ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA HỌC MÁY TÍNH

BÁO CÁO ĐỒ ÁN CUỐI KỲ

PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN

BÀI TOÁN QUY HOẠCH ĐỘNG VÀ ỨNG DỤNG

GVHD: Ths.Phạm Nguyễn Trường An
Nhóm: Dương Lê Tường Khang - 18520882
Bùi Đào Gia Huy - 18520818
Lã Trường Hải - 18520698
Lê Cao Hưng - 17520539

TP. HỒ CHÍ MINH, THÁNG 1/2021

Mục lục

1 Thuật toán Dijsktra

1.1 Giới thiệu

Tham khảo: "**link: https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_to%C3%A1n_Dijkstra**"

1.2 Mô tả thuật toán

Thuật toán Dijkstra là thuật toán tìm đường đi có chi phí nhỏ nhất từ một đỉnh đến tất cả các đỉnh còn lại trong độ thị có trọng số(trọng số không âm).

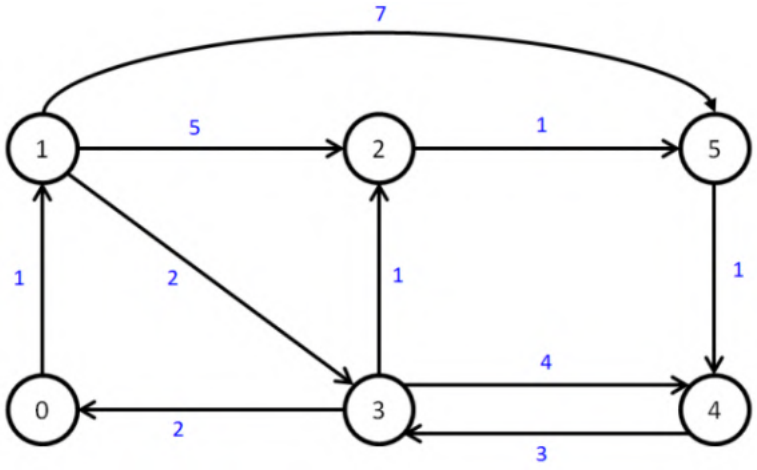
Input:

- Ma trận kề.

Output:

- Hai mảng gồm n phần tử lưu vết đường đi và chi phí ứng với mỗi đỉnh

Ví dụ:



Input: Ma trận kề

```
6
0 1 0 0 0 0
0 0 5 2 0 7
0 0 0 0 0 1
2 0 1 0 4 0
0 0 0 3 0 0
0 0 0 0 1 0
```

Output:

```
Đỉnh 0
dist: [0, 1, 4, 3, 6, 5]
path: [-1, 0, 3, 1, 5, 2]
Đỉnh 1
dist: [4, 0, 3, 2, 5, 4]
path: [3, -1, 3, 1, 5, 2]
Đỉnh 2
dist: [7, 8, 0, 5, 2, 1]
path: [3, 0, -1, 4, 5, 2]
Đỉnh 3
dist: [2, 3, 1, 0, 3, 2]
path: [3, 0, 3, -1, 5, 2]
Đỉnh 4
dist: [5, 6, 4, 3, 0, 5]
path: [3, 0, 3, 4, -1, 2]
Đỉnh 5
dist: [6, 7, 5, 4, 1, 0]
path: [3, 0, 3, 4, 5, -1]
```

Với mỗi đỉnh, xuất ra ma trận chứa đường đi từ đỉnh đó đến các đỉnh còn lại

1.3 Cài đặt

Cài đặt thuật toán Dijsktra cho bài toán tìm đường đi ngắn nhất giữa mọi cặp đỉnh trong đồ thị.

1.3.1 Cài đặt hàm Dijsktra

```
def Dijsktra(s, graph, dist):
    pq = queue.PriorityQueue()
    pq.put(Node(s,0))
    dist[s] = 0
    while pq.empty() == False:
        top = pq.get()
        u = top.id
        w = top.dist
        if dist[u] != w:
            continue
        for neighbor in graph[u]:
            if w + neighbor.dist < dist[neighbor.id]:
                dist[neighbor.id] = w + neighbor.dist
                pq.put(Node(neighbor.id, dist[neighbor.id]))
```

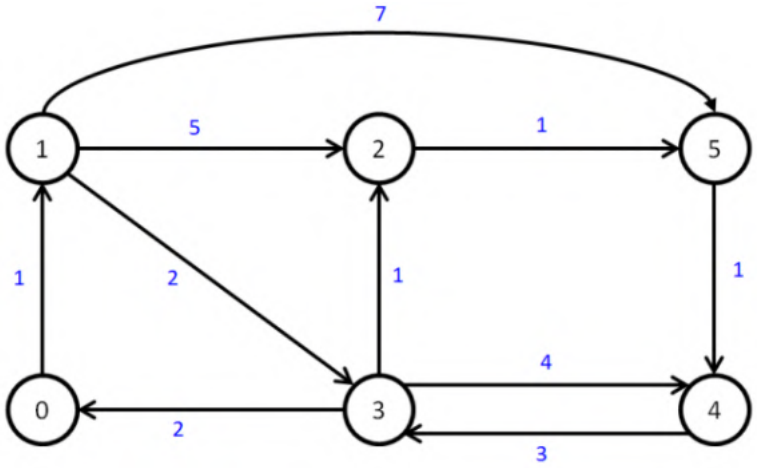
1.3.2 Cài đặt hoàn chỉnh

```
import queue
import sys
sys.stdin = open('D:/src/DA_algo/inp_d.txt', 'r')

MAX = 100
INF = int(1e9)
```

```
7
8 class Node:
9     def __init__(self, id, dist):
10        self.dist = dist
11        self.id = id
12    def __lt__(self, other):
13        return self.dist <= other.dist
14
15 def Dijsktra(s, graph, dist):
16    pq = queue.PriorityQueue()
17    pq.put(Node(s,0))
18    dist[s] = 0
19    while pq.empty() == False:
20        top = pq.get()
21        u = top.id
22        w = top.dist
23        if dist[u] != w:
24            continue
25        for neighbor in graph[u]:
26            if w + neighbor.dist < dist[neighbor.id]:
27                dist[neighbor.id] = w + neighbor.dist
28                pq.put(Node(neighbor.id, dist[neighbor.id]))
29                path[neighbor.id] = u
30
31 if __name__ == '__main__':
32    n = int(input())
33    graph = [[] for i in range(n+5)]
34
35    for i in range(n):
36        d = list(map(int, input().split()))
37        for j in range(n):
38            if d[j] > 0:
39                graph[i].append(Node(j, d[j]))
40    for s in range(n):
41        dist = [INF for i in range(n+5)]
42        path = [-1 for i in range(n+5)]
43        Dijsktra(s, graph, dist)
44        print("Dinh:", s)
45        for d in range(n):
46            if d != s:
47                print(path[:n])
48                break
```

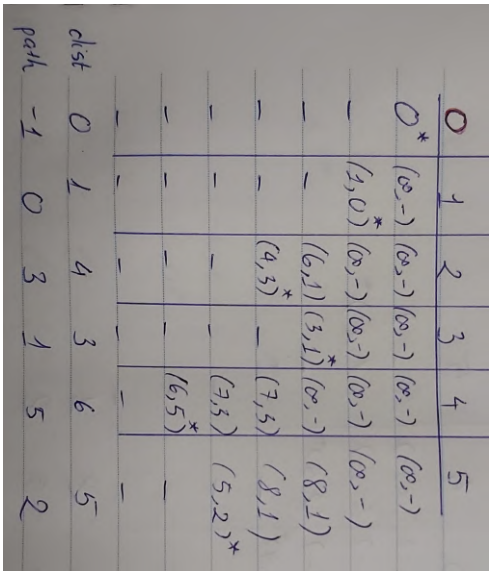
1.4 Kiểm nghiệm tính đúng đắn

Với đồ thị cho như hình dưới:

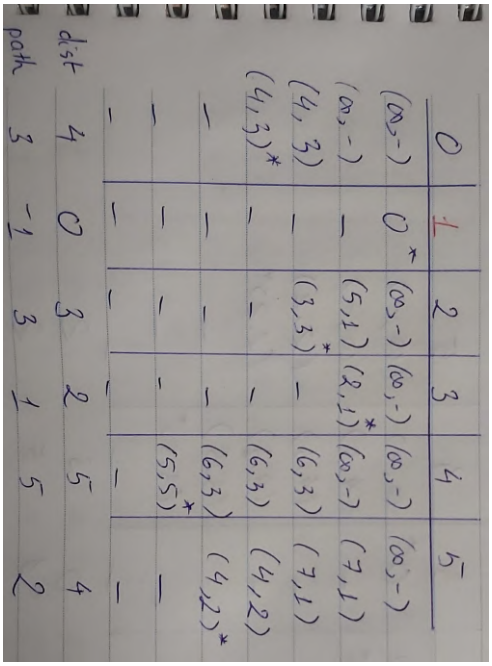


Duyệt lần lượt từng đỉnh để tính khoảng cách ngắn nhất đến các đỉnh còn lại

- *Đỉnh 0:*



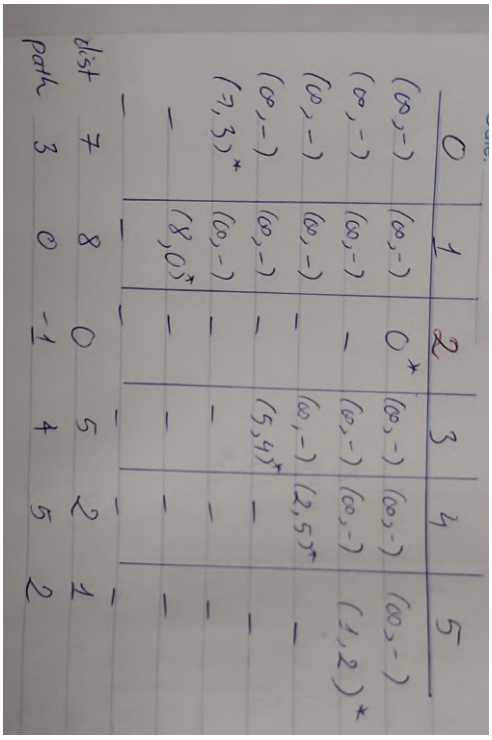
- *Đỉnh 1:*



0	1	2	3	4	5
(∞,-)	0 *	(∞,-)	(∞,-)	(∞,-)	(∞,-)
(∞,-)	-	(5,1)	(2,1) *	(∞,-)	(7,1)
(4,3)	-	(3,3) *	-	(6,3)	(7,1)
(4,3) *	-	-	-	(6,3)	(4,2)
-	-	-	-	(5,5) *	(4,2) *
-	-	-	-	-	-
-	-	-	-	-	-

dist 4 0 3 2 5 4
path 3 -1 3 1 5 2

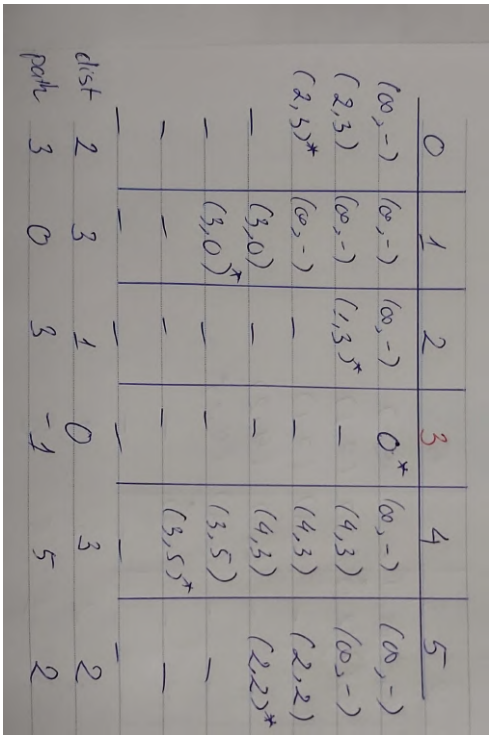
- *Đỉnh 2:*



0	1	2	3	4	5
(∞,-)	(∞,-)	0 *	(∞,-)	(∞,-)	(∞,-)
(∞,-)	(∞,-)	-	(∞,-)	(∞,-)	(1,2) *
(∞,-)	(∞,-)	-	(∞,-)	(2,5) *	-
(∞,-)	(∞,-)	-	(5,4) *	-	-
(7,3) *	(∞,-)	-	-	-	-
-	(8,0) *	-	-	-	-
-	-	-	-	-	-

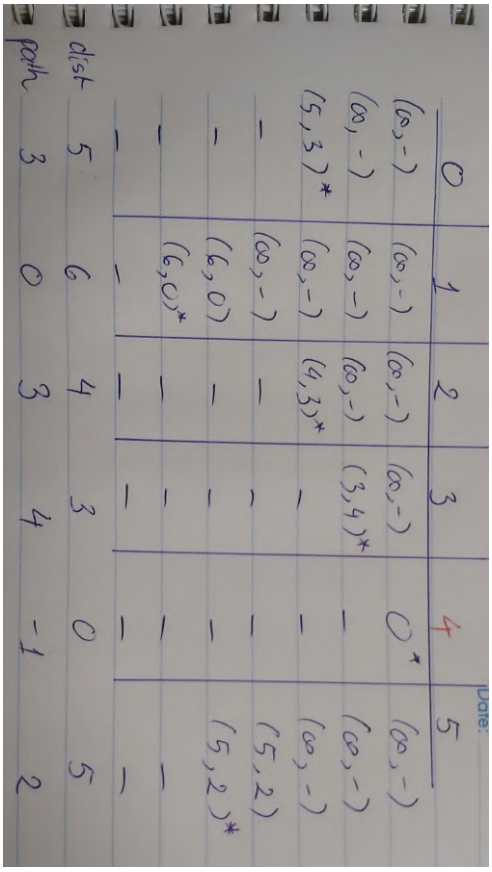
dist 7 8 0 5 2 1
path 3 0 -1 4 5 2

- *Đỉnh 3:*



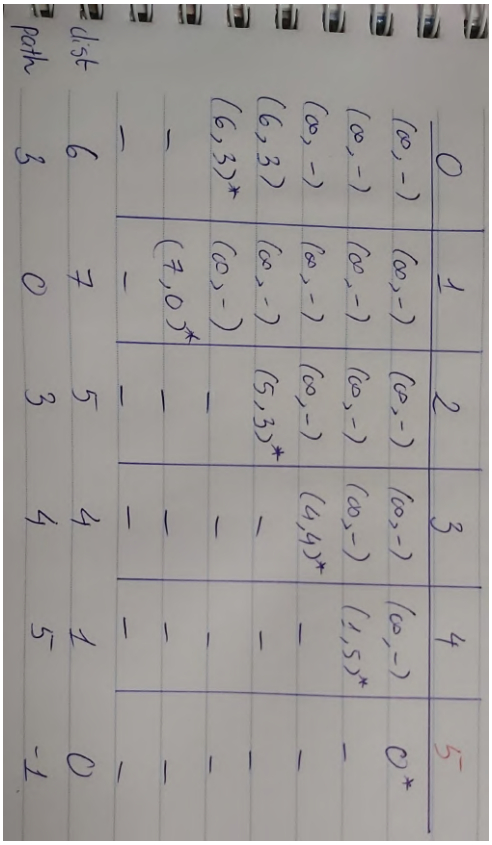
	0	1	2	3	4	5
	(∞,-)	(∞,-)	(∞,-)	0*	(∞,-)	(∞,-)
	(2,3)	(∞,-)	(1,3)*	-	(4,3)	(∞,-)
	(2,5)*	(∞,-)	-	-	(4,3)	(2,2)
	-	(3,0)	-	-	(4,3)	(2,2)*
	-	(3,0)*	-	-	(3,5)	-
	-	-	-	-	(3,5)*	-
dist	2	3	1	0	3	2
path	3	0	3	-1	5	2

- *Đỉnh 4:*



	0	1	2	3	4	5
	(0,-)	(∞,-)	(∞,-)	(∞,-)	0*	(∞,-)
	(∞,-)	(∞,-)	(∞,-)	(3,4)*	-	(∞,-)
	(5,3)*	(∞,-)	(4,3)*	-	-	(∞,-)
	-	(∞,-)	-	-	-	(5,2)
	-	(6,0)	-	-	-	(5,2)*
	-	(6,0)*	-	-	-	-
	-	-	-	-	-	-
path	3	0	3	4	-1	2
dist	5	6	4	3	0	5

- *Đỉnh 5:*



	0	1	2	3	4	5
	(∞, -)	(∞, -)	(∞, -)	(∞, -)	(∞, -)	0*
	(∞, -)	(∞, -)	(∞, -)	(∞, -)	(1, 5)*	-
	(∞, -)	(∞, -)	(∞, -)	(4, 4)*	-	-
	(6, 3)	(∞, -)	(5, 3)*	-	-	-
	(6, 3)*	(∞, -)	-	-	-	-
	-	(7, 0)*	-	-	-	-
path	-	0	3	4	5	-1
dist	6	7	5	4	1	0
	3	0	3	4	5	-1

Kiểm tra kết quả với output của chương trình:

```
Đỉnh 0
dist: [0, 1, 4, 3, 6, 5]
path: [-1, 0, 3, 1, 5, 2]
Đỉnh 1
dist: [4, 0, 3, 2, 5, 4]
path: [3, -1, 3, 1, 5, 2]
Đỉnh 2
dist: [7, 8, 0, 5, 2, 1]
path: [3, 0, -1, 4, 5, 2]
Đỉnh 3
dist: [2, 3, 1, 0, 3, 2]
path: [3, 0, 3, -1, 5, 2]
Đỉnh 4
dist: [5, 6, 4, 3, 0, 5]
path: [3, 0, 3, 4, -1, 2]
Đỉnh 5
dist: [6, 7, 5, 4, 1, 0]
path: [3, 0, 3, 4, 5, -1]
```

1.5 Độ phức tạp

Tính toán độ phức tạp dựa trên hàm dijsktra cho bài toán tìm đường đi ngắn nhất giữa mọi cặp đỉnh trong đồ thị

Trong đó:

- *V: số đỉnh của đồ thị*
- *E: số cạnh của đồ thị*
- *S: tập hợp các đỉnh có đường đi ngắn nhất cuối cùng với đỉnh source được xác định*
- *Z: độ phức tạp của việc thêm 1 phần tử vào priority queue*

Với mỗi đỉnh $u \in V$ được thêm vào tập S duy nhất một lần, từng cạnh trong danh sách kề được thực hiện trong vòng lặp duy nhất 1 lần trong toàn bộ thuật toán. Do đó, tổng số cạnh là E nên có tổng cộng E lần lặp trong vòng while.

$$\begin{aligned} T(n) &= \sum_{i=0}^{v}(3 + \sum^{E}(Z+1)) \\ &= V(3 + EZ + E) \\ &= 3V + 3EVZ + E \end{aligned}$$

Với độ phức tạp của việc thêm 1 phần tử vào priority queue là O(logN) với N là số phần tử trong hàng đợi ưu tiên.

Do đó, độ phức tạp của thuật toán là O(V(ElogV))

1.6 Thực nghiệm

- *Time Complexity*

V	T	Edge	E*V^2 Squared_Error	Sqrt(V) Squared_Error	Log(V) Squared_Error	V Squared_Error	V(Elog(V)) Squared_Error	V^2 Squared_Error
12	0.002156	120	0.000004	0.026482	0.104715	0.002068	0.000003	0.000015
13	0.002597	138	0.000006	0.028569	0.111322	0.002401	0.000004	0.000019
14	0.003290	168	0.000009	0.030558	0.117424	0.002733	0.000006	0.000033
15	0.003986	191	0.000014	0.032531	0.123215	0.003086	0.000009	0.000036
16	0.004545	213	0.000017	0.034541	0.128825	0.003476	0.000011	0.000042
17	0.005085	237	0.000021	0.036546	0.134197	0.003893	0.000013	0.000063
18	0.006058	287	0.000029	0.038372	0.139017	0.004275	0.000016	0.000063
19	0.006942	308	0.000038	0.040215	0.143681	0.004688	0.000021	0.000082
20	0.007993	342	0.000049	0.041975	0.148015	0.005097	0.000027	0.000100
21	0.008600	379	0.000055	0.043901	0.152502	0.005588	0.000028	0.000130
22	0.009430	415	0.000064	0.045723	0.156645	0.006067	0.000031	0.000134
23	0.011046	455	0.000087	0.047189	0.159996	0.006439	0.000043	0.000143
25	0.013314	543	0.000119	0.050482	0.167001	0.007381	0.000055	0.000216
27	0.017052	645	0.000188	0.053029	0.172233	0.008121	0.000086	0.000223
29	0.019987	724	0.000245	0.055858	0.177613	0.009048	0.000108	0.000289
31	0.022885	828	0.000296	0.058630	0.182574	0.010032	0.000119	0.000405
33	0.027538	947	0.000407	0.060466	0.185614	0.010701	0.000161	0.000419
35	0.035222	1077	0.000666	0.060702	0.185640	0.010753	0.000290	0.000353
38	0.041936	1252	0.000843	0.063245	0.189161	0.011858	0.000340	0.000487
41	0.055301	1497	0.001395	0.062241	0.186211	0.011542	0.000594	0.000388
44	0.057264	1697	0.001144	0.066808	0.192560	0.013778	0.000358	0.000826
47	0.067507	1952	0.001351	0.066986	0.191162	0.014171	0.000383	0.000930
51	0.085159	2326	0.001764	0.064906	0.185137	0.013752	0.000477	0.000891
55	0.104580	2656	0.002235	0.061715	0.177033	0.012933	0.000631	0.000866
59	0.122105	3053	0.002144	0.059298	0.170100	0.012561	0.000502	0.001017
64	0.149324	3605	0.001936	0.053578	0.156718	0.010963	0.000364	0.001068
69	0.182173	4209	0.001540	0.045461	0.139046	0.008409	0.000233	0.000831
75	0.244022	5006	0.001864	0.028291	0.103665	0.002880	0.000576	0.000025
81	0.281021	5830	0.000067	0.021718	0.087061	0.001639	0.000000	0.000100
88	0.361365	6868	0.000324	0.007251	0.050888	0.000146	0.000035	0.000337
MSE			0.000631	0.046242	0.150632	0.007349	0.000184	0.000351

- *Space Comlexity*

V	Edge	Size	E*V^2 Squared_Error	Sqrt(V) Squared_Error	(V+E) Squared_Error	V Squared_Error	V(Log(V)) Squared_Error	V^3 Squared_Error	V^2 Squared_Error
12	120	8300	67,964,435	5,655,404,383	26,918,519	234,852,327	65,219,313	61,964,779	27,542,501
13	138	9196	83,183,393	6,039,788,644	31,769,125	268,882,382	79,347,976	74,849,214	31,519,803
14	168	10136	100,588,719	6,409,111,223	34,171,437	303,678,526	95,007,370	89,415,709	35,784,539
15	191	11096	120,052,885	6,767,126,597	38,935,996	339,853,483	112,571,719	105,259,679	40,035,978
16	213	12092	141,978,208	7,111,260,911	44,805,383	376,664,894	132,252,060	122,698,555	44,440,769
17	237	13124	166,468,003	7,441,436,734	50,927,439	413,902,649	153,981,587	141,764,028	48,985,945
18	287	14176	192,514,052	7,760,405,539	48,805,607	452,044,518	175,314,532	162,071,384	53,424,609
19	308	15268	222,249,038	8,064,682,576	57,145,818	490,094,569	201,316,693	184,096,513	58,019,359
20	342	16400	254,628,982	8,354,172,815	61,880,483	527,842,383	228,505,518	207,863,038	62,765,784
21	379	17552	289,369,802	8,632,513,792	65,977,024	566,038,407	257,091,748	232,774,767	67,330,903
22	415	18740	327,237,188	8,896,793,400	71,206,712	603,798,060	288,053,697	259,249,769	71,947,853
23	455	19964	368,053,832	9,146,984,644	75,618,606	640,952,704	320,714,590	287,261,788	76,606,046
25	543	22496	457,842,056	9,609,756,518	82,924,451	714,093,385	390,408,063	346,847,603	85,560,373
27	645	25160	558,739,475	10,018,735,583	86,837,213	783,776,456	465,027,356	411,367,596	94,278,876
29	724	27956	675,203,877	10,374,194,828	104,146,108	848,993,319	553,527,831	480,134,342	102,657,971
31	828	30872	800,653,803	10,679,024,786	112,835,710	909,563,592	642,644,894	551,745,147	110,350,256
33	947	33920	935,205,645	10,931,351,268	117,029,758	964,006,499	733,059,126	625,708,551	117,504,928
35	1077	37100	1,077,715,095	11,131,840,098	118,512,296	1,011,617,572	823,959,124	700,915,353	124,046,446
38	1252	42096	1,313,541,015	11,341,749,513	136,568,744	1,070,347,074	976,870,610	812,120,843	132,070,086
41	1497	47392	1,540,150,640	11,439,538,431	124,011,611	1,110,647,758	1,096,534,612	918,825,831	138,421,084
44	1697	52972	1,792,274,199	11,432,520,690	142,340,215	1,132,497,851	1,244,099,311	1,015,179,142	142,583,517
47	1952	58840	2,014,181,281	11,324,376,849	137,280,929	1,135,071,925	1,347,831,373	1,096,324,944	144,550,846
51	2326	67116	2,258,990,805	11,031,018,252	122,809,268	1,108,210,293	1,436,285,886	1,172,562,640	143,783,290
55	2656	75900	2,488,806,136	10,581,826,820	143,817,790	1,048,513,408	1,538,951,413	1,201,959,669	138,975,853
59	3053	85196	2,579,479,161	9,991,717,692	140,078,026	958,504,317	1,535,579,961	1,176,448,772	130,426,381
64	3605	97536	2,473,027,161	9,082,963,911	121,993,140	810,166,622	1,400,848,595	1,060,936,096	115,052,632
69	4209	110672	2,097,076,457	8,020,953,401	96,527,203	633,585,846	1,131,773,950	856,239,830	95,421,502
75	5006	127492	1,319,550,217	6,603,846,928	59,526,686	406,569,854	673,299,293	526,418,234	68,508,117
81	5830	145476	468,128,625	5,107,912,023	37,618,585	195,776,724	239,639,782	189,761,232	41,262,393
88	6868	167896	18,469,341	3,390,689,773	15,359,775	28,656,866	2,615,540	970,201	14,222,960
MSE			906,777,451	8,745,789,954	83,612,655	669,640,142	611,411,118	502,457,842	85,269,387

2 Thuật toán Bellman Ford

2.1 Giới thiệu

*Tham khảo: "**link: https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_ algorithm** "*

2.2 Mô tả bài toán

Thuật toán Bellman-Ford là thuật toán tim đường đi có chi phí nhỏ nhất từ một đỉnh đến tất cả các đỉnh còn lại trong đồ thị có hướng hoặc vô hướng, có trọng số(trọng số có thể dương hoặc âm).

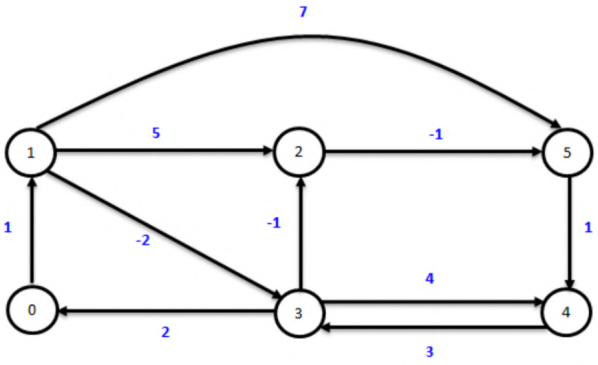
Input:

- *Danh sách cạnh kề.*

Output:

- *Hai mảng gồm n phần tử lưu vết đường đi cho mỗi đỉnh.*

Ví dụ:



Input: Danh sách cạnh kề và đỉnh xuất phát 0.

Edge List

```
6 10
0  1  1
1  2  5
1  3 -2
1  5  7
2  5 -1
3  0  2
3  2 -1
3  4  4
4  3  3
5  4  1
```

Output:

```
D:\src\DA_algo>bellman.py
Đỉnh:  0
path:  [-1, 0, 3, 1, 5, 2]
Đỉnh:  1
path:  [3, -1, 3, 1, 5, 2]
Đỉnh:  2
path:  [3, 0, -1, 4, 5, 2]
Đỉnh:  3
path:  [3, 0, 3, -1, 5, 2]
Đỉnh:  4
path:  [3, 0, 3, 4, -1, 2]
Đỉnh:  5
path:  [3, 0, 3, 4, 5, -1]

D:\src\DA_algo>
```

Với mỗi đỉnh, tương ứng 1 mảng path chứa đường đi từ đỉnh đó đến mọi đỉnh còn lại.

2.3 Cài đặt

Cài đặt thuật toán Bellman Ford cho bài toán tìm đường đi ngắn nhất giữa mọi cặp đỉnh trong đồ thị.

2.3.1 Cài đặt hàm Bellman ford

```
def BellmanFord(s):
    dist[s] = 0
    for i in range(1, n):
        for j in range(m):
            u = graph[j].source
            v = graph[j].target
            w = graph[j].weight
            if dist[u] != INF and dist[u] + w < dist[v] :
                dist[v] = dist[u] + w
    return
```

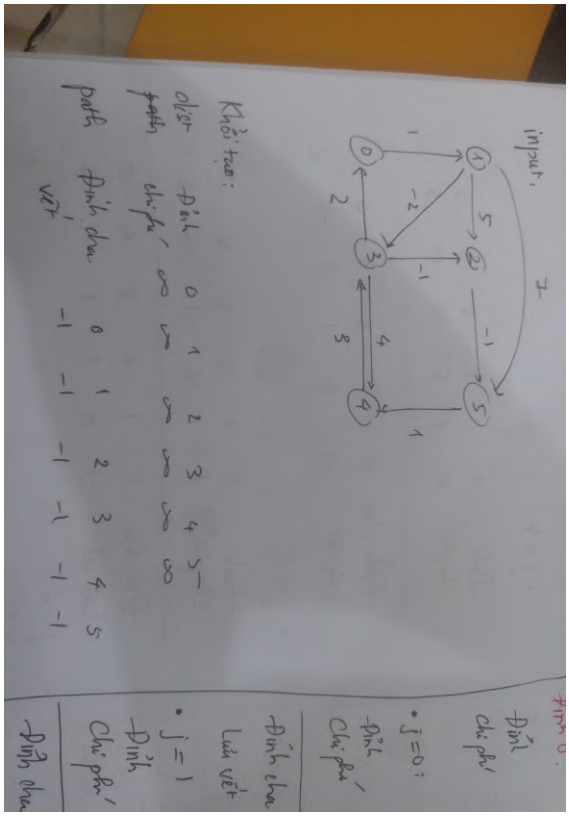
2.3.2 Cài đặt hoàn chỉnh

```
import sys
import queue
sys.stdin = open('D:/src/DA_algo/inp_bellman.txt', 'r')

INF = 10**9
MAX = 105

class edge:
    def __init__(self, source, target, weight):
        self.source = source
        self.target = target
        self.weight = weight


n, m = map(int, input().split())
#n, m: so dinh, so canh
def BellmanFord(s, dist, path):
    dist[s] = 0
    for i in range(1, n):
        for j in range(m):
            u = graph[j].source
            v = graph[j].target
            w = graph[j].weight
            if dist[u] != INF and dist[u] + w < dist[v] :
                dist[v] = dist[u] + w
                path[v] = u
    return

def traceback(path, source, destination):
    flag = destination
    res = []
    while flag != -1:
        res.append(flag)
        flag = path[flag]
```

```
    print("path: ", path[:n])
    '''
    for i in range(len(res)-1, 0, -1):
        print(res[i], end = '->')
    print(destination)
    '''
    return

if __name__ == '__main__':

    graph = []
    #do thi dau vao
    for i in range(m):
        u, v, w = map(int, input().split())
        graph.append(edge(u, v, w))

    for s in range(n):
        dist = [INF for _ in range(MAX)]
        #mang chua trong so nho nhat tu dinh i den dinh j
        path = [-1 for _ in range(MAX)]
        #luu vet duong di
        res = BellmanFord(s, dist, path)

        print("  nh  : ", s)
        for d in range(n):
            if (d != s):
                #print("trong so duong di ngan nhat tu dinh{} den dinh {}: {}
                    ".format(s, d, dist[d]))
                traceback(path, s, d)
                break
```

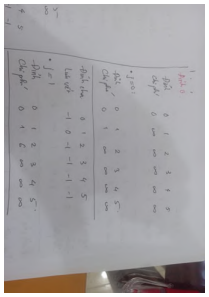
2.4 Kiểm nghiệm tính đúng đắn

Khởi tạo input như hình:

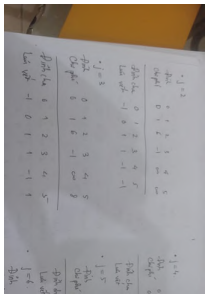


Duyệt qua từng đỉnh:
-Đỉnh 0:

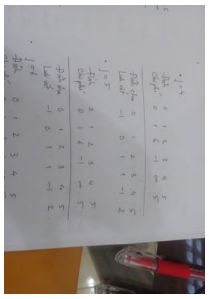
- *i = 0:*
  - *j = 0 - 1*



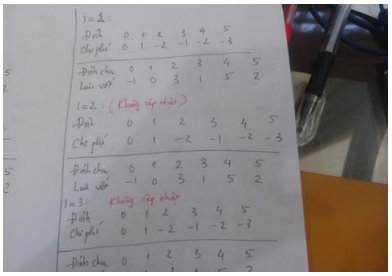
  - *j = 2 - 3*



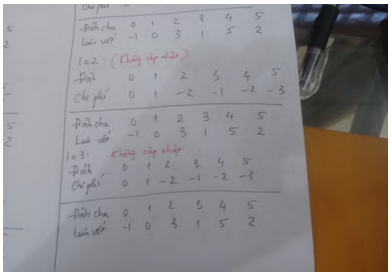
– *j = 4 - 5*



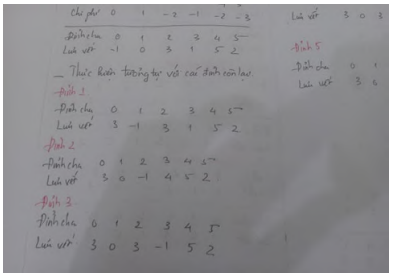
- *i = 1 2:*



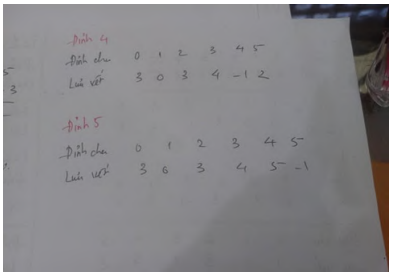
- *i = 2 3:*



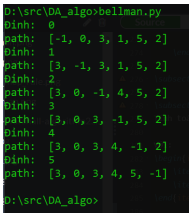
- Với đỉnh 1 2 3:



- *Với đỉnh 4 5:*



So sánh với output của code:



2.5 Độ phức tạp

Tính toán độ phức tạp của hàm bellman ford:
Với:

- *n: số đỉnh của đồ thị*
- *m: số cạnh của đồ thị*

$$\begin{aligned} T(n) &= \sum_{i=0}^{n}(\sum_{i=1}^{n}(\sum_{j=0}^{m}(3+5))) \\ &= n(8mn-8m) \\ &= 8mn^2-8mn \end{aligned}$$

Do đó thuật toán thuộc độ phức tạp $O(mn^2)$

2.6 Thực nghiệm

- *Time Compexity*

V	T	Edge	E*V^2 Squared_Error	Sqrt(V) Squared_Error	Log(V) Squared_Error	V Squared_Error	Vlog(E) Squared_Error	V^3 Squared_Error	V^2 Squared_Error
12	0.0153	120	0.000006	440.6674	1,639.2229	38.4047	0.1984	0.0132	0.7271
13	0.0204	138	0.000013	477.1926	1,746.1395	45.0203	0.2771	0.0209	0.9972
14	0.0292	168	0.000021	513.5341	1,847.7892	52.1080	0.4299	0.0312	1.3265
15	0.0392	191	0.000025	549.7976	1,944.8572	59.6966	0.5775	0.0457	1.7340
16	0.0505	213	0.000031	585.9621	2,037.7189	67.7771	0.7427	0.0658	2.2274
17	0.0635	237	0.000048	622.0155	2,126.7115	76.3425	0.9484	0.0933	2.8173
18	0.0907	287	0.000024	657.3023	2,210.9468	85.1541	1.4180	0.1199	3.4680
19	0.1032	308	0.0001	693.2948	2,293.4071	94.7344	1.6845	0.1688	4.2966
20	0.1273	342	0.0002	728.6261	2,371.8324	104.5862	2.1185	0.2234	5.2153
21	0.1608	379	0.0001	763.3791	2,446.6096	114.7243	2.6303	0.2854	6.2360
22	0.1891	415	0.0003	798.3436	2,519.3523	125.4474	3.2136	0.3720	7.4413
23	0.2328	455	0.0002	832.3539	2,588.1624	136.2899	3.8856	0.4626	8.7331
25	0.3308	543	0.0003	899.4438	2,718.0691	159.0569	5.6144	0.7077	11.8076
27	0.4461	645	0.0014	965.0281	2,838.1374	183.1103	8.0544	1.0606	15.5857
29	0.5905	724	0.0013	1,028.2799	2,947.9023	208.0202	10.0862	1.5340	20.0574
31	0.7701	828	0.0023	1,088.6370	3,047.2584	233.4377	13.1230	2.1459	25.2195
33	0.9962	947	0.0042	1,145.1810	3,135.4278	258.8217	16.9679	2.8892	31.0001
35	1.3006	1077	0.0032	1,195.5170	3,209.1717	282.8766	21.2976	3.6686	36.9951
38	1.8119	1252	0.0023	1,265.2911	3,303.8003	319.0064	27.2893	5.3087	47.4869
41	2.4842	1497	0.0109	1,321.0534	3,369.2197	351.2481	37.2617	7.2137	58.4893
44	3.2597	1697	0.0144	1,366.5094	3,412.9533	380.9992	44.3725	9.7988	70.7165
47	4.3001	1952	0.0184	1,389.4009	3,417.0075	401.2777	53.4442	12.1656	81.2507
51	6.0577	2326	0.0275	1,387.5694	3,367.3517	413.9243	65.1417	15.1502	92.5314
55	8.0859	2656	0.0321	1,360.7376	3,275.4348	415.6597	69.8423	19.3083	102.9641
59	10.7037	3053	0.0523	1,287.1663	3,108.9443	393.6552	72.7317	22.1024	105.6234
64	14.7788	3605	0.1689	1,138.0884	2,809.9153	336.8728	70.2816	23.8550	98.1915
69	20.3226	4209	0.0849	903.0772	2,370.8087	237.1237	51.9768	18.6484	70.1135
75	28.8280	5006	0.0193	561.2344	1,725.9579	99.9932	20.8150	7.9299	25.7557
81	39.8599	5830	0.2618	216.6444	1,009.1556	4.3015	0.0785	0.00001	0.0991
88	54.5341	6868	0.0316	5.5419	340.1821	80.5736	49.3931	11.6832	61.7649
MSE			0.0246	872.8957	2,505.9816	192.0081	21.8632	5.5691	33.3624

- *Space Compexity*

V	Edge	Size	E*v^2 squared_error	Sqrt(v) squared_error	Log(v) squared_error	V squared_error	Vlog(e) squared_error	V^3 squared_error	V^2 squared_error
12	120	24192	582710326	3976573786	24241600026	5.29395592033938E-23	512095869	565516318	448440722
13	138	26208	683142753	4174044415	25432483269	5.29395592033938E-23	593082578	659715321	513878166
14	168	28224	790945634	4358510800	26511827055	5.29395592033938E-23	670854103	760143002	581748302
15	191	30240	906562065	4530742704	27490671216	5.29395592033938E-23	757887270	866505118	651686698
16	213	32256	1029767956	4691431411	28378511713	5.29395592033938E-23	850395803	978488324	723339250
17	237	34272	1160320903	4841202221	29183575532	5.29395592033938E-23	945783267	1095761074	796362587
18	287	36288	1296353695	4980624328	29913038559	5.29395592033938E-23	1020249959	1217973704	870423646
19	308	38304	1441379566	5110218742	30573199356	5.29395592033938E-23	1125739842	1344759747	945200054
20	342	40320	1592291953	5230464728	31169618760	5.29395592033938E-23	1221611647	1475735966	1020379909
21	379	42336	1749512200	5341805100	31707232880	5.29395592033938E-23	1316954875	1610503742	1095661835
22	415	44352	1913232400	5444650621	32190445260	5.29395592033938E-23	1416047836	1748649529	1170754980
23	455	46368	2082575174	5539383693	32623202594	5.29395592033938E-23	1512631861	1889745577	1245379018
25	543	50400	2437083327	5705918559	33351220110	2.11758236813575E-22	1700863781	2179013157	1392151092
27	645	54432	2809056393	5844009439	33915856990	2.11758236813575E-22	1874288066	2474639345	1533945947
29	724	58464	3204731135	5955960829	34337458097	2.11758236813575E-22	2087544356	2772792507	1668899794
31	828	62496	3608835898	6043833985	34633083931	2.11758236813575E-22	2265504543	3069511246	1795317403
33	947	66528	4018091144	6109486935	34817201323	2.11758236813575E-22	2418460537	3360740146	1911671784
35	1077	70560	4428039336	6154605840	34902199394	2.11758236813575E-22	2549849790	3642369698	2016604610
38	1252	76608	5055801350	6187114892	34866876433	2.11758236813575E-22	2781968600	4037805114	2150030837
41	1497	82656	5624254123	6181533856	34662831870	2.11758236813575E-22	2865089721	4388645267	2251813878
44	1697	88704	6194023854	6142015940	34315252987	2.11758236813575E-22	3031040610	4681686813	2319500222
47	1952	94752	6662618698	6072281075	33845197426	2.11758236813575E-22	3066054728	4904643780	2351488709
51	2326	102816	7123008132	5938054424	33058344166	8.470329472543E-22	3013497718	5074227051	2337439178
55	2656	110880	7468500148	5763169984	32120430004	8.470329472543E-22	3037087155	5079182701	2259277620
59	3053	118944	7497863628	5553961590	31060541534	8.470329472543E-22	2885254031	4906451362	2120282004
64	3605	129024	7067916741	5252777718	29600804362	8.470329472543E-22	2546626450	4437037977	1870344180
69	4209	139104	6099320497	4916693754	28027605015	8.470329472543E-22	2090571755	3707652381	1552466855
75	5006	151200	4286968438	4479651929	26035336598	8.470329472543E-22	1440149685	2576306041	1115818531
81	5830	163296	2194711026	4018489300	23972363183	8.470329472543E-22	844072625	1351823834	670737171
88	6868	177408	239999012	3465883146	21526339824	8.470329472543E-22	265063534	229790006	232158855
MSE			3374987250	5266836525	30615478316	3.17637355220363E-22	1756877420	2569593862	1387106795

3 Thuật toán Floyd-warshall

3.1 Giới thiệu

*Tham khảo: "**link:https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_to%C3%A1n_Floyd-Warshall**"*

3.2 Mô tả bài toán

Tìm kiếm đường đi cho tất cả các đỉnh trên của đồ thị có hướng sao cho đường đi đến tất cả các cạnh là ngắn nhất.

Thuật toán Floyd-Warshall so sánh tất cả các đường đi có thể giữa từng cặp đỉnh. Nó là một dạng của quy hoạch động (Dynamic Programming). Đặt hàm adj(i,j,k) là đường đi ngắn nhất từ i đến j, chỉ dùng các đỉnh trong tập (1,2,...,k). Giả sử ta muốn tính adj(i,j,k+1). Với mỗi cặp đỉnh i và j, đường đi ngắn nhất có thể là: (1) đường đi chỉ sử dụng các đỉnh trong tập (1,...k) hoặc (2) đường đi từ i đến k+1 rồi từ k+1 đến j, cũng chỉ sử dụng các đỉnh trong tập (1,...k). Do vậy:

```
    for k in range(N):
        for i in range(N):
            for j in range(N):
                adj[i][j] = min(adj[i][j], adj[i][k] + adj[k][j])
```

Thông qua đó thu được kết quả là một ma trân chứa giá trị khoảng cách tương ứng giữa các đỉnh với nhau.

Vậy Input và Output của bài toán là:

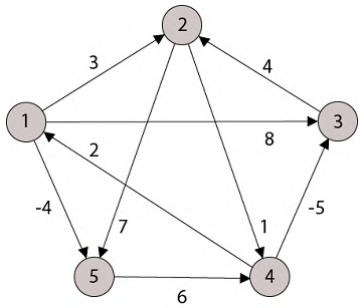
Input:

- *Ma trận đường đi của đồ thị có hướng*

Output:

- *Ma trận kết quả đường đi ngắn nhất theo từng đỉnh đồ thị*

Ví dụ:



Hình 1: Đồ Thị Hướng

Input:

Ma trận E:

0	*3*	*8*	*inf*	*-4*
inf	*0*	*inf*	*1*	*7*
inf	*4*	*0*	*inf*	*inf*
2	*inf*	*-5*	*0*	*inf*
inf	*inf*	*inf*	*6*	*0*

Trong đó inf(infinite) là một số nguyên vô cùng lớn

Output:

Ma trận Đường Đi Ngắn Nhất:

0	*1*	*3*	*2*	*4*
3	*0*	*4*	*1*	*-1*
7	*4*	*0*	*5*	*3*
2	*-1*	*-5*	*0*	*-2*
8	*5*	*1*	*6*	*0*

3.3 Mô tả bài toán ứng dụng thực tế - hệ thống giao thông

Dựa trên thuật toán floyd warshall có thể xây dựng hệ thống giao thông với:

- *Đỉnh(node): biểu diễn cho các giao lộ trong thành phố*
- *Cạnh(edge): biểu diễn cho đoạn đường giữa 2 giao lộ*
- *trọng số: biểu diễn cho độ dài đường đi (có thể thêm các yếu tố kẹt xe, cước phí vào trọng số)*

3.4 Công thức truy hồi

$$d_{(ij)}^{(k)} = \begin{cases} w(i,j) & (k=0) \quad (1a) \\ Min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{còn lại} \quad (1b) \end{cases}$$

Trong đó:

- $d_{(ij)}^{(k)}$*: trọng số của đường đi ngắn nhất từ đỉnh i đến đỉnh j với tập các đỉnh trung gian là {1,2,3,4,...,k}*
- $w(i,j)$*: trọng số của cạnh (i;j)*
- k*: số lượng đỉnh trung gian*

3.5 Cài đặt

3.5.1 Bottom - up

- *V: số đỉnh của ma trận*
- *graph: ma trận đầu vào*
- *dist: trọng số đường đi ngắn nhất từ đỉnh i đến j*

```
1
2       def floydwarshall(graph,dist):
3           for i in range(V):
4               for j in range(V):
5                   dist[i][j] = graph[i][j]
6
7           for k in range(V):
8               for i in range(V):
9                   for j in range(V):
10                      dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])
11          return
```

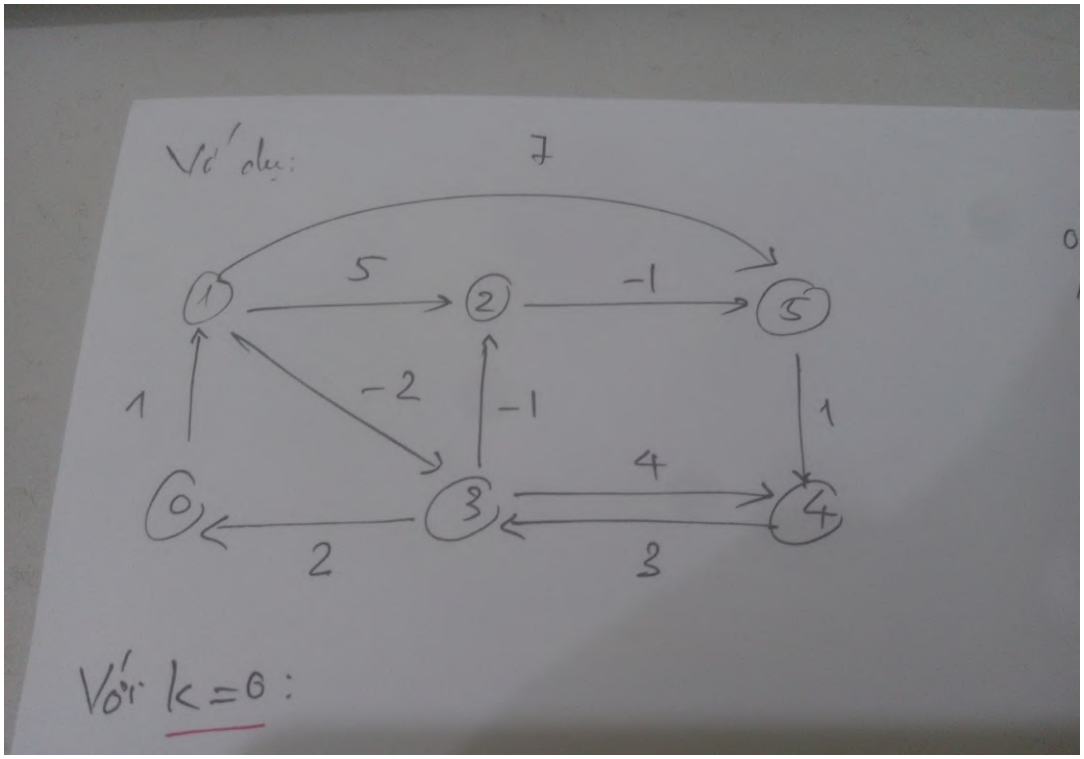
3.5.2 Cài đặt hoàn chỉnh

```
1
2 #V:  s    nh   input
3 #graph: ma  trn    u    vo
4 import sys
5 sys.stdin = open('D:/src/DA_algo/inp.txt', 'r')
6
7
8 def floydwarshall(graph,dist, V, path):
9     for i in range(V):
10        for j in range(V):
11            dist[i][j] = graph[i][j]
12            if graph[i][j] != INF and i != j:
13                path[i][j] = i
14            else:
15                path[i][j] = -1
16    '''
17    print("\ nKT   QU  MA  TRN  PATH   TRC   KHI  CHY \n")
18    for i in range(V):
19        print(path[i])
20    '''
21
22    for k in range(V):
23        for i in range(V):
24            for j in range(V):
25                if dist[i][j] > dist[i][k] + dist[k][j]:
26                    dist[i][j] = dist[i][k] + dist[k][j]
27                    path[i][j] = path[k][j]
28    for i in range(V):
29        if dist[i][i] < 0:
30            return False
31
32    return True
33
34 #traceback
35
```

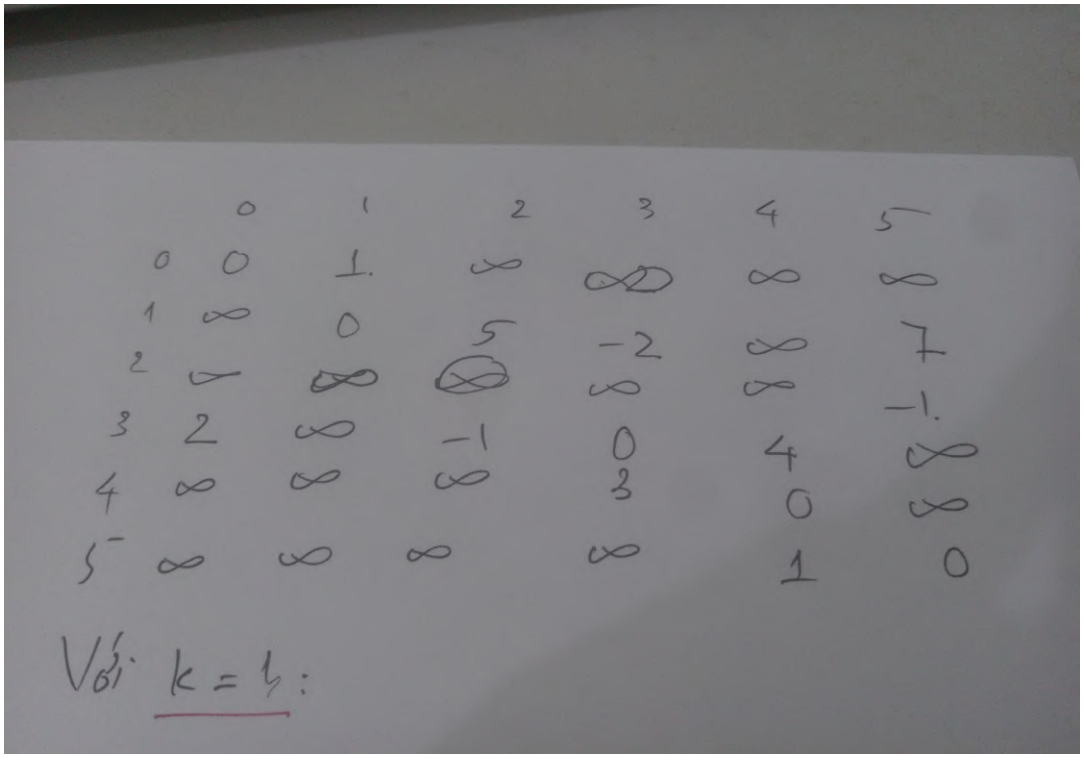
```
V = int(input())
graph = [[None for i in range(V)] for j in range(V)]
dist  = [[None for i in range(V)] for j in range(V)]
path = [[None for i in range(V)] for j in range(V)]
INF = int(1e9)
MAX = 100

for i in range(V):
    line = list(map(int, input().split()))
    for j in range(V):
        graph[i][j] = INF if line[j] == 0 and i!=j else line[j]
'''
for i in range(V):
    print(graph[i])
'''


floydwarshall(graph,dist, V, path)

'''
print("\nket qua ma tran dist\n")
for i in range(V):
    print(dist[i])
'''
print("\nkat qua ma tran path\n")

for i in range(V):
    print(path[i])
```

3.6 Kiểm tra tính đúng đắn

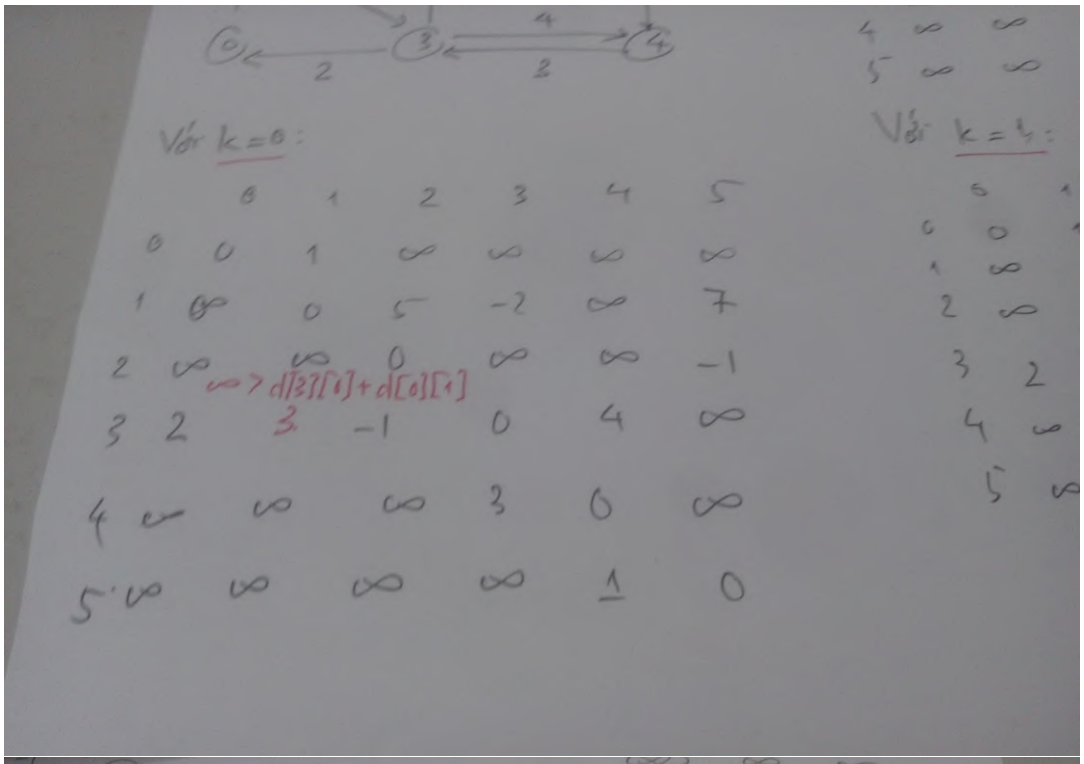
Khởi tạo input như hình:

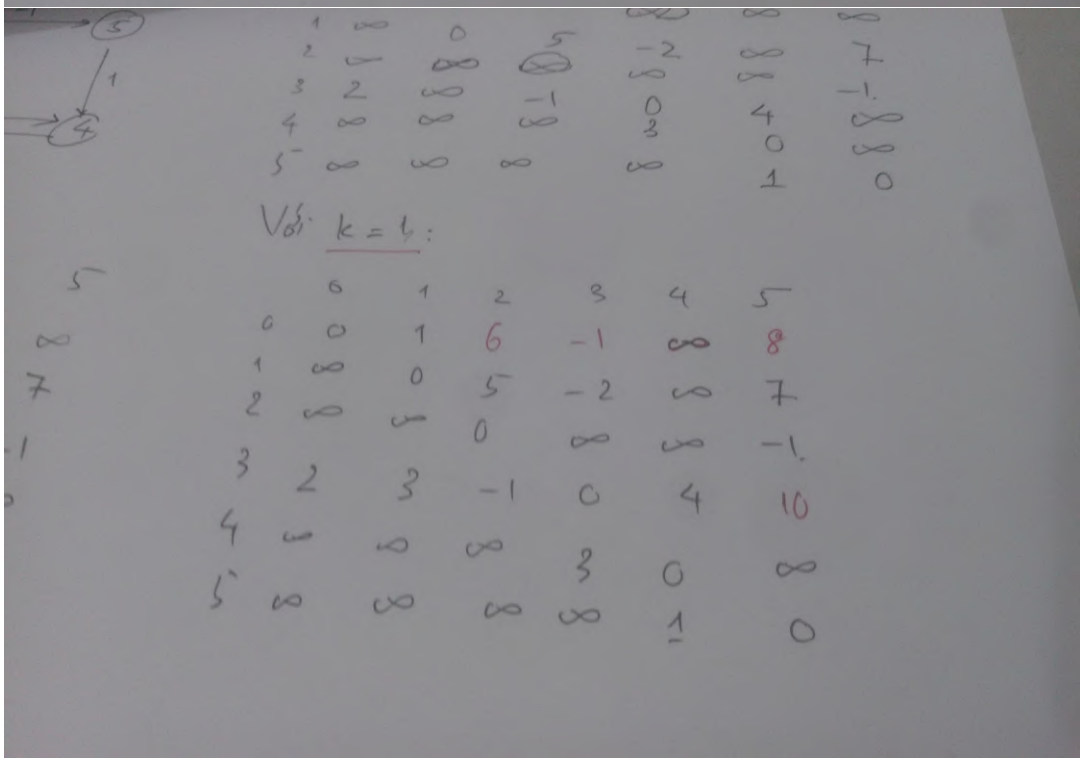


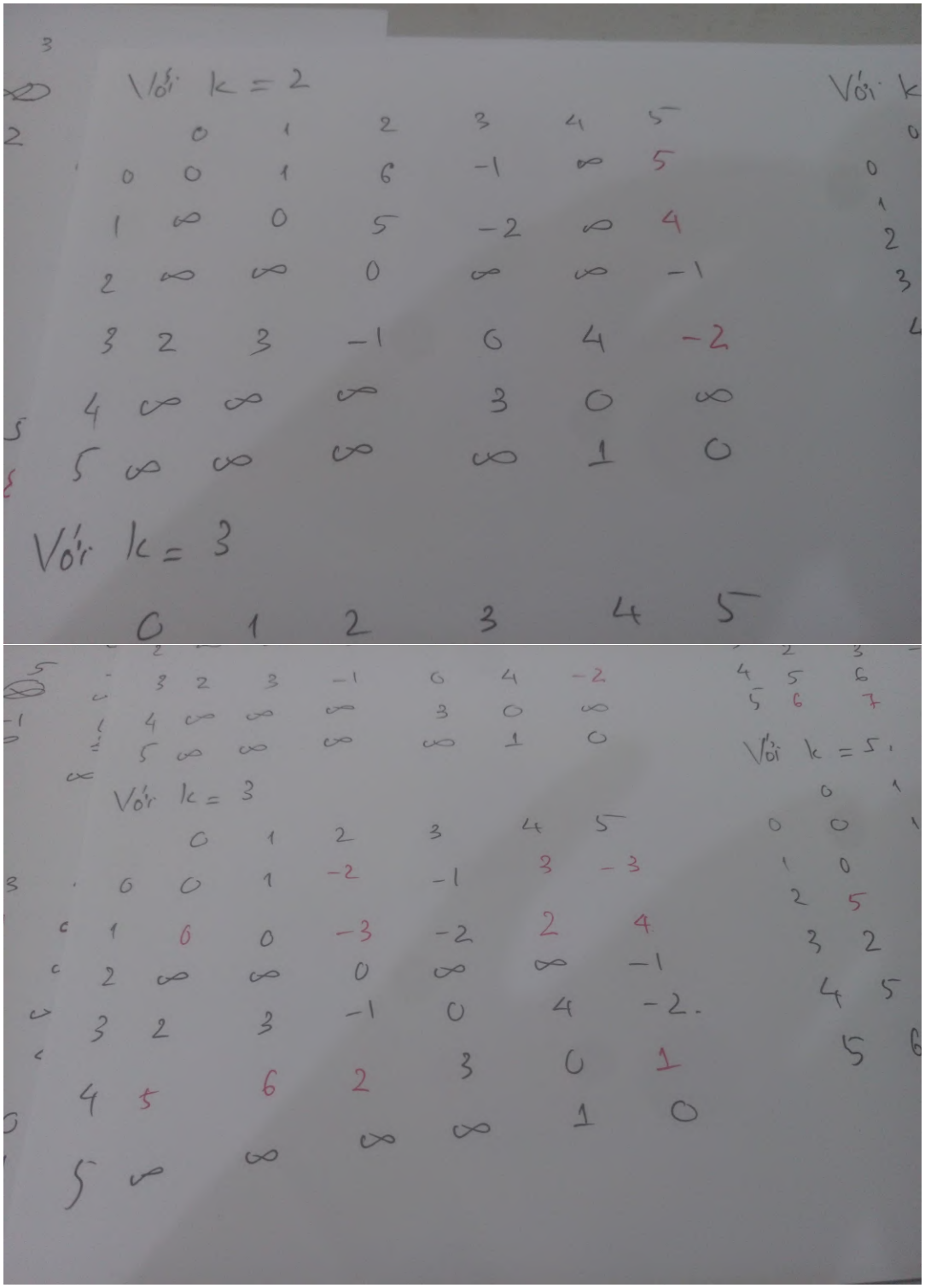
Khởi tạo ma trận đầu vào như hình:

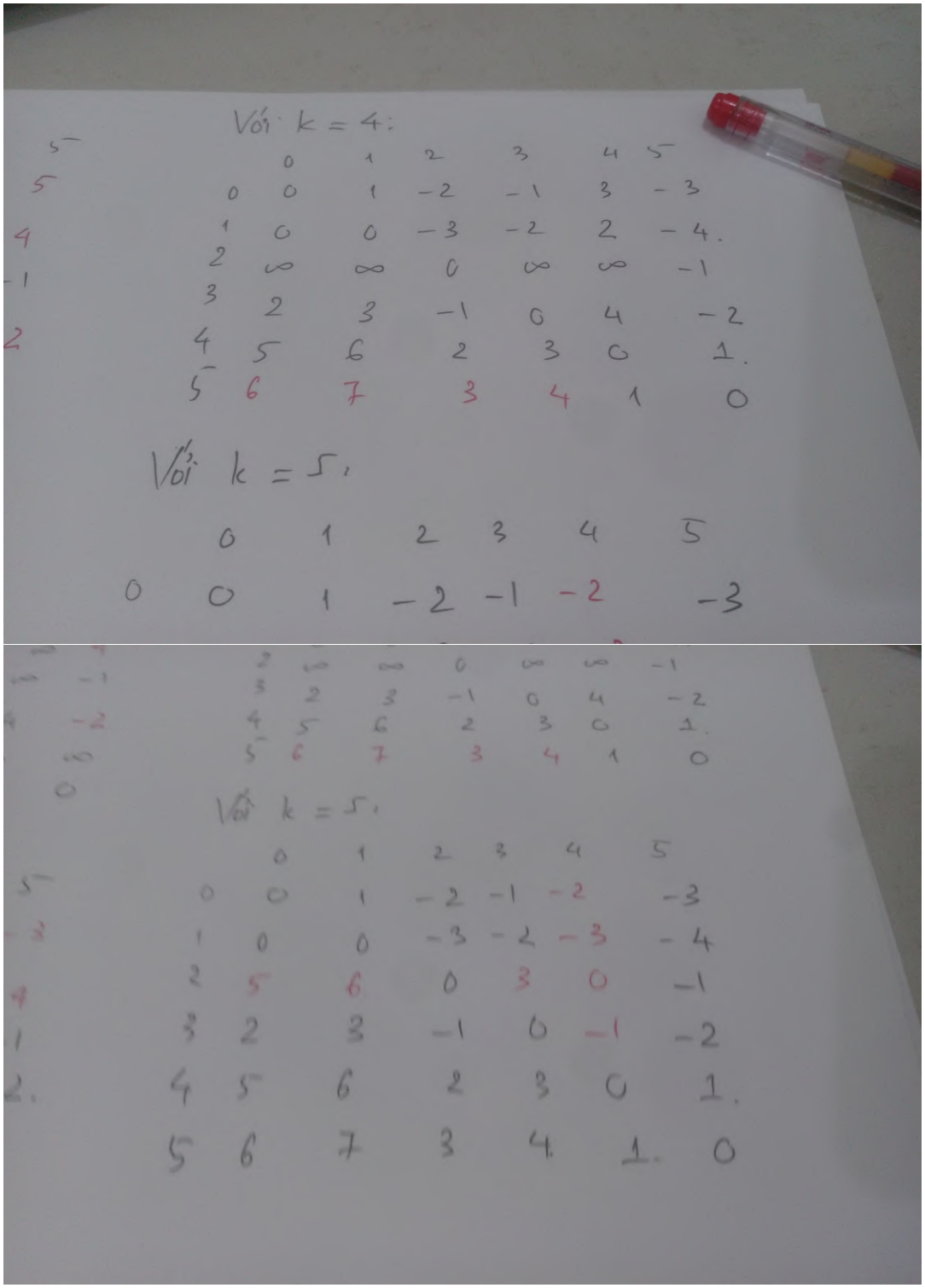


Chạy tay các bước với k từ 0 -> 5 như bảng sau:



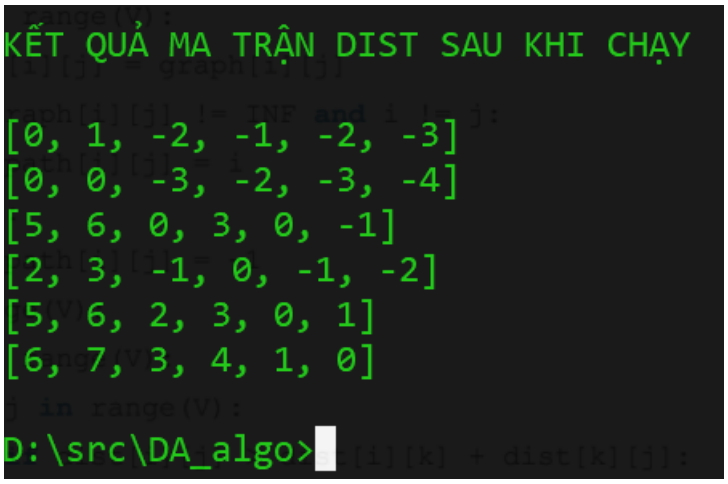



Với k = 2
Với k = 3
Với k = 3
Với k = 5.



Kết quả ở bảng 5 chính là output của bài toán.

Kiểm tra với output của bài toán bởi thuật toán bottom-up:



3.7 Độ phức tạp thuật toán

Độ phức tạp của thuật toán Floyd-Warshall dựa trên code bottom-up
Với:

- *n: số đỉnh của đồ thị*
- *x: số đỉnh trung gian*

Ta có:

$$\begin{aligned} T(n) &= \sum_{k=0}^{n}(\sum_{i=0}^{n}(\sum_{j=0}^{n}2)) \\ &= 2(n^3) \end{aligned}$$

Do đó thuật toán thuộc lớp $O(n^3)$ *với n là số đỉnh của đồ thị*

3.8 Thực nghiệm

- *Khảo sát sự ảnh hưởng của số cạnh đến thời gian thực hiện*
- *Time Complexity*
  - *n: số đỉnh trong đồ thị*
  - *t: thời gian thực hiện*

V	T	Edge	E*V^2 Squared_Error	Sqrt(V) Squared_Error	Log(V) Squared_Error	V Squared_Error	Vlog(E) Squared_Error	V^3 Squared_Error	V^2 Squared_Error
12	0.00400	120	0.00001	0.14039	0.54609	0.01126	0.00001	0.00000401	0.00012
13	0.00496	138	0.00002	0.15147	0.58057	0.01307	0.00002	0.00000385	0.00017
14	0.00371	168	0.00001	0.16429	0.61678	0.01556	0.00006	0.00000051	0.00027
15	0.00448	191	0.00001	0.17549	0.64838	0.01773	0.00008	0.00000023	0.00034
16	0.00551	213	0.00002	0.18642	0.67813	0.01997	0.00010	0.00000026	0.00046
17	0.00671	237	0.00003	0.19716	0.70629	0.02228	0.00012	0.00000051	0.00054
18	0.00944	287	0.00006	0.20645	0.73064	0.02425	0.00016	0.00000594	0.00060
19	0.01059	308	0.00008	0.21708	0.75653	0.02681	0.00018	0.00000254	0.00070
20	0.01069	342	0.00007	0.22867	0.78326	0.02987	0.00027	0.00000048	0.00092
21	0.01250	379	0.00009	0.23860	0.80605	0.03247	0.00032	0.00000025	0.00112
22	0.01392	415	0.00011	0.24883	0.82863	0.03532	0.00040	0.00000085	0.00130
23	0.01961	455	0.00024	0.25469	0.84253	0.03665	0.00033	0.00002124	0.00125
25	0.02013	543	0.00021	0.27717	0.88793	0.04379	0.00068	0.00000002	0.00201
27	0.02574	645	0.00032	0.29410	0.92104	0.04929	0.00092	0.00000054	0.00253
29	0.03150	724	0.00045	0.31048	0.95123	0.05504	0.00108	0.00000025	0.00308
31	0.03761	828	0.00058	0.32611	0.97841	0.06093	0.00140	0.00000037	0.00389
33	0.05078	947	0.00111	0.33318	0.98936	0.06351	0.00134	0.00003340	0.00387
35	0.05595	1077	0.00113	0.34904	1.01424	0.07033	0.00204	0.00000378	0.00505
38	0.07062	1252	0.00160	0.36393	1.03430	0.07731	0.00246	0.00000264	0.00630
41	0.08895	1497	0.00215	0.37337	1.04326	0.08252	0.00334	0.00000379	0.00723
44	0.11436	1697	0.00345	0.37306	1.03449	0.08374	0.00304	0.00005419	0.00751
47	0.14737	1952	0.00553	0.36251	1.00764	0.08059	0.00260	0.00026794	0.00666
51	0.16630	2326	0.00408	0.37748	1.01869	0.09100	0.00564	0.00000048	0.01075
55	0.20709	2656	0.00505	0.36438	0.98225	0.08855	0.00545	0.00000363	0.01143
59	0.25719	3053	0.00596	0.33932	0.92541	0.08076	0.00509	0.00000328	0.01078
64	0.34070	3605	0.00822	0.28501	0.81504	0.06079	0.00303	0.00011446	0.00694
69	0.41156	4209	0.00522	0.24653	0.73004	0.04909	0.00346	0.00000208	0.00663
75	0.53654	5006	0.00356	0.16826	0.56909	0.02300	0.00115	0.00003068	0.00216
81	0.65414	5830	0.00004	0.10873	0.43532	0.00794	0.00049	0.00022091	0.00067
88	0.86883	6868	0.00101	0.02455	0.22078	0.00377	0.00328	0.00011738	0.00433
MSE			0.00168	0.25623	0.80275	0.04524	0.00162	0.00003002	0.00365

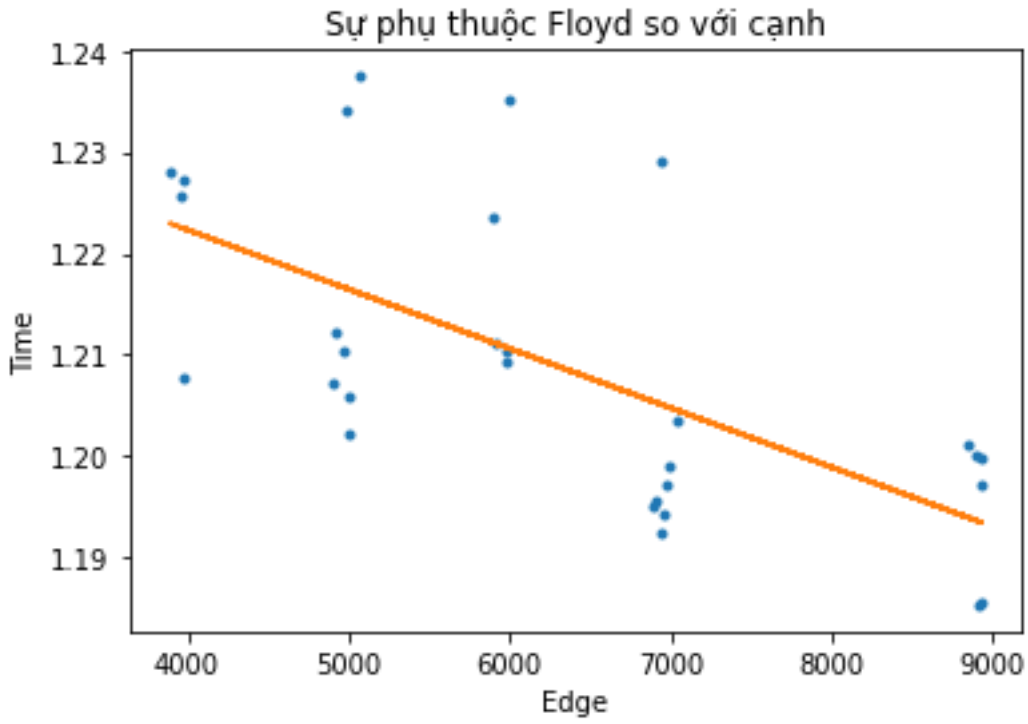
- *Space Complexity*

V	Edge	Size	E*V^2 Squared_Error	Sqrt(V) Squared_Error	Log(V) Squared_Error	V Squared_Error	Vlog(E) Squared_Error	V^3 Squared_Error	V^2 Squared_Error
12	120	2612	6600281	3510583464	14726058284	225385562	1977422	5224904	94864
13	138	3012	8726716	3767037332	15611006034	258617337	2498528	6745822	94864
14	168	3444	11304751	4016363745	16437544933	293036914	2726363	8561558	94864
15	191	3908	14449969	4258347676	17209974702	328445239	3299442	10698787	94864
16	213	4404	18221205	4492791524	17931991395	364649403	4063997	13182687	94864
17	237	4932	22676275	4719513874	18606795437	401462641	4914016	16036597	94864
18	287	5492	27679707	4938348485	19237177849	438704330	4569278	19281689	94864
19	308	6084	33732492	5149143434	19825589318	476199993	5840938	22936676	94864
20	342	6708	40556319	5351760389	20374195839	513781297	6579465	27017520	94864
21	379	7364	48289513	5546073992	20884923830	551286052	7279518	31537198	94864
22	415	8052	57053110	5731971317	21359496941	588558213	8207010	36505450	94864
23	455	8772	66821759	5909351402	21799466304	625447877	9018185	41928629	94864
25	543	10308	89595267	6238213385	22581081624	697510832	10498819	54147007	94864
27	645	11972	116741174	6532076935	23239580458	766398585	11396047	68168719	94864
29	724	13764	150121369	6790528349	23783242186	831133107	15254716	83907102	94864
31	828	15684	187925682	7013315174	24219188810	890834673	17525336	101210542	94864
33	947	17732	230182523	7200339588	24553657866	944721864	18857032	119862273	94864
35	1077	19908	276646135	7351652867	24792205312	992111563	19573165	139581820	94864
38	1252	23412	358109853	7512131030	24981056654	1049760918	24958642	170401642	94864
41	1497	27204	439184037	7593981789	24980633400	1089939106	22317016	201462936	94864
44	1697	31284	534894340	7599140687	24804827050	1111584769	28261659	231155322	94864
47	1952	35652	622364565	7530208704	24466555120	1114134215	27683176	257708569	94864
51	2326	41924	723859245	7328754458	23783964666	1087785804	24427995	285050377	94864
55	2656	48708	827254708	7010881762	22862439232	1028648259	32134995	299336573	94864
59	3053	56004	877374008	6587993529	21729572453	939514187	32090766	297029748	94864
64	3605	65844	851832283	5932169429	20058371000	792714259	27221700	267617110	94864
69	4209	76484	714794920	5163326277	18154922014	617963309	19531601	209417598	94864
75	5006	90308	416238299	4140593327	15643957668	393914964	8466421	113860488	94864
81	5830	105284	106587367	3073443937	12982121841	187244040	2840537	24638705	94864
88	6868	124212	61238078	1875891836	9829944992	25370109	17892	19621826	94864
MSE			264701865	5662197657	20381718107	654228647	13467723	106127862	94864

3.9 Sự ảnh hưởng của cạnh đến thời gian thực hiện

Bảng sự ảnh hưởng của cạnh lên độ phức tạp của thuật toán

V	T	Edge	Size
100	1.22727513313293	3969	160308
100	1.21043229103088	5975	160308
100	1.21038579940796	4974	160308
100	1.18536162376404	8918	160308
100	1.20362973213196	7046	160308
100	1.1855366230011	8931	160308
100	1.19551348686218	6908	160308
100	1.2076723575592	3974	160308
100	1.20008969306946	8887	160308
100	1.20109510421753	8847	160308
100	1.20720553398132	4896	160308
100	1.1971116065979	6977	160308
100	1.22370195388794	5890	160308
100	1.20224356651306	4994	160308
100	1.198979139328	6991	160308
100	1.19237327575684	6943	160308
100	1.19990706443787	8930	160308
100	1.2281641960144	3888	160308
100	1.23748016357422	5073	160308
100	1.2111074924469	5909	160308
100	1.22914695739746	6934	160308
100	1.19503736495972	6898	160308
100	1.21222472190857	4926	160308
100	1.19719314575195	8925	160308
100	1.2256076335907	3952	160308
100	1.20600819587708	5005	160308
100	1.23410940170288	4978	160308
100	1.23521041870117	5998	160308
100	1.20922207832336	5976	160308
100	1.19435691833496	6958	160308



Dựa vào đồ thị ta có:

- *Các điểm màu xanh là các thể hiện sự thay đổi của thời gian khi cạnh thay đổi → thời gian có xu hướng giảm khi cạnh tăng.*
- *Tuy nhiên, các điểm dữ liệu xảy ra một cách ngẫu nhiên, không tuyến tính (mô hình Logistic Regression bị Underfitting). Do đó, không thể kết luận Floyd phụ thuộc vào giá trị của cạnh.*

4 Kết luận

- *So sánh:*

  Nhóm thực hiện so sánh giữa 3 thuật toán, áp dụng cho bài toán tìm đường đi ngắn nhất với mọi cặp đỉnh thuộc đồ thị

  Trong đó:

  - *V: số đỉnh thuộc đồ thị*
  - *E: số cạnh thuộc đồ thị*

	DIJSKTRA	BELLMAN FORD	FLOYD WARSHALL
Time Complexity	O(V(ElogV))	O(V(E*V))	O(V^3)
Space Complexity	O(V+E)	O(V)	O(V^2)

Từ kết quả thực nghiệm, cài đặt thuật toán dijsktra thu được thời gian thực thi nhỏ nhất.

- *Nhận xét:*

  Qua đồ án môn học, nhóm tìm hiểu được:

  - *Các bài toán về đồ thị: tìm đường đi ngắn nhất từ 1 đỉnh đến mọi đỉnh, đường đi giữa mọi cặp đỉnh.*
  - *Các thuật toán liên quan về đường đi trong đồ thị: dijsktra, bellma-ford, floyd-warshall.*
  - *Việc tính toán độ phức tạp của thuật toán và thực nghiệm theo phương pháp Machine Learning.*
  - *Tìm hiểu ứng dụng các thuật toán đó vào cuộc sống như: google map, hệ thống giao thông, ...*

Tài liệu

[1] algorithms "***link: https://www.giaithuatlaptrinh.com***"