线性表，堆栈和队列

1.顺序表

顺序存储：

```
# include <stdio.h>

int a[10000];
int length;
int x,y;

void print ();           //遍历函数
int  search(int);       //查找函数
void insert(int ,int );     //插入函数
void Delete (int );         //删除函数
void change(int );          //修改函数
void sort ();              //排序函数

int main()
{
    scanf("%d",&length);
    for(int i=1;i<=length;i++)
    scanf("%d",&a[i]);

    print();

    printf("\n请输入您要查询的节点序号:");
    scanf("%d",&x);
    printf("%d",search(x));
    print();

    printf("\n请输入您要插入的序号以及数值");
    scanf("%d%d",&x,&y);
    insert (x,y);
    print();

    printf("\n请输入您要删除的序号");
    scanf("%d",&x);
    Delete (x);
    print();

    printf("\n请输入您要修改的序号:");
    scanf("%d",&x);
    change (x);
    print();

    sort ();
    print();
```

```
    return 0;

 }

void sort ()
{
    int flag=1;
    int temp;
    int l=length;
    while (flag)
    {
        flag=0;
        for(int i=1;i<=l-1;i++)
        if(a[i]>a[i+1])
        {
            flag=1;
            temp=a[i];
            a[i]=a[i+1];
            a[i+1]=temp;
        }
        l--;
    }
}
void change (int x)
{
    if(x<1||x>length)
    {
        printf("ERROR");
        return ;
    }
    printf("请输入修改后的值");
    scanf("%d",&a[x]);
}
void Delete (int x)
{
    if(x<1||x>length)
    {
        printf("ERROR");
        return ;
    }
    for(int i=x+1;i<=length;i++)
    a[i-1]=a[i];
    length--;
}
void insert (int x,int y)
{
    if(x<0||x>length)
    {
        printf("ERROR");
        return ;
    }
    for(int i=length;i>=x+1;i--)
    a[i+1]=a[i];
    a[x]=y;
    length++;
}
int search (int k)
{
```

```
    if(k<1||k>length)
    {
        printf("查询该节点不存在") ;
        return -1;
    }
    return a[k];
}

void print ()
{
    for(int i=1;i<=length;i++)
    printf("%d ",a[i]);
}
```

链接存储

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

int x;

typedef struct node
{
    int data;
    struct node *next;
}node,*nodelist;

# define setup (nodelist)malloc(sizeof(node))

void print (nodelist head)
{
    nodelist p=head->next;
    while(p!=NULL)
    {
        printf("%d ",p->data);
        p=p->next;
    }
    return ;
}
void createlist (nodelist & head)
{
    head=setup;
    int n;
    printf("您要创造的节点数:");
    scanf("%d",&n);
    nodelist tail,p;
    head->next=NULL;
    tail=head;
    int i=1;
    while(i<=n)
    {
        p=setup;
        scanf("%d",&p->data);
        p->next=NULL;
        tail->next=p;
```

```
        tail=p;
        i++;                //超级重要
    }
}

void insert (nodelist head,int k)   //在第k个节点后插入节点
{
  nodelist p=head; int i=0;

  while(i<k&&p!=NULL)
  {
      p=p->next;
      i++;
  }

  nodelist q;
  q=setup;
  scanf("%d",&q->data);
  q->next=p->next;
  p->next=q;

}

void Delete (nodelist head,int k)
{
    nodelist p=head;
    int i=0;
    while(i<k-1&&p!=NULL)
    {
        p=p->next;
        i++;
    }
    nodelist q=p->next;
    p->next=q->next;
    q->next=NULL;
    free (q);
    return ;
}

nodelist reverse (nodelist head)
{
    nodelist newhead;
    nodelist newtail;
    newhead=setup;
    newhead->next=NULL;
    newtail=newhead;

    nodelist q;
    nodelist p=head->next;
    while(p!=NULL)
    {
        q=setup;
        q->data=p->data;
        if(newhead->next==NULL)
        {
            q->next=NULL;
            newhead->next=q;
            newtail=q;
```

```
        }
        else
        {
        q->next=newtail;
        newhead->next=q;
        newtail=q;
        }
        p=p->next;
    }

    return newhead;
}
int main()
{
    nodelist psc;
    createlist (psc);
    print(psc);

    printf("请输入要删除节点的序号:");
    scanf("%d",&x);
    Delete(psc,x);
    print(psc);

    printf("请输入要插入的序号以及值");
    scanf("%d",&x);
    insert (psc,x);
    print(psc);

    printf("反转链表:");
    psc=reverse (psc);
    print(psc);

    return 0;
}
```

2.堆栈

顺序栈

```
# include <stdio.h>

const int size=100;
int top=0;
int stack[size];
int popnum;

void push (int data)          //入栈
{
    if(top==size)
    {
        printf("栈满无法压入.");
        return ;
    }
```

```
    stack[++top]=data;
}

void pop ()                     //弹栈
{
    if(top==0)
    {
        printf("栈空无法弹出");
        return ;
    }
    popnum=stack[top--];
}

int peek ()                     //存取栈顶元素
{
    if(top==0)
    {
        printf("栈空");
        return -1;
    }

    return stack[top];
}

int main ()
{
    int d;
    scanf("%d",&d);
    push(d);
    pop();
    printf("%d",peek());
    return 0;
 }
```

链接形式

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

typedef struct node       //定义链式栈
{
    int data;
    struct node *next;
}node,*nodelist;

//保存栈顶点的字段值
int tmp;

# define setup (nodelist) malloc (sizeof(node))

void  push(nodelist & top,int item)     //压栈
{
   nodelist p;
   p=setup;
   p->data=item;
```

```
    p->next=top;
    top=p;
}


void print(nodelist top)    //打印栈里的数值
{
    nodelist p=top;
    while(p!=NULL)
    {
        printf("%d ",p->data);
        p=p->next;
    }

    return ;
}

void  pop(nodelist & top)             //弹栈
{
    if(top==NULL)
    {
        printf("空栈不能弹出");
        return ;
    }
    nodelist q=top->next;
    free(top);
    top=q;

}


int peek (nodelist top)           //存取栈顶元素
{
    if(top==NULL)
    {
        printf("空栈无值");
        return -1 ;
    }
    return top->data;
 }


void clear(nodelist  &top)          //清空栈
{
    nodelist q;
    while(top!=NULL)
    {
        q=top->next;
        free(top);
        top=q;
    }

    return ;
}


int main()
{
```

```
    nodelist psc;
    psc=NULL;
    int data;

    for(int i=1;i<=5;i++)         //压栈
    {
    scanf("%d",&data);
    push(psc,data);
    }
    print(psc);

    pop(psc);                  //弹栈
    print(psc);

    printf("%d",peek(psc));       //取栈顶元素

    clear(psc);                   //栈清空
    print(psc);
    return 0;
}
```

3.队列

顺序队列

```
# include <stdio.h>


//保存删除的队首元素
int tmp;

//元素的数量
int count=0;

//队头指针和队尾指针
int front=1,rear=1;

//队列的长度
const int size=10;

int queue[100];

void insert(int data)
{
    if(count==size)
    {
        printf("队列已满");
        return ;
    }

    queue[rear]=data;
    rear=(rear+1)%size;
    count++;
```

```
}

void dele ()
{
    if(count==0)
    {
        printf("队列已空");
        return ;
    }
    tmp=queue[front];
    front=(front+1)%size;
    count--;
}

int  qfront()
{
    if(count==0)
    {
        printf("队列为空");
        return ;
    }

    return queue[front];

}

int main()
{
    int item;
    scanf("%d",&item);
    insert(item);
    dele();
    printf("%d",qfront());
    return 0;
}
```

链接存储

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

typedef struct node
{
    int data;
    struct node *next;
 } node,*nodelist;

# define setup (nodelist)malloc(sizeof(node))

void pushqueue (nodelist &front,nodelist &rear)    //从队尾增添元素
{
  nodelist p;
  p=setup;
  scanf("%d",&p->data);
```

```
  p->next=NULL;                 //每个节点要处理干净
  if(front==NULL)
  front=p;
  else
  rear->next=p;

  rear=p;
  return ;
}

void popqueue (nodelist &front)   //删除队首
{
    if(front==NULL)
    {
        printf("队列为空");
        return;
    }
    nodelist p=front;
    front=front->next;
    p->next=NULL;
    free(p);
    return ;
}

int fdata (nodelist &front)
{
    if(front==NULL)
    {
        printf("队列为空");
        return -1;
    }
    return front->data;
 }

void print (nodelist front)
{
    nodelist p=front;
    while(p!=NULL)
    {
        printf("%d ",p->data);
        p=p->next;
    }
    return ;
}
int main()
{
    nodelist front ,rear;
    front=NULL;
    rear=NULL;
    int n;
    printf("请输入您要创建的节点数目");
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
    pushqueue(front,rear);

    print(front);
    popqueue(front);
    print(front);
```

```
    printf("队首的元素为%d",fdata(front));
    return 0;
}
```

例题

4.大整数加法

```
#include <stdio.h>
#include <string.h>
char s1[1000],s2[1000];
int a[1000],b[1000],c[1000];
int x;


int main()
{
    scanf("%s",s1);
    scanf("%s",s2);
    int lena=strlen(s1);
    int lenb=strlen(s2);

    for(int i=1;i<=lena;i++)
    a[i]=s1[lena-i]-48;
    for(int i=1;i<=lenb;i++)
    b[i]=s2[lenb-i]-48;

    int lenc=1;
    while(lenc<=lena||lenc<=lenb)
    {
        c[lenc]=a[lenc]+b[lenc]+x;
        x=c[lenc]/10;
        c[lenc]%=10;
        lenc++;
    }
    c[lenc]=x;
    while(c[lenc]==0&&lenc>1) lenc--;
    for(int i=lenc;i>=1;i--)
    printf("%d",c[i]);
    return 0;
 }
```

5.火车入栈

思路：把小于出栈顺序的统统压入栈内

判断栈顶和出栈元素

1.深度搜索

```
# include <stdio.h>
int stack[1000];
int data[1000];
int vis[1000];
int n;

void check ()
{
    int top=0;
    int k=1;
    for(int i=1;i<=n;i++)
    {
        while(k<=data[i])
        {
            stack[++top]=k;
            k++;
        }
        if(data[i]==stack[top])
        top--;
        else
        return ;
    }
    for(int i=1;i<=n;i++)
    printf("%d ",data[i]);
    printf("\n");
    return ;
}

void dfs (int t)
{
    for(int i=1;i<=n;i++)
    if(vis[i]==0)
    {
        data[t]=i;
        vis[i]=1;
        if(t==n)  check();
        dfs(t+1);
        vis[i]=0;
    }
}

int main()
{
    printf("请输入一共有多少辆火车:");
    scanf("%d",&n);

    dfs(1);
    return 0;

}
```

2.判断给出的是否合理

```c
# include <stdio.h>
int stack[1000];
int data[1000];
int vis[1000];
int n;

void check ()
{
    int top=0;
    int k=1;
    for(int i=1;i<=n;i++)
    {
        while(k<=data[i])
        {
            stack[++top]=k;
            k++;
        }
        if(data[i]==stack[top])
        top--;
        else
        return ;
    }
    for(int i=1;i<=n;i++)
    printf("%d ",data[i]);
    printf("\n");
    return ;
}

void dfs (int t)
{
    for(int i=1;i<=n;i++)
    if(vis[i]==0)
    {
        data[t]=i;
        vis[i]=1;
        if(t==n)  check();
        dfs(t+1);
        vis[i]=0;
    }
}

int main()
{
    printf("请输入一共有多少辆火车:");
    scanf("%d",&n);

    dfs(1);
    return 0;

}
```

6.括号匹配

用top和flag判断是否匹配

flag用来判断是否第一个就是右括号

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

char str[1000];
int stack[1000];

int main()
{
  gets(str);
  int i=0;
  int top=0;
  int flag=1;
  while(str[i]!='\0')
  {
    if(str[i]=='(')
    stack[++top]=1;
    if(str[i]=='[')
    stack[++top]=2;
    if(str[i]=='{')
    stack[++top]=3;

   if(str[i]==')')
   if(stack[top]==1)
   top--;
   else
   flag=0;

   if(str[i]==']')
   if(stack[top]==2)
   top--;
   else
   flag=0;

   if(str[i]=='}')
   if(stack[top]==3)
   top--;
   else
   flag=0;

   i++;
  }
  if(top==0&&flag==1)
  printf("括号匹配");
  else
  printf("括号不匹配");
  return 0;
}
```

7.约瑟夫环

```c
# include <stdio.h>

int next[100];


int main()
{
    int n,m;
    scanf("%d%d",&n,&m);

    for(int i=1;i<=n-1;i++) next[i]=i+1;
    next[n]=1;

    int i=1,k=1,j=1;
    while(k<=n-1)
    {
        i=1;
        while(i<=m-2)      //走向m-1的标号走几步
        {
          j=next[j];
          i++;
        }

        printf("编号为%d的人死去\n",next[j]);
        next[j]=next[next[j]];
        j=next[j];
        k++;
    }


    printf("最后编号为%d的人活了下来",j);

    return 0;
 }
```

8.一元多项式相加

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

typedef struct node
{
    int datap;
    int datax;
    struct node *next;
 } node,*nodelist;

 # define setup (nodelist)malloc(sizeof(node))

 char c='A';
 //链表的长度
 int n;
```

```
nodelist create(nodelist head)
{
    nodelist tail,p;
    head=setup;
    head->next=NULL;
    tail=head;
    printf("请输入多项式%c的项数\n",c);
    scanf("%d",&n);
    printf("请输入多项式%c\n",c++);
    int i=1;
    while(i<=n)
    {
        p=setup;
        scanf("%d",&p->datap);
        scanf("%d",&p->datax);
        p->next=NULL;
        tail->next=p;
        tail=p;
        i++;
    }
    return head;
}


void print(nodelist head)    //打印多项式
{
    nodelist p=head->next;
    int flag=1;

    while(p!=NULL)
    {
        if(p->datap==0)
        {
          p=p->next;
          continue;
          flag=0;
        }
        if(p->datap>0&&flag==1)
        {
          printf("%d",p->datap);
        }
        else if(p->datap<0)
        printf("%d",p->datap);
        else
        printf("+%d",p->datap);


        printf("x^%d",p->datax);

        p=p->next;
        flag=0;
    }

    return ;
}


void sort(nodelist head)    //按照系数的大小排序
```

```
{
    int tmpp,tmpx;

    for(int i=1;i<=n;i++)
 {
    nodelist p=head->next;
    nodelist q=p->next;
    while(q!=NULL)
    {
        if(p->datax<q->datax)
        {
            tmpp=p->datap;
            p->datap=q->datap;
            q->datap=tmpp;

            tmpx=p->datax;
            p->datax=q->datax;
            q->datax=tmpx;
        }
        q=q->next;
        p=p->next;
    }
}
 }


nodelist plus(nodelist head,nodelist s1head,nodelist s2head)      //将两个链表相加
{
    nodelist tail;
    head=setup;
    head->next=NULL;
    tail=head;

    nodelist p=s1head->next;
    nodelist q=s2head->next;
    nodelist m;

    while(p!=NULL)
    {
        m=setup;
        m->datap=p->datap;
        m->datax=p->datax;
        m->next=NULL;
        tail->next=m;
        tail=m;
        p=p->next;
    }


    while(q!=NULL)
    {
        m=setup;
        m->datap=q->datap;
        m->datax=q->datax;
        m->next=NULL;
        tail->next=m;
        tail=m;
        q=q->next;
```

```
    }

   return head;
}

nodelist order (nodelist head)            //将系数相同的项合并
{
    nodelist p=head->next;
    nodelist q;
    nodelist pre;
    while(p!=NULL)
    {
        pre=p;
        q=p->next;
        while(q!=NULL)
      {
            if(p->datax==q->datax)
        {
            p->datap+=q->datap;
            pre->next=q->next;
            q=pre->next;
        }
        else
        {

        pre=q;
        q=q->next;
       }
      }


        p=p->next;
    }

    return head;
}


int main()
{
    nodelist psc1,psc2,psc3;
    psc1=create(psc1);
    psc2=create(psc2);

    psc3=plus(psc3,psc1,psc2);
    psc3=order(psc3);
    sort(psc3);
    print(psc3);

}
```

9.中缀转后缀

要点：

1.左括号直接入栈

2.右括号一直弹栈直到遇到左括号

3.要想入栈需要与栈顶元素相比较，优先级高才能压入

4.最后将栈内剩余元素弹出

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

char str[100];
char stack[1000];
int top=0;

int main()
{

    scanf("%s",str+1);
    int len=strlen(str+1);

    for(int i=1;i<=len;i++)                                  //遍历过程
    {
        if(str[i]>='0'&&str[i]<='9')                         //1.数字直接输出
        printf("%c",str[i]);

        if(str[i]=='*'||str[i]=='/')                         //2. 乘,除,左括号压栈
        {
           while(stack[top]=='/'||stack[top]=='*')
           printf("%c",stack[top--]);

           stack[++top]=str[i];
        }

        if(str[i]=='(')
        stack[++top]=str[i];

        if(str[i]=='+'||str[i]=='-')                         //3.加减判断栈顶
        {

while(stack[top]=='/'||stack[top]=='*'||stack[top]=='+'||stack[top]=='-')    //等于或者小于优先级弹栈
            printf("%c",stack[top--]);

            stack[++top]=str[i];                             //然后再压栈
        }



        if(str[i]==')')                                      //4.右括号和左括号之间的全部弹栈
        {
            while(stack[top]!='(')
            printf("%c",stack[top--]);
```

```
            top--;
        }

    }

    while(top!=0)                                   //5.栈内剩余符号全部出栈
    printf("%c",stack[top--]);

    return 0;
}
```

10.后缀表达式的计算

要点:

1.数字直接压入栈内

2.遇到符号进行运算

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

char str[1000];
int stack[1000];
int top;

int main()
{
   gets(str+1);
   int len=strlen(str+1);

   for(int i=1;i<=len;i++)
   {
      if(str[i]>='0'&&str[i]<='9')
      stack[++top]=str[i]-48;

      if(str[i]=='+')
      stack[--top]+=stack[top+1];
      if(str[i]=='-')
      stack[--top]-=stack[top+1];
      if(str[i]=='*')
      stack[--top]*=stack[top+1];
      if(str[i]=='/')
      stack[--top]/=stack[top+1];
   }

   printf("%d",stack[1]);
   return 0;
}# include <stdio.h>
# include <string.h>

char str[1000];
```

```
int stack[1000];
int top;

int main()
{
   gets(str+1);
   int len=strlen(str+1);

   for(int i=1;i<=len;i++)
   {
      if(str[i]>='0'&&str[i]<='9')
      stack[++top]=str[i]-48;

      if(str[i]=='+')
      stack[--top]+=stack[top+1];
      if(str[i]=='-')
      stack[--top]-=stack[top+1];
      if(str[i]=='*')
      stack[--top]*=stack[top+1];
      if(str[i]=='/')
      stack[--top]/=stack[top+1];
   }

   printf("%d",stack[1]);
   return 0;
}
```

树

二叉树

1.二叉树的建立以及遍历

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

typedef struct bitnode
{
    char data;
    struct bitnode *left,*right;
}bitnode,*bitree;

void createtree (bitree &t)        //主要要加引用符号 （变量）
{
    char ch;
    scanf("%c",&ch);
```

```
    if(ch=='#')
    t=NULL;
    else
    {
        t=new bitnode;
        t->data=ch;                          //先根创建
        createtree(t->left);
        createtree(t->right);
    }

}


void preordertraverse(bitree t)        //（指针形式）
{
    if(t==NULL) return ;
    printf("%c ",t->data);
    preordertraverse(t->left);
    preordertraverse(t->right);
}

void inordertraverse(bitree t)
{
    if(t==NULL)  return ;
    inordertraverse(t->left);
    printf("%c ",t->data);
    inordertraverse(t->right);
}

void postordertraverse (bitree t)
{
    if(t==NULL) return ;
    postordertraverse(t->left);
    postordertraverse(t->right);
    printf("%c ",t->data);
}
int main()
{
    bitree root;
    createtree (root);
    printf("二叉树的先根遍历为:");
    preordertraverse(root);
    printf("\n二叉树的中根遍历为:");
    inordertraverse(root);
    printf("\n二叉树的后根遍历为:");
    postordertraverse(root);
    return 0;
 }
```

2.二叉树的非递归中序遍历以及后序遍历

非递归中序遍历

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

typedef struct bitnode
{
    char data;
    struct bitnode *left,*right;
 } bitnode,*bitree;


void createtree (bitree &t)        //前序建立二叉树
{
    char ch;
    scanf("%c",&ch);
    if(ch=='#')
    t=NULL;
    else
    {
        t=new bitnode;
        t->data=ch;
        createtree(t->left);
        createtree(t->right);
    }
}


void nio (bitree t)
{
    bitree stack[100];       //初始化
    int top=0;
    bitree p=t;

    while(1)
    {
        while(p!=NULL)         //入栈
        {
            stack[++top]=p;
            p=p->left;
        }
        if(top==0)  return ;
        else p=stack[top--];
        printf("%c",p->data);  //访问
        p=p->right;
    }

}

int main()
{
    bitree root;
    createtree(root);
    printf("二叉数的非递归中根遍历为:");
    nio(root);
}
```

非递归后序遍历

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

typedef struct bitnode
{
    char data;
    struct bitnode *left,*right;
}bitnode,*bitree;

typedef struct stack                    //栈
{
    bitree  pointer;
    int num;
}stack;

void createtree (bitree &t)    //前序建立二叉树
{
    char ch;
    scanf ("%c",&ch);
    if(ch=='#')
    t=NULL;
    else
    {
        t= new bitnode;
        t->data=ch;
        createtree(t->left);
        createtree(t->right);
    }
}

void npo (bitree t)
{
    if(t==NULL)  return ;
    stack s[100]; int top=0;
    bitree p=t; int i=0;
    s[++top].pointer=p; s[top].num=i;            //根先入栈

    while(top!=0)
    {
        p=s[top].pointer; i=s[top--].num;                    //弹栈
        if(i==0)
        {
            s[++top].pointer=p; s[top].num=1;                //入栈，次数改为1    左
            if(p->left!=NULL)
            {
                s[++top].pointer=p->left; s[top].num=0;
             }
        }else
        if(i==1)
        {
            s[++top].pointer=p; s[top].num=2;                  //入栈，次数改为2 右
```

```
            if(p->right!=NULL)
            {
                s[++top].pointer=p->right; s[top].num=0;
             }
        }else
        if(i==2)                                                    //打印
根
        printf("%c ",p->data);
    }


 }


int main()
{
    bitree root;
    createtree(root);
    npo(root);
    return 0;
}
```

3.二叉树的层次遍历

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

typedef struct bitnode
{
    char data;
    struct bitnode *left,*right;
 } bitnode,*bitree;


void createtree (bitree &t)
{
    char ch;
    scanf("%c",&ch);
    if(ch=='#')
    t=NULL;
    else
    {
        t=new bitnode;
        t->data=ch;
        createtree (t->left);
        createtree (t->right);
    }
}

void levelorder (bitree t)
```

```
{
    bitree queue[100];
    bitree p=t;
    int front=1,rear=1;
    queue[rear++]=t;

    while(rear-front!=0)
    {
       p=queue[front++];
       printf("%c ",p->data);
       if(p->left!=NULL)   queue[rear++]=p->left;
       if(p->right!=NULL)   queue[rear++]=p->right;
    }
}


int main()
{
    bitree root;
    createtree (root);
    printf("二叉树的层次遍历为:");
    levelorder(root);
    return 0;
}
```

4.二叉树的其它操作

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

typedef struct bitnode
{
    char data;
    struct bitnode *left,*right;
}bitnode,*bitree;

void createtree (bitree &t)        //前序创建一个二叉树
{
    char ch;
    scanf("%c",&ch);
    if(ch=='#')
    t=NULL;
    else
    {
        t= new bitnode;
        t->data=ch;
        createtree (t->left);
        createtree (t->right);
    }
}
```

```
bitree  copytree (bitree t)
{
    bitree nt,newlptr,newrptr;        //每个节点
    if(t==NULL)  return NULL;

    if(t->left!=NULL)                   //复制左子树
    newlptr=copytree (t->left);
    else  newlptr=NULL;

    if(t->right!=NULL)                  //复制右子树
    newrptr=copytree (t->right);
    else  newrptr=NULL;
    nt=new bitnode;                     //拼接
    nt->data=t->data;
    nt->left=newlptr;
    nt->right=newrptr;
    return nt;
 }

bitree father (bitree t,bitree p)
{
    if(t==NULL||p==NULL)   return NULL;

    if(t->left==p||t->right==p)  return t;

    bitree f;
    f=father (t->left,p);           //查找左子树
    if(f!=NULL)  return f;
    f=father (t->right,p);          //查找右子树
    if(f!=NULL)  return f;

    return NULL;


}


bitree find (bitree t,char d)      //找节点
{
    if(t==NULL)  return NULL;

    if(t->data==d)  return t;

    bitree f;
    f=find(t->left,d);
    if(f!=NULL)  return f;
    f=find(t->right,d);
    if(f!=NULL)   return f;

    return NULL;
}

void del (bitree &t)
{
    if(t==NULL)
    return ;
```

```
    del(t->left);    //释放左子树
    del(t->right);   //释放右子树
    delete(t);         //释放节点
    t=NULL;
}

void insert (bitree p,bitree s)    //插入节点p,作为节点s的左儿子，s原来的左儿子成为p的左儿子
{
    if(p==NULL||s==NULL)  return ;

    p->left=s->left;
    s->left=p;
}

void delnode (bitree &root,bitree t)                //删除节点以及子树
{
    if(t==NULL)  return ;
    if(t==root)
    {
        del(t);
        root=NULL;
        return ;
    }

    bitree fth;
    fth=father (root,t);

    if(fth->left==t)  fth->left=NULL;
    if(fth->right==t) fth->right=NULL;
    del(t);

 }

void levelorder (bitree t)
{
    if(t==NULL)  return ;
    bitree q;
    bitree queue[100];
    int front=1,rear=1;
    queue[rear++]=t;

    while(rear-front!=0)
    {
        q=queue[front++];
        printf("%c ",q->data);
        if(q->left!=NULL)  queue[rear++]=q->left;
        if(q->right!=NULL) queue[rear++]=q->right;
    }
}

int main()
{
    bitree root,newroot,p,tmp;
    //1.创建二叉树
    createtree(root);
    createtree(newroot);
    //2.找节点
```

```
    char ch;
    scanf("%c",&ch);
    tmp=find(root,ch);
    printf("%c ",tmp->data);
    //3.找爸爸
    tmp=father(root,tmp);
    printf("%c ",tmp->data);
    //4.复制二叉树
    newroot=copytree(root);
    levelorder (newroot);
    //5.删除节点以及子树
    delnode(root,tmp);
    levelorder (root);
    //6.释放树
    del(root);
    levelorder(root);
    //7.插入节点
    insert(root,newroot);
    levelorder(newroot);
 }
```

5.二叉树的最长路径

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

typedef struct bitnode
{
    char data;
    struct bitnode *left,*right;
}bitnode,*bitree;

void createtree (bitree &t)
{
    char ch;
    scanf("%c",&ch);
    if(ch=='#')
    t=NULL;
    else
    {
      t=new bitnode;
      t->data=ch;
      createtree (t->left);
      createtree (t->right);
    }
}

int depth (bitree t)
{
    if(t==NULL) return 0;            //到二叉树的底部了
    return 1+(depth(t->left)>depth(t->right)?depth(t->left):depth(t->right));
     //=取一个节点的左子树和右子树的高度的较大值
}
```

```
void longestpath (bitree t)
{
    if(t==NULL)  return ;

    printf("%c ",t->data);         //访问节点
    if(depth(t->left)>depth(t->right))              //判断往左走还是往右走
    longestpath(t->left);
    else
    longestpath(t->right);
 }

int main()
{
    bitree root;
    createtree (root);
    printf("二叉树的最大路径长度为%d \n路径上各节点分别为：",depth(root));
    longestpath (root);
    return 0;
}
```

6.二叉树边的个数

```
# include <stdio.h>
int  num;          //二叉树的节点个数
typedef struct bitnode
{
    char data;
    struct  bitnode *left,*right;
 } bitnode,*bitree;

void createtree (bitree &t)
{
    char ch;
    scanf("%c",&ch);
    if(ch=='#')
    t=NULL;
    else
    {
        t=new bitnode;
        t->data=ch;
        createtree (t->left);
        createtree (t->right);
    }
}

void edgenum (bitree t)
{
    if(t==NULL)  return ;

    num++;
    edgenum(t->left);
    edgenum(t->right);
```

```
}
int main()
{
    bitree root;
    createtree(root);
    edgenum (root);
    printf("该二叉树共有%d个边",num-1);
    return 0;
 }
```

7.复制二叉树

```
# include <stdio.h>
int  num;          //二叉树的节点个数
typedef struct bitnode
{
    char data;
    struct  bitnode *left,*right;
 } bitnode,*bitree;

void createtree (bitree &t)
{
    char ch;
    scanf("%c",&ch);
    if(ch=='#')
    t=NULL;
    else
    {
        t=new bitnode;
        t->data=ch;
        createtree (t->left);
        createtree (t->right);
    }
}

void edgenum (bitree t)
{
    if(t==NULL)  return ;

    num++;
    edgenum(t->left);
    edgenum(t->right);

}
int main()
{
    bitree root;
    createtree(root);
    edgenum (root);
    printf("该二叉树共有%d个边",num-1);
```

```
    return 0;
 }
```

8.检验是否是完全二叉树

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

typedef struct bitnode
{
    char data;
    struct bitnode *left,*right;
}bitnode,*bitree;

void createtree (bitree &t)
{
    char ch;
    scanf("%c",&ch);
    if(ch=='#')
    t=NULL;
    else
    {
        t=new bitnode;
        t->data=ch;
        createtree (t->left);
        createtree (t->right);
    }
}

int check (bitree t)
{
    bitree queue[1000];
    if(t==NULL) return 0;       //空树不符合条件
    int front=1,rear=1;
    queue[rear++]=t;
    bitree q=queue[front++];
    while(q!=NULL)                  //采用层次遍历      遇到空停
    {
        queue[rear++]=q->left;          //左子树 右子树都进入队列
        queue[rear++]=q->right;
        q=queue[front++];
    }
    while(rear-front!=0)            //检测空后还有没有非空节点
    {
        q=queue[front++];
        if(q!=NULL)  return 0;
     }
    return 1;
}

int main()
```

```
{
  bitree root;
  createtree (root);
  if(check(root))
  printf("yes") ;
  else
  printf("NO");
  return 0;
}
```

9.检验二叉树是否相似

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

typedef struct bitnode
{
    char data;
    struct bitnode *left,*right;
}bitnode,*bitree;

void createtree (bitree &t)
{
    char ch;
    scanf("%c",&ch);
    if(ch=='#')
    t=NULL;
    else
    {
        t=new bitnode;
        t->data=ch;
        createtree (t->left);
        createtree (t->right);
    }
}

int issame (bitree t1,bitree t2)
{
    if(t1==NULL&&t2==NULL)
    return 1;
    else  if(t1==NULL||t2==NULL)
    return 0;
    else
    {
        int left=issame(t1->left,t2->left);
        int right=issame(t1->right,t2->right);
        return left&&right;
    }
 }

void preorder (bitree t)
{
    if(t==NULL)  return ;
```

```
    printf("%c ",t->data);
    preorder (t->left);
    preorder (t->right);
}
int main()
{
  bitree root1,root2;
  createtree (root1);              //输入时连续输入
  createtree (root2);
  int flag=issame (root1,root2);
  if(flag)
  printf("yes");
  else
  printf("no");
  return 0;
}
```

图

1.图的建立以及深度优先搜索算法（有向 无向均可）

两个结构体： 边结点和顶点结点

三个数组：头结点数组 尾结点数组 标志数组

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

typedef struct edgenode           //边结点
{
    int  veradj;                 //存放v的邻接顶点在顶点表中的序号
    int  cost;                    //边的权值
    struct edgenode *link;       //指向下一个结点的指针
}edgenode;

typedef struct vertexnode     //顶点结点
{
    char vername;          //该顶点的名称
    edgenode *adjacent;      //指针
}vertexnode;

edgenode *Tail[100];        //链表的尾指针
vertexnode Head [100];
int visited[100];       //辅助数组，表示标志顶点是否被访问
# define setup (edgenode *)malloc(sizeof(edgenode))


void  createedge (int f,int t,int w)     //创造单向图
```

```
{
    edgenode *p;
    p=setup;
    p->veradj=t;
    p->cost=w;
    p->link=NULL;
    if(Head[f].adjacent==NULL)
    {
      Head[f].adjacent=p;
      Tail[f]=p;
    }
    else
    {
        Tail[f]->link=p;
        Tail[f]=p;
    }

    //创造无向图
    p=setup;
    p->veradj=f;
    p->cost=w;
    p->link=NULL;
    if(Head[t].adjacent==NULL)
    {
      Head[t].adjacent=p;
      Tail[t]=p;
    }
    else
    {
        Tail[t]->link=p;
        Tail[t]=p;
    }

}
//深度优先遍历
void DFS (int v)      //V表示遍历的起始顶点      //非连通图还要遍历结点
{
    edgenode *p;
    printf("%d ",v);
    visited[v]=1;
    p=Head[v].adjacent;
    while(p!=NULL)
    {
        if(visited[p->veradj]!=1)
        DFS(p->veradj);
        p=p->link;
    }
}
int main()
{
    int from,to,w;    //边的元素
    //创建图
    int n;           //共有多少条边
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
    {
        scanf("%d%d%d",&from,&to,&w);
        createedge(from,to,w);
```

```
    }

/*  edgenode *p=Head[1].adjacent;            //check;
    while(p!=NULL)
    {
        printf("%d ",p->veradj);
        p=p->link;
     }

     printf("\n");*/
     DFS(1);
     return 0;
}
```

2.广度优先搜索（有向 无向均可）

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

typedef struct edgenode
{
    int veradj;
    int cost;
    struct edgenode *link;
} edgenode;

typedef struct vertexnode
{
    char vername;
    edgenode *adjacent;
} vertexnode;

edgenode *Tail[100];
vertexnode Head [100];
int visited[100];

# define setup (edgenode *)malloc(sizeof(edgenode))

void createedge (int f,int t,int w)
{
    edgenode *p;
    p=setup;
    p->veradj=t;
    p->cost=w;
    p->link=NULL;
    if(Head[f].adjacent==NULL)
    {
      Head[f].adjacent=p;
      Tail[f]=p;
    }
    else
```

```
    {
        Tail[f]->link=p;
        Tail[f]=p;
    }
 }

void BFS (int v)      //广度优先遍历
{
    edgenode *p;
    int queue[100];
    int front=1,rear=1;
    printf("%d ",v);
    visited[v]=1;
    queue[rear++]=v;
    while(rear-front!=0)
    {
       v=queue[front++];
       p=Head[v].adjacent;
       while(p!=NULL)
       {
         if(visited[p->veradj]==0)
         {
            printf("%d ",p->veradj);
            visited[p->veradj]=1;
            queue[rear++]=p->veradj;
         }
         p=p->link;
       }
    }
}
int main()
{
    int from,to,w;
    int n;
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
    {
        scanf("%d%d%d",&from,&to,&w);
        createedge(from,to,w);
    }
    BFS(1);
    return 0;
}
```

3.拓扑排序（有向图）

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

typedef struct edgenode
{
```

```
    int veradj;
    int cost;
    struct edgenode *link;
}edgenode;

typedef struct vertexnode
{
    char vername;
    edgenode *adjacent;
}vertex;

# define setup (edgenode *)malloc(sizeof(edgenode))

vertexnode Head [100];
edgenode *Tail[100];
int visited [100];
int count [100];    //模拟栈     存储次栈顶值
int top;

void createedge (int f,int t,int w)
{
    edgenode *p;
    p=setup;
    p->veradj=t;
    p->cost=w;
    p->link=NULL;
    if(Head[f].adjacent==NULL)
    {
        Head[f].adjacent=p;
        Tail[f]=p;
    }else
    {
        Tail[f]->link=p;
        Tail[f]=p;
    }



}

void topoorder (int n)              //n表示顶点个数
{
    int j,k;
    edgenode *p;
    for (int i=1;i<=n;i++)
    count[i]=0;
    for(int i=1;i<=n;i++)
    {
        p=Head[i].adjacent;
        while(p!=NULL)
        {
            count[p->veradj]=count[p->veradj]+1;
            p=p->link;
        }
    }

    top=-1;
    for(int i=1;i<=n;i++)
```

```
    if(count[i]==0)
    {
        count[i]=top;
        top=i;
    }


    for(int i=1;i<=n;i++)
    if(top==-1)
    {
        printf("There is a cycle in network!");
        return ;
    }
    else
    {
        j=top;
        top=count[top];
        printf("%d ",j);
        p=Head[j].adjacent;
        while(p!=NULL)
        {
          k=p->veradj;
          count[k]--;
          if(count[k]==0)
          {
            count[k]=top;
            top=k;
          }
          p=p->link;
        }
    }
}

int main()
{
    int m,n,from,to,w;
    scanf("%d",&n);   //边的个数
    scanf("%d",&m);   //点的个数
    for(int i=1;i<=n;i++)
    {
        scanf("%d%d%d",&from,&to,&w);
        createedge(from,to,w);
     }
     topoorder (8);
     return 0;

}
```

4.关键路径（有向图）

在求关键路径之前已经对诸顶点实现了拓扑排序，并按拓扑排序对各顶点重新编号。

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

typedef struct edgenode
{
    int veradj;
    int cost;
    struct edgenode *link;
}edgenode;

typedef struct vertexnode
{
    char vername;
    edgenode *adjacent;
}vertexnode;

# define setup (edgenode *)malloc(sizeof(edgenode))

edgenode *Tail[100];
vertexnode Head[100];
int ve[100],vl[100];
int e,l;

void criticalpath (int n)              //n个结点
{
    //计算事件的最早发生时间
    int k;
    edgenode *p;
    for(int i=1;i<=n;i++)
    ve[i]=0;
    for(int i=1;i<=n-1;i++)
    {
        p=Head[i].adjacent;
        while(p!=NULL)                 //找最长路径
        {
            k=p->veradj;
            if(ve[i]+p->cost>ve[k])
            ve[k]=ve[i]+p->cost;
            p=p->link;
        }
    }

    //计算时间的最迟发生时间

    for(int i=1;i<=n;i++)
    vl[i]=ve[n];
    for(int i=n-1;i>=1;i--)
    {
        p=Head[i].adjacent;
        while(p!=NULL)
        {
            k=p->veradj;
            if(vl[k]-p->cost<vl[i])
            vl[i]=vl[k]-p->cost;
            p=p->link;
        }
    }
```

```
    //求诸活动的最早开始时间和最迟开始时间
    for(int i=1;i<=n;i++)
    {
      p=Head[i].adjacent;
      while(p!=NULL)
      {
        k=p->veradj;
        e=ve[i];
        l=vl[k]-p->cost;
        if(e==l)
        printf("<%d,%d>",i,k);
        p=p->link;
      }
    }
}

void createedge (int f,int t,int w)
{
    edgenode *p;
    p=setup;
    p->cost=w;
    p->veradj=t;
    p->link=NULL;
    if(Head[f].adjacent==NULL)
    {
        Head[f].adjacent=p;
        Tail[f]=p;
    }
    else
    {
        Tail[f]->link=p;
        Tail[f]=p;
    }
}

int main()
{
    int from,to,w;
    int m,n;  //m个点 n个边
    scanf("%d%d",&m,&n);
    for(int i=1;i<=n;i++)
    {
        scanf("%d%d%d",&from,&to,&w);
        createedge(from,to,w);
     }
     printf("该图的关键路径为:");
     criticalpath(m);
     return 0;
}
```

5.最短路径问题

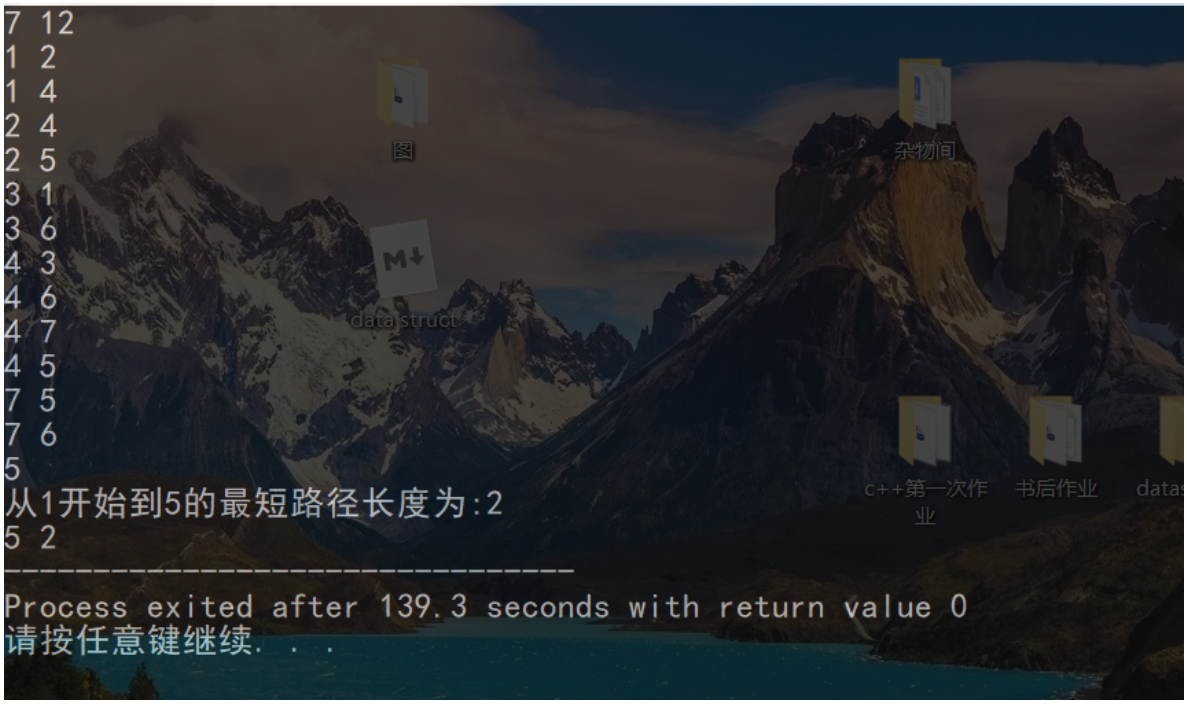
无权最短路径问题

寻找最短路径的次序与按层次进行的**图的广度优先遍历次序**是一致的

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

typedef struct edgenode
{
    int veradj;
    int cost;
    struct edgenode *link;
}edgenode;

typedef struct vertexnode
{
    char vername;
    edgenode *adjacent;
}vertexnode;

# define setup (edgenode *)malloc(sizeof(edgenode))

edgenode *Tail[100];
vertexnode Head[100];
int path[1000],dist[1000];

void createedge (int f,int t,int w)       //创建图
{
    edgenode *p;
    p=setup;
    p->cost=w;
    p->veradj=t;
    p->link=NULL;
    if(Head[f].adjacent==NULL)
    {
        Head[f].adjacent=p;
        Tail[f]=p;
    }
    else
    {
        Tail[f]->link=p;
        Tail[f]=p;
    }
}

void shortestpath (int v,int n)    //从v开始
{
    edgenode *p;
    int u,k;
    int queue[1000];
    int front=1,rear=1;
    for(int i=1;i<=n;i++)
    {
        path[i]=-1;
        dist[i]=-1;
    }
    dist[v]=0;
```

```
    queue[rear++]=v;

    while(rear-front!=0)
    {
      u=queue[front++];
      p=Head[u].adjacent;
      while(p!=NULL)
      {
        k=p->veradj;
        if(dist[k]==-1)
        {
            queue[rear++]=k;
            dist[k]=dist[u]+1;
            path[k]=u;
        }
        p=p->link;
      }
    }
}
int main()
{
    int tmp[100],t=0;
    int from,to,w;
    int m,n;  //m个点 n个边
    scanf("%d%d",&m,&n);
    for(int i=1;i<=n;i++)
    {
        scanf("%d%d",&from,&to);
        createedge(from,to,1);
     }
     shortestpath(1,m);                    //1是开始点，可以改变

     //验证
     int s;   //s为要查找的顶点
     scanf("%d",&s);
     printf("从1开始到%d的最短路径长度为:%d\n",s,dist[s]);
     for(int i=s;dist[i]!=0;i=path[i])
     tmp[++t]=i;
     printf("%d ",1);   //起始点
     for(int i=t;i>=1;i--)
     printf("%d ",tmp[i]);
     return 0;
}
```

运行界面



有权最短路径问题

迪杰斯特拉算法

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

typedef struct edgenode
{
    int veradj;
    int cost;
    struct edgenode *link;
}edgenode;

typedef struct vertexnode
{
   char vername;
   edgenode *adjacent;
}vertexnode;

# define setup (edgenode *)malloc(sizeof(edgenode))
edgenode *Tail[100];
vertexnode Head[100];

int s[100];
int dist[100];
int path[100];

void Dshortestpath (int n,int v)
{
    //初始化
```

```
    int u,k,Idist;
    edgenode *p;
    for(int i=1;i<=n;i++)
    {
        path[i]=-1;
        dist[i]=0x7fffffff;
        s[i]=0;
    }
    dist[v]=0;
    s[v]=1;
    p=Head[v].adjacent;
    u=v;                        //u为即将访问的结点

    //求初始顶点到其他各顶点的最短路径
    for(int j=1;j<=n-1;j++)
    {
      while(p!=NULL)
      {
         k=p->veradj;
         if(s[k]!=1&&dist[u]+p->cost<dist[k])
         {
           dist[k]=dist[u]+p->cost;
           path[k]=u;
         }
         p=p->link;
      }

      Idist=0x7fffffff;
      for(int i=1;i<=n;i++)
      if(dist[i]<Idist&&s[i]==0)
      {
         Idist=dist[i];
         u=i;
      }
      s[u]=1;
      p=Head[u].adjacent;
    }
 }
void createedge (int f,int t,int w)           //构建图
{
    edgenode *p;
    p=setup;
    p->cost=w;
    p->link=NULL;
    p->veradj=t;
    if(Head[f].adjacent==NULL)
    {
        Head[f].adjacent=p;
        Tail[f]=p;
    }else
    {
        Tail[f]->link=p;
        Tail[f]=p;
    }

    p=setup;
    p->veradj=f;
    p->cost=w;
```

```
    p->link=NULL;
    if(Head[t].adjacent==NULL)
    {
      Head[t].adjacent=p;
      Tail[t]=p;
    }
    else
    {
        Tail[t]->link=p;
        Tail[t]=p;
    }

}

int main()
{
    int from,to,w;
    int n,m;  //n个顶点，m条边
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m;i++)
    {
      scanf("%d%d%d",&from,&to,&w);
      createedge(from,to,w);
     }

     Dshortestpath (n,1);             //从一开始

     for(int i=1;i<=n;i++)
     printf("%d ",dist[i]);

     return 0;
}
```

弗洛伊德算法

```
# include <stdio.h>

#define max 0x7fffffff
//从0开始的floyed算法

int edge[100][100];
int A[100][100];
int path[100][100];

int main()
{
    int from,to,w;
    int n,m;
    scanf("%d%d",&n,&m);    //n个顶点 ,m条边

    for(int i=0;i<n;i++)
    for(int j=0;j<n;j++)
    {
```

```
      if(i==j) edge[i][j]=0;
      else edge[i][j]=max;
    }

    for (int i=1;i<=m;i++)
    {
        scanf("%d%d%d",&from,&to,&w);
        edge[from][to]=w;
        edge[to][from]=w;
    }

    //初始化
    for(int i=0;i<n;i++)
    for(int j=0;j<n;j++)
    {
      A[i][j]=edge[i][j];
      if(i!=j&&A[i][j]<max)
      path[i][j]=i;
      else
      path[i][j]=-1;
    }

    //求图中两顶点的最短路径
    for(int k=0;k<n;k++)
    for(int i=0;i<n;i++)
    if(i!=k)
    for(int j=0;j<n;j++)
    if(j!=k&&j!=i&&A[i][k]<max&&A[k][j]<max&&A[i][k]+A[k][j]<A[i][j])
    {
        A[i][j]=A[i][k]+A[k][j];
        path[i][j]=path[k][j];
    }

    for(int i=0;i<n;i++)
    for(int j=0;j<n;j++)
    {
        printf("%d ",A[i][j]);
        if(j==n-1) printf("\n");
    }
    return 0;
}
```

普里姆算法

```
# include <stdio.h>

const int MAXN = 10000;
int edge[MAXN][MAXN];
struct A
{
```

```
    int lowcost;
    int vex;
};
A closedge[MAXN];

struct B
{
    int head;
    int tail;
    int cost;
};

B TE[MAXN];

int main()
{
    int n;
    int count;
    int min, max;
    int v;
    for (int i = 1; i <= n; i++)
    {
        closedge[i].lowcost = edge[1][i];
        closedge[i].vex = 1;
    }
    closedge[1].vex = -1;
    count = 1;
    for (int i = 2; i <= n; i++)
    {
        min = max;
        v = 0;
        for (int j = 1; j <= n; j++)
            if (closedge[j].vex != -1 && closedge[j].lowcost < min)
            {
                v = j;
                min = closedge[j].lowcost;
             }
    }
    if (v != 0)
    {
        TE[count].head = closedge[v].vex;
        TE[count].tail = v;
        TE[count].cost = closedge[v].lowcost;
    }
    count++;
    closedge[v].lowcost = 0;
    closedge[v].vex = -1;
    for(int j=1;j<=n;j++)
        if (closedge[j].vex != -1 && edge[v][j] < closedge[j].lowcost)
        {
            closedge[j].lowcost = edge[v][j];
            closedge[j].vex = v;
        }
    return 0;
}
```

克鲁斯卡尔算法

```
# include <stdio.h>
# include <algorithm>
const int MAXN = 10000;


struct B
{
    int head;
    int tail;
    int cost;
};

B TE[MAXN],E[MAXN];
int make_set[MAXN];
int rank[MAXN];      //默认为0

void init(int n)
{
    for (int i = 1; i <= n; i++)
        make_set[i] = i;

}

int  find(int z)
{
    int r = z;
    while (r != make_set[r])
    {
        r = make_set[r];
    }
     int k = z;
     int j;
    while (k != r)
    {
        j = make_set[k];
        make_set[k] = r;
        k = j;
    }
}
void Union (int x, int y)
{
    int f1 = find(x);
    int f2 = find(y);
    if (rank[f1] <= rank[f2])
    {
        make_set[f1] = f2;
        if (rank[f1] == rank[f2])
        {
            rank[f2] = rank[f2] + 1;

        }
    }
    else
        make_set[f2] = f1;
```

```
}
int main()
{
    int n;
    int vex1, vex2, cost;
    init(n);
    int T = n;
    //sort(E);     一种排序方法
    int j = 1;
    int count = 0;
    while (T > 1)
    {
        vex1 = E[j].head;
        vex2 = E[j].tail;
        cost = E[j].cost;
        if (find(vex1) != find(vex2))
        {
            TE[count].head = vex1;
            TE[count].tail = vex2;
            TE[count].cost = cost;
        }
        count++;
        Union(vex1, vex2);
        T--;
    }
    j++;
    return 0;
}
```

排序（课本版）

堆排序

```
# include <stdio.h>

int R[50002];


void restore (int f,int e)
{
    int j=f,m;
    int tmp;
    while(j<=e/2)
    {
        if(2*j<e&&R[2*j]<R[2*j+1])
        m=2*j+1;
        else m=2*j;
        if(R[m]>R[j])
        {
```

```
            tmp=R[m];
            R[m]=R[j];
            R[j]=tmp;
            j=m;
        } else
        j=e;
    }

}

void heapsort (int n)
{
    int tmp;
    for(int i=n/2;i>=1;i--)
    restore (i,n);

    for(int i=n;i>=2;i--)
    {
        tmp=R[1];
        R[1]=R[i];
        R[i]=tmp;
        restore(1,i-1);
    }
}

int main()
{
    int n;
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
    scanf("%d",&R[i]);

    heapsort(n);

    for(int i=1;i<=n;i++)
    printf("%d ",R[i]);

    return 0;
}
```

插入排序

```
# include <stdio.h>

int R[50002];

void insertsort(int n)
{
    int i, j;
    int tmp;
```

```
    for ( j = 2;j <= n; j++)
    {
        i = j - 1;
        tmp = R[j];
        while (i >= 0&&tmp<R[i])
        {
            R[i + 1] = R[i];
            i--;
        }
        R[i + 1] = tmp;
    }
}
int main()
{
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
        scanf("%d", &R[i]);
    insertsort(n);

    for (int i = 1; i <= n; i++)
        printf("%d ",R[i]);
    return 0;
}
```

快速排序

```
# include <stdio.h>

int R[50001];

int partition(int m, int n)
{
    int i = m;
    int j = n + 1;
    int r= R[m];
    int tmp;
    while (i < j)
    {
        i++;
        while (R[i] <= r) i++;
        j--;
        while (R[j] > r) j--;
        if (i < j)
        {
            tmp = R[i];
            R[i] = R[j];
            R[j] = tmp;
        }
    }
```

```
    tmp = R[m];
    R[m] = R[j];
    R[j] = tmp;
    return  j;
}

void qsort(int m, int n)
{
    int j;
    if (m < n)
    {
        j = partition(m, n);
        qsort(m, j - 1);
        qsort(j + 1, n);
    }
}

int main()
{
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
        scanf("%d", &R[i]);
    qsort(1, n);
    for (int i = 1; i <= n; i++)
        printf("%d ", R[i]);
    return 0;
}
```

希尔排序

```
# include <stdio.h>

void shellsort(int *R,int n)
{
    int i, j,tmp;
    int d = n / 2;
    while (d > 0)
    {
        for (j = d; j <= n; j++)
        {
            i = j - d;
            tmp = R[j];
            while (i > 0 && R[i] > tmp)
            {
                R[i + d] = R[i];
                i = i - d;
            }
            R[i + d] = tmp;
        }
        d /= 2;
    }
}
```

```
int main()
{
    int n;
    int R[100];
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
        scanf("%d", &R[i]);

    shellsort(R,n);

    for (int i = 1; i <= n; i++)
        printf("%d ", R[i]);

    return 0;
}
```

冒泡排序

```
# include <stdio.h>

void Bubble(int *R, int n)
{
    int  t, tmp;
    int BOUND = n;
    while (BOUND != 0)
    {
        t = 0;
        for(int j=1;j<=BOUND-1;j++)
            if (R[j] > R[j + 1])
            {
                tmp = R[j];
                R[j] = R[j + 1];
                R[j + 1] = tmp;
                t = j;
            }
        BOUND = t;
    }
}

int main()
{
    int n;
    int R[100];
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
        scanf("%d", &R[i]);

    Bubble(R, n);

    for (int i = 1; i <= n; i++)
```

```
        printf("%d ", R[i]);

    return 0;
}
```

选择排序

```
# include <stdio.h>

void ssort(int *R, int n)
{
    int tmp,t, i, j;
    for (j = n; j >= 2; j--)
    {
        t = 1;
        for ( i = 2; i <= j; i++)
            if (R[t] < R[i])
                t = i;
        tmp = R[j];
        R[j] = R[t];
        R[t] = tmp;
    }
}

int main()
{
    int n;
    int R[100];
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
        scanf("%d", &R[i]);

    ssort(R, n);

    for (int i = 1; i <= n; i++)
        printf("%d ", R[i]);

    return 0;
}
```

合并排序

```
# include <iostream>

using namespace std;

const int MAXN=10000;
//a为待排序的数组，b为辅助数组
```

```
int a[MAXN],b[MAXN],n;

void msort (int ,int );

int main()
{
    cin>>n;                 //输入n个数
    for(int i=1;i<=n;i++)
    cin>>a[i];

    msort(1,n);

    for(int i=1;i<=n;i++)
    cout<<a[i]<<" ";

    return 0;
}

void msort(int L,int R )
{
   if(L==R)  return ;
   //分解序列
   int mid=(L+R)/2;
   msort(L,mid);
   msort(mid+1,R);

   //合并子序列到临时数组b
   int i=L,j=mid+1,k=L;
   while(i<=mid&&j<=R)
   if(a[i]<=a[j])
   b[k++]=a[i++];
   else
   b[k++]=a[j++];

   //复制剩余部分到临时数组b
   while(i<=mid) b[k++]=a[i++];
   while(j<=R)   b[k++]=a[j++];

   //将合并序列倒回原数组a
   for(i=L;i<=R;i++) a[i]=b[i];

}
```

查找