6. 对于如图8.4所示的带权无向图，给出利用Prim算法（从顶点0开始构造）和Kruskal算法构造出的最小生成树的结果，要求结果按构造边的顺序列出。

答：使用Prime算法得到的结果：（0，1）、（0，3）、（1，2）、（2，5）、（5，4）

使用Kruskal算法构造出的结果：（0，1）、（0，3）、（1，2）、（5，4）、（2，5）

7. 对于一个顶点个数大于4的带权无向图，回答以下问题：

（1）该图的最小生成树一定是唯一的吗？如果所有边的权都不相同，那么其最小生成树一定是唯一的吗？

（2）如果该图的最小生成树不是唯一的，那么调用Prim算法和Kruskal算法构造出的最小生成树一定相同吗？

（3）如果图中有且仅有两条权最小的边，它们一定出现在该图的所有的最小生成树中吗？简要说明回答的理由。

（4）如果图中有且仅有3条权最小的边，它们一定出现在该图的所有的最小生成树中吗？简要说明回答的理由。

答：（1）不唯一，除非所有边的权都不相同。如果所有边的权不同，则生成树一定唯一。

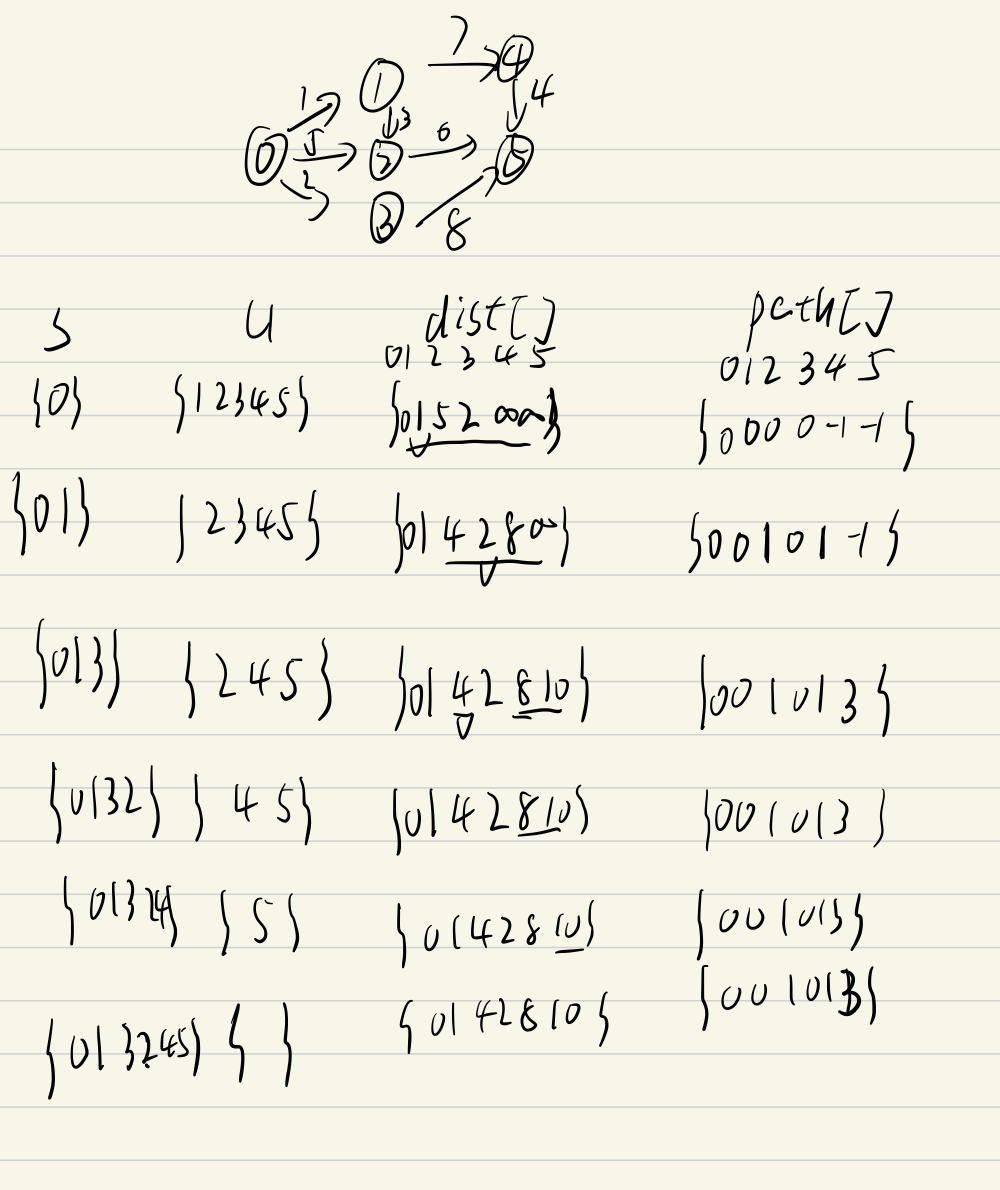
（2）因为该图最小生成树不唯一，则调用两种算法，其结果不一定相同。

（3）它们一定会出现。

证明使用反证法：如果不选择这两条边，假设（a，b）为所有边的权最小的，如果不选择这一条边，则其他选择从a到b的路径，权值一定大于等于（等于为另一个最小边，其一定不会被选中），则生成的不是最小生成树，则证明这两条边必须选中。

（4）不一定，如果选中这三个最小边，出现了回路，则不满足条件，必须舍去其中任意一条。

8.对于如图8.6所示的带权有向图，采用Dijkstra算法求出从顶点0到其他各顶点的最短路径及其长度，要求给出求解过程。



14. 假设不带权有向图采用邻接矩阵g 存储，设计实现以下功能的算法：

（1）求出图中每个顶点的入度。

（2）求出图中每个顶点的出度。

（3）求出图中出度为0的顶点数。

void InDs1(MatGraph g) //求出图G 中每个顶点的入度

{

int i, j, n;

printf("各顶点入度:\n");

for (j = 0;j < g.n;j++)//入度为矩阵中第j行元素之和

{

n = 0;

for (i = 0;i < g.n;i++)

if (g.edges[i][j] != 0)

n++; //n累计入度数

printf(" 顶点%d:%d\n", j, n);

}

}

void OutDs1(MatGraph g) //求出图G中每个顶点的出度

{

int i, j, n;

printf("各顶点出度:\n");

for (i = 0;i < g.n;i++)//出度为矩阵中第i行元素之和

{

n = 0;

for (j = 0;j < g.n;j++)

if (g.edges[i][j] != 0)

n++; //n累计出度数

printf(" 顶点%d:%d\n", i, n);

}

}

void ZeroOutDs1(MatGraph g) //求出图G中出度为0的顶点个数

{

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) //存在一条出边

n++;

if (n == 0)

printf("%2d\n", i);

}

printf("\n");

}

15. 假设不带权有向图采用邻接表G存储，设计实现以下功能的算法：

（1）求出图中每个顶点的入度。

（2）求出图中每个顶点的出度。

（3）求出图中出度为0的顶点数。

void InDs2(AdjGraph\* G) //求出图G中每个顶点的入度

{

ArcNode\* p;

int A[MAXV], i; //A存放各顶点的入度

for (i = 0;i < G->n;i++) //A中元素置初值0

A[i] = 0;

for (i = 0;i < G->n;i++) //扫描所有头结点

{

p = G->adjlist[i].firstarc;

while (p != NULL) //扫描边结点

{

A[p->adjvex]++; //表示i到p->adjvex顶点有一条边

p = p->nextarc;

}

}

printf("各顶点入度:\n"); //输出各顶点的入度

for (i = 0;i < G->n;i++)

printf(" 顶点%d:%d\n", i, A[i]);

}

void OutDs2(AdjGraph\* G) //求出图G中每个顶点的出度

{

int i, n;

ArcNode\* p;

printf("各顶点出度:\n");

for (i = 0;i < G->n;i++) //扫描所有头结点

{

n = 0;

p = G->adjlist[i].firstarc;

while (p != NULL) //扫描边结点

{

n++; //累计出边的数

p = p->nextarc;

}

printf(" 顶点%d:%d\n", i, n);

}

}

void ZeroOutDs2(AdjGraph\* G) //求出图G中出度为0的顶点数

{

int i, n;

ArcNode\* p;

printf("出度为0的顶点:");

for (i = 0;i < G->n;i++) //扫描所有头结点

{

p = G->adjlist[i].firstarc;

n = 0;

while (p != NULL) //扫描边结点

{

n++; //累计出边的数

p = p->nextarc;

}

if (n == 0) //输出出边数为0的顶点编号

printf("%2d", i);

}

printf("\n");

}

20. 对于一个带权有向图，设计一个算法输出从顶点i到顶点j的所有路径及其路径长度。调用该算法求出《教程》图8.35中顶点0到顶点3的所有路径及其长度。

函数如下：

int visited[MAXV];

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

{ //d表示path中顶点个数，初始为0；length表示路径长度，初始为0

int w, i;

ArcNode\* p;

path[d] = u; d++; //顶点u加入到路径中，d增1

visited[u] = 1; //置已访问标记

if (u == v && d > 0) //找到一条路径则输出

{

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; //p指向顶点u的第一个邻接点

while (p != NULL)

{

w = p->adjvex; //w为顶点u的邻接点

if (visited[w] == 0) //若w顶点未访问,递归访问它

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

p = p->nextarc; //p指向顶点u的下一个邻接点

}

visited[u] = 0; //恢复环境,使该顶点可重新使用

}

编写主函数求取结果：

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); //创建《教程》中图8.35的邻接表

printf("图G的邻接表:\n");

DispAdj(G); //输出邻接表

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;

}

结果如下：

