Assignment 9 (Implement a Shortest Path Algorithm)

#include <stdio.h>

#include<stdlib.h>

#define TRUE 1

#define FALSE 0

#define MAX\_NUMBER\_OF\_NODES 50

typedef struct EdgeNodeTag

{

int IndexOfEndpoint;

int Time;

struct EdgeNodeTag \*NextEdge;

} EdgeNodeType;

typedef struct

{

int NumberOfNodes;

int NumberOfEdges;

EdgeNodeType \*EdgeListHead[MAX\_NUMBER\_OF\_NODES];

} GraphType;

typedef struct

{

int AccessedFromIndex;

int CostFromSource;

int StillInConsideration;

} StatusOfNodeType;

GraphType Graph;

StatusOfNodeType StatusOfNode[MAX\_NUMBER\_OF\_NODES];

int StartNode, EndNode;

int main(void)

{

void explore(int CurrentNodeIndex);

int HeightOfX;

int c1, c2, t, d; /\* same as in problem stmt. \*/

int ReversePath[MAX\_NUMBER\_OF\_NODES];

int i, j;

EdgeNodeType \*EdgeNodePtr;

scanf("%d %d %d", &HeightOfX, &(Graph.NumberOfEdges), &(Graph.NumberOfNodes));

for(i=0; i<Graph.NumberOfNodes; i++) Graph.EdgeListHead[i] = NULL;

for(i=0; i < Graph.NumberOfEdges; i++)

{

scanf("%d %d %d %d", &c1, &c2, &t, &d);

if (d < HeightOfX)

{

/\* edge exists (for X), since water level is less than X's height \*/

EdgeNodePtr = (EdgeNodeType \*) malloc(sizeof(EdgeNodeType));

(\*EdgeNodePtr).IndexOfEndpoint = c2 - 1;

(\*EdgeNodePtr).Time = t;

(\*EdgeNodePtr).NextEdge = Graph.EdgeListHead[c1 - 1];

Graph.EdgeListHead[c1 - 1] = EdgeNodePtr;

EdgeNodePtr = (EdgeNodeType \*) malloc(sizeof(EdgeNodeType));

(\*EdgeNodePtr).IndexOfEndpoint = c1 - 1;

(\*EdgeNodePtr).Time = t;

(\*EdgeNodePtr).NextEdge = Graph.EdgeListHead[c2 - 1];

Graph.EdgeListHead[c2 - 1] = EdgeNodePtr;

}

}

/\*

for(i=0; i<Graph.NumberOfNodes; i++)

{

printf("%d ", i);

EdgeNodePtr = Graph.EdgeListHead[i];

while (EdgeNodePtr != NULL)

{

printf("(%d %d) ", (\*EdgeNodePtr).IndexOfEndpoint,

(\*EdgeNodePtr).Time);

EdgeNodePtr = (\*EdgeNodePtr).NextEdge;

}

printf("\n");

}

printf("thru with input ...\n");

\*/

for(i=0; i<Graph.NumberOfNodes; i++)

{

StatusOfNode[i].AccessedFromIndex = -1;

StatusOfNode[i].CostFromSource = -1;

StatusOfNode[i].StillInConsideration = TRUE;

}

/\*

printf("thru with status of node initialization ...\n");

\*/

scanf("%d %d", &StartNode, &EndNode);

StatusOfNode[StartNode - 1].CostFromSource = 0;

StatusOfNode[StartNode - 1].StillInConsideration = FALSE;

if (StartNode != EndNode)

{

explore(StartNode - 1);

}

/\*

printf("thru with exploration ...\n");

\*/

i = 0; j = EndNode - 1;

ReversePath[i] = j + 1;

while (StatusOfNode[j].AccessedFromIndex != -1)

{

/\*

printf("ReversePath while(%d)...\n", StatusOfNode[j].AccessedFromIndex);

\*/

j = StatusOfNode[j].AccessedFromIndex;

ReversePath[++i] = j + 1;

}

for(j = i; j >= 0; j--)

{

printf("%d ", ReversePath[j]);

}

printf("\n");

return 0;

}

void explore(int CurrentNodeIndex)

{

EdgeNodeType \*EdgeNodePtr;

int MinCostAmongstRemainingNodes, IndexOfNextNodeToBeExplored;

int i;

/\*

printf("explore(%d): ...\n", CurrentNodeIndex);

\*/

EdgeNodePtr = Graph.EdgeListHead[CurrentNodeIndex];

while (EdgeNodePtr != NULL)

{

if (((StatusOfNode[(\*EdgeNodePtr).IndexOfEndpoint].AccessedFromIndex == -1)

&&

(StatusOfNode[(\*EdgeNodePtr).IndexOfEndpoint].StillInConsideration)

)

||

((StatusOfNode[CurrentNodeIndex].CostFromSource + (\*EdgeNodePtr).Time)

<

StatusOfNode[(\*EdgeNodePtr).IndexOfEndpoint].CostFromSource

)

)

{

StatusOfNode[(\*EdgeNodePtr).IndexOfEndpoint].CostFromSource =

StatusOfNode[CurrentNodeIndex].CostFromSource + (\*EdgeNodePtr).Time;

StatusOfNode[(\*EdgeNodePtr).IndexOfEndpoint].AccessedFromIndex =

CurrentNodeIndex;

}

/\*

printf("(%d %d %d) ",

(\*EdgeNodePtr).IndexOfEndpoint,

(\*EdgeNodePtr).Time,

StatusOfNode[(\*EdgeNodePtr).IndexOfEndpoint].CostFromSource

);

\*/

EdgeNodePtr = (\*EdgeNodePtr).NextEdge;

}

/\*

printf(" after while loop ...\n");

\*/

MinCostAmongstRemainingNodes = -1;

for(i=0; i<Graph.NumberOfNodes; i++)

{

if ((StatusOfNode[i].StillInConsideration)

&&

(StatusOfNode[i].AccessedFromIndex != -1)

)

{

if (MinCostAmongstRemainingNodes == -1)

{

MinCostAmongstRemainingNodes = StatusOfNode[i].CostFromSource;

IndexOfNextNodeToBeExplored = i;

/\*

printf("Min changes to %d having consideration %d\n", i,

StatusOfNode[i].StillInConsideration);

\*/

}

else

{

if (StatusOfNode[i].CostFromSource < MinCostAmongstRemainingNodes)

{

MinCostAmongstRemainingNodes = StatusOfNode[i].CostFromSource;

IndexOfNextNodeToBeExplored = i;

/\*

printf("Min changes to %d having consideration %d\n", i,

StatusOfNode[i].StillInConsideration);

\*/

}

}

}

}

/\*

printf(" after for loop, index of next node to be explored %d ...\n\n",

IndexOfNextNodeToBeExplored);

\*/

StatusOfNode[IndexOfNextNodeToBeExplored].StillInConsideration = FALSE;

if (IndexOfNextNodeToBeExplored != (EndNode - 1))

{

explore(IndexOfNextNodeToBeExplored);

}

}