1. n(n-1)/2=28,n=8,8+1=9
2. (1)矩阵无向图1元素个数/2，表无向图边节点/2

矩阵有向图1元素个数，表有向图边节点数

(2)1表示有边相连

1. 本行中1的个数
2. 深度 广度

014523 012345

015423 012354

014532 013245

015432 013254

021453 021345

021543 021354

023145 023145

023154 023154

031452 031245

031542 031254

032145 032145

032154 032154

1. prim：{（0,1），（0,3），（1,2），（2,5），（5,4）}

Kruskal：{（0,1），（0,3），（1,2），（5,4），（2,5）}

8.0~1:1,0-1 0~2:4,0-1-2 0~3:2,0-3 0~4:8,0-4-1 0~5:10,0-3-5

1. 时间复杂度相同但floyd更快，因为dijkstra算法每次调用独立，而floyd每次会保存信息用于下次比较，因此提高效率

14.void inds1(matgraph *g*)

{

    int i,j,n;

    printf("各定点入度：\n");

    for(j=0;j<*g*.n;j++)

    {

        n=0;

        for(i=0;i<*g*.n;i++)

        if(*g*.edges[i][j]!=0)

        n++;

        printf("顶点%d:%d\n",j,n);

    }

}

void outds1(matgraph *g*)

{

    int i,j,n;

    printf("各顶点出度：\n");

    for(i=0;i<*g*.n;i++)

    {

        n=0;

        for(j=0;j<*g*.n;j++)

        if(*g*.edges[i][j]!=0)

        n++;

        printf("顶点%d:%d\n",i,n);

    }

}

void zerooutds1(matgraph *g*)

{

    int i,j,n;

    printf("出度为0的顶点:");

    for(i=0;i<*g*.n;i++)

    {

        n=0;

        for(j=0;j<*g*.n;j++)

        if(*g*.edges[i][j]!=0)

        n++;

        if(n==0)

        printf("%2d\n",i);

    }

    printf("\n");

}

15.void inds2(adjgraph \**g*)

{

    arcnode \*p;

    int a[maxv],i;

    for(i=0;i<*g*->n;i++)

    a[i]=0;

    for(i=0;i<*g*->n;i++)

    {

        p=*g*->adjlist[i].firstarc;

        while(p!=NULL)

        {

            a[p->adjvex]++;

            p=p->nextarc;

        }

    }

    printf("各顶点入度：\n");

    for(i=0;i<*g*->n;i++)

    printf("顶点%d:%d\n",i,a[i]);

}

void outds2(adjgraph \**g*)

{

    int i,n;

    arcnode \*p;

    printf("各点的出度:\n");

    for(i=0;i<*g*->n;i++)

    {

        n=0;

        p=*g*->adjlist[i].firstarc;

        while(p!=NULL)

        {

            n++;

            p=p->nextarc;

        }

        printf("顶点%d:%d\n",i,n);

    }

}

void zerooutds2(adjgraph \**g*)

{

    int i,n;

    arcnode \*p;

    printf("出度·为0的顶点：");

    for(i=0;i<*g*->n;i++)

    {

        p=*g*->adjlist[i].firstarc;

        n=0;

        while(p!=NULL)

        {

            n++;

            p=p->nextarc;

        }

        if(n==0)

        printf("%2d",i);

    }

    printf("\n");

}

20.int visited[MAXV];

void findpath(AdjGraph \**G*,int *u*,int *v*,int *path*[],int *d*,int *length*)

{

    int w,i;

    ArcNode \*p;

*path*[*d*]=*u*; *d*++;

    visited[*u*]=1;

    if (*u*==*v* && *d*>0)

    {

        printf(" 路径长度:%d, 路径:",*length*);

        for (i=0;i<*d*;i++)

        printf("%2d",*path*[i]);

        printf("\n");

    }

    p=*G*->adjlist[*u*].firstarc;

    while (p!=NULL)

    {

        w=p->adjvex;

        if (visited[w]==0)

        findpath(*G*,w,*v*,*path*,*d*,p->weight+*length*);

        p=p->nextarc;

    }

    visited[*u*]=0;

}

int main()

{

    AdjGraph \*G;

    int A[MAXV][MAXV]={

    {0,4,6,6,INF,INF,INF}, {INF,0,1,INF,7,INF,INF},

    {INF,INF,0,INF,6,4,INF},{INF,INF,2,0,INF,5,INF},

    {INF,INF,INF,INF,0,INF,6},{INF,INF,INF,INF,1,0,8},

    {INF,INF,INF,INF,INF,INF,0} };

    int n=7, e=12;

    CreateAdj(G,A,n,e);

    printf("图 G 的邻接表:\n");

    DispAdj(G);

    int u=0,v=5;

    int path[MAXV];

    printf("从%d->%d 的所有路径:\n",u,v);

    findpath(G,u,v,path,0,0);

    DestroyAdj(G);

    return 1;

}

图 G 的邻接表:

0: 1[4] → 2[6] → 3[6] → ∧

1: 2[1] → 4[7] → ∧

2: 4[6] → 5[4] → ∧

3: 2[2] → 5[5] → ∧

4: 6[6] → ∧

5: 4[1] → 6[8] → ∧

6: ∧

从 0->5 的所有路径:

路径长度:9, 路径: 0 1 2 5

路径长度:10, 路径:0 2 5

路径长度:12, 路径:0 3 2 5

路径长度:11, 路径:0 3 5