Advanced Data Structures and Algorithms

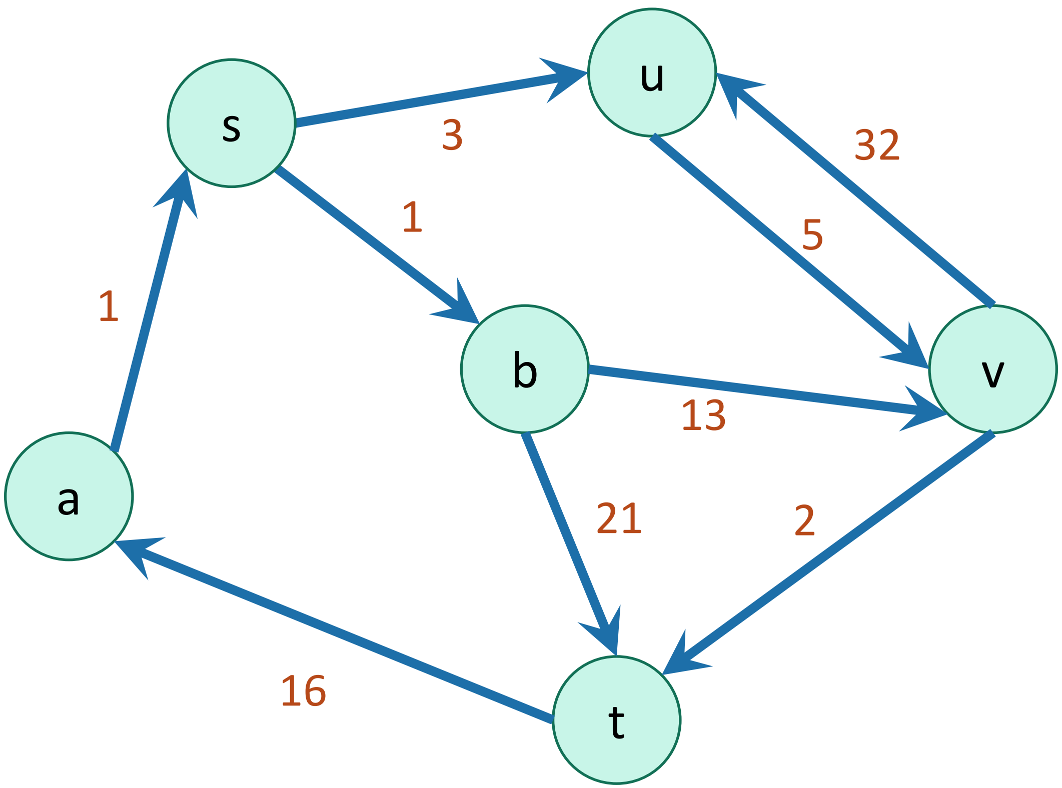
Single Source Shortest Paths (SSSP):

Bellman-Ford Algo

Recall

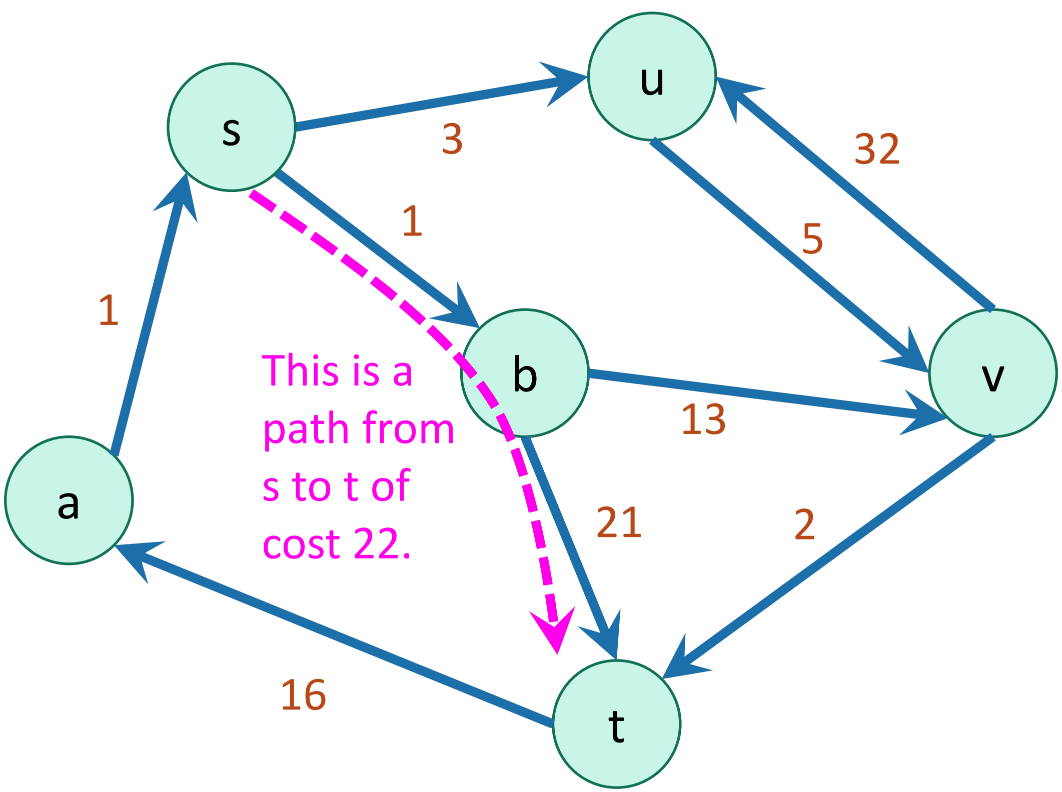
- Weights on edges represent costs.

- A weighted directed graph:



Recall

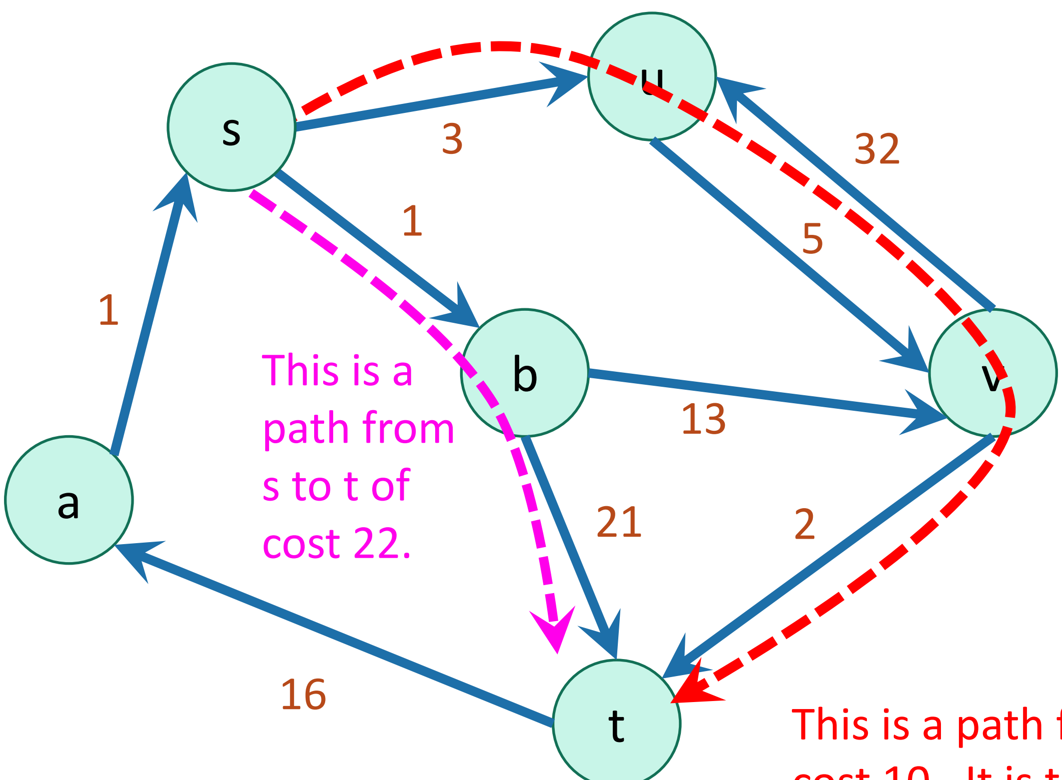
- A weighted directed graph:



- Weights on edges represent costs.
- The cost of a path is the sum of the weights along that path.

Recall

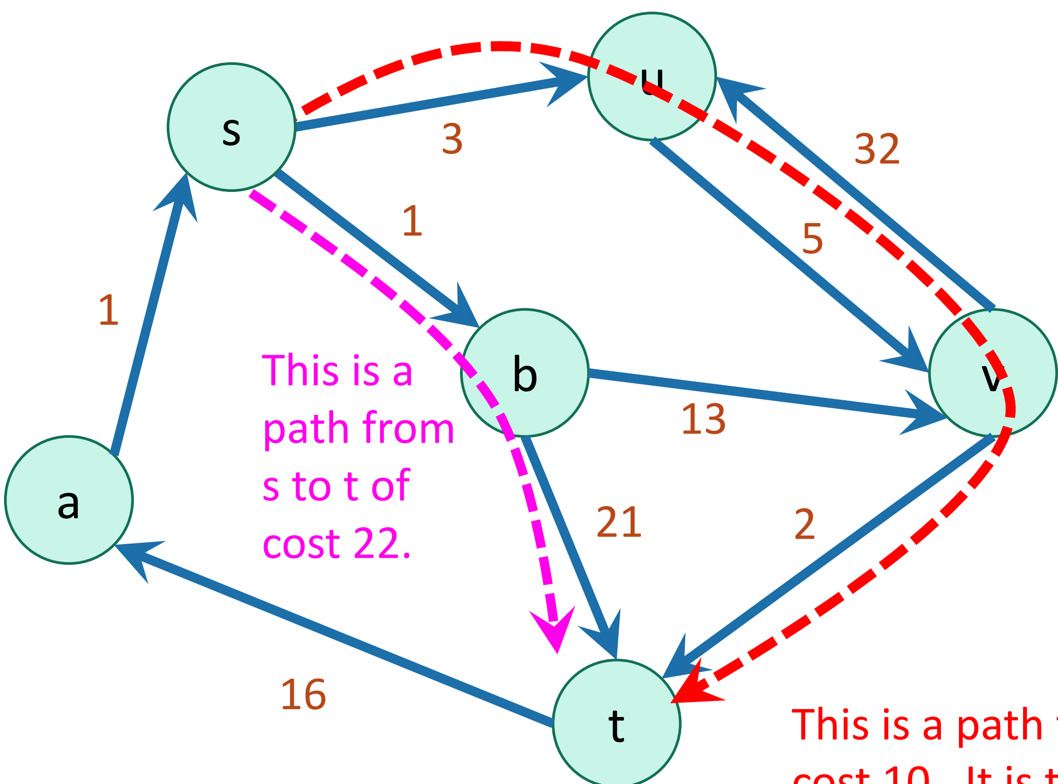
- A weighted directed graph:



- Weights on edges represent costs.
- The cost of a path is the sum of the weights along that path.
- A shortest path from s to t is a directed path from s to t with the smallest cost.

Recall

- A weighted directed graph:



This is a path from s to t of cost 10. It is the shortest path from s to t.

- Weights on edges represent costs.
- The cost of a path is the sum of the weights along that path.
- A shortest path from s to t is a directed path from s to t with the smallest cost.
- The single-source shortest path problem is to find the shortest path from s to v for all v in the graph.

Intro to Bellman-Ford

- Basic idea:
  - Instead of picking the u with the smallest d[u] to update, just update all of the u's simultaneously.

Bellman-Ford algorithm

Bellman-Ford(G,s):

- d[v] = ∞ for all v in V
- d[s] = 0
- **For** i=0,…,n-1:
  - **For** u in V:
    - **For** v in u.neighbors:
      - d[v] ← min(d[v] , d[u] + edgeWeight(u,v))

Instead of picking u cleverly,
just update for all of the u's.

Bellman-Ford algorithm

Bellman-Ford(G,s):

- d[v] = ∞ for all v in V
- d[s] = 0
- **For** i=0,…,n-1:
  - **For** u in V:
    - **For** v in u.neighbors:
      - d[v] ← min(d[v] , d[u] + edgeWeight(u,v))

Instead of picking u cleverly, just update for all of the u's.

Compare to Dijkstra:

- **While** there are **not-sure** nodes:
  - Pick the **not-sure** node u with the smallest estimate **d[u].**
  - **For** v in u.neighbors:
    - d[v] ← min(d[v] , d[u] + edgeWeight(u,v))
  - Mark u as **sure**.

For pedagogical reasons

- We are actually going to change this to be less smart.
- Keep n arrays: $d^{(0)}$, $d^{(1)}$, …, $d^{(n-1)}$

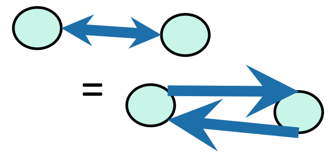
Bellman-Ford*(G,s):

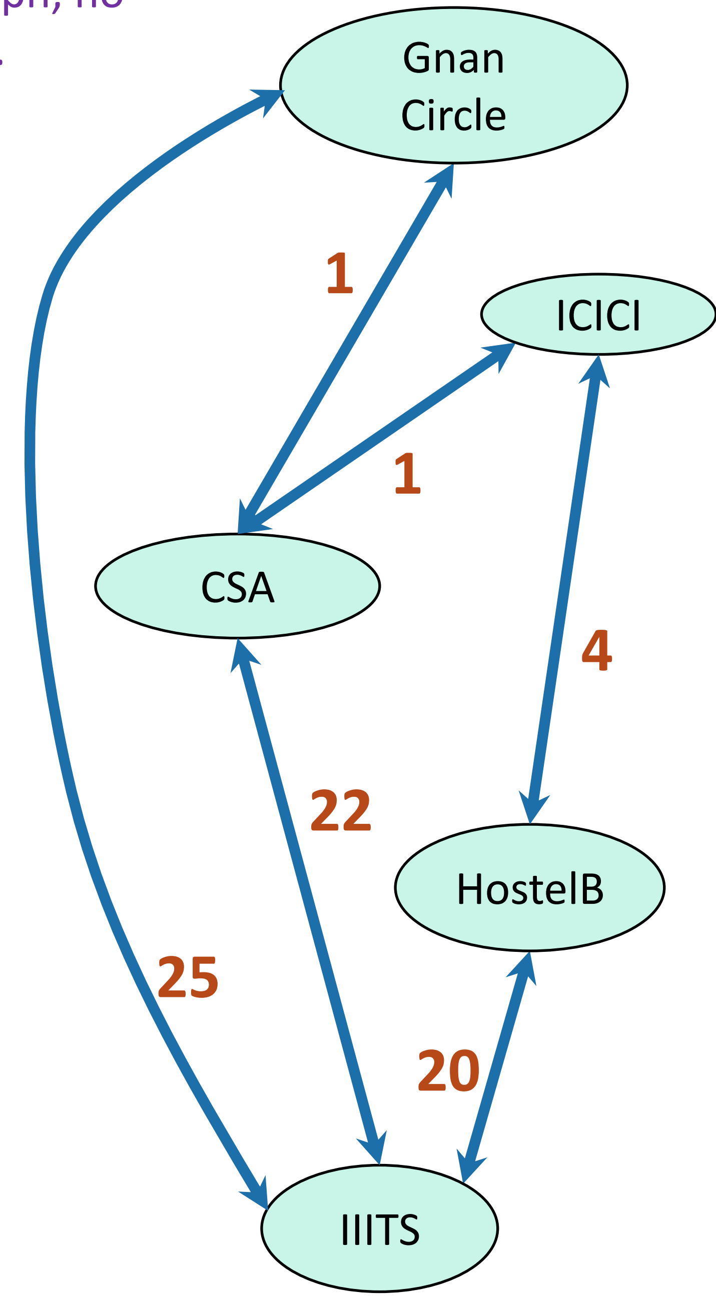
- $d^{(0)}[v] = \infty$ for all v in V
- $d^{(0)}[s] = 0$
- **For** i=0,…,n-2:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\ ,\ d^{(i+1)}[v],\ d^{(i)}[u] + \mathrm{edgeWeight}(u,v))$
- Then $\mathrm{dist}(s,v) = d^{(n-1)}[v]$

Slightly different than the original Bellman-Ford algorithm, but the analysis is basically the same.

Bellman-Ford

Start with the same graph, no negative weights.





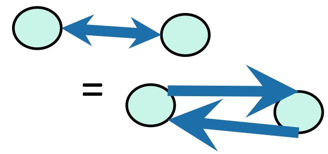
How far is a node from Gnan Circle

	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$					
$d^{(1)}$					
$d^{(2)}$					
$d^{(3)}$					
$d^{(4)}$					

- **For** i=0,…,n-2:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\ ,\ d^{(i+1)}[v],\ d^{(i)}[u] + \text{edgeWeight}(u,v))$

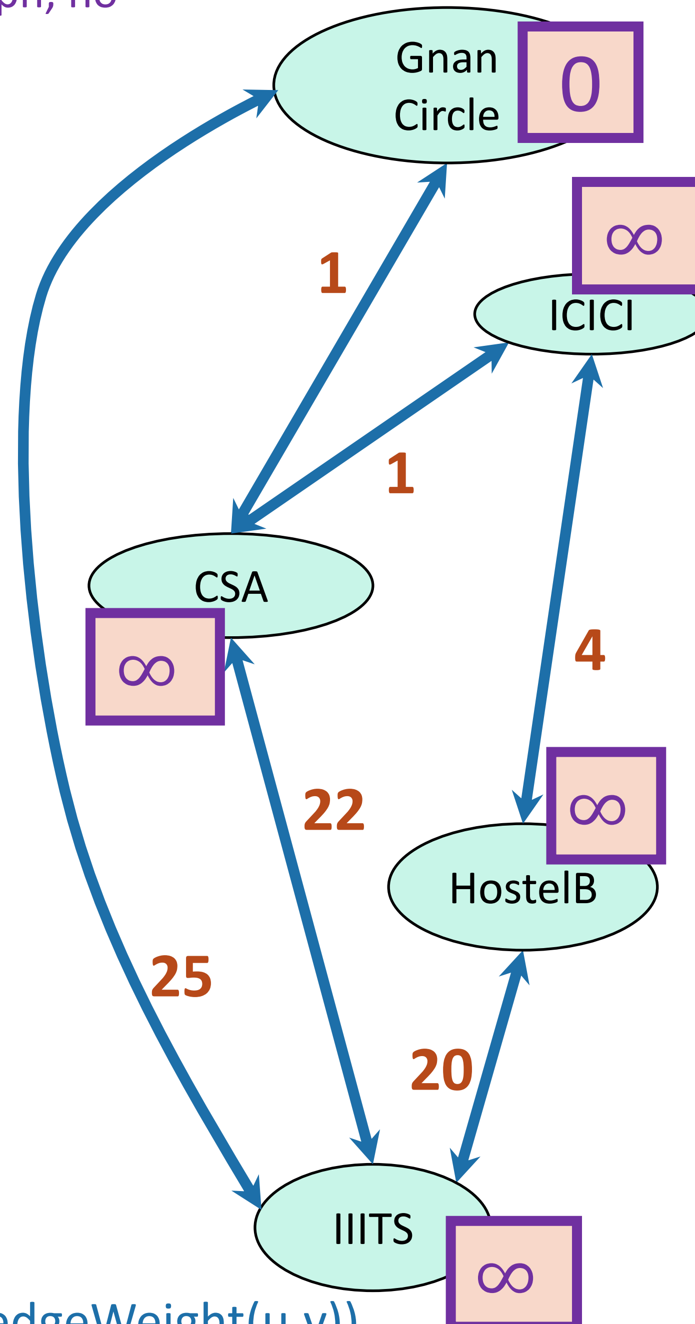
Bellman-Ford

Start with the same graph, no negative weights.



How far is a node from Gnan Circle

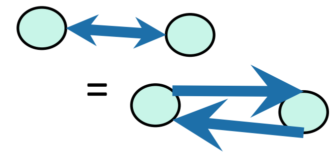
	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$					
$d^{(2)}$					
$d^{(3)}$					
$d^{(4)}$					



- **For** i=0,…,n-2:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\ ,\ d^{(i+1)}[v],\ d^{(i)}[u] + \text{edgeWeight}(u,v))$

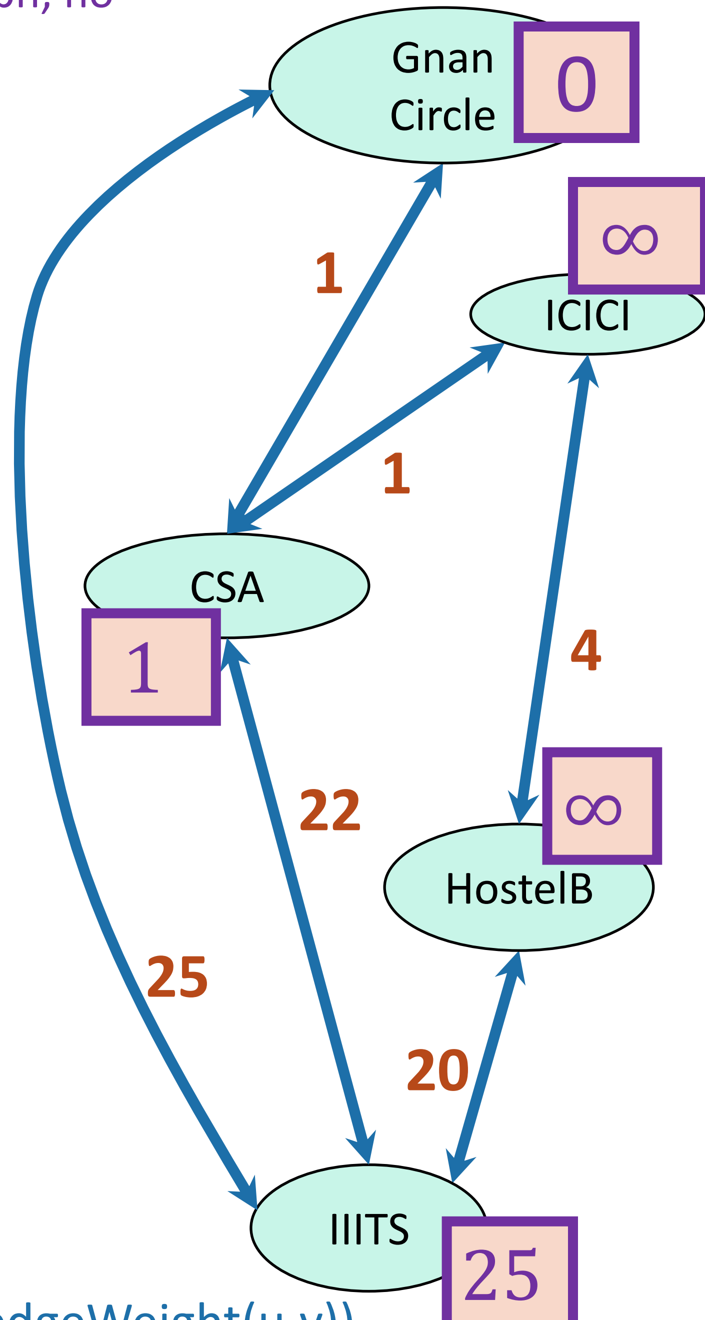
Bellman-Ford

Start with the same graph, no negative weights.



How far is a node from Gnan Circle

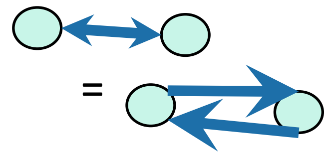
	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**25**
$d^{(2)}$					
$d^{(3)}$					
$d^{(4)}$					



- **For** i=0,…,n-2:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\ ,\ d^{(i+1)}[v],\ d^{(i)}[u] + \text{edgeWeight}(u,v))$

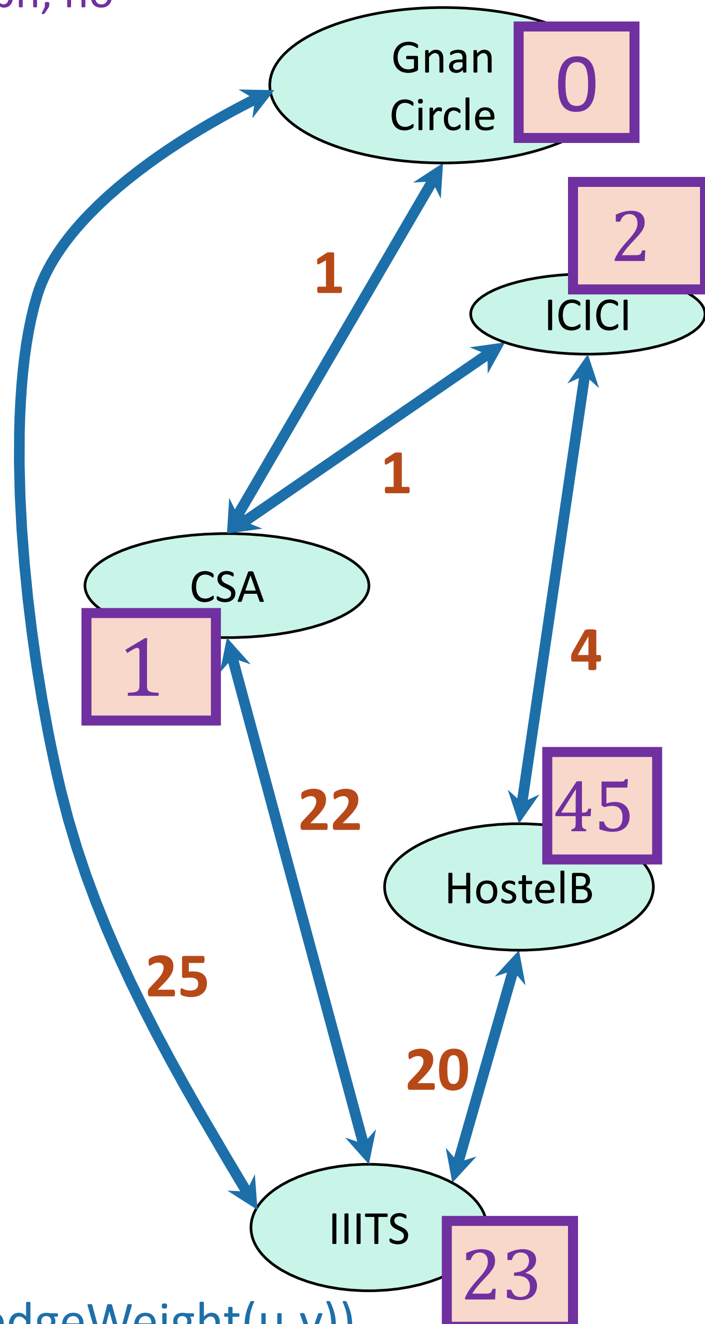
Bellman-Ford

Start with the same graph, no negative weights.



How far is a node from Gnan Circle

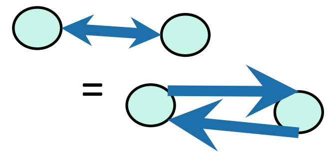
	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**25**
$d^{(2)}$	0	1	**2**	**45**	**23**
$d^{(3)}$					
$d^{(4)}$					



- **For** i=0,…,n-2:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\ ,\ d^{(i+1)}[v],\ d^{(i)}[u] + \text{edgeWeight}(u,v))$

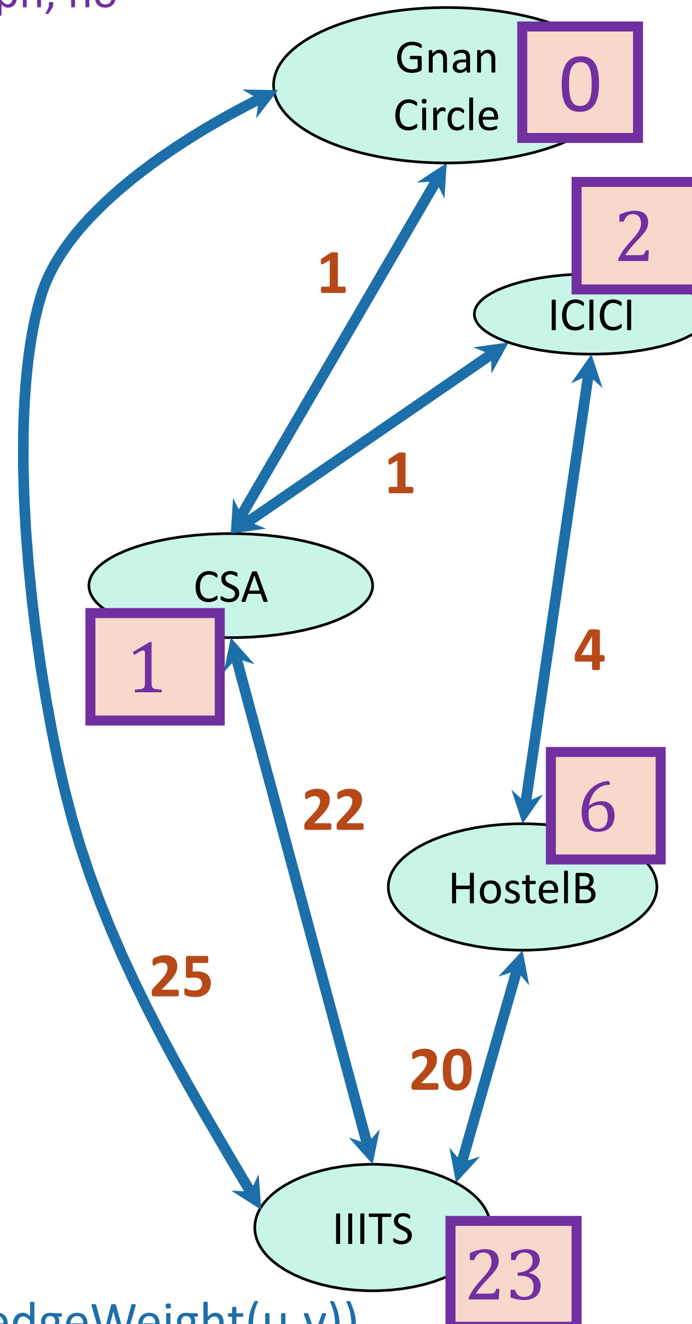
Bellman-Ford

Start with the same graph, no negative weights.



How far is a node from Gnan Circle

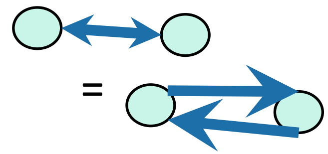
	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**25**
$d^{(2)}$	0	1	**2**	**45**	**23**
$d^{(3)}$	0	1	2	**6**	23
$d^{(4)}$					



- **For** i=0,…,n-2:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\ ,\ d^{(i+1)}[v],\ d^{(i)}[u] + \text{edgeWeight}(u,v))$

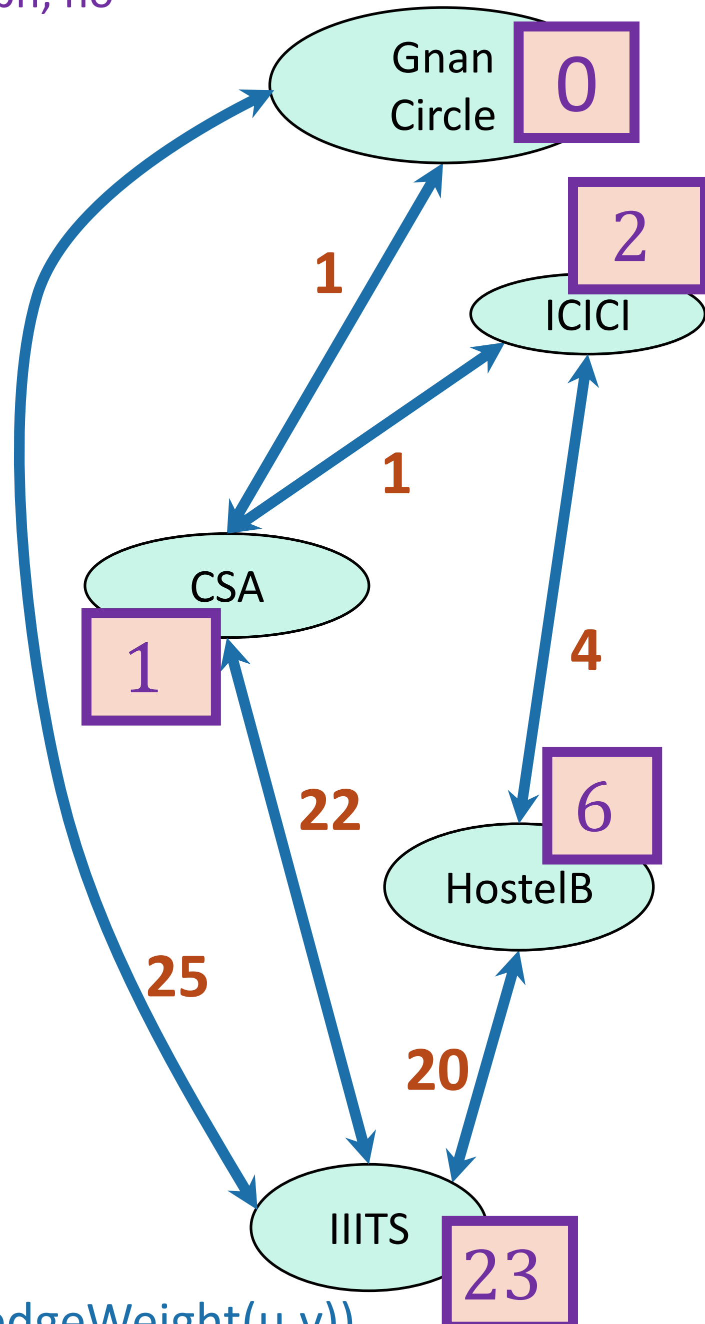
Bellman-Ford

Start with the same graph, no negative weights.



How far is a node from Gnan Circle

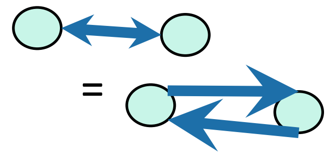
	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**25**
$d^{(2)}$	0	1	**2**	**45**	**23**
$d^{(3)}$	0	1	2	**6**	23
$d^{(4)}$	0	1	2	6	23



- **For** i=0,…,n-2:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v] , d^{(i+1)}[v], d^{(i)}[u] + \text{edgeWeight}(u,v))$

Bellman-Ford

Start with the same graph, no negative weights.

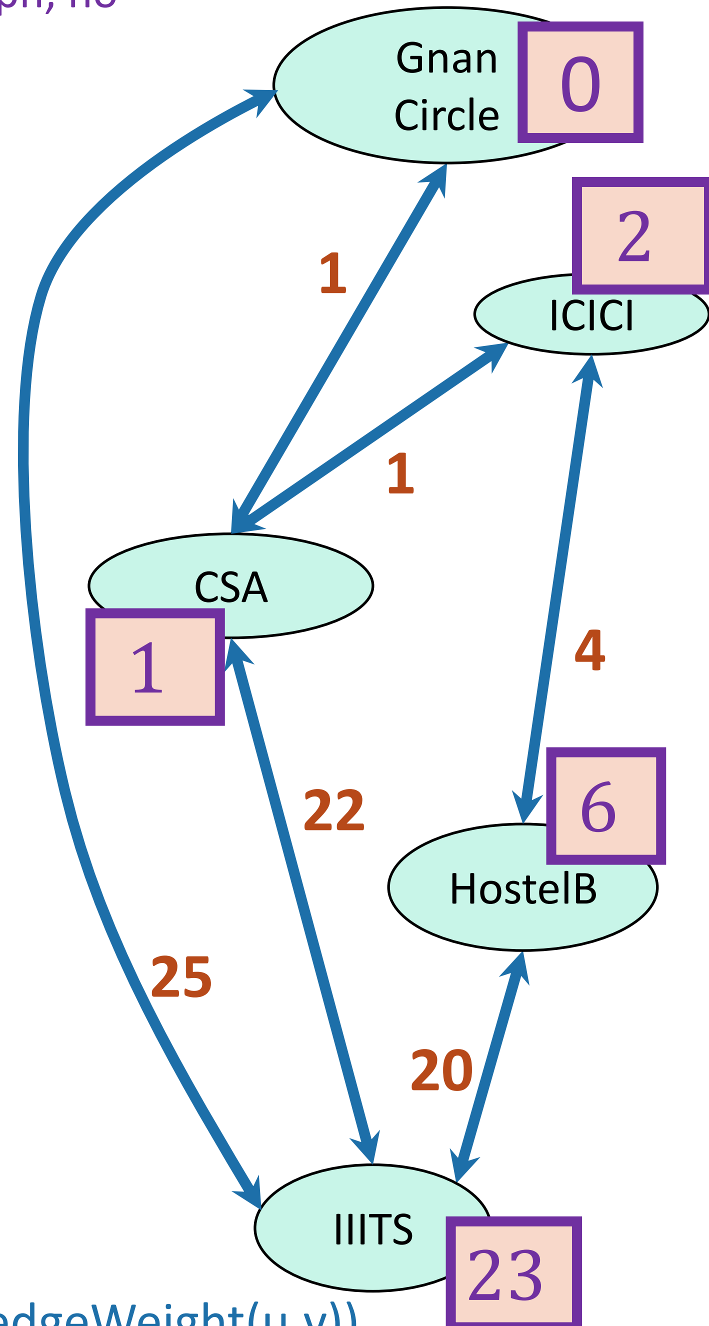


How far is a node from Gnan Circle

	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**25**
$d^{(2)}$	0	1	**2**	**45**	**23**
$d^{(3)}$	0	1	2	**6**	23
$d^{(4)}$	0	1	2	6	23

These are the final distances!

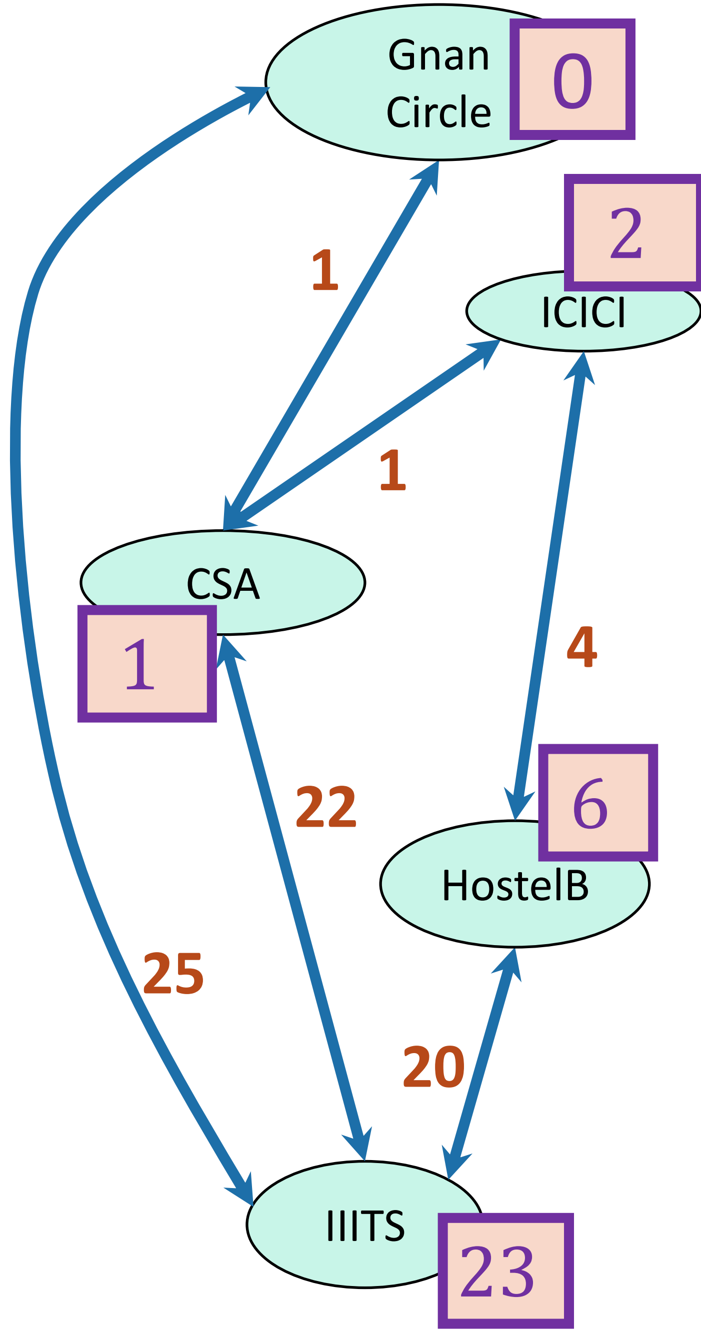
- **For** i=0,…,n-2:
    - **For** u in V:
        - **For** v in u.neighbors:
            - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\ ,\ d^{(i+1)}[v],\ d^{(i)}[u] + \text{edgeWeight}(u,v))$



Interpretation of $d^{(i)}$

$d^{(i)}[v]$ is equal to the cost of the shortest path between s and v **with at most i edges**.

	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**25**
$d^{(2)}$	0	1	**2**	**45**	**23**
$d^{(3)}$	0	1	2	**6**	23
$d^{(4)}$	0	1	2	6	23



As usual

	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**25**
$d^{(2)}$	0	1	**2**	**45**	**23**
$d^{(3)}$	0	1	2	**6**	23
$d^{(4)}$	0	1	2	6	23

- Does it work?

- Is it fast?

As usual

- Does it work?
  - Yes
  - Idea to the right.
- Is it fast?

	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**25**
$d^{(2)}$	0	1	**2**	**45**	**23**
$d^{(3)}$	0	1	2	**6**	23
$d^{(4)}$	0	1	2	6	23

Idea: proof by induction.
Inductive Hypothesis:
$d^{(i)}[v]$ is equal to the cost of the shortest path between s and v **with at most i edges**.

As usual

- Does it work?
  - Yes
  - Idea to the right.

- Is it fast?

	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**25**
$d^{(2)}$	0	1	**2**	**45**	**23**
$d^{(3)}$	0	1	2	**6**	23
$d^{(4)}$	0	1	2	6	23

Idea: proof by induction.
Inductive Hypothesis:
$d^{(i)}[v]$ is equal to the cost of the shortest path between s and v **with at most i edges**.
Conclusion:
$d^{(n-1)}[v]$ is equal to the cost of the shortest simple path between s and v. **(Since all simple paths have at most n-1 edges).**

As usual

- Does it work?
  - Yes
  - Idea to the right.
- Is it fast?
  - Not really…

	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**25**
$d^{(2)}$	0	1	**2**	**45**	**23**
$d^{(3)}$	0	1	2	**6**	23
$d^{(4)}$	0	1	2	6	23

Idea: proof by induction.
Inductive Hypothesis:
$d^{(i)}[v]$ is equal to the cost of the shortest path between s and v **with at most i edges**.
Conclusion:
$d^{(n-1)}[v]$ is equal to the cost of the shortest simple path between s and v. **(Since all simple paths have at most n-1 edges).**

Aside: simple paths

Assume there is no negative cycle.

Aside: simple paths

Assume there is no negative cycle.

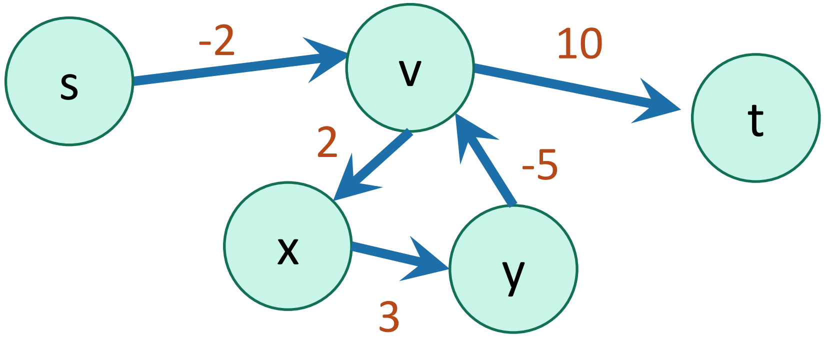
- Then there is a shortest path from s to t, and moreover there is a simple shortest path.

"Simple" means that the path has no cycles in it.

Aside: simple paths

Assume there is no negative cycle.

- Then there is a shortest path from s to t, and moreover there is a simple shortest path.

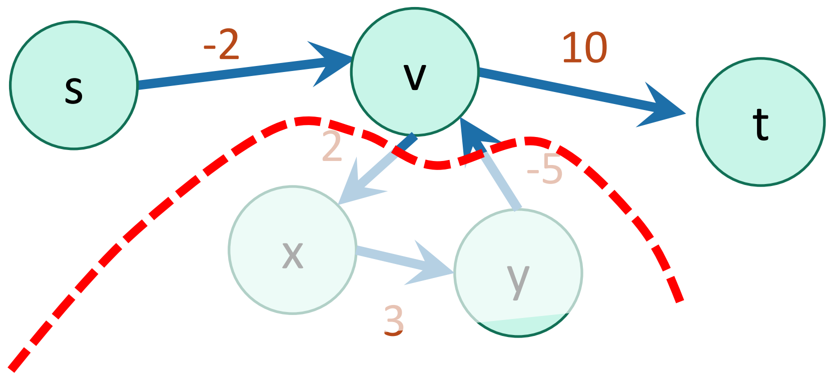


"Simple" means that the path has no cycles in it.

Aside: simple paths

Assume there is no negative cycle.

- Then there is a shortest path from s to t, and moreover there is a simple shortest path.



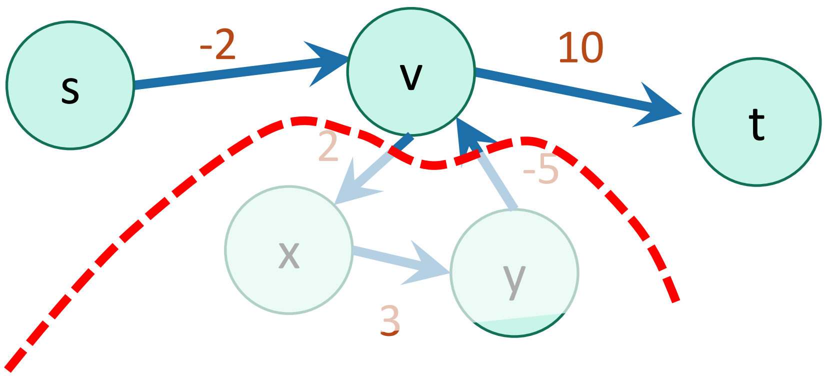
This cycle isn't helping. Just get rid of it.

"Simple" means that the path has no cycles in it.

Aside: simple paths

Assume there is no negative cycle.

- Then there is a shortest path from s to t, and moreover there is a simple shortest path.



This cycle isn't helping. Just get rid of it.

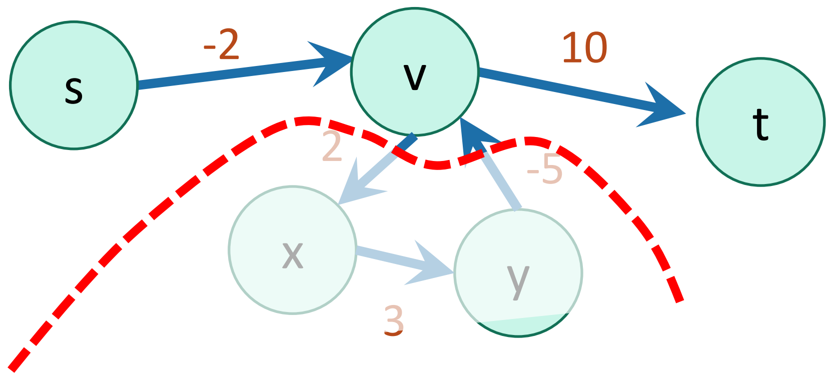
- A simple path in a graph with n vertices has at most n-1 edges in it.

"Simple" means that the path has no cycles in it.

Aside: simple paths

Assume there is no negative cycle.

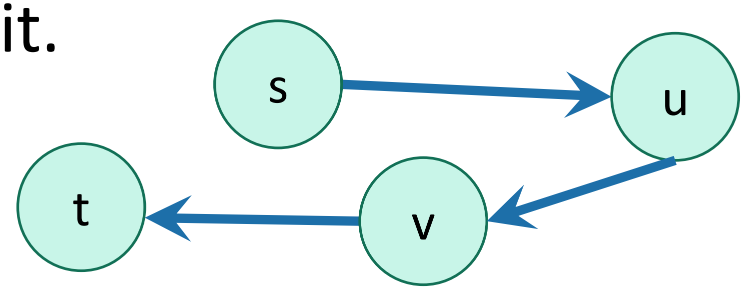
- Then there is a shortest path from s to t, and moreover there is a simple shortest path.



This cycle isn't helping. Just get rid of it.

- A simple path in a graph with n vertices has at most n-1 edges in it.

Can't add another edge without making a cycle!

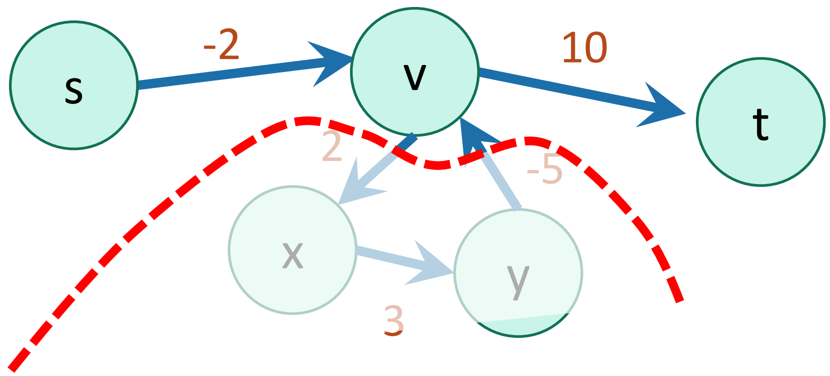


"Simple" means that the path has no cycles in it.

Aside: simple paths

Assume there is no negative cycle.

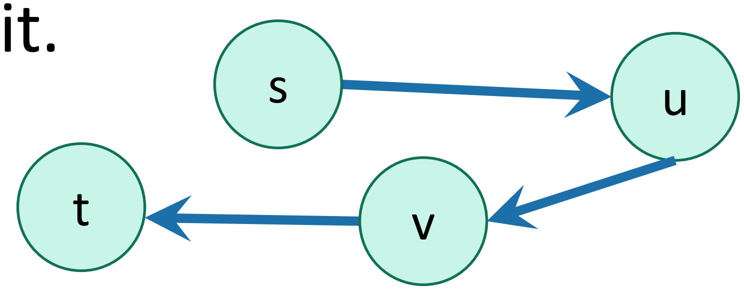
- Then there is a shortest path from s to t, and moreover there is a simple shortest path.



This cycle isn't helping.
Just get rid of it.

- A simple path in a graph with n vertices has at most n-1 edges in it.

Can't add another edge
without making a cycle!



"Simple" means
that the path has
no cycles in it.

- So there is a shortest path with at most n-1 edges

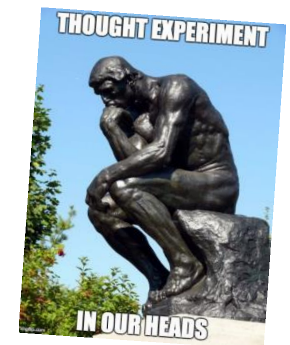
Proof by induction

- **Inductive Hypothesis:**
  - After iteration i, for each v, $d^{(i)}[v]$ is equal to the cost of the shortest path between s and v with at most i edges.
- **Base case:**
  - After iteration 0…
- **Inductive step:**

Inductive step

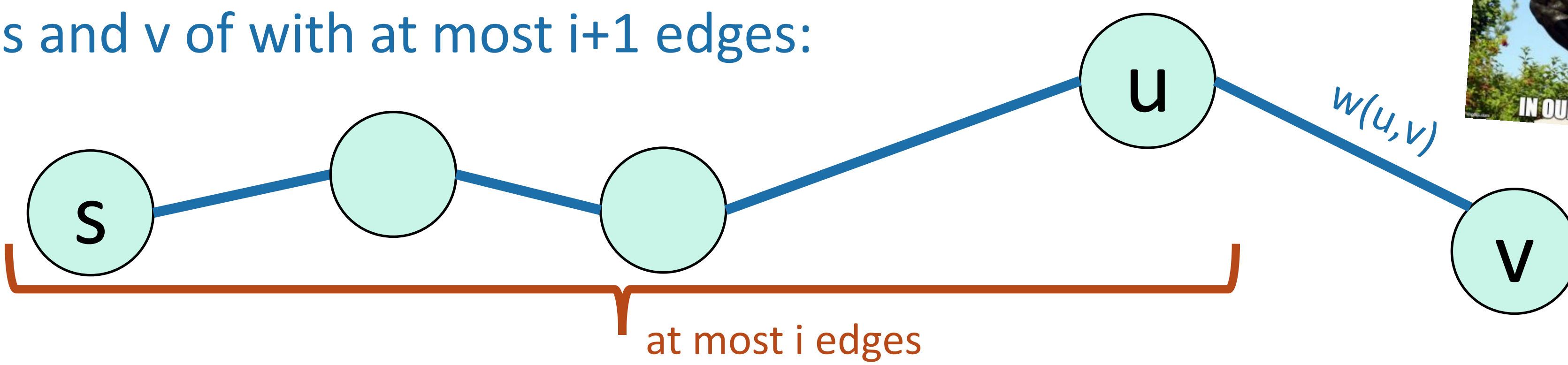
Hypothesis: After iteration i, for each v, $d^{(i)}[v]$ is equal to the cost of the shortest path between s and v with at most i edges.

- Suppose the inductive hypothesis holds for i.
- We want to establish it for i+1.



Say this is the shortest path between s and v of with at most i+1 edges:

Let u be the vertex right before v in this path.



- By induction, $d^{(i)}[u]$ is the cost of a shortest path between s and u of i edges.
- By setup, $d^{(i)}[u] + w(u,v)$ is the cost of a shortest path between s and v of i+1 edges.
- In the i+1'st iteration, we ensure $\mathbf{d^{(i+1)}[v] <= d^{(i)}[u] + w(u,v)}$.
- So $d^{(i+1)}[v]$ <= cost of shortest path between s and v with i+1 edges.
- But $d^{(i+1)}[v]$ = cost of a particular path of at most i+1 edges >= cost of shortest path.
- So d[v] = cost of shortest path with at most i+1 edges.

Proof by induction

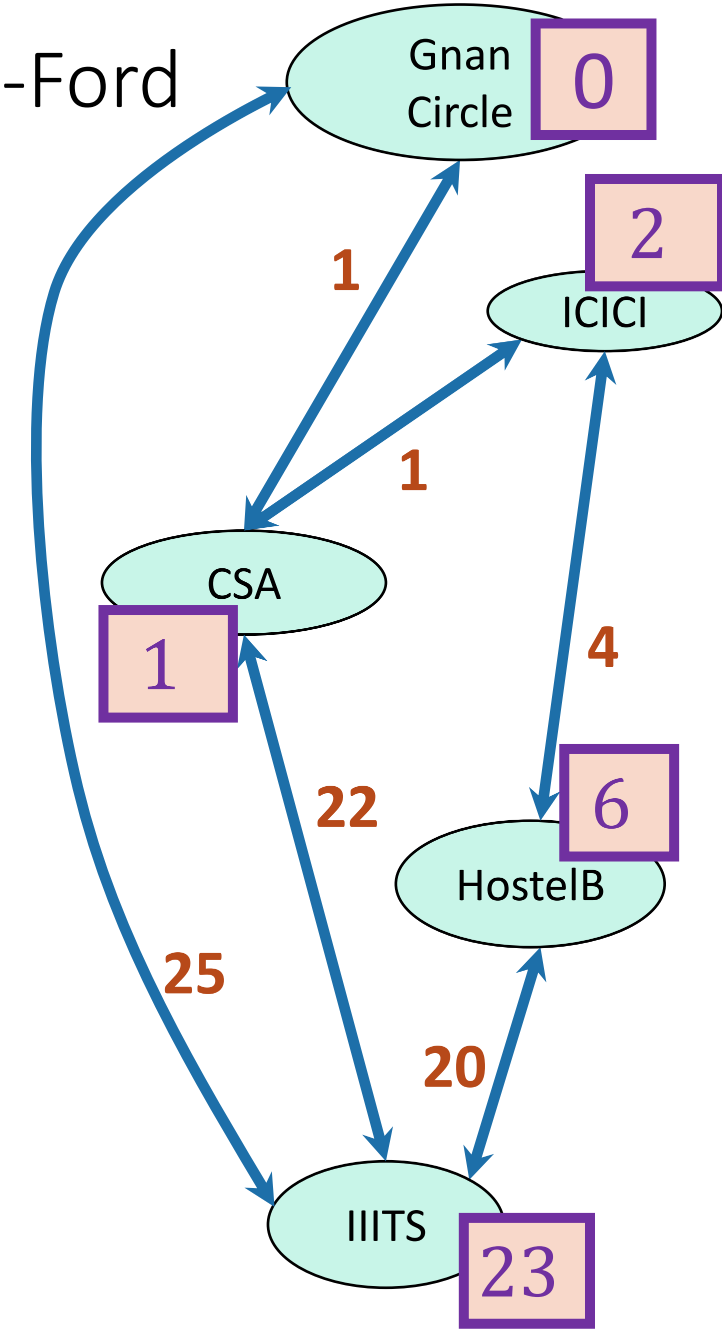
- **Inductive Hypothesis:**
  - After iteration i, for each v, $d^{(i)}[v]$ is equal to the cost of the shortest path between s and v of length at most i edges.
- **Base case:**
  - After iteration 0… ✓
- **Inductive step:** ✓
- **Conclusion:**
  - After iteration n-1, for each v, d[v] is equal to the cost of the shortest path between s and v of length at most n-1 edges.
  - **Aka, d[v] = d(s,v) for all v** as long as there are no cycles! ✓

Important thing about Bellman-Ford

for the rest of this lecture

$d^{(i)}[v]$ is equal to the cost of the shortest path between s and v **with at most i edges**.

	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**25**
$d^{(2)}$	0	1	**2**	**45**	**23**
$d^{(3)}$	0	1	2	**6**	23
$d^{(4)}$	0	1	2	6	23



G = (V,E) is a graph with n vertices and m edges.

Bellman-Ford* algorithm

Bellman-Ford*(G,s):

- Initialize arrays $d^{(0)},\ldots,d^{(n-1)}$ of length n to be all ∞
- $d^{(0)}[s] = 0$
- **For** i=0,…,n-2:
  - **For** u in V:
    - **For** v in u.outNeighbors:
      - $d^{(i+1)}[v] \leftarrow \min(\, d^{(i)}[v]\,,\, d^{(i+1)}[v]\,,\, d^{(i)}[u] + w(u,v))$
- Now, dist(s,v) = $d^{(n-1)}[v]$ for all v in V.
  - (Assuming G has no negative cycles)

Here, Dijkstra picked a special vertex u – Bellman-Ford will just look at all the vertices u.

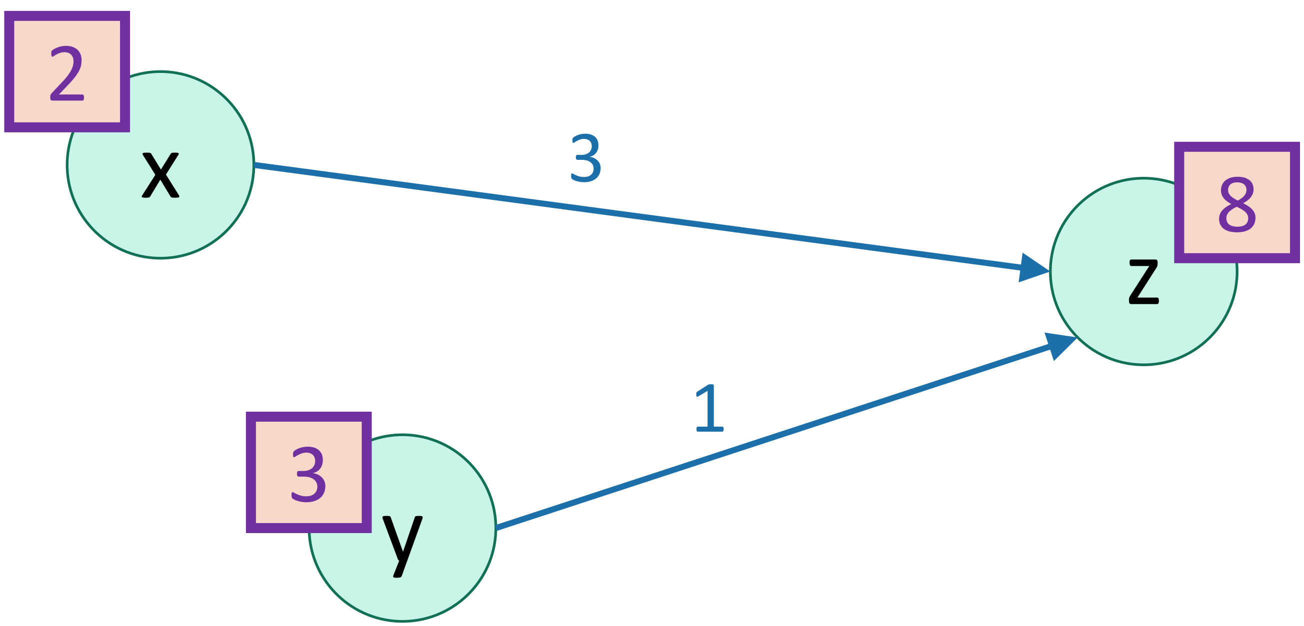
*Slightly different than some versions of Bellman-Ford…but this way is pedagogically convenient for today's lecture.

We can simplify the pseudocode a bit

- This will be useful later…

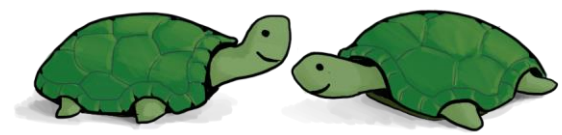
One step of Bellman-Ford

- **For** u in V:
  - **For** v in u.outNeighbors:
    - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v] , d^{(i+1)}[v] , d^{(i)}[u] + w(u,v))$

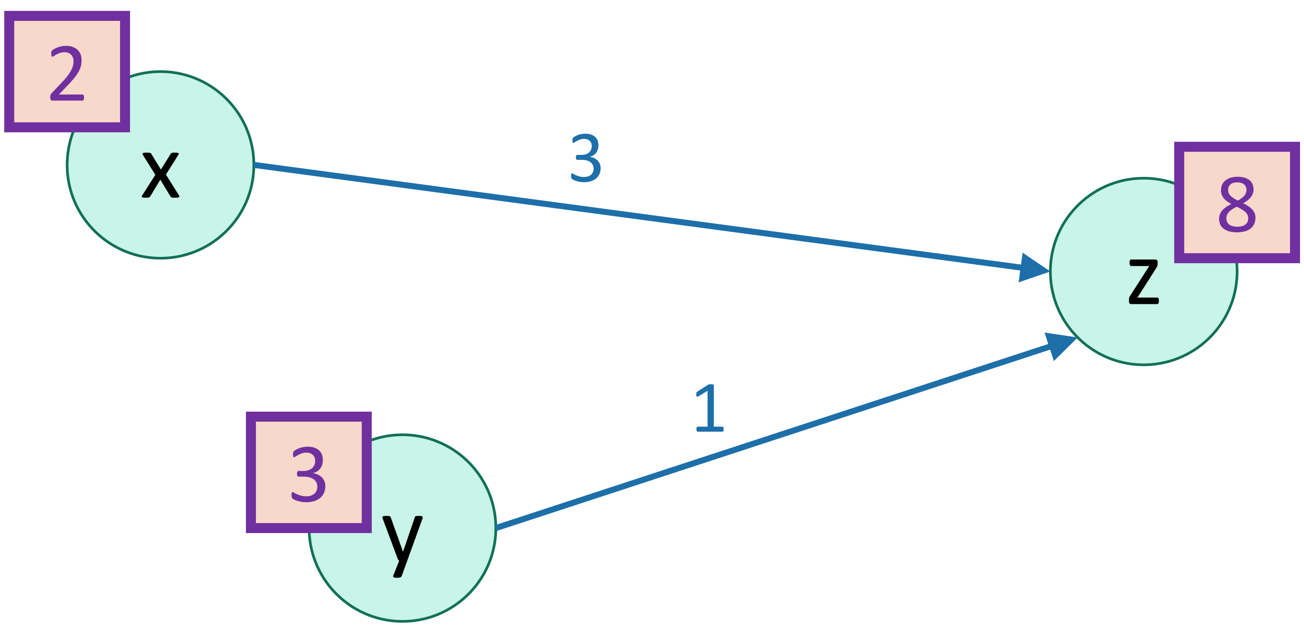


One step of Bellman-Ford

What will happen to z if we run these for-loops?

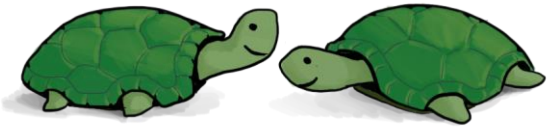


- **For** u in V:
    - **For** v in u.outNeighbors:
        - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v] , d^{(i+1)}[v] , d^{(i)}[u] + w(u,v))$

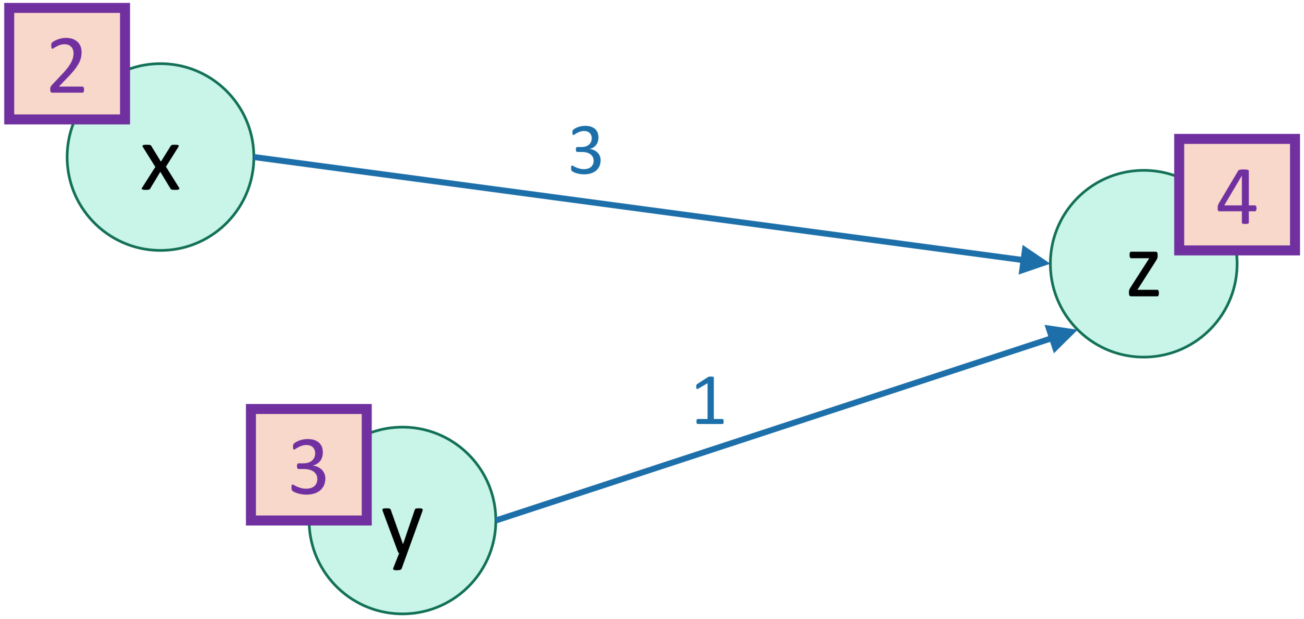


One step of Bellman-Ford

What will happen to z if we run these for-loops?

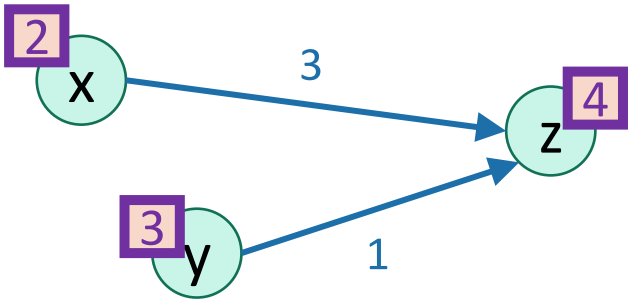


- **For** u in V:
  - **For** v in u.outNeighbors:
    - $d^{(i+1)}[v] \leftarrow \min(\, d^{(i)}[v] \,,\, d^{(i+1)}[v] \,,\, d^{(i)}[u] + w(u,v))$



One step of Bellman-Ford

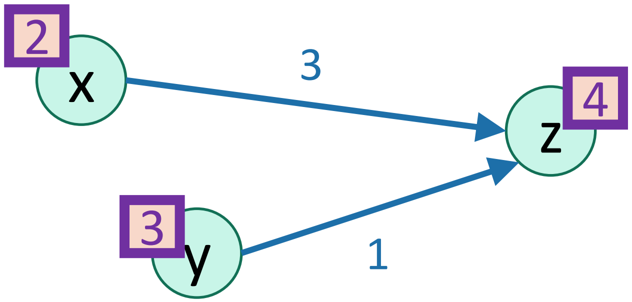
- **For** u in V:
  - **For** v in u.outNeighbors:
    - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v] , d^{(i+1)}[v], d^{(i)}[u] + w(u,v))$



One step of Bellman-Ford

- **For** u in V:
  - **For** v in u.outNeighbors:
    - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v] , d^{(i+1)}[v], d^{(i)}[u] + w(u,v))$

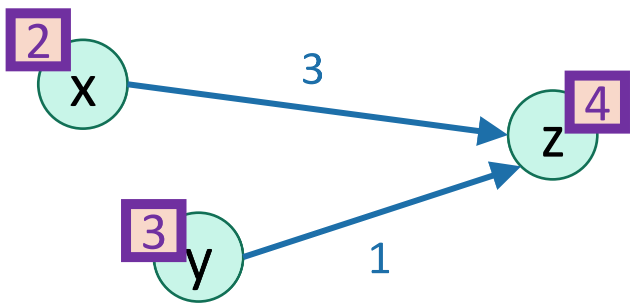
- Each vertex z finds the in-neighbor u so that $d^{(i)}[u] + w(u,z)$ is smallest and goes with that.
- (Unless z chooses not to update).



One step of Bellman-Ford

- **For** u in V:
  - **For** v in u.outNeighbors:
    - $d^{(i+1)}[v] \leftarrow \min(\ d^{(i)}[v]\ ,\ d^{(i+1)}[v],\ d^{(i)}[u] + w(u,v))$

- Each vertex z finds the in-neighbor u so that $d^{(i)}[u] + w(u,z)$ is smallest and goes with that.
- (Unless z chooses not to update).



- So we can equivalently write:
- **For** z in V:
  - $d^{(i+1)}[z] \leftarrow \min(\ d^{(i)}[z]\ ,\ \min_{u \text{ in } z.\text{inNbrs}}\{d^{(i)}[u] + w(u,z)\}\ )$

$G = (V,E)$ is a graph with n vertices and m edges.

Bellman-Ford* algorithm

Bellman-Ford*(G,s):

- Initialize arrays $d^{(0)},\ldots,d^{(n-1)}$ of length n
- $d^{(0)}[v] = \infty$ for all v in V
- $d^{(0)}[s] = 0$
- **For** i=0,…,n-2:
  - **For** v in V:
    - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v] , \min_{u \text{ in } v.\text{inNbrs}}\{d^{(i)}[u] + w(u,v)\})$
- Now, dist(s,v) = $d^{(n-1)}[v]$ for all v in V.

*Slightly different than some versions of Bellman-Ford…but this way is pedagogically convenient for today's lecture.

Note on implementation

- Don’t actually keep all n arrays around.
- Just keep two at a time: “last round” and “this round”

	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**25**
$d^{(2)}$	0	1	**2**	**45**	**23**
$d^{(3)}$	0	1	2	**6**	23
$d^{(4)}$	0	1	2	6	23

Note on implementation

- Don't actually keep all n arrays around.
- Just keep two at a time: "last round" and "this round"

	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**25**
$d^{(2)}$	0	1	**2**	**45**	**23**
$d^{(3)}$	0	1	2	**6**	23
$d^{(4)}$	0	1	2	6	23

Only need these two in order to compute $d^{(4)}$

Pros and cons of Bellman-Ford

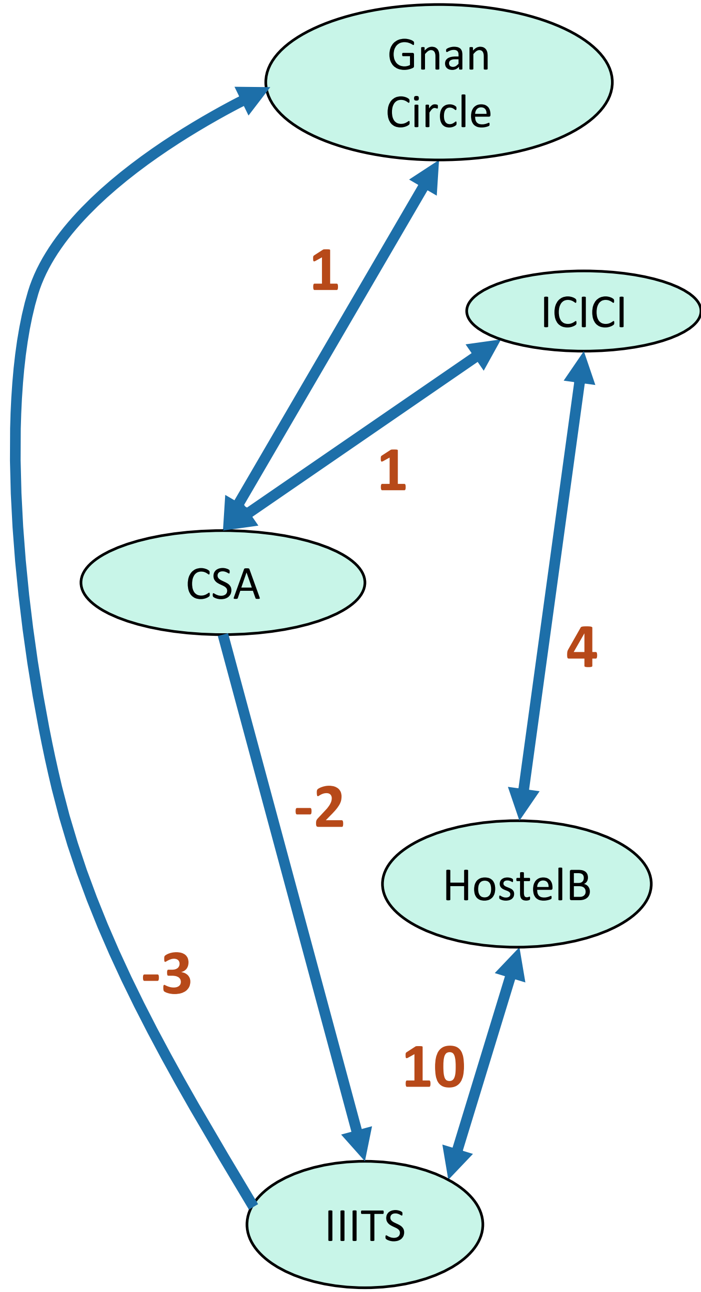
- Running time: O(mn) running time
  - For each of n steps we update m edges
  - Slower than Dijkstra

Pros and cons of Bellman-Ford

- Running time: O(mn) running time
  - For each of n steps we update m edges
  - Slower than Dijkstra
- However, it's also more flexible in a few ways.
  - Can handle negative edges
  - If we constantly do these iterations, any changes in the network will eventually propagate through.

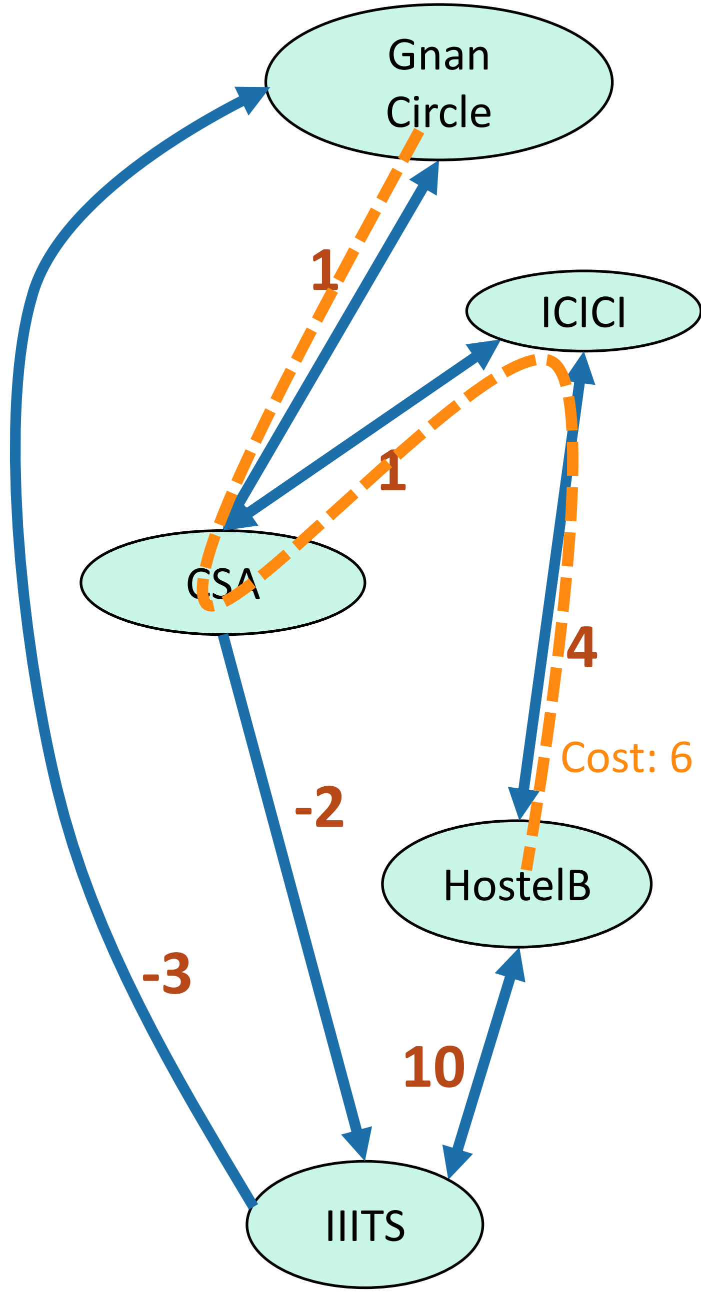
Wait a second…

- What is the shortest path from the Gnan Circle to the HostelB?



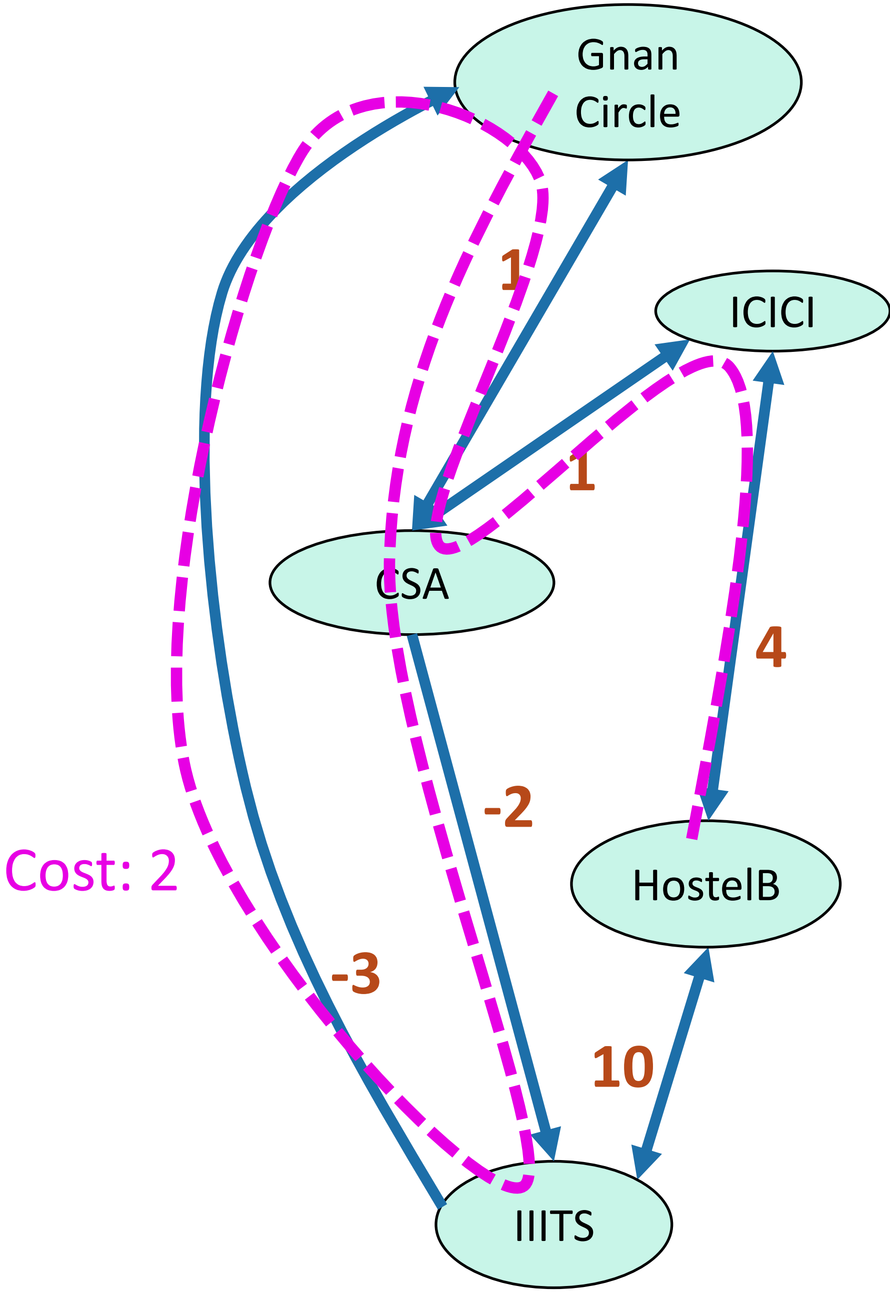
Wait a second…

- What is the shortest path from the Gnan Circle to the HostelB?



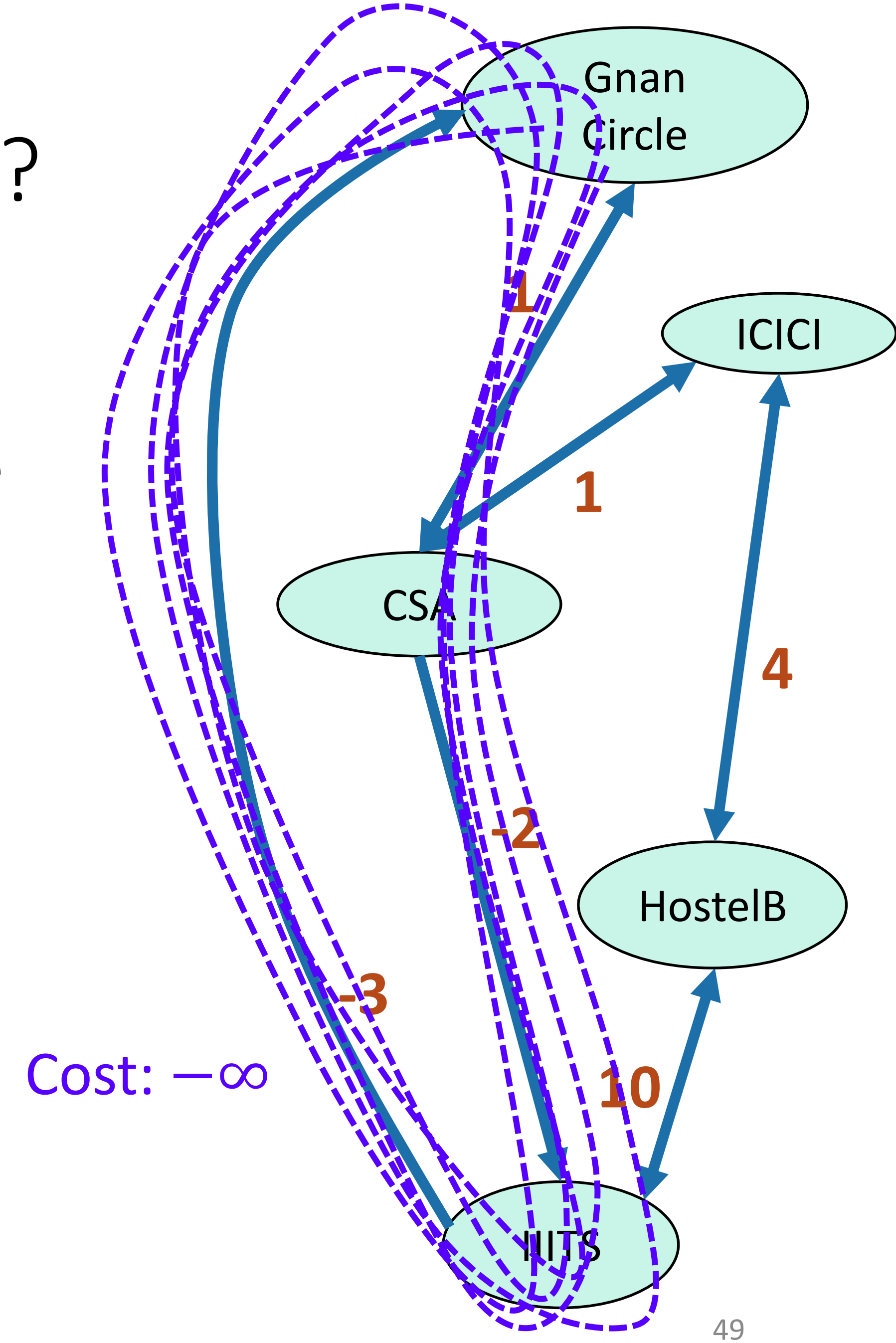
Wait a second…

- What is the shortest path from the Gnan Circle to the HostelB?



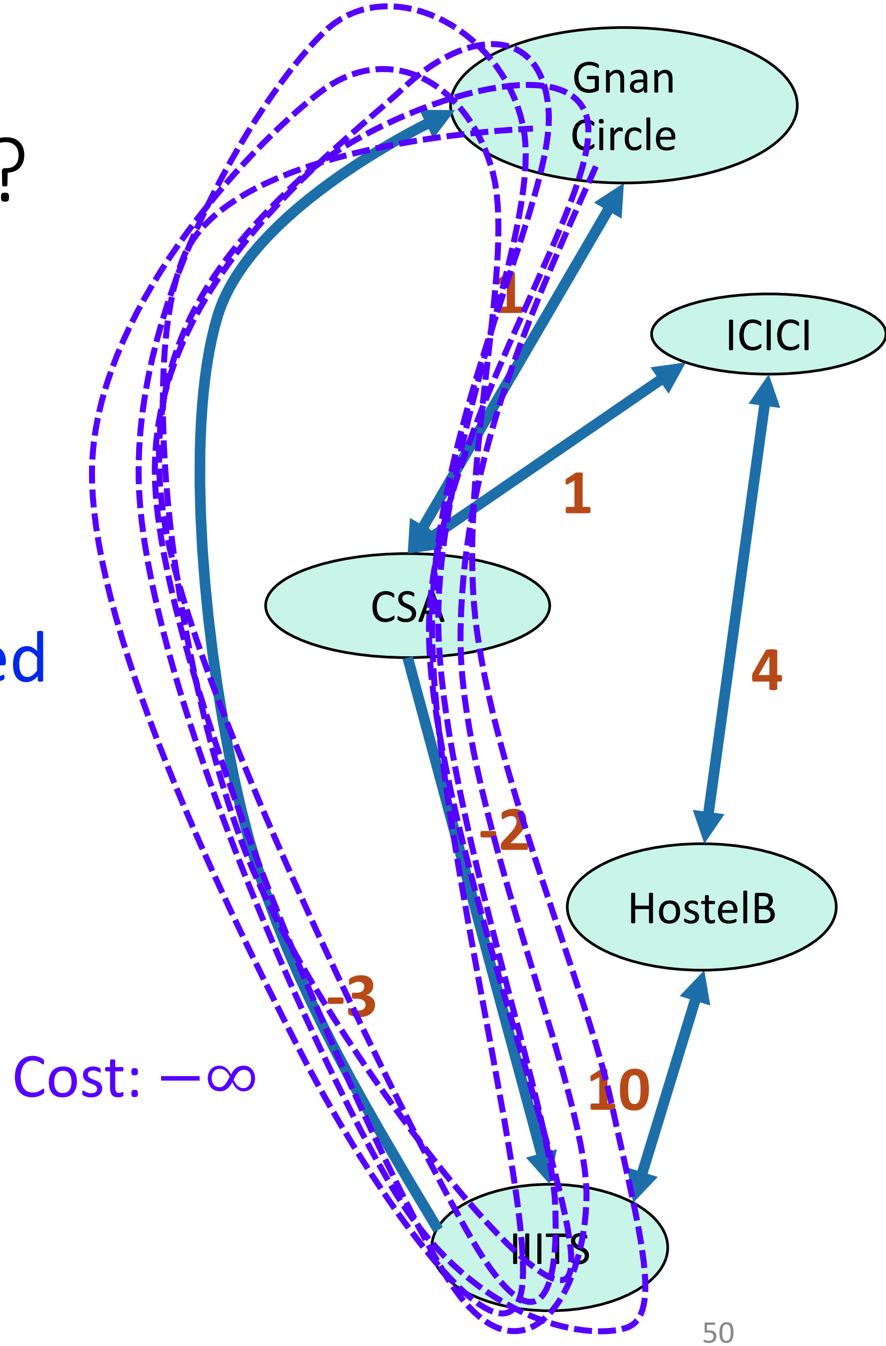
Negative edge weights?

- What is the shortest path from the Gnan Circle to the HostelB?



Negative edge weights?

- What is the shortest path from the Gnan Circle to the HostelB?
- Shortest paths aren't defined if there are negative cycles!



Bellman-Ford and negative edge weights

- Bellman-Ford works with negative edge weights…as long as there are not negative cycles.
  - A negative cycle is a path with the same start and end vertex whose cost is negative.
- However, Bellman-Ford can detect negative cycles.

Back to the correctness

- Does it work?
  - Yes
  - Idea to the right.

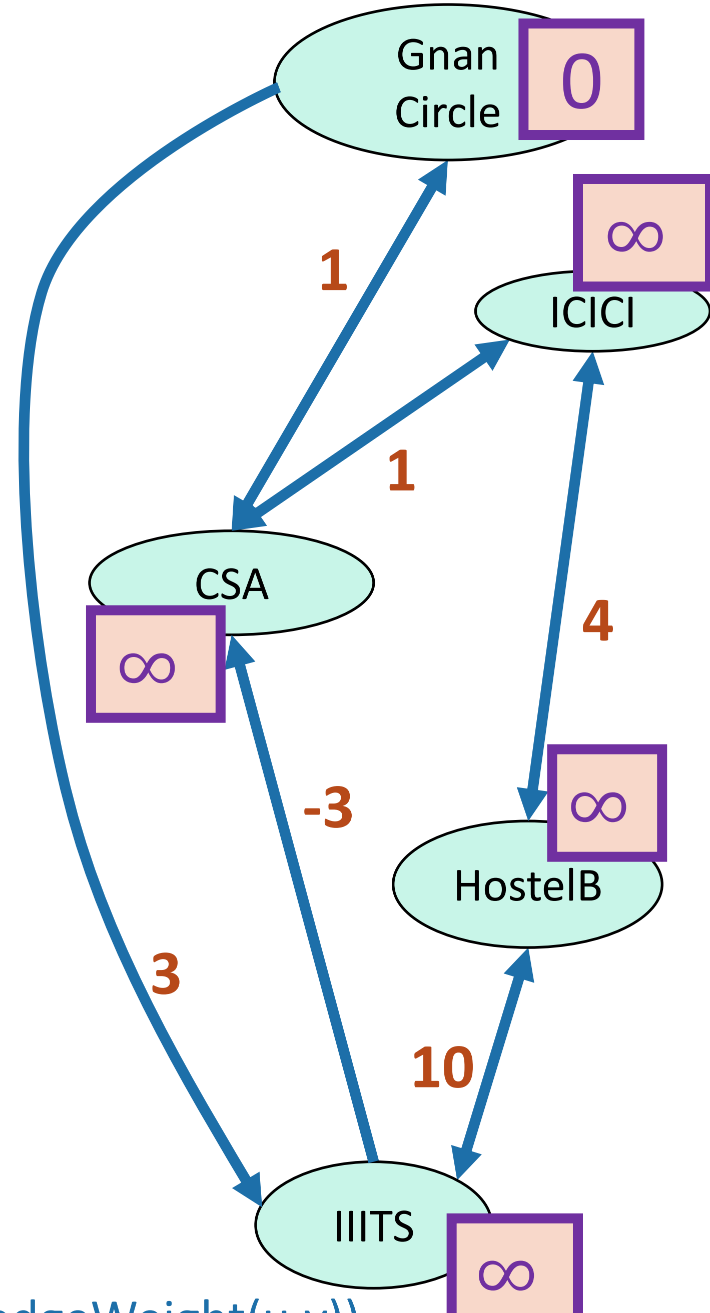
	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**25**
$d^{(2)}$	0	1	**2**	**45**	**23**
$d^{(3)}$	0	1	2	**6**	23
$d^{(4)}$	0	1	2	6	23

Idea: proof by induction.
Inductive Hypothesis:
$d^{(i)}[v]$ is equal to the cost of the shortest path between s and v **with at most i edges**.
Conclusion:
$d^{(n-1)}[v]$ is equal to the cost of the shortest simple path between s and v. **(Since all simple paths have at most n-1 edges).**

If there are negative cycles, then non-simple paths matter!
So the proof breaks for negative cycles.

Negative edge weights

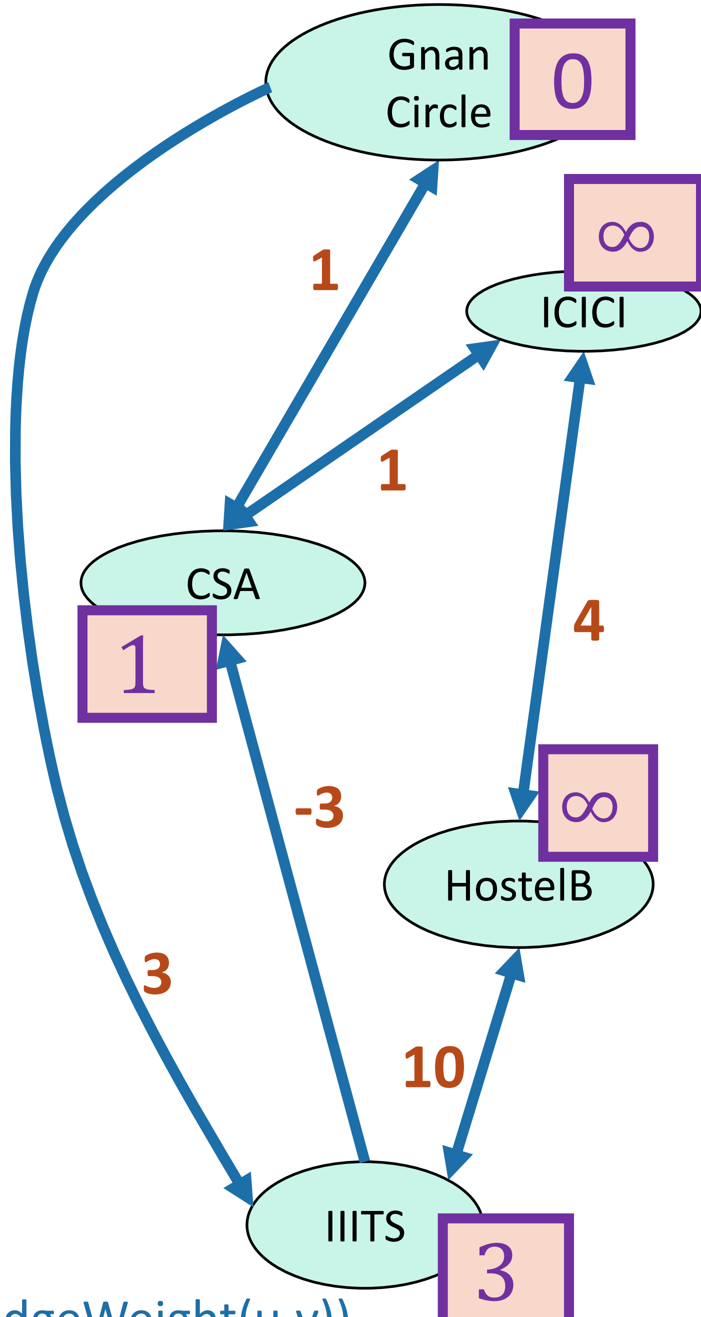
	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞



- **For** i=0,…,n-2:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\ ,\ d^{(i+1)}[v],\ d^{(i)}[u] + \text{edgeWeight}(u,v))$

Negative edge weights

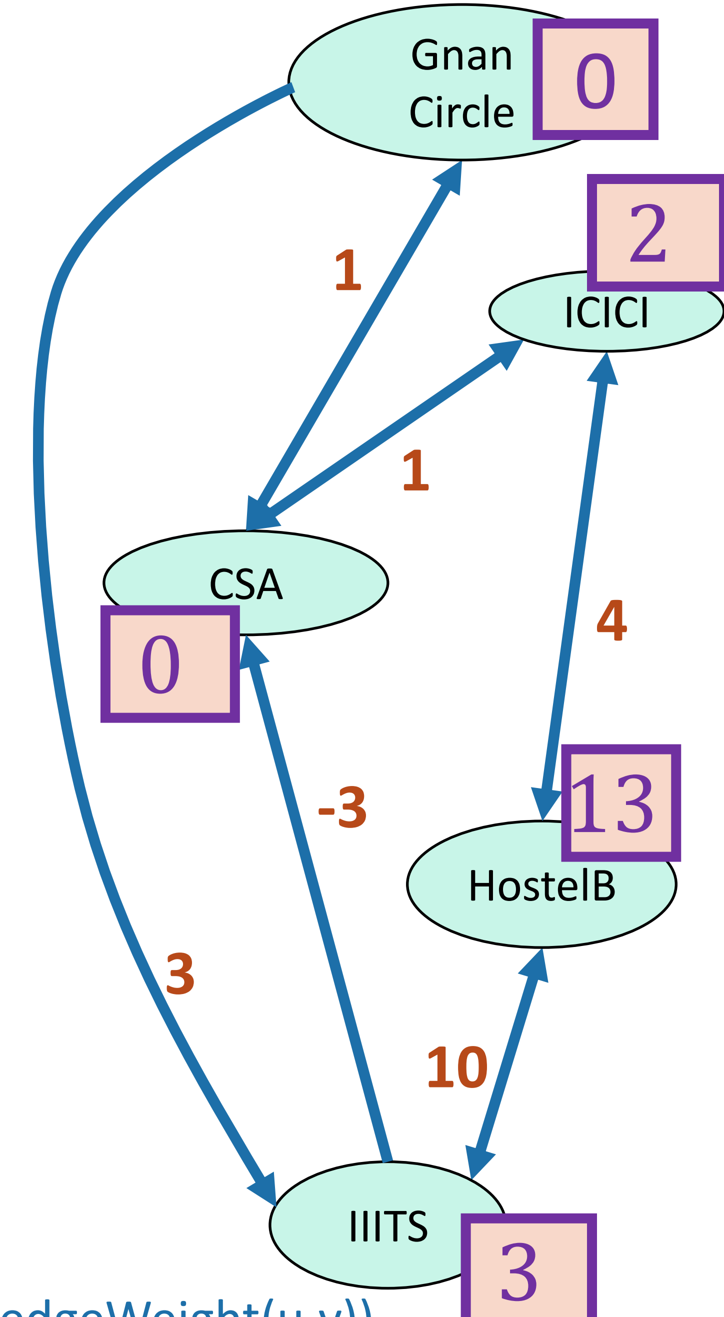
	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**3**



- **For** i=0,…,n-2:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\ ,\ d^{(i+1)}[v],\ d^{(i)}[u] + \mathrm{edgeWeight}(u,v))$

Negative edge weights

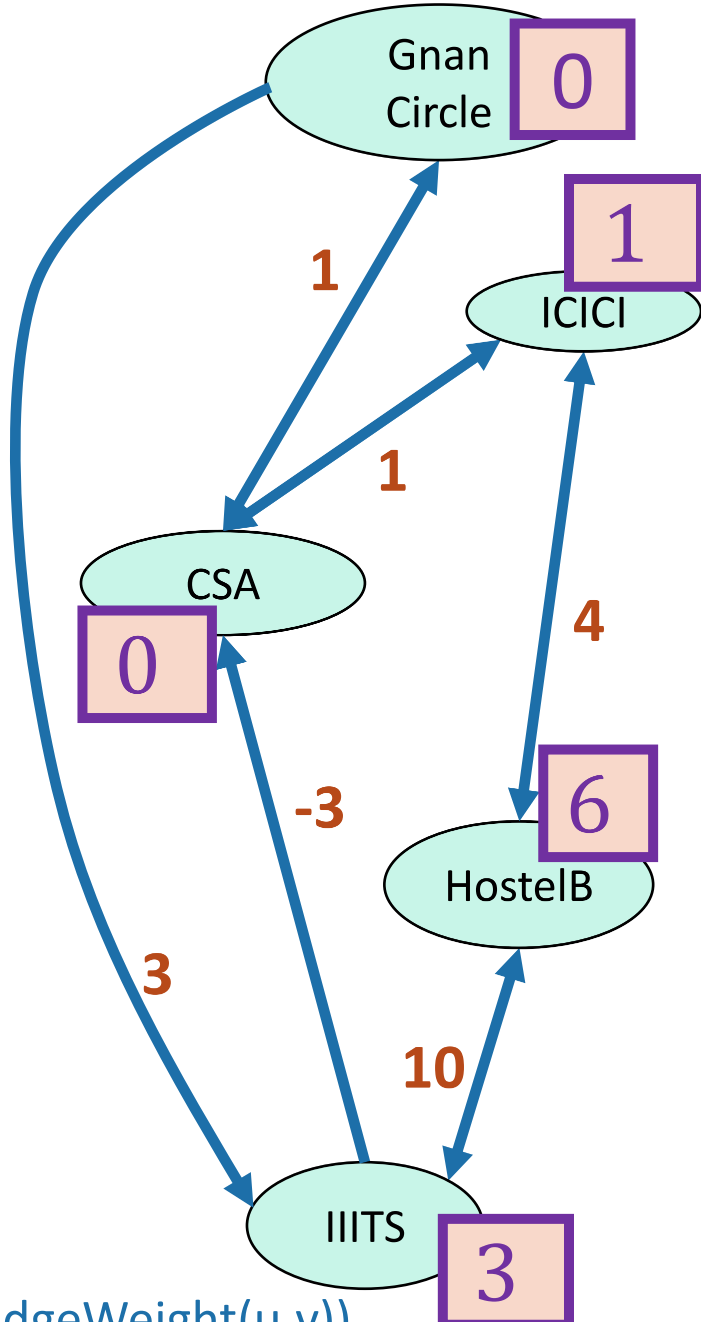
	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**3**
$d^{(2)}$	0	**0**	**2**	**13**	3



- **For** i=0,…,n-2:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\ ,\ d^{(i+1)}[v],\ d^{(i)}[u] + \text{edgeWeight}(u,v))$

Negative edge weights

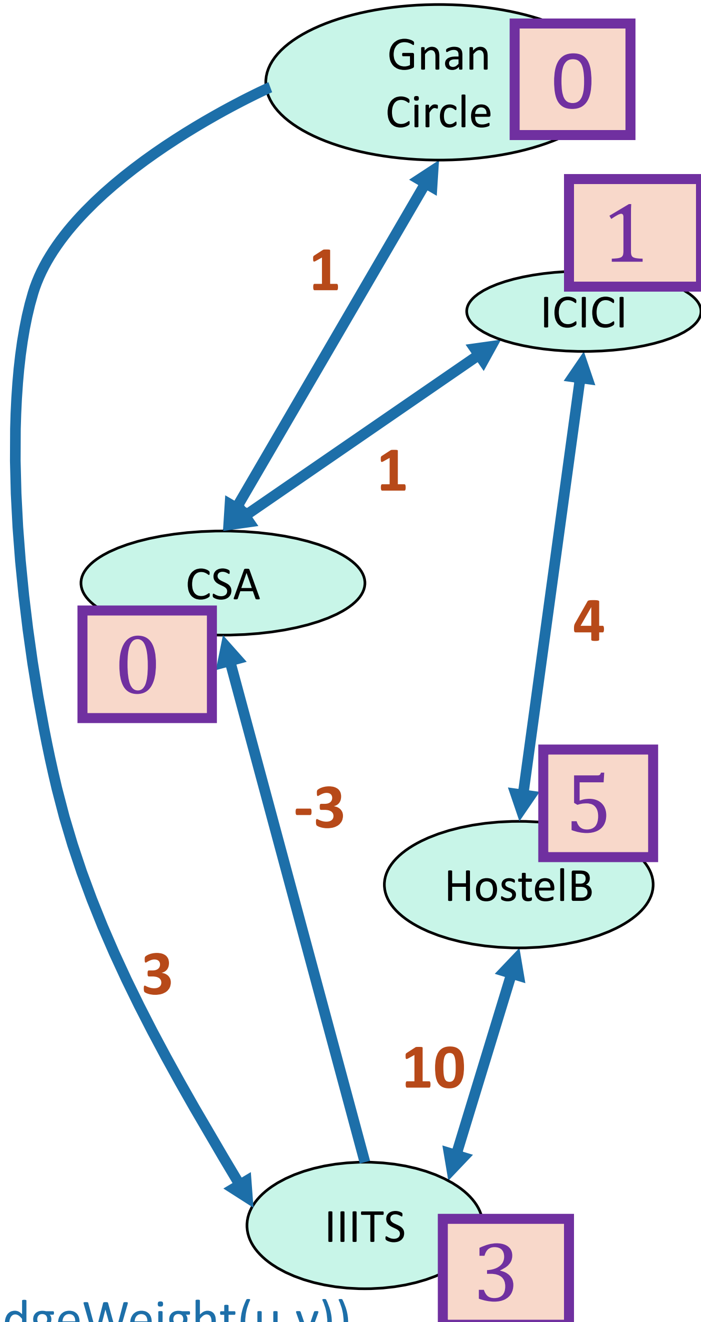
	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**3**
$d^{(2)}$	0	**0**	**2**	**13**	3
$d^{(3)}$	0	0	**1**	**6**	3



- **For** i=0,…,n-2:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\ ,\ d^{(i+1)}[v],\ d^{(i)}[u] + \text{edgeWeight}(u,v))$

Negative edge weights

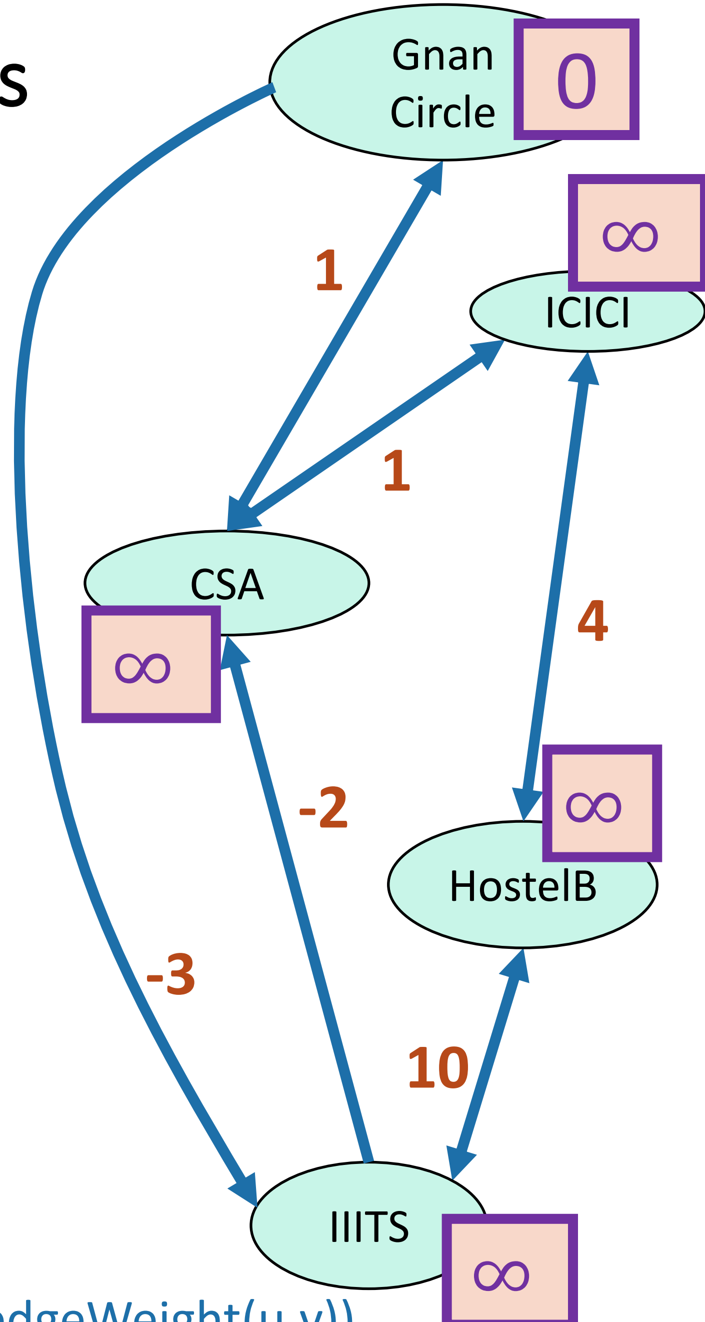
	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**3**
$d^{(2)}$	0	**0**	**2**	**13**	3
$d^{(3)}$	0	0	**1**	**6**	3
$d^{(4)}$	0	0	1	**5**	3



- **For** i=0,…,n-2:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\ ,\ d^{(i+1)}[v],\ d^{(i)}[u] + \text{edgeWeight}(u,v))$

Bellman-Ford with negative cycles

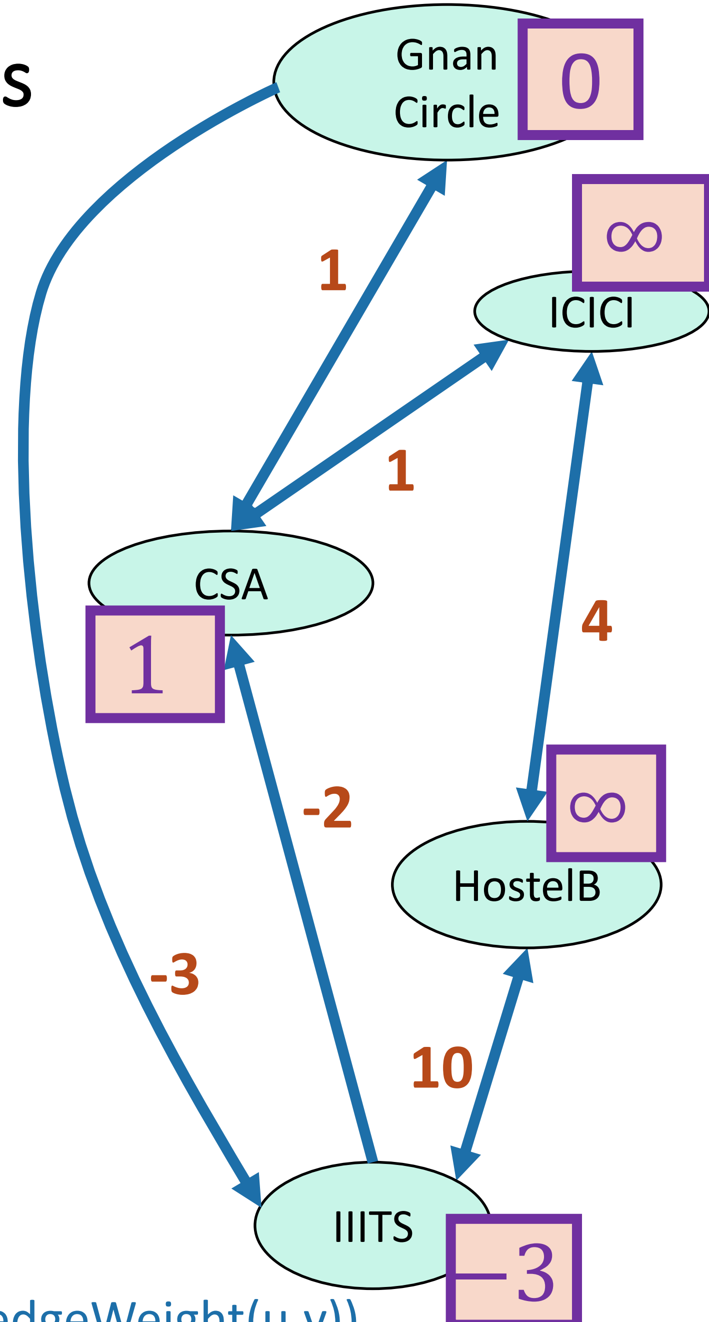
	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞



- **For** i=0,…,n-2:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\ ,\ d^{(i+1)}[v],\ d^{(i)}[u] + \text{edgeWeight}(u,v))$

Bellman-Ford with negative cycles

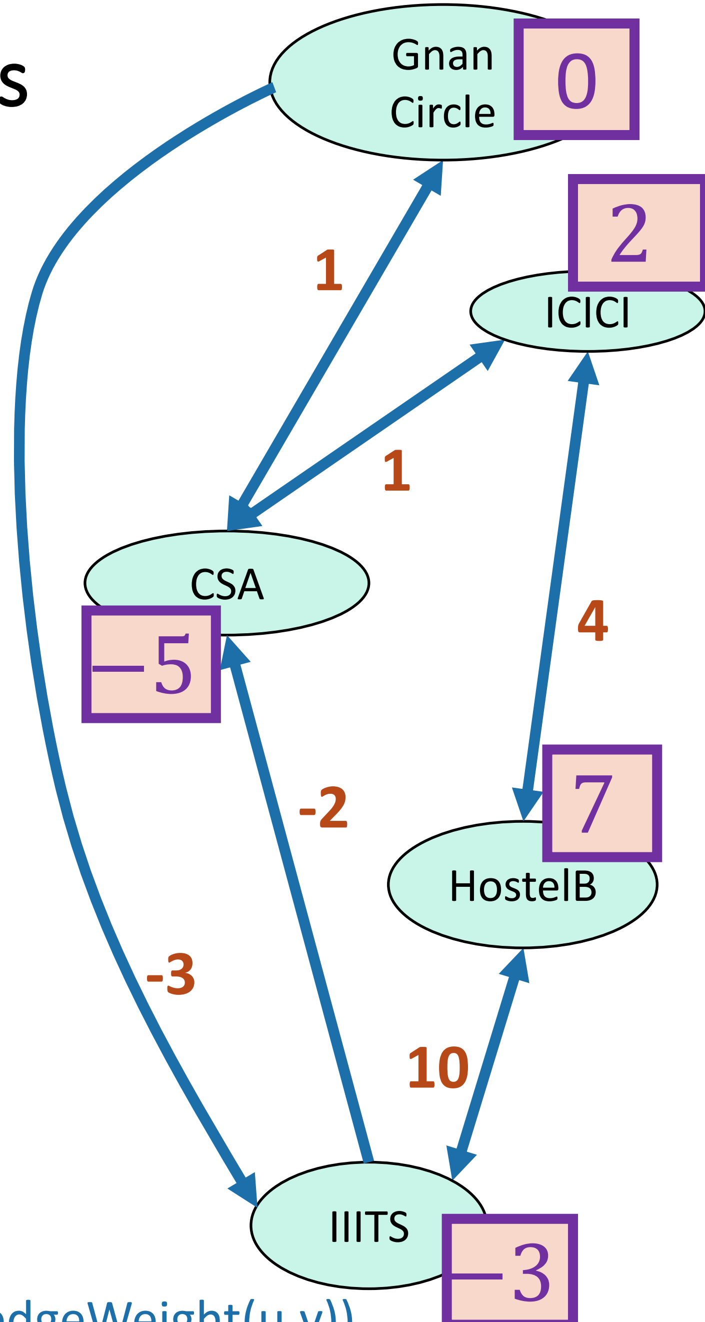
	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**-3**



- **For** i=0,…,n-2:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v],\ d^{(i+1)}[v],\ d^{(i)}[u] + \text{edgeWeight}(u,v))$

Bellman-Ford with negative cycles

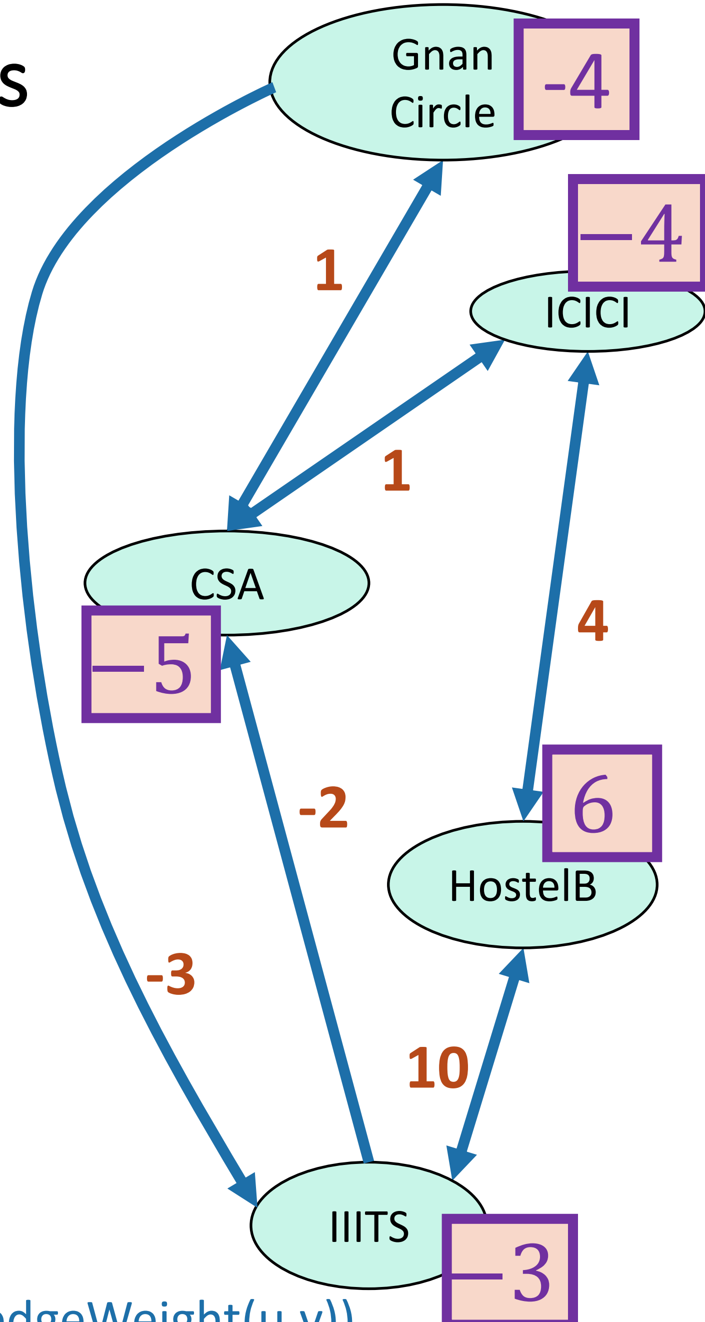
	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**-3**
$d^{(2)}$	0	**-5**	**2**	**7**	**-3**



- **For** i=0,…,n-2:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\ ,\ d^{(i+1)}[v],\ d^{(i)}[u] + \text{edgeWeight}(u,v))$

Bellman-Ford with negative cycles

	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**-3**
$d^{(2)}$	0	**-5**	**2**	**7**	**-3**
$d^{(3)}$	**-4**	-5	**-4**	**6**	**-3**

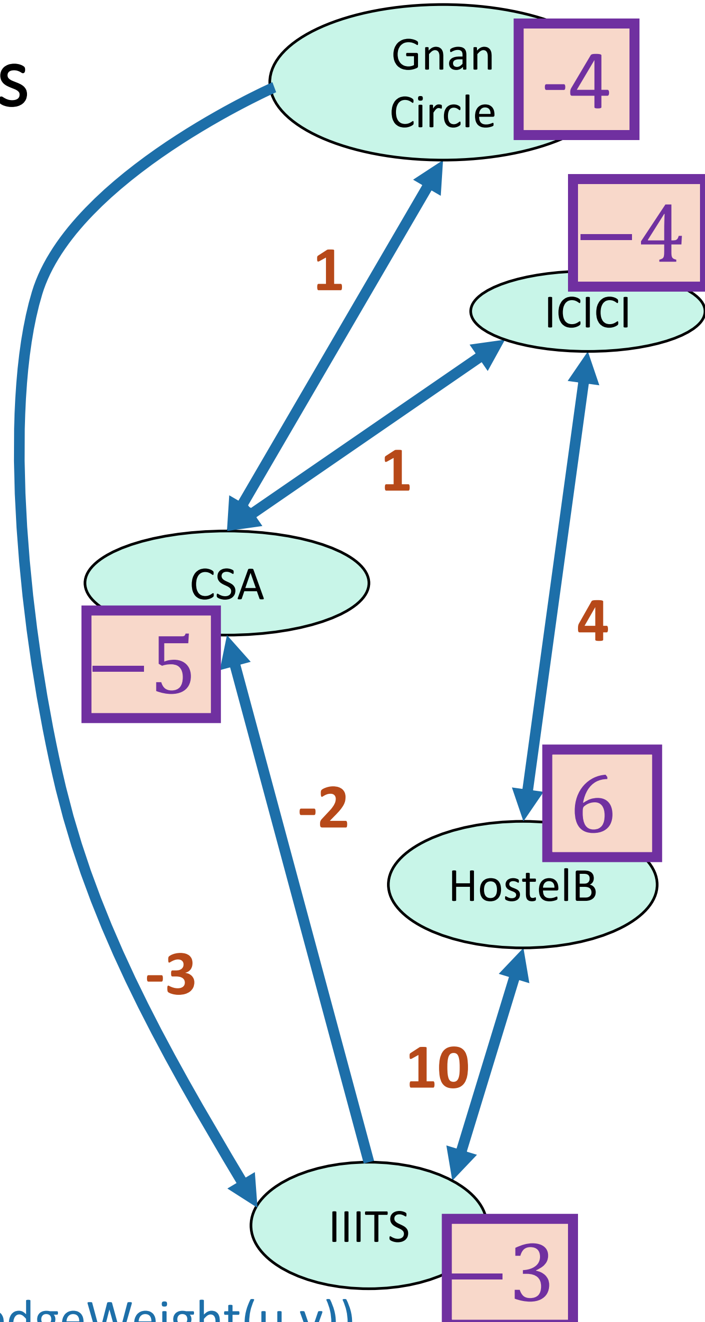


- **For** i=0,…,n-2:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\ ,\ d^{(i+1)}[v],\ d^{(i)}[u] + \text{edgeWeight}(u,v))$

Bellman-Ford with negative cycles

	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**-3**
$d^{(2)}$	0	**-5**	**2**	**7**	**-3**
$d^{(3)}$	**-4**	-5	**-4**	**6**	**-3**

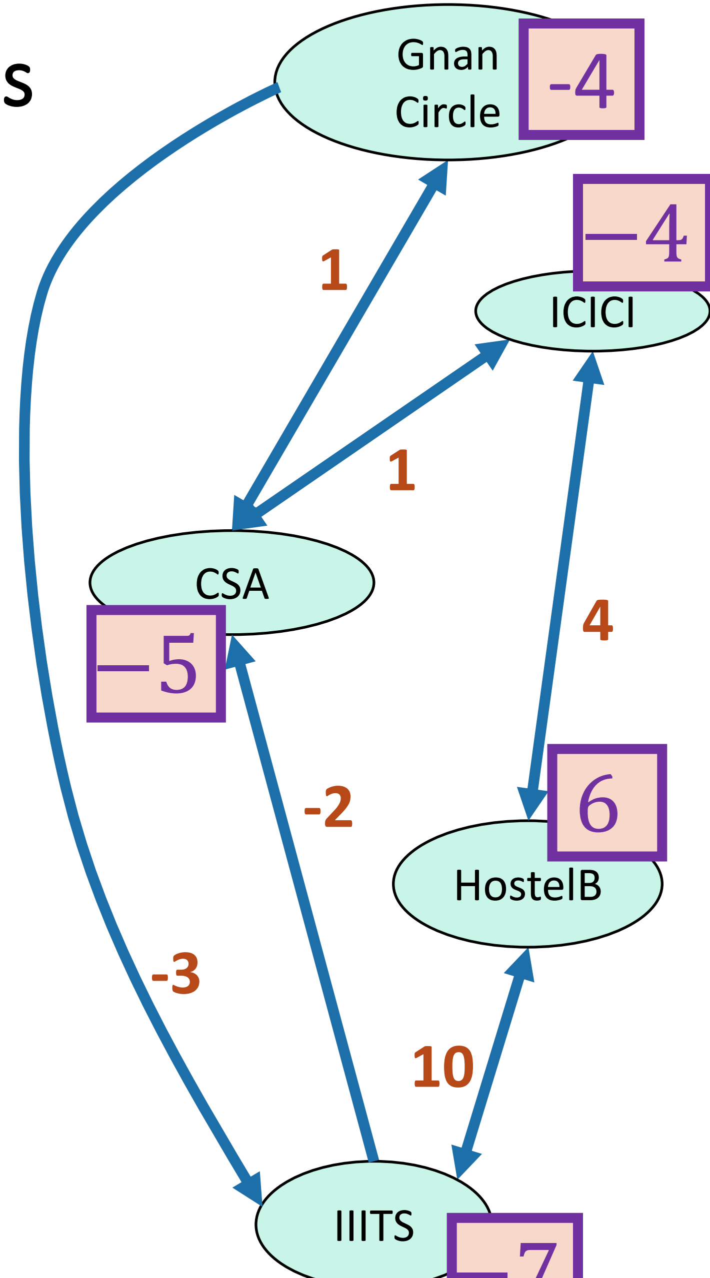
This is not looking good!



- **For** i=0,…,n-2:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\ ,\ d^{(i+1)}[v],\ d^{(i)}[u] + \text{edgeWeight}(u,v))$

Bellman-Ford with negative cycles

	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**-3**
$d^{(2)}$	0	**-5**	**2**	**7**	**-3**
$d^{(3)}$	**-4**	-5	**-4**	**6**	**-3**
$d^{(4)}$	-4	-5	-4	6	**-7**

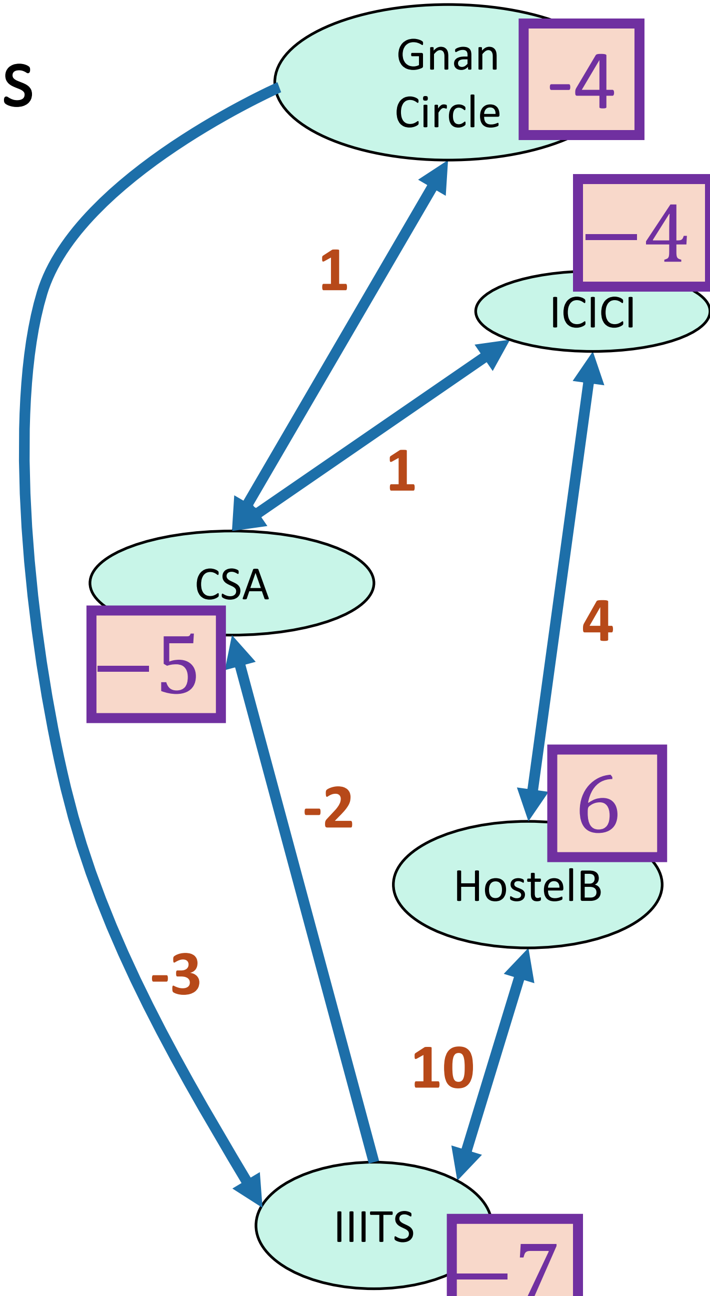


- **For** i=0,…,n-1:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\ ,\ d^{(i+1)}[v],\ d^{(i)}[u] + \text{edgeWeight}(u,v))$

Bellman-Ford with negative cycles

	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**-3**
$d^{(2)}$	0	**-5**	**2**	**7**	**-3**
$d^{(3)}$	**-4**	-5	**-4**	**6**	**-3**
$d^{(4)}$	-4	-5	-4	6	**-7**

But **we can tell** that it's not looking good:



- **For** i=0,…,n-1:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\,,\, d^{(i+1)}[v],\, d^{(i)}[u] + \text{edgeWeight}(u,v))$

Bellman-Ford with negative cycles

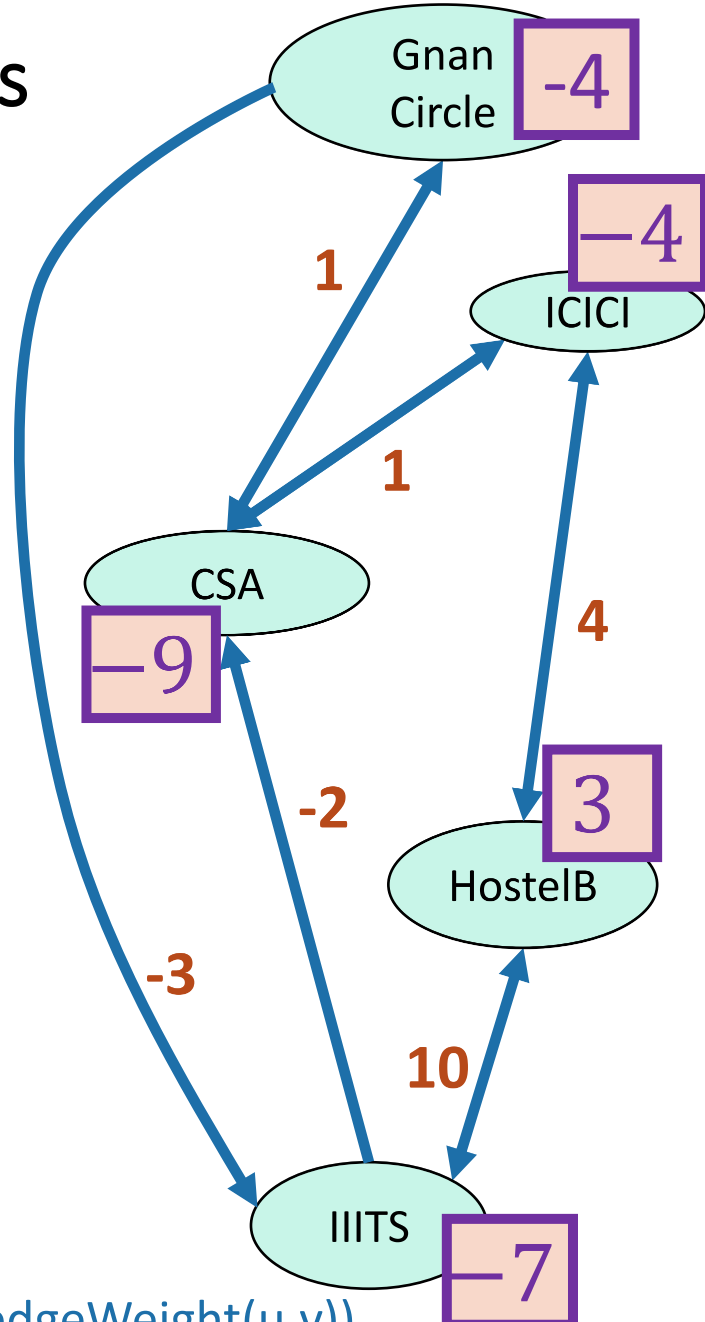
	Gnan	CSA	ICICI	HostelB	IIITS
$d^{(0)}$	0	∞	∞	∞	∞
$d^{(1)}$	0	**1**	∞	∞	**-3**
$d^{(2)}$	0	**-5**	**2**	**7**	**-3**
$d^{(3)}$	**-4**	-5	**-4**	**6**	**-3**
$d^{(4)}$	-4	-5	-4	6	**-7**

But **we can tell** that it's not looking good:

$d^{(5)}$	-4	**-9**	-4	**3**	-7

Some stuff changed!

- **For** i=0,…,n-1:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\ ,\ d^{(i+1)}[v],\ d^{(i)}[u] + \text{edgeWeight}(u,v))$



How Bellman-Ford deals with negative cycles

- If there are no negative cycles:
  - Everything works as it should.
  - The algorithm stabilizes after n-1 rounds.
  - Note: Negative ***edges*** are okay!!
- If there are negative cycles:
  - Not everything works as it should…
    - Note: it couldn't possibly work, since shortest paths aren't well-defined if there are negative cycles.
  - The d[v] values will keep changing.
- Solution:
  - Go one round more and see if things change.
    - If so, return NEGATIVE CYCLE ☹

Bellman-Ford algorithm

Bellman-Ford*(G,s):

- $d^{(0)}[v] = \infty$ for all v in V
- $d^{(0)}[s] = 0$
- **For** i=0,…,n-1:
  - **For** u in V:
    - **For** v in u.neighbors:
      - $d^{(i+1)}[v] \leftarrow \min(d^{(i)}[v]\ ,\ d^{(i+1)}[v],\ d^{(i)}[u] + \text{edgeWeight}(u,v))$
- If $d^{(n-1)}$!= $d^{(n)}$:
  - **Return** NEGATIVE CYCLE ☹
- Otherwise, dist(s,v) = $d^{(n-1)}[v]$

Summary

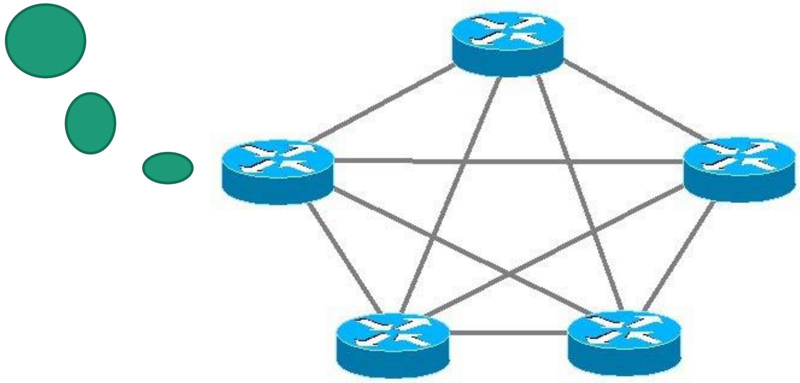
- The Bellman-Ford algorithm:
  - Finds shortest paths in weighted graphs with negative edge weights
  - runs in time O(nm) on a graph G with n vertices and m edges.
- If there are no negative cycles in G:
  - the Bellman-Ford algorithm terminates with $d^{(n-1)}[v] = d(s,v)$.
- If there are negative cycles in G:
  - the Bellman-Ford algorithm returns negative cycle.

Bellman-Ford is also used in practice.

- eg, Routing Information Protocol (RIP) uses something like Bellman-Ford.
  - Older protocol, not used as much anymore.

- Each router keeps a **table** of distances to every other router.
- Periodically we do a Bellman-Ford update.
- This means that if there are changes in the network, this will propagate. (maybe slowly…)

Destination	Cost to get there	Send to whom?
172.16.1.0	34	172.16.1.1
10.20.40.1	10	192.168.1.2
10.155.120.1	9	10.13.50.0



Recap: shortest paths

- Breadth-First Search:
    - (+) O(n+m)
    - (-) only unweighted graphs
- Dijkstra's algorithm:
    - (+) weighted graphs
    - (+) O(nlog(n) + m) if you implement it right.
    - (-) no negative edge weights
    - (-) very "centralized" (need to keep track of all the vertices to know which to update).
- The Bellman-Ford algorithm:
    - (+) weighted graphs, even with negative weights
    - (+) can be done in a distributed fashion, every vertex using only information from its neighbors.
    - (-) O(nm), i.e., Slower than Dijkstra's algorithm

Acknowledgement

- Stanford University