classmate
Date
Page 1

PSA Class 10
Ruchika Shashidhara
NU 002245068

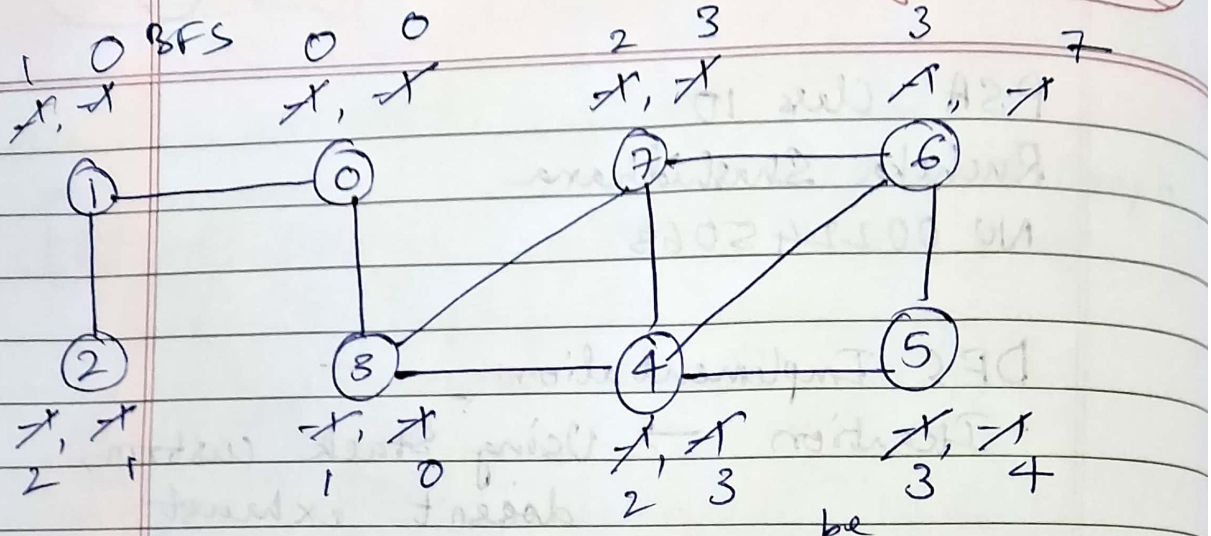
DFS Implementation:
· Iteration → Using Stack custom, doesn't exhaust stack frame of machine
· Recursion → Easy to implement, exhausts stacke frame of machine when deep graphs are tested / used for DFS

· DFS (→ uses Stack) gives the longest path, if the weights of the graph are uniform / unweighted graph
· DFS can check whether a graph has a loop or not → DAG

· BFS (→ uses Queue) provides a shortest path, if the weights of the graph are uniform / unweighted graph

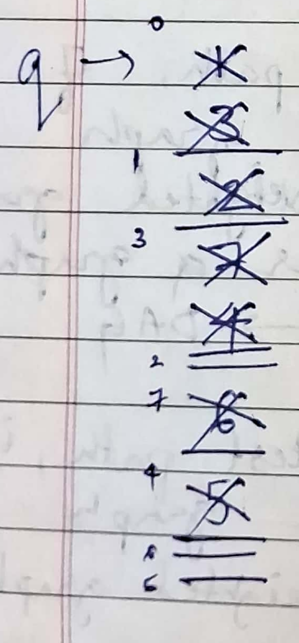
· Dijaktra's Algo can give the shortest path to any weighted (not uniform) graph.

· In both DFS & BFS each node is visited exactly once



Starting node will always be given for BFS unlike DFS, Here 0.

	0	1	2	3	4	5	6	7
cost	0	1	2	1	2	3	3	2
from	0	0	1	0	3	4	7	3



How did we reach 7?
7 → 3 → 0 cost = 2
rev
0 → 3 → 7 cost = 2
∴ This is also the shortest path

How did we reach 5?
5 → 4 → 3 → 0
rev
0 → 3 → 4 → 5
cost = 4
Shortest path from 0 → 5

∴ BFS gives shortest path to ~~from~~ every node give an uniformly weighted graph

Space used in this algo, N: Nodes
E: Edges

DS: 2N

Queue: N $\approx O(N)$

Work Done in this BFS algo:

N+E : Directed Graph

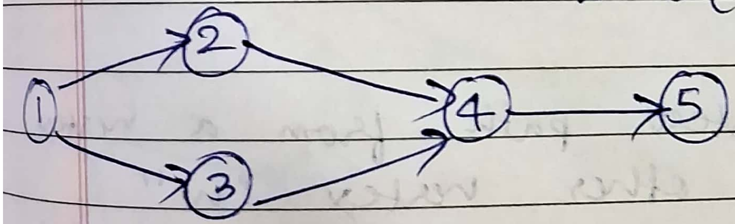
N+2E: Undirected graph

$\approx O(N+E)$

∴ Shortest Path from all cities to other cities Matrix Construction:

work done = $O(\cancel{N}(N+E))$

$\approx O(N^2)$



1	2	3	4	5	
0	1	1	2	3	Cost
0	1	1	2	4	From

q → ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~

Shortest paths

2 → 1, 3 → 1

1 → 2, 1 → 3, cost = 1

4 → 2 → 1

1 → 2 → 4, cost = 2

5 → 4 → 2 → 1

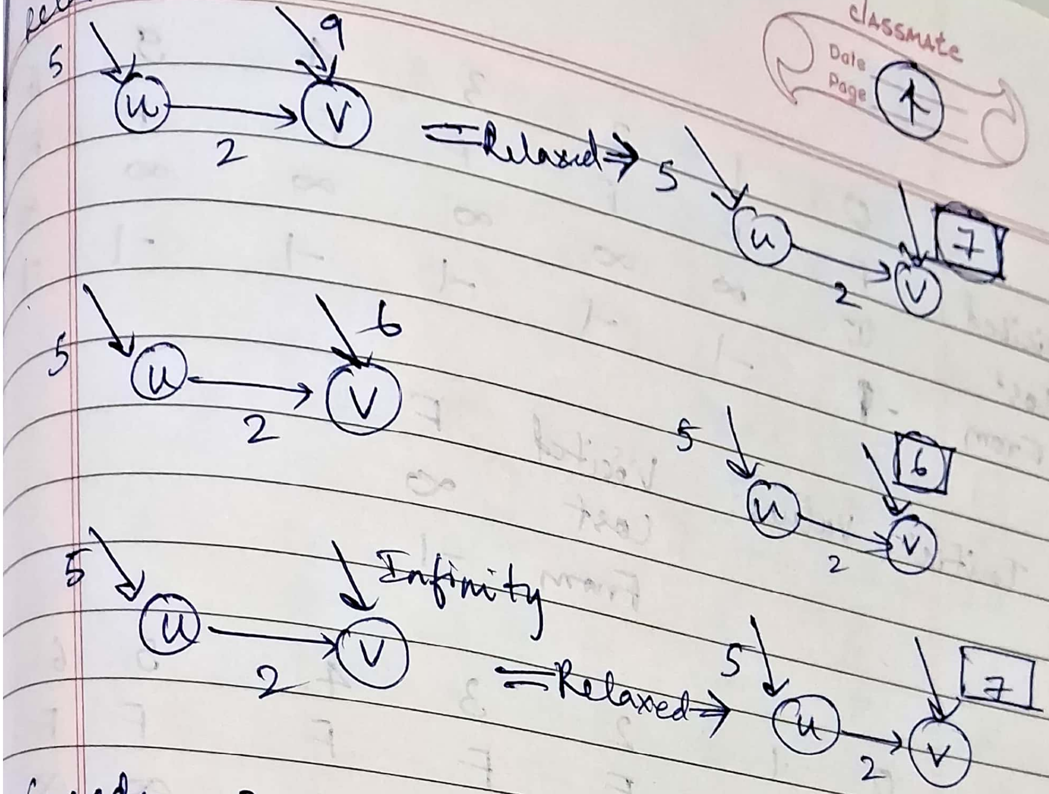
1 → 2 → 4 → 5, cost = 3

Graph Type	Longest Path	Shortest Path
Unweighted/ Constant weight	DFS	BFS
Weight ≥ 0 (+ve only)	Currently No Polynomial Time Algo exists NP - complete	Dijaktra's Algo / DP
Any weight (-ve, 0, +ve)	Currently No Polynomial Time Algo exists NP - complete	Bellman-Ford's Algo

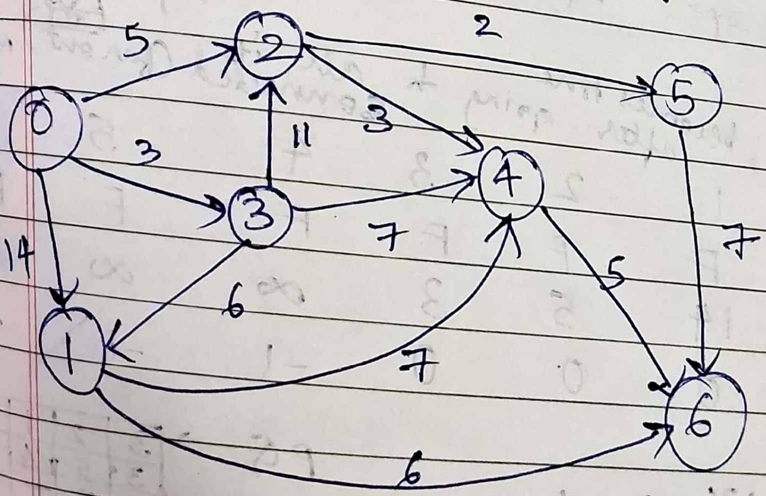
Dijaktra's Algorithm :

- Finds the shortest path from a vertex "s" to any other vertex "w"
- Graph G can ~~must~~ be directed/undirected & all edge weights must be ≥ 0
- Graph G does not need to be a DAG
- Graph G can have cycles.
- Greedy Algorithm
- Uses : BFS
  Min Heap (Priority Queue)
  Relaxation

Relaxation:



Greedy May not always work:



Local Greedy $\neq$ Global Greedy

$0 \xrightarrow{?} 6$ Shortest Path ?

Local Greedy: $0 \xrightarrow{3} 3 \xrightarrow{6} 1 \xrightarrow{6} 6$ Cost = 15

Global Greedy: $0 \xrightarrow{5} 2 \xrightarrow{3} 4 \xrightarrow{5} 6$ Cost = 13

	0	1	2	3	4	5	6
Visited	F	F	F	F	F	F	F
Cost	∞	∞	∞	∞	∞	∞	∞
From	-1	-1	-1	-1	-1	-1	-1

Initial Values: Visited F
Cost ∞
From -1

	0	1	2	3	4	5	6
Visited	F	F	F	F	F	F	F
Cost	0	∞	∞	∞	∞	∞	∞
From	0	-1	-1	-1	-1	-1	-1

PQ → ~~0~~

PQ →

0
0
F

0 → becomes true after going to all it connected (fan out) nodes

	0	1	2	3	4	5	6
Visited	T	F	F	F	F	F	F
Cost	0	14	5	3	∞	∞	∞
From	0	0	0	0	-1	-1	-1

PQ → ~~3~~, 2, 1

PQ →

3	2	1
3	5	14
T	T	T

	0	1	2	3	4	5	6
Visited	T	F	F	T	F	F	F
Cost	0	9	5	3	10	∞	∞
From	0	3	0	0	3	-1	-1

PQ → ~~2~~, 1, 4

PQ →

2	1	4	1
5	9	10	14
T	F	T	F

	0	1	2	3	4	5	6
Visited	T	F	T	T	F	F	F
Cost	0	9	5	3	8	7	∞
From	0	3	0	0	2	2	-1

PQ → ~~3~~, 4, 1

PQ →

5	4	1	4	1
7	8	9	10	14
T	T	T	F	F

	0	1	2	3	4	5	6
Visited	T	F	T	T	F	T	F
Cost	0	9	5	3	8	7	14
From	0	3	0	0	2	2	5

PQ → ~~4~~, 1, 6

PQ →

4	1	4	1	6
8	9	10	14	14
T	T	F	F	T

	0	1	2	3	4	5	6
Visited	T	F	T	T	T	T	F
Cost	0	9	5	3	8	7	13
From	0	3	0	0	2	2	4

PQ → ~~1~~, 6

PQ →

1	4	6	1	6
9	10	13	14	14
T	F	T	F	F

	0	1	2	3	4	5	6
Visited	T	T	T	T	T	T	F
Cost	0	9	5	3	8	7	13
From	0	3	0	0	2	2	4

PQ → ~~6~~

PQ →

6	1	6
13	14	14
T	F	F

	0	1	2	3	4	5	6
Visited	T	T	T	T	T	T	T
Cost	0	9	5	3	8	7	13
From	0	3	0	0	2	2	4

6 → 4 → 2 → 0 ← Cost = 13
13 8 5
5 3 5

0 → 2 → 4 → 6 Cost = 13

∴ Shortest Path with least Cost

5 → 2 → 0 ←
0 → 2 → 5 Cost = 7

4 → 2 → 0
0 → 2 → 4 Cost = 8

3 → 0
0 → 3 Cost = 3

2 → 0
0 → 2 Cost = 5

1 → 3 → 0
0 → 3 → 1 Cost = 9

Dijaktra's Algo gives the shortest path with least cost for any given node from a single starting node.

Space used   N: Vertex/Nodes
$4N$ ⟨ 3N: DS   E: Edges
       1N: PQ
$\approx \Theta(N)$

Work Done: $O(\log N * (N+E))$

$\approx O(N\log N)$ since we are using PQ (Min Heap) rather than Python List.

PQ Implementation:
Create a new DS that stores
- Node
- Cost
- Bool → represents if val is correct/relaxed (T) or redundant (F)

During relaxation Whenever we change cost in DS, mark the node in the PQ's Bool=F & ~~insert~~ a new ele in PQ with new cost & Bool =T

If we encounter a ele with Bool = F, when we pop from heap, ignore it

Dynamic Programming:

	0	1	2	3	4	5	6
Visited	T	F	F	F	F	F	F
Cost	0	-1	-1	-1	-1	-1	-1
From	0	-1	-1	-1	-1	-1	-1

DFS Order : 0 3 2 5 1 4 6

Using Timestamp (Topologically Sorted Order)

In this algo, we traverse DFS, then in DFS ~~order~~ we check :

fan ins → all fan ins ~~must be~~ which were visited

fan outs →

DFS → ~~0~~ ~~3~~ 2 5 1 4 6

	0	1	2	3	4	5	6
Visited	T	F	F	T	F	F	F
Cost	0	-1	-1	3	-1	-1	-1
From	0	-1	-1	0	-1	-1	-1

DFS → ~~0~~ ~~3~~ 2 5 1 4 6

	0	1	2	3	4	5	6
Visited	T	F	T	T	F	F	F
Cost	0	-1	5	3	-1	-1	-1
From	0	-1	0	0	-1	-1	-1

DFS → ~~0~~ ~~3~~ ~~2~~ 5 1 4 6

	0	1	2	3	4	5	6
Visited	T	F	T	T	F	T	F
Cost	0	-1	5	3	-1	7	-1
From	0	-1	0	0	-1	2	-1

DFS → ~~0~~ ~~3~~ ~~2~~ ~~5~~ 1 4 6

	0	1	2	3	4	5	6
Visited	T	T	T	T	F	T	F
Cost	0	9	5	3	-1	7	-1
From	0	3	0	0	-1	2	-1

DFS → ~~0~~ ~~3~~ ~~2~~ ~~5~~ ~~1~~ 4 6

	0	1	2	3	4	5	6
Visited	T	T	T	T	T	T	F
Cost	0	9	5	3	8	7	-1
From	0	3	0	0	2	2	-1

DFS → ~~0~~ ~~3~~ ~~2~~ ~~5~~ ~~1~~ ~~4~~ 6

	0	1	2	3	4	5	6
Visited	T	T	T	T	T	T	T
Cost	0	9	5	3	8	7	13
From	0	3	0	0	2	2	4

DFS → ~~0~~ ~~3~~ ~~2~~ ~~5~~ ~~1~~ ~~4~~ ~~6~~

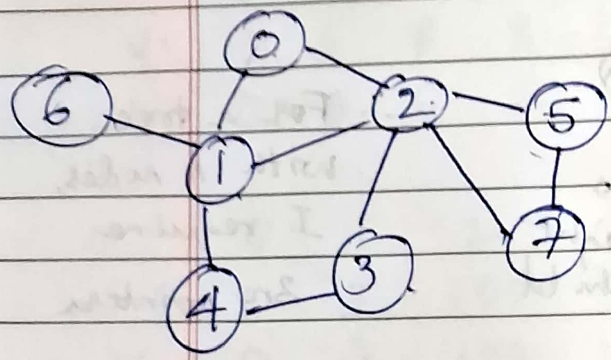
DP only works if graph has no loop ∴ DFS order is correct for DAG / No loop graph

Finding Shortest Paths

BFS Undirected Graph

· Constant Weight

0 → 7 ?



	0	1	2	3	4	5	6	7
Cost	0	-1	-1	-1	-1	-1	-1	-1
From	0	-1	-1	-1	-1	-1	-1	-1

q → ~~0~~

	0	1	2	3	4	5	6	7
Cost	0	1	1	-1	-1	-1	-1	-1
From	0	0	0	-1	-1	-1	-1	-1

q → ~~1~~ 2

	0	1	2	3	4	5	6	7
Cost	0	1	1	-1	2	-1	2	-1
From	0	0	0	-1	1	-1	1	-1

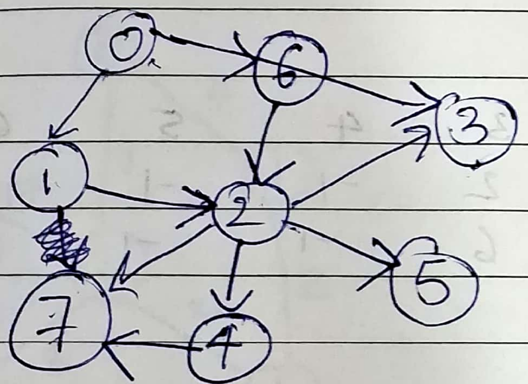
q → ~~2~~ 6 4

	0	1	2	3	4	5	6	7
Cost	0	1	1	2	2	2	2	2
From	0	0	0	2	1	1	1	1

q → ~~6~~ ~~4~~ ~~3~~ ~~5~~ ~~7~~

Cost = 2
Elements = { 7, 2, 0 }
Order rev ⇒ $0 \xrightarrow[1]{} 2 \xrightarrow[1]{} 7$

$\boxed{\text{BFS Directed Graph}}$: Constant Weight



0 → 7 ?

	0	1	2	3	4	5	6	7
Cost	0	-1	-1	-1	-1	-1	-1	-1
From	0	-1	-1	-1	-1	-1	-1	-1

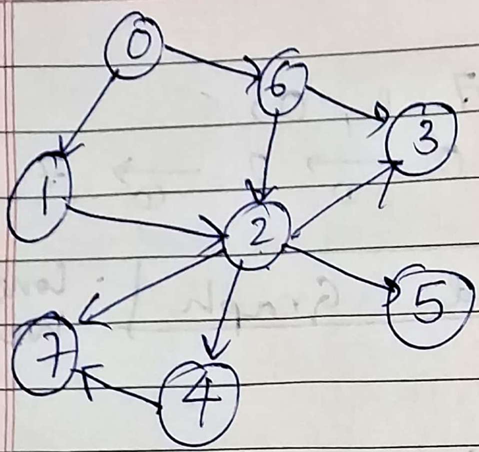
q → ~~0~~

	0	1	2	3	4	5	6	7
Cost	0	1	-1	-1	-1	-1	1	-1
From	0	0	-1	-1	-1	-1	0	-1

q → ~~6~~ 1

	0	1	2	3	4	5	6	7
Cost	0	1	2	2	-1	-1	1	-1
From	0	0	6	6	-1	-1	0	-1

q → ~~1~~ 3 2



	0	1	2	3	4	5	6	7
Cost	0	1	2	2	-1	-1	1	-1
From	0	0	6	6	-1	-1	0	-1

q → ~~3~~ 2

	0	1	2	3	4	5	6	7
Cost	0	1	2	2	-1	-1	1	-1
From	0	0	6	6	-1	-1	0	-1

q → ~~2~~

	0	1	2	3	4	5	6	7
Cost	0	1	2	2	3	3	1	3
From	0	0	6	6	2	2	0	2

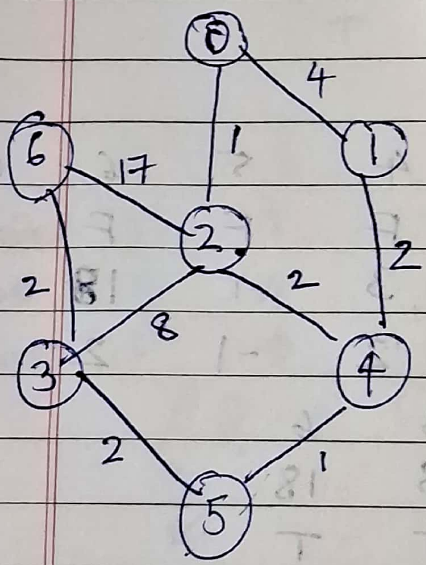
q → ~~4~~ 5 7

	0	1	2	3	4	5	6	7
Cost	0	1	2	2	3	3	1	3
From	0	0	6	6	2	2	0	2

q → ~~5~~ ~~7~~

Cost = 3
Elements = { 7, 2, 6, 0 }
Order rev => $0 \xrightarrow{1} 6 \xrightarrow{1} 2 \xrightarrow{1} 7$

$\boxed{\text{Dijakstra's Undirected Graph}}$. Any +ve weight



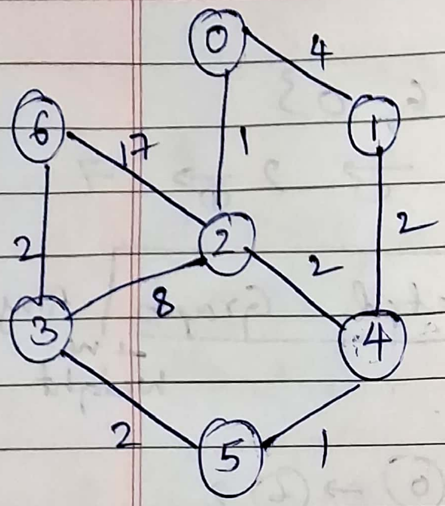
$0 \rightarrow 6$?

	0	1	2	3	4	5	6
Visited	F	F	F	F	F	F	F
Cost	0	∞	∞	∞	∞	∞	∞
From	-1	-1	-1	-1	-1	-1	-1

PQ → ~~0~~
0
T

	0	1	2	3	4	5	6
Visited	T	F	F	F	F	F	F
Cost	0	4	1	∞	∞	∞	∞
From	0	0	0	-1	-1	-1	-1

PQ → 2 1
1 4
T T



PQ → ~~2~~ 1
1 4
T T

	0	1	2	3	4	5	6
Visited	~~T~~	~~F~~	T	F	F	F	F
cost	0	4	1	8	3	-1	18
From	0	0	0	2	2	-1	2

PQ →	~~4~~	1	3	6
	3	4	8	18
	T	T	T	T

	0	1	2	3	4	5	6
Visited	T	F	T	F	T	F	F
Cost	0	4	1	8	3	4	18
From	0	0	0	2	2	4	2

PQ →	~~1~~	5	3	6
	4	4	8	18
	T	T	T	T

	0	1	2	3	4	5	6
Visited	T	T	T	F	T	F	F
cost	0	4	1	8	3	4	18
From	0	0	0	2	2	4	2

PQ →	~~5~~	3	6
	4	8	18
	T	T	T

classmate

Page 6

	0	1	2	3	4	5	6
Visited	T	T	T	~~F~~	T	T	F
Cost	0	4	1	~~6~~	3	4	18
From	0	0	0	5	2	4	2

PQ →

~~3~~	3	6
6	8	18
T	F	T

	0	1	2	3	4	5	6
Visited	T	T	T	T	T	T	F
Cost	0	4	1	6	3	4	8
From	0	0	0	5	2	4	5

PQ →

~~3~~	~~6~~	~~6~~
8	8	18
F	T	F

	0	1	2	3	4	5	6
Visited	T	T	T	T	T	T	T
Cost	0	4	1	6	3	4	8
From	0	0	0	5	2	4	5

0 → 6 Cost : 8

Elements : {6, 5, 4, 2, 0}

rev = {0, 2, 4, 5, 6}

0 → 1 Cost : 4 Elements order = {0, 1}

0 → 2 Cost : 1 Elements order = {0, 2}

0 → 3 Cost : 6 Elements order = {0, 2, 4, 5, 3}

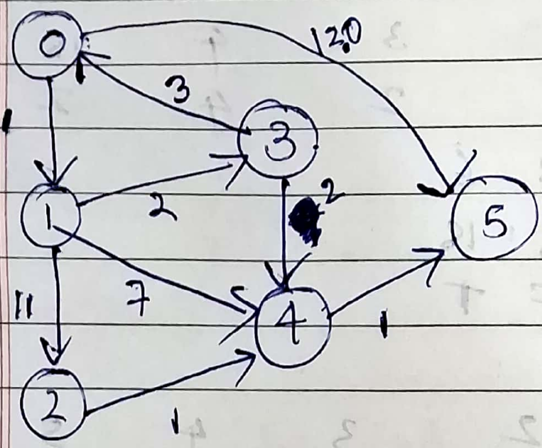
0 → 4 Cost : 3 Elements order = {0, 2, 4}

0 → 5 Cost : 4 Elements order = {0, 2, 4, 5}

Dijaktra's Directed Graph

Any +ve weight

0 → 5 ?



	0	1	2	3	4	5
Visited	F	F	F	F	F	F
cost	∞	∞	∞	∞	∞	∞
From	-1	-1	-1	-1	-1	-1

PQ → ~~0~~
0
T

	0	1	2	3	4	5
Visited	T	F	F	F	F	F
cost	0	1	∞	∞	∞	12
From	0	0	-1	-1	-1	0

PQ → ~~1~~ 5
1 12
T T

	0	1	2	3	4	5
Visited	T	T	F	F	F	F
cost	0	1	12	3	7	12
From	0	0	1	1	1	0

PQ → ~~3~~ 4 2 5
3 7 12 12
T T T T

	0	1	2	3	4	5
Visited	T	T	F	T	F	F
cost	0	1	12	3	5	12
From	0	0	1	1	3	0

PQ → ~~4~~ 4 2 5
5 7 12 12
T F T T

	0	1	2	3	4	5
Visited	T	T	F	T	T	F
cost	0	1	12	3	5	6
From	0	0	1	1	3	4

PQ → ~~5~~ 4 2 5
6 7 12 12
T F T F

	0	1	2	3	4	5
Visited	T	T	F	T	T	T
cost	0	1	12	3	5	6
From	0	0	1	1	3	4

PQ → ~~4~~ ~~2~~ ~~5~~
7 12 12
F T F

	0	1	2	3	4	5
Visited	T	T	T	T	T	T
cost	0	1	12	3	5	6
From	0	0	1	1	3	4

0 → 5 Cost: 6

Elements: {5, 4, 3, 1, 0}

rev: {0, 1, 3, 4, 5}

0 → 1 Cost: 1 Elements: {0, 1}

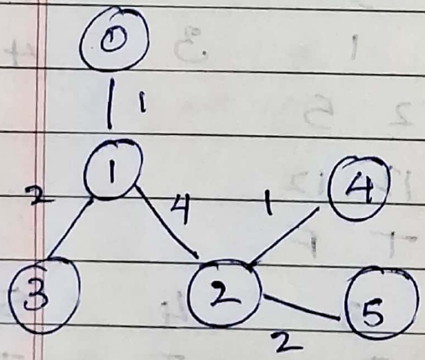
0 → 2 Cost: 12 Elements order: {0, 1, 2}

0 → 3 Cost: 3 Elements order: {0, 1, 3}

0 → 4 Cost: 5 Elements order: {0, 1, 3, 4}

Dynamic Programming
Undirected Graph

- Any the weight
- DAG



0 → 5 ?

	0	1	2	3	4	5
in	0	0	0	0	0	0
out	0	0	0	0	0	0

unvisited set → ~~0~~ 1 2 3 ~~4~~ 5

dfs order →

	0	1	2	3	4	5
in	1	2	0	0	0	0
out	0	0	0	0	0	0

unvisited set → ~~0~~ ~~1~~ 2 3 4 5

dfs order →

	0	1	2	3	4	5
in	1	2	0	3	0	0
out	0	0	0	0	0	0

unvisited set → ~~0~~ ~~1~~ 2 ~~3~~ 4 5

dfs order → ~~0~~

	0	1	2	3	4	5
in	1	2	0	3	0	0
out	0	0	0	4	0	0

unvisited set → ~~0~~ ~~1~~ 2 ~~3~~ 4 5

dfs order → 3

	0	1	2	3	4	5
in	1	2	5	3	0	0
out	0	0	0	4	0	0

unvisited set → ~~0~~ ~~1~~ ~~2~~ ~~3~~ 4 5

dfs order → 3

	0	1	2	3	4	5
in	1	2	5	3	6	0
out	0	0	0	4	0	0

unvisited set → ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5

dfs order → 3

	0	1	2	3	4	5
in	1	2	5	3	6	0
out	0	0	0	4	7	0

unvisited set → ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5

dfs order → 3 4

	0	1	2	3	4	5
in	1	2	5	3	6	8
out	0	0	0	4	7	0

unvisited set → ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~

dfs order → 3 4

	0	1	2	3	4	5
in	1	2	5	3	6	8
out	0	0	0	4	7	9

unvisited set →

dfs order → 3 4 5

	0	1	2	3	4	5
in	1	2	5	3	6	8
out	0	0	10	4	7	9

unvisited set →

dfs order → 3 4 5 2

	0	1	2	3	4	5
in	1	2	5	3	6	8
out	0	11	10	4	7	9

unvisited set →

dfs order → 3 4 5 2 1

	0	1	2	3	4	5
in	1	2	5	3	6	8
out	12	11	10	4	7	9

unvisited set →

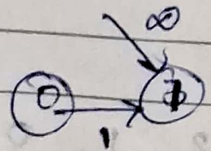
dfs order → 3 4 5 2 1 0

(iv)

dfs order → 0 1 2 5 4 3

dfs → ~~0~~ 1 2 5 4 3

	0	1	2	3	4	5
Visited	T	F	F	F	F	F
Cost	0	∞	∞	∞	∞	∞
From	0	-1	-1	-1	-1	-1



dfs → ~~0~~ ~~1~~ 2 5 4 3

	0	1	2	3	4	5
Visited	T	T	F	F	F	F
Cost	0	1	∞	∞	∞	∞
From	0	0	-1	-1	-1	-1

dfs → ~~0~~ ~~1~~ ~~2~~ 5 4 3

	0	1	2	3	4	5
Visited	T	T	T	F	F	F
Cost	0	1	5	∞	∞	∞
From	0	0	1	-1	-1	-1

dfs → ~~0~~ ~~1~~ ~~2~~ ~~5~~ 4 3

	0	1	2	3	4	5
Visited	T	T	T	F	F	T
Cost	0	1	5	∞	∞	7
From	0	0	1	-1	-1	2

dfs → ~~0~~ ~~1~~ ~~2~~ ~~5~~ ~~4~~ 3

	0	1	2	3	4	5
Visited	T	T	T	F	T	T
Cost	0	1	5	∞	6	7
From	0	0	1	-1	2	2

dfs → ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~

	0	1	2	3	4	5
Visited	T	T	T	T	T	T
Cost	0	1	5	3	6	7
From	0	0	1	1	2	2

0 → 5 Cost : 7

Elements Order : {5, 2, 1, 0}

rev : {0, 1, 2, 5}

0 → 1 Cost : 1 Elements Order : {0, 1}

0 → 2 Cost : 5 Elements Order : {0, 1, 2}

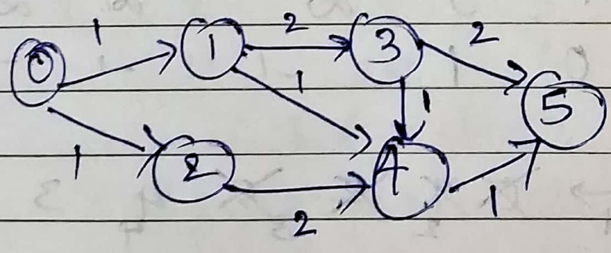
0 → 3 Cost : 3 Elements Order : {0, 1, 3}

0 → 4 Cost : 6 Elements Order : {0, 1, 2, 4}

$\boxed{\text{Dynamic Programming Directed Graph}}$

· Any the weight

· DAG



0 → 5 ?

	0	1	2	3	4	5
in	0	0	0	0	0	0
out	0	0	0	0	0	0

unvisited set → ~~0~~ 1 2 3 4 5

dfs order →

unvisited set → ~~0~~ ~~1~~ 2 3 4 5
dfs order →

	0	1	2	3	4	5
in	1	2	0	0	0	0
out	0	0	0	0	0	0

unvisited set → ~~0~~ ~~1~~ 2 ~~3~~ 4 5
dfs order →

	0	1	2	3	4	5
in	1	2	0	3	0	0
out	0	0	0	0	0	0

unvisited set → ~~0~~ ~~1~~ 2 ~~3~~ ~~4~~ 5
dfs order →

	0	1	2	3	4	5
in	1	2	0	3	4	0
out	0	0	0	0	0	0

unvisited set → ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~
dfs order →

	0	1	2	3	4	5
in	1	2	0	3	4	5
out	0	0	0	0	0	0

unvisited set → ~~0~~ ~~1~~ 2 ~~3~~ ~~4~~ ~~5~~
dfs order → 5

	0	1	2	3	4	5
in	1	2	0	3	4	5
out	0	0	0	0	0	6

unvisited set → ~~0~~ ~~1~~ 2 ~~3~~ ~~4~~ ~~5~~
dfs order → 5 4

	0	1	2	3	4	5
in	1	2	0	3	4	5
out	0	0	0	0	7	6

dfs order → 5 4 3
unvisited set → ~~0~~ ~~1~~ 2 ~~3~~ ~~4~~ ~~5~~

	0	1	2	3	4	5
in	1	2	0	3	4	5
out	0	0	0	8	7	6

unvisited set → ~~0~~ ~~1~~ 2 ~~3~~ ~~4~~ ~~5~~
dfs order → 5 4 3 1

	0	1	2	3	4	5
in	1	2	0	3	4	5
out	0	9	0	8	7	6

unvisited set → ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~
dfs order → 5 4 3 1 ~~2~~

	0	1	2	3	4	5
in	1	2	10	3	4	5
out	0	9	0	8	7	6

unvisited set → ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~
dfs order → 5 4 3 1 2

	0	1	2	3	4	5
in	1	2	10	3	4	5
out	0	9	11	8	7	6

unvisited set → 0 1 2 3 4 5

dfs order → 5 4 3 1 2 0

	0	1	2	3	4	5
in	1	2	10	3	4	5
out	12	9	11	8	7	6

new

dfs order → ~~0~~ 2 1 3 4 5

	0	1	2	3	4	5
Visited	T	F	F	F	F	F
cost	0	∞	∞	∞	∞	∞
from	0	-1	-1	-1	-1	-1

dfs order → ~~0~~ ~~2~~ 1 3 4 5

	0	1	2	3	4	5
Visited	T	F	T	F	F	F
cost	0	∞	1	∞	∞	∞
From	0	-1	1	-1	-1	-1

dfs order → ~~0~~ ~~2~~ ~~1~~ 3 4 5

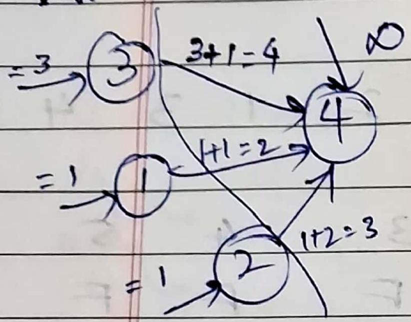
	0	1	2	3	4	5
Visited	T	T	T	F	F	F
cost	0	1	1	∞	∞	∞
From	0	1	1	-1	-1	-1

dfs order → ~~0~~ ~~2~~ ~~1~~ ~~3~~ 4 5

	0	1	2	3	4	5
Visited	T	T	T	T	F	F
cost	0	1	1	3	∞	∞
From	0	1	1	1	-1	-1

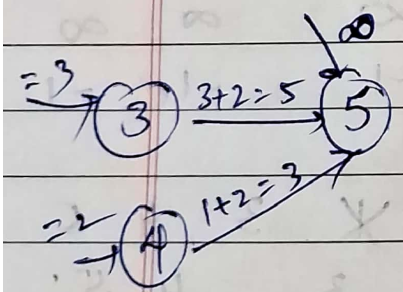
dfs order → ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5

	0	1	2	3	4	5
Visited	T	T	T	T	T	F
cost	0	1	1	3	2	∞
From	0	1	1	1	1	-1



dfs order → ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~

	0	1	2	3	4	5
Visited	T	T	T	T	T	T
Cost	0	1	1	3	2	3
From	0	1	1	1	1	4



0 → 5 ? Cost = 3
Elements order : {5, 4, 1, 0}
~~Rev~~ : {0, 1, 4, 5}

0 → 1 cost: 1 Elements order : {0, 1}
0 → 2 cost: 1 elements order : {0, 2}
0 → 3 cost: 3 elements order : {0, 1, 3}
0 → 4 cost: 2 Elements order : {0, 1, 4}