1. C
2. A
3. C
4. B
5. (1)存在

(2）ID(d)=2,OD(d)=1,TD(d)=3

(3)



1. typedef struct EdgeNode

    {

        int adjvex;

        int weight;

        struct EdgeNode\*next;

    }EdgeNode;

    typedef struct VertexNode

    {

        int outdata;

        EdgeNode\*firstedge;

    }VertexNode,AdjList[15];

    typedef struct

    {

        AdjList adjList;

        int numVertexes,numEdges;

    }graphAdjList,\*GraphAdjList;

void CreateAdjList(GraphAdjList &*G*)

    {

        int i,j,k,w;

        EdgeNode \*e;

        cout<<"输入顶点数和边数:\n";

        cin>>*G*->numVertexes>>*G*->numEdges;

        for(int i=0;i<*G*->numVertexes;i++)

        {

            cin>>*G*->adjList[i].outdata;

*G*->adjList[i].firstedge=NULL;

        }

        for(int k=0;k<*G*->numEdges;k++)

        {

            cout<<"输入边(vi,vj)中的顶点下标i,j,以及权值w:\n";

            cin>>i>>j>>w;

            e=new EdgeNode;

            e->adjvex=j;

            e->weight=w;

            e->next=*G*->adjList[i].firstedge;

*G*->adjList[i].firstedge=e;

        }

    }

1. (1)

|  |  |  |
| --- | --- | --- |
|  | ID | OD |
| V1 | 3 | 0 |
| V2 | 2 | 2 |
| V3 | 1 | 1 |
| V4 | 1 | 3 |
| V5 | 2 | 1 |
| V6 | 1 | 3 |

(2)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | V1 | V2 | V3 | V4 | V5 | V6 |
| V1 | 0 | INF | INF | INF | INF | INF |
| V2 | 1 | 0 | INF | 1 | INF | INF |
| V3 | INF | 1 | 0 | INF | INF | INF |
| V4 | INF | INF | 1 | 0 | 1 | 1 |
| V5 | 1 | INF | INT | INF | 0 | INF |
| V6 | 1 | 1 | INF | INF | 1 | 0 |

(3)

|  |  |
| --- | --- |
| V1 | ^ |
| V2 |  | 0 |  | 3 | ^ |
| V3 |  | 1 | ^ |
| V4 |  | 2 |  | 4 |  | 5 | ^ |
| V5 |  | 0 | ^ |
| V6 |  | 0 |  | 1 |  | 4 | ^ |

1. D
2. (1)V1->V2->V3->V6->V4->V5

(2)V1 V2 V3 V4 V5; V1->V2,V1->V3,V1->V4,V1->V5,V2->V6

1. 在一个连通网的所有生成树中，代价之和最小的生成树。
2. D
3. C
4. typedef struct

    {

        char vexs[MAX\_SIZE];

        int edges[MAX\_SIZE][MAX\_SIZE];

        int numVertexes,numEdges;

    }MGraph;

typedef struct

    {

        int begin;

        int end;

        int weight;

    }Edge;

    int Find(int \**parent*,int *f*)

    {

        while(*parent*[*f*]>0)

        {

*f*=*parent*[*f*];

        }

        return *f*;

    }

    void MiniSpanTree\_Kruskal(MGraph *G*)

    {

        int i,n,m;

        Edge edges[100]; *//定义边集数组 begin,end,weight*

        int parent[100]; *//定义parent数组用来判断边与边是否形成环路*

*//邻接矩阵转化为边集数组（省略），并按权值从小到大排序*

        for(int i=0;i<*G*.numVertexes;i++) //n=G.numVertexes

        {

            for(int j=0;j<i;j++)

            {

                if(*G*.edges[i][j]!=65535)

                {

                    edges[i].begin=j;

                    edges[i].end=i;

                    edges[i].weight=*G*.edges[i][j];

                }

            }

        }

        for(int i=0;i<*G*.numVertexes-1;i++)

        {

            Edge temp;

            for(int j=i+1;j<*G*.numEdges;j++) //m=G.numEdges

            {

                if(edges[i].weight>edges[j].weight)

                {

                    temp=edges[i];

                    edges[i]=edges[j];

                    edges[j]=temp;

                }

            }

        }

        for(int i=0;i<*G*.numVertexes;i++)

        {

            parent[i]=0;

        }

        for(int i=0;i<*G*.numEdges;i++)

        {

            n=Find(parent,edges[i].begin);

            m=Find(parent,edges[i].end);

            if(n!=m)*//不形成回路*

            {

                parent[n]=m;*//将这条边的结尾顶点放进下标为起点的parent数组中，表示此项已经在生成树集合中*

                cout<<"("<<edges[i].begin<<","<<edges[i].end<<")"<<edges[i].weight<<" ";

            }

        }

    }

1. D
2. C
3. typedef struct EdgeNode

    {

        int adjvex;

        int weight;

        struct EdgeNode\*next;

    }EdgeNode;

    typedef struct VertexNode

    {

        int outdata;

        EdgeNode\*firstedge;

    }VertexNode,AdjList[15];

    typedef struct

    {

        AdjList adjList;

        int numVertexes,numEdges;

    }graphAdjList,\*GraphAdjList;

bool visited[MAX\_SIZE];

bool isExit(GraphAdjList *G*,int *v1*,int *v2*,int *k*)

    {

        for(int i=0;i<*G*->numVertexes;i++)

        {

            visited[i]=false;

        }

        return exist\_path\_len(*G*,*v1*,*v2*,*k*);

    }

    bool exist\_path\_len(GraphAdjList *G*,int *v1*,int *v2*,int *k*)

    {

        if(*v1*==*v2*&&*k*==0)

            return true;

        else if(*k*>0)

        {

            visited[*v1*]=true;

            for(auto p=*G*->adjList[*v1*].firstedge;p;p=p->next)

            {

                int m=p->adjvex;

                if(!visited[m])

                {

                    if(exist\_path\_len(*G*,m,*v2*,*k*-1))

                        return true;

                }

            }

            visited[*v1*]=false;

        }

        return false;

    }

1. int Graph[n][n];//假设已经建立好邻接矩阵

int P[MAX\_SIZE];

int D[MAX\_SIZE];

void Dijkstra(int *v0*,int *n*)

    {

        int v,w,k,min;

        int final[*MAX\_SIZE*];

        for(v=0;v<*n*;v++)

        {

            final[v]=0;

            D[*n*]=Graph[*v0*][v];

            P[*n*]=0;

        }

        D[*v0*]=0;

        final[*v0*]=1;

        for(v=1;v<*n*;v++)

        {

            min=INT\_MAX;

            for(w=0;w<*n*;w++)

            {

                if(!final[w]&&D[w]<min)

                {

                    k=w;

                    min=D[w];

                }

            }

            final[k]=1;

            for(w=0;w<*n*;w++)

            {

                if(!final[w]&&(min+Graph[k][w]<D[w]))

                {

                    D[w]=min+Graph[k][w];

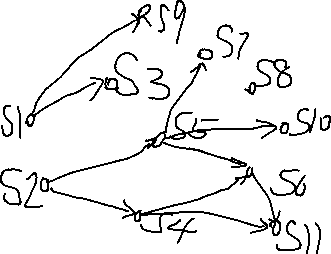
                    P[w]=k;

                }

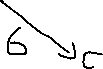
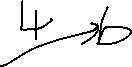
            }

        }

    }



1. (1)



(2)a->b->c->d->f



长度为16

1. C