ITS202: Algorithms and Data Structure

Shortest Paths

Ms. Sonam Wangmo

Gyalpozhing College of Information Technology
Royal University of Bhutan

December 6, 2020

Edge-Weighted Graphs

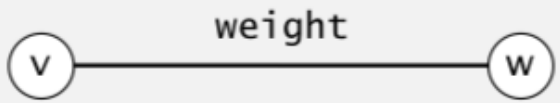
An edge-weighted graph is a graph model where we associate weights or costs with each edge.

Types of Edge Weighted Graph

1. Edge Weighted Graph
2. Edge Weighted Digraph

Weighted Undirected Edge API

public class Edge implements Comparable<Edge>	
Edge(int v, int w, double weight)	*create a weighted edge v-w*
int either()	*either endpoint*
int other(int v)	*the endpoint that's not v*
int compareTo(Edge that)	*compare this edge to that edge*
double weight()	*the weight*
String toString()	*string representation*



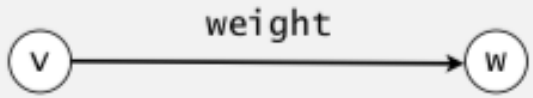
Idiom for processing an edge e: `int v = e.either(), w = e.other(v);`

Edge-Weighted graph API

public class EdgeWeightedGraph		
	EdgeWeightedGraph(int V)	*create an empty graph with V vertices*
	EdgeWeightedGraph(In in)	*create a graph from input stream*
void	addEdge(Edge e)	*add weighted edge e to this graph*
Iterable<Edge>	adj(int v)	*edges incident to v*
Iterable<Edge>	edges()	*all edges in this graph*
int	V()	*number of vertices*
int	E()	*number of edges*
String	toString()	*string representation*

Weighted Directed Edge API

public class DirectedEdge		
	DirectedEdge(int v, int w, double weight)	*weighted edge v→w*
int	from()	*vertex v*
int	to()	*vertex w*
double	weight()	*weight of this edge*
String	toString()	*string representation*



Idiom for processing an edge e: `int v = e.from(), w = e.to();`

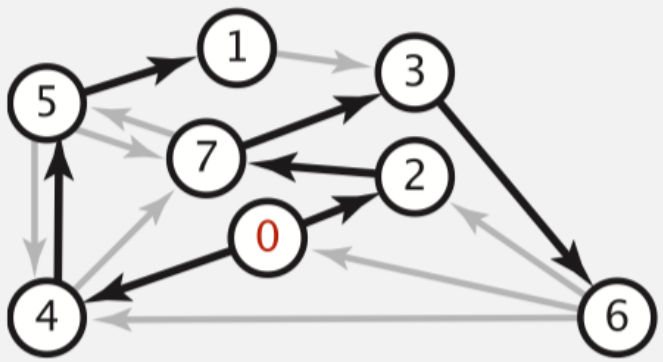
Edge-Weighted Digraph API

public class	EdgeWeightedDigraph	
	EdgeWeightedDigraph(int V)	*edge-weighted digraph with V vertices*
	EdgeWeightedDigraph(In in)	*edge-weighted digraph from input stream*
void	addEdge(DirectedEdge e)	*add weighted directed edge e*
Iterable<DirectedEdge>	adj(int v)	*edges pointing from v*
int	V()	*number of vertices*
int	E()	*number of edges*
Iterable<DirectedEdge>	edges()	*all edges*
String	toString()	*string representation*

Data structures for single-source shortest paths

Goal. Find the shortest path from s to every other vertex.
Can represent the shortest-paths tree (SPT) with two vertex-indexed arrays:

- distTo[v] is length of shortest path from s to v.
- edgeTo[v] is last edge on shortest path from s to v.



shortest-paths tree from 0

	edgeTo[]	distTo[]
0	null	0
1	5->1 0.32	1.05
2	0->2 0.26	0.26
3	7->3 0.37	0.97
4	0->4 0.38	0.38
5	4->5 0.35	0.73
6	3->6 0.52	1.49
7	2->7 0.34	0.60

parent-link representation

Edge Relaxation

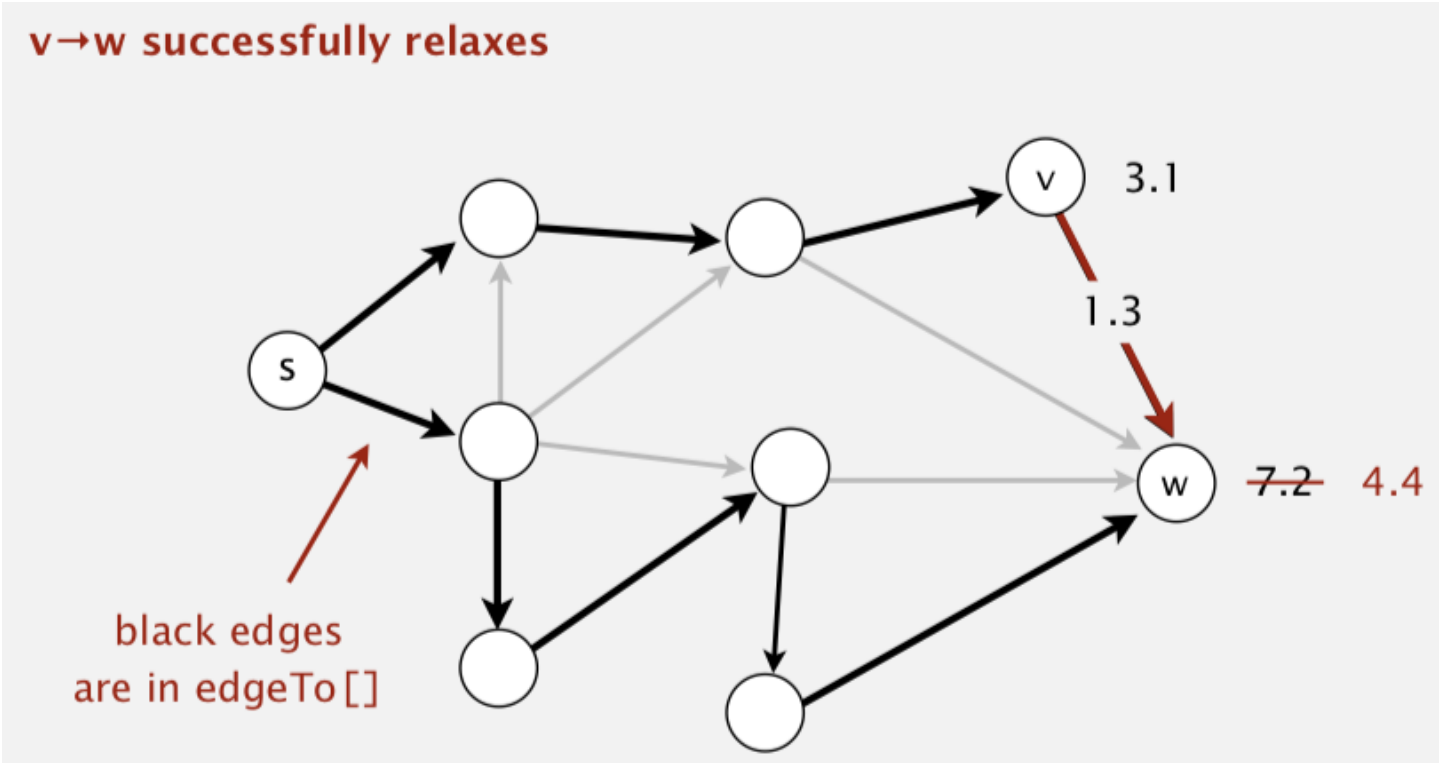
Relax edge e = v→w.

- distTo[v] is length of shortest known path from s to v.
- distTo[w] is length of shortest known path from s to w.
- edgeTo[w] is last edge on shortest known path from s to w.
- If e = v→w gives shorter path to w through v, update both distTo[w] and edgeTo[w].

Edge Relaxation

Relax(u,v,w)

- if d[v] >= d[u] + w(u,v)
- —> d[v] = d[u] + w(u,v)



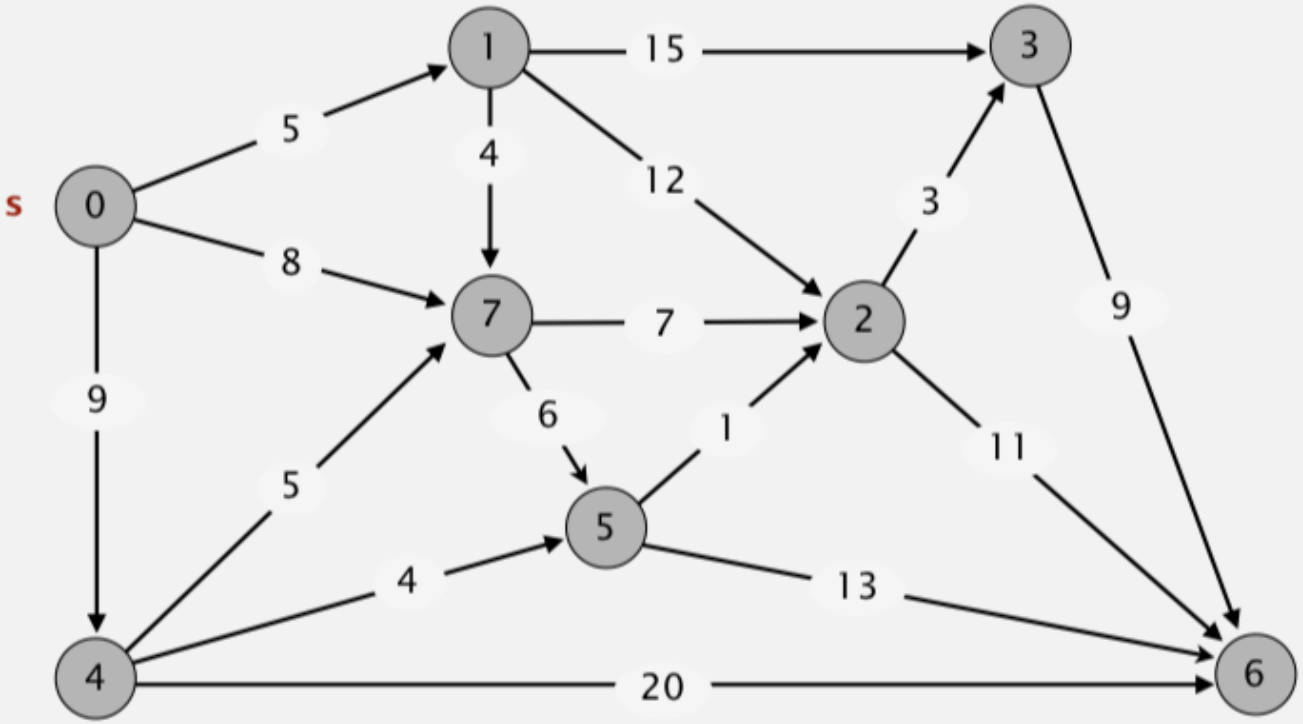
Generic shortest-paths algorithm

Generic algorithm (to compute SPT from s)
Initialize distTo[s] = 0 and distTo[v] = ∞ for all other vertices.
Repeat until optimality conditions are satisfied:
——>Relax any edge.
Efficient implementations. How to choose which edge to relax?
1. Dijkstra's algorithm (nonnegative weights).
2. Bellman-Ford algorithm (nonnegative cycles and nonnegative weights).

Dijkstra's algorithm

- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest distTo[] value).
- Add vertex to tree and relax all edges pointing from that vertex.

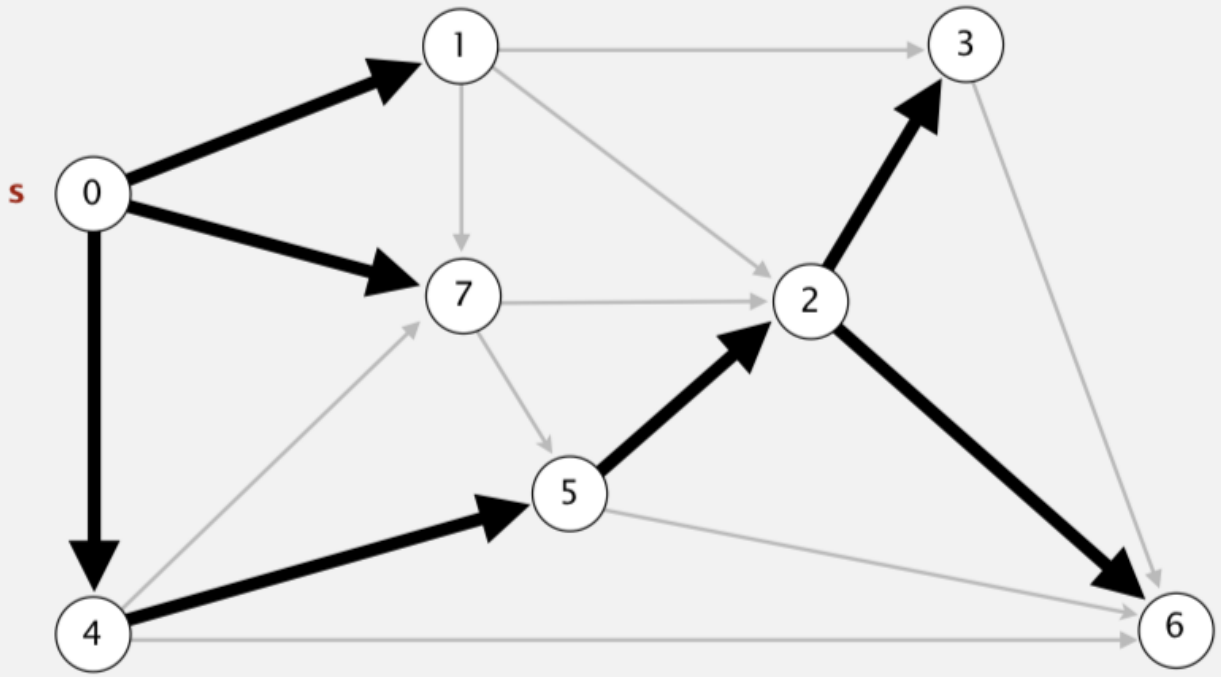
Dijkstra's algorithm



an edge-weighted digraph

0→1	5.0
0→4	9.0
0→7	8.0
1→2	12.0
1→3	15.0
1→7	4.0
2→3	3.0
2→6	11.0
3→6	9.0
4→5	4.0
4→6	20.0
4→7	5.0
5→2	1.0
5→6	13.0
7→5	6.0
7→2	7.0

Dijkstra's algorithm

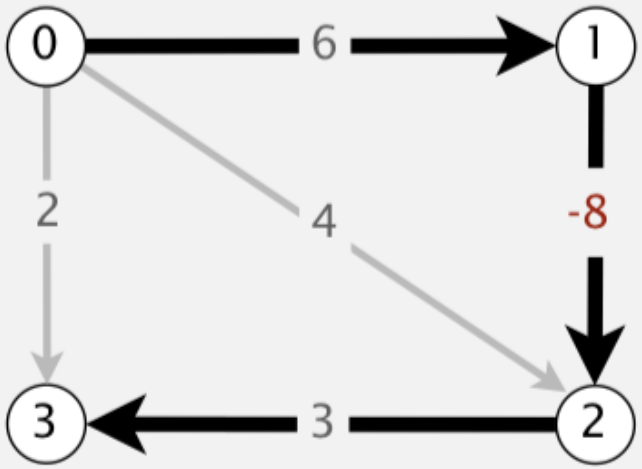


v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

shortest-paths tree from vertex s

Shortest paths with negative weights: failed attempts

Dijkstra. Doesn't work with negative edge weights.



Dijkstra selects vertex 3 immediately after 0.
But shortest path from 0 to 3 is 0→1→2→3.

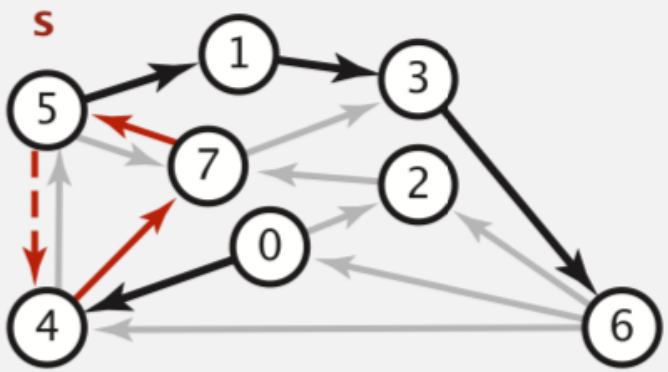
Conclusion. Need a different algorithm.

Negative cycles

Def. A negative cycle is a directed cycle whose sum of edge weights is negative.

digraph

4->5	0.35
5->4	-0.66
4->7	0.37
5->7	0.28
7->5	0.28
5->1	0.32
0->4	0.38
0->2	0.26
7->3	0.39
1->3	0.29
2->7	0.34
6->2	0.40
3->6	0.52
6->0	0.58
6->4	0.93



negative cycle (-0.66 + 0.37 + 0.28)

5->4->7->5

shortest path from 0 to 6

0->4->7->5->4->7->5...->1->3->6

Bellman-Ford algorithm

Bellman–Ford algorithm

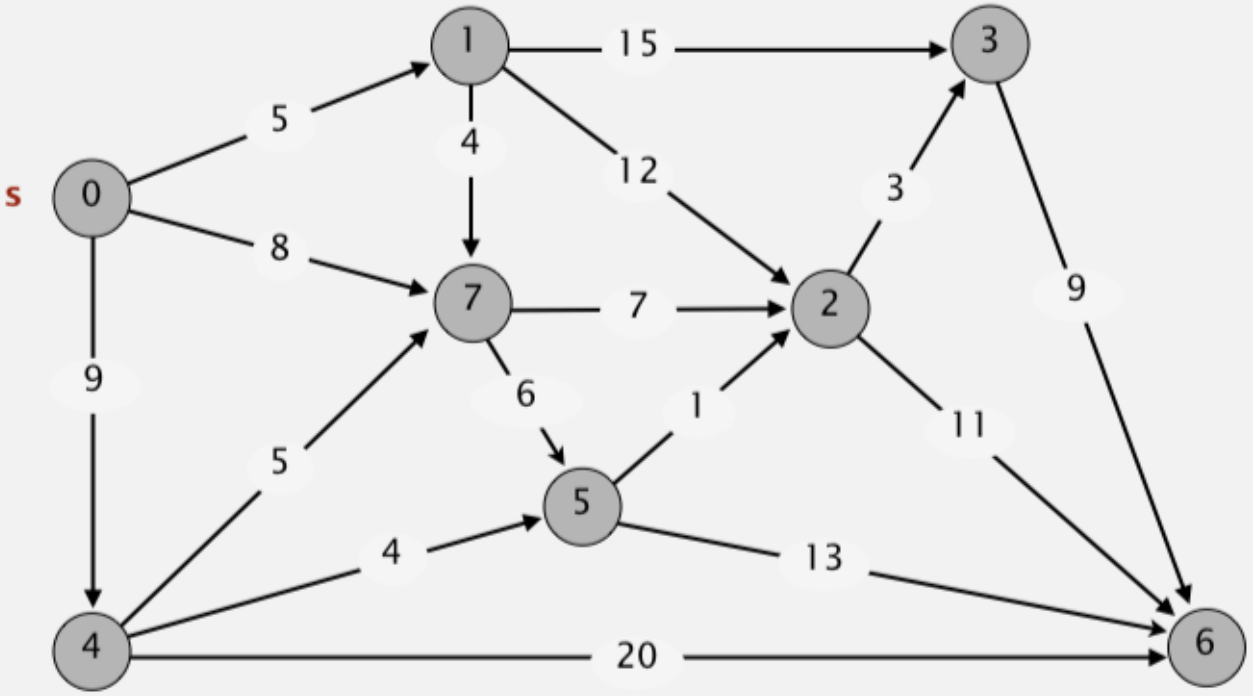
Initialize distTo[s] = 0 and distTo[v] = ∞ for all other vertices.

Repeat V times:

- **Relax each edge.**

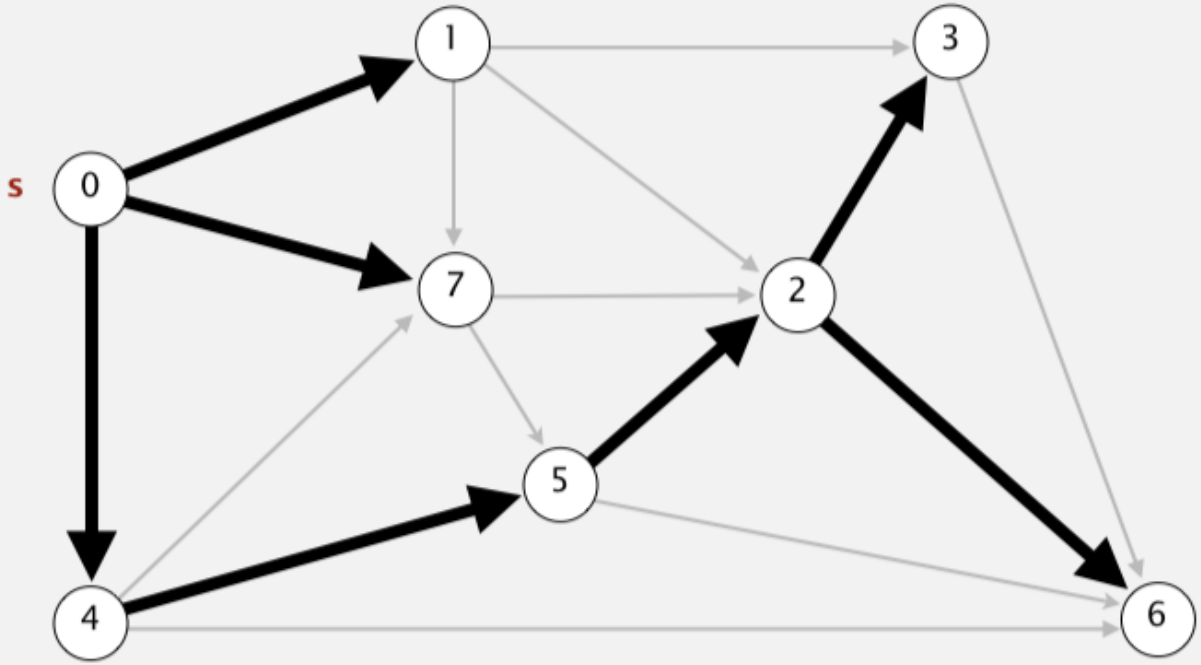
Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



0→1	5.0
0→4	9.0
0→7	8.0
1→2	12.0
1→3	15.0
1→7	4.0
2→3	3.0
2→6	11.0
3→6	9.0
4→5	4.0
4→6	20.0
4→7	5.0
5→2	1.0
5→6	13.0
7→5	6.0
7→2	7.0

Bellman-Ford algorithm demo



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

shortest-paths tree from vertex s