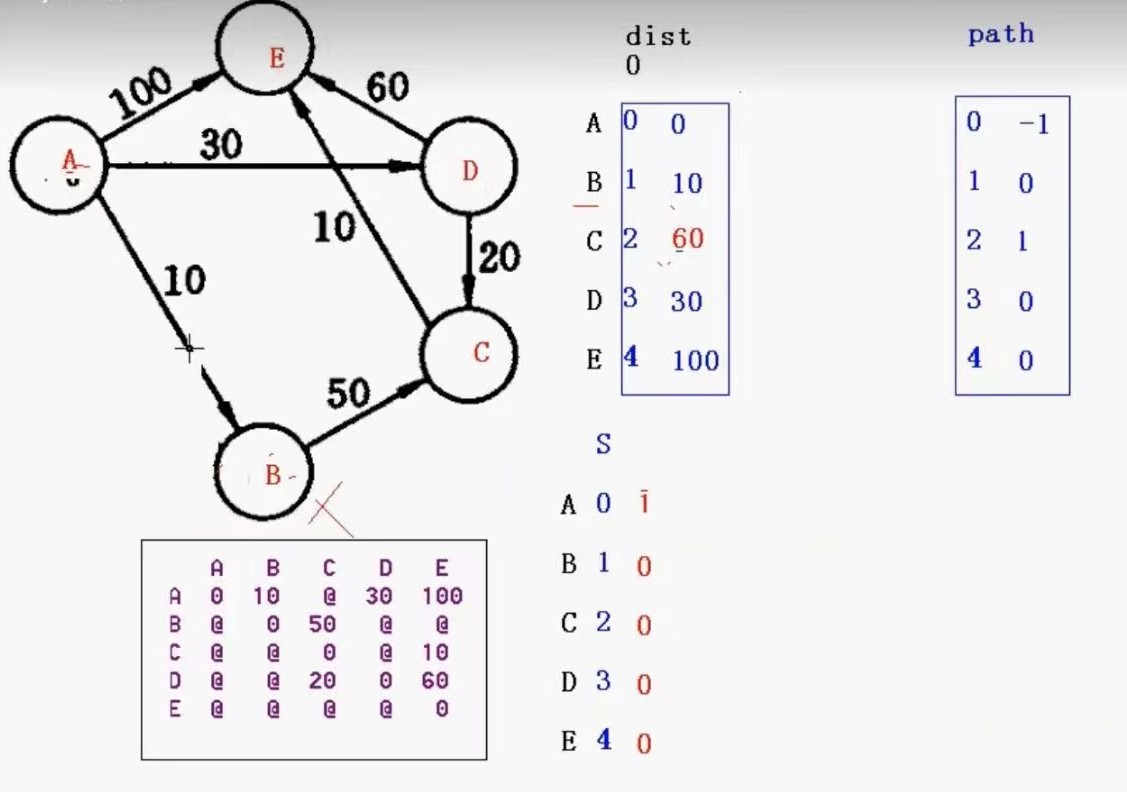
**最短路径求解（Dijkstra算法实现）**

****

E GetWeight(GraphMtx& g, int v1, int v2) {

if (v1 == -1 || v2 == -1) {

return MAX\_COST;

}

return g.Edge[v1][v2];

}

//按照路径递增找到最短路径

void ShortestPath(GraphMtx& g,T vertex,E dist[],int path[]) {

int n = g.NumVertices;

bool\*S = (bool\*)malloc(sizeof(bool)\*n);

assert(S != nullptr);

int v = GetVertexPos(g, vertex);

//初始化

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

dist[i] = GetWeight(g, v, i);

S[i] = false;

if (i != v && dist[i] < MAX\_COST) {

path[i] = v;

}

else {

path[i] = -1; //不能直达的情况

}

}

S[v] = true; //表示起始点已经加入了集合

int min;

for (int i = 0; i < n-1; i++) { //这里的i表示循环的次数

min = MAX\_COST;

int u = v; //u起记录作用

for (int j = 0; j < n; j++) { //这里的j代表顶点的下标 //一轮循环下来找到的min值为A-B的权值，此时u==1（表示路线是从A-B）

if (!S[j] && dist[j] < min) {

u = j;

min = dist[j];

}

}

S[u] = true;

//(第一轮)下面找顶点B到各顶点的最小值

for (int k = 0; k < n; k++) {

int w = GetWeight(g, u, k);

if (!S[k] && w < MAX\_COST && dist[u] + w < dist[k]) { //!S[k]说明要找不在集合里的点 // w<MAX\_COST表示要能够到达 //

dist[k] = dist[u] + w;

path[k] = u;

}

}

}

}

int main() {

GraphMtx gm;

InitGraph(gm);

InsertVertex(gm, 'A');

InsertVertex(gm, 'B');

InsertVertex(gm, 'C');

InsertVertex(gm, 'D');

InsertVertex(gm, 'E');

InsertEdge(gm, 'A', 'B', 10);

InsertEdge(gm, 'A', 'D', 30);

InsertEdge(gm, 'A', 'E', 100);

InsertEdge(gm, 'B', 'C', 50);

InsertEdge(gm, 'C', 'E', 10);

InsertEdge(gm, 'D', 'C', 20);

InsertEdge(gm, 'D', 'E', 60);

ShowGraph(gm);

int n = gm.NumVertices;

E \*dist = (E\*)malloc(sizeof(int)\*n); //记录到各个顶点的最短路径长度，下标即为顶点的下标

int \*path = (int\*)malloc(sizeof(int)\*n); //记录最短路径的起始值 path[4]=2;path[2]=3;path[3]=0; 则表示到E点的最短路径为A-D-C-E

assert(dist != nullptr&&path != nullptr);

ShortestPath(gm, 'A', dist, path);

}