Topological Sorting

#include&lt;stdio.h&gt;

#define SIZE 10

#define MAX 10

int G[SIZE][SIZE], i, j, k;

int front, rear;

int n, edges;

int b[SIZE], Q[SIZE], indegree[SIZE];

int create()

{

front = -1; rear = -1;

for (i=0; i&lt;MAX; i++) //initialising the graph

{

for (j = 0; j&lt;MAX; j++)

{

G[i][j] = 0;

}

}

for (i=0; i&lt;MAX; i++)

{

indegree[i] = -99;

}

n = 5;

edges=7;

G[0][2]=1;

G[0][3]=1;

G[1][0]=1;

G[1][3]=1;

G[2][4]=1;

G[3][2]=1;

G[3][4]=1;

return n;

}

void Display(int n)

{

int V1, V2;

for (V1= 0; V1&lt;n; V1++)

{

for (V2= 0; V2&lt;n; V2++)

printf(&quot;%d&quot;, G[V1][V2]);

printf(&quot;\n&quot;);

}

}

void Insert\_Q(int vertex, int n)

{

if (rear == n)

printf(&quot;Queue Overflow\n&quot;);

else

{

if (front == -1)/\*Empty Queue condition\*/

front =0;

rear=rear + 1;

Q[rear]=vertex;/\* Inserting node into the Q\*/

}

}

int Delete\_Q()

{

int item;

if (front==-1||front &gt; rear)

{

printf(&quot;Queue Underflow\n&quot;);

return -1;

}

else

{

item=Q[front];

front = front + 1;

return item;

}

}

int Compute\_Indeg(int node, int n)

{

int v1, indeg\_count=0;

for (v1 = 0; v1&lt;n; v1++)

if (G[v1][node] == 1)//checking for incoming edge

indeg\_count++;

return indeg\_count++;

}

void Topo\_ordering(int n)

{

j = 0;

for (i=0; i&lt;n; i++)

{

indegree[i] = Compute\_Indeg(i, n);

if (indegree[i]==0)

Insert\_Q(i, n);

}

while (front &lt;= rear)

{

k = Delete\_Q();

b[j++] = k;

for (i=0; i&lt;n; i++)

{

if (G[k][i]==1)

{

G[k][i] = 0;

indegree[i]=indegree[i] - 1;

if (indegree[i]==0)

Insert\_Q(i, n);

}

}

}

printf(&quot;\nThe result of after topological sorting is ...&quot;);

for (i=0; i&lt;n; i++)

printf(&quot;%d&quot;,b[i]);

printf(&quot;\n&quot;);

}

int main()

{

n = create();

printf(&quot;The adjacency matrix is : \n&quot;);

Display(n);

Topo\_ordering(n);

return 0;

}

Minimum spanning tree using prims algorithm

#include&lt;stdio.h&gt;

#define SIZE 20

#define INFINITY 32767

/\*This function finds the minimal spanning tree by Prim&#39;s

Algorithm \*/

void Prim(int G[][SIZE], int nodes)

{

int tree[SIZE], i, j, k;

int min\_dist, v1, v2,total=0;

// Initialize the selected vertices list

for (i=0; i&lt;nodes; i++)

tree[i] = 0;

printf(&quot;\n\n The Minimal Spanning Tree Is :\n&quot;);

tree[0] = 1;

for (k=1; k&lt;=nodes-1; k++)

{

min\_dist = INFINITY;

//initially assign minimum dist as infinity

for (i=0; i&lt;=nodes-1; i++)

{

for (j=0; j&lt;=nodes-1; j++)

{

if (G[i][j] &amp;&amp; ((tree[i] &amp;&amp; !tree[j]) || (!tree[i] &amp;&amp; tree[j])))

{

if (G[i][j] &lt;min\_dist)

{

min\_dist=G[i][j];

v1 = i;

v2 = j;

}

}

}

}

printf(&quot;\n Edge (%d %d ) and weight = %d&quot;,v1,v2,min\_dist);

tree[v1] = tree [v2] = 1;

total = total+min\_dist;

}

printf(&quot;\n\n\t Total Path Length Is = %d&quot;,total);

}

void main()

{

int G[SIZE][SIZE], nodes;

int v1, v2, length, i, j, n;

printf(&quot;\n\t Prim&#39;S Algorithm\n&quot;);

printf(&quot;\n Enter Number of Nodes in The Graph &quot;);

scanf(&quot;%d&quot;,&amp;nodes);

printf(&quot;\n Enter Number of Edges in The Graph &quot;);

scanf(&quot;%d&quot;,&amp;n);

for (i=0; i&lt;nodes; i++) // Initialize the graph

for (j=0; j&lt;nodes; j++)

G[i][j] = 0;

//entering weighted graph

printf(&quot;\n Enter edges and weights \n&quot;);

for (i=0; i&lt;n; i++)

{

printf(&quot;\n Enter Edge by V1 and V2 :&quot;);

printf(&quot;[Read the graph from starting node 0]&quot;);

scanf(&quot;%d %d&quot;, &amp;v1,&amp;v2);

printf(&quot;\n Enterscanf(&quot;%d&quot;, &amp;length);

G[v1][v2] = G[v2][v1] = length;

}

printf(&quot;\n\t&quot;);

Prim(G,nodes);

} ;