好

▼ 二叉Tree

▼ 递归遍历

```
#include<iostream>
#define ll long long
using namespace std;
typedef struct bitnode
{
    char data;
    struct bitnode *l,*r;
}bitnode,*bitree;
void creattree(bitree &bt)
{
    char c;
    c=getchar();
    if(c=='#')
    {
        bt=NULL;
    }
    else
    {
        bt=(bitnode *)malloc(sizeof(bitnode));
        if(!bt)
        {
            return;
        }
        bt->data=c;
        creattree(bt->l);
        creattree(bt->r);
    }
}
void pre(bitree bt)
```

```
{
    if(bt)
    {
        cout<<bt->data;
        pre(bt->l);
        pre(bt->r);
    }
}
void mid(bitree bt)
{
    if(bt)
    {
        mid(bt->l);
        cout<<bt->data;
        mid(bt->r);
    }
}
void post(bitree bt)
{
    if(bt)
    {
        post(bt->l);
        post(bt->r);
        cout<<bt->data;
    }
}
int main()
{
    bitree t1=(bitree)malloc(sizeof(bitnode));
    creattree(t1);
    pre(t1);
    cout<<'\n';
    mid(t1);
    cout<<'\n';
    post(t1);
}
```

▼ 非递归遍历

```
#include<iostream>
#define ll long long
using namespace std;
typedef struct bitnode
{
    char data;
    struct bitnode *l,*r;
}bitnode,*bitree;
typedef struct
{
    bitree data[10086];
    int final;
    int num;
}queue;
void initqueue(queue *q)
{
    q->final=-1;
    q->num=0;
}
int queueempty(queue q)
{
    if(q.final+1==q.num)
    {
        return 1;
    }
    return 0;
}
void push(queue *q,bitree e)
{
    if(q->final==10086-1)
    {
        return;
    }
    q->data[++(q->final)]=e;
}
void pop(queue *q,bitree *e)
{
    if(q->final+1==q->num)
```

```
    {
        return;
    }
    *e=q->data[(q->num)++];
}
typedef struct
{
    bitree data[10086];
    int top;
}stack;
void initstack(stack *s)
{
    s->top=-1;
}
int stackempty(stack s)
{
    if(s.top==-1)
    {
        return 1;
    }
    return 0;
}
void push(stack *s,bitree e)
{
    if(s->top==10086-1)
    {
        return;
    }
    s->data[++(s->top)]=e;
}
void pop(stack *s,bitree *e)
{
    if(s->top==-1)
    {
        return;
    }
    *e=s->data[(s->top)--];
}
```

```
void gettop(stack *s,bitree *e)
{
    if(s->top==-1)
    {
        return;
    }
    *e=s->data[s->top];
}
void creattree(bitree &bt)
{
    char c;
    c=getchar();
    if(c=='#')
    {
        bt=NULL;
    }
    else
    {
        bt=(bitnode *)malloc(sizeof(bitnode));
        if(!bt)
        {
            return;
        }
        bt->data=c;
        creattree(bt->l);
        creattree(bt->r);
    }
}
void pre(bitree bt)//前序遍历，需要stack
{
    stack s;
    initstack(&s);
    bitree p=bt;
    while(p||!stackempty(s))
    {
        if(p)
        {
            cout<<p->data;
```

```
            push(&s,p);
            p=p->l;
        }
        else
        {
            pop(&s,&p);
            p=p->r;
        }
    }
}
void mid(bitree bt)//中序遍历，需要stack
{
    stack s;
    initstack(&s);
    bitree p=bt;
    while(p||!stackempty(s))
    {
        if(p)
        {
            push(&s,p);
            p=p->l;
        }
        else
        {
            pop(&s,&p);
            cout<<p->data;
            p=p->r;
        }
    }
}
void level(bitree bt)//层序遍历，需要queue
{
    if(bt)
    {
        queue q;
        initqueue(&q);
        push(&q,bt);
        bitree p=bt;
```

```
        while(!queueempty(q))
        {
            pop(&q,&p);
            if(p)
            {
                cout<<p->data;
                if(p->l)
                {
                    push(&q,p->l);
                }
                if(p->r)
                {
                    push(&q,p->r);
                }
            }
        }
    }
}
void post(bitree bt)//后序遍历，需要stack
{
    stack s;
    initstack(&s);
    bitree p=bt,temp=NULL;
    while(p||!stackempty(s))
    {
        if(p)
        {
            push(&s,p);
            p=p->l;
        }
        else
        {
            gettop(&s,&p);
            if(!p->r||p->r==temp)
            {
                pop(&s,&p);
                cout<<p->data;
                temp=p;
```

```
                p=NULL;
            }
            else
            {
                p=p->r;
            }
        }
    }
}
int main()
{
    bitree t1=(bitree)malloc(sizeof(bitnode));
    creattree(t1);
    pre(t1);
    cout<<'\n';
    mid(t1);
    cout<<'\n';
    level(t1);
        cout<<'\n';
}
```

▼ 前序、中序推二叉树

```
#include <iostream>
typedef long long ll;
using namespace std;
typedef struct bitnode
{
    char data;
    struct bitnode *l,*r;
}bitnode,*bitree;
char a[80],b[80];
void Createtree(bitree bt,char prea[],char ina[],int n)
{
    if(n<1)
    {
        cout<<'#';
        bt=NULL;
```

```
        return;
    }
    bt=(bitree)malloc(sizeof(bitnode));
    bt->data=prea[0];
    cout<<prea[0];
    int i=0;
    while(i<n&&ina[i]!=prea[0])
    {
        i++;
    }
    int in=i;
    int rn=n-i-1;
    char preal[80],inal[80],prear[80],inar[80];
    for(int k=0;k<i;k++)
    {
        preal[k]=prea[k+1];
        inal[k]=ina[k];
    }
    for(int k=i+1;k<n;k++)
    {
        prear[k-i-1]=prea[k];
        inar[k-i-1]=ina[k];
    }
    Createtree(bt->l,preal,inal,in);
    Createtree(bt->r,prear,inar,rn);
}
signed main()
{
    bitree t1=(bitree)malloc(sizeof(bitnode));
    cin>>a;
    cin>>b;
    Createtree(t1,a,b,8);
}
```

▼ 判断完全二叉树

```
#include <iostream>
typedef long long ll;
```

```
using namespace std;
typedef struct bitnode
{
    char data;
    struct bitnode *l,*r;
}bitnode,*bitree;
typedef struct
{
    bitree data[80];
    int final;
    int num;
}queue;
void initqueue(queue *q)
{
    q->final=-1;
    q->num=0;
}
bool queueempty(queue q)
{
    if(q.final+1==q.num)
    {
        return 1;
    }
    return 0;
}
void push(queue *q,bitree e)
{
    if(q->final==80-1)
    {
        return;
    }
    q->data[++(q->final)]=e;
}
void pop(queue *q,bitree *e)
{
    if(q->final+1==q->num)
    {
        return;
```

```
    }
    *e=q->data[(q->num)++];
}
void createtree(bitree &bt)
{
    char c;
    c=getchar();
    if(c=='#')
    {
        bt=NULL;
    }
    else
    {
        bt=(bitree)malloc(sizeof(bitnode));
        if(!bt)
        {
            return;
        }
        bt->data=c;
        createtree(bt->l);
        createtree(bt->r);
    }
}
bool complete(bitree bt)
{
    if(bt)
    {
        queue q;
        initqueue(&q);
        push(&q,bt);
        bool f=0;
        bitree p=bt;
        while(!queueempty(q))
        {
            pop(&q,&p);
            if(!p)
            {
                f=1;
```

```
            }
            else
            {
                if(f)
                {
                    return 0;
                }
                push(&q,p->l);
                push(&q,p->r);
            }
        }
    }
    return 1;
}
signed main()
{
    bitree t1=(bitree)malloc(sizeof(bitnode));
    createtree(t1);
    cout<<complete(t1);
}
```

▼ 哈夫曼树

```
#include<iostream>
#include <cstring>

using namespace std;
typedef struct taghtnode
{
    int wight;//权值
    int p,l,r;//数组下标
}htnode,*htree;
void creatht(htree &HT,int *w,int n)
{
    int m=2*n-1;//总结点数
    HT=(htree)malloc((m+1)*sizeof(htnode));//0号不用
    int i;
    for(i=1;i<=m;i++)
```

```
    {
        if(i<=n)
        {
            HT[i].wight=w[i];
        }
        HT[i].p=0;
        //HT[i].p=HT[i].l=HT[i].r=0;
    }
    for(i=n+1;i<=m;i++)
    {
        int min1=9999999;
        int min2=9999999;//查询最小和次小用的
        int id1=0,id2=0;//保存最小和次小的下标
        for(int j=1;j<i;j++)
        {
            if(HT[j].p==0)
            {
                int temp=HT[j].wight;
                if(temp<min1)
                {
                    min2=min1;
                    id2=id1;
                    min1=temp;
                    id1=j;
                }
                else if(temp<min2)
                {
                    min2=temp;
                    id2=j;
                }
            }
        }
        HT[i].wight=HT[id1].wight+HT[id2].wight;//i号的权
        HT[i].l=id1,HT[i].r=id2;//i号的左右子树
        HT[id1].p=HT[id2].p=i;//id1和id2的父节点
    }
}
typedef char **hfmcode;
```

```
void hfmcoding(hfmcode &code,htree &ht,int n)
{
    code=(hfmcode)malloc((n+1)*sizeof(char *));
    char *cd=(char *)malloc(n*sizeof(char));
    cd[0]='\0';
    int i;
    int strst=n-2;
    for(i=1;i<=n;i++)
    {
        int c=i;
        int p=ht[i].p;
        while(p)
        {
            if(ht[p].l==c)
            {
                cd[strst]='0';
            }
            else
            {
                cd[strst]='1';
            }
            strst--;
            c=p;
            p=ht[p].p;
            code[i]=(char *)malloc((n-1-strst)*sizeof(c
            strcpy(code[i],&cd[strst-1]);
        }
    }
}
int main()
{

}
```

▼ 图

▼ 邻接矩阵

```
#include<iostream>
using namespace std;
typedef struct
{
    int vex[10000];
    int arcs[1000][1000];
    int vexnum,arcnum;
}Mgraph;
int loacateve(Mgraph g,int v)
{
    for(int i=0;i<=g.vexnum;i++)
    {
        if(g.vex[i]==v)
        {
            return i;
        }
    }
    return -1;
}
void creategraph(Mgraph &g)
{
    int u,v;
    cin>>g.vexnum>>g.arcnum;
    for(int i=0;i<g.vexnum;i++)
    {
        cin>>g.vex[i];
    }
    for(int i=0;i<g.vexnum;i++)
    {
        for(int j=0;j<g.vexnum;i++)
        {
            g.arcs[i][j]=0;
        }
    }
    for(int k=0;k<g.arcnum;k++)
    {
        cin>>u>>v;
        int i=loacateve(g,u);
```

```
        int j=loacateve(g,v);
        g.arcs[i][j]=g.arcs[j][i]=1;
    }
}
int firstadjves(Mgraph g,int v)//查找v的第一个邻接点
{
    for(int i=0;i<g.vexnum;i++)
    {
        if(g.arcs[i][v]==1)
        {
            return i;
        }
    }
    return -1;
}
int nextadjvex(Mgraph g,int v,int w)//插找v的下一个邻接点
{
    for(int i=w+1;i<g.vexnum;i++)
    {
        if(g.arcs[i][v]==1)
        {
            return i;
        }
    }
    return -1;
}
int main()
{

}
```

▼ 邻接表

```
typedef struct arcnode
{
    int adjvex;
    int weight;
    struct arcnode *nextvex;
```

```
}arcnode;
typedef struct vnode
{
    int data;
    arcnode *firstarc;
}vnode;
typedef struct
{
    vnode verices[100];
    int vexnum;
    int arcnum;
}algraph;
void create(algraph &g)//建图
{
    FILE *fp=fopen("C:/Users/Administrator/Desktop/11.t
    fscanf(fp,"%d%d",&g.vexnum,&g.arcnum);
    for(int i=0;i<g.vexnum;i++)
    {
        fscanf(fp,"%d",&g.verices[i].data);
        g.verices[i].firstarc=NULL;
    }
    int u,v;
    for(int i=0;i<g.arcnum;i++)
    {
        fscanf(fp,"%d%d",&u,&v);
        arcnode *s1=(arcnode *)malloc(sizeof(arcnode));
        arcnode *s2=(arcnode *)malloc(sizeof(arcnode));
        s1->adjvex=u;
        s1->nextvex=g.verices[v].firstarc;
        g.verices[v].firstarc=s1;
        s2->adjvex=v;
        s2->nextvex=g.verices[u].firstarc;
        g.verices[u].firstarc=s2;
    }
    for(int i=0;i<g.vexnum;i++)//输出边
    {
        arcnode *p=g.verices[i].firstarc;
        while(p)
```

```
        {
            cout<<"("<<g.verices[i].data<<','<<g.verice
            p=p->nextvex;
        }
        cout<<'\n';
    }
}
```

▼ DFS

▼ 邻接矩阵

```
#include "iostream"
using namespace std;
typedef struct
{
    int vexnum,arcnum;
    int arcs[100][100];
    int ves[100];
}mgraph;
void create(mgraph &g)
{
    int u,v;
    FILE *fp=fopen("C:/Users/Administrator/Desktop/1
    fscanf(fp,"%d%d",&g.vexnum,&g.arcnum);
    for(int i=0;i<g.vexnum;i++)
    {
        fscanf(fp,"%d",&g.ves[i]);
    }
    for(int i=0;i<g.arcnum;i++)
    {
        fscanf(fp,"%d%d",&u,&v);
        g.arcs[u][v]=g.arcs[v][u]=1;
    }
    fclose(fp);
}
int vis[100];
int firstsdjvex(mgraph g,int v)
{
```

```
    for(int i=0;i<g.vexnum;i++)
    {
        if(g.arcs[i][v])
        {
            return i;
        }
    }
    return -1;
}
int nextadjvex(mgraph g,int v,int w)
{
    for(int i=w+1;i<g.vexnum;i++)
    {
        if(g.arcs[i][v])
        {
            return i;
        }
    }
    return -1;
}
void dfs(mgraph g,int v)
{
    cout<<v<<' ';
    vis[v]=1;
    for(int w=firstsdjvex(g,v);w>=0;w=nextadjvex(g,v
    {
        if(!vis[w])
        {
            dfs(g,w);
        }
    }
}
void dfstraverse(mgraph g)
{
    for(int i=0;i<g.vexnum;i++)
    {
        vis[i]=0;
    }
```

```
    for(int i=0;i<g.vexnum;i++)
    {
        if(!vis[i])
        {
            dfs(g,i);
            cout<<'\n';
        }
    }
}
```

▼ 邻接表

```
#include "iostream"
using namespace std;
typedef struct arcnode
{
    int adjvex;
    int weight;
    struct arcnode *nextvex;
}arcnode;
typedef struct vnode
{
    int data;
    arcnode *firstarc;
}vnode;
typedef struct
{
    vnode verices[100];
    int vexnum;
    int arcnum;
}algraph;
typedef struct
{
    arcnode *data[100];
    int top;
}stack;
void init(stack *s)
{
```

```
    s->top=-1;
}
int empty(stack *s)
{
    if(s->top==-1)
    {
        return 1;
    }
    return 0;
}
void push(stack *s,arcnode *p)
{
    if(s->top==99)
    {
        return;
    }
    s->data[++(s->top)]=p;
}
void pop(stack *s,arcnode **p)
{
    if(s->top==-1)
    {
        return;
    }
    *p=s->data[(s->top)--];
}
void create(algraph &g)//建图
{
    FILE *fp=fopen("C:/Users/Administrator/Desktop/1
    fscanf(fp,"%d%d",&g.vexnum,&g.arcnum);
    for(int i=0;i<g.vexnum;i++)
    {
        fscanf(fp,"%d",&g.verices[i].data);
        g.verices[i].firstarc=NULL;
    }
    int u,v;
    for(int i=0;i<g.arcnum;i++)
    {
```

```
        fscanf(fp,"%d%d",&u,&v);
        arcnode *s1=(arcnode *)malloc(sizeof(arcnode
        arcnode *s2=(arcnode *)malloc(sizeof(arcnode
        s1->adjvex=u;
        s1->nextvex=g.verices[v].firstarc;
        g.verices[v].firstarc=s1;
        s2->adjvex=v;
        s2->nextvex=g.verices[u].firstarc;
        g.verices[u].firstarc=s2;
    }
}
int vis[100];
void dfs(algraph G,int v){
    vis[v]=true;
    ArcNode *p=G.verices[v].firstarc;
    int u;
    while(p){
        u=p->adjvex;
        if(!visited[u]) DFS(G,u);
        p=p->nextvex;
    }
}
void dfst(algraph g)
{
    for(int i=0;i<g.vexnum;i++)
    {
        vis[i]=0;
    }
    for(int i=0;i<g.vexnum;i++)
    {
        if(!vis[i])
        {
            dfs(g,i);
        }
    }
}
int main()
{
```

```
    algraph g;
    create(g);
    dfst(g);
}
```

▼ BFS

▼ 邻接矩阵

```
#include "iostream"
using namespace std;
typedef struct
{
    int vexnum,arcnum;
    int arcs[100][100];
    int ves[100];
}mgraph;
typedef struct
{
    int data[100];
    int final,num;
}queue;
void init(queue *q)
{
    q->final=-1;
    q->num=-1;
}
void push(queue *q,int v)
{
    if(q->final-q->num==100)
    {
        return;
    }
    q->data[++(q->final)]=v;
}
void pop(queue *q,int *v)
{
    if(q->final==q->num)
    {
```

```
        return;
    }
    *v=q->data[++(q->num)];
}
int empty(queue *q)
{
    if(q->num==q->final)
    {
        return 1;
    }
    return 0;
}
void create(mgraph &g)
{
    int u,v;
    FILE *fp=fopen("C:/Users/Administrator/Desktop/1
    fscanf(fp,"%d%d",&g.vexnum,&g.arcnum);
    for(int i=0;i<g.vexnum;i++)
    {
        fscanf(fp,"%d",&g.ves[i]);
    }
    for(int i=0;i<g.arcnum;i++)
    {
        fscanf(fp,"%d%d",&u,&v);
        g.arcs[u][v]=g.arcs[v][u]=1;
    }
    fclose(fp);
}
int vis[100];
int firstsdjvex(mgraph g,int v)
{
    for(int i=0;i<g.vexnum;i++)
    {
        if(g.arcs[i][v])
        {
            return i;
        }
    }
```

```
    return -1;
}
int nextadjvex(mgraph g,int v,int w)
{
    for(int i=w+1;i<g.vexnum;i++)
    {
        if(g.arcs[i][v])
        {
            return i;
        }
    }
    return -1;
}
void bfs(mgraph g,int v)
{
    cout<<v<<' ';
    vis[v]=1;
    queue q;
    init(&q);
    push(&q,v);
    while(!empty(&q))
    {
        pop(&q,&v);
        for(int w=firstsdjvex(g,v);w>=0;w=nextadjvex
        {
            if(!vis[w])
            {
                cout<<w<<' ';
                vis[w]=1;
                push(&q,w);
            }
        }
    }
}
void bfstraverse(mgraph g)
{
    for(int i=0;i<g.vexnum;i++)
    {
```

```
        vis[i]=0;
    }
    for(int i=0;i<g.vexnum;i++)
    {
        if(!vis[i])
        {
            bfs(g,i);
            cout<<'\n';
        }
    }
}
```

▼ 邻接表

```
#include "iostream"
using namespace std;
typedef struct arcnode
{
    int adjvex;
    int weight;
    struct arcnode *nextarc;
}arcnode;
typedef struct vnode
{
    int data;
    arcnode *firstarc;
}vnode;
typedef struct
{
    vnode verices[100];
    int vexnum,arcnum;
}algraph;
typedef struct
{
    int data[100];
    int num;
    int final;
}queue;
```

```
void init(queue *q)
{
    q->num=-1;
    q->final=-1;
}
void push(queue *q,int v)
{
    if(q->final-q->num==100)
    {
        return;
    }
    q->data[++(q->final)]=v;
}
void pop(queue *q,int *v)
{
    if(q->final==q->num)
    {
        return;
    }
    *v=q->data[++(q->num)];
}
int empty(queue *q)
{
    if(q->num==q->final)
    {
        return 1;
    }
    return 0;
}
int locate(algraph g,int v)
{
    for(int i=0;i<g.vexnum;i++)
    {
        if(g.verices[i].data==v)
        {
            return i;
        }
    }
```

```
    return -1;
}
void CreateGraph(algraph &G)//建图
{

    int i, j, k;
    arcnode *p, *p1, *p2;

    //输入总顶点数和边数

    scanf("%d%d", &G.vexnum, &G.arcnum);

    //输入顶点值，初始化表头结点的指针域

    for (i = 0; i < G.vexnum; i++)
    {
        getchar();
        scanf("%d", &G.verices[i].data);
        G.verices[i].firstarc = NULL;
    }

    //输入各边 (Vi,Vj)中的下标i,j
    for (k = 0; k < G.arcnum; k++)
    {
        scanf("%d%d", &i, &j);  //一个边依附的两个顶点

        p1 = (arcnode *)malloc(sizeof(arcnode));   /
        p1->adjvex = j;                            /
        p1->nextarc = G.verices[i].firstarc;
        G.verices[i].firstarc = p1;   //将新结点*p1插入

        //无向图需要生成对称的边结点*p2，有向图不需要
        p2 = (arcnode *)malloc(sizeof(arcnode));   /
        p2->adjvex = i;                            /
        p2->nextarc = G.verices[j].firstarc;
        G.verices[j].firstarc = p2;  //将新结点*p2插入
    }
}
```

```
int vis[100];
void bfs(algraph g,int v)
{
    cout<<g.verices[v].data<<' ';
    vis[v]=1;
    queue q;
    init(&q);
    push(&q,v);
    while(!empty(&q))
    {
        pop(&q,&v);
        for(arcnode *p=g.verices[v].firstarc;p;p=p->
        {
            int u=p->adjvex;
            if(!vis[u])
            {
                push(&q,u);
                vis[u]=1;
                cout<<u<<' ';
            }
        }
    }
}
void bfstraverse(algraph g)
{
    for(int i=0;i<g.vexnum;i++)
    {
        vis[i]=0;
    }
    for(int i=0;i<g.vexnum;i++)
    {
        if(!vis[i])
        {
            bfs(g,i);
            cout<<'\n';
        }
    }
}
```

```
int main()
{
    algraph g;
    CreateGraph(g);
    bfstraverse(g);
}
```

▼ 最小生成树

▼ P

```
#include<iostream>
using namespace std;
struct tagclosest
{
    int u;
    int mincost;
}closestedge[10000];//下标是位置
typedef struct
{
    int vex[10000];
    int arcs[100][100];
    int vexnum,arcnum;
}Mgraph;
int Locatevex(Mgraph g,int v)
{
    for(int i=0;i<=g.vexnum;i++)
    {
        if(g.vex[i]==v)
        {
            return i;
        }
    }
    return -1;
}
void creategraph(Mgraph &g)
{
    int u,v,w;
    cin>>g.vexnum>>g.arcnum;
```

```
    for(int i=0;i<g.vexnum;i++)
    {
        cin>>g.vex[i];
    }
    for(int i=0;i<g.vexnum;i++)
    {
        for(int j=0;j<g.vexnum;j++)
        {
            g.arcs[i][j]=9999;
        }
    }
    for(int k=0;k<g.arcnum;k++)
    {
        cin>>u>>v>>w;
        int i=Locatevex(g,u);
        int j=Locatevex(g,v);
        g.arcs[i][j]=g.arcs[j][i]=w;
    }
}
void P(Mgraph g,int u0)
{
    int pos=Locatevex(g,u0);
    for(int k=0;k<g.vexnum;k++)
    {
        if(k!=pos)
        {
            closestedge[k].mincost=g.arcs[k][pos];
            closestedge[k].u=u0;
        }
    }
    closestedge[pos].mincost=0;//u中的点
    for(int i=1;i<g.vexnum;i++)
    {
        int xiao=999999;
        for(int j=0;j<g.vexnum;j++)
        {
            if(closestedge[j].mincost!=0&&closestedg
            {
```

```
                xiao=closestedge[j].mincost;
                pos=j;
            }
        }
        closestedge[pos].mincost=0;//并入u中
        cout<<closestedge[pos].u<<' '<<g.vex[pos]<<'
        for(int j=0;j<g.vexnum;j++)
        {
            if(closestedge[j].mincost!=0&&g.arcs[pos
            {
                closestedge[j].mincost=g.arcs[pos][j
                closestedge[j].u=g.vex[pos];
            }
        }
    }
}
int main()
{
    Mgraph g;
    creategraph(g);
    P(g,2);
}
```

▼ K

```

```

▼ 最短路

▼ dij

```
#include<iostream>
#include <cstring>
using namespace std;
typedef struct
{
    int vexnum,arcnum;
    int vex[10000];
    int arcs[100][200];
}Mgraph;
```

```
void createg(Mgraph &g)
{
    int u,v,w;
    cin>>g.vexnum>>g.arcnum;
    for(int i=0;i<g.vexnum;i++)
    {
        cin>>g.vex[i];
    }
    for(int i=0;i<g.vexnum;i++)
    {
        for(int j=0;j<g.vexnum;j++)
        {
            g.arcs[i][j]=999999;
        }
    }
    for(int i=0;i<g.arcnum;i++)
    {
        cin>>u>>v>>w;
        g.arcs[u][v]=g.arcs[v][u]=w;
    }
}
void dij(Mgraph g,int v,int final[],int path[],int d
{
    for(int i=0;i<g.vexnum;i++)
    {
        final[i]=0;
        d[i]=g.arcs[i][v];
        if(d[i]<999999)
        {
            path[i] = v;
        }
        else
        {
            path[i]=-1;
        }
    }
    final[v]=1;
    for(int i=1;i<g.vexnum;i++)
```

```
    {
        int Min=999999;
        int k=i;
        for(int w=0;w<g.vexnum;w++)
        {
            if(!final[w]&&d[w]<Min)
            {
                Min=d[w];
                k=w;
            }
        }
        final[k]=1;
        for(int w=0;w<g.vexnum;w++)
        {
            if(!final[w]&&(Min+g.arcs[k][w]<d[w]))
            {
                d[w]=Min+g.arcs[k][w];
                path[w]=k;
            }
        }
    }
    for(int i=0;i<g.vexnum;i++)
    {
        if(i!=v)
        {
            int p=path[i];
            if(p==-1)
            {
                cout<<"asdfg";
            }
            else
            {
                int m=0;
                b[m++]=i;
                while(p!=v)
                {
                    b[m++]=p;
                    p=path[p];
```

```
                }
                b[m]=v;
                for(int j=m;j>0;j--)
                {
                    cout<<g.vex[b[j]]<<"->";
                }
                cout<<g.vex[b[0]]<<' ';
                cout<<d[i]<<'\n';
            }
        }
    }
}
int d[1000],path[1000],final[1000],b[1000];
int main()
{
    memset(d,999999,sizeof d);
    Mgraph g;
    createg(g);
    dij(g,4,final,path,d,b);
}
```

▼ F

```
#include<iostream>
#include <cstring>
using namespace std;
typedef struct
{
    int vexnum,arcnum;
    int vex[10000];
    int arcs[100][200];
}Mgraph;
void createg(Mgraph &g)
{
    int u,v,w;
    cin>>g.vexnum>>g.arcnum;
    for(int i=0;i<g.vexnum;i++)
    {
```

```
        cin>>g.vex[i];
    }
    for(int i=0;i<g.vexnum;i++)
    {
        for(int j=0;j<g.vexnum;j++)
        {
            g.arcs[i][j]=999999;
        }
    }
    for(int i=0;i<g.arcnum;i++)
    {
        cin>>u>>v>>w;
        g.arcs[u][v]=g.arcs[v][u]=w;
    }
}
void F(Mgraph g,int **d,int **path)
{
    for(int i=0;i<g.vexnum;i++)
    {
        for(int j=0;j<g.vexnum;j++)
        {
            d[i][j]=g.arcs[i][j];
            if(i!=j&&g.arcs[i][j]<999999)
            {
                path[i][j]=i;
            }
            else
            {
                path[i][j]=-1;
            }
        }
    }
    for(int k=0;k<g.vexnum;k++)
    {
        for(int i=0;i<g.vexnum;i++)
        {
            for(int j=0;j<g.vexnum;j++)
            {
```

```
                if(d[i][k]+d[k][j]<d[i][j])
                {
                    d[i][j]=d[i][k]+d[k][j];
                    path[i][j]=path[k][j];
                }
            }
        }
    }
}
int main()
{

}
```

▼ 二叉排序树

```
#include "iostream"
using namespace std;
typedef struct bitnode
{
    char data;
    struct bitnode *l,*r;
}bitnode,*bitree;
bitnode *search(bitnode *t,char k)
{
    while(t!=NULL&&k!=t->data)
    {
        if(k<t->data)
        {
            t=t->l;
        }
        else
        {
            t=t->r;
        }
    }
    return t;
}
```

```
int insert(bitree &t,int k)
{
    if(t==NULL)
    {
        t=(bitree)malloc(sizeof(bitnode));
        t->data=k;
        t->l=t->r=NULL;
        return 1;
    }
    else if(t->data==k)
    {
        return 0;
    }
    else if(t->data>k)
    {
        return insert(t->l,k);
    }
    else if(t->data<k)
    {
        return insert(t->r,k);
    }
}
void create(bitree &t,int n,char str[])
{
    t=NULL;
    int i=0;
    while(i<n)
    {
        insert(t,str[i]);
        i++;
    }
}
void del(bitree &t,char k)
{
    bitree p=t,f=NULL;
    while(p&&p->data!=k)
    {
        f=p;
```

```
        if(p->data>k)
        {
            p=p->l;
        }
        else
        {
            p=p->r;
        }
    }
    if(!p)
    {
        return;
    }
    if(p->l&&p->r)
    {
        bitree q=p,s=p->l;
        while(s->r)
        {
            q=s;
            s=s->r;
        }
        p->data=s->data;
        if(q==p)
        {
            q->l=s->l;
        }
        else
        {
            q->r=s->r;
        }
        free(s);
    }
    else
    {
        bitree pc;
        if(p->l)
        {
            pc=p->l;
```

```
        }
        else
        {
            pc=p->r;
        }
        if(!f)
        {
            t=pc;
        }
        else if(f->l==p)
        {
            f->l=pc;
        }
        else
        {
            f->r=pc;
        }
        free(p);
    }
}
int main()
{

}
```