NPTEL MOOC,JAN-FEB 2015
Week 4, Module 3

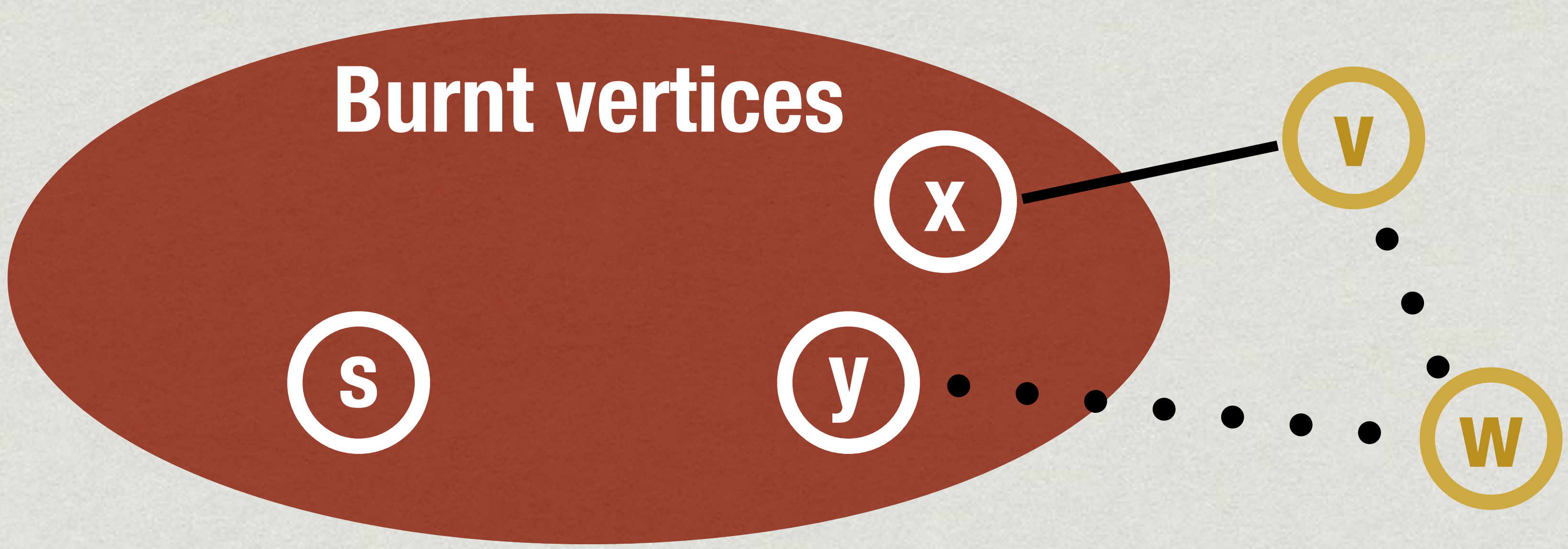
DESIGN AND ANALYSIS OF ALGORITHMS

Negative edges: Bellman-Ford algorithm

MADHAVAN MUKUND, CHENNAI MATHEMATICAL INSTITUTE
http://www.cmi.ac.in/~madhavan

Correctness for Dijsktra's algorithm

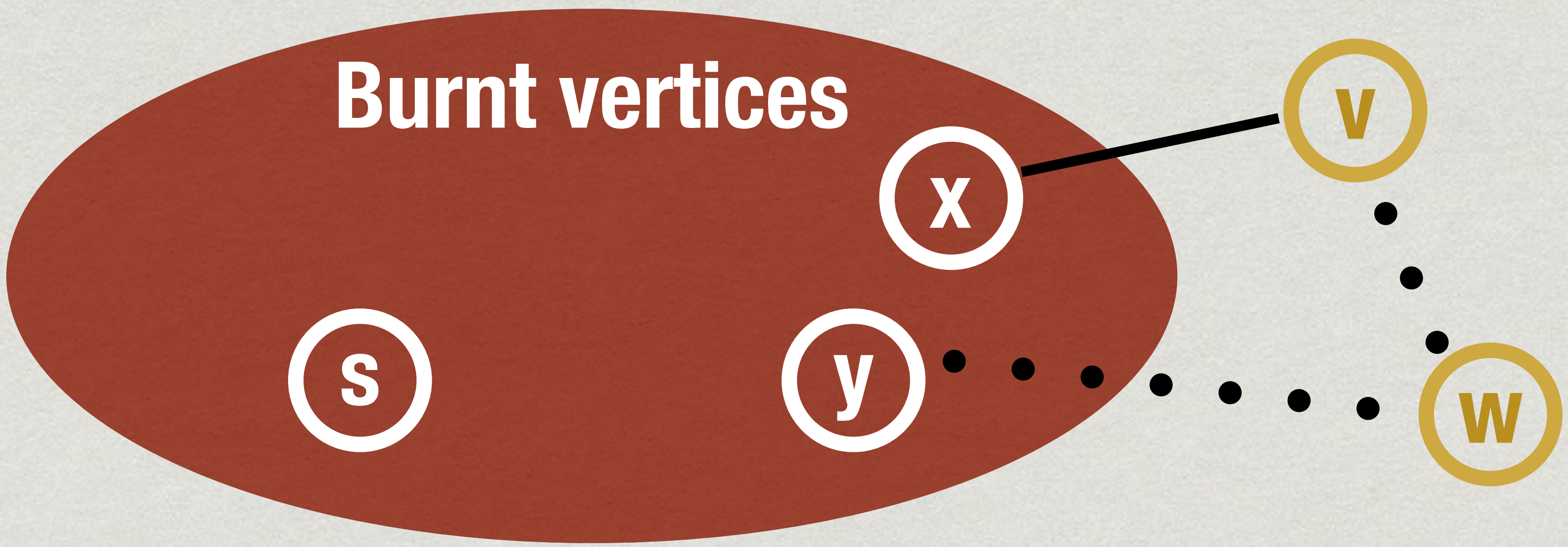
- By induction, assume we have identified shortest paths to all vertices already burnt



- Next vertex to burn is v, via x
- Cannot later find a shorter path from y to w to v

Negative weights

- Our correctness argument is no longer valid



- Next vertex to burn is v, via x
- Might find a shorter path later with negative weights from y to w to v

Negative weights …

- **Negative cycle**: loop with a negative total weight
  - Problem is not well defined with negative cycles
  - Repeatedly traversing cycle pushes down cost without a bound
- With negative edges, but no negative cycles, shortest paths do exist

About shortest paths

- Shortest paths will never loop
  - Never visit the same vertex twice
  - At most length n-1
- Every prefix of a shortest path is itself a shortest path
  - Suppose the shortest path from s to t is

$$s \to v_1 \to v_2 \to v_3 \dots \to v_m \to t$$

  - Every prefix $s \to v_1 \to \dots \to v_r$ is a shortest path to v_r

Updating Distance( )

- When vertex j is "burnt", for each edge (j,k) update

  Distance(k) = min(Distance(k), Distance(j)+weight(j,k))

- Refer to this as update(j,k)
- Dijkstra's algorithm
  - When we compute update(j,k), Distance(j) is always guaranteed to be correct distance to j
- What can we say in general?

Properties of update(j,k)

```
update(j,k):
    Distance(k) = min(Distance(k), Distance(j)+weight(j,k))
```

- Distance(k) is no more than Distance(j)+weight(j,k)
- If Distance(j) is correct and j is the second-last node on shortest path to k, Distance(k) is correct
- Update is safe
  - Distance(k) never becomes "too small"
  - Redundant updates cannot hurt

Updating Distance( ) …

```
update(j,k):
  Distance(k) = min(Distance(k), Distance(j)+weight(j,k))
```

- Dijkstra's algorithm performs a particular "greedy" sequence of updates
  - Computes shortest paths without negative weights
- With negative edges, this sequence does not work
- Is there some sequence that does work?

Updating distance( ) …

- Suppose the shortest path from s to t is

$$s \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \ldots \rightarrow v_m \rightarrow t$$

- If our update sequence includes …,update(s,v_1), …,update(v_1,v_2),…,update(v_2,v_3),…,update(v_m,t),…, in that order, Distance(t) will be computed correctly
  - If Distance(j) is correct and j is the second-last node on shortest path to k, Distance(k) is correct after update(j,k)

Bellman-Ford algorithm

- Initialize Distance(s) = 0, Distance(u) = ∞ for all other vertices
- Update all edges n-1 times!

Bellman-Ford algorithm

- Initialize Distance(s) = 0, Distance(u) = ∞ for all other vertices
- Update all edges n-1 times!

Iteration 1
…
update(s,v_1)
…
update(v_1,v_2)
…
update(v_2,v_3)
…
update(v_m,t)
…

Bellman-Ford algorithm

- Initialize Distance(s) = 0, Distance(u) = ∞ for all other vertices
- Update all edges n-1 times!

Iteration 1	Iteration 2
…	…
update(s,v_1)	update(s,v_1)
…	…
update(v_1,v_2)	update(v_1,v_2)
…	…
update(v_2,v_3)	update(v_2,v_3)
…	…
update(v_m,t)	update(v_m,t)
…	…

Bellman-Ford algorithm

- Initialize Distance(s) = 0, Distance(u) = ∞ for all other vertices
- Update all edges n-1 times!

Iteration 1	Iteration 2	…
…	…	…
update(s,v_1)	update(s,v_1)	…
…	…	…
update(v_1,v_2)	update(v_1,v_2)	…
…	…	…
update(v_2,v_3)	update(v_2,v_3)	…
…	…	…
update(v_m,t)	update(v_m,t)	…
…	…	…

Bellman-Ford algorithm

- Initialize Distance(s) = 0, Distance(u) = ∞ for all other vertices
- Update all edges n-1 times!

Iteration 1	Iteration 2	…	Iteration n-1
…	…	…	…
update(s,v_1)	update(s,v_1)	…	update(s,v_1)
…	…	…	…
update(v_1,v_2)	update(v_1,v_2)	…	update(v_1,v_2)
…	…	…	…
update(v_2,v_3)	update(v_2,v_3)	…	update(v_2,v_3)
…	…	…	…
update(v_m,t)	update(v_m,t)	…	update(v_m,t)
…	…	…	…

Bellman-Ford algorithm

- Initialize Distance(s) = 0, Distance(u) = ∞ for all other vertices
- Update all edges n-1 times!

Iteration 1	Iteration 2	…	Iteration n-1
…	…	…	…
update(s,v_1)	update(s,v_1)	…	update(s,v_1)
…	…	…	…
update(v_1,v_2)	update(v_1,v_2)	…	update(v_1,v_2)
…	…	…	…
update(v_2,v_3)	update(v_2,v_3)	…	update(v_2,v_3)
…	…	…	…
update(v_m,t)	update(v_m,t)	…	update(v_m,t)
…	…	…	…

Bellman-Ford algorithm

- Initialize Distance(s) = 0, Distance(u) = ∞ for all other vertices
- Update all edges n-1 times!

Iteration 1	Iteration 2	…	Iteration n-1
…	…	…	…
update(s,v_1)	update(s,v_1)	…	update(s,v_1)
…	…	…	…
update(v_1,v_2)	update(v_1,v_2)	…	update(v_1,v_2)
…	…	…	…
update(v_2,v_3)	update(v_2,v_3)	…	update(v_2,v_3)
…	…	…	…
update(v_m,t)	update(v_m,t)	…	update(v_m,t)
…	…	…	…

Bellman-Ford algorithm

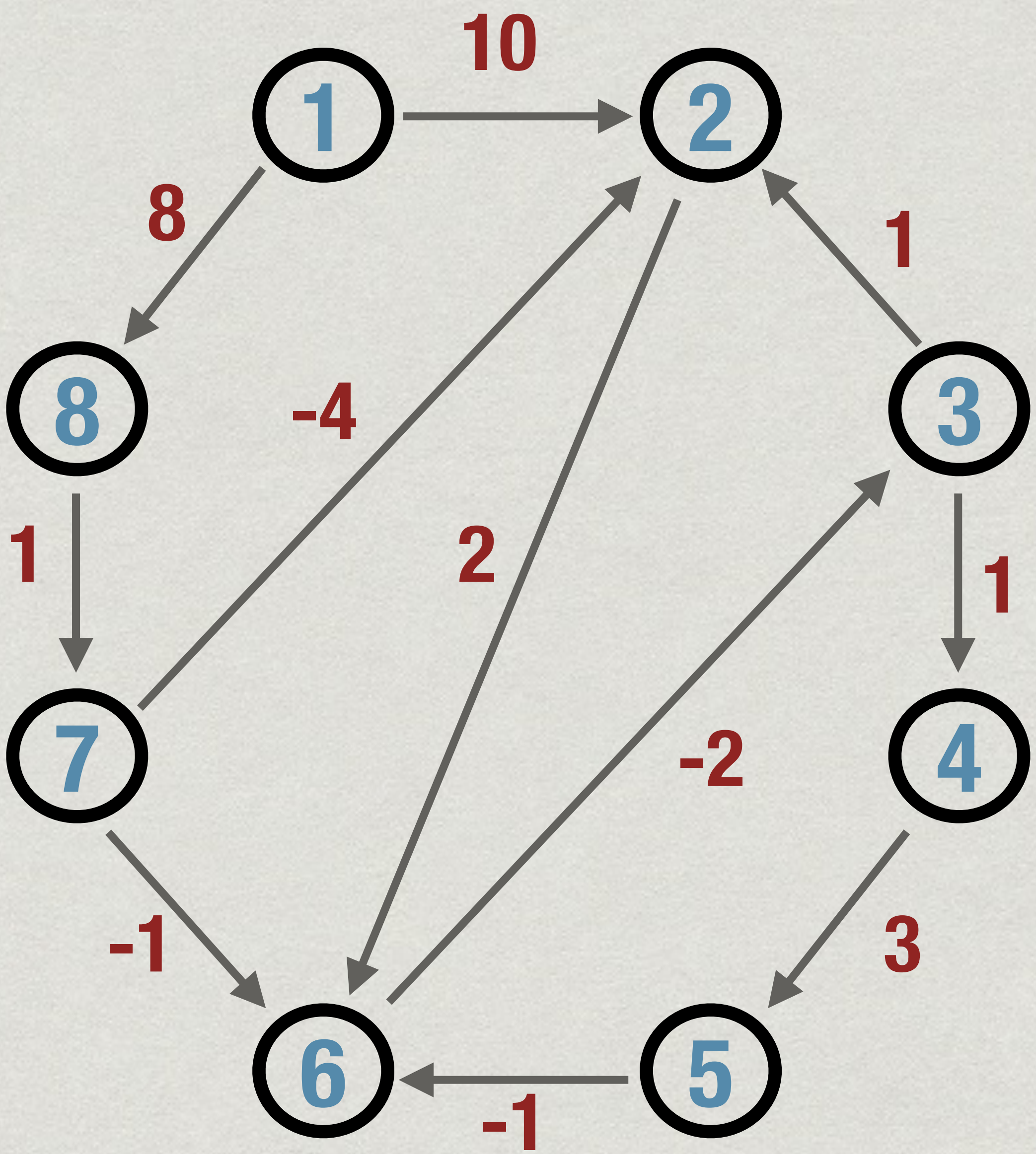
- Initialize Distance(s) = 0, Distance(u) = ∞ for all other vertices
- Update all edges n-1 times!

Iteration 1	Iteration 2	…	Iteration n-1
…	…	…	…
update(s,v_1)	update(s,v_1)	…	update(s,v_1)
…	…	…	…
update(v_1,v_2)	**update(v_1,v_2)**	…	update(v_1,v_2)
…	…	…	…
update(v_2,v_3)	update(v_2,v_3)	…	update(v_2,v_3)
…	…	…	…
update(v_m,t)	update(v_m,t)	…	**update(v_m,t)**
…	…	…	…

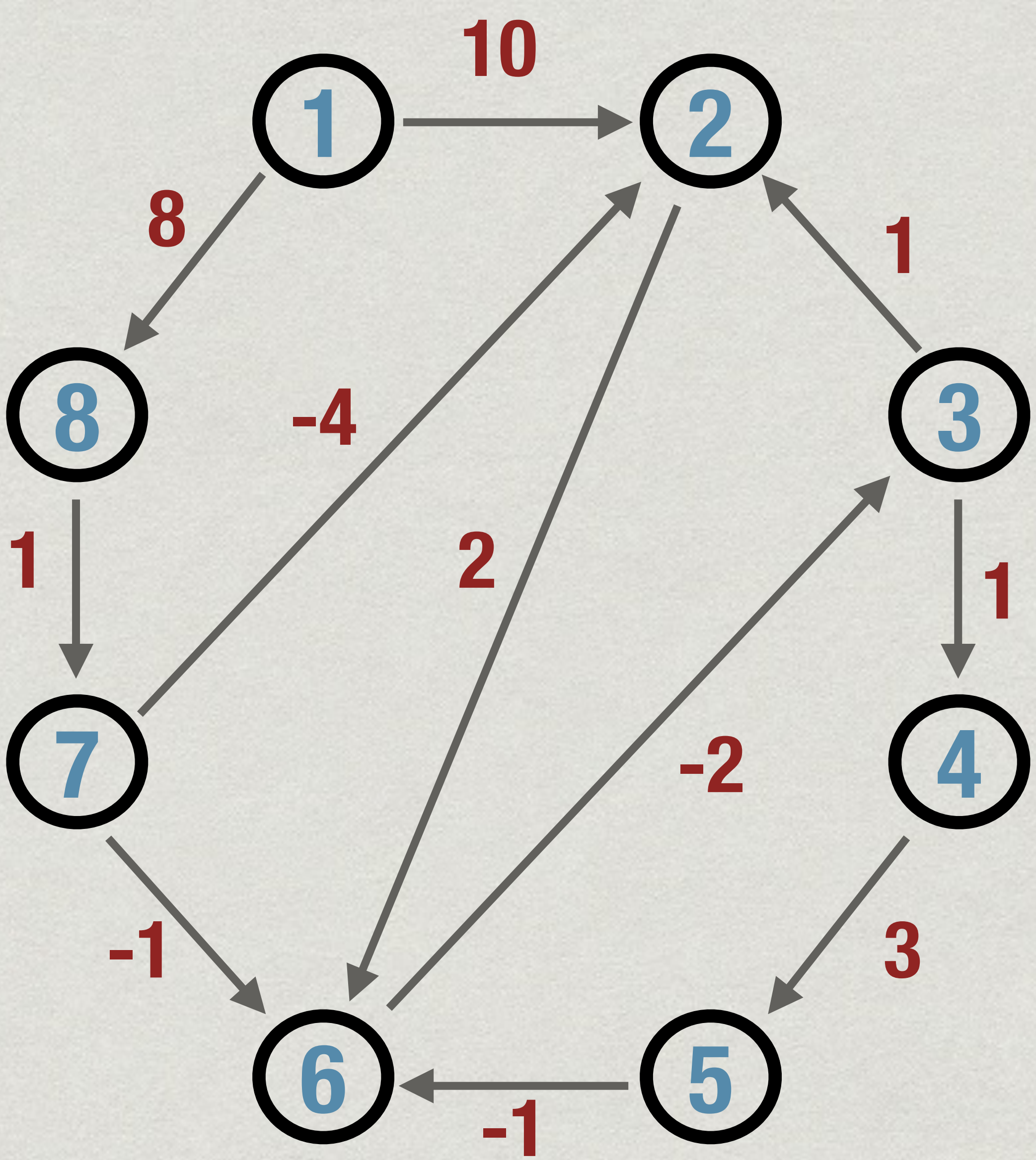
Bellman-Ford algorithm

```
function BellmanFord(s)//source s, with -ve weights

for i = 1 to n
  Distance[i] = infinity

Distance[s] = 0

for i = 1 to n-1 //repeat n-1 times
  for each edge(j,k) in E
    Distance(k) = min(Distance(k),
                      Distance(j) + weight(j,k))
```

Example

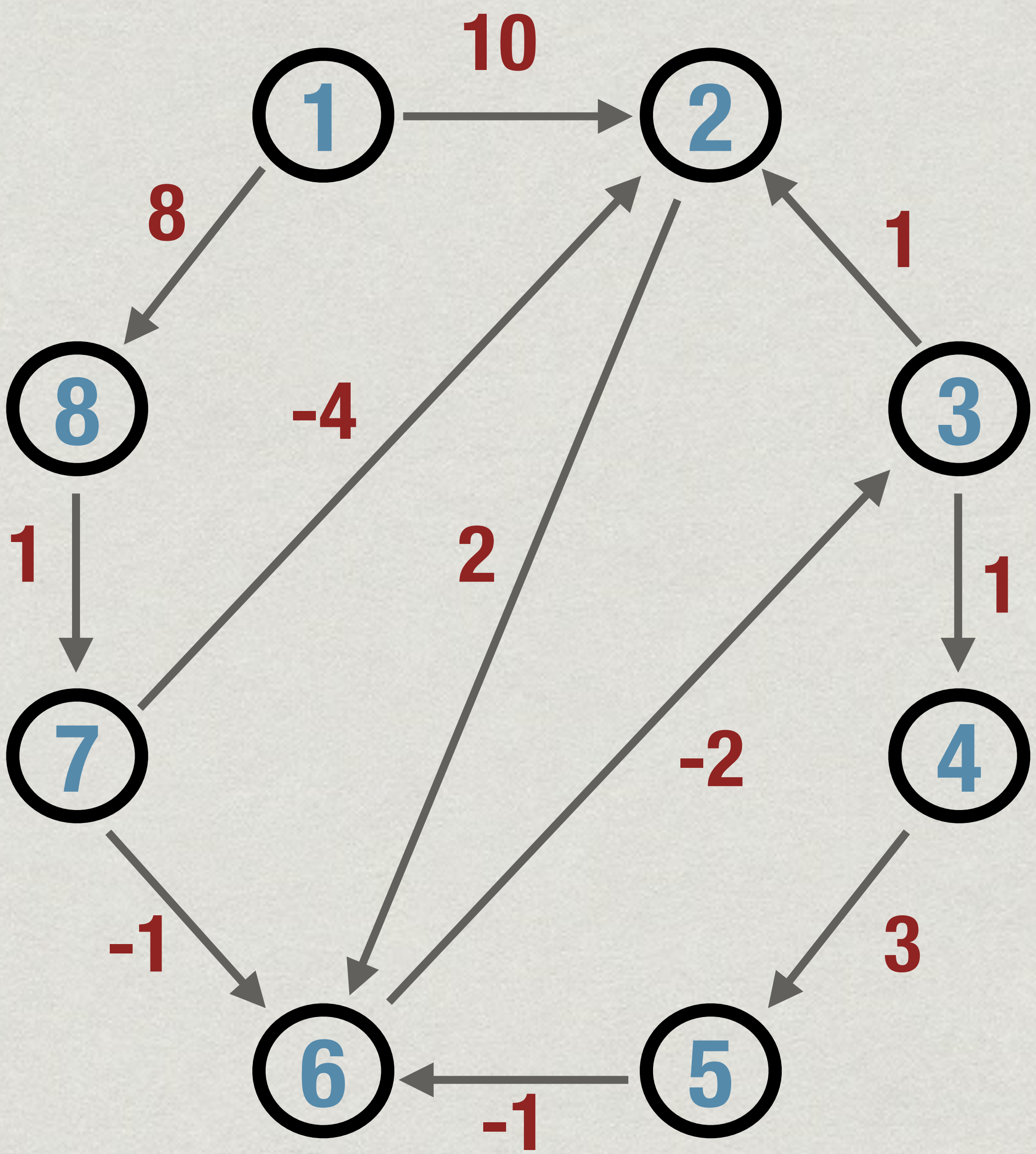


Example



Node
1
2
3
4
5
6
7
8

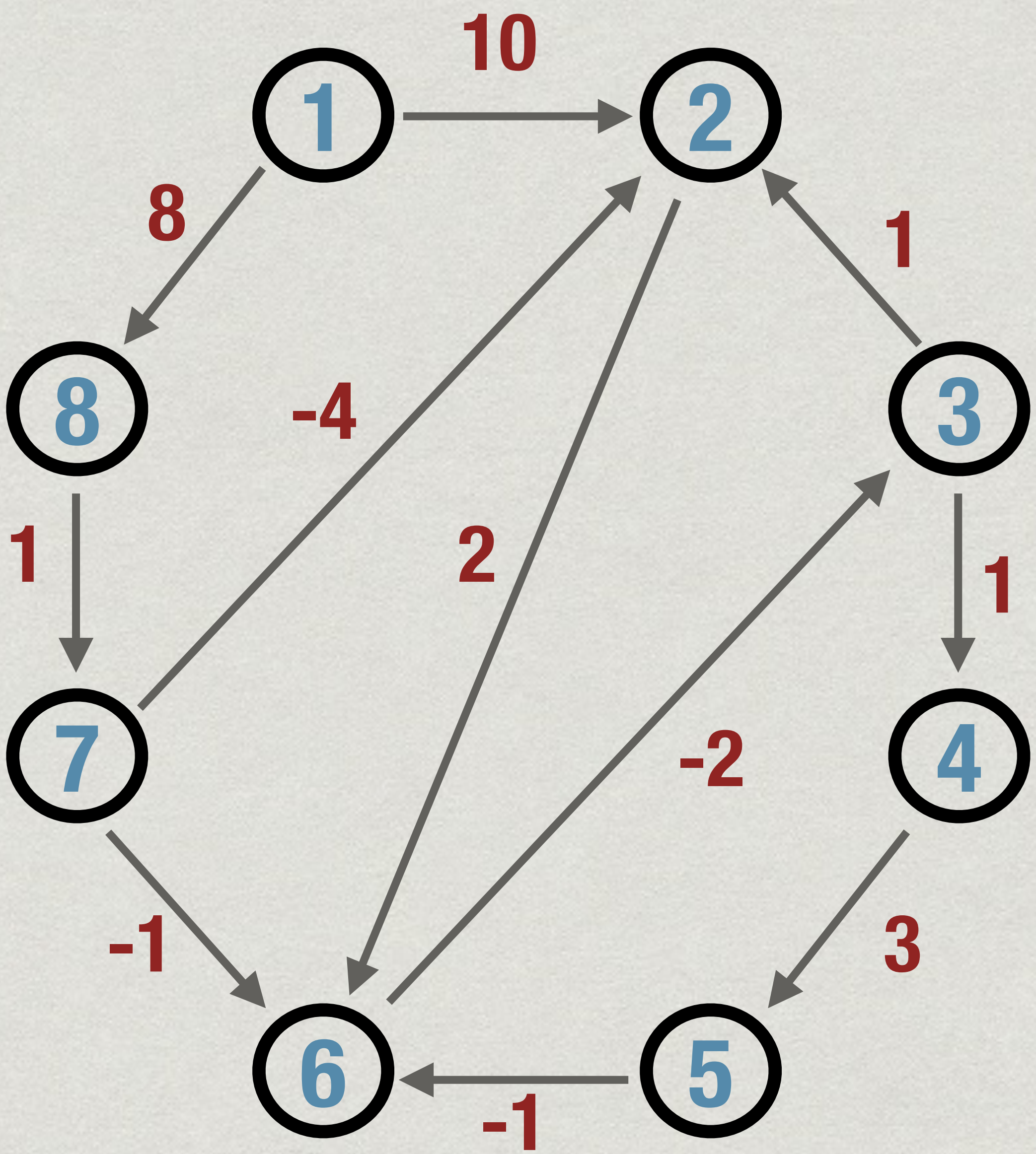
Example



Iteration

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

Example

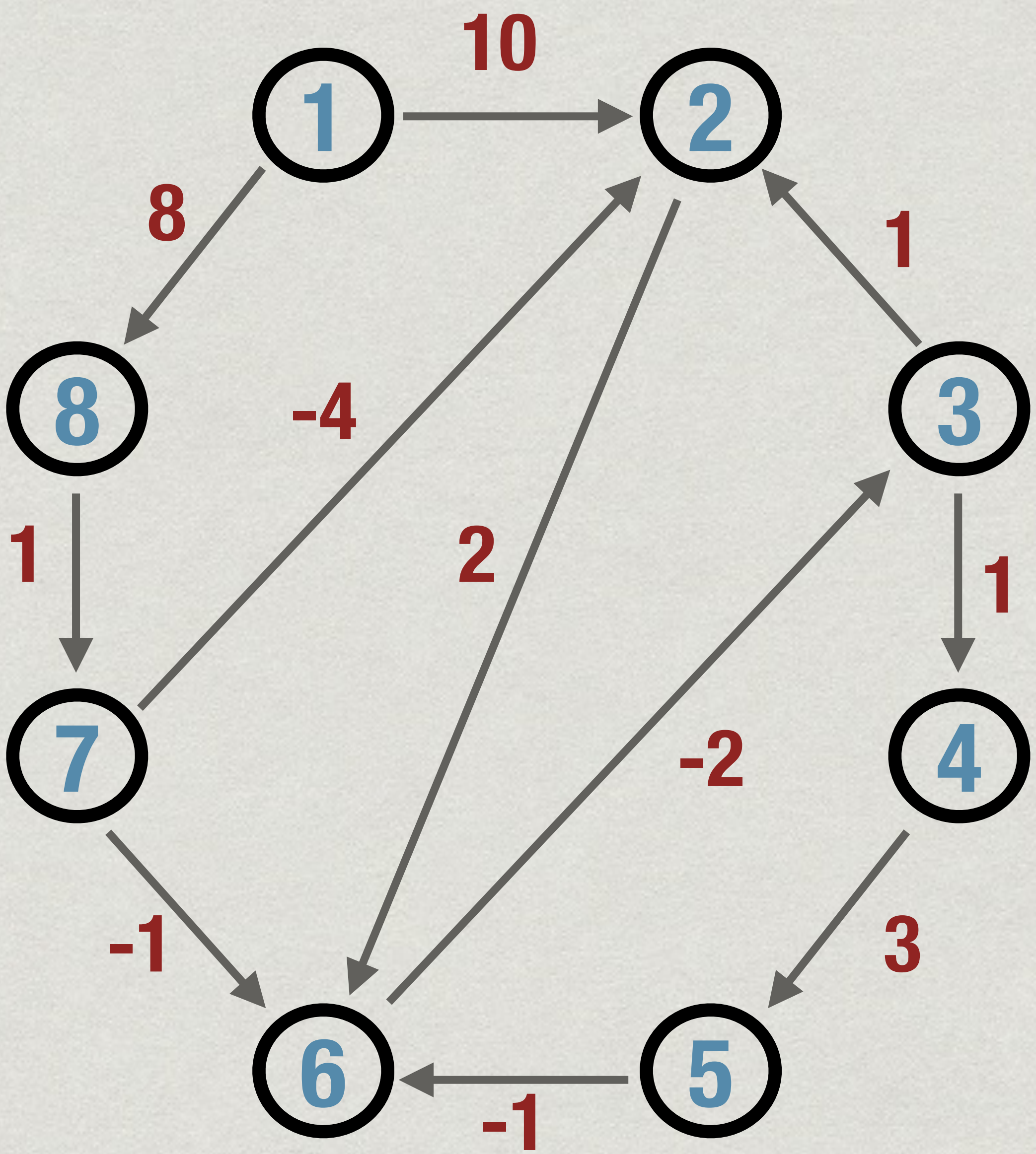


Iteration

Node

	0	1
1	0	0
2	∞	10
3	∞	∞
4	∞	∞
5	∞	∞
6	∞	∞
7	∞	∞
8	∞	8

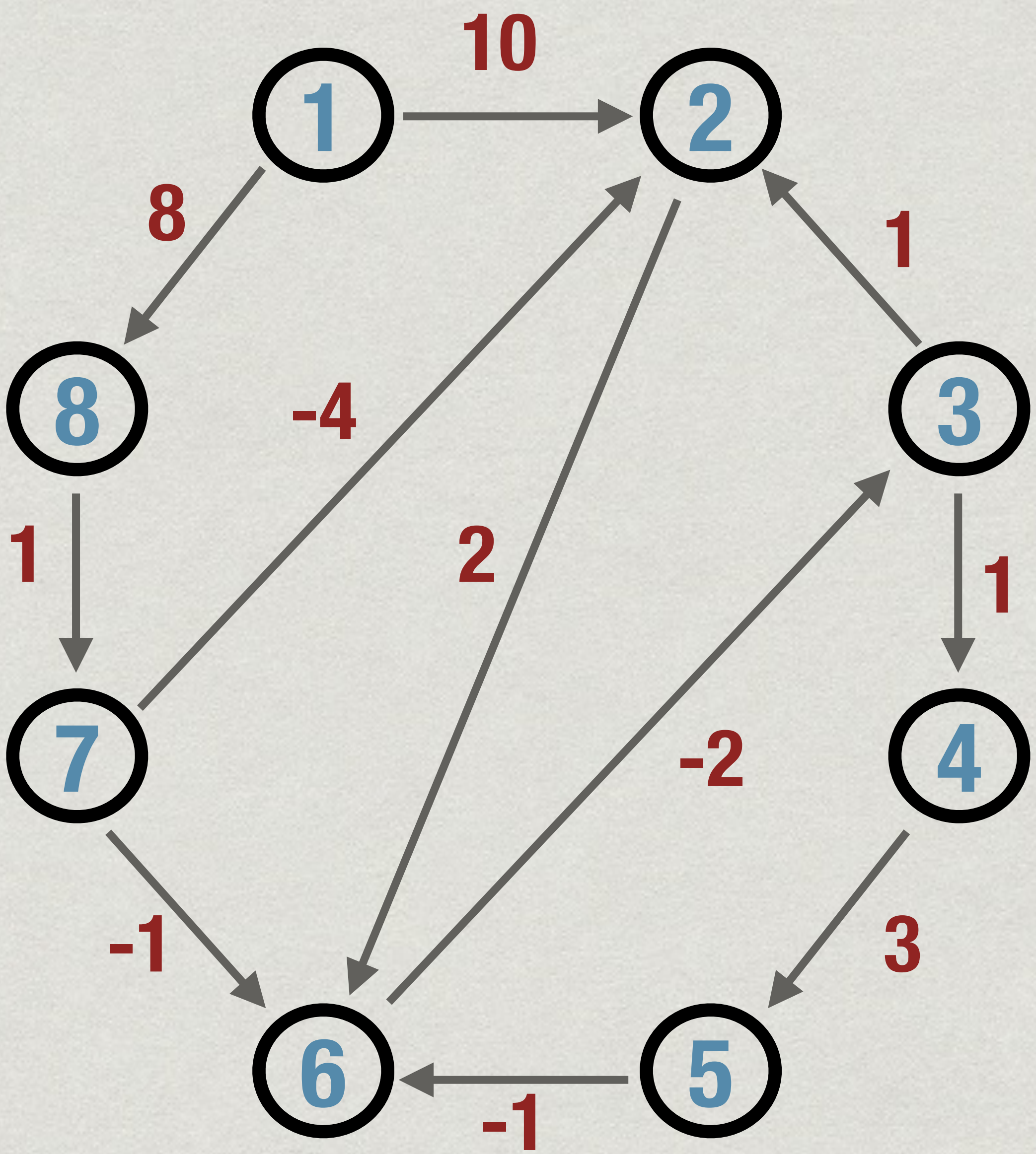
Example



Iteration

Node	0	1	2
1	0	0	0
2	∞	10	10
3	∞	∞	∞
4	∞	∞	∞
5	∞	∞	∞
6	∞	∞	12
7	∞	∞	9
8	∞	8	8

Example

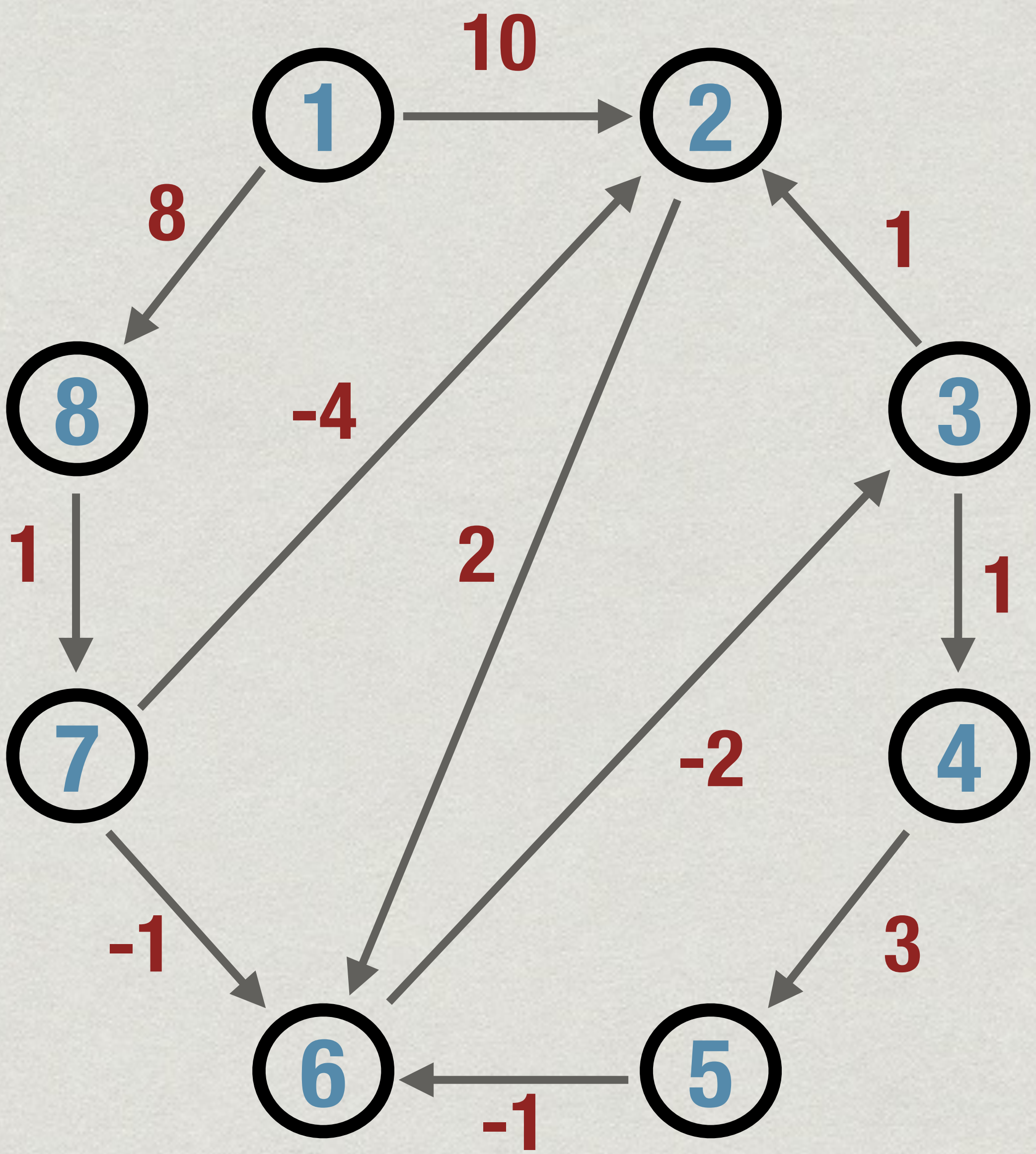


Iteration

Node

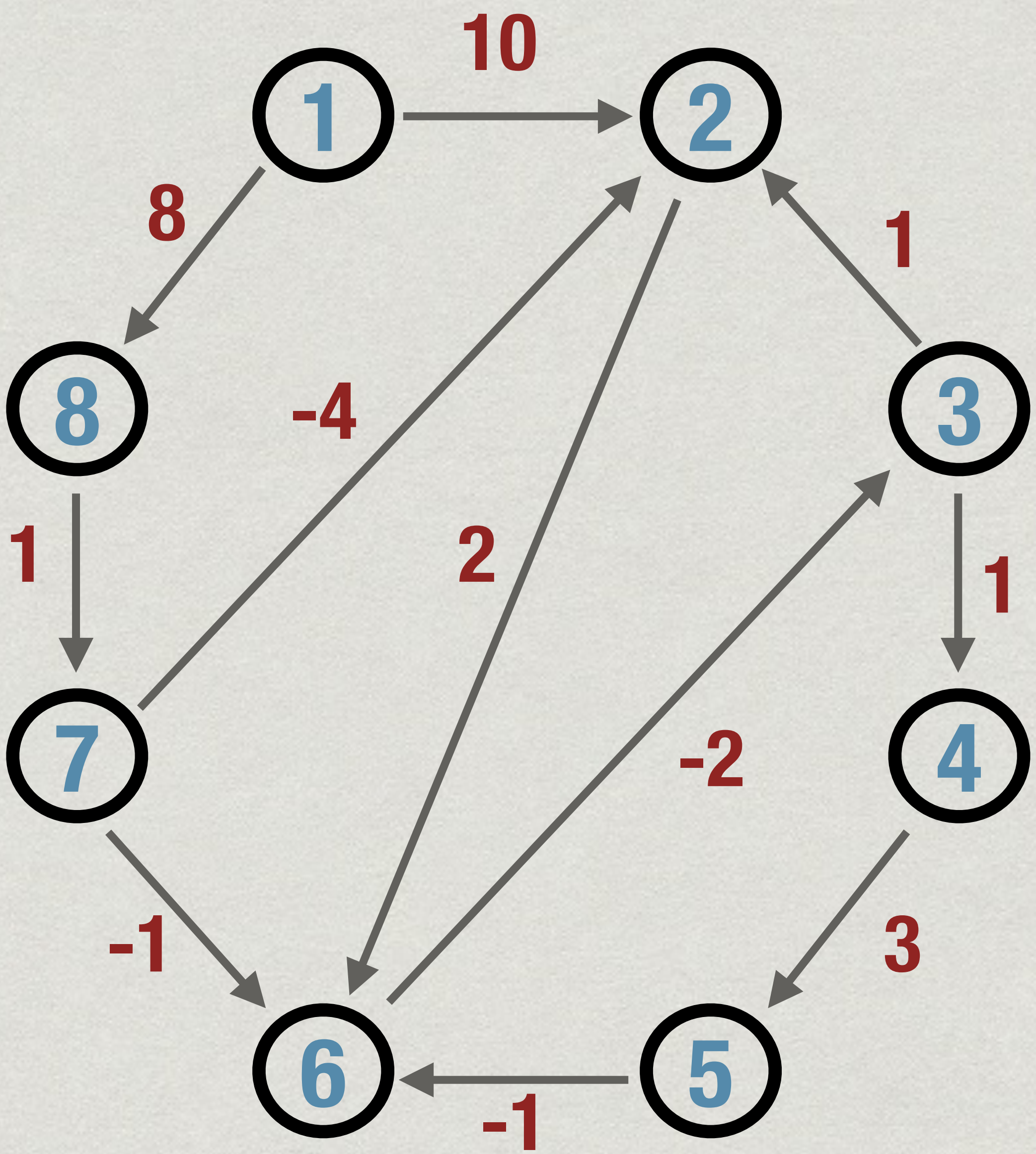
	0	1	2	3
1	0	0	0	0
2	∞	10	10	5
3	∞	∞	∞	10
4	∞	∞	∞	∞
5	∞	∞	∞	∞
6	∞	∞	12	8
7	∞	∞	9	9
8	∞	8	8	8

Example



Node	Iteration 0	1	2	3	4
1	0	0	0	0	0
2	∞	10	10	5	5
3	∞	∞	∞	10	6
4	∞	∞	∞	∞	11
5	∞	∞	∞	∞	∞
6	∞	∞	12	8	7
7	∞	∞	9	9	9
8	∞	8	8	8	8

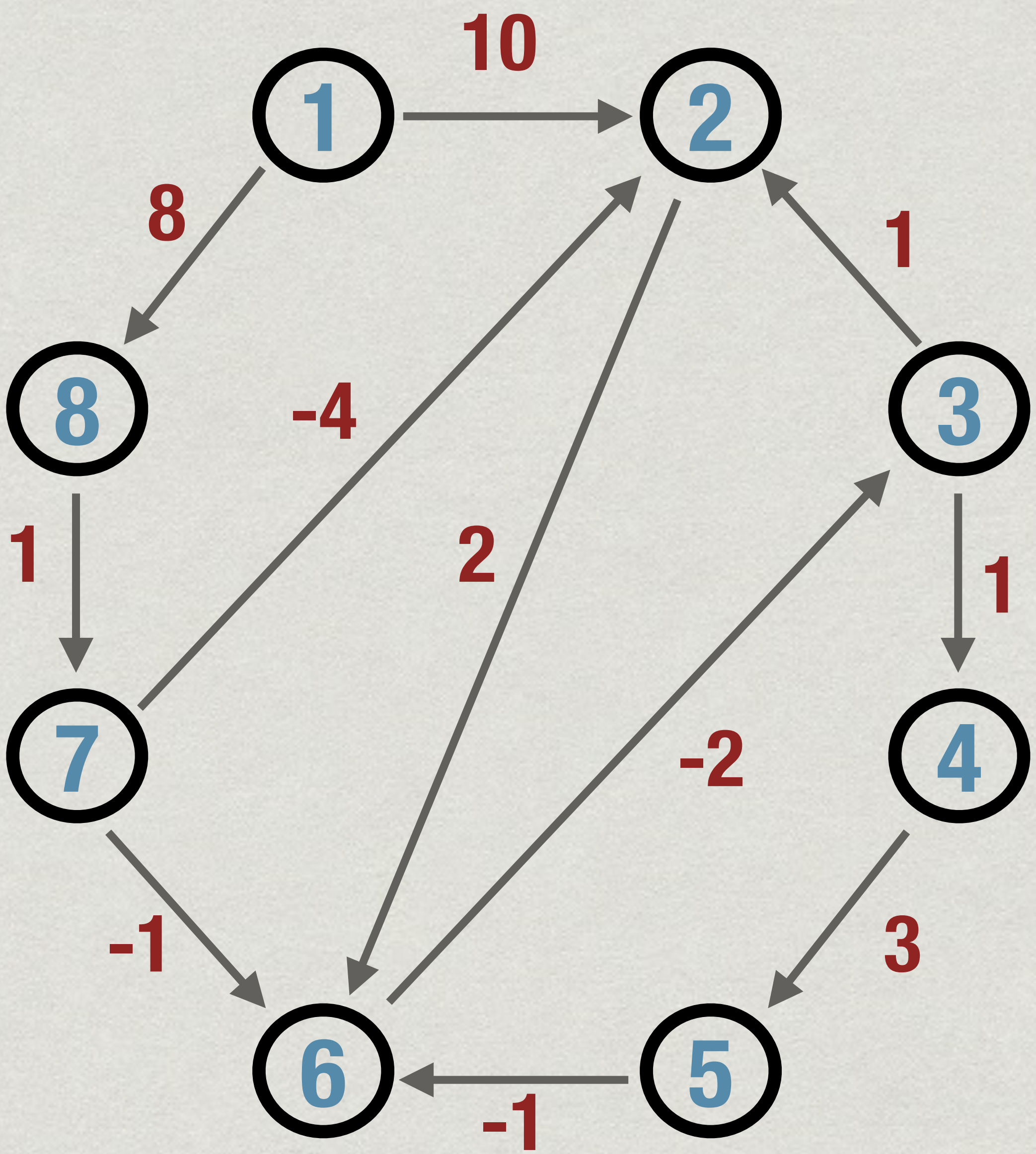
Example



Iteration

Node	0	1	2	3	4	5
1	0	0	0	0	0	0
2	∞	10	10	5	5	5
3	∞	∞	∞	10	6	5
4	∞	∞	∞	∞	11	7
5	∞	∞	∞	∞	∞	14
6	∞	∞	12	8	7	7
7	∞	∞	9	9	9	9
8	∞	8	8	8	8	8

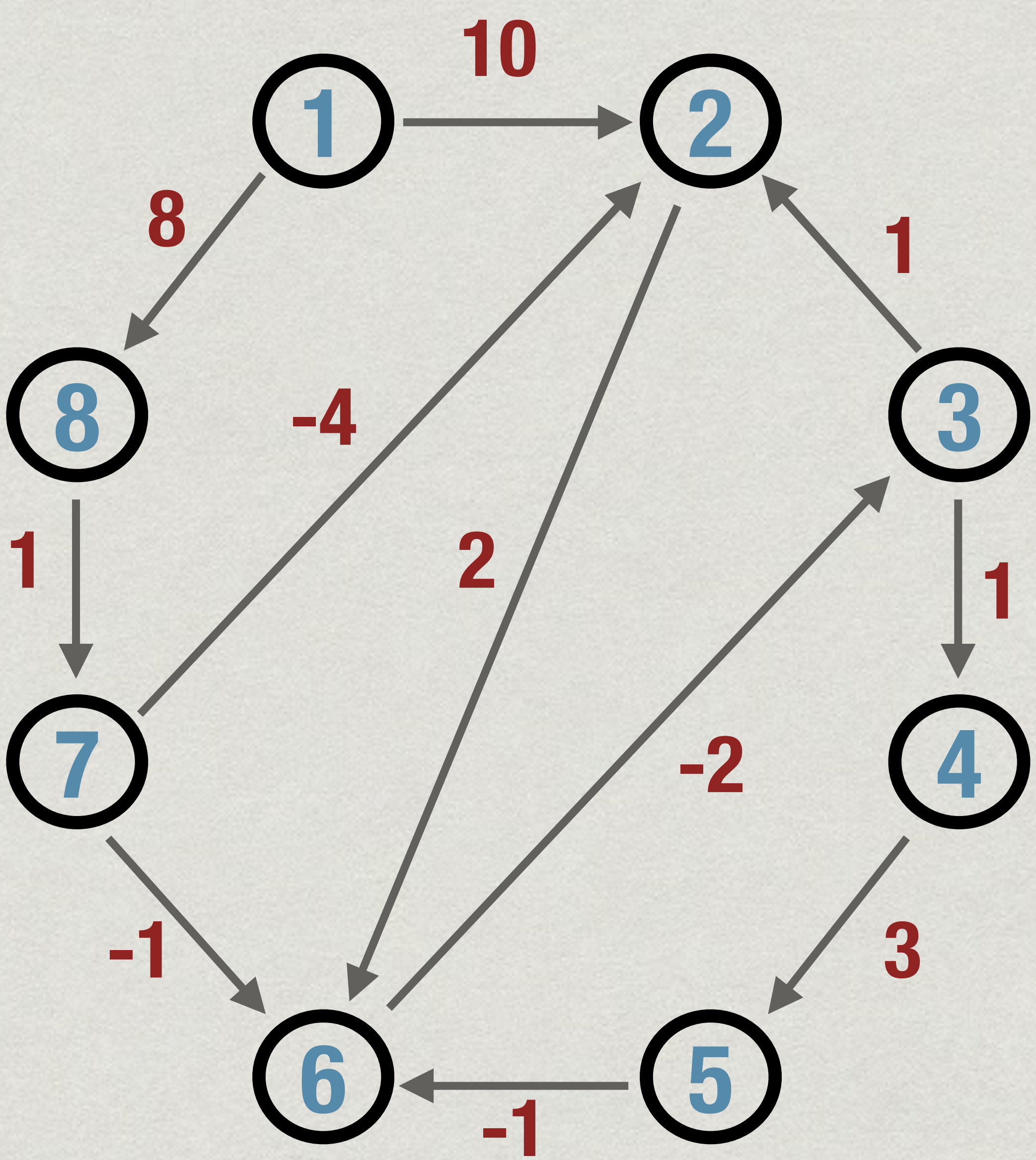
Example



Iteration

Node	0	1	2	3	4	5	6
1	0	0	0	0	0	0	0
2	∞	10	10	5	5	5	5
3	∞	∞	∞	10	6	5	5
4	∞	∞	∞	∞	11	7	6
5	∞	∞	∞	∞	∞	14	10
6	∞	∞	12	8	7	7	7
7	∞	∞	9	9	9	9	9
8	∞	8	8	8	8	8	8

Example



Iteration

Node	0	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0	0
2	∞	10	10	5	5	5	5	5
3	∞	∞	∞	10	6	5	5	5
4	∞	∞	∞	∞	11	7	6	6
5	∞	∞	∞	∞	∞	14	10	9
6	∞	∞	12	8	7	7	7	7
7	∞	∞	9	9	9	9	9	9
8	∞	8	8	8	8	8	8	8

Complexity

- Outer loop runs n times
- In each loop, for each edge (j,k), we run update(j,k)
  - Adjacency matrix — $O(n^2)$ to identify all edges
  - Adjacency list — $O(m)$
- Overall
  - Adjacency matrix — $O(n^3)$
  - Adjacency list — $O(mn)$