第 1 个元素没有前驱");
          exit(0);
        }
        else return L.data[i-2];
```

```
}

Elemtype SeqListNext (SeqList& L, Elemtype e){     //求后驱函数
        int i = SeqListLocate(L,e);
        if (L.length==i) {
              printf("最后 1 个元素没有后继");
           exit(0);
        }
            else return L.data[i];
}

int SeqListInsert ( SeqList &L, int i, ElemType b){     //前插操作
        if ( L.length == maxSize ){
        printf("表已满，无法插入");
      return 0;
      }
      if ( i < 1 || i > L.length ) {
      printf("i 不合法");
      return 0;
        }
        for (int j=L.length-1; j >= i-1; j--)
      L.data[j+1] = L.data[j];
        L.data[i-1] = b;              //实际插在数组第 i-1 个位置
        L.length++;
        return 1;                        //插入成功
}

int SeqListDel ( SeqList& L, int i ) {                        //删除操作
        if ( i < 1 || i > L.length) {
              printf("i 不合法");             //i 值越界
              return 0;
        }
        for (int j = i; j <= L.length-1; j++)
      L.data[j-1] = L.data[j];
      L.length--;
        return 1;                              //成功删除
}

void SeqListTraverse(SeqList &L){                                  //判空表函数
        if (SeqListEmpty(L))
           printf("该表为空");
        else
        for (int i = 1; i <= L.length; i++)
           printf("%d\n", L.data[i - 1]);
```

```
}

int main(){
    int i,j;
    SeqList L1,L2;
    Elemtype e;
    SeqListInit (L1);
    SeqListInit (L2);
    SeqListLength (L1);
    SeqListGet (L1,i);
    SeqListGet (L2,i);
    SeqListLocate (L1,e);
    SeqListPrior (L1,e);
    SeqListNext (L1,e);
    SeqListInsert (L1,i,b);
    SeqListDel (L1,i);
    SeqListTraverse(L1);
    return 0;
}
```

作业 3



答案 ：

1.

```
#include<stdio.h>
#include<stdlib.h>
#define maxSize 100                //最大允许长度

typedef char ElemType;
struct Node {                                    //链表结点
        ElemType    data;                            //结点数据域
      struct Node * next;                       //结点指针域
};
typedef    struct Node LNode;
typedef    struct Node *LinkedList;
LNode * Head;                                    //链表头指针

int LinkedListInit1(LNode *L) //不带头的单链表初始化
    {
            L=NULL;
            return 1;        }


int LinkedListLength(LNode *L) //不带头单链表求长度
{
      Lnode *p;              //p 需要声明为 Lnode 指针类型
      p=L;
      j=0;
      while(p!=NULL)
      {
            j++;
            p=p->next;           //将 p 向下移动一个结点
      }
      return j;
}

LNode    LinkedListGet(LNode *L,int i) //不带头单链表取元素的算法
{
      Lnode    *p=L;
      int j=1;
      while((p!=NULL)&&(j<i))
      {
            p=p->next;
          j++;
```

```
    }
    return p;
}

LNode LinkedListLocate(LNode *L, ElemType e)//不带头单链表的定位操作
{
    LNode *p=L;
    while((p! = NULL)&&(p->data !=  e))
    p=p->next;
    return p;
}

int LinkedListInsert(LNode *L, LNode * p,ElemType e){   //不带头单链表的插入操作代码
    LNode *q=(LNode *)malloc(sizeof(LNode));   //创建一个新的结点 q
    if(q==NULL)
    {   printf("申请空间失败！");
    return 0;
    }
    q->data=e;
    if(p==L)                //在表头插入
    {
    q->next=L;
    L=q;
    exit(0);
    }
     //在表的中间或末尾进行插入
    LNode *pre=L;
    while((pre!=NULL)&&(pre->next!=p))
    pre=pre->next;//找 P 前驱
    q->next=pre->next;
    pre->next=q;
    return 1;
}


void LinkedListDel(LNode *L, ElemType e) {  //不带头单链表的删除操作代码
    Lnode *pre=L;
    Lnode * p;
    if (L->data==e)
    {p=L;L=L->next; free(p);}
    else {                  //查找 e 的前驱
        while((pre!=NULL)&&(pre->next->data!=e))
          pre=pre->next;
```

```
            if(pre!=NULL) //找到需要删除的结点
            {    p=pre->next;
              pre->next=p->next;
            free(p);
            }
       }
}

int main(){
      int i,j;
      ElemType e;
      LNode * p;
      LNode *L1,*L2,*L3;
      ElemType a[100];
      LinkedListInit(L1);
      LinkedListInit(L2);
      LinkedListInit(L3);
      LinkedListLength(L1);
      LinkedListGet(L1,i);
      LinkedListLocate(L1,e);
      LinkedListInsert(L1,p,e);
      LinkedListDel(L1,e);
      return 0;
}
```

2.

```
#include<stdio.h>
#include<stdlib.h>
#define maxSize 100              //最大允许长度

typedef char ElemType;
struct Node {                                  //链表结点
          ElemType    data;                        //结点数据域
       struct Node * next;                     //结点指针域
};
typedef    struct Node LNode;
typedef    struct Node *LinkedList;
LNode * Head;                                  //链表头指针

int LinkedListInit(LNode *L){              //带头的单链表初始化
          L=(LNode *)malloc(sizeof(LNode));
          if(L==NULL){
          printf("申请空间失败！ ");
```

```
        return 0;
        }
        L->next=NULL;
     return 1;
}

int LinkedListLength(LNode *L) //带头单链表求长度
{
    Lnode *p;              //p 需要声明为 Lnode 指针类型
    p=L->next;
    j=0;
    while(p!=NULL)
    {
        j++;
        p=p->next;          //将 p 向下移动一个结点
    }
    return j;
}

LNode   LinkedListGet(LNode *L,int i) //带头单链表取元素的算法
{
    Lnode   *p=L->next;
    int j=1;
    while((p!=NULL)&&(j<i))
    {
        p=p->next;
      j++;
    }
    return p;
}

LNode LinkedListLocate(LNode *L, ElemType e)//带头单链表的定位操作
{
    LNode *p=L->next;
    while((p! = NULL)&&(p->data !=   e))
    p=p->next;
    return p;
}

int LinkedListInsert(LNode *L, LNode * p,ElemType e){    //带头单链表的插入操作
代码
    LNode * q=(LNode *)malloc(sizeof(LNode));//创建一个新的结点 q
    if(q==NULL)
    {    printf("申请空间失败！ ");
```

```
        return 0;
    }
    q->data=e;                          //插入新结点
    LNode * pre=L;
    while((pre!=NULL)&&(pre->next!=p))
    pre=pre->next;                      //找 P 前驱
    q->next=pre->next;
    pre->next=q;
    return 1;
}

void LinkedListDel(LNode *L, ElemType e) { //带头单链表的删除操作代码
    Lnode * pre=L;              //查找 e 的前驱
    while((pre!=NULL)&&(pre->next->data!=e)) pre=pre->next;
    Lnode * p=pre->next;
    if(p!==NULL)                //找到需要删除的结点
    {
        pre->next=p->next;
       free(p);
    }
}

int LinkedListCreate1(LNode *&L, ElemType a[],int n) {    //用头插法创建带头结点
的单链表
    LNode * pre=L;
    L=(LNode *)malloc(sizeof(LNode));
    if(L==NULL)       {
    printf("申请空间失败！");return 0;
    }
    L->next=NULL;
    for(int i=n-1;i>=0;i--)
    {    LNode *p=(LNode *)malloc(sizeof(LNode));
    if(p==NULL)    {
        printf("申请空间失败！");
       return 0;
       }
       p->data=a[i];
        p->next=L->next;
       L->next=p;
    }
    return 1;
}

int LinkedListCreate2(LNode *&L, ElemType a[],int n) //用尾插法创建带头结点的单
```

链表

```
{    L=(LNode *)malloc(sizeof(LNode));
      if(L==NULL)       {
      printf("申请空间失败！"); return 0;
      }
      L->next=NULL;
      LNode * tail=L;              //设置一个尾指针，方便插入
      for(int i=0;i<n;i++)
      {
      LNode *p=(LNode *)malloc(sizeof(LNode));
      if(p==NULL) {
            printf("申请空间失败！");
          return 0; }
            p->data=a[i];
          p->next=NULL;
          tail-> next=p;
          tail=p;
      }
      return 1;
}

void LinkedListMerge(LNode *La, LNode *Lb, LNode *&Lc)    //带头结点的单链表保序合并操作
{     LNode *pa, pb, pc;
      pa=La->next;
      pb=Lb->next;
      Lc=La;
      pc=Lc;     //借用表 La 的头结点作为表 Lc 的头结点
      while((pa!=NULL)&&(pb!=NULL))         {
            if(pa->data<=pb->data)
              {
            pc->next=pa;
            pc=pa;
            pa=pa->next;
            }
            else
              {
            pc->next=pb;
            pc=pb;
            pb=pb->next;
            }
      }
      if(pa!=NULL)
      pc->next=pa;
```

```
        else pc->next=pb;
        free(Lb);                    //将 Lb 的表头释放
}

int main(){
        int i,j;
        ElemType e;
        LNode * p;
        LNode *L1,*L2,*L3;
        ElemType a[100];
        LinkedListInit(L1);
        LinkedListInit(L2);
        LinkedListInit(L3);
        LinkedListLength(L1);
        LinkedListGet(L1,i);
        LinkedListLocate(L1,e);
        LinkedListInsert(L1,p,e);
        LinkedListDel(L1,e);
        LinkedListCreate1(L1,a,n);
        LinkedListCreate2(L2,a,n);
        LinkedListMerge(L1, L1, L3);
        return 0;
}
```

作业 4

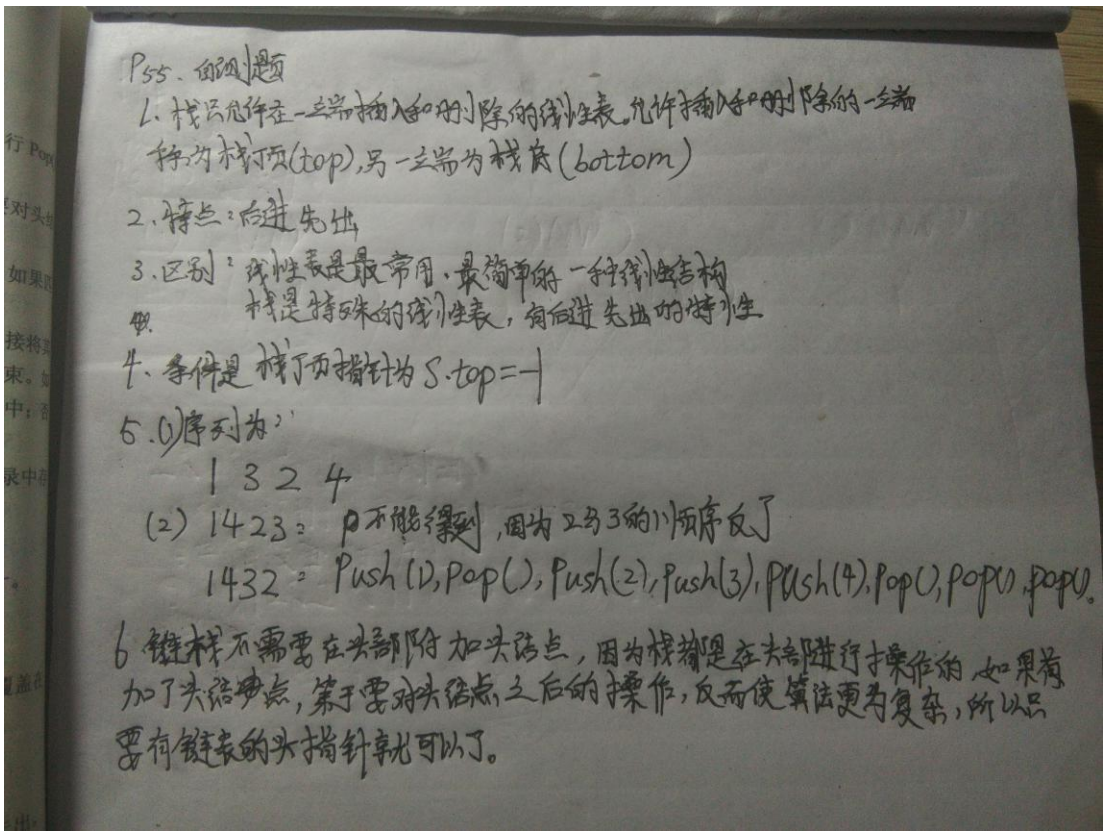
栈的作业

1、完成教材55页的自测题。

2、自学栈的应用，用c语言编程实现带括号的+-*/四则混合表达式的计算。

答案：

1.



2.

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
//#include"SeqStack.h"
#define StackSize 100

typedef char DataType;
typedef struct
{
    DataType stack[StackSize];
    int top;
}SeqStack;

void InitStack(SeqStack *S)
/*将栈初始化为空栈只需要把栈顶指针 top 置为 0*/
{
S->top=0;       /*把栈顶指针置为 0*/
}
int StackEmpty(SeqStack S)
/*判断栈是否为空，栈为空返回 1，否则返回 0*/
```

```
{
    if(S.top==0)                /*判断栈顶指针 top 是否为 0*/
        return 1;               /*当栈为空时，返回 1；否则返回 0*/
    else
        return 0;
}
int GetTop(SeqStack S, DataType *e)
/*取栈顶元素。将栈顶元素值返回给 e，并返回 1 表示成功；否则返回 0 表示失
败。*/
{
   if(S.top<=0)          /*在取栈顶元素之前，判断栈是否为空*/
{
    printf("栈已经空!\n");
    return 0;
}
else
{
    *e=S.stack[S.top-1];      /*在取栈顶元素*/
    return 1;
}
}
int PushStack(SeqStack *S,DataType e)
/*将元素 e 进栈，元素进栈成功返回 1，否则返回 0.*/
{
if(S->top>=StackSize)                  /*在元素进栈前，判断是否栈已经满*/
{
        printf("栈已满，不能进栈！\n");
        return 0;
}
else
{
        S->stack[S->top]=e;              /*元素 e 进栈*/
        S->top++;                         /*修改栈顶指针*/
        return 1;
}
}
int PopStack(SeqStack *S,DataType *e)
/*出栈操作。将栈顶元素出栈，并将其赋值给 e。出栈成功返回 1，否则返回 0*/
{
    if(S->top<=0)       /*元素出栈之前，判断栈是否为空*/
    {
        printf("栈已经没有元素，不能出栈!\n");
        return 0;
    }
```

```
    else
{
    S->top--;                   /*先修改栈顶指针，即出栈*/
        *e=S->stack[S->top];   /*将出栈元素赋值给 e*/
        return 1;
    }
}
int StackLength(SeqStack S)
/*求栈的长度，即栈中元素个数，栈顶指针的值就等于栈中元素的个数*/
{
    return S.top;
}

typedef struct/*操作数栈的类型定义*/
{
    float data[MaxSize];
    int top;
}OpStack;
void TranslateExpress(char s1[],char s2[]);
float ComputeExpress(char s[]);
void mai()
{
    char a[MaxSize],b[MaxSize];
    float f;
    printf("请输入一个算术表达式：\n");
    gets(a);
    printf("中缀表达式为：%s\n",a);
    TranslateExpress(a,b);
    printf("后缀表达式为：%s\n",b);
    f=ComputeExpress(b);
    printf("计算结果：%f\n",f);
}
float ComputeExpress(char a[])
/*计算后缀表达式的值*/
{
    OpStack S;
    int i=0,value;
    float x1,x2;
    float result;
    S.top=-1;
    while(a[i]!='\0')
    {
        if(a[i]!=' '&&a[i]>='0'&&a[i]<='9')/*如果当前字符是数字字符，则将其转换
为数字并存入栈中*/
```

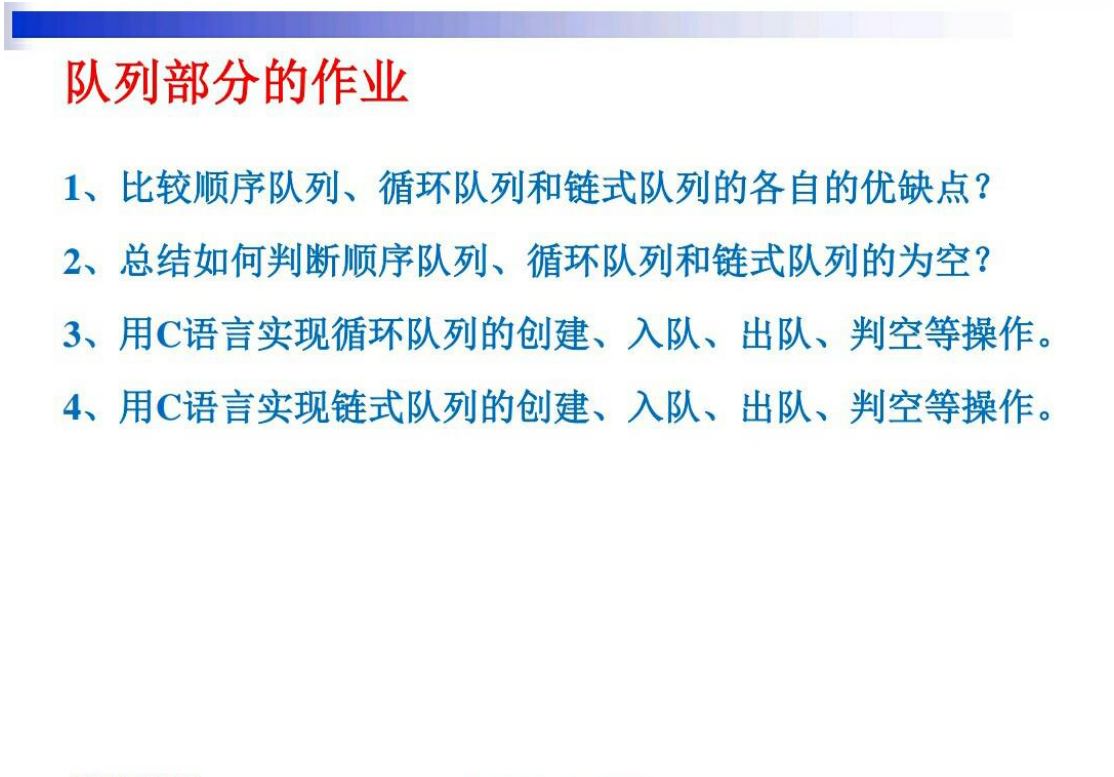
```
        {
            value=0;
            while(a[i]!=' ')
            {
                value=10*value+a[i]-'0';
                i++;
            }
            S.top++;
            S.data[S.top]=value;
        }
        else                    /*如果当前字符是运算符，则对栈中的数据进行求值，
并将结果保存到栈中*/
        {
            switch(a[i])
            {

            case '+':
                x1=S.data[S.top];
                S.top--;
                x2=S.data[S.top];
                S.top--;
                result=x1+x2;
                S.top++;
                S.data[S.top]=result;
                break;
            case '-':
                x1=S.data[S.top];
                S.top--;
                x2=S.data[S.top];
                S.top--;
                result=x2-x1;
                S.top++;
                S.data[S.top]=result;
                break;
            case '*':
                x1=S.data[S.top];
                S.top--;
                x2=S.data[S.top];
                S.top--;
                result=x1*x2;
                S.top++;
                S.data[S.top]=result;
                break;
            case '/':
```

```
                    x1=S.data[S.top];
                    S.top--;
                    x2=S.data[S.top];
                    S.top--;
                    result=x2/x1;
                    S.top++;
                    S.data[S.top]=result;
                    break;
                }
                i++;
            }
        }

        if(!S.top!=-1)
        {
            result=S.data[S.top];
            S.top--;
            if(S.top==-1)
                return result;
            else
            {
                printf("表达式错误");
                exit(0);
            }
        }
}
void TranslateExpress(char str[],char exp[])
/*将中缀表达式转换为后缀表达式*/
{
        OpStack S;
        char ch;
        DataType e;
        int i=0,j=0;
        InitStack(&S);
        ch=str[i];
        i++;
        while(ch!='\0')
        {
            switch(ch)
            {
            case'(':      /*左括号入栈*/
                PushStack(&S,ch);
                break;
            case')':      /*如果当前字符是右括号，则将栈中的字符出栈，直到栈中
```

```
的一个左括号出栈为止*/
                while(GetTop(S,&e)&&e!='(')
                {
                    PopStack(&S,&e);
                    exp[j]=e;
                    j++;
                }
                PopStack(&S,&e);/*左括号出栈*/
                break;
            case'+':
            case'-':
                while(!StackEmpty(S)&&GetTop(S,&e)&&e!='(')/*如果当前字符是+号
或-号，则将栈中字符出栈，直到遇到左括号为止*/
                {
                    PopStack(&S,&e);
                    exp[j]=e;
                    j++;
                }
                PushStack(&S,ch);/*将当前字符入栈*/
                break;
            case'*':
            case'/':
                while(!StackEmpty(S)&&GetTop(S,&e)&&e=='/'||e=='*')/* 如 果 当 前
字符是*号或者是/号，则将栈中字符出栈*/
                {
                    PopStack(&S,&e);
                    exp[j]=e;
                    j++;
                }
                PushStack(&S,ch);    /*当前字符入栈*/
                break;
            case' ':
                break;
            default:                        /*处理数字字符*/
                while(ch>='0'&&ch<='9')
                {
                    exp[j]=ch;
                    j++;
                    ch=str[i];
                    i++;
                }
                i--;
                exp[j]=' ';
                j++;
```

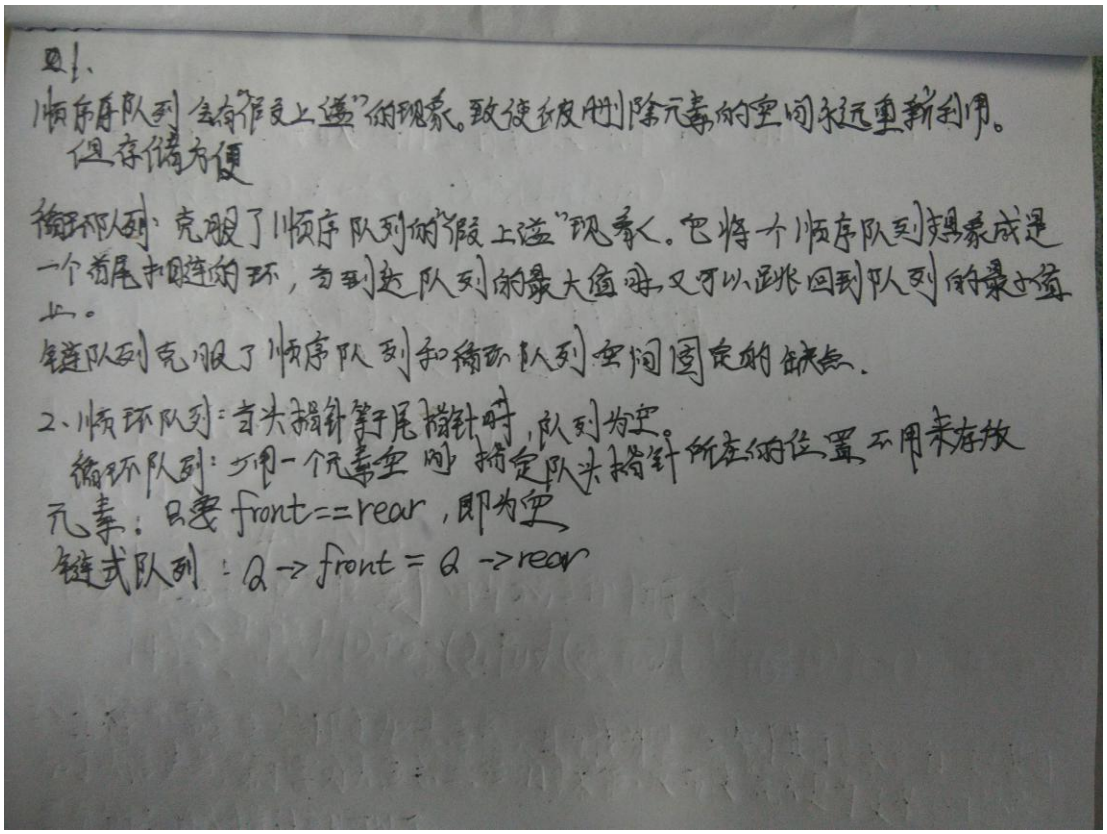
```
            }
            ch=str[i];
            i++;
        }
        while(!StackEmpty(S))
        {
            PopStack(&S,&e);
            exp[j]=e;
            j++;
        }
        exp[j]='\0';
}
```

作业 5



答案：

1.2



3.

```
#include<stdio.h>
# define MAXSIZE 100;
typedef    int    ElemType;
typedef struct {
        ElemType data[MAXSIZE];
      int front;
      int rear;
} CirQueue;

void EnCirQueue(CirQueue    &Q,ElemType x)       //循环队列入队操作
{
        if ((Q.rear+1) %MAXSIZE == Q.front )
        {
                printf("队列已满，无法进行插入！ ");
           exit(0);
        }
          Q.rear =(Q.rear+1) %MAXSIZE;
          Q.data[Q.rear]=x;
```

```
}

Elemtype DeCirQueue(CirQueue    &Q)                  //循环队列出队操作
{
        if (Q.rear == Q.front )
        {
                printf("队列已空，无法进行出队操作！");
           exit(0);
        }
         Q.front=(Q.front+1)    % MAXSIZE;
         return (Q.data[Q.front]);
}

int em(CirQueue    &Q)                                  //循环队列判空
{
        if (Q.rear == Q.front )
        {
                return 1;
        }
      else
              return 0;
}
```

4.

```
#include<stdio.h>
#include<stdlib.h>

struct Node {                                    /*链式队列的结点结构*/
      ElemType    data;                        /*队列的数据元素类型*/
      struct Node    *next;                  /*指向后继结点的指针*/
};
typedef struct node LQNode；
typedef struct node *LinkedQNode；
struct Queue{                                    /*封装的链式队列*/
      LQNode      *front;                    /*队头指针*/
      LQNode *rear;                       /*队尾指针*/
};
typedef struct Queue LQueue；
typedef struct Queue *LinkedQueue；

void    LinkedQueueInit(LinkedQueue    Q)          //链式队列的初始化
{     LinkedQNode    p = (LinkedQNode) malloc(sizeof(LQNode));
```

```
    if   (p==NULL)
      {
    printf("头结点空间申请失败！");
    exit(0);
      }
      else
      {          p->next=NULL;
                 Q->rear=p;
                 Q->front=p;
      }
}

int   LinkedQueueEmpty(LinkedQueue   Q)    //链式队列的判空操作
{
    if   (Q->front== Q->rear)
                  return 1;
    else
              return 0;
}

void   EnLinkedQueue (LinkedQueue   Q,ElemType x)   {   //链式队列的入队操作
    LinkedQNode q = (LinkQNode) malloc(sizeof(LQNode));
    if   (q==NULL)
    {
          printf("空间申请失败！");
          exit(0);
    }
    else
    {
          q->data=x
          q->next=NULL;
          Q->rear->next=q;
        Q->rear=q;
    }
}

Elemtype   OutLinkedQueue (LinkedQueue   Q)   {      //链式队列的出队操作
      if   (Q->front==Q->rear)
      {
        printf("队列为空，无法进行出队操作！");
        exit(0);
      }
      LinkedQNode q=Q->front->next;
      Q->front->next=q->next;
```

```
        x=q->data;
        free(q);
        if      (Q->front->next=NULL)
              Q->rear=Q->front;
           return x;
}
```

作业 6

作业

1. 编写算法，从串 s 中删除所有和串 t 相同的子串。
2. 编写算法，实现串的基本操作 Replace(&S,T,V)。
3. 假设以块链结构作串的存储结构。试编写判别给定串是否具有对称性的算法，并要求算法的时间复杂度为 O(StringLength(S))。

答案：

1. 2. 3.

```
#include<stdio.h>
#define MAXSIZE 256;

char String [MAXSIZE];
struct Node{
        char *str;          //按串长分配空间
        int length;         //串长
```

```
}
typedef struct Node string;

void    SeqStringAssign(string *S,string *T)
{    if((S->str)!=NULL) free(S->str);
S->length = T->length;          if (S->length==0) {
              S->str = (char *)malloc(sizeof(char));
            if(!S->str)    {      printf("空间分配失败！ ");
            exit(0);    }
              S->str='\0';
             }
             else {
                        S->str = (char *)malloc((S->length+1)*sizeof(char));
                  if(!S->str) {    printf("空间分配失败！ ");
                  exit(0);
                  }
                  for(int i = 0; i < =S->length; i++)
                  S->str[i] = T->str[i];
                    }
}

string* SeqStringAssign(string *S, char* V) //串的赋值操作
{
      if ((S->str) != NULL)    free(S->str);
      S->length = StrLength(V);
      S->str = (char*)malloc((S->length)*sizeof(char));
        if (!S->str)
      {      printf("overflow");
        exit(0);
      }
      for (int i = 0; i < S->length; i++)
      {      S->str[i] = V[i];           }
        return S;
}

void findnext(String P,int *next)
{
         next[0]=-1;
         j=0;
         k=-1;
         while(j<Length(P)){
            while(k==0||P[j]==P[k]){
                  j++;
                  k++;
```

```
                next[j]=k;
            }
            k=next[k];
        }
}

int KMP(string *S, string *P, int *next)
{
        findNext(P, next);
        int i = 0, j = 0;
        while (i <= S->length - P->length)
        {   while (j == -1 || (j<P->length && S->str[i] == P->str[j]))
      { i++; j++;     }
      if (j == P->length)    return i - P->length;
        else j = next[j];
        }
        return -1;
}

int Replace(Stringtype &S,Stringtype T,Stringtype V);//将串 S 中所有子串 T 替换为 V,
并返回置换次数
{
   for(n=0,i=1;i<=Strlen(S)-Strlen(T)+1;i++) //注意 i 的取值范围
      if(!StrCompare(SubString(S,i,Strlen(T)),T)) //找到了与 T 匹配的子串
      { //分别把 T 的前面和后面部分保存为 head 和 tail
         StrAssign(head,SubString(S,1,i-1));
         StrAssign(tail,SubString(S,i+Strlen(T),Strlen(S)-i-Strlen(T)+1));
         StrAssign(S,Concat(head,V));
         StrAssign(S,Concat(S,tail)); //把 head,V,tail 连接为新串
         i+=Strlen(V); //当前指针跳到插入串以后
         n++;
      }//if
   return n;
}//Replace

int String_Palindrome(LString s){//判别给定串是否具有对称性的算法，并要求算法
的时间复杂度为 O(StringLength(S))。
      IntStack(stack);
      p=S.head;
      i=0;
      k=1;
      for(k=1;k<=S.Length){
            if(k<=S.Length/2)
                Push(S,p->ch[i]);
```

```
            else if(k>(S.Length)/2){
             Pop(S,c);
             if(p->ch[i]!=c)
                  return 0;
            }
        if(++i==CHUNKSIZE){
             p=p->next;
             i=0;
        }
    }
      return 1;
}
```

作业 7

作业

- 请大家结合前面的循环队列的数据结构、各种操 作算法，上机编程实现二叉树的层次遍历算法。

答案：

```
#include<stdio.h>
```

```
#include<stdlib.h>
typedef     struct    Node {
      datatype     data;
      struct Node *lchild,    *rchild,    *parent;
} BinTNode,*BinTree;

void LevelOrderTraverse(BinTree bt){ //层次遍历
    BinTreeNode Queue[MAXNODE];    /*定义队列*/
    int front, rear;
    if (bt==NULL) return; /*空二叉树，遍历结束*/ front=-1; rear=0;
    Queue[rear]=bt; /*根结点入队列*/
    while(rear!=front){ /*队列不空，继续遍历，否则，遍历结束*/
        front++;    /*出队*/
        visit(Queue[front]->data);    /*访问刚出队的元素*/
      if (queue[front]->lchild!=NULL){ /*如果有左孩子，左孩子入队*/
              rear++;
                Queue[rear]=Queue[front]->lchild;
        }
        if (queue[front]->rchild!=NULL){    /*如果有右孩子，右孩子入队*/
              rear++;
                Queue[rear]=Queue[front]->rchild;
        }
    }
}
```

作业 8

作业

- 编程实现先序、中序、后序非递归算法

答案：

```
#include<stdio.h>
#include<stdlib.h>
#define Max 100

typedef struct
{
    int data[Max];
    int top;
}stack;

typedef    struct    Node {
    datatype    data;
    struct Node *lchild,   *rchild,   *parent;
} BinTNode,*BinTree;


void PreOrderNoRec (BinTree BT)   { //前序遍历非递归算法
    stack S;
    BinTree p=BT->root;
    while ((p != NULL) || !StackEmpty(S)){
        if (p!=NULL){
            printf ("%c", p->data);   /*访问当前结点*/
              Push (S,p);                              /*将 p 压入栈 S*/
```

```
                p = p->lchild;
            }                   /*将 p 指向其左子树*/
          else{
               p = Top(S);
                Pop(S);                          /*从栈 S 弹出栈顶元素*/
                p = p->rchild;
            }          /*将 p 指向其右子树*/ }
}

void InOrderNoRec (BinTree BT)   { //中序遍历非递归算法
       stack S;
     BinTree p=BT->root;
     while ((p != NULL) || !StackEmpty(S))
     {
          if (p!=NULL){ Push (S,p);   /*将 p 压入栈 S*/
               p = p->lchild;
           }  /*将 p 指向其左子树*/
          else{
               p = Top(S);
                 Pop(S);   /*从栈 S 弹出栈顶元素*/
             printf ("%c", p->data);   /*访问当前结点*/
                p = p->rchild;
           }  /*将 p 指向其右子树*/
       }
}

void PostOrderNoRec (BinTree BT)   {       //后序遍历非递归算法
      stack S, tag;    BinTree p=BT->root;
      while ((p != NULL) || !StackEmpty(S)) {
            while (p!=NULL) {
               Push (S,p);
             Push (tag,0);
             p = p->lchild;
          }  /*扫描左子树*/
          if(!StackEmpty(S)){
               if (Pop(tag)==1) /*左右子树均访问*/
             {       p = Top(S);
                      Pop(S);
                   printf ("%c", p->data);
                   Pop(tag);
             }  /*访问并出栈*/
             else {
                      p=Top(s);
                   if(!StackEmpty(S))   /*扫描右子树*/
```

```
                    {       p = p->rchild;
                      Pop(tag);
                    Push(tag,1);
                }
            }
        }
    }
}
```

作业 9

课后作业题

- 仅知道二叉树的先序序列，能建立二叉树吗？
- 如果同时知道后序和中序呢？如何设计算法？

答案：

仅知道二叉树的先序序列，不能建立二叉树。

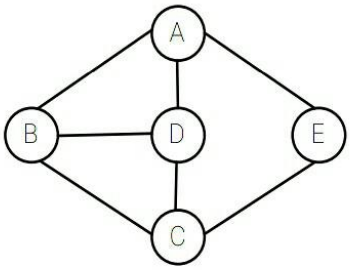
```
#include<stdio.h>
#include<stdlib.h>
#define Max 100

typedef struct
{
    int data[Max];
```

```
    int top;
}stack;

typedef    struct    Node {
    datatype    data;
    struct Node *lchild,   *rchild,   *parent;
} BinTNode,*BinTree;

void PreInOrd( char preord[], char inord[], int i, int j, int k, int h, BinTree t){
/*  先序序列中从 i 到 j，中序从序列从 k 到 h，建立一棵二叉树放在 t 中*/
    int m;
    (*t)=new BiNode;
    (*t)->data=preord[i];   /*二叉树的根*/
    m=k;
    while (inord[m]!=preord[i]) m++; /*在中序序列中定位树根*/
      /*递归调用建立左子树*/
    if(m==k) (*t)->lchild=NULL;      /*左子树空*/
    else PreInOrd(preord, inord, i+1, i+m-k, k, m-1, &((*t)->lchild)); /*递归调用建立
右子树*/
    if(m==h) (*t)->rchild=NULL;     /*右子树空*/
    else PreInOrd( preord, inord, i+m-k+1, j, m+1, h, &((*t)->lchild));
}
void CreateBinTree(char preord[], char inord[], int n, BinTree root){
    /* n  为二叉树结点的个数，建立的二叉树放在 root 中*/
    if(n<=0) root=NULL;
    else
    PreInOrd(preord, inord, 1, n,1, n, &root);
}
```

作业 10

课堂作业

对如下无向图：

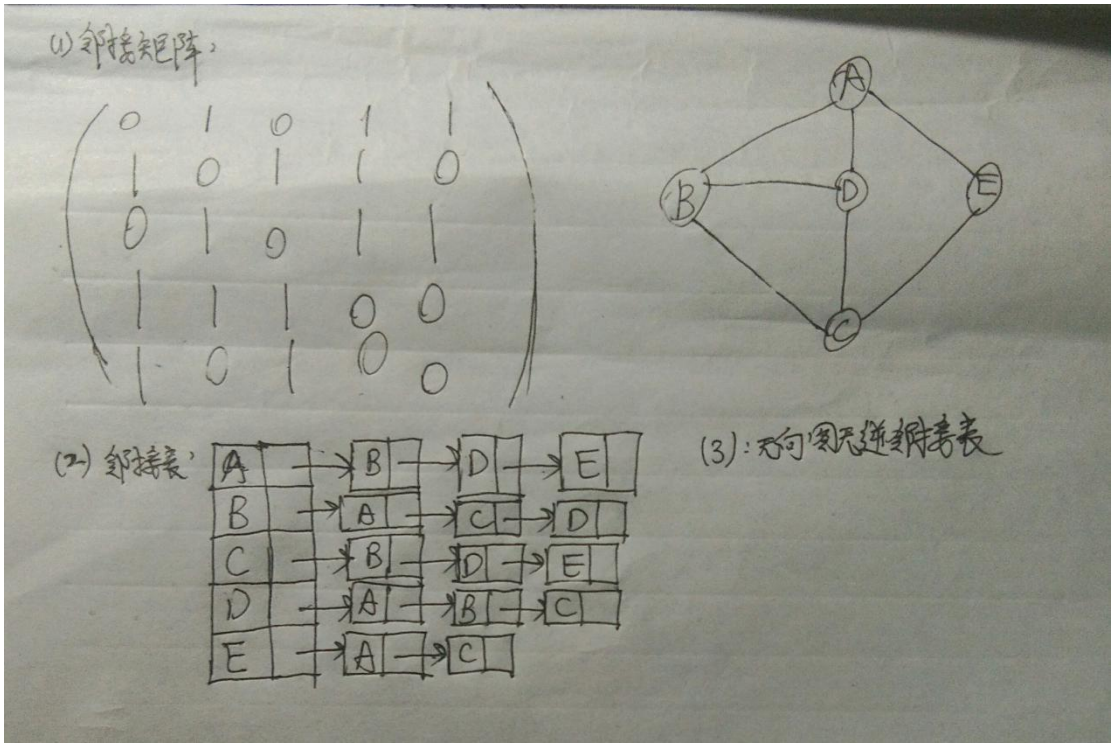


试画出：（1）邻接矩阵

（2）邻接表

（3）逆邻接表

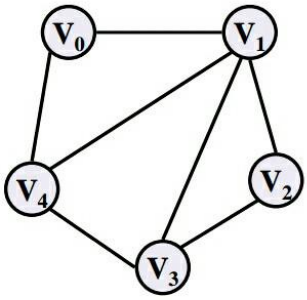
答案：



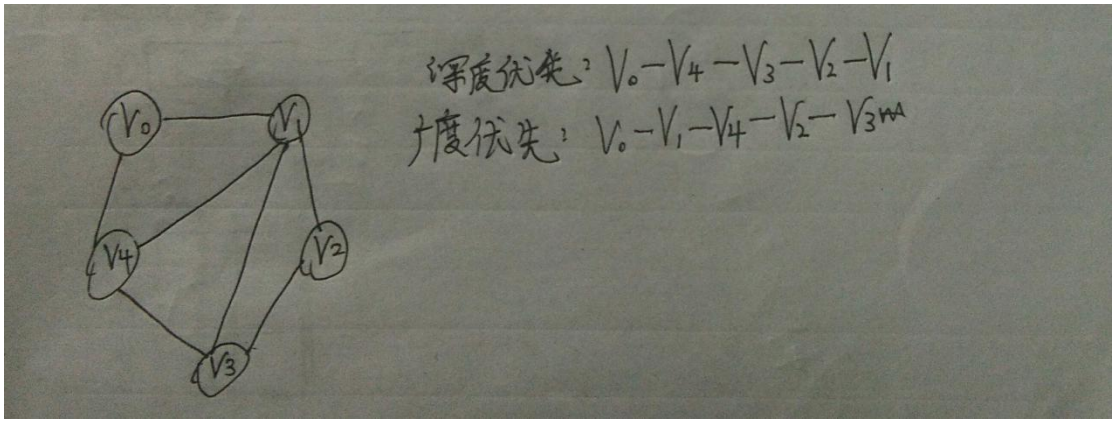
作业 11

图的遍历作业

求下图以V0起点的深度优先和广度优先序列:



答案:

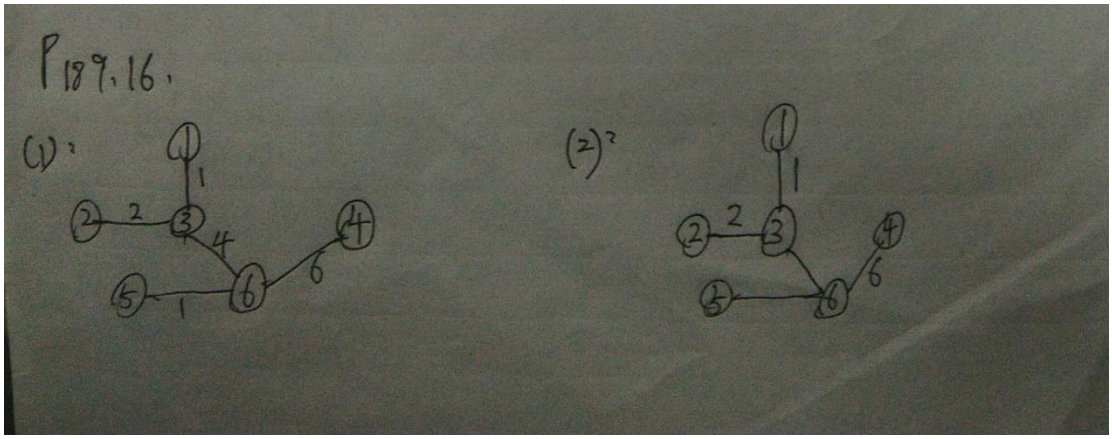


作业 12

课内作业

写出教材189页的第16题的最小生成树

答案：

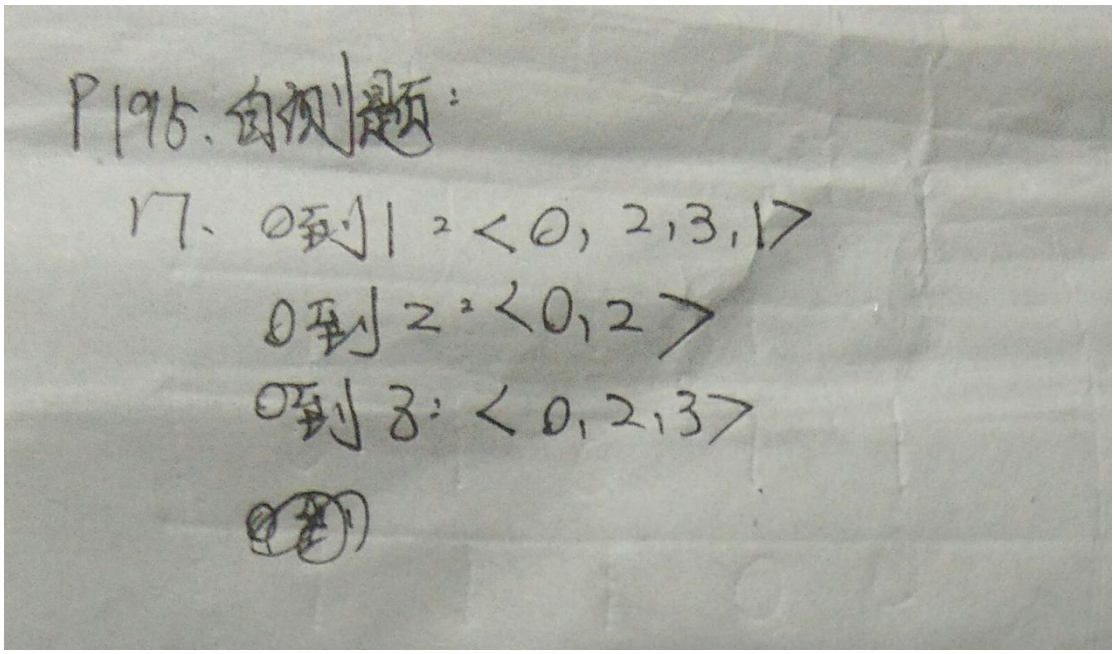


作业 13

最短路径作业

完成教材195页的自测题

答案：



作业 14

有向无环图的应用作业

完成教材204页的自测题

答案：