Network Algorithm UE709

Rabih AMHAZ

Head of Digital department at ECAM Strasbourg-Europe
Head of Professional Master's Degree Curriculum
Industry 4.0 and IoT
Researcher in AI and Blockchain at ICube Laboratory, CSTB team

February 14, 2024

Shortest Path utility

- **Routing in Computer Networks**: Shortest Path algorithm can be used in distance-vector routing protocols, such as the Routing Information Protocol (RIP). Why? to determine the shortest path between routers in a network, considering factors like network congestion and link quality.
- **Flight Path Planning**: Airlines use the algorithm. The edges of the graph represent flight connections between airports, and the edge weights can represent factors such as distance, flight duration, and fuel consumption.
- **Telecommunication Networks**: to optimize the routing of data packets in their networks. Algorithm find the shortest path for transmitting data packets while considering factors like latency, bandwidth, and cost.
- **Traffic Management Systems**: in cities or transportation networks to optimize traffic flow.

- **Routing in Computer Networks**: Shortest Path algorithm can be used in distance-vector routing protocols, such as the Routing Information Protocol (RIP). Why? to determine the shortest path between routers in a network, considering factors like network congestion and link quality.
- **Flight Path Planning**: Airlines use the algorithm. The edges of the graph represent flight connections between airports, and the edge weights can represent factors such as distance, flight duration, and fuel consumption.
- **Telecommunication Networks**: to optimize the routing of data packets in their networks. Algorithm find the shortest path for transmitting data packets while considering factors like latency, bandwidth, and cost.
- **Traffic Management Systems**: in cities or transportation networks to optimize traffic flow.

- **Routing in Computer Networks**: Shortest Path algorithm can be used in distance-vector routing protocols, such as the Routing Information Protocol (RIP). Why? to determine the shortest path between routers in a network, considering factors like network congestion and link quality.
- **Flight Path Planning**: Airlines use the algorithm. The edges of the graph represent flight connections between airports, and the edge weights can represent factors such as distance, flight duration, and fuel consumption.
- **Telecommunication Networks**: to optimize the routing of data packets in their networks. Algorithm find the shortest path for transmitting data packets while considering factors like latency, bandwidth, and cost.
- **Traffic Management Systems**: in cities or transportation networks to optimize traffic flow.

- **Routing in Computer Networks**: Shortest Path algorithm can be used in distance-vector routing protocols, such as the Routing Information Protocol (RIP). Why? to determine the shortest path between routers in a network, considering factors like network congestion and link quality.
- **Flight Path Planning**: Airlines use the algorithm. The edges of the graph represent flight connections between airports, and the edge weights can represent factors such as distance, flight duration, and fuel consumption.
- **Telecommunication Networks**: to optimize the routing of data packets in their networks. Algorithm find the shortest path for transmitting data packets while considering factors like latency, bandwidth, and cost.
- **Traffic Management Systems**: in cities or transportation networks to optimize traffic flow.

DAG

Recall that a Directed Acyclic Graph (DAG) is a graph with directed edges and no cycles. By definition this means all **trees** are automatically DAGs since they do not contains cycles.

The Single Source Shortest Path (SSSP) problem can be solved efficiently on a DAG in **O(V+E)** time. This is due to the fact that the nodes can be ordered in a topological ordering via topsort and processed sequentially.

Shortest Path : Question

Imaging that you are driving and to take each road/edge you need to pay some manats (some money), you have infinity of gaz to use but you need to pay the less manats you can.

Edges weight : Edge cost, means taking the edge will cost for us. You can define the cost that you like. ex: Speed, money, gray level (in images) etc.

Shortest Path : Question becomes

We are searching for the shortest path.

Shortest Path : Question

Imaging that you are driving and to take each road/edge you need to pay some manats (some money), you have infinity of gaz to use but you need to pay the less manats you can.

Edges weight : Edge cost, means taking the edge will cost for us.

You can define the cost that you like. ex: Speed, money, gray level (in images) etc.

Shortest Path : Question becomes

We are searching for the shortest path.

Shortest Path : Question

Imaging that you are driving and to take each road/edge you need to pay some manats (some money), you have infinity of gaz to use but you need to pay the less manats you can.

Edges weight : Edge cost, means taking the edge will cost for us.
You can define the cost that you like. ex: Speed, money, gray level (in images) etc.

Shortest Path : Question becomes

We are searching for the shortest path.

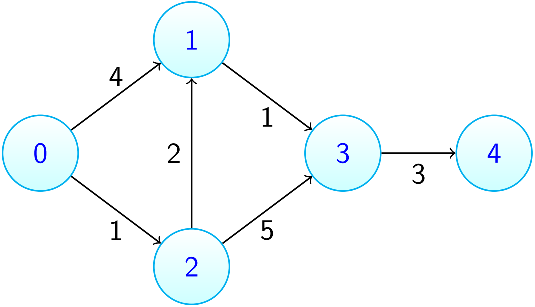
Shortest Path : Question

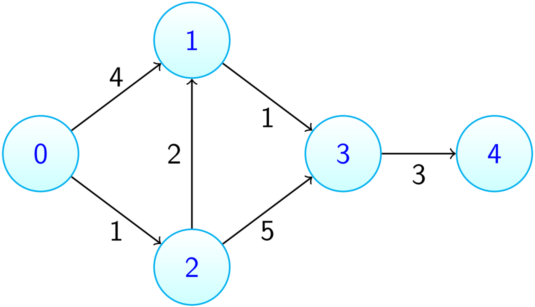
Imaging that you are driving and to take each road/edge you need to pay some manats (some money), you have infinity of gaz to use but you need to pay the less manats you can.

Edges weight : Edge cost, means taking the edge will cost for us.
You can define the cost that you like. ex: Speed, money, gray level (in images) etc.

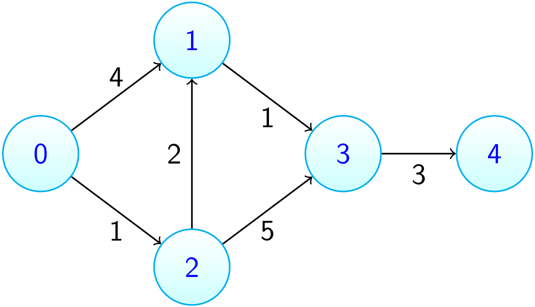
Shortest Path : Question becomes

We are searching for the shortest path.

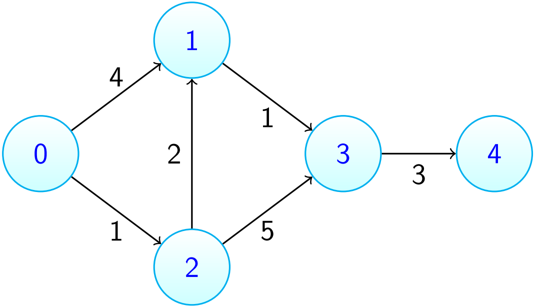

0
1
2
3
4
4
1
2
1
5
3



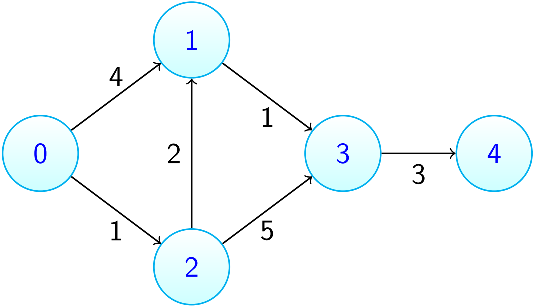
My Position	(index,dist)
	(0,0)
	(1,4)
	(2,1)
	(1,3)
	(3,6)
	(3,4)
	(4,7)



My Position	(index,dist)
	(0,0)
	(1,4)
	(2,1)
	(1,3)
	(3,6)
	(3,4)
	(4,7)

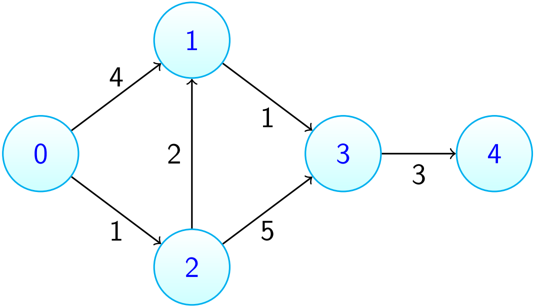


My Position	(index,dist)
	(0,0)
	(1,4)
	(2,1)
	(1,3)
	(3,6)
	(3,4)
	(4,7)

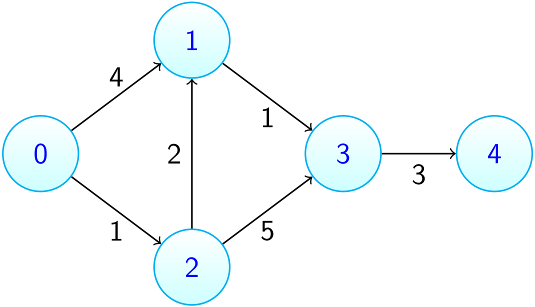


My Position	(index,dist)
	(0,0)
	(1,4)
	(2,1)
	(1,3)
	(3,6)
	(3,4)
	(4,7)

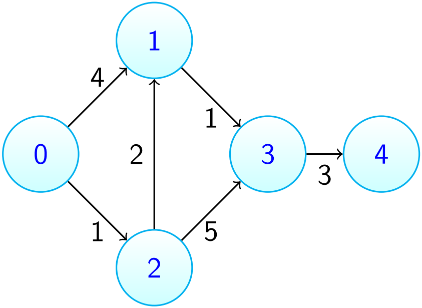
Shortest Path: Dijkstra's



My Position	(index,dist)
	(0,0)
	(1,4)
	(2,1)
	(1,3)
	(3,6)
	(3,4)
	(4,7)

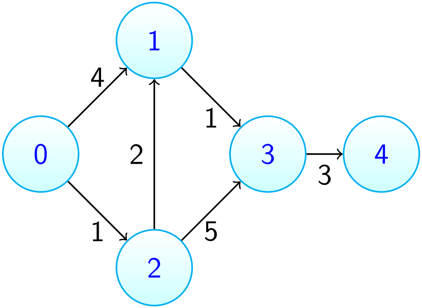


My Position	(index,dist)
	(0,0)
	(1,4)
	(2,1)
	(1,3)
	(3,6)
	(3,4)
	(4,7)



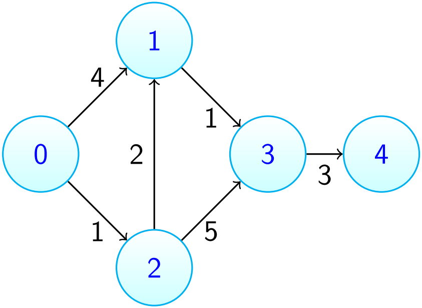
-	0	1	2	3	4
	∞	∞	∞	∞	∞
from 0	0	4	1	∞	∞
from 2	0	3	1	6	∞
from 1	0	3	1	4	∞
from 3	0	3	1	4	7

Shortest Path: $(0, 2, 1, 3, 4)$ with cost 7



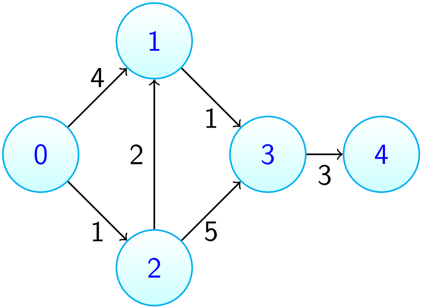
-	0	1	2	3	4
	∞	∞	∞	∞	∞
from 0	0	4	1	∞	∞
from 2	0	3	1	6	∞
from 1	0	3	1	4	∞
from 3	0	3	1	4	7

Shortest Path: $(0, 2, 1, 3, 4)$ with cost 7



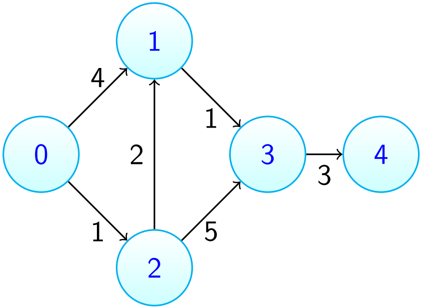
-	0	1	2	3	4
	∞	∞	∞	∞	∞
from 0	0	4	1	∞	∞
from 2	0	3	1	6	∞
from 1	0	3	1	4	∞
from 3	0	3	1	4	7

Shortest Path: $(0, 2, 1, 3, 4)$ with cost 7



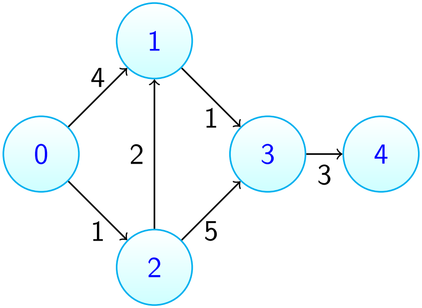
-	0	1	2	3	4
	∞	∞	∞	∞	∞
from 0	0	4	1	∞	∞
from 2	0	3	1	6	∞
from 1	0	3	1	4	∞
from 3	0	3	1	4	7

Shortest Path: $(0, 2, 1, 3, 4)$ with cost 7



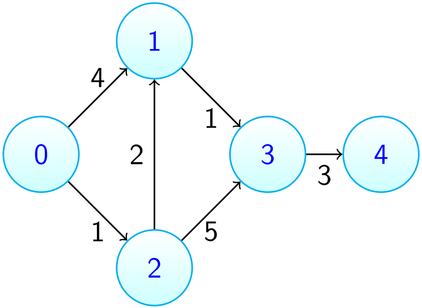
-	0	1	2	3	4
	∞	∞	∞	∞	∞
from 0	0	4	1	∞	∞
from 2	0	3	1	6	∞
from 1	0	3	1	4	∞
from 3	0	3	1	4	7

Shortest Path: $(0, 2, 1, 3, 4)$ with cost 7



-	0	1	2	3	4
	∞	∞	∞	∞	∞
from 0	0	4	1	∞	∞
from 2	0	3	1	6	∞
from 1	0	3	1	4	∞
from 3	0	3	1	4	7

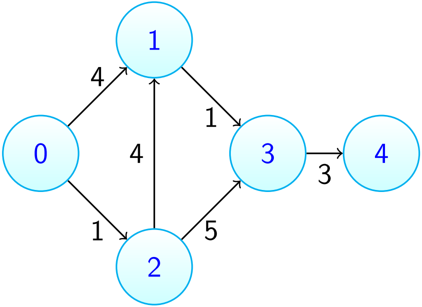
Shortest Path: $(0, 2, 1, 3, 4)$ with cost 7



-	0	1	2	3	4
	∞	∞	∞	∞	∞
from 0	0	4	1	∞	∞
from 2	0	3	1	6	∞
from 1	0	3	1	4	∞
from 3	0	3	1	4	7

Shortest Path: $(0, 2, 1, 3, 4)$ with cost 7

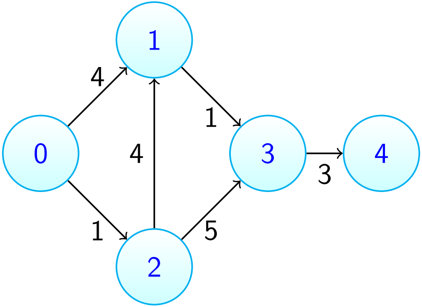
What If we change the weight of edge between 2 and 1 from 2 to 4 ?



-	0	1	2	3	4
	∞	∞	∞	∞	∞
from 0	0	4	1	∞	∞
from 2	0	4	1	6	∞
from 1	0	4	1	5	∞
from 3	0	4	1	5	8

Shortest Path: with cost

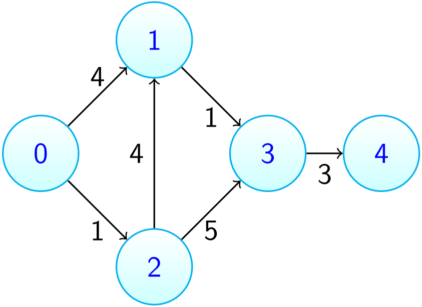
What If we change the weight of edge between 2 and 1 from 2 to 4 ?



-	0	1	2	3	4
	∞	∞	∞	∞	∞
from 0	0	4	1	∞	∞
from 2	0	4	1	6	∞
from 1	0	4	1	5	∞
from 3	0	4	1	5	8

Shortest Path: with cost

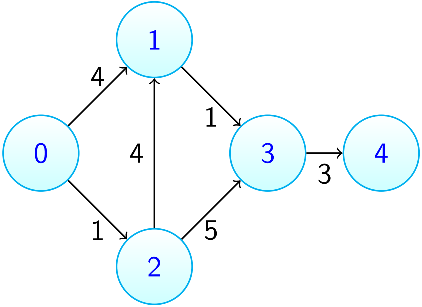
What If we change the weight of edge between 2 and 1 from 2 to 4 ?



-	0	1	2	3	4
	∞	∞	∞	∞	∞
from 0	0	4	1	∞	∞
from 2	0	4	1	6	∞
from 1	0	4	1	5	∞
from 3	0	4	1	5	8

Shortest Path: with cost

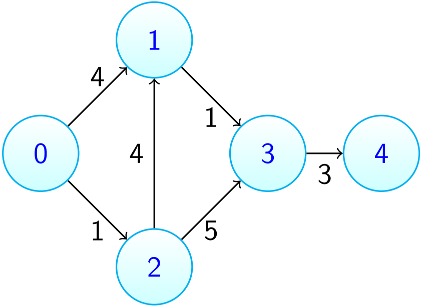
What If we change the weight of edge between 2 and 1 from 2 to 4 ?



-	0	1	2	3	4
	∞	∞	∞	∞	∞
from 0	0	4	1	∞	∞
from 2	0	4	1	6	∞
from 1	0	4	1	5	∞
from 3	0	4	1	5	8

Shortest Path: with cost

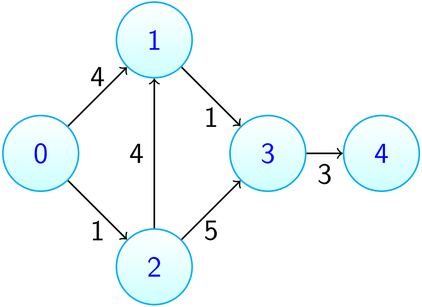
What If we change the weight of edge between 2 and 1 from 2 to 4 ?



-	0	1	2	3	4
	∞	∞	∞	∞	∞
from 0	0	4	1	∞	∞
from 2	0	4	1	6	∞
from 1	0	4	1	5	∞
from 3	0	4	1	5	8

Shortest Path: with cost

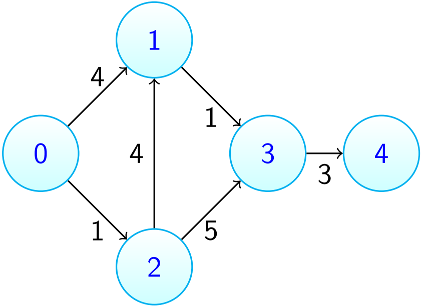
What If we change the weight of edge between 2 and 1 from 2 to 4 ?



-	0	1	2	3	4
	∞	∞	∞	∞	∞
from 0	0	4	1	∞	∞
from 2	0	4	1	6	∞
from 1	0	4	1	5	∞
from 3	0	4	1	5	8

Shortest Path: with cost

What If we change the weight of edge between 2 and 1 from 2 to 4 ?



-	0	1	2	3	4
	∞	∞	∞	∞	∞
from 0	0	4	1	∞	∞
from 2	0	4	1	6	∞
from 1	0	4	1	5	∞
from 3	0	4	1	5	8

Shortest Path: with cost

```
function dijkstra(g,n,s):
    vis=[false, .., false]
    #prev=[null, .., null]
    dist=[inf, .., inf]
    tdist=0
    pq=empty priority queue
    pq.insert((s,0))
    while pq.size()!=0:
        index,minValue=pg.poll()
        vis[index] = true
        #if dist[index]<minValue: continue
        for (edge : g[index]):
            if visited[edge.to]: continue
            newDist = dist[index] + edge.cost
            if newDist<dist[edge.to]:
                #prev[edge.to]=index
                dist[edge.to]=newDist
                pq.insert((edge.to,newDist))
    return dist
    #return (dist,prev)
```

```
function findShortestPath(g,n,s,e):
    dist,prev=dijkstra(g,n,s)
    path=[]
    if(dis[e]==inf) return path
    for (at=e;at!=null;at=prev[at]):
        path.add(at)
    path.reverse()
    return path
```

- Single source shortest path (SSSP) algorithm. This means it can find the shortest path from one node to any other node.
- Bellman-Ford is not ideal for most SSSP problems because complexity of $O(EV)$.
- Dijkstra's is much faster with $O(Vlog(V))$ when using a binary heap priority queue (fibonacci heap).
- But Dijkstra's can fail when graph have negative edge weights.
- Bellman-Ford can be used to detect **negative cycles** and **determine where they occurs.**
- Useful in many types of applications finance when performing an **arbitrage** between two or more markets.

- Single source shortest path (SSSP) algorithm. This means it can find the shortest path from one node to any other node.
- Bellman-Ford is not ideal for most SSSP problems because complexity of $O(EV)$.
- Dijkstra's is much faster with $O(Vlog(V))$ when using a binary heap priority queue (fibonacci heap).
- But Dijkstra's can fail when graph have negative edge weights.
- Bellman-Ford can be used to detect **negative cycles** and **determine where they occurs.**
- Useful in many types of applications finance when performing an **arbitrage** between two or more markets.

- Single source shortest path (SSSP) algorithm. This means it can find the shortest path from one node to any other node.
- Bellman-Ford is not ideal for most SSSP problems because complexity of $O(EV)$.
- Dijkstra's is much faster with $O(Vlog(V))$ when using a binary heap priority queue (fibonacci heap).
- But Dijkstra's can fail when graph have negative edge weights.
- Bellman-Ford can be used to detect **negative cycles** and **determine where they occurs.**
- Useful in many types of applications finance when performing an **arbitrage** between two or more markets.

- Single source shortest path (SSSP) algorithm. This means it can find the shortest path from one node to any other node.
- Bellman-Ford is not ideal for most SSSP problems because complexity of $O(EV)$.
- Dijkstra's is much faster with $O(Vlog(V))$ when using a binary heap priority queue (fibonacci heap).
- But Dijkstra's can fail when graph have negative edge weights.
- Bellman-Ford can be used to detect **negative cycles** and **determine where they occurs.**
- Useful in many types of applications finance when performing an **arbitrage** between two or more markets.

- Single source shortest path (SSSP) algorithm. This means it can find the shortest path from one node to any other node.
- Bellman-Ford is not ideal for most SSSP problems because complexity of $O(EV)$.
- Dijkstra's is much faster with $O(Vlog(V))$ when using a binary heap priority queue (fibonacci heap).
- But Dijkstra's can fail when graph have negative edge weights.
- Bellman-Ford can be used to detect **negative cycles** and **determine where they occurs.**
- Useful in many types of applications finance when performing an **arbitrage** between two or more markets.

Bellman-Ford Algorithm

D is the array of size V that track best distance from the source S

1. Set every entry in D to $+\infty$
2. Set D[S]=0
3. Relax each edge V-1 times.
   Update the D array.

```
for (i=0;i<v-1;i++):
        for edge in graph.edges:
        #Relax edge (update D)
            if D[edge.from]+edge.cost<D[edge.to]:
                D[edge.to]=D[edge.from]+edge.cost
```

Bellman-Ford Algorithm

D is the array of size V that track best distance from the source S

1. Set every entry in D to $+\infty$
2. Set D[S]=0
3. Relax each edge V-1 times.
   Update the D array.

```
for (i=0;i<v-1;i++):
        for edge in graph.edges:
        #Relax edge (update D)
            if D[edge.from]+edge.cost<D[edge.to]:
                D[edge.to]=D[edge.from]+edge.cost
```

Bellman-Ford Algorithm

D is the array of size V that track best distance from the source S

1. Set every entry in D to $+\infty$
2. Set D[S]=0
3. Relax each edge V-1 times.
   Update the D array.

```
for (i=0;i<v-1;i++):
        for edge in graph.edges:
        #Relax edge (update D)
            if D[edge.from]+edge.cost<D[edge.to]:
                D[edge.to]=D[edge.from]+edge.cost
```

Bellman-Ford Algorithm

D is the array of size V that track best distance from the source S

1. Set every entry in D to $+\infty$
2. Set D[S]=0
3. Relax each edge V-1 times.
   Update the D array.

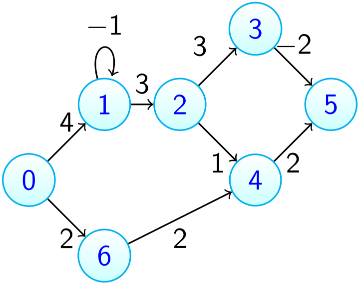
```
for (i=0;i<v-1;i++):
        for edge in graph.edges:
        #Relax edge (update D)
            if D[edge.from]+edge.cost<D[edge.to]:
                D[edge.to]=D[edge.from]+edge.cost
```

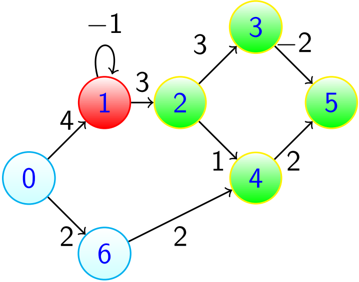
To detect the negative cycles

```
#Repeat to find nodes caught in a negative cycle.
for (i=0;i<v-1;i++):
    for edge in graph.edges:
        if(D[edge.from]+edge.cost<D[edge.to]):
            D[edge.to]= - inf
```

To detect the negative cycles

```
#Repeat to find nodes caught in a negative cycle.
for (i=0;i<v-1;i++):
    for edge in graph.edges:
        if(D[edge.from]+edge.cost<D[edge.to]):
            D[edge.to]= - inf
```

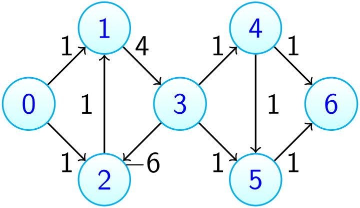

−1
3
3
−2
4
1
2
2
2
0
1
2
3
4
5
6

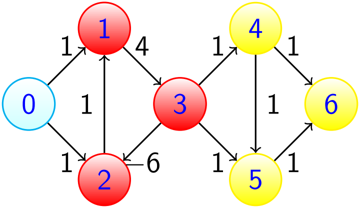


Invovled directly in a negative cycle (cost is $-\infty$)

Reachable by negative cycle also the cost will be $-\infty$

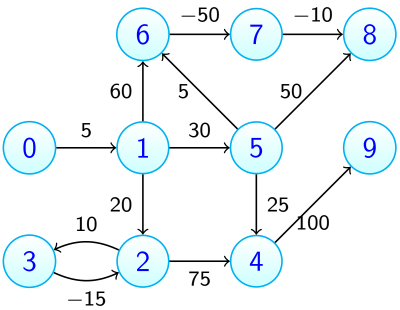
Shortest Path: BF



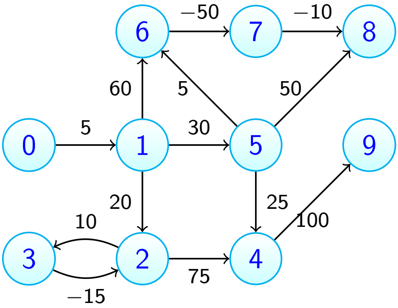


Invovled directly in a negative cycle (cost is $-\infty$)

Reachable by negative cycle also the cost will be $-\infty$

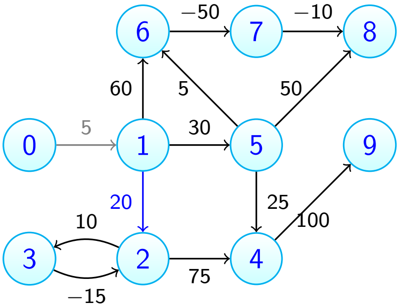


0	∞
1	∞
2	∞
3	∞
4	∞
5	∞
6	∞
7	∞
8	∞
9	∞



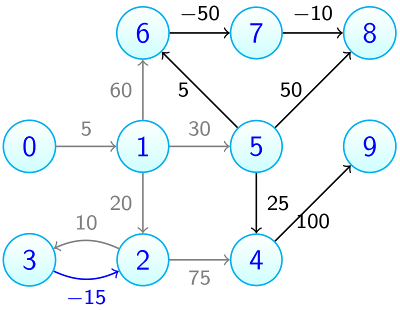
0	0
1	∞
2	∞
3	∞
4	∞
5	∞
6	∞
7	∞
8	∞
9	∞

No need to process edges in particular **order.**



0	0
1	5
2	25
3	∞
4	∞
5	∞
6	∞
7	∞
8	∞
9	∞

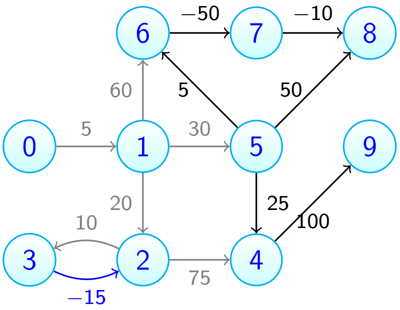
No need to process edges in particular **order.**



0	0
1	5
2	25
3	35
4	100
5	35
6	65
7	∞
8	∞
9	∞

We can reach the node 2 from 3. Can we update the path?

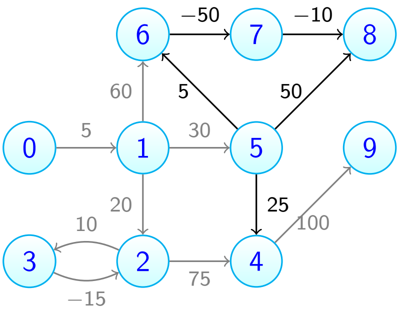
new path cost 20 the old path has a cost of 25



0	0
1	5
2	20
3	35
4	100
5	35
6	65
7	∞
8	∞
9	∞

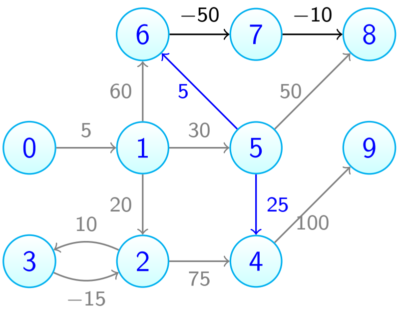
We can reach the node 2 from 3. Can we update the path?

new path cost 20 the old path has a cost of 25

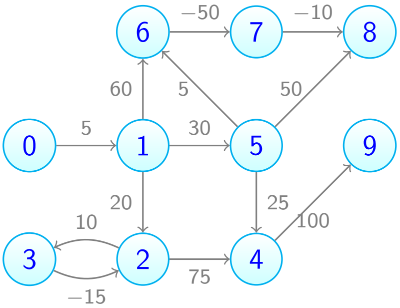


0	0
1	5
2	20
3	35
4	100
5	35
6	65
7	∞
8	∞
9	200

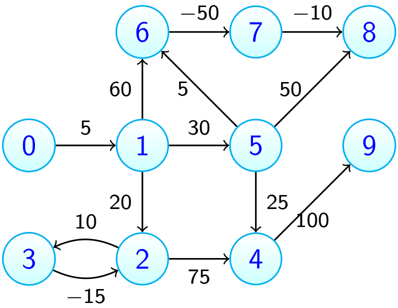
Shortest Path: BF



0	0
1	5
2	20
3	35
4	60
5	35
6	40
7	∞
8	85
9	200



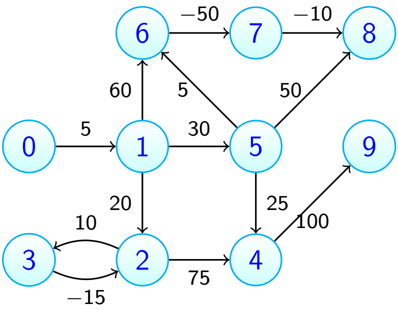
0	0
1	5
2	20
3	35
4	60
5	35
6	40
7	−10
8	−20
9	200



0	0
1	5
2	20
3	35
4	60
5	35
6	40
7	−10
8	−20
9	200

First iteration is done we need more 8 iterations.

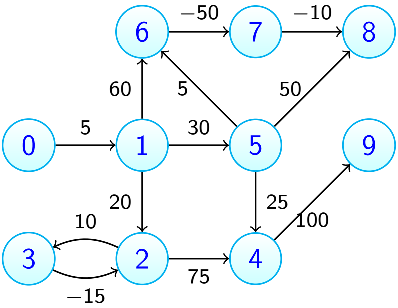
```
for (i=0;i<v-1;i++):
 for edge in graph.edges:
  #Relax edge (update D)
   if D[edge.from]+edge.cost<D[edge.to]
    D[edge.to]=D[edge.from]+edge.cost
```



0	0
1	5
2	20
3	35
4	60
5	35
6	40
7	−10
8	−20
9	200

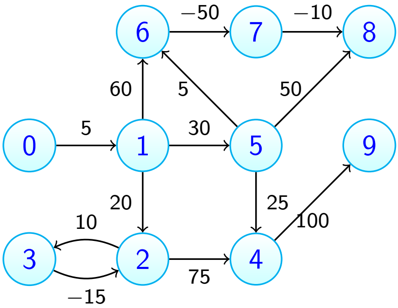
First iteration is done we need more 8 iterations.

```
for (i=0;i<v-1;i++):
 for edge in graph.edges:
  #Relax edge (update D)
   if D[edge.from]+edge.cost<D[edge.to]
    D[edge.to]=D[edge.from]+edge.cost
```



0	0
1	5
2	−20
3	−5
4	60
5	35
6	40
7	−10
8	−20
9	160

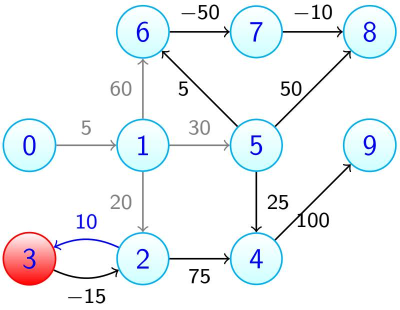
Once you finish all 9th Iteration here is the final result.



0	0
1	5
2	−20
3	−5
4	60
5	35
6	40
7	−10
8	−20
9	160

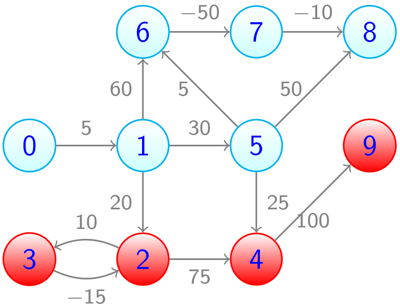
We finish the SSSP.

How to detect the negative cycles?

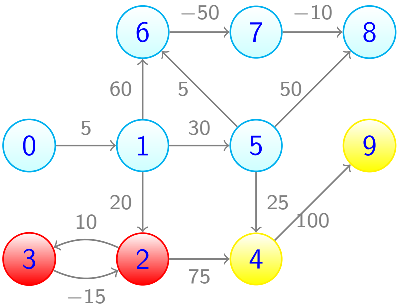


0	0
1	5
2	−20
3	$-\infty$
4	60
5	35
6	40
7	−10
8	−20
9	160

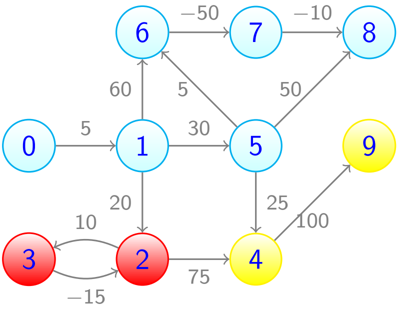
Shortest Path: BF



0	0
1	5
2	$-\infty$
3	$-\infty$
4	$-\infty$
5	35
6	40
7	-10
8	-20
9	$-\infty$



0	0
1	5
2	$-\infty$
3	$-\infty$
4	$-\infty$
5	35
6	40
7	-10
8	-20
9	$-\infty$



0	0
1	5
2	$-\infty$
3	$-\infty$
4	$-\infty$
5	35
6	40
7	-10
8	-20
9	$-\infty$

We have done one iteration, we need to continue with the 8 other iterations to propagate the $-\infty$

Arbitrage Opportunities

Imagine you're a trader in the stock market and you've identified a set of stocks or assets where you believe there are potential arbitrage opportunities. Arbitrage involves exploiting price differences of the same asset in different markets or related assets in the same market to make a profit with little or no risk.

- **Model the Market as a Graph**: Represent the market with its various assets and their relationships as a graph.
- **Assign Edge Weights**: Calculate the potential profits or losses from trading between different assets and assign them as edge weights. Negative edge weights would indicate potential profits from short-selling or hedging.
- **Run Bellman-Ford Algorithm**: to find paths with the highest potential profits, considering both positive and negative edge weights. The algorithm will identify opportunities for arbitrage where the sum of edge weights along a path is negative, indicating a profitable trade or hedging strategy.
- **Execute Trades**

Arbitrage Opportunities

Imagine you're a trader in the stock market and you've identified a set of stocks or assets where you believe there are potential arbitrage opportunities. Arbitrage involves exploiting price differences of the same asset in different markets or related assets in the same market to make a profit with little or no risk.

- **Model the Market as a Graph**: Represent the market with its various assets and their relationships as a graph.
- **Assign Edge Weights**: Calculate the potential profits or losses from trading between different assets and assign them as edge weights. Negative edge weights would indicate potential profits from short-selling or hedging.
- **Run Bellman-Ford Algorithm**: to find paths with the highest potential profits, considering both positive and negative edge weights. The algorithm will identify opportunities for arbitrage where the sum of edge weights along a path is negative, indicating a profitable trade or hedging strategy.
- **Execute Trades**

Arbitrage Opportunities

Imagine you're a trader in the stock market and you've identified a set of stocks or assets where you believe there are potential arbitrage opportunities. Arbitrage involves exploiting price differences of the same asset in different markets or related assets in the same market to make a profit with little or no risk.

- **Model the Market as a Graph**: Represent the market with its various assets and their relationships as a graph.
- **Assign Edge Weights**: Calculate the potential profits or losses from trading between different assets and assign them as edge weights. Negative edge weights would indicate potential profits from short-selling or hedging.
- **Run Bellman-Ford Algorithm**: to find paths with the highest potential profits, considering both positive and negative edge weights. The algorithm will identify opportunities for arbitrage where the sum of edge weights along a path is negative, indicating a profitable trade or hedging strategy.
- **Execute Trades**

Arbitrage Opportunities

Imagine you're a trader in the stock market and you've identified a set of stocks or assets where you believe there are potential arbitrage opportunities. Arbitrage involves exploiting price differences of the same asset in different markets or related assets in the same market to make a profit with little or no risk.

- **Model the Market as a Graph**: Represent the market with its various assets and their relationships as a graph.
- **Assign Edge Weights**: Calculate the potential profits or losses from trading between different assets and assign them as edge weights. Negative edge weights would indicate potential profits from short-selling or hedging.
- **Run Bellman-Ford Algorithm**: to find paths with the highest potential profits, considering both positive and negative edge weights. The algorithm will identify opportunities for arbitrage where the sum of edge weights along a path is negative, indicating a profitable trade or hedging strategy.
- Execute Trades

Arbitrage Opportunities
Imagine you're a trader in the stock market and you've identified a set of stocks or assets where you believe there are potential arbitrage opportunities. Arbitrage involves exploiting price differences of the same asset in different markets or related assets in the same market to make a profit with little or no risk.

- **Model the Market as a Graph**: Represent the market with its various assets and their relationships as a graph.
- **Assign Edge Weights**: Calculate the potential profits or losses from trading between different assets and assign them as edge weights. Negative edge weights would indicate potential profits from short-selling or hedging.
- **Run Bellman-Ford Algorithm**: to find paths with the highest potential profits, considering both positive and negative edge weights. The algorithm will identify opportunities for arbitrage where the sum of edge weights along a path is negative, indicating a profitable trade or hedging strategy.
- **Execute Trades**