**int main() // for MST**

{

dgraph T;

T.filltable(); // for each adjacent vertex, the cost to it is stored

T.makeTree(0); // 0th entry is marked as Tree. This is your starting vertex.

T.displayTable();

while (!T.allTree()) // until all vertices are in Tree

{

T.makeTree(T.findSmallest()); // Find a Fringe vertex with smallest cost and mark it as Tree

// Note that this causes its adjacent vertices to be marked as Fringe

// and updates or enters DISTO for them

T.displayTable();

}

// MST has been found – all the vertices in it are marked as Tree

T.displayMST(); // display the edges of the MST

}//end of main

**int main() // for Shortest Path**

{

dgraph T;

T.filltable(); // for each adjacent vertex, the cost to it is stored

T.makeTree(0); // 0th entry is now in Tree. This is your starting vertex.

T.displayTable();

while (!T.allTree()) // Until all vertices are in Tree

{

T.makeTree(T.findSmallest()); // Find a Fringe vertex with smallest total distance and mark it as Tree

// Note that this causes its adjacent vertices to be marked as Fringe

// and updates or enters DISTO for them

T.displayTable();

}

// The shortest path from the 0's vertex to all others have been determined.

cout << "Enter a goal vertex or Q: ";

cin >> a;

while (a != 'Q')

{ T.displayPath(a); // display the shortest path to a – use the FROM entries to backtrace

cout << "Enter a goal vertex or Q: ";

cin >> a;

}

}// end of main