Computing Science 204 (B1)

The Dijkstra/Prim Algorithm for Minimal Spanning Trees

Thursday, August 10

Given a graph $G = (V, E)$, with vertex set V and edge set E, with each edge $e = \{a, b\}$ of this graph we assign a positive real number called the *weight* of e and denoted by wt(e) or wt($\{a, b\}$). The graph G with such an assignment of weights to its edges is called a *weighted* graph (also called a *network*).

A *tree* is a graph which is connected and contains no cycles. A *spanning tree* of a graph G is a subgraph which is a tree and which spans G, that is, contains every vertex of G.

The Dijkstra/Prim algorithm is a greedy algorithm which generates a minimal spanning tree of a weighted graph G, that is, a spanning subgraph of G which forms a tree and which has the property that no other spanning tree has a smaller sum of weights on its edges. The algorithm will either find a minimal spanning tree or conclude that the network does not have a spanning tree, that is, is not connected.

Given a weighted graph $G = (V, E)$, the Dijkstra/Prim algorithm finds a spanning tree which minimizes the sum of the weights of the edges by building up the tree one edge at a time. It begins by selecting an arbitrary starting vertex, and then "branches out" from the part of the tree constructed so far by choosing a new vertex and new edge at each iteration. During the course of the algorithm, the vertices may be thought of as being divided into three disjoint categories:

Tree vertices: in the tree constructed so far.

Fringe vertices: not in the tree, but adjacent to some vertex in the tree.

Unseen vertices: all others.

The algorithm always chooses an edge from a tree vertex to a fringe vertex which has minimum weight.

A description of the algorithm is given below:

Dijkstra/Prim Algorithm
Input: A weighted graph $G = (V, E)$.
Output: A set of edges defining a minimal spanning tree of G or the message that G is disconnected.

(i) Select an arbitrary vertex to start the tree

(ii) While there are fringe vertices, select an edge of minimum weight between a tree vertex and a finge vertex, and add the selected edge and fringe vertex to the tree

(iii) If T has $n - 1$ edges (where n is the number of vertices of G), stop and output T; else output the message that the graph has no spanning tree.

The following C program implements the above algorithm, it takes as its input a weighted undirected graph G, and has as its output a minimal spanning tree T for the graph G.

The input consists of data representing a number of weighted undirected graphs, each specified by

- A natural number N, where $N \leq 100$, indicating the number of vertices in the graph (the vertices will be labeled $1, 2, 3, \cdots, N$) and `0` signals the end of data for the vertices.

- A number of triples `a b c` where the pair `a b` represents an edge as follows: The elements `a b` are natural numbers indicating the vertices which are the endpoints of the edge; and the positive real number `c` represents the weight of this edge. The end of the data for the current graph will be signalled by the triple `0 0 0`.

For each graph the output is:

- A list of the vertices and the edges with their weights (i.e. echo the input).
- If the graph is not connected, a message so indicating.
- If the graph is connected, a list of the edges in the minimal spanning tree together with the sum of the weights.

An implementation of this algorithm can be found in the book *Computer Algorithms: Introduction to Design and Analysis*, by Sara Baase, on pages 162–166.

Another implementation is given below.

```
/* PRIM'S ALGORITHM TO FIND A MINIMUM SPANNING TREE */

#include<stdio.h>
#define maxV 100
#define maxE (maxV*(maxV-1)/2)

struct edge
{
   int v1;
   int v2;
   int w;  /* weights must be positive */
   int in_tree;
} *edges;

int q=0,V,E,End,weight;

build_edge_list()
{                              /* sorts the edges as they come in */
   int i,j,k,x,y,w,slot;
   int tempv1,tempv2,tempw;
   printf("Enter the number of Vertices followed by 0:\n");
   scanf("%d%d",&V,&End);
   edges = (struct edge *)malloc(maxE*sizeof(struct edge));
   printf("Enter the edges and weights: x y w\n");
   scanf("%d%d%d",&x,&y,&w);
   edges[0].v1 = x; edges[0].v2 = y; edges[0].w = w;
   edges[0].in_tree = 0;
   i = 0; k = 0;
   while (x+y+w != 0)
   {   scanf("%d %d %d",&x,&y,&w);
       edges[i+1].v1 = x; edges[i+1].v2 = y; edges[i+1].w = w;
       edges[i+1].in_tree = 0;
       tempv1 = edges[i+1].v1; tempv2 = edges[i+1].v2; tempw = edges[i+1].w;
       slot = (edges[i].w > tempw);
       j = i;
       while (slot)
       {   edges[j+1].v1 = edges[j].v1; edges[j+1].v2 = edges[j].v2;
           edges[j+1].w = edges[j].w;
           j--;
           if (j >= 0)
               slot = (edges[j].w > tempw);
           else
               slot = 0;
       }
       edges[j+1].v1 = tempv1; edges[j+1].v2 = tempv2;
       edges[j+1].w = tempw;
       i++; k++;
   }
   E = k;   /* Number of Edges */
}
```

```
build_spanning_tree(s)
int s;
{  int i,k,w,found,*nodes,*rest;
   int rest_not_empty;
   nodes = (int *)malloc((V+1)*sizeof(int));
   rest = (int *)malloc((V+1)*sizeof(int));
   for (k = 1; k <= V; k++)
   {   nodes[k] = 0; rest[k] = 1;
   }
   nodes[s] = 1; rest[s] = 0;
   found = 0; k = 1;
   while (k <= E && !found)
   {   if (edges[k].v1 == s || edges[k].v2 == s)
       {   found = 1;
           if (edges[k].v1 == s)
               w = edges[k].v2;
           else
               w = edges[k].v1;
           edges[k].in_tree = 1;
           nodes[w] = 1; rest[w] = 0;
       }
       k++;
   }
   rest_not_empty = V-2;
   while (rest_not_empty)
   {
       found = 0; k = 1;
       while (k <= E && !found)
       {   if ((nodes[edges[k].v1] == 1 && nodes[edges[k].v2] == 0)
               || (nodes[edges[k].v2] == 1 && nodes[edges[k].v1] == 0))
           {   found = 1;
               edges[k].in_tree = 1;
               if (nodes[edges[k].v1])
               {   nodes[edges[k].v2] = 1;
                   rest[edges[k].v2] = 0;
               }
               else if (nodes[edges[k].v2])
               {   nodes[edges[k].v1] = 1;
                   rest[edges[k].v1] = 0;
               }
           }
           k++;
       }
       rest_not_empty--;
   }
   for (k = 1; k <= E; k++)
       if (edges[k].in_tree == 1)
           q++;
   if (q == V-1)
       print_output(edges,E);
   else
       printf("The graph is not connected. It has no spanning tree.\n");
}
```

```
echo_input(e,N)
struct edge *e;
int N;
{
   int k;
   printf("\nThe Edges Input:\n");
   for (k = 1; k <= N; k++)
   {
    printf("edge %d: %4d%4d weight: %4d\n",k,e[k].v1,e[k].v2,e[k].w);
   }
   printf("Number of Edges: %d\n",N);
}

print_output(e,N)
struct edge *e;
int N;
{
   int k;
   weight = 0;
   printf("\nThe Minimal Spanning Tree:\n");
   for (k = 1; k <= N; k++)
   {
       if (e[k].in_tree == 1)
       {
           printf("edge %d: %4d%4d weight: %4d\n",k+1,e[k].v1,e[k].v2,e[k].w);
           weight = weight + e[k].w;
       }
   }
   printf("The Weight of a Minimal Spanning Tree is: %d\n",weight);
   printf("Number of Edges in the Minimal Spanning Tree is: %d\n",q);
}

main()
{
   build_edge_list();
   echo_input(edges,E);
   build_spanning_tree(1);
}
```