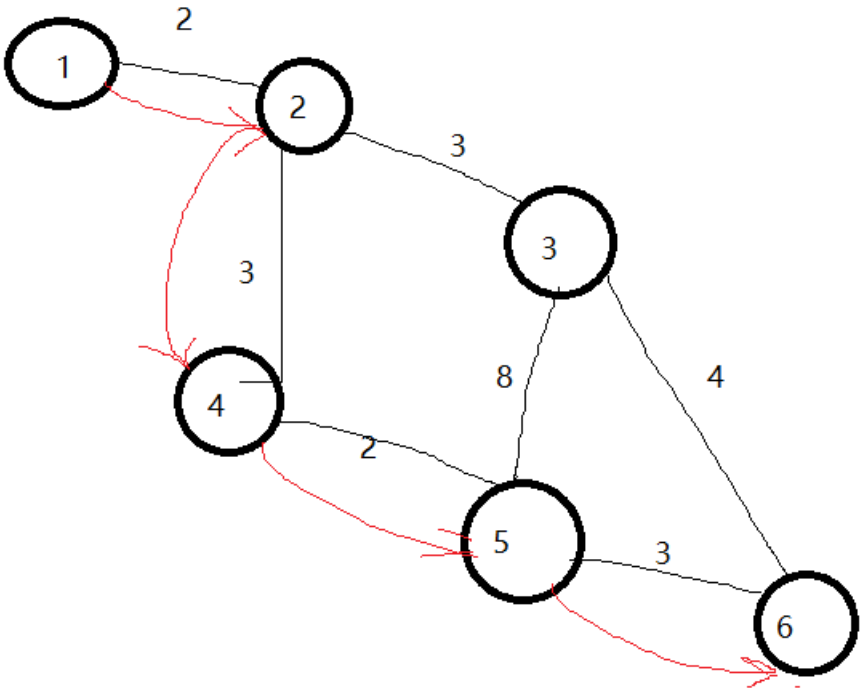
Project2

The 2nd-shortest Path

November 28, 2020



Chapter 1: Introduction

This function is to design an algorithm which is able to implement “The 2nd-shortest Path” problem. That is to say, the path is longer than the shortest path but no longer than any other path at the same time. One possible application that might occur in our daily life is as follows. Suppose that Lisa wants to go home by train. She has decided to take the second-shortest route rather than the shortest one and our job is to help her find the path. There are M stations and N unidirectional railways between these stations. For simplicity, Lisa starts at No. 1 station and her home is at No. M station. What deserves our attention is that firstly, as we have already mastered the Dijkstra Algorithm which is used to find the shortest path, all we need to do is to just make some simple adjustment to the Dijkstra Algorithm so that it can find the 2nd shortest path rather than the shortest one, and the key to do this is to change the judge conditions in the for-loop. Secondly, if there exists two or more shortest path, the 2nd shortest path is the one that is exactly longer than these shortest ones but no longer than any other ones except for the shortest ones. Thirdly, backtrack is OK, in other words, it is legal to use the same road more than once.

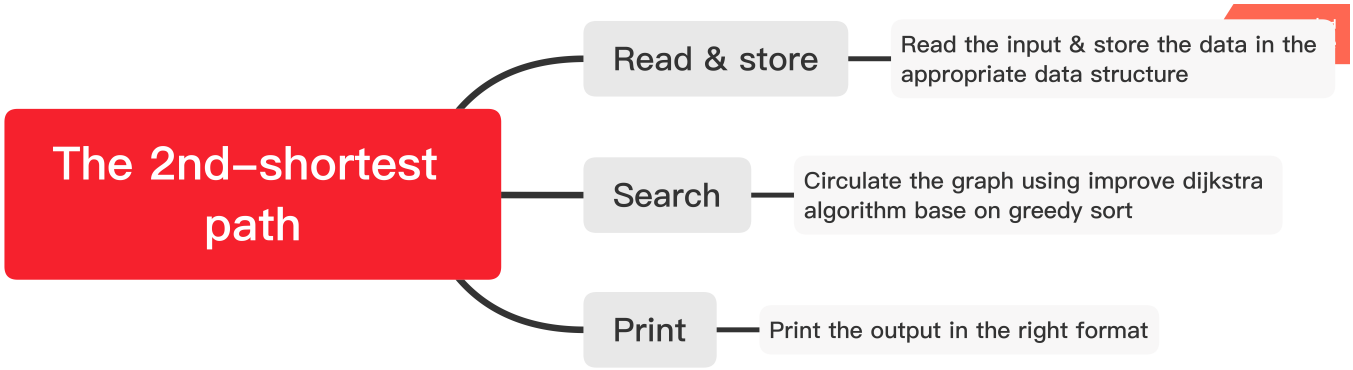
Chapter 2: Algorithm Specification

To implement this function, we need to do some simple improvement of the Dijkstra Algorithm. The Dijkstra Algorithm only need to store the shortest path and update the values based on the value of the shortest path. However, in this “finding the 2nd shortest path” condition, we need to do a little further, that is, to store 2 values, shortest and 2nd shortest. As a consequence, each circular is comparison of three values, the shortest path, the 2nd shortest path, and the present value of the distance of path. The configuration of this project has 3 parts:

```
-Read & store
 Read the input & store the data in the appropriate data structure.

-Search
 Circulate the graph using improve dijkstra algorithm base on greedy sort.

-Print
 Print the output in the right format.
```



Based on this idea, relevant algorithm and data strutures are as follow:

2.1 Graph

The struct Graph, is used to record the infomation of each vertex, which consists of the edges of each vertex, and the total number of edges of each vertex.

```
typedef struct VNode Graph;
struct VNode{
    PtrToAdjVNode Edge[MaxVertexNum];
    int RailwayNum;
};

typedef struct AdjVNode PtrToAdjVNode;
struct AdjVNode{
    Vertex AdjV;
    int Length;
};

typedef int Vertex;
```

2.2 List

The struct List is used to record the shortest and the 2nd-shortest path. Meanwhile, it also has the function of recording the last points of both the shortest and the 2nd-shortest path, respectively.

```
typedef struct ShortestDistance List;
struct ShortestDistance{
    int Shortest;
    int Last;
    int Shortest_2nd;
    int Last_2nd;
};
```

2.3 Read

In the N-for-loop, the inputs are the vertex a, b, and the distance between them. Read these inputs and then store them in the Graph A[M+1]. Integer indexa is used to store the value of the RailwayNum of the vertex a; Integer indexb is used to store the value of the RailwayNum of the vertex b.

```
for (i = 0; i < N; i++)
    {
        int a, b, dis, indexa, indexb;
        scanf("%d %d %d", &a, &b, &dis);

        indexa = A[a].RailwayNum;
        indexb = A[b].RailwayNum;

        A[a].Edge[indexa].AdjV = b;
        A[a].Edge[indexa].Length = dis;
```

```
        A[b].Edge[indexb].AdjV = a;
        A[b].Edge[indexb].Length = dis;

        A[a].RailwayNum++;
        A[b].RailwayNum++;
    }
```

2.4 Search in Dijkstra Algorithm

2.4.1 Queue

integer array Queue[N+1] is used to implement the function of Enqueue and Dequeue. The integer array Queue[N+1] is a circular queue used to store the vertex when serching the graph using greedy algorithm. Moreover, the integers front and rear is used to point at the location of the latest point in the queue and the oldest point remains in the queue so that the functions Enqueue and Dequeue can be realized successfully.

```
int Queue[N+1];
int front, rear;
front = rear = 0;
```

2.4.2 List Dis[M+1]

List Dis[M+1] here is used to record the data of the shortest path of each vertex. Firstly, initialize the Dis[1].shortest and the Dis[1].shortest_2nd as 0, Dis[1].Last and Dis[1].Last_2nd are initialized as 1.

```
List Dis[M+1];
Dis[1].Shortest = 0;
Dis[1].Shortest_2nd = 0;
Dis[1].Last = 1;
Dis[1].Last_2nd = 1;
```

After that,Initialize both the lengths of the shortest and 2nd-shortest paths from 1 to each vertex as infinity, which is -1 In the numerical.

```
#define Infinity -1
for (i = 2; i < M+1; i++)
    {
        Dis[i].Shortest = Infinity;
        Dis[i].Shortest_2nd = Infinity;
    }
```

2.4.3 Dijkstra-Plus Algorithm

The key point of this part is that we not only store the shortest path, but also do we store the 2nd-shortest path. What we should also pay attention to is these integers below, which are used to store some specific values later:

```
/*
temp: store the former value of the shortest path of the point.
temp_2nd: store the former value of the 2nd-shortest path of the point.
point: The point which we are going to deal with.
distance: the distance from the former point to the point we are going to deal
with.
last: The fomer point.
*/
int temp, temp_2nd, point, distance, last;
```

if the queue is not empty, continue the circular:

```
while(front != rear)
```

find the next vertex by using the for-loop bellow:

```
for (i = 0; i < A[last].RailwayNum; i++)
```

There are many conditions. Firstly, if the temp is Infinity, which means that the shortest path of this vertex has not yet been found, so the distance we read at the present is supposed to be the shortest path for a while:

```
if (temp == Infinity)
                {
                    Dis[point].Shortest = distance + Dis[last].Shortest;
                    Dis[point].Last = last;
                }
```

Secondly, if the shortest path has been found, but not for the 2nd-shortest path, we need to do some further judges:

```
else if (temp != Infinity && temp_2nd == Infinity)
                {
                    if (point == M)
                    {
                        if (temp == distance + Dis[last].Shortest)
                        {
                            Queue[front] = point;
                            front++;
                            front%=(N+1);
```

```
                        continue;
                    }
                }

                Dis[point].Shortest_2nd = distance + Dis[last].Shortest;
                Dis[point].Last_2nd = last;

                /*
                shortest > shortest_2nd,
                swap is needed.
                */
                if (Dis[point].Shortest > Dis[point].Shortest_2nd)
                {
                    int buffer = Dis[point].Shortest;
                    Dis[point].Shortest = Dis[point].Shortest_2nd;
                    Dis[point].Shortest_2nd = buffer;

                    buffer = Dis[point].Last;
                    Dis[point].Last = Dis[point].Last_2nd;
                    Dis[point].Last_2nd = buffer;
                }

                /*
                If shortest <= shortest_2nd,
                then traceback is needed to change the value of the last_2nd
                for every vertex in the path.
                */
                else
                {
                    int obj = last;
                    while (1)
                    {
                        Dis[obj].Last_2nd = Dis[obj].Last;
                        obj = Dis[obj].Last;
                        if (obj == 1) break;
                    }
                }
            }
```

Thirdly, if the shortest and 2nd-shortest path have been found, we need to compare the values of the shortest, 2nd-shortest, and the present distance to decide if we need to do swap job.

```
else if (temp_2nd > distance + Dis[last].Shortest)
            {
                if (point == M)
                {
                    if (temp == distance + Dis[last].Shortest)
                    {
                        Queue[front] = point;
                        front++;
                        front%=(N+1);
```

```
                        continue;
                    }
                }

                Dis[point].Shortest_2nd = distance + Dis[last].Shortest;
                Dis[point].Last_2nd = last;

                /*
                Judge about the value of the shortest and the shortest_2nd
                in case that swap is needed.
                */
                if (Dis[point].Shortest > Dis[point].Shortest_2nd)
                {
                    int buffer = Dis[point].Shortest;
                    Dis[point].Shortest = Dis[point].Shortest_2nd;
                    Dis[point].Shortest_2nd = buffer;

                    buffer = Dis[point].Last;
                    Dis[point].Last = Dis[point].Last_2nd;
                    Dis[point].Last_2nd = buffer;
                }
            }
```

2.5 Print

Print the results in right format. Here uses a integer array to store the vertex in the 2nd-shortest path.

```
/*
    The codes below is used to print in the right format.
    TraceBack the path, store the value in an integer array
    and print it out from the opposite order.
    */
    int length = 0;
    int data[M], adj = M, count = 0;
    for (i = 0; ; i++, adj = Dis[adj].Last_2nd)
    {
        data[i] = Dis[adj].Last_2nd;
        int j;
        for (j = 0; j < M; j++)
        {
            if (A[data[i]].Edge[j].AdjV == adj)
                break;
        }
        count += A[data[i]].Edge[j].Length;
        if (data[i] == 1) break;
    }

    /*
    The count represents the distance of the 2nd shortest path.
    Meanwhile, the path is printed with each vertex followed by a "space",
    but no extra space is supposed to occur at the beginning or the end of the
```

```
line.
    So the last element, M, is printed separately from the others.
    */
    printf("%d ", count);
    for (;i >= 0; i--)
        printf("%d ", data[i]);
    printf("%d", M);
```

Chapter 3: Testing Results

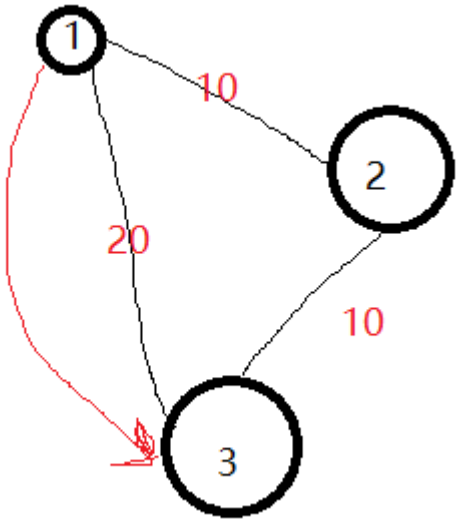
3.1 The basic test from PTA

One shortest path, and the last vertex of the point M are different. Purpose: check if the comprehensive test could work.

Input	result
5 6 1 2 50 2 3 100 2 4 150 3 4 130 3 5 70 4 5 40	240 1 2 4 5

3.2 One shortest path with edges 2, one 2nd-shortest path with edges 1

Input	result
3 3 1 2 10 3 2 5 1 3 20	20 1 3



Pupose: to check if the simple two-path problem can pass.

3.3 One shortest path with edges 1, one 2nd-shortest path with edges 2

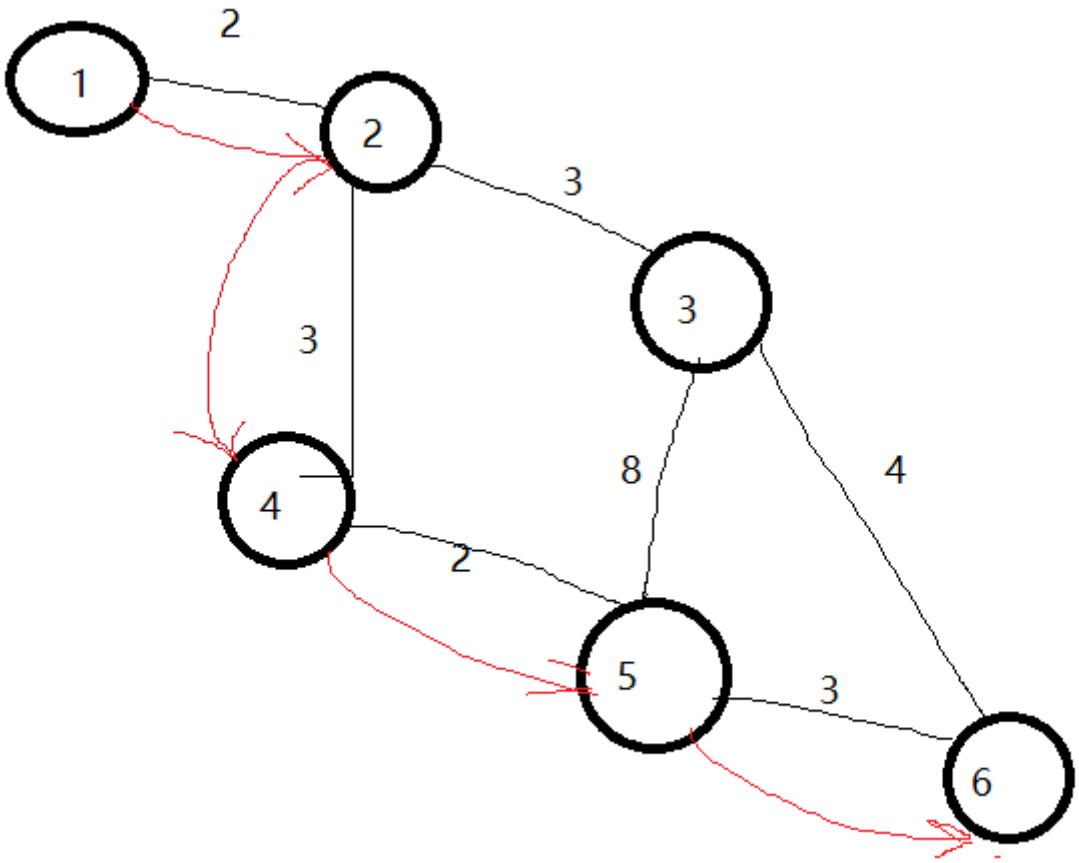
Input	result
3 3 1 2 10 3 2 5 1 3 10	15 1 2 3

Purpose: chage the number of edges in 3.2 and see if it also passes.

3.4 Comprehensive work

One shortest path with edges 3, one 2nd-shortest path with edges 4, several other paths

Input	result
6 7 1 2 2 2 3 3 2 4 3 4 5 2 3 5 8 5 6 3 3 6 4	10 1 2 4 5 6

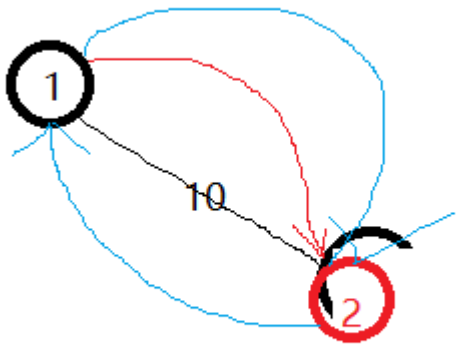


Purpose: to see if the complex condition can still output the right answer.

3.5 Smallest Input

If there are only 2 vertex, we have to go back to the start and back to the finish vertex again.

Input	result
2 1 1 2 10	30 1 2 1 2



Chapter 4: Analysis and Comments

4.1 Analysis

4.1.1 Algorithm 1

Dijkstra Algorithm based on greedy sort is a common way to implement this function of finding the 2nd-shortest path. To do this, we need to search the each vertex, and in each vertex, we have to traverse vertex times, and the number of the edges is E, so the time complexity is O(V^2+E), and the space complexity is O(1) as no extra space is needed in this algorithm.

4.1.2 Algoithm 2

Dijkstra Algorithm based on heap sort is a better way to save time, as time complexity of each heap sort is O(logN), so in this case, the time complexity of the algorithm is O(VlogV + E), and again, the space complexity is O(1) as no extra space is needed in this algorithm.

4.2 Comments

More work should be done to reduce time and reduce the waste of space. For time reducing, I think the algorithm of the dijkstra based on greedy sort needs to be improved because it does a lot of repeated work. Besides, it is better if using the abstract queue data structure instead of integer array, which can reduce the cost in space.

Appendix: Source Code (in C)

```
/*
Project 2. The 2nd-shortest path

Key point: the algorithm is based on Dijkstra Agorithm.
We need to find the shortest and the 2nd-shortest paths at the same time.
If there are M points in tatal, then we have to find the 2nd-shortest path
from 1 to M.
*/

#include <stdio.h>
#include <stdlib.h>

/*
The max num of the vertexs.
*/
#define MaxVertexNum     1001

/*
The MaxRailwayNum of the total graph.
*/
#define MaxRailwayNum    5001

/*
Use negative number -1 to represent the length of a path is infinity.
*/
#define Infinity         -1
```

```
/*
define Vertex as a type to represent the serial number of vertex.
*/
typedef int Vertex;

/*
The struct PtrToAdjVNode is used to store the next point Adjv,
and the length as well.
*/
typedef struct AdjVNode PtrToAdjVNode;
struct AdjVNode{
    Vertex AdjV;
    int Length;
};

/*
The struct Graph, is used to record the infomation of each point,
which consists of the edges of each node,
and the total number of edges of each node.
*/
typedef struct VNode Graph;
struct VNode{
    PtrToAdjVNode Edge[MaxVertexNum];
    int RailwayNum;
};

/*
The struct List is used to record the shortest and the 2nd-shortest path.
Meanwhile, it also has the function of recording the last points of
both the shortest and the 2nd-shortest path, respectively.
*/
typedef struct ShortestDistance List;
struct ShortestDistance{
    int Shortest;
    int Last;
    int Shortest_2nd;
    int Last_2nd;
    int path[MaxVertexNum];
    int path_2nd[MaxVertexNum+2];
};

int main ()
{
    /*
    M: The number of Vertex;
    N: The number of total railways;
    i: used in for-loop.
    */
    int M, N, i;
    scanf("%d %d", &M, &N);

    /*
    The integer array Queue[N+1] is a circular queue used to store the vertex
```

```
    when serching the graph using greedy algorithm.
    It has the basic functions: enqueue and dequeue.
    Moreover, the integers front and rear is used to point at
    the location of the latest point in the queue and the oldest point remains in
the queue.
    */
    int Queue[N+1];
    int front, rear;
    front = rear = 0;

    /*
    Dis[M+1] here is used to record the infomation of the shortest path
    of each vertex. First, initialize the Dis[1].shortest and the
Dis[1].shortest_2nd as 0,
    Dis[1].Last and Dis[1].Last_2nd are initialized as 1.
    */
    List Dis[M+1];
    Dis[1].Shortest = 0;
    Dis[1].Shortest_2nd = 0;
    Dis[1].Last = 1;
    Dis[1].Last_2nd = 1;

    /*
    Initialize the RailwayNum of railways of each vertex as 0.
    */
    Graph A[M+1];
    for (i = 0; i < M+1; i++)
        A[i].RailwayNum = 0;

    /*
    Initialize both the lengths of the shortest and 2nd-shortest paths
    from 1 to each vertex as infinity, -1.
    */
    for (i = 2; i < M+1; i++)
    {
        Dis[i].Shortest = Infinity;
        Dis[i].Shortest_2nd = Infinity;
    }

    /*
    In this N-for-loop, the inputs are the vertex a, b, and the distance between
them.
    Then store these information in the Graph A[M+1].
    */
    for (i = 0; i < N; i++)
    {
        /*
        indexa: used to store the value of the RailwayNum of the vertex a;
        indexb: used to store the value of the RailwayNum of the vertex b;
        */
        int a, b, dis, indexa, indexb;
        scanf("%d %d %d", &a, &b, &dis);

        indexa = A[a].RailwayNum;
```

```
        indexb = A[b].RailwayNum;

        A[a].Edge[indexa].AdjV = b;
        A[a].Edge[indexa].Length = dis;

        A[b].Edge[indexb].AdjV = a;
        A[b].Edge[indexb].Length = dis;

        A[a].RailwayNum++;
        A[b].RailwayNum++;
    }

    Queue[front] = 1;
    front++;
    front%=(N+1);

    /*
    temp: store the former value of the shortest path of the point.
    temp_2nd: store the former value of the 2nd-shortest path of the point.
    point: The point which we are going to deal with.
    distance: the distance from the former point to the point we are going to deal
with.
    last: The fomer point.
    */
    int temp, temp_2nd, point, distance, last;
    int board[M+1];
    for (i = 0; i < M+1; i++)
        board[i] = 0;
    /*
    front != rear means the queue is not empty,
    so the circulation is not going to stop.
    */
    while(front != rear)
    {
        last = Queue[rear];

        /*
        if the RailwayNum is happens to be 0,
        then we don't have to do anything,
        what we only need to do is to skip it.
        */
        if (A[last].RailwayNum != 0)
        {
            /*
            Do the circulation to traverse the next stations of the point "last".
            The total number of the next station is A[last].RailwayNum.
            */
            for (i = 0; i < A[last].RailwayNum; i++)
            {
                /*
                point and distance are used to store the adjv and length of the
last vertex.
                */
                point = A[last].Edge[i].AdjV;
```

```
                distance = A[last].Edge[i].Length;

                temp = Dis[point].Shortest;
                temp_2nd = Dis[point].Shortest_2nd;
                if (board[point] == 1)
                {
                    continue;
                }
                /*
                If temp is Infinity, means it is the fist time to pass through the
point,
                so, any result is considered as the shortest path.
                */
                if (temp == Infinity)
                {
                    Dis[point].Shortest = distance + Dis[last].Shortest;
                    Dis[point].Last = last;
                }

                /*
                If temp_2nd is Infinity, means the 2nd-shortest path has not been
found.
                so, any result is considered as the 2nd-shortest path.
                But what deserves your attention is that further conparison of
                the 2nd-shortest path and the shortest path is need,
                maybe we have to swap the values of the two.
                */
                else if (temp != Infinity && temp_2nd == Infinity)
                {
                    if (point == M)
                    {
                        if (temp == distance + Dis[last].Shortest)
                        {
                            Queue[front] = point;
                            front++;
                            front%=(N+1);
                            continue;
                        }
                    }

                    Dis[point].Shortest_2nd = distance + Dis[last].Shortest;
                    Dis[point].Last_2nd = last;

                    /*
                    shortest > shortest_2nd,
                    swap is needed.
                    */
                    if (Dis[point].Shortest > Dis[point].Shortest_2nd)
                    {
                        int buffer = Dis[point].Shortest;
                        Dis[point].Shortest = Dis[point].Shortest_2nd;
                        Dis[point].Shortest_2nd = buffer;

                        buffer = Dis[point].Last;
```

```
                Dis[point].Last = Dis[point].Last_2nd;
                Dis[point].Last_2nd = buffer;
            }

            /*
            If shortest <= shortest_2nd,
            then traceback is needed to change the value of the last_2nd
            for every vertex in the path.
            */
            else
            {
                int obj = last;
                while (1)
                {
                    Dis[obj].Last_2nd = Dis[obj].Last;
                    obj = Dis[obj].Last;
                    if (obj == 1) break;
                }
            }
        }

        /*
        If temp_2nd > the distance in the present,
        swap is needed.
        */
        else if (temp_2nd > distance + Dis[last].Shortest)
        {
            if (point == M)
            {
                if (temp == distance + Dis[last].Shortest)
                {
                    Queue[front] = point;
                    front++;
                    front%=(N+1);
                    continue;
                }
            }

            Dis[point].Shortest_2nd = distance + Dis[last].Shortest;
            Dis[point].Last_2nd = last;

            /*
            Judge about the value of the shortest and the shortest_2nd
            in case that swap is needed.
            */
            if (Dis[point].Shortest > Dis[point].Shortest_2nd)
            {
                int buffer = Dis[point].Shortest;
                Dis[point].Shortest = Dis[point].Shortest_2nd;
                Dis[point].Shortest_2nd = buffer;

                buffer = Dis[point].Last;
                Dis[point].Last = Dis[point].Last_2nd;
                Dis[point].Last_2nd = buffer;
```

```
                    }
                }

                Queue[front] = point;
                front++;
                front%=(N+1);
            }
        }
        rear++;
        rear%=(N+1);
        board[last] = 1;
    }

    /*
    The codes below is used to print in the right format.
    TraceBack the path, store the value in an integer array
    and print it out from the opposite order.
    */
    int length = 0;
    int data[M], adj = M, count = 0;
    for (i = 0; ; i++, adj = Dis[adj].Last_2nd)
    {
        data[i] = Dis[adj].Last_2nd;
        int j;
        for (j = 0; j < M; j++)
        {
            if (A[data[i]].Edge[j].AdjV == adj)
                break;
        }
        count += A[data[i]].Edge[j].Length;
        if (data[i] == 1) break;
    }

    /*
    The count represents the distance of the 2nd shortest path.
    Meanwhile, the path is printed with each vertex followed by a "space",
    but no extra space is supposed to occur at the beginning or the end of the
line.
    So the last element, M, is printed separately from the others.
    */
    printf("%d ", count);
    for (;i >= 0; i--)
        printf("%d ", data[i]);
    printf("%d", M);
    return 0;
}

/*
Test Examples
1:
5 6
1 2 50
2 3 100
2 4 150
```

```
3 4 130
3 5 70
4 5 40

2:
3 3
1 2 10
3 2 5
1 3 20

3:
3 3
1 2 10
3 2 5
1 3 10

4:
6 7
1 2 2
2 3 3
2 4 3
4 5 2
3 5 8
5 6 3
3 6 4

*/
```

Declaration

I hereby declare that all the work done in this project titled "Project 2.The 2nd-shortest Path" is of my independent effort.