Dijkstra Sequence

Name: Wu Xinbei

Date: 2020-12-12

Chapter 1: Introduction

Problem Description

1. Input Specification

The First Line: Number of Vertices(Nv <= 1000) and Edges(Ne <= 100000)

Following Ne Lines: Vertex i   Vertex j   weight (<= 100)

The (1+Ne) Line: Number of Sequences to be checked(K <= 100)

Following K Lines: Each line contain a specific path

Different to index of arrays, the vertices are numbered from 1 to Nv

Note: According to given Input Specification, there's no need for us to detect whether size of input overflows.

2. Goal

Tell **if it is a Dijkstra Sequence** and **output the result** for each given path.

3. Output Specification

Each line print "Yes" for valid Dijkstra Sequence and "No" for invalid Ones.

4. Properties of Dijkstra Sequence

Key: The distance of vertex i in the path is equal to the shortest path for the same start vertex.

*Note: Dijkstra sequence could be generated by Dijkstra Algorithm, but there could be **more than one** Dijkstra sequence.*

Problem Analysis

We have to judge whether given path is Dijkstra Sequence for an undirected graph. To make problem easier, we can divide the problem into several part.

1. **Part 1: Graph Building**

   We have to check sequence on the basis of the given undirected weighted graph. Thus it's essential to initialize and build graph by input first.

2. **Part 2: Check Sequence Line by Line**

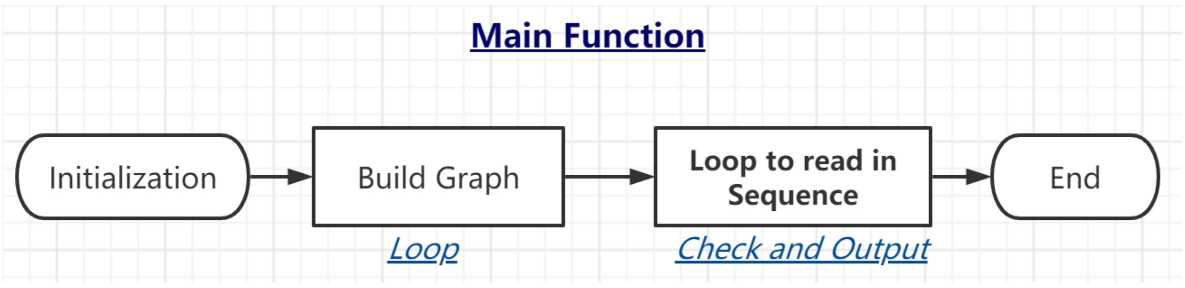
- Get start vertex and initialize
- Loop all vertices
    - Find current shortest path(min) and mark it
    - Compare whether the distance of vertex i in given sequence is equal to min. If not equal, then it’s invalid.
    - Update distance of other vertex according to current shortest path

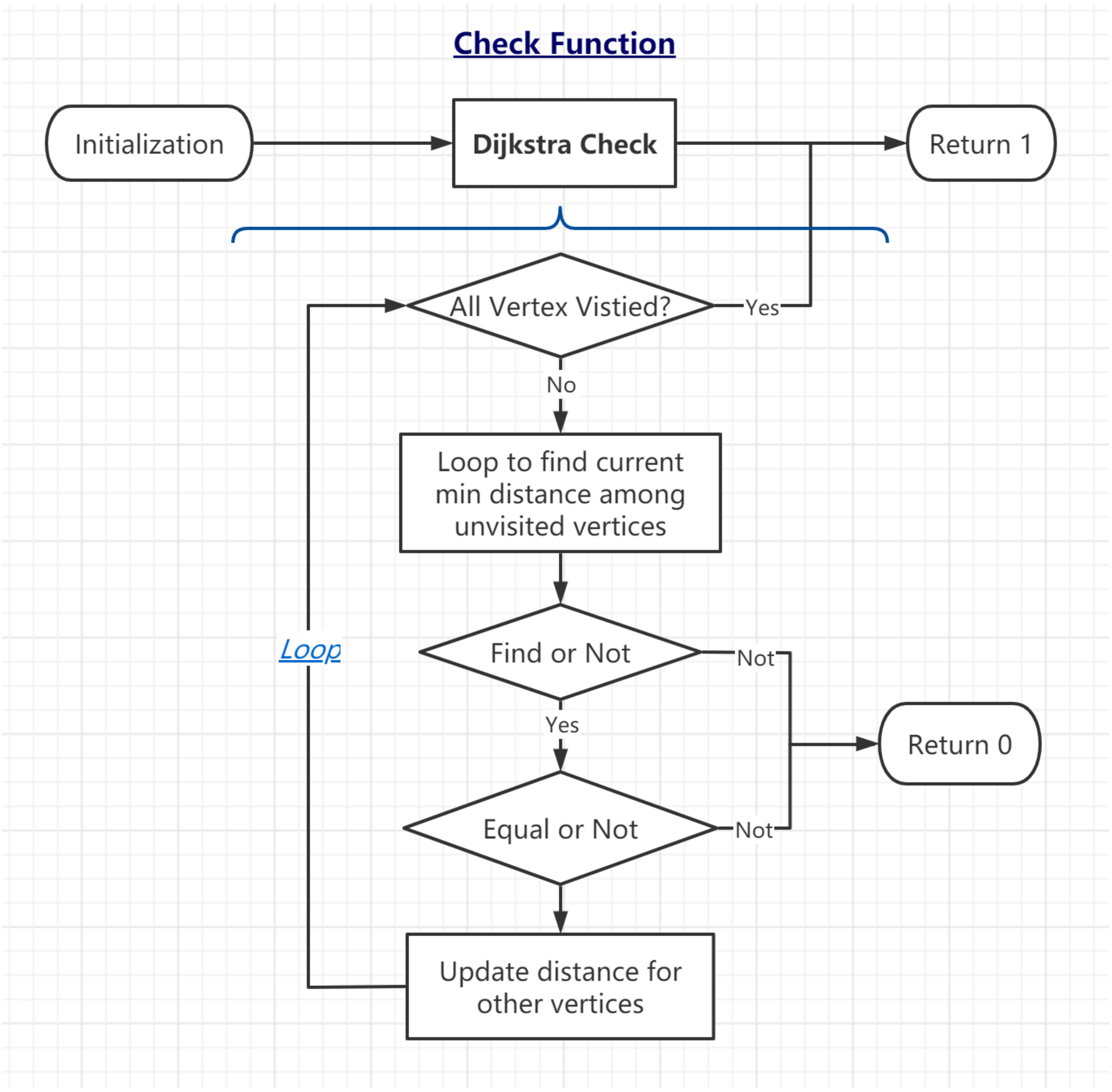
Chapter 2: Algorithm Specification

1. Main Algorithm: Dijkstra Algorithm

Dijkstra Algorithm is applied to solve **Single-Source Shortest-Paths Problem**. Thus, check the given sequence while generating Dijkstra sequence is a superb choice.

2. Overall of Flow Chart





3. Pseudo-code for Core Part Function

1. **Data Structure**

```
#define Vertex int
#define INF 1000000000

int Nv, Ne;              // Number of vertices and edges, record size of
graph
Vertex graph[MAX][MAX]; // store weight of edges; INF stands for unconnected
edges
Vertex Seq[MAX];        // store given path to be checked
```

Note: It's all-right to declare an array to store the results, but to save space, i output the result while detecting.

2. **Pseudo-code for Graph Building**

```
read in size;
loop to initialize graph:
    set weight as INF
loop to read in graph:
    set weight of connected vertices
next part;
```

3. **Pseudo-code for Dijkstra Sequence Check**

```
----------------- outline -----------------
read in number of lines
loop to read in and check:
    loop to read in given sequence
    check sequence;
    a valid Dijkstra sequence or not?
        Yes, output "Yes\n";
        No, output "No\n";
------------------ check ------------------
set start vertex;
initialize distance as INF;
mark vertices as unvisited;
loop to visit vertices in given path:
    set current min distance as INF
    loop to find current shortest path from previous vertex:
        update min distance
    connected graph or not?
        Yes, continue;
        No, invalid Dijkstra sequence and return 0;
    min distance equals distance of vertex i in given path or not?
        Yes, continue;
        No, invalid Dijkstra sequence and return 0;
    mark current vertex;
    loop to update distance:
        if current vertex is connected to another vertex i:
            if their distance is shorter:
                update distance[i] as (distance[cv] + weight[cv][i]);
return 1 for all vertices are checked and sequence is valid.
```

Chapter 3: Testing Results

1. **Given Sample**

   Sample

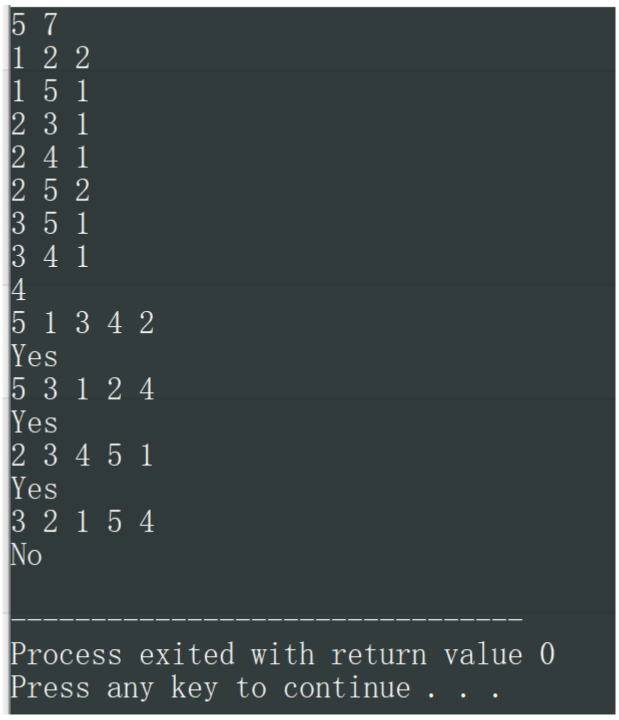
```
5 7         // Nv and Ne, size of graph
1 2 2       // Weight of Edges
1 5 1
2 3 1
2 4 1
2 5 2
3 5 1
3 4 1
4           // Number of Sequences
5 1 3 4 2   // Specific Path
5 3 1 2 4
2 3 4 5 1
3 2 1 5 4
```

   Expected Output

```
Yes
Yes
Yes
No
```

Actual Output

```
Yes
Yes
Yes
No
```



```
5 7
1 2 2
1 5 1
2 3 1
2 4 1
2 5 2
3 5 1
3 4 1
4
5 1 3 4 2
Yes
5 3 1 2 4
Yes
2 3 4 5 1
Yes
3 2 1 5 4
No

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

Result: Pass

2. **Equal Weight**

Sample

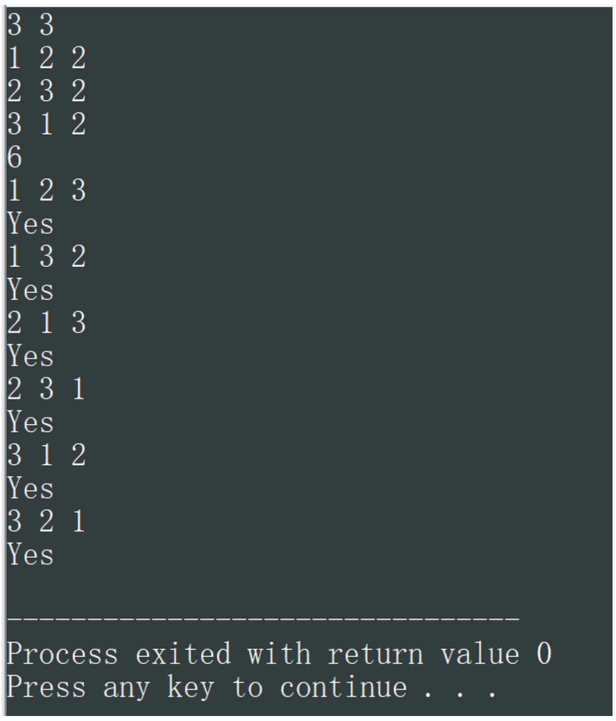
```
3 3     // Nv and Ne, size of graph
1 2 2   // Weight of Edges
2 3 2
3 1 2
6       // Number of Sequences
1 2 3   // Specific Path
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
```

Expected Output

```
Yes
Yes
Yes
Yes
Yes
Yes
```

Actual Result

```
Yes
Yes
Yes
Yes
Yes
Yes
```



```
3 3
1 2 2
2 3 2
3 1 2
6
1 2 3
Yes
1 3 2
Yes
2 1 3
Yes
2 3 1
Yes
3 1 2
Yes
3 2 1
Yes

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

Result: Pass

3. **Only One Connected Path**

   Sample

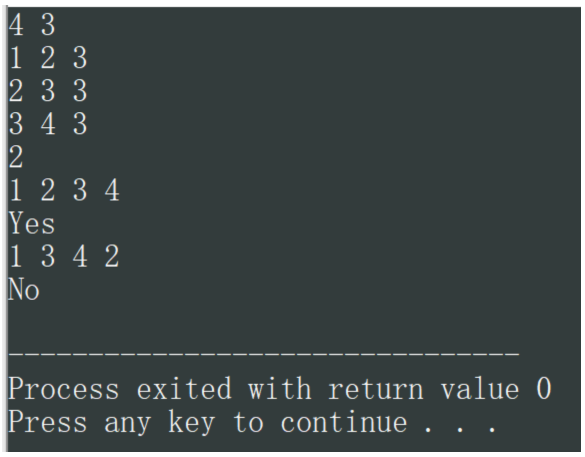
```
4 3     // Nv and Ne, size of graph
1 2 3   // weight of Edges
2 3 3
3 4 3
2       // Number of Sequences
1 2 3 4 // Specific Path
1 3 4 2 // unconnected
```

   Expected Output

```
Yes
No
```

Actual Result

```
Yes
No
```



```
4 3
1 2 3
2 3 3
3 4 3
2
1 2 3 4
Yes
1 3 4 2
No

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

Result: Pass

4. **Unconnected Graph**

   Sample

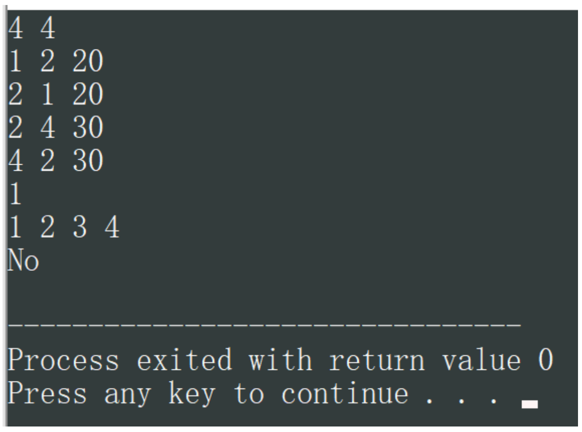
```
4 4     // Nv and Ne, size of graph
1 2 20  // weight of Edges
2 1 20
2 4 30
4 2 30
1       // Number of Sequences
1 2 3 4 // Unconnected
```

Expected Output

```
No
```

Actual Result

```
No
```



```
4 4
1 2 20
2 1 20
2 4 30
4 2 30
1
1 2 3 4
No

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

Result: Pass

5. **Multi-visited**

   Sample

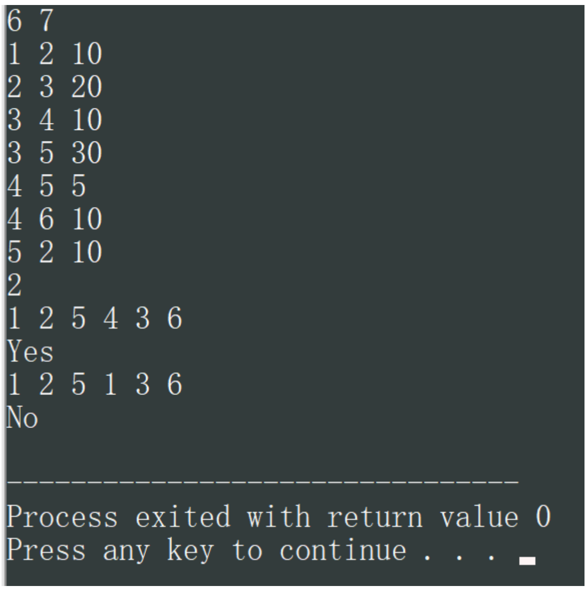
```
6 7          // Nv and Ne, size of graph
1 2 10       // Weight of Edges
2 3 20
3 4 10
3 5 30
4 5 5
4 6 10
5 2 10
2           // Number of Sequences
1 2 5 4 3 6 // Specific Path
1 2 5 1 3 6 // Multi-visted Path
```

Expected Result

```
Yes
No
```

Actual Output

```
Yes
No
```



```
6 7
1 2 10
2 3 20
3 4 10
3 5 30
4 5 5
4 6 10
5 2 10
2
1 2 5 4 3 6
Yes
1 2 5 1 3 6
No

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

Result: Pass

6. **Invalid Input**

   Sample

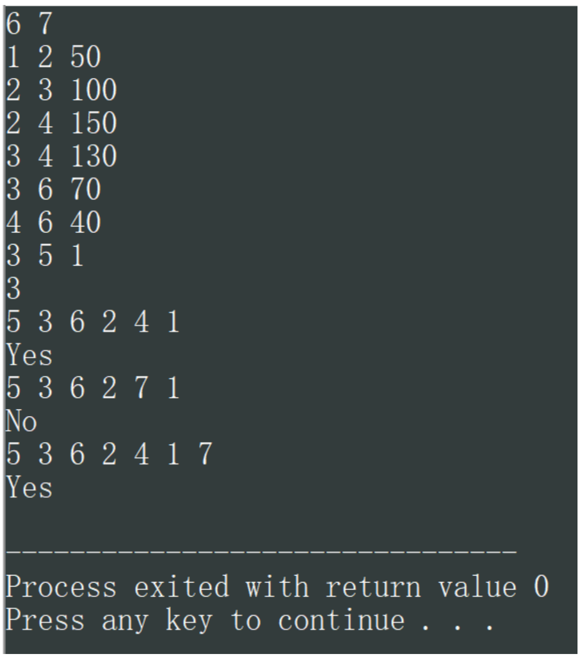
```
6 7                // Nv and Ne, size of graph
1 2 50             // Weight of Edges
2 3 100
2 4 150
3 4 130
3 6 70
4 6 40
3 5 1
3                  // Number of Sequences
5 3 6 2 4 1        // Specific Path
5 3 6 2 7 1        // Invalid Vertex
5 3 6 2 4 1 7      // More than Nv Vertex in one path
```

Expected Output

```
Yes
No
Yes
```

Actual Result

```
Yes
No
Yes
```



```
6 7
1 2 50
2 3 100
2 4 150
3 4 130
3 6 70
4 6 40
3 5 1
3
5 3 6 2 4 1
Yes
5 3 6 2 7 1
No
5 3 6 2 4 1 7
Yes

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

Result: Pass

7. **Big-data**

Sample

```
50 1000
1 2 4831
1 3 1023
1 4 1075
1 5 4002
4 6 3627
6 7 1752
```

```
6 8 1627
1 9 3731
5 10 4295
9 11 2548
9 12 1454
9 13 2262
5 14 2377
2 15 2534
2 16 4597
15 17 2569
16 18 4787
10 19 3859
7 20 2183
10 21 1385
20 22 2966
22 23 1489
6 24 2339
22 25 3824
2 26 1353
10 27 3836
18 28 2673
20 29 2385
9 30 3689
7 31 3293
13 32 1912
20 33 1557
19 34 3120
4 35 3067
9 36 3870
26 37 1683
10 38 1942
18 39 2879
27 40 4323
39 41 4209
22 42 4331
21 43 2506
21 44 4092
35 45 2431
3 46 4561
1 47 3415
24 48 1234
35 49 4207
28 50 3131
31 7 1306
43 26 1563
34 30 4662
45 39 4826
32 8 4322
48 47 4980
26 12 4999
26 4 3055
9 4 3630
6 3 2200
6 3 1843
48 22 1602
5 1 1421
35 9 2345
11 7 3581
```

```
2 1 1160
28 22 1010
7 4 2276
46 28 1316
33 13 2712
34 15 1812
14 1 2015
34 14 1612
22 2 3300
45 39 1481
42 8 3153
41 37 2088
40 15 3737
18 6 3522
18 7 2563
21 13 3098
37 34 2405
16 10 4476
45 3 2436
20 11 3639
7 1 3454
13 11 4479
22 19 1150
12 8 3370
28 18 4729
25 8 2600
12 1 2844
20 3 4591
24 10 4668
15 9 4421
21 13 3503
24 16 1843
11 9 4128
43 39 1625
39 21 2327
36 19 1005
34 14 3533
4 2 1333
15 13 2553
9 5 2676
4 2 3533
23 15 4985
23 11 3411
21 15 1085
42 34 1936
21 16 1321
24 6 4255
22 1 1190
21 20 3660
25 19 2088
34 25 1875
2 1 2161
40 5 2606
28 13 4265
38 14 3213
3 2 4015
46 22 1029
38 16 1291
```

```
49 15 1796
23 3 3444
9 8 4511
3 1 3296
39 21 1947
5 1 2524
31 19 1760
17 15 2790
22 3 3681
6 3 4381
42 16 2541
11 10 4429
43 24 1839
39 20 3980
34 6 2380
27 2 2890
50 34 2709
45 19 4709
12 3 4122
18 15 4026
11 4 3619
16 1 2477
36 12 4284
3 1 4366
18 15 2785
26 24 4857
41 28 4512
40 36 3054
14 6 4391
23 16 1738
28 4 4468
28 2 3583
7 1 1823
13 2 3343
30 2 3928
47 29 2467
20 15 2442
16 10 1440
8 6 4932
32 15 1400
27 15 1961
17 3 1114
41 23 4671
34 15 4696
16 1 4246
27 13 4373
20 5 1652
23 21 4051
26 21 3894
9 3 2142
37 16 2422
44 28 1799
16 8 2171
22 15 4233
6 5 2082
33 13 4348
5 3 4965
50 25 4740
```

```
18 12 3851
49 28 4854
38 10 4048
48 38 1545
15 14 2825
29 1 2955
38 5 4479
38 23 2125
22 10 3511
30 7 3663
49 47 1942
21 5 1225
17 5 4357
6 5 1551
31 17 2328
4 1 4488
13 9 4648
15 11 2033
45 30 4385
5 4 4432
17 6 3127
13 4 1451
36 35 4471
24 15 4805
12 3 4291
25 17 1156
20 7 2580
34 23 3314
24 15 4262
5 1 3715
40 25 2745
21 8 1511
10 7 3204
27 23 3086
41 30 2091
8 2 3904
7 1 4998
44 18 1749
39 30 4178
25 21 1628
37 35 2185
43 39 2985
29 23 1310
16 1 2144
35 2 4798
30 19 1737
32 7 2806
14 5 2414
29 21 4111
49 33 3059
32 22 3045
3 1 4254
40 13 3271
37 31 2377
23 21 2968
38 25 1400
48 10 1437
27 10 4452
```

```
6 3 1337
36 1 3265
17 1 2962
18 7 4977
27 9 4499
18 13 3704
28 17 2594
18 1 3842
40 26 4605
24 9 4287
39 12 2446
7 1 3173
27 17 4457
24 5 2441
48 28 2972
7 6 4884
23 9 3133
23 7 2255
31 25 3306
15 9 4519
49 2 4555
17 7 1200
25 12 1211
40 30 4122
30 3 3446
22 10 1794
4 3 4823
44 3 1125
47 24 1530
10 5 3603
46 10 1509
36 34 1306
42 29 4216
37 32 2481
33 28 3186
20 17 4086
2 1 3341
22 3 4301
4 3 2044
24 13 4667
16 12 4171
31 6 1271
38 5 1091
37 29 1763
29 17 1302
13 3 4866
15 9 2079
41 39 4371
38 19 2991
20 10 4350
16 9 3425
43 30 3622
3 1 4352
12 8 4916
5 3 2480
37 36 4193
44 29 1481
35 17 2939
```

```
48 23 2105
17 10 1719
34 19 3293
40 4 2228
38 18 2039
27 12 3835
5 4 4413
29 8 3226
9 2 2864
8 3 2464
30 25 1663
3 1 4249
44 12 1287
44 27 1126
24 21 3960
11 7 1337
5 2 3655
8 4 1750
37 2 4967
16 7 4883
13 6 4992
6 4 1098
32 16 3505
11 5 4759
12 2 3103
33 14 3495
12 1 1277
32 25 3326
19 5 2870
42 39 3171
31 14 2706
26 2 4299
4 3 3455
44 12 1748
21 3 1696
18 17 1624
36 12 2003
41 34 1336
33 17 1907
11 5 4118
21 4 2293
43 4 2271
34 23 3041
34 5 4211
20 14 4186
29 19 3469
9 4 3304
5 2 3096
46 13 2140
46 36 3172
11 4 1677
19 10 4009
9 1 4749
45 5 3770
33 19 3423
13 1 1700
11 6 2942
45 18 2537
```

```
24 15 3338
9 5 1812
33 14 4307
41 28 4891
17 7 1060
18 15 2964
18 6 1179
45 34 3733
14 4 4325
11 8 1532
14 5 4373
45 17 3845
29 27 2311
34 32 4573
22 8 4076
26 19 1667
23 13 1612
18 9 2607
43 37 1962
24 2 4125
29 22 1601
26 11 4259
45 39 2276
24 17 2485
25 11 2936
14 7 2579
21 9 1589
34 23 4425
27 14 3701
33 4 2124
28 11 4760
40 21 2410
7 5 3792
39 8 4284
27 4 2669
20 1 4836
22 15 1812
24 19 4790
40 12 4992
2 1 1136
24 2 1579
37 10 3582
36 16 4894
2 1 1014
35 21 4998
20 12 1130
2 1 1759
2 1 4250
46 19 4886
11 3 2179
18 16 4335
39 38 1888
17 7 3527
37 3 2573
35 3 1794
13 9 2545
32 12 3865
44 13 3549
```

```
7 6 2918
42 24 4440
41 15 1295
42 11 2043
11 2 1223
48 23 4555
30 13 4662
23 17 3642
2 1 3306
12 11 2651
49 25 4087
37 35 3124
41 36 2656
10 2 2366
44 18 1011
44 30 3419
47 42 2218
17 9 1373
38 16 3867
35 6 3540
11 7 4281
34 1 3188
46 35 4826
33 28 3973
18 13 2430
33 5 2561
11 1 1020
47 23 3302
43 9 2328
48 44 3902
6 3 3936
5 3 3892
47 27 4558
7 4 2764
5 1 1898
10 4 1400
48 28 4870
11 9 4074
50 29 1797
4 3 3412
43 34 1361
40 27 3002
37 18 3233
36 31 2798
36 23 2829
9 3 1525
48 5 3644
34 12 2225
8 6 4761
24 18 2926
36 28 1938
5 1 3383
4 1 3491
17 2 4224
4 3 3022
39 15 3049
3 2 2095
41 14 3116
```

```
6 3 3495
44 18 3905
42 18 3049
25 10 3236
3 2 2460
18 7 1991
6 1 3163
2 1 2216
37 35 4001
21 17 1070
47 31 1998
38 11 4455
46 39 2149
18 12 2160
43 7 1480
17 8 4060
46 14 4485
39 7 3685
21 12 2074
6 4 1388
19 15 4616
41 33 2395
39 13 3656
30 2 4348
2 1 3257
19 5 2026
27 15 1161
49 20 1837
40 8 3276
31 8 2852
43 35 3126
27 15 1004
10 9 2410
7 3 1549
46 9 2069
45 21 3211
36 1 1855
17 3 1249
48 34 4283
6 3 3054
49 38 3634
50 35 4804
9 3 3131
40 6 3072
25 18 3262
6 4 2641
9 7 4213
31 23 2204
10 1 4217
6 1 2171
27 16 3160
40 21 1243
23 7 2359
9 2 3128
12 5 1493
40 34 3739
7 3 2235
17 12 3751
```

```
25 13 2483
8 5 3211
47 37 1899
24 23 2893
40 9 1386
12 7 3212
31 15 2987
26 14 2813
32 21 4545
45 10 4632
40 18 1849
45 30 1520
21 9 3389
43 18 1442
3 2 1688
32 20 1350
23 1 1670
6 4 2102
33 14 1529
16 14 3806
48 40 1726
17 8 1046
19 17 3024
31 18 1745
28 21 4729
35 30 2585
5 4 3592
25 10 1285
50 24 4128
49 22 3741
44 26 4612
19 8 1927
36 25 1805
18 16 1353
21 17 4900
46 3 4536
14 3 1569
50 17 2231
48 37 3309
34 19 2625
2 1 1742
25 5 3316
25 12 2618
42 25 1282
18 6 1040
11 7 1991
27 22 2576
11 4 1339
28 26 1737
7 1 2920
22 9 1214
21 13 3228
21 8 4458
14 12 2662
47 8 1183
15 7 3464
46 35 4625
7 6 2141
```

```
3 1 4476
32 25 2559
2 1 3942
41 24 1270
26 23 1121
33 7 4388
13 6 4864
17 14 4245
41 9 4272
34 10 3578
29 3 2291
41 16 1049
30 4 1462
10 4 3211
6 2 2109
44 20 4902
29 17 1521
34 20 2500
4 1 1728
49 38 4961
19 4 3797
29 19 4398
5 4 3519
25 11 4019
27 26 3030
34 20 2544
36 34 4862
19 9 2438
10 2 1561
16 2 3105
21 13 1778
45 18 3103
6 4 4115
2 1 1156
15 9 4197
6 3 3834
35 13 4734
7 4 4630
24 23 4696
12 5 4320
46 23 2533
11 8 2633
44 3 4308
48 30 4349
3 1 3080
11 10 4685
15 13 2067
16 2 2838
34 27 1680
6 4 1218
28 20 2077
30 11 1875
48 12 1989
3 1 2023
42 41 3587
5 1 4409
9 7 1671
21 18 2335
```

```
44 31 3426
14 12 3409
33 3 2036
21 12 4831
37 7 1900
29 8 2262
11 4 2190
12 3 1232
11 5 2747
42 4 3034
30 5 4808
20 5 3672
21 16 4414
22 13 1017
2 1 2461
18 14 1402
6 2 3804
9 8 2607
14 6 3496
20 17 2390
27 4 3152
8 5 4297
36 19 1767
7 4 2091
4 3 3093
47 35 3327
22 18 1595
10 6 2960
47 15 3085
11 10 2520
17 5 1115
6 1 2287
31 17 1083
9 3 1027
34 19 2627
20 16 2657
22 13 3060
42 33 2520
37 1 2668
12 11 2785
31 7 3336
28 24 1612
20 5 3241
33 1 3074
24 1 4156
23 9 4590
33 18 2447
18 10 2338
5 2 1056
20 10 3673
39 24 3195
7 6 3257
19 6 4791
40 19 2568
4 3 2757
19 8 3500
41 38 4246
45 14 1194
```

```
42 11 4339
18 3 3345
38 5 2937
26 19 3760
36 2 4954
48 10 4612
31 16 3216
8 7 3063
17 10 4091
38 18 3305
32 21 2843
2 1 3422
24 15 1134
48 40 2772
5 4 1321
19 12 1889
6 3 3152
43 6 4557
34 32 1053
4 1 1712
20 5 3125
31 16 3318
43 6 4535
48 12 3846
39 33 3589
27 19 3880
25 12 4170
42 10 3002
29 20 1883
44 43 4305
2 1 2350
42 14 3254
8 7 3179
23 15 1045
4 3 4929
35 22 4159
17 15 1388
11 7 3749
13 11 3846
25 12 4873
50 48 1200
14 8 3385
21 13 4115
17 6 1758
48 25 4757
30 2 1894
35 20 3260
8 1 1038
2 1 4935
38 29 4372
9 1 1482
40 11 2540
32 11 2054
20 14 3980
13 12 3606
26 6 4066
10 2 3722
39 16 3666
```

```
38 11 2444
49 31 4135
19 1 4794
26 23 3016
11 8 4172
19 6 3752
14 4 4413
14 3 1284
15 2 1119
35 12 2661
11 9 3578
21 4 3606
3 1 1317
32 22 4723
31 22 1686
47 16 4242
25 10 4879
8 2 4528
5 3 4064
35 32 1494
37 35 4703
48 31 3748
36 22 3036
9 6 3950
10 4 4623
25 14 4052
9 6 3290
47 9 4538
35 20 3605
25 7 4409
19 11 1838
12 3 2121
50 16 1599
46 24 2836
38 21 3345
21 8 1342
30 28 1853
7 1 2465
26 10 3431
18 15 4168
22 19 1166
6 2 2081
25 17 3563
6 1 3485
20 4 2065
9 8 2129
50 30 3233
29 10 4245
8 1 3809
16 8 1387
36 23 4143
25 15 1342
17 8 2450
25 5 3481
28 26 3834
34 9 1365
35 7 1717
37 18 4597
```

```
13 6 2858
40 19 4307
26 4 3306
9 5 4405
18 1 4086
38 10 2655
27 5 4740
36 27 3914
46 12 2513
17 10 3393
10 9 2801
43 27 3013
25 18 1861
38 21 2249
46 6 1991
13 7 1875
20 16 1736
18 14 4542
23 11 1203
4 1 1310
16 1 2694
49 27 3618
12 2 2011
10 4 3206
40 26 2564
17 12 4063
17 6 3634
13 1 3102
7 3 2094
38 28 1604
4 2 1831
31 7 2239
40 13 2830
19 16 1005
33 29 2090
38 8 1538
38 26 4037
10 7 2469
48 25 2111
46 40 2096
14 3 2186
19 8 1310
31 18 4052
33 12 1304
26 2 1970
3 1 2162
11 7 3972
2 1 3225
39 32 1721
21 12 1695
8 4 1327
6 5 3157
12 4 1785
34 13 2653
6 2 1199
21 16 3640
23 15 4446
37 35 1203
```

```
6 5 2461
49 5 1159
40 11 4132
18 5 1681
4 3 4527
44 12 3841
13 3 1630
36 27 4287
3 2 2348
13 7 3435
44 14 3905
12 11 4652
35 33 3025
41 32 4333
42 40 3693
50 48 1537
9 4 4311
16 7 4509
38 25 4203
19 5 4587
12 3 2561
6 4 4627
12 2 2663
50 3 2342
3 2 2989
21 10 2098
15 12 2030
45 29 4613
42 6 3408
8 5 4570
8 1 1501
39 7 4697
28 2 3922
45 31 4008
32 9 2568
4 2 3224
41 32 1879
37 10 3339
20 18 3027
27 21 4668
49 18 4422
15 13 3291
27 23 4300
46 9 3237
26 16 4381
20 14 2952
19 18 4233
32 12 4409
22 6 4710
3 2 1815
8 7 1453
6 3 4094
15 9 4087
36 16 3201
13 7 3705
15 3 2583
23 22 4113
50 35 4752
```

```
30 26 2970
37 8 4796
34 33 2320
32 29 2177
42 6 2434
26 13 1326
38 17 3887
9 8 2740
29 23 3526
32 16 3825
46 24 1770
7 6 1932
49 2 3781
46 7 3707
8 1 3279
48 36 3805
42 1 4589
22 20 3586
42 34 1701
14 11 3594
12 1 3535
49 21 3827
5 2 1040
23 17 3465
20 3 3379
31 6 2900
28 27 4544
17 3 3552
14 3 2072
25 9 2776
4 2 3107
48 44 3114
48 1 4752
50 11 4099
9 3 1920
35 27 4198
11 4 2796
44 22 3791
22 19 3954
16 2 2429
41 13 4683
26 6 1558
13 1 1617
4 2 4978
43 24 2059
31 24 1616
28 20 4572
18 4 4925
13 10 2342
28 14 4244
45 27 3947
7 6 2455
26 15 4846
12 10 1723
7 3 3075
47 27 4630
46 16 2715
23 22 2678
```

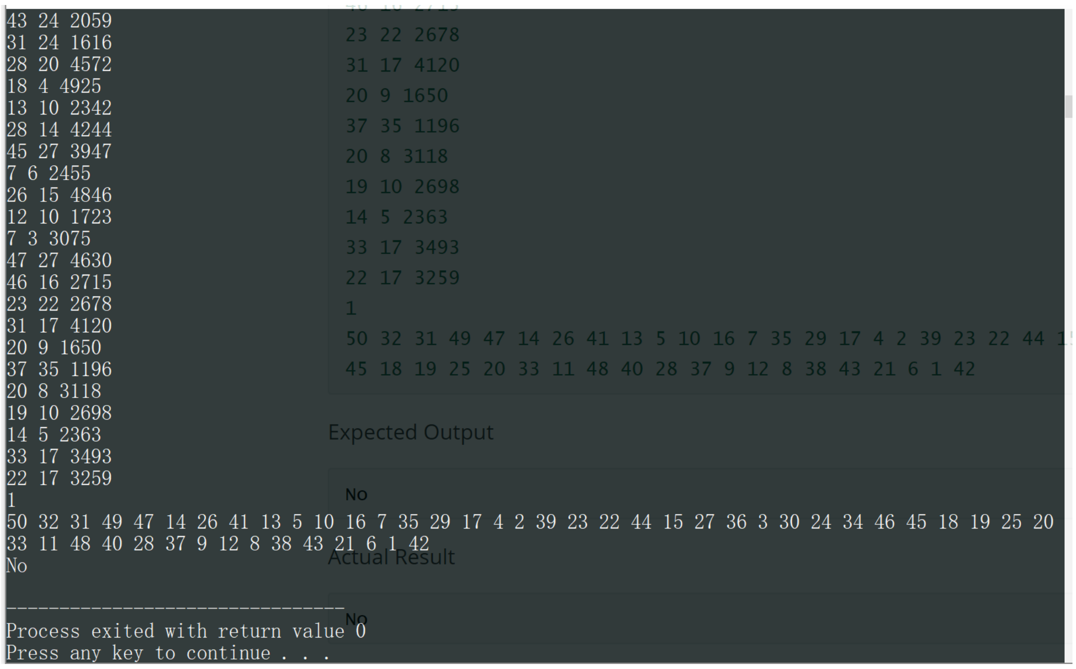
```
31 17 4120
20 9 1650
37 35 1196
20 8 3118
19 10 2698
14 5 2363
33 17 3493
22 17 3259
1
50 32 31 49 47 14 26 41 13 5 10 16 7 35 29 17 4 2 39 23 22 44 15 27 36 3 30
24 34 46 45 18 19 25 20 33 11 48 40 28 37 9 12 8 38 43 21 6 1 42
```

Expected Output

```
No
```

Actual Result

```
No
```



```
43 24 2059
31 24 1616
28 20 4572
18 4 4925
13 10 2342
28 14 4244
45 27 3947
7 6 2455
26 15 4846
12 10 1723
7 3 3075
47 27 4630
46 16 2715
23 22 2678
31 17 4120
20 9 1650
37 35 1196
20 8 3118
19 10 2698
14 5 2363
33 17 3493
22 17 3259
1
50 32 31 49 47 14 26 41 13 5 10 16 7 35 29 17 4 2 39 23 22 44 15 27 36 3 30 24 34 46 45 18 19 25 20
33 11 48 40 28 37 9 12 8 38 43 21 6 1 42
No

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

Result: Pass

Final Result

The program works successfully for all testing samples.

Note: All testing cases are provided in file "test cases", including big-data.

Chapter 4: Analysis and Comments

Time Complexity

Firstly, assume the number of vertices as **Nv**, the number of edges as **Ne**, the number of given sequences as **K**.

Secondly, compute each part’s time complexity.

1. Main Function
    - Initialization: It takes a two-layer loop to initialize graph, set its’ weight of edges as INF. T(Nv) = O(Nv^2);
    - Graph Building: It takes a one-layer loop to store weight of edges in. For there’ re Ne edges, it takes O(Ne);
    - Read in and Check for Each Sequence
        - Reading in Sequence takes O(Nv);
        - Assume the time complexity of checking a sequence is Tc;

      Then, this part T(K, Nv) = O(K * (O(Nv) + Tc)) = O(K* (Nv + Tc);

   Tm(K, Nv) = O(Nv^2) + O(Ne) + O(K* (Nv + Tc);
2. Check Function
    - Initialization: It takes O(Nv) to initialize arrays to store distance and visit-state
    - Dijkstra Check: visit every vertex in given sequence takes O(Nv)
        - Loop to find current shortest distance takes O(Nv);
        - Judging whether the path is valid takes O(1);
        - Marking visited vertex takes O(1);
        - Loop to update distance takes O(Nv);

      Then, this part T(Nv) = O(Nv*(2 *O(Nv) + 2 *O(1))) = O(Nv^2);

   Tc = O(Nv) + O(Nv^2);

Thus, T(Nv, Ne, K) = O(Nv^2) + O(Ne) + O(K* (Nv + (O(Nv) + O(Nv^2)))) = O(K*Nv^2)

Result: T(Nv, Ne, K) = (K*Nv^2);

Space Complexity

Firstly, assume the number of vertices as **Nv**, the number of edges as **Ne**, the number of given sequences as **K**.

Secondly, compute each part’s space complexity.

1. Main Function
    - Build Graph: an MAX*MAX array is used to store graph and two integer are used to store the size of graph. For MAX = 1111, Sg = O(MAX^2) + O(2) = O(MAX^2);
    - Read in and Check for Each Sequence: it takes an array sized MAX to store given sequence. Sr = O(MAX);
2. Check Function

   It takes two arrays sized MAX to store distance and visit-state. Sc = 2*O(MAX);

S(Nv, Ne, MAX) = Sg + Sr + Sc = O(MAX^2) + O(MAX) + 2*O(MAX) = 5 * O(MAX) = O(MAX^2);

Result: S(Nv, Ne, MAX) = O(MAX^2);

Appendix: Source Code in C

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

#define Vertex int
#define MAX 1111
#define INF 100000000

int Nv, Ne;             // record size of graph
Vertex graph[MAX][MAX]; // store weight of edges
Vertex Seq[MAX];        // store given path

int Check( Vertex start ); // check dijkstra sequence

/*---- Part 1: Main Function ----*/
/* Get input and Output results  */
int main() {
    /*
    ------ Part 1.1: Build up Graph -------
     */
    // initialize and read in size of graph
    scanf("%d %d", &Nv, &Ne);
    int i, j;
    for ( i = 0; i <= Nv; i++ ) {
        for ( j = 0; j <= Nv; j++ ) {
                graph[i][j] = INF;
        }
    }
    // connect vertex and build up graph
    for ( i = 0; i < Ne; i++ ) {
        Vertex v1, v2;
        int weight;
        scanf("%d %d %d", &v1, &v2, &weight);
        // store weight in
        // Note: it's an undirected graph
        graph[v1][v2] = weight;
        graph[v2][v1] = weight;
    }

    /*
    ------ Part 1.2: Dijkstra Sequence Check -------
     */
    // read in number of lines
    int k;
    scanf("%d", &k);
    // check for each queues
    for ( i = 0; i < k; i++ ) {
        int j;
        // read in given sequence
        for ( j = 0; j < Nv; j++ ) {
            scanf("%d", &Seq[j]);
        }
        // check if it's a dijkstra sequence and print result
        if ( Check(Seq[0]) )
            printf("Yes\n");
        else
            printf ("No\n");
    }
    return 0;
}


/*---- Part 2: Check Function ----*/
/* Check whether is Dijkstra Path */
int Check(Vertex start) {
    // initialization
    Vertex s = start;
    int i, j, dist[MAX], flag[MAX];
    for( i = 0; i <= Nv; i++ ) {
        dist[i] = INF;
        flag[i] = 0;    // unvisited
    }
    dist[s] = 0;
    // visit vertex in given sequence
    for ( i = 0; i < Nv; i++ ) {
        int min = INF;      // get current min distance
        int cv = -1;        // get current vertex
        // if there's shorter path, then update min distance
        for ( j = 1; j <= Nv; j++ ) {
            if ( !flag[j] && dist[j] < min ) {
                cv = j;
                min = dist[j];
            }
        }
        // record and compare
            // unconnected
            // current vertex's distance not equal to current min distance
        if ( cv == -1 || dist[cv] != dist[Seq[i]] )
            return 0;
        flag[cv] = 1; // mark as visited
        // update distance
        for ( j = 1; j <= Nv; j++ ) {
            // unvisited and connected
            if ( !flag[j] && graph[cv][j] != INF )
                // shorter than previous distance
                if( dist[cv] + graph[cv][j] < dist[j] )
                    dist[j] = dist[cv] + graph[cv][j];  // update
        }
    }

    return 1;
}
```

Declaration

I hereby declare that all the work done in this project titled "Dijkstra Sequence" is of my independent effort.