CS062

DATA STRUCTURES AND ADVANCED PROGRAMMING

27: Shortest Paths

Tom Yeh
he/him/his

Class News

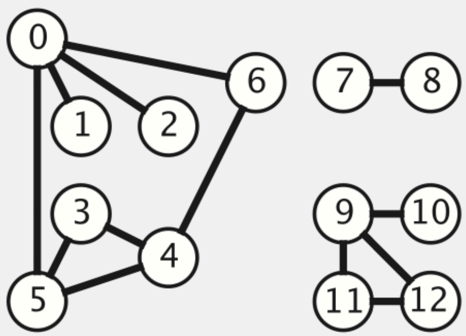
- Midterm problem: deletion of LLRB was not covered (removed from Midterm 2 grade)
- Quiz 8: redo today 2:30-4pm (max = 9)
- OH: extra office hours today 2:30-4pm, Wed 11-12
- Tomorrow's lab will start at 1pm
  - Lab component will finish at 2pm - TAs will be available to help with Assignment 9 questions

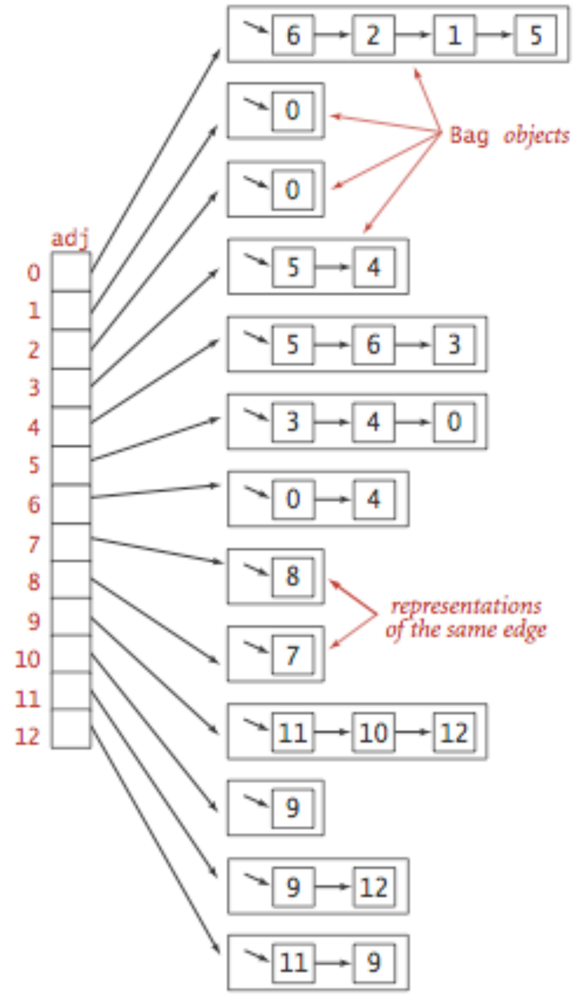
Graphs - Clarification

Graph input format.

tinyG.txt

```
V → 13
    13 ← E
    0 5
    4 3
    0 1
    9 12
    6 4
    5 4
    0 2
    11 12
    9 10
    0 6
    7 8
    9 11
    5 3
```

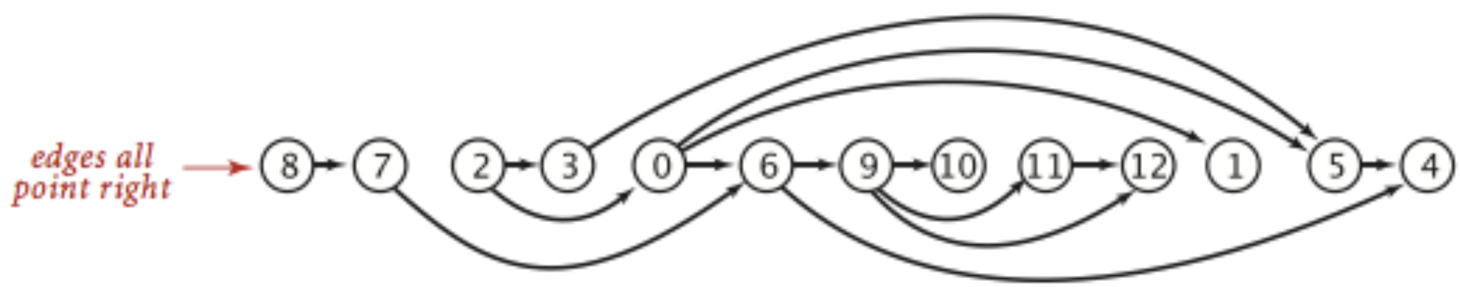




Adjacency-lists representation (undirected graph)

Topological sort

- Goal: Order the vertices of a DAG so that all edges point from an earlier vertex to a later vertex.
    - Think of modeling major requirements as a DAG. Or tasks that depend on completion of prior tasks.
- Reverse postorder in DAG is a topological sort.
- With DFS, we can topologically sort a DAG in $|E| + |V|$ time.



Depth-first orders

- If we save the vertex given as argument to recursive dfs in a data structure, we have three possible orders of seeing the vertices:
  - Preorder: Put the vertex on a queue before the recursive calls.
  - Postorder: Put the vertex on a queue after the recursive calls.
  - Reverse postorder: Put the vertex on a stack after the recursive calls.

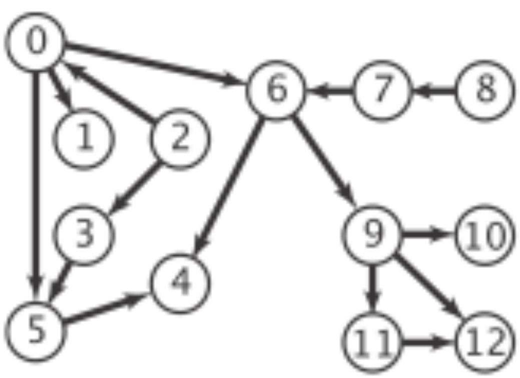
Depth-first orders

```
public class DepthFirstOrder {
    private boolean[] marked;          // marked[v] = has v been marked in dfs?
    private Queue<Integer> preorder;   // vertices in preorder
    private Queue<Integer> postorder;  // vertices in postorder
    private Stack<Integer> reversePostOrder;  // vertices in reverse postorder

    /**
     * Determines a depth-first order for the digraph {@code G}.
     * @param G the digraph
     */
    public DepthFirstOrder(Digraph G) {
        postorder = new Queue<Integer>();
        preorder  = new Queue<Integer>();
        reversePostOrder  = new Stack<Integer>();
        marked    = new boolean[G.V()];
        for (int v = 0; v < G.V(); v++)
            if (!marked[v]) dfs(G, v);
    }

    // run DFS in digraph G from vertex v and compute preorder/postorder
    private void dfs(Digraph G, int v) {
        marked[v] = true;
        preorder.enqueue(v);
        for (int w : G.adj(v)) {
            if (!marked[w]) {
                dfs(G, w);
            }
        }
        postorder.enqueue(v);
        reversePostorder.push(v);
   }
```

http://algs4.cs.princeton.edu

4.2 TOPOLOGICAL SORT DEMO

Depth-first orders



preorder is order of dfs() *calls*

postorder is order in which vertices are done

	pre *(queue)*	post *(queue)*	reversePost *(stack)*
dfs(0)	0		
dfs(5)	0 5		
dfs(4)	0 5 4		
4 done		4	4
5 done		4 5	5 4
dfs(1)	0 5 4 1		
1 done		4 5 1	1 5 4
dfs(6)	0 5 4 1 6		
dfs(9)	0 5 4 1 6 9		
dfs(11)	0 5 4 1 6 9 11		
dfs(12)	0 5 4 1 6 9 11 12		
12 done		4 5 1 12	12 1 5 4
11 done		4 5 1 12 11	11 12 1 5 4
dfs(10)	0 5 4 1 6 9 11 12 10		
10 done		4 5 1 12 11 10	10 11 12 1 5 4
check 12			
9 done		4 5 1 12 11 10 9	9 10 11 12 1 5 4
check 4			
6 done		4 5 1 12 11 10 9 6	6 9 10 11 12 1 5 4
0 done		4 5 1 12 11 10 9 6 0	0 6 9 10 11 12 1 5 4
check 1			
dfs(2)	0 5 4 1 6 9 11 12 10 2		
check 0			
dfs(3)	0 5 4 1 6 9 11 12 10 2 3		
check 5			
3 done		4 5 1 12 11 10 9 6 0 3	3 0 6 9 10 11 12 1 5 4
2 done		4 5 1 12 11 10 9 6 0 3 2	2 3 0 6 9 10 11 12 1 5 4
check 3			
check 4			
check 5			
check 6			
dfs(7)	0 5 4 1 6 9 11 12 10 2 3 7		
check 6			
7 done		4 5 1 12 11 10 9 6 0 3 2 7	7 2 3 0 6 9 10 11 12 1 5 4
dfs(8)	0 5 4 1 6 9 11 12 10 2 3 7 8		
check 7			
8 done		4 5 1 12 11 10 9 6 0 3 2 7 8	8 7 2 3 0 6 9 10 11 12 1 5 4
check 9			
check 10			
check 11			
check 12			

reverse postorder

Summary

- Single-source reachability in a digraph: DFS/BFS.
- Shortest path in a digraph: BFS.
- Topological sort in a DAG: DFS.

Is a digraph strongly connected?

- Pick a random starting vertex s.
- Run DFS/BFS starting at s.
  - If have not reached all vertices, return false.
- Reverse edges.
- Run DFS/BFS again on reversed graph.
  - If have not reached all vertices, return false.
  - Else return true.

Lecture 24-25: Graphs

- Undirected Graphs
  - Graph API
  - Depth-First Search
  - Breadth-First Search
  - Connected Components
- Directed Graphs
  - Digraph API
  - Depth-First Search
  - Breadth-First Search
  - Topological Sort
  - Strongly Connected Components

Some slides adopted from Algorithms 4th Edition or COS226

Readings:

- Textbook: Chapter 4.1 (Pages 522-556), Chapter 4.2 (Pages 566-594)
- Website:
  - https://algs4.cs.princeton.edu/41graph/
  - https://algs4.cs.princeton.edu/42digraph/

Practice Problems:

- 4.1.1-4.1.6, 4.1.9, 4.1.11
- 4.2.1-4.27

Lecture 27: Shortest Paths

- Introduction to Shortest Paths
- API
- Properties
- Dijkstra's Algorithm
- Belman-Ford Algorithm

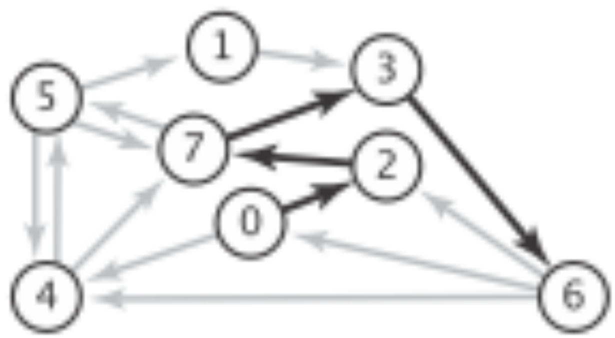
Some slides adopted from Algorithms 4th Edition or COS226

Edge-weighted digraph

- Edge-weighted digraph: a digraph where we associate weights or costs with each edge.

edge-weighted digraph

4->5	0.35
5->4	0.35
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

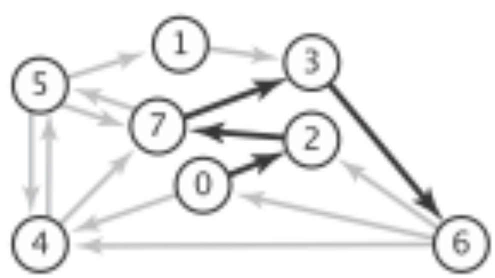


Shortest Paths

- **Shortest path from vertex s to vertex t**: a directed path from s to t with the property that no other such path has a lower weight (total weight sum of edges it consists of).

edge-weighted digraph

4->5	0.35
5->4	0.35
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



shortest path from 0 to 6

0->2	0.26
2->7	0.34
7->3	0.39
3->6	0.52

An edge-weighted digraph and a shortest path

Shortest Path variants

- Single source: from one vertex `s` to every other vertex.
- Single sink: from every vertex to one vertex `t`.
- Source-sink: from one vertex `s` to another vertex `t`.
- All pairs: from every vertex to every other vertex.

- What version is there in your navigation app?

Shortest Paths Assumptions

- Not all vertices need to be reachable.
  - We will assume so in this lecture.
- Weights are non-negative.
  - There are algorithms that can handle negative weights.
- Shortest paths are not necessarily unique but they are simple.

Lecture 27: Shortest Paths

- Introduction to Shortest Paths
- API
- Properties
- Dijkstra's Algorithm
- Belman-Ford Algorithm

Weighted directed edge API

- `public class DirectedEdge`
  - `DirectedEdge(int v, int w, double weight)`
    - Constructs a weighted edge from `v` to `w` `(v->w)` with the provided `weight`.
  - `int from()`
    - Returns vertex source of this edge.
  - `int to()`
    - Returns vertex destination of this edge.
  - `double weight()`
    - Returns weight of this edge.
  - `String toString()`
    - Returns the string representation of this edge.

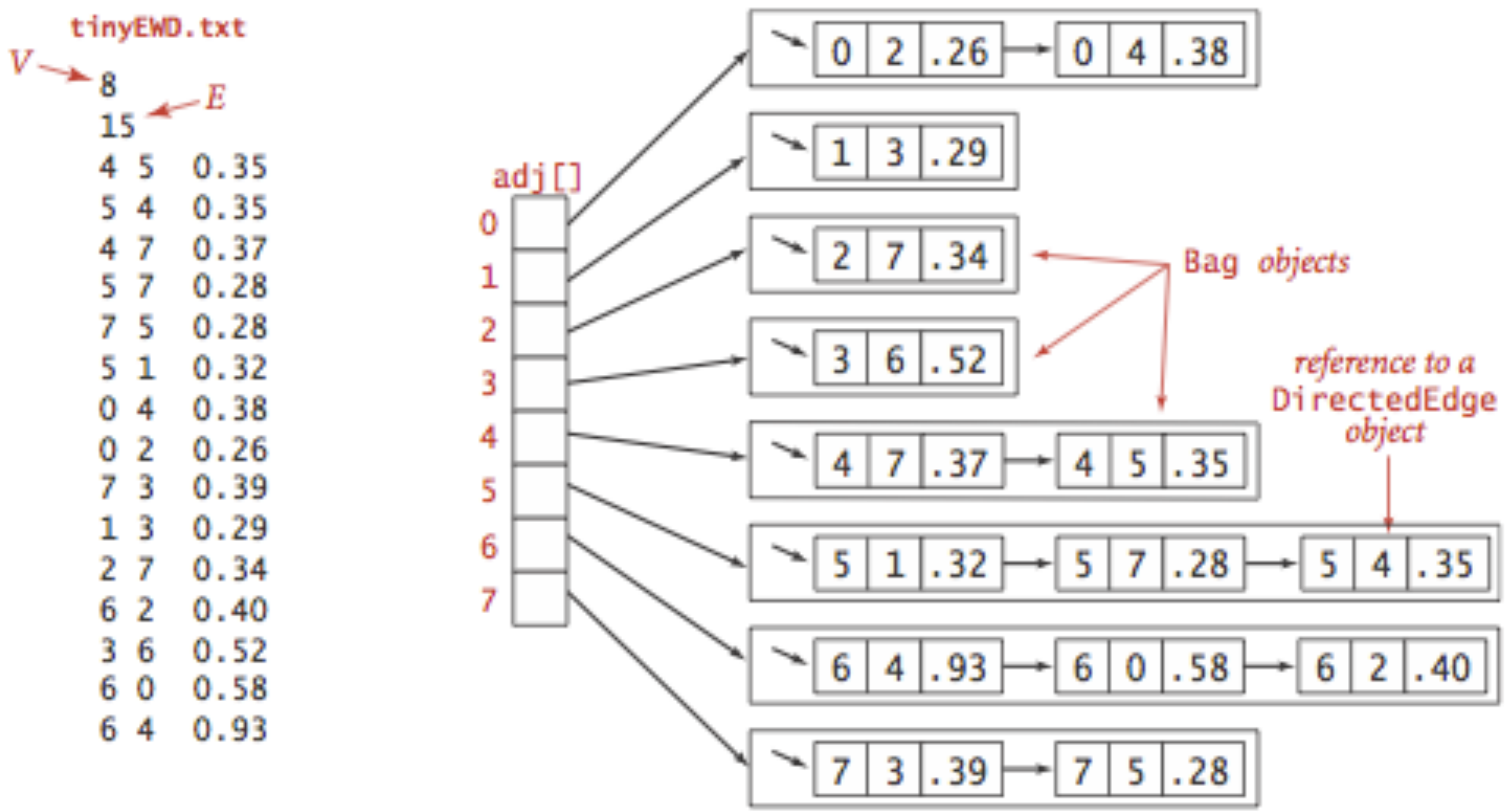
Weighted directed edge in Java

```
public class DirectedEdge {
    private final int v;
    private final int w;
    private final double weight;

   public DirectedEdge(int v, int w, double weight) {
        this.v = v;
        this.w = w;
        this.weight = weight;
    }

   public int from() {
        return v;
    }

    public int to() {
        return w;
    }

    public double weight() {
        return weight;
    }
```

Edge-weighted digraph API

- `public class EdgeWeightedDigraph`
    - `EdgeWeightedDigraph(int v)`
        - Constructs an edge-weighted digraph with v vertices.
    - `void addEdge(DirectedEdge e)`
        - Add weighted directed edge e.
    - `Iterable<DirectedEdge> adj(int v)`
        - Returns edges adjacent from v.
    - `int V()`
        - Returns number of vertices.
    - `int E()`
        - Returns number of edges.
    - `Iterable<DirectedEdge> edges()`
        - Returns all edges.

Edge-weighted digraph adjacency list representation



Edge-weighted digraph representation

Edge-weighted digraph in Java

```
public class EdgeWeightedDigraph {
    private final int V;                   // number of vertices in this digraph
    private int E;                         // number of edges in this digraph
    private Bag<DirectedEdge>[] adj;       // adj[v] = adjacency list for vertex v

    public EdgeWeightedDigraph(int V) {
       this.V = V;
       this.E = 0;
       adj = (Bag<DirectedEdge>[]) new Bag[V];
        for (int v = 0; v < V; v++)
            adj[v] = new Bag<DirectedEdge>();
    }
    public void addEdge(DirectedEdge e) {
        int v = e.from();
        int w = e.to();
        adj[v].add(e);
        E++;
    }

   public Iterable<DirectedEdge> adj(int v) {
       return adj[v];
    }
```

Single-source shortest path API

- Goal: find shortest path from `s` to every other vertex in the digraph.
- `public class SP`
  - `SP(EdgeWeightedDigraph G, int s)`
    - Shortest paths from `s` in digraph `G`.
  - `double distTo(int v)`
    - Length of shortest path from `s` to `v`.
  - `Iterable<DirectedEdge> pathTo(int v)`
    - Returns edges along the shortest path from `s` to `v`.
  - `boolean hasPathTo(int v)`
    - Returns whether there is a path from `s` to `v`.

Lecture 27: Shortest Paths

- Introduction to Shortest Paths
- API
- Properties
- Dijkstra's Algorithm
- Belman-Ford Algorithm

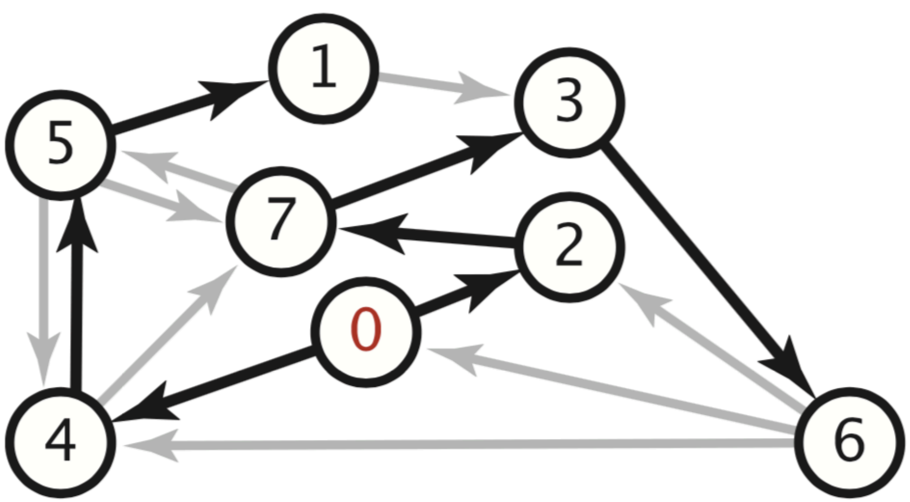
Data structures for single-source shortest paths

- Goal: find shortest path from `s` to every other vertex in the digraph.
- Shortest-paths tree (SPT): a subgraph containing `s` and all the vertices reachable from `s` that forms a directed tree rooted at s such that every tree path in the SPT is a shortest path in the digraph.
- Representation of shortest paths with two vertex-indexed arrays.
  - Edges on the shortest-paths tree: `edgeTo[v]` is the last edge on a shortest path from `s` to `v`.
  - Distance to the source: `distTo[v]` is the length of the shortest path from `s` to `v`.

edge-weighted digra

4->5	0.35
5->4	0.35
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

```
public Iterable<DirectedEdge> pathTo(int v) {
       Stack<DirectedEdge> path = new Stack<DirectedEdge>();
       for (DirectedEdge e = edgeTo[v]; e != null; e = edgeTo[e.from()]) {
           path.push(e);
       }
       return path;
   }
```

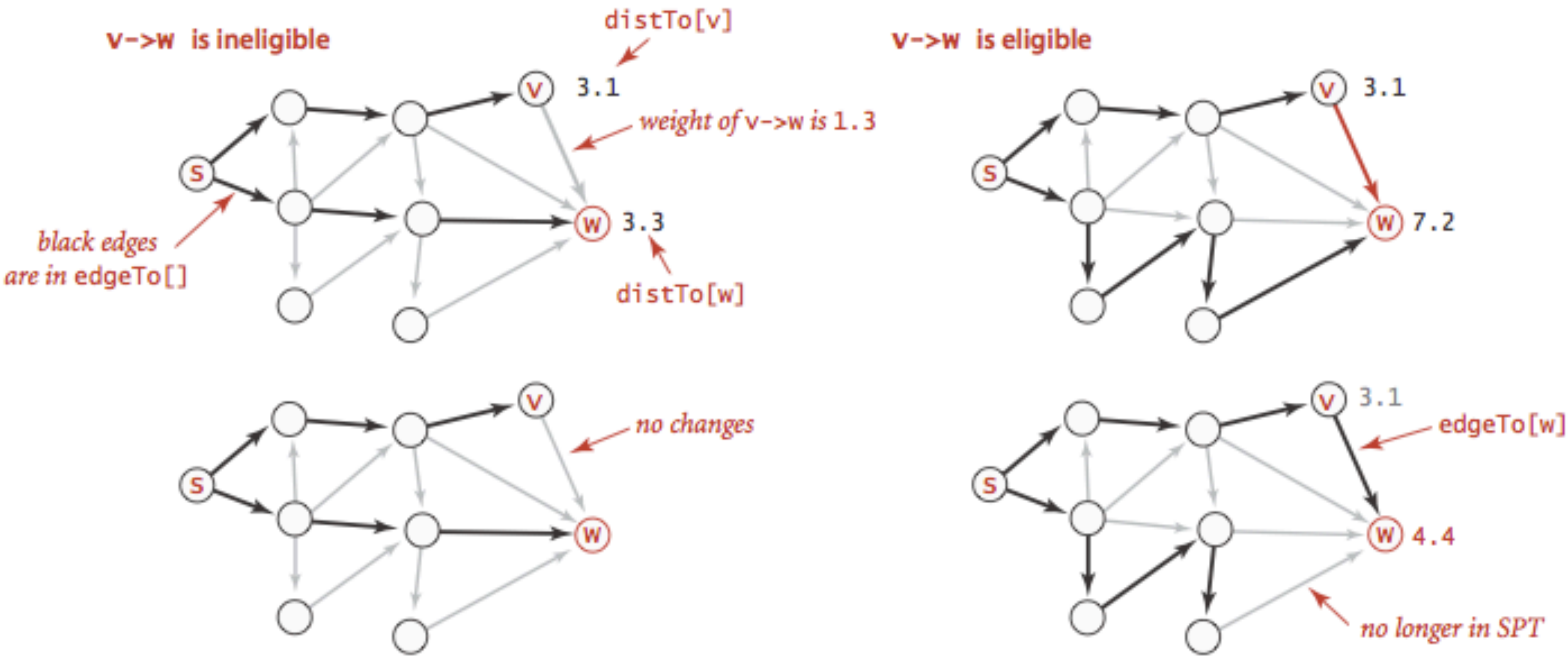


	edgeTo[]		distTo[]
0	null		0
1	5->1	0.32	1.05
2	0->2	0.26	0.26
3	7->3	0.39	0.99
4	0->4	0.38	0.38
5	4->5	0.35	0.73
6	3->6	0.52	1.51
7	2->7	0.34	0.60

Edge relaxation

- Relax edge `e = v->w`
  - `distTo[v]` is the length of the shortest **known** path from `s` to `v`.
  - `distTo[w]` is the length of the shortest **known** path from `s` to `w`.
  - `edgeTo[w]` is the last edge on shortest **known** path from `s` to `w`.
  - If `e = v->w` yields shorter path to `w`, update `distTo[w]` and `edgeTo[w]`.

Edge relaxation



Edge relaxation implementation

```
private void relax(DirectedEdge e) {
    int v = e.from(), w = e.to();
    if (distTo[w] > distTo[v] + e.weight()) {
       distTo[w] = distTo[v] + e.weight();
       edgeTo[w] = e;
    }
}
```

Framework for shortest-paths algorithm

- Generic algorithm to compute a SPT from s
  - `distTo[v]`=∞ for each vertex `v`.
  - `edgeTo[v]=null` for each vertex `v`.
  - `distTo[s]=0`.
  - Repeat until done:
    - Relax any edge.
- `distTo[v]` is the length of a simple path from `s` to `v`.
- `distTo[v]` does not increase.

Lecture 27: Shortest Paths

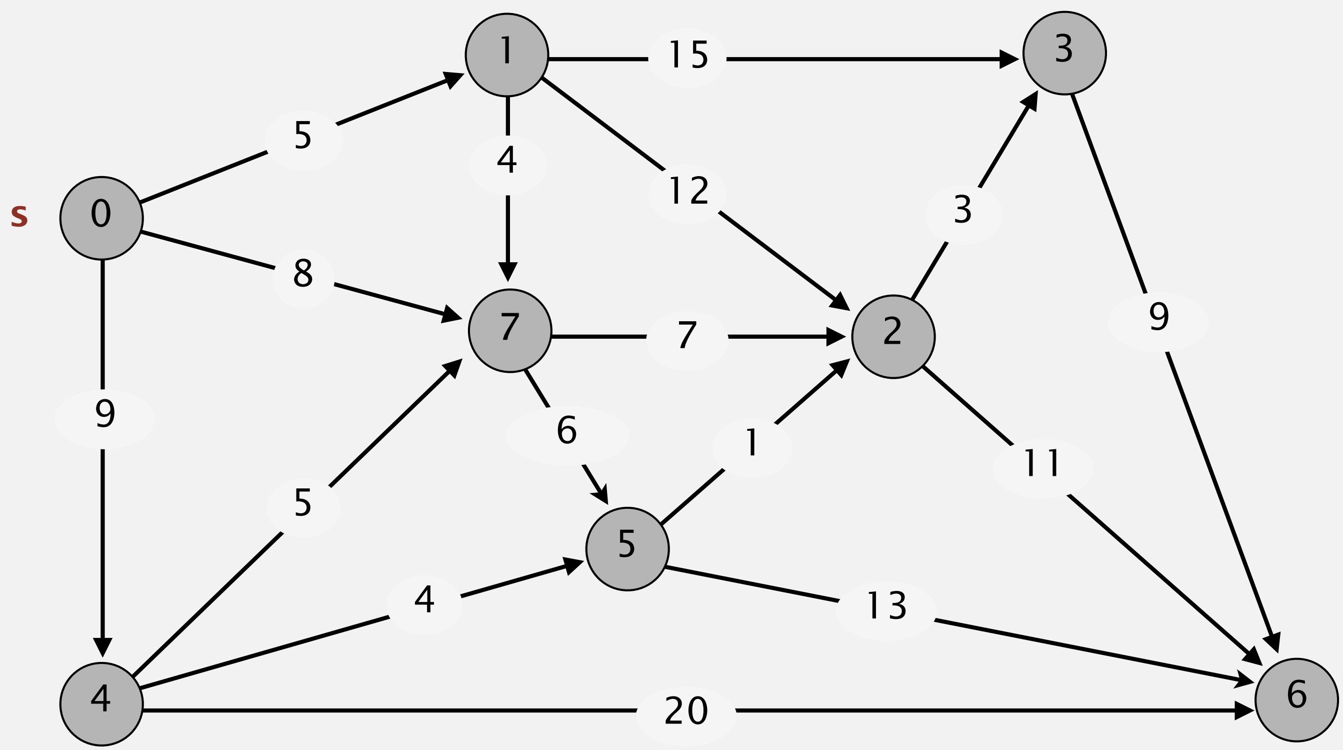
- Introduction to Shortest Paths
- API
- Properties
- Dijkstra's Algorithm
- Belman-Ford Algorithm

http://algs4.cs.princeton.edu

DIJKSTRA'S ALGORITHM DEMO

Dijkstra's algorithm demo

- 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 adjacent from that vertex.

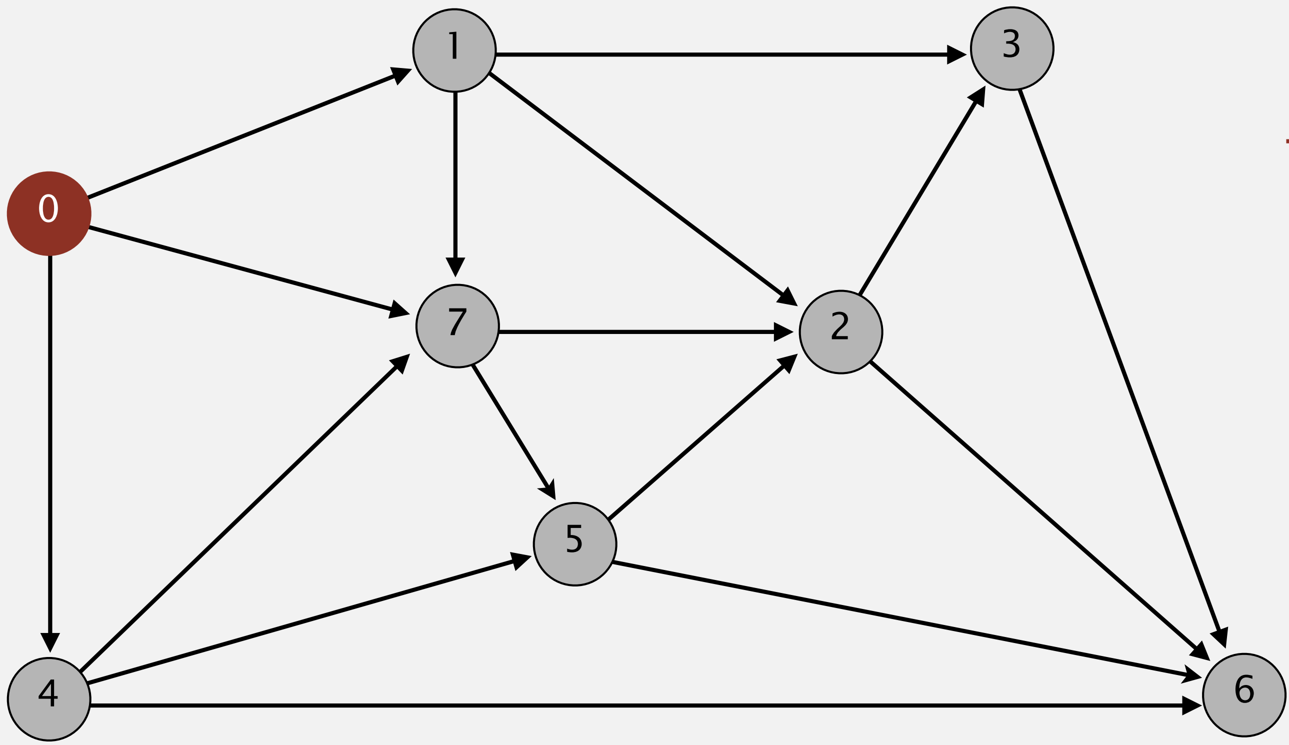


an edge-weighted digraph

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 demo

- 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 adjacent from that vertex.

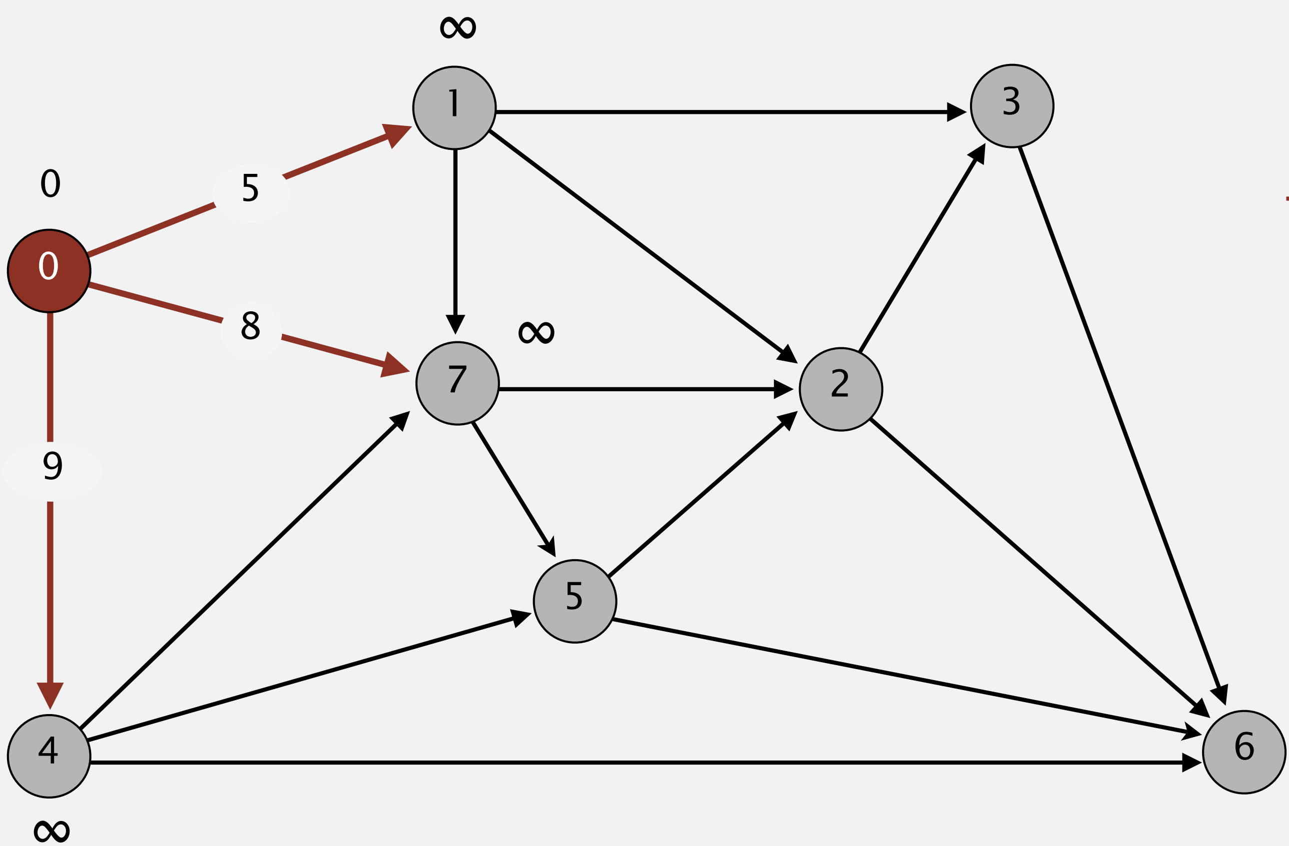


v	distTo[]	edgeTo[]
0	0.0	-
1		
2		
3		
4		
5		
6		
7		

choose source vertex 0

Dijkstra's algorithm demo

- 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 adjacent from that vertex.

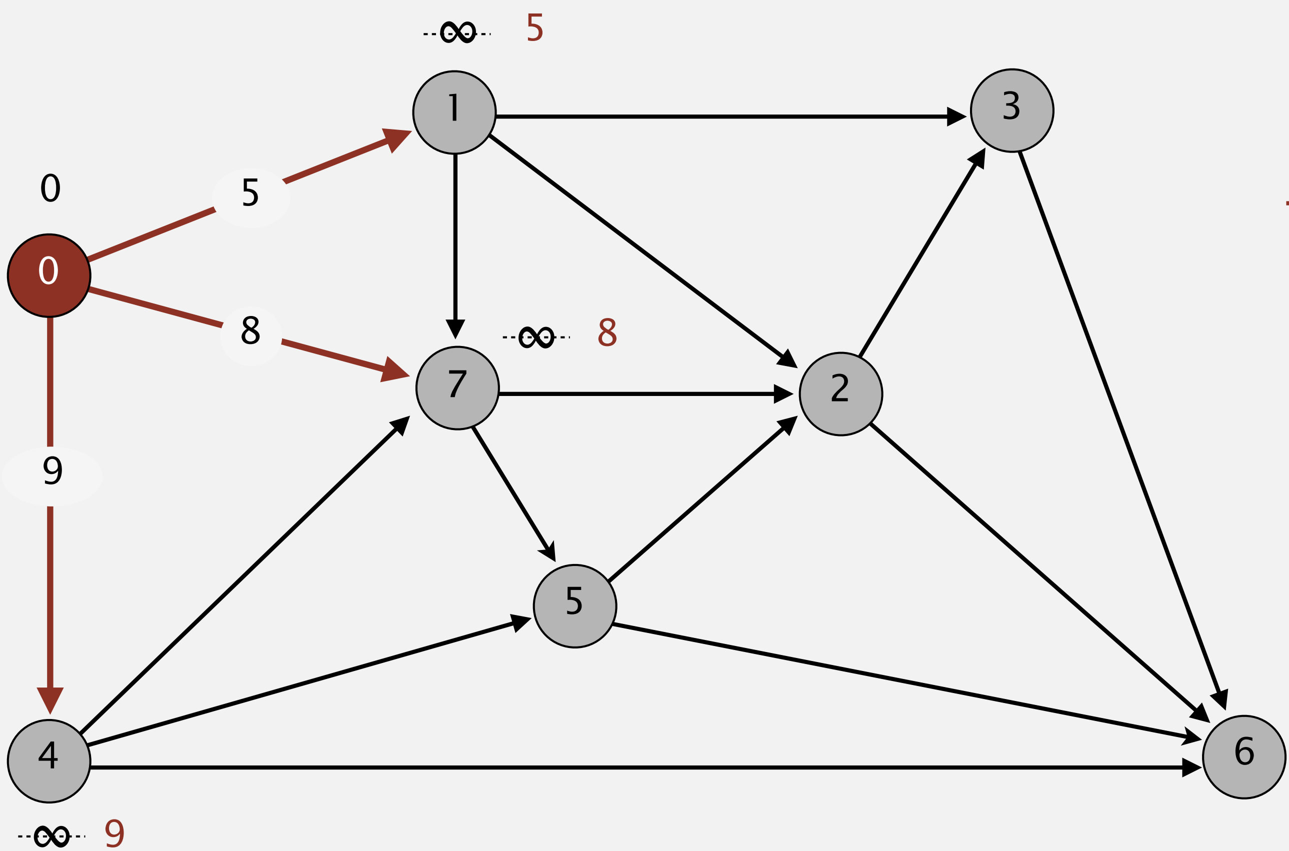


v	distTo[]	edgeTo[]
0	0.0	-
1		
2		
3		
4		
5		
6		
7		

relax all edges adjacent from 0

Dijkstra's algorithm demo

- 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 adjacent from that vertex.

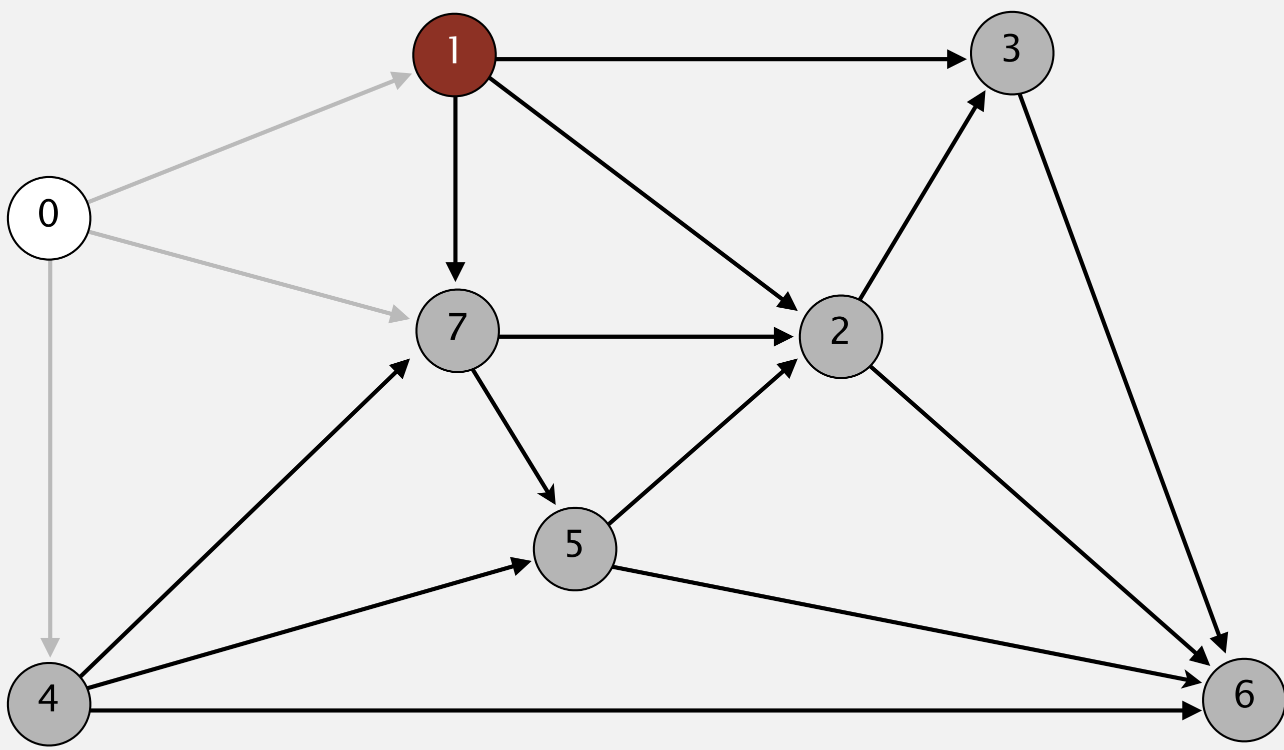


v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2		
3		
4	9.0	0→4
5		
6		
7	8.0	0→7

relax all edges adjacent from 0

Dijkstra's algorithm demo

- 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 adjacent from that vertex.

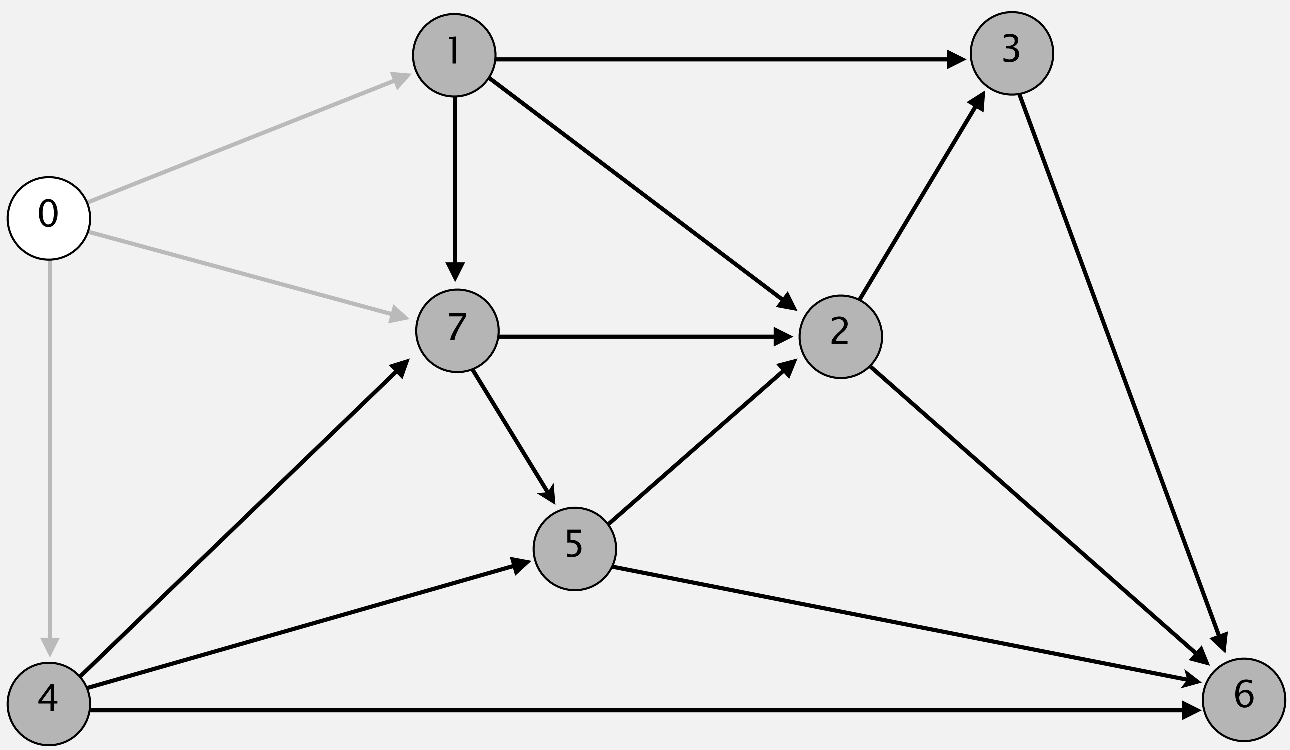


v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2		
3		
4	9.0	0→4
5		
6		
7	8.0	0→7

choose vertex 1

Dijkstra's algorithm demo

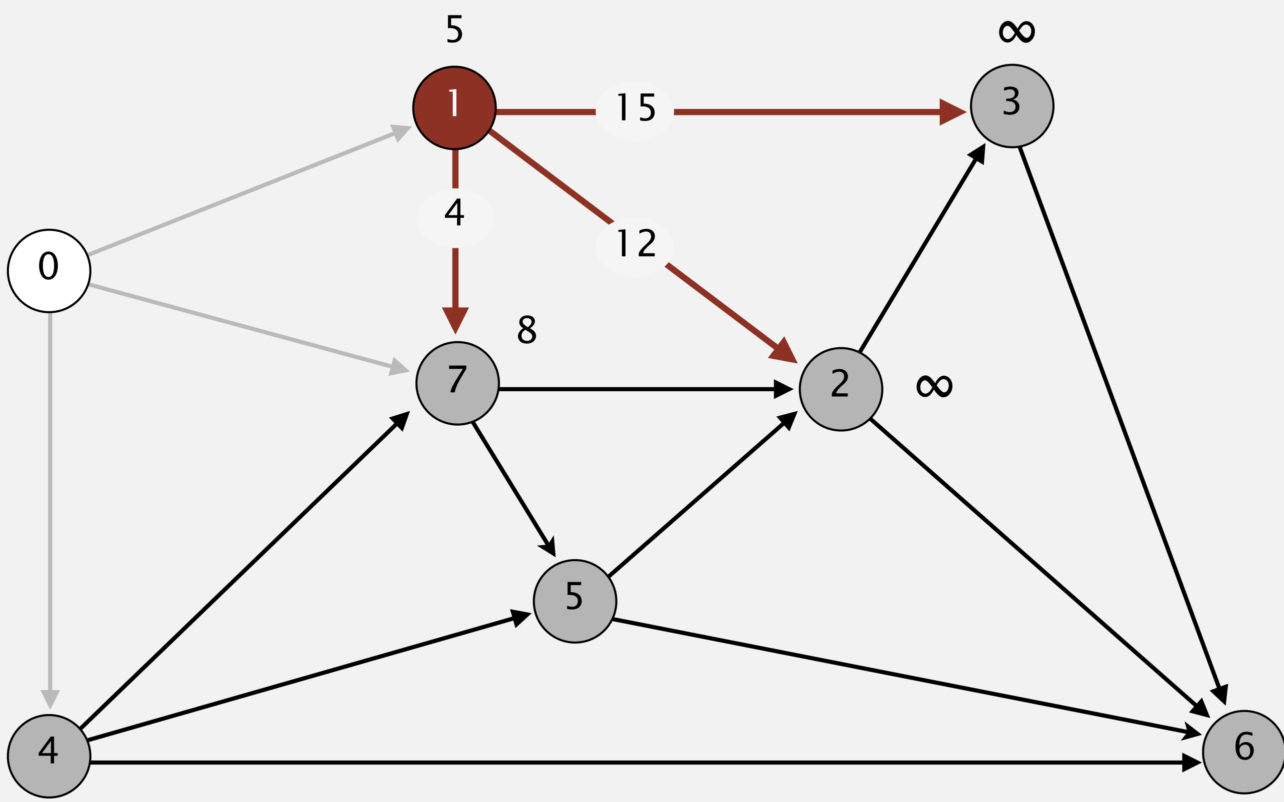
- Consider vertices in increasing order of distance from `s` (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges adjacent from that vertex.



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

Dijkstra's algorithm demo

- 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 adjacent from that vertex.

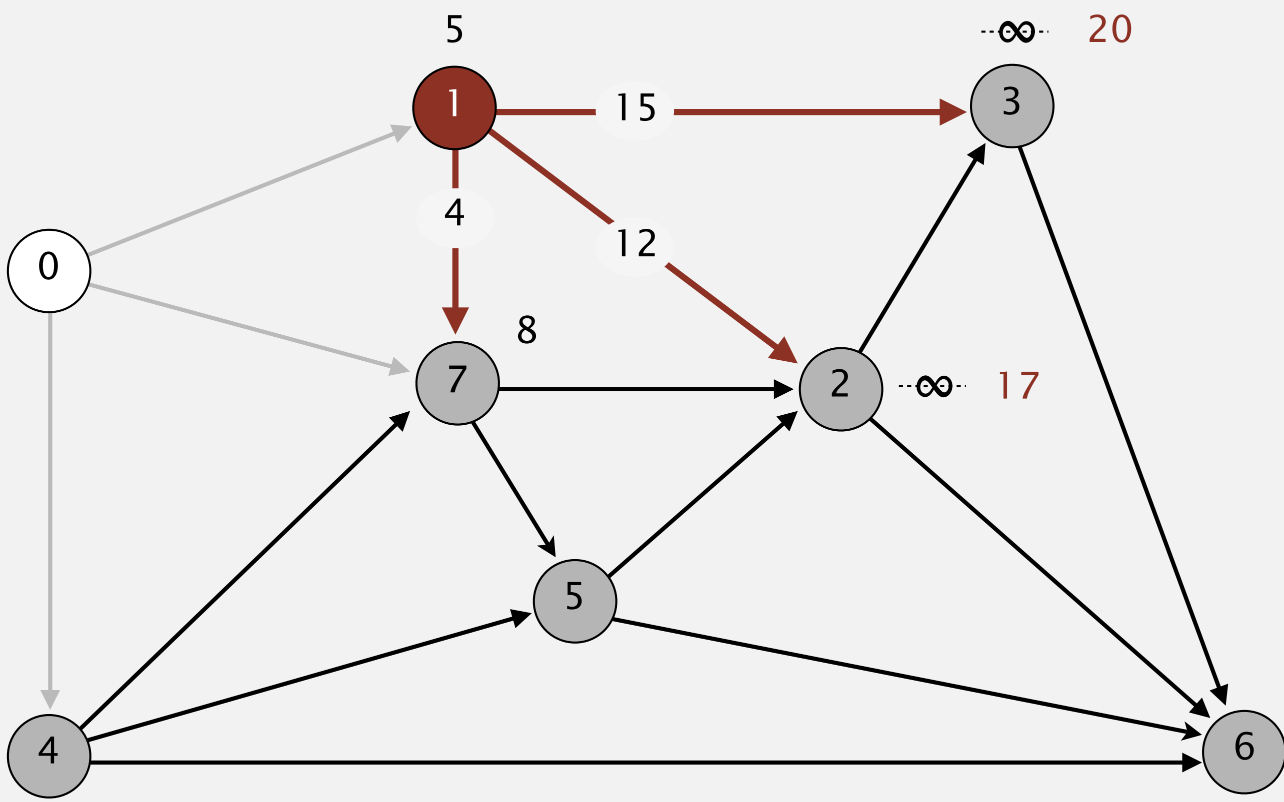


v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2		
3		
4	9.0	0→4
5		
6		
7	8.0	0→7

relax all edges adjacent from 1

Dijkstra's algorithm demo

- 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 adjacent from that vertex.

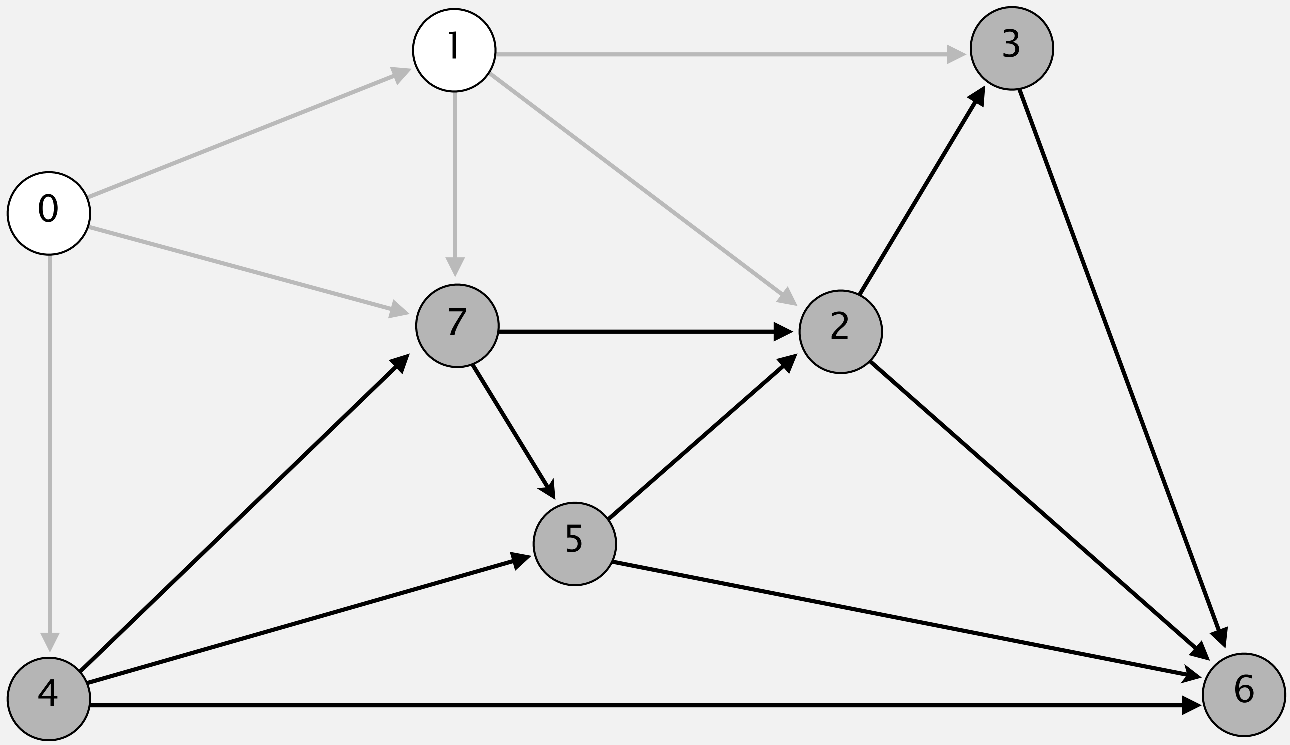


v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5		
6		
7	8.0 ✔	0→7

relax all edges adjacent from 1

Dijkstra's algorithm demo

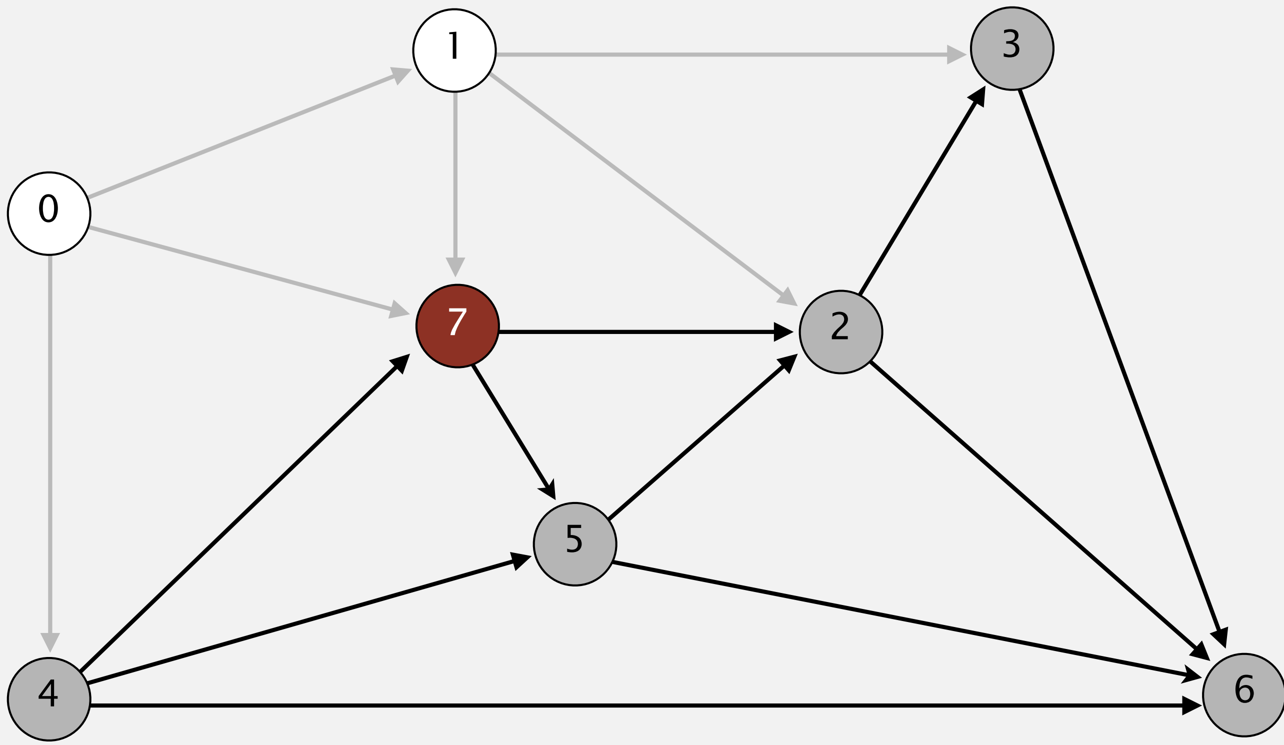
- Consider vertices in increasing order of distance from `s` (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges adjacent from that vertex.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5		
6		
7	8.0	0→7

Dijkstra's algorithm demo

- 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 adjacent from that vertex.

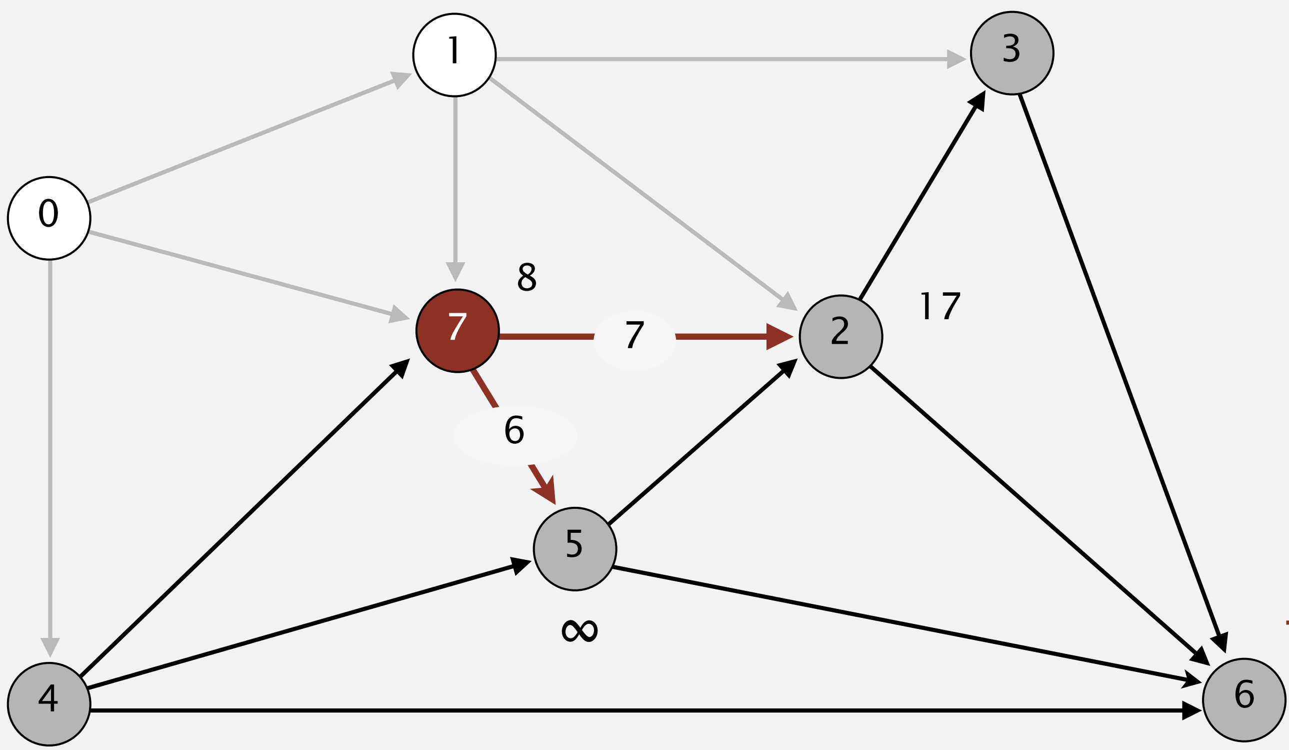


v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5		
6		
7	8.0	0→7

choose vertex 7

Dijkstra's algorithm demo

- 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 adjacent from that vertex.

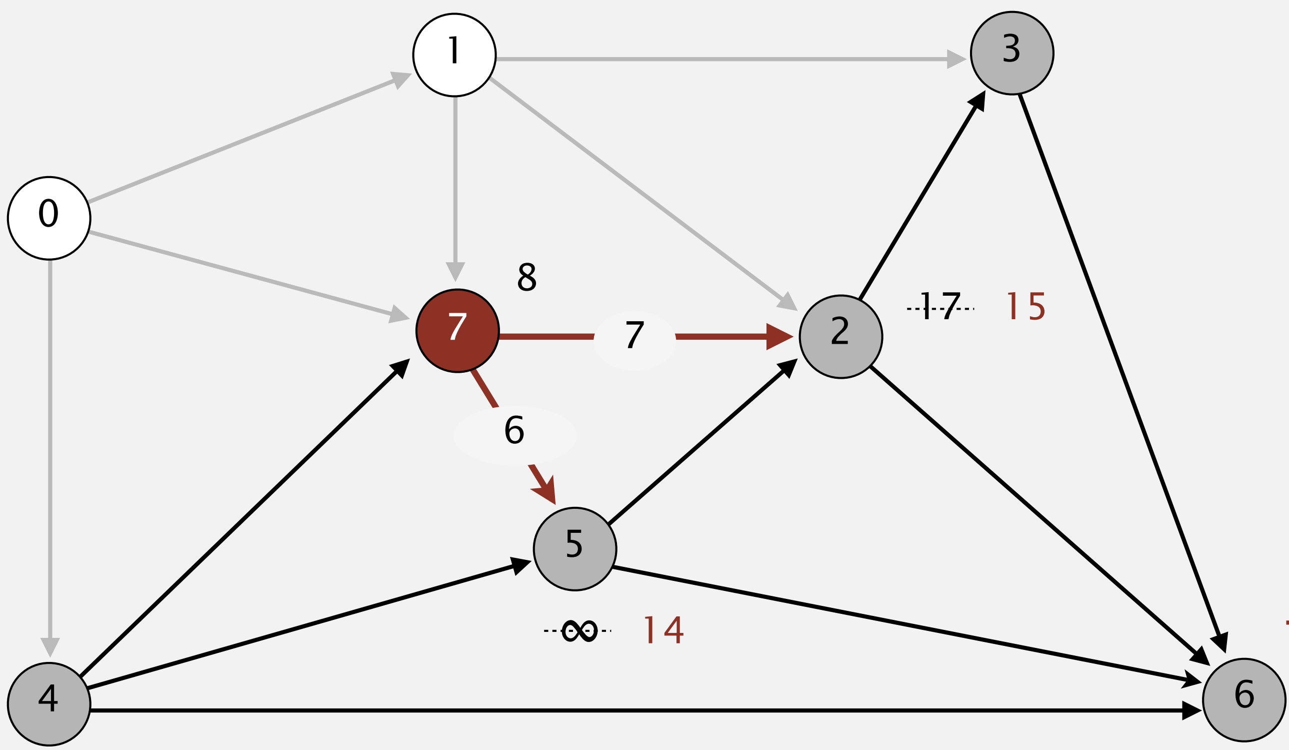


v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5		
6		
7	8.0	0→7

relax all edges adjacent from 7

Dijkstra's algorithm demo

- 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 adjacent from that vertex.

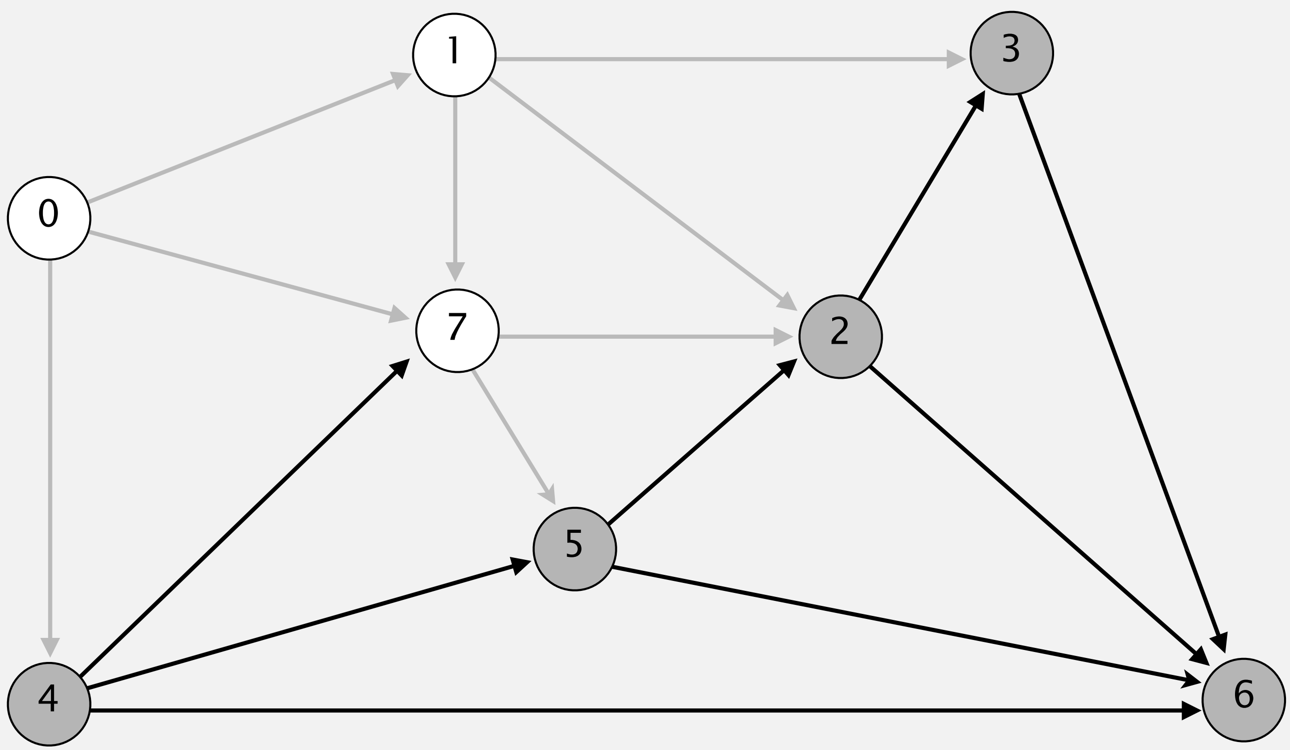


v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	15.0	7→2
3	20.0	1→3
4	9.0	0→4
5	14.0	7→5
6		
7	8.0	0→7

relax all edges adjacent from 7

Dijkstra's algorithm demo

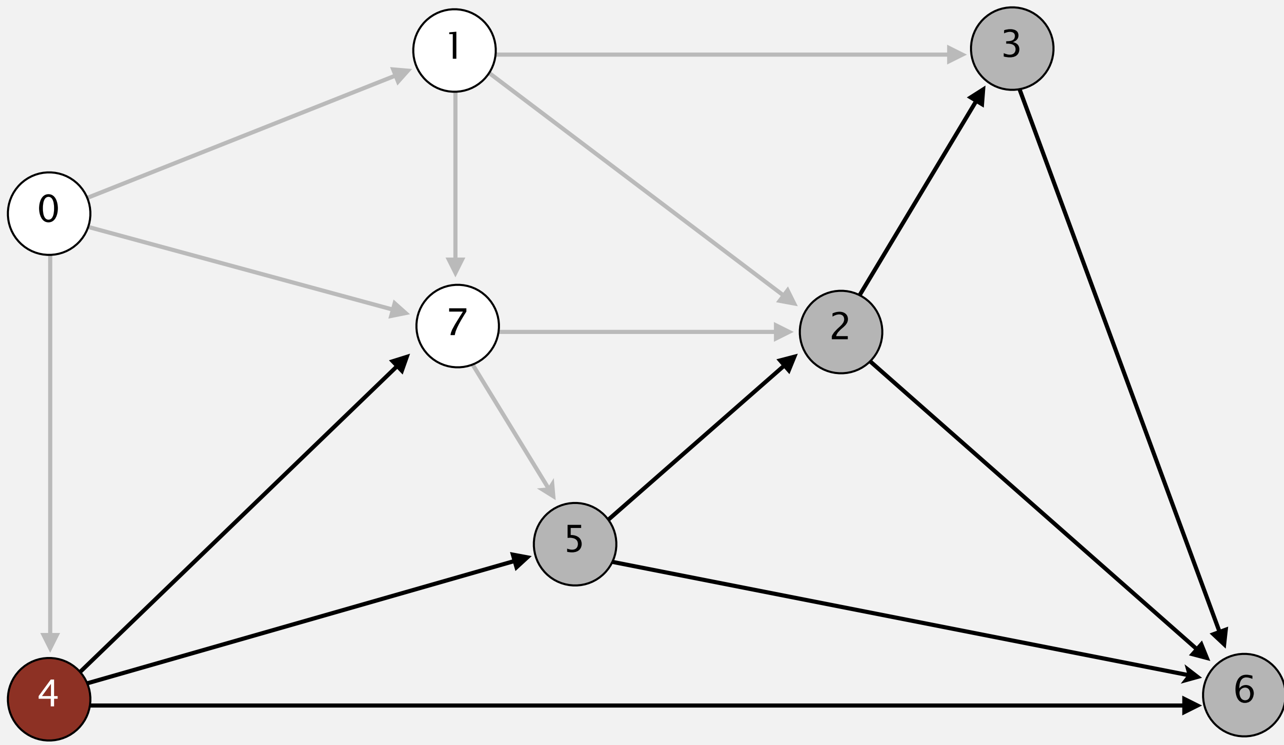
- Consider vertices in increasing order of distance from `s` (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges adjacent from that vertex.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	15.0	7→2
3	20.0	1→3
4	9.0	0→4
5	14.0	7→5
6		
7	8.0	0→7

Dijkstra's algorithm demo

- 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 adjacent from that vertex.

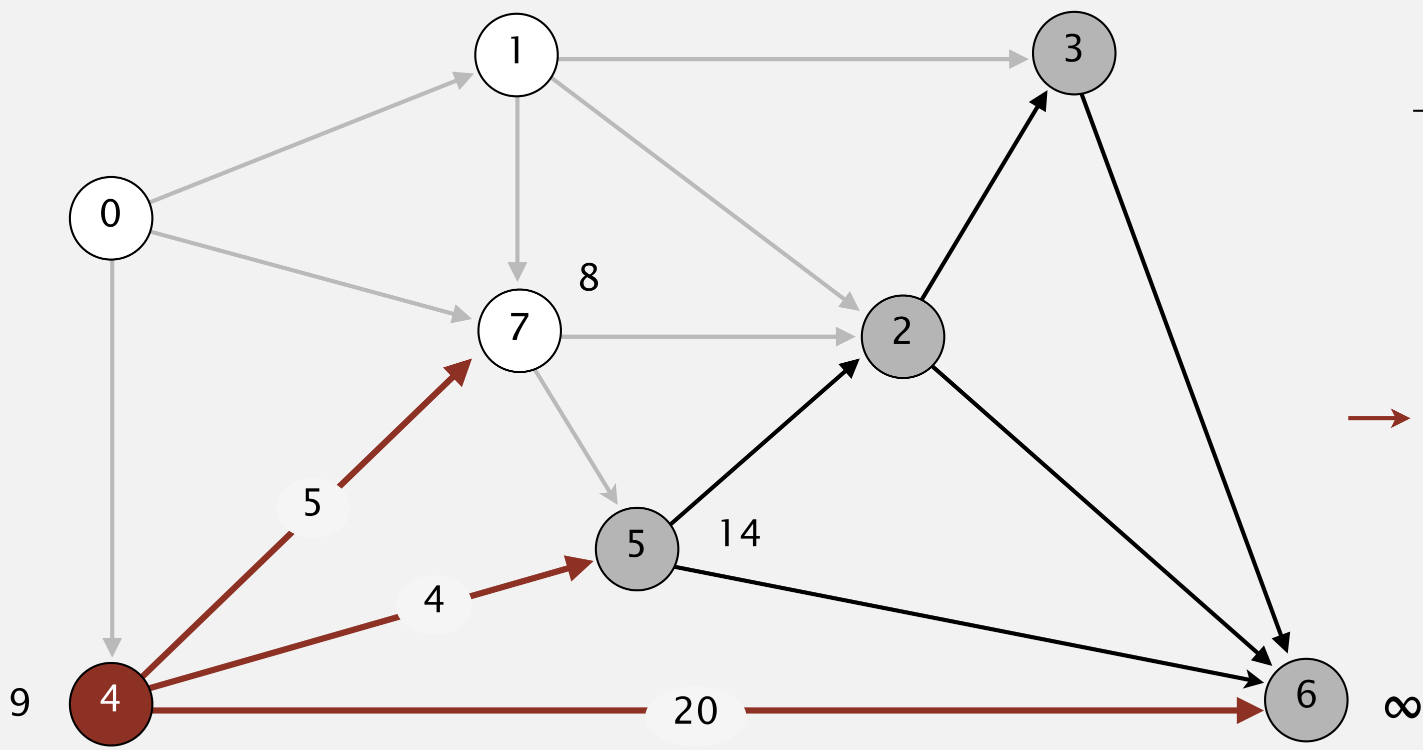


v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	15.0	7→2
3	20.0	1→3
4	9.0	0→4
5	14.0	7→5
6		
7	8.0	0→7

select vertex 4

Dijkstra's algorithm demo

- 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 adjacent from that vertex.

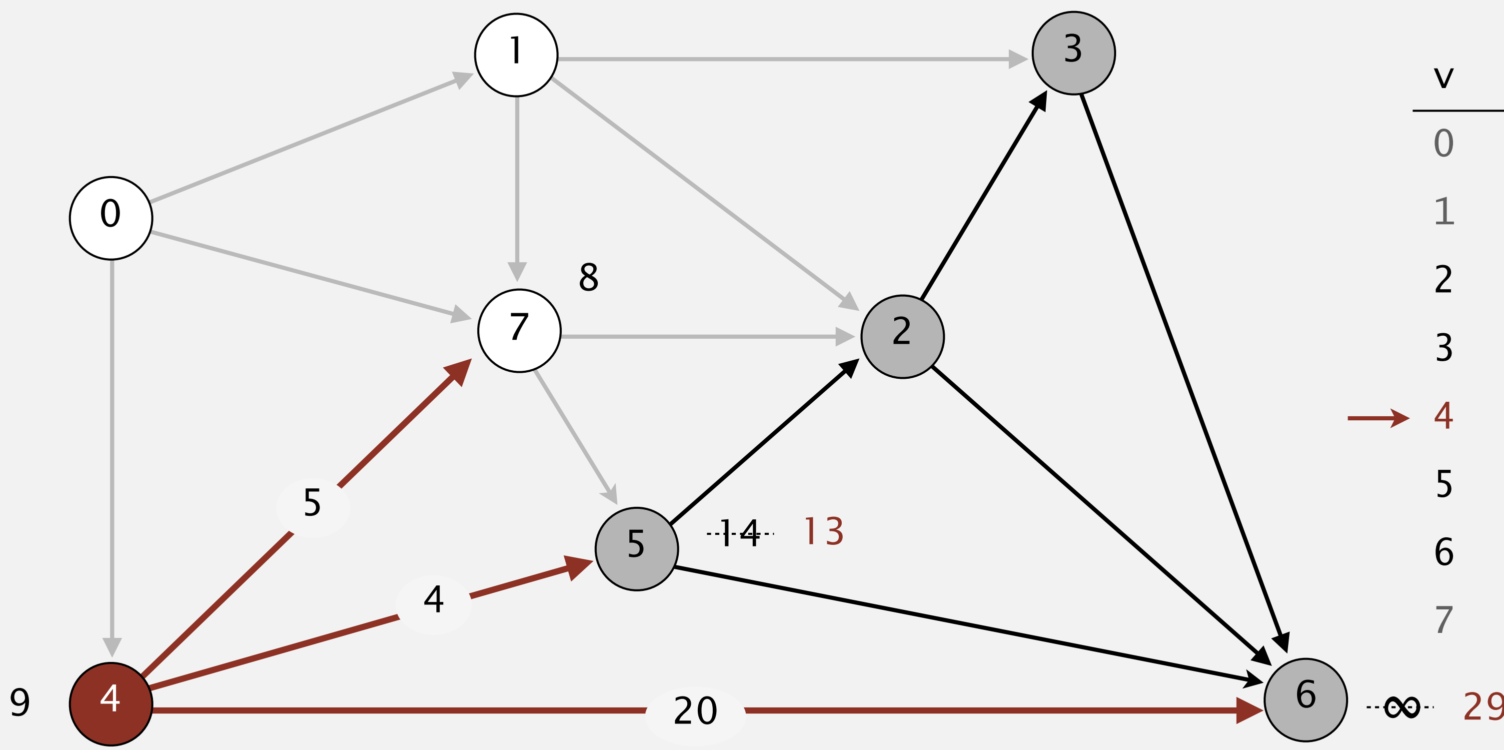


v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	15.0	7→2
3	20.0	1→3
4	9.0	0→4
5	14.0	7→5
6		
7	8.0	0→7

relax all edges adjacent from 4

Dijkstra's algorithm demo

- 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 adjacent from that vertex.

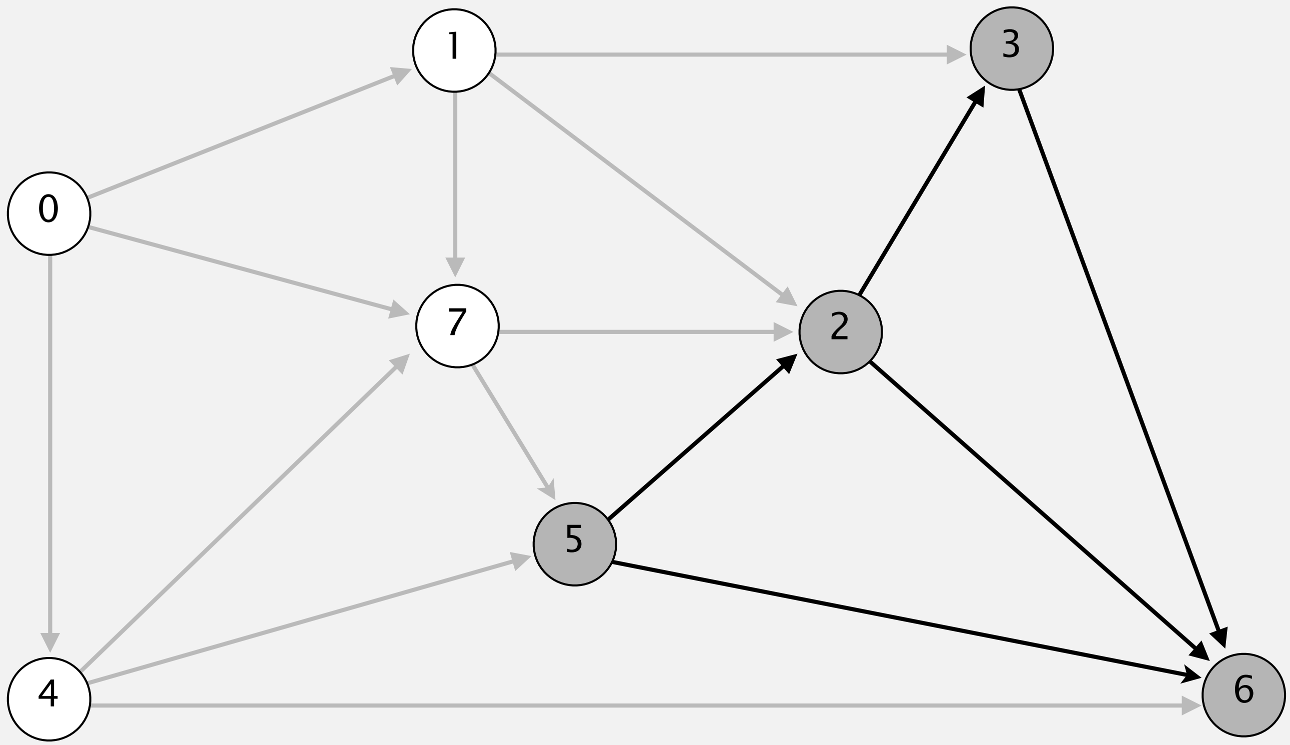


	v	distTo[]		edgeTo[]
	0	0.0		-
	1	5.0		0→1
	2	15.0		7→2
	3	20.0		1→3
→	4	9.0		0→4
	5	13.0		4→5
	6	29.0		4→6
	7	8.0	✔	0→7

relax all edges adjacent from 4

Dijkstra's algorithm demo

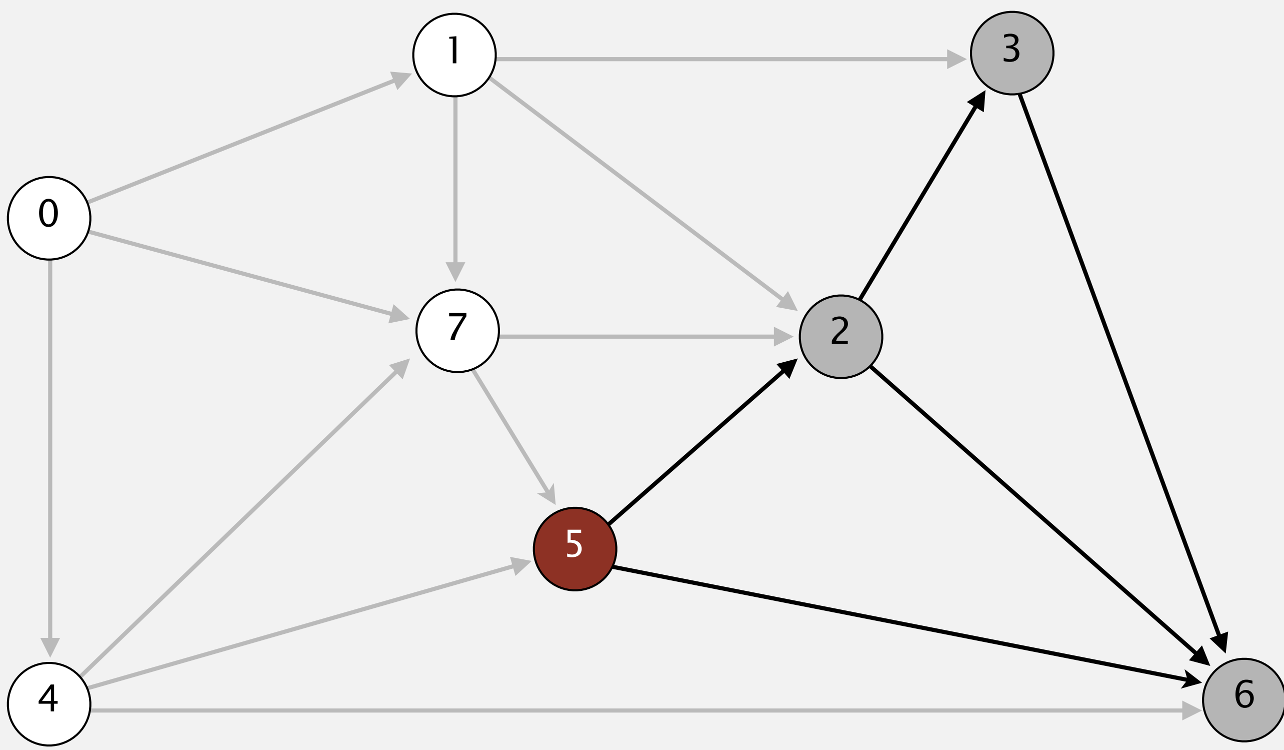
- Consider vertices in increasing order of distance from `s` (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges adjacent from that vertex.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	15.0	7→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	29.0	4→6
7	8.0	0→7

Dijkstra's algorithm demo

- 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 adjacent from that vertex.

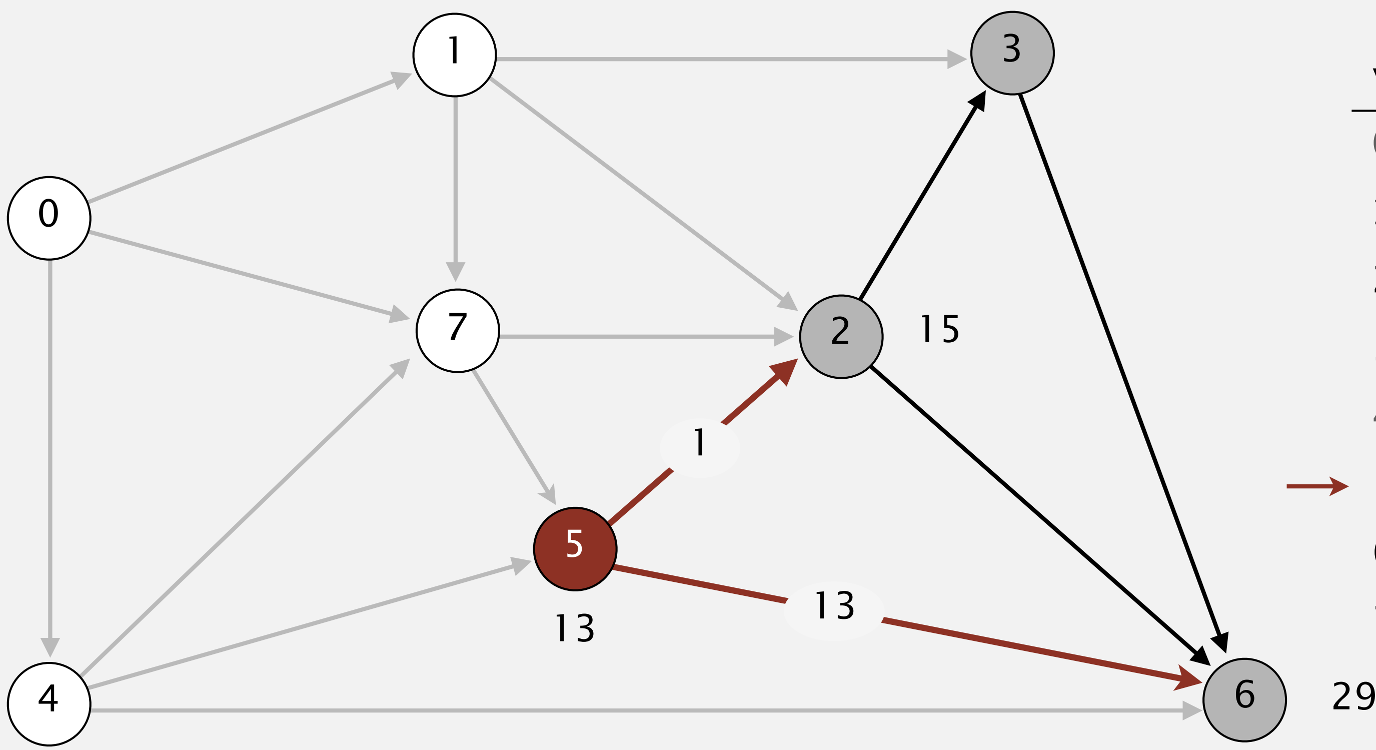


v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	15.0	7→2
3	20.0	1→3
4	9.0	0→4
→ 5	13.0	4→5
6	29.0	4→6
7	8.0	0→7

select vertex 5

Dijkstra's algorithm demo

- 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 adjacent from that vertex.

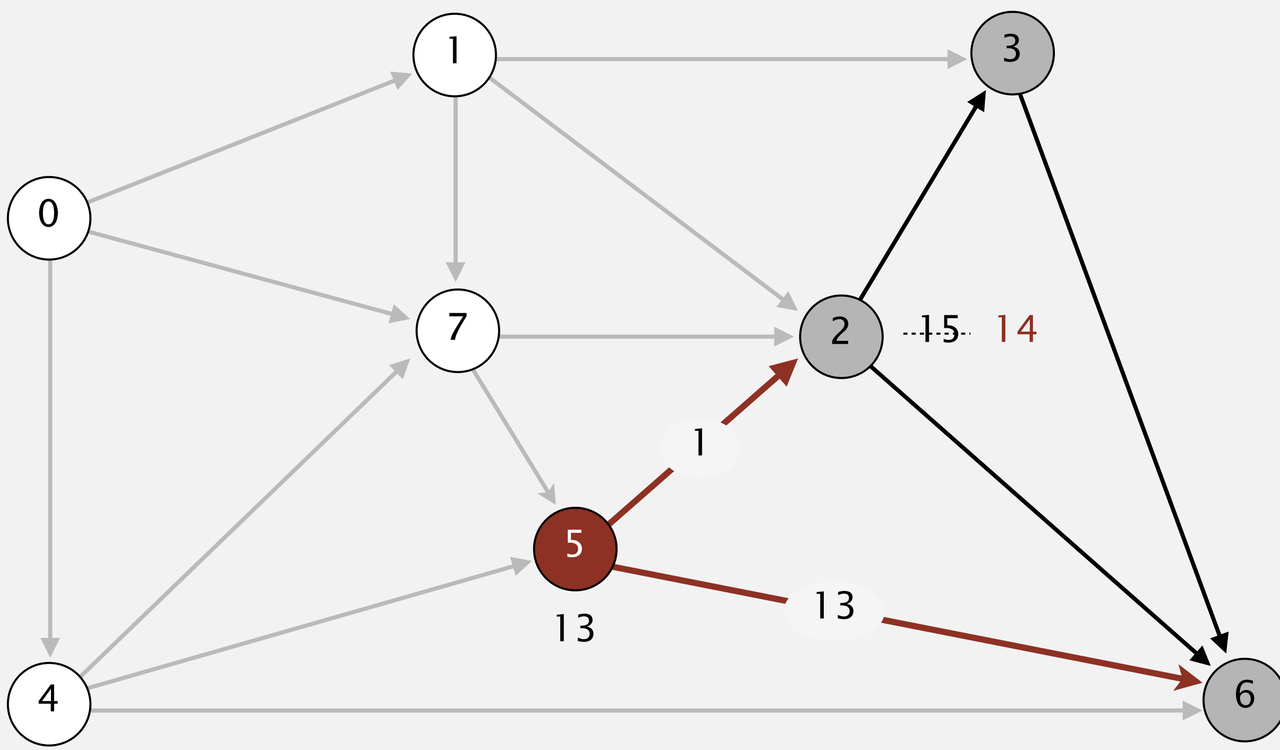


v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	15.0	7→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	29.0	4→6
7	8.0	0→7

relax all edges adjacent from 5

Dijkstra's algorithm demo

- 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 adjacent from that vertex.

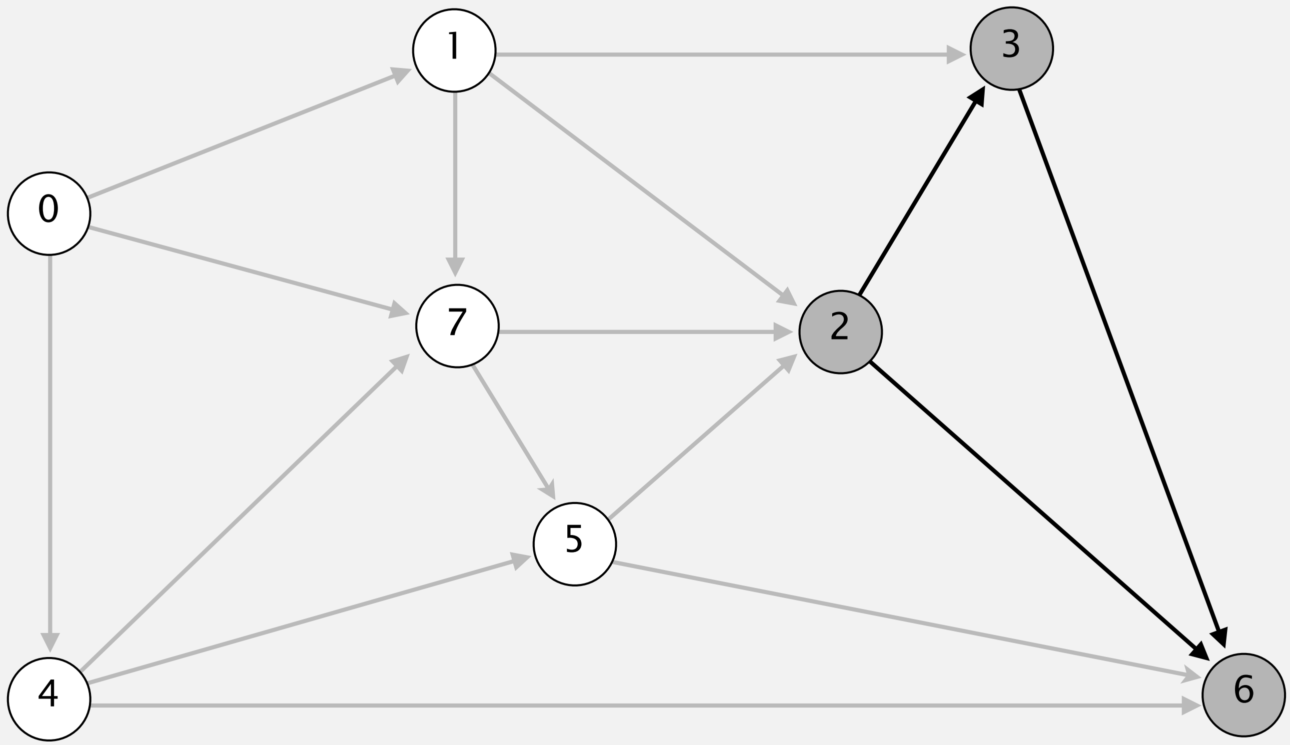


	v	distTo[]	edgeTo[]
	0	0.0	-
	1	5.0	0→1
	2	14.0	5→2
	3	20.0	1→3
	4	9.0	0→4
→	5	13.0	4→5
	6	26.0	5→6
	7	8.0	0→7

relax all edges adjacent from 5

Dijkstra's algorithm demo

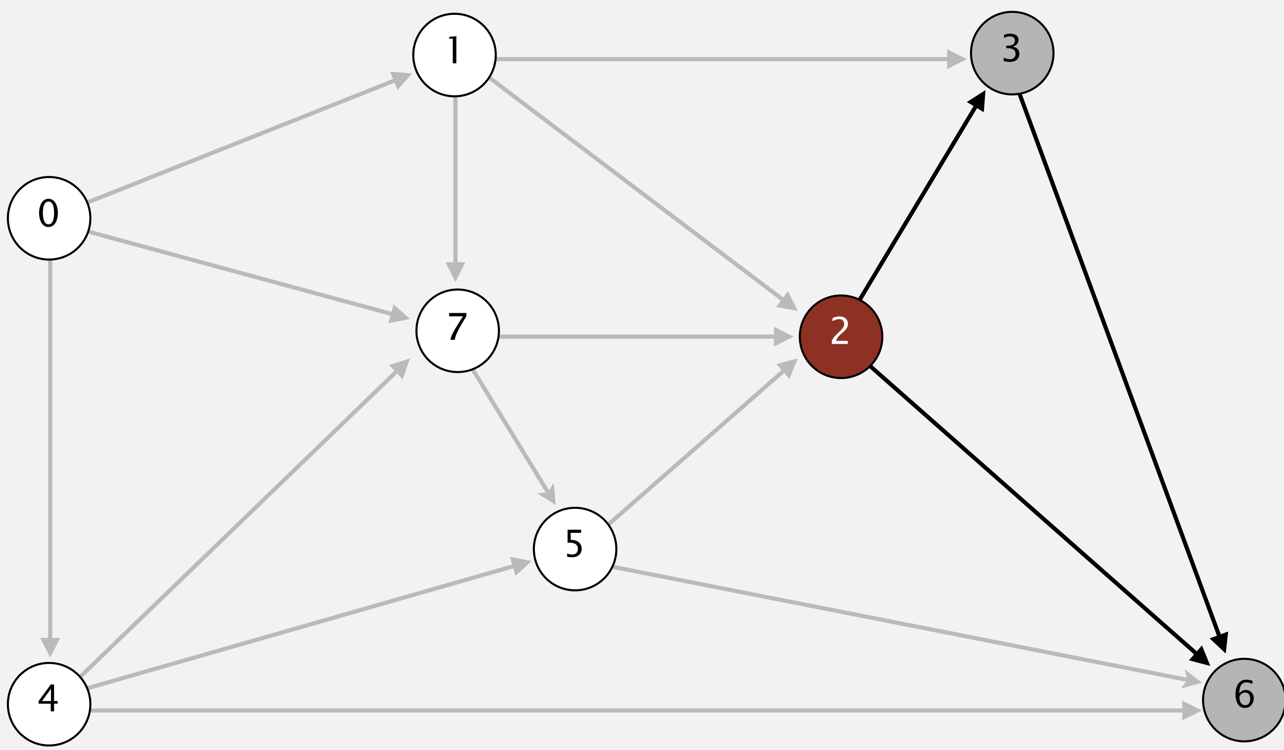
- Consider vertices in increasing order of distance from `s` (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges adjacent from that vertex.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

Dijkstra's algorithm demo

- 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 adjacent from that vertex.

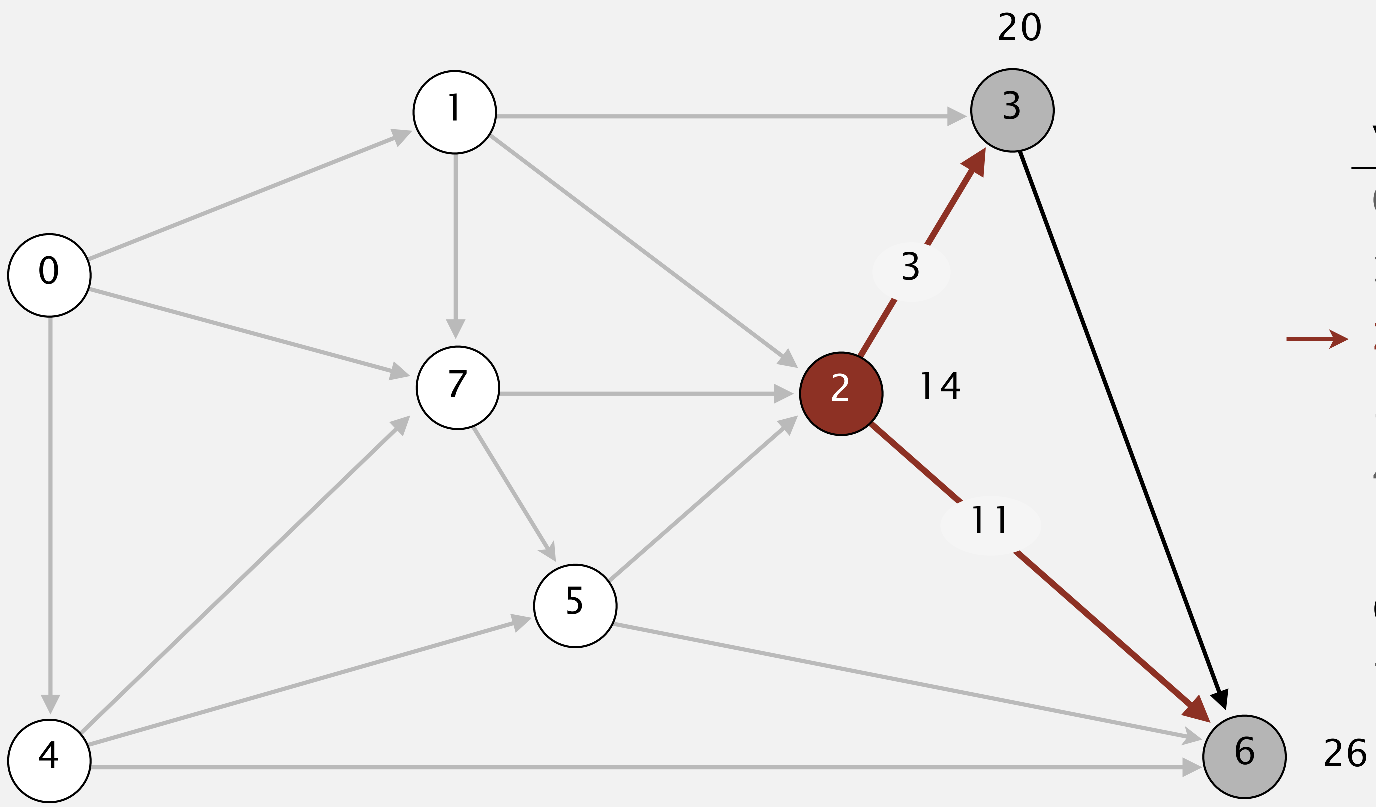


	v	distTo[]	edgeTo[]
	0	0.0	-
	1	5.0	0→1
→	2	14.0	5→2
	3	20.0	1→3
	4	9.0	0→4
	5	13.0	4→5
	6	26.0	5→6
	7	8.0	0→7

select vertex 2

Dijkstra's algorithm demo

- 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 adjacent from that vertex.

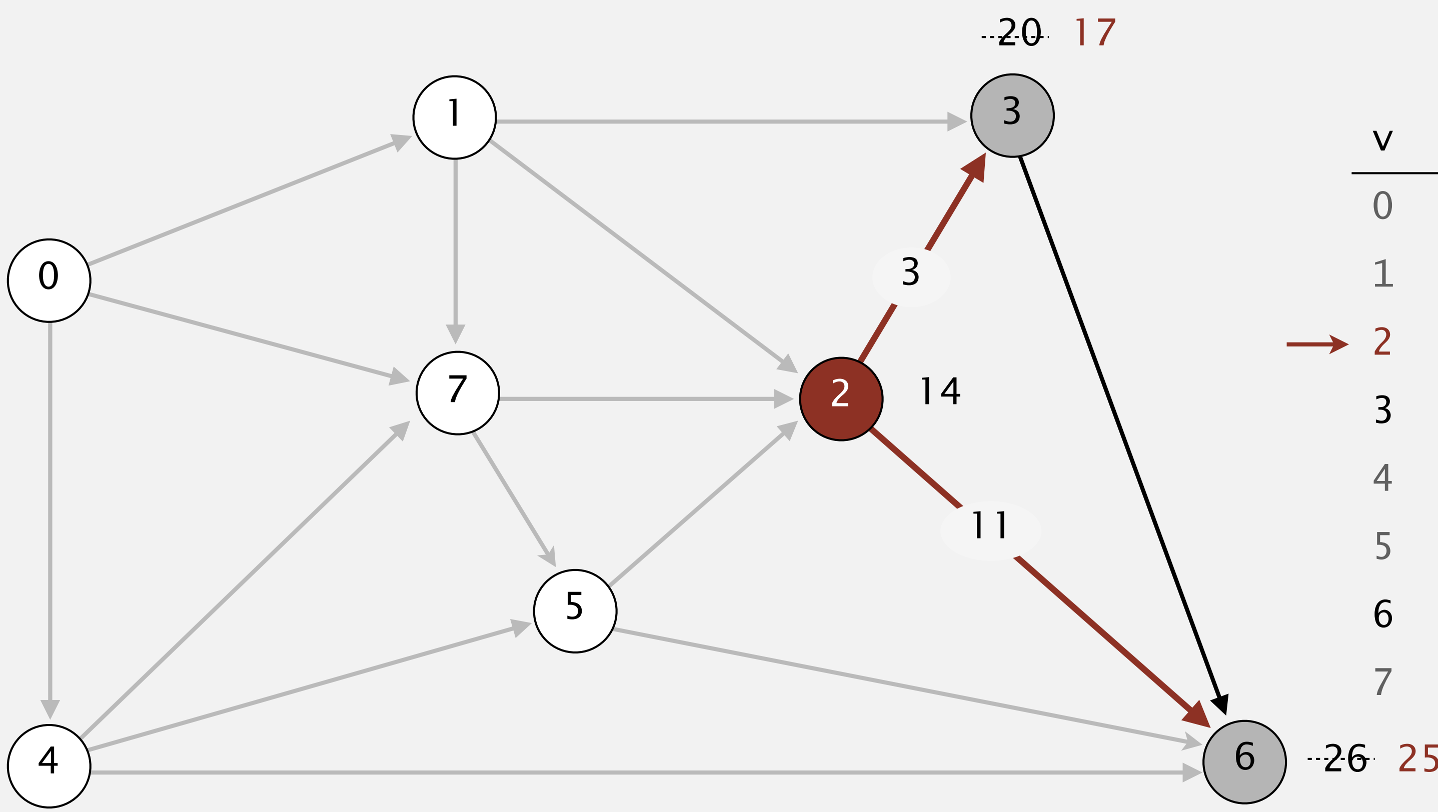


	v	distTo[]	edgeTo[]
	0	0.0	-
	1	5.0	0→1
→	2	14.0	5→2
	3	20.0	1→3
	4	9.0	0→4
	5	13.0	4→5
	6	26.0	5→6
	7	8.0	0→7

relax all edges adjacent from 2

Dijkstra's algorithm demo

- 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 adjacent from that vertex.

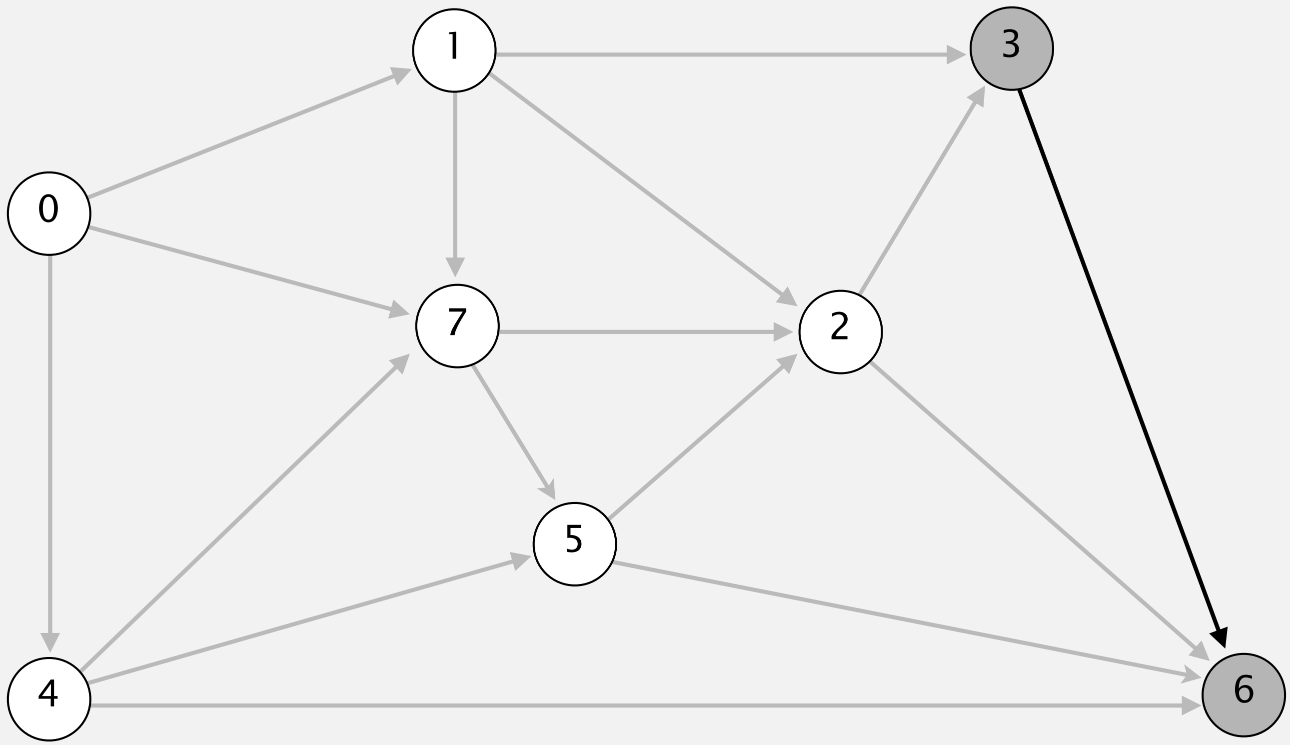


v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

relax all edges adjacent from 2

Dijkstra's algorithm demo

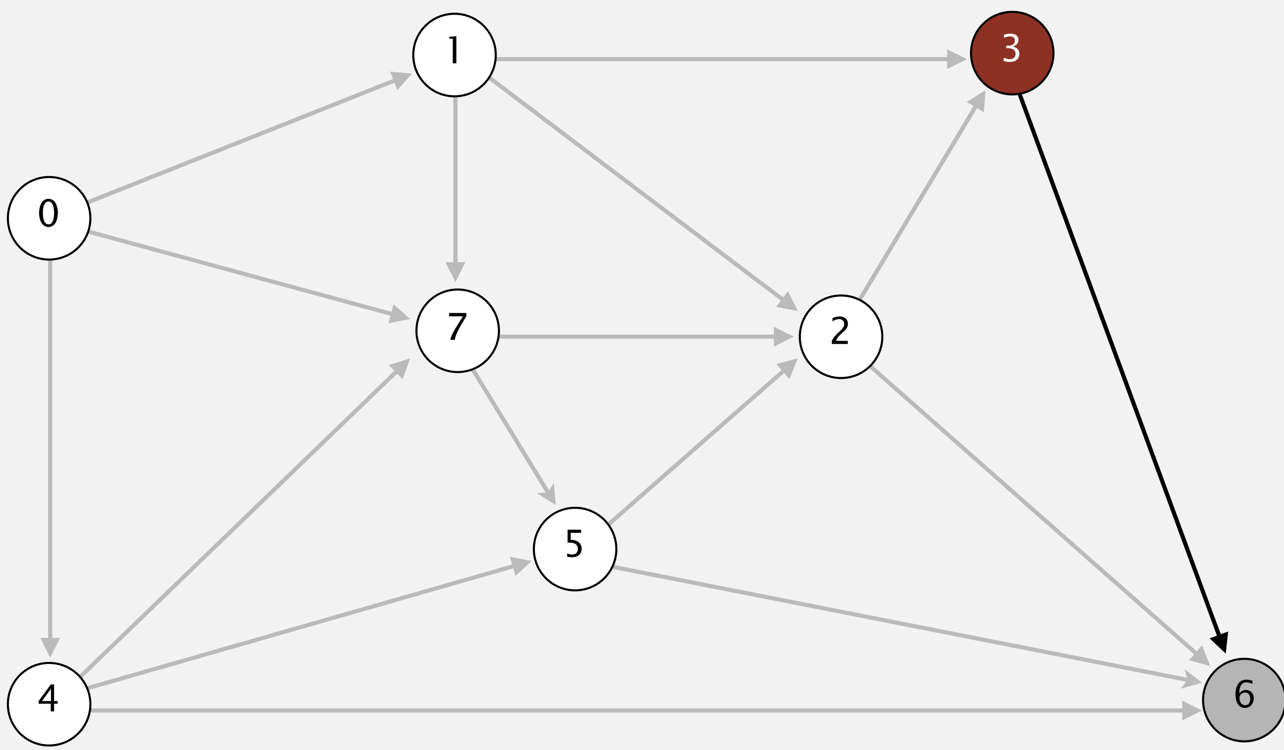
- Consider vertices in increasing order of distance from `s` (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges adjacent from that vertex.



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

Dijkstra's algorithm demo

- 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 adjacent from that vertex.

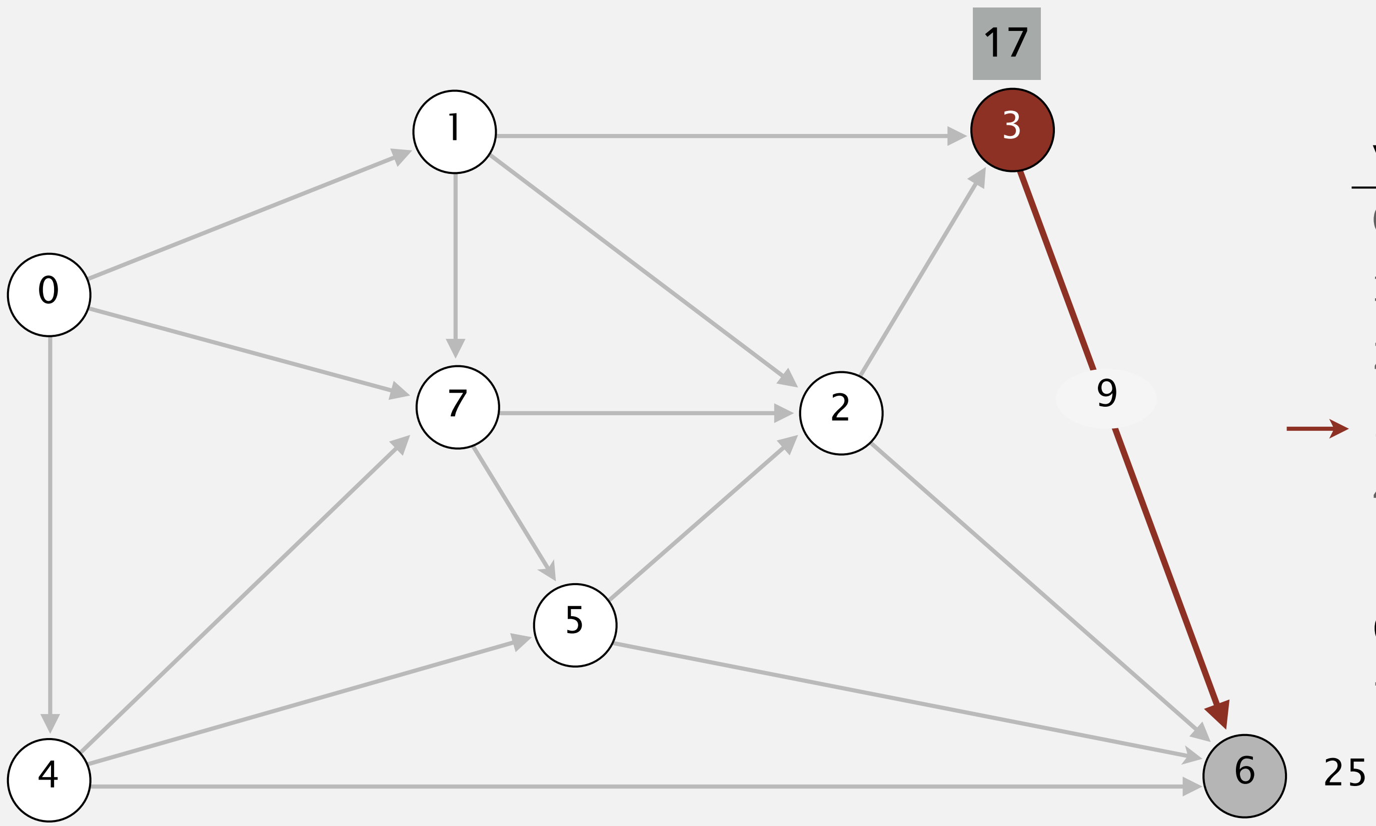


	v	distTo[]	edgeTo[]
	0	0.0	-
	1	5.0	0→1
	2	14.0	5→2
→	3	17.0	2→3
	4	9.0	0→4
	5	13.0	4→5
	6	25.0	2→6
	7	8.0	0→7

select vertex 3

Dijkstra's algorithm demo

- 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 adjacent from that vertex.

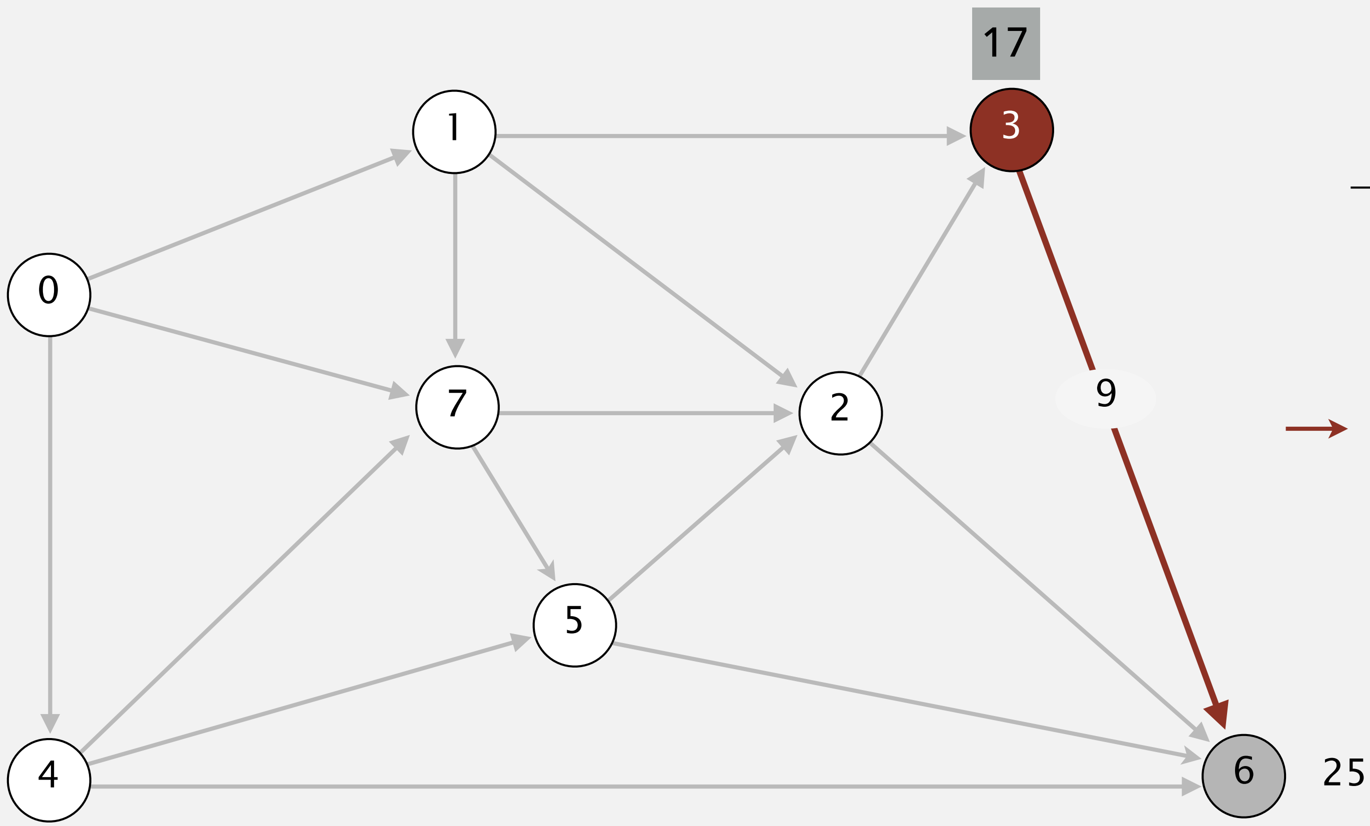


v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

relax all edges adjacent from 3

Dijkstra's algorithm demo

- 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 adjacent from that vertex.

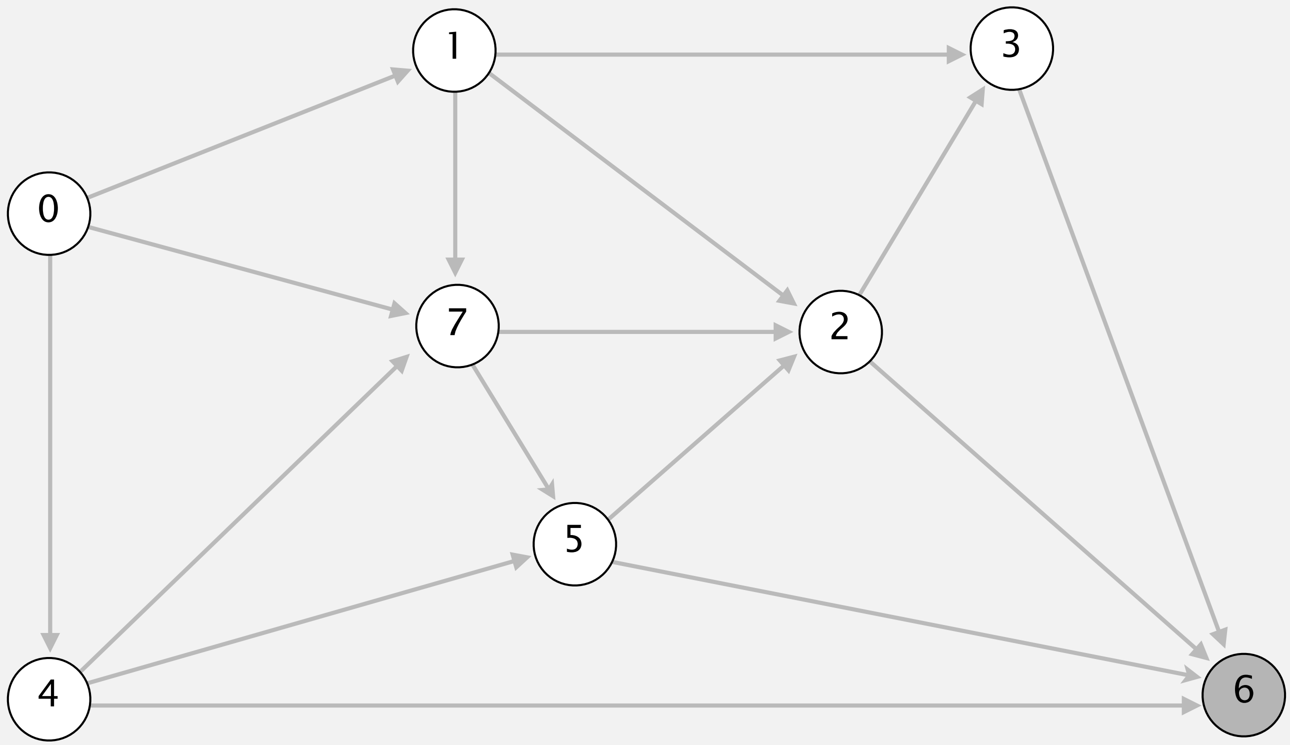


v	distTo[]		edgeTo[]
0	0.0		-
1	5.0		0→1
2	14.0		5→2
3	17.0		2→3
4	9.0		0→4
5	13.0		4→5
6	25.0	✔	2→6
7	8.0		0→7

relax all edges adjacent from 3

Dijkstra's algorithm demo

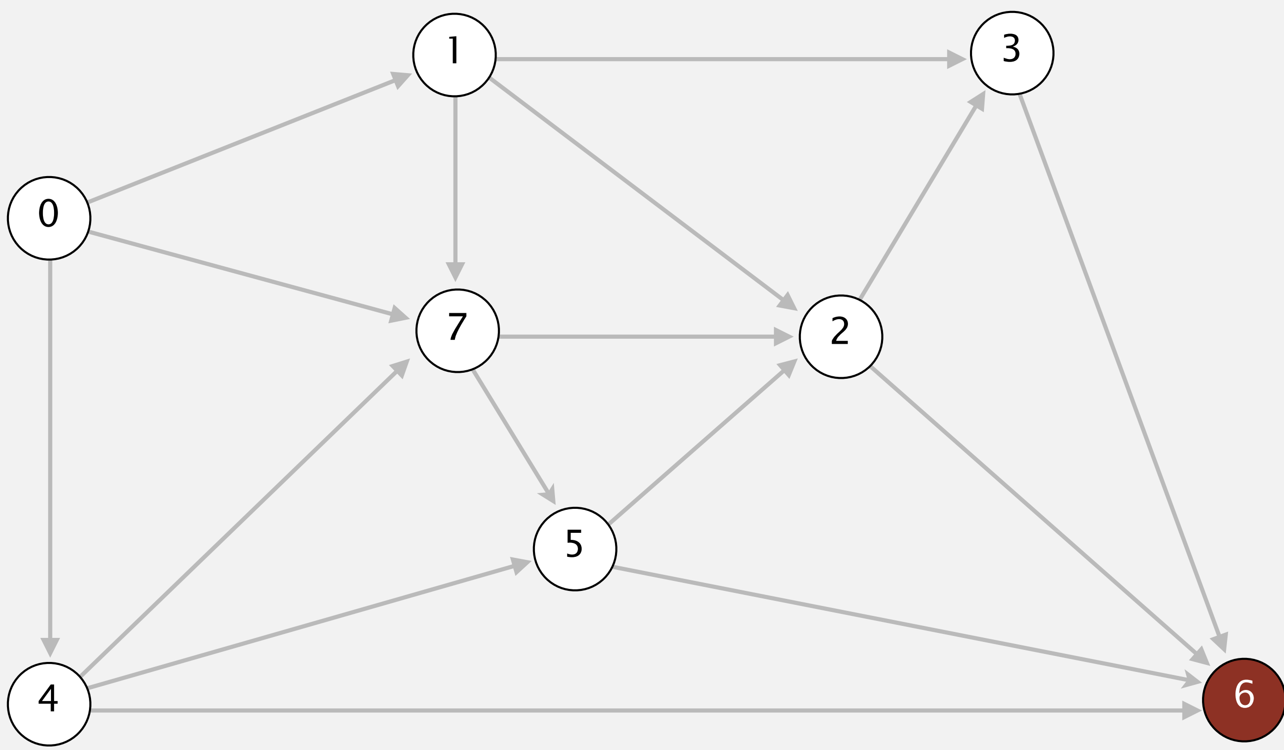
- Consider vertices in increasing order of distance from `s`
  (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges adjacent from that vertex.



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

Dijkstra's algorithm demo

- 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 adjacent from that vertex.

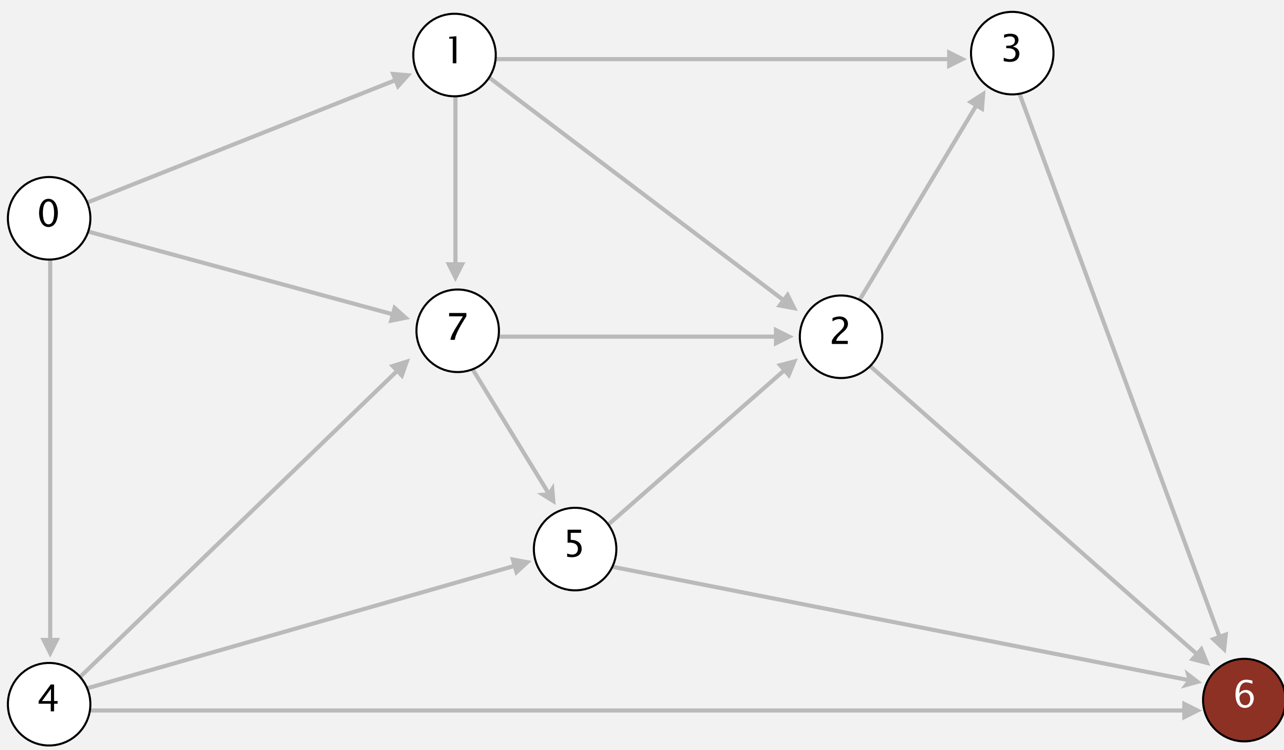


v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
→ 6	25.0	2→6
7	8.0	0→7

select vertex 6

Dijkstra's algorithm demo

- 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 adjacent from that vertex.

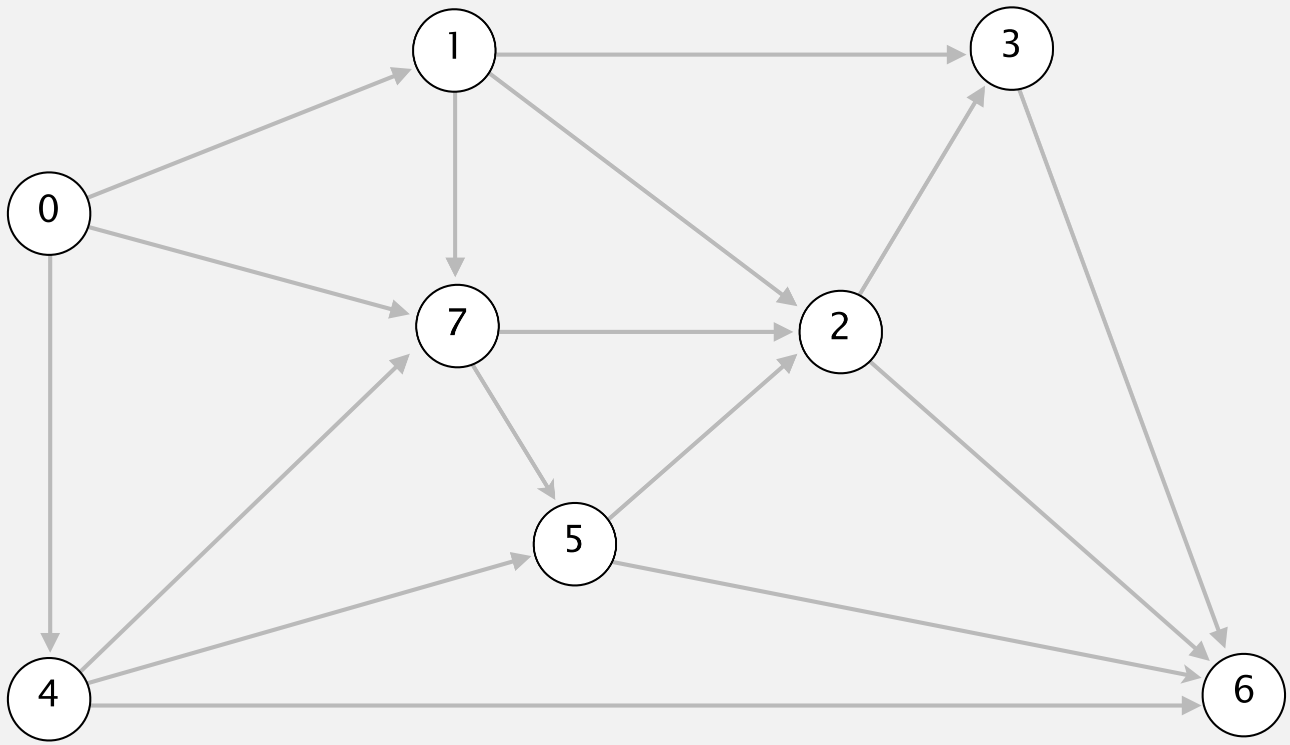


v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

relax all edges adjacent from 6

Dijkstra's algorithm demo

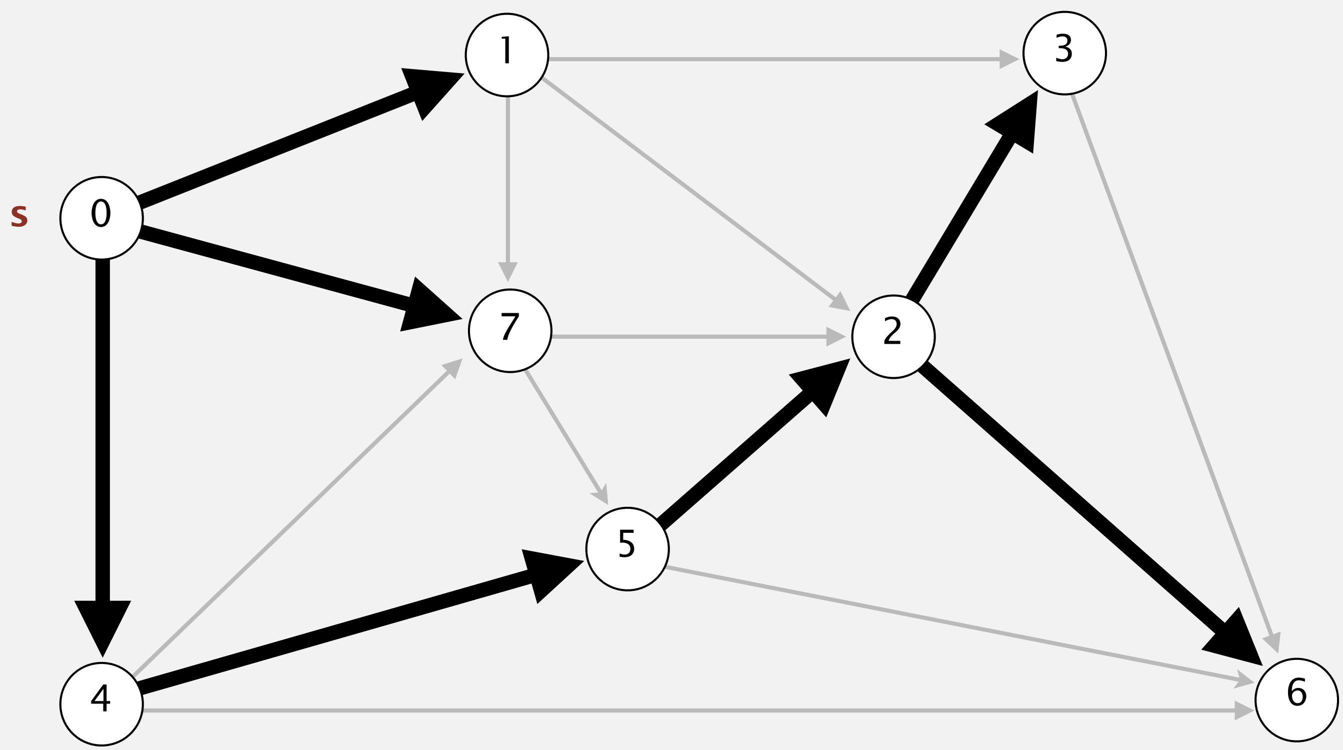
- Consider vertices in increasing order of distance from `s` (non-tree vertex with the lowest `distTo[]` value).
- Add vertex to tree and relax all edges adjacent from that vertex.



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

Dijkstra's algorithm demo

- 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 adjacent from that vertex.



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

Indexed min-priority queue (Section 2.4 in textbook)

- Associate an index between 0 and n-1 with each key in a priority queue.
    - Insert a key associated with a given index.
    - Delete a minimum key and return associated index.
    - Decrease the key associated with a given index.
- `public class IndexMinPQ<Key extends Comparable<Key>>`
    - `IndexMinPQ(int n)`
        - Create indexed PQ with indices 0,1,...n-1
    - `void insert(int i, Key key)`
        - Associate key with index i.
    - `int delMin()`
        - Remove a minimal key and return its associated index.
    - `void decreaseKey(int i, Key key)`
        - Decrease the key with index i to the specified value.

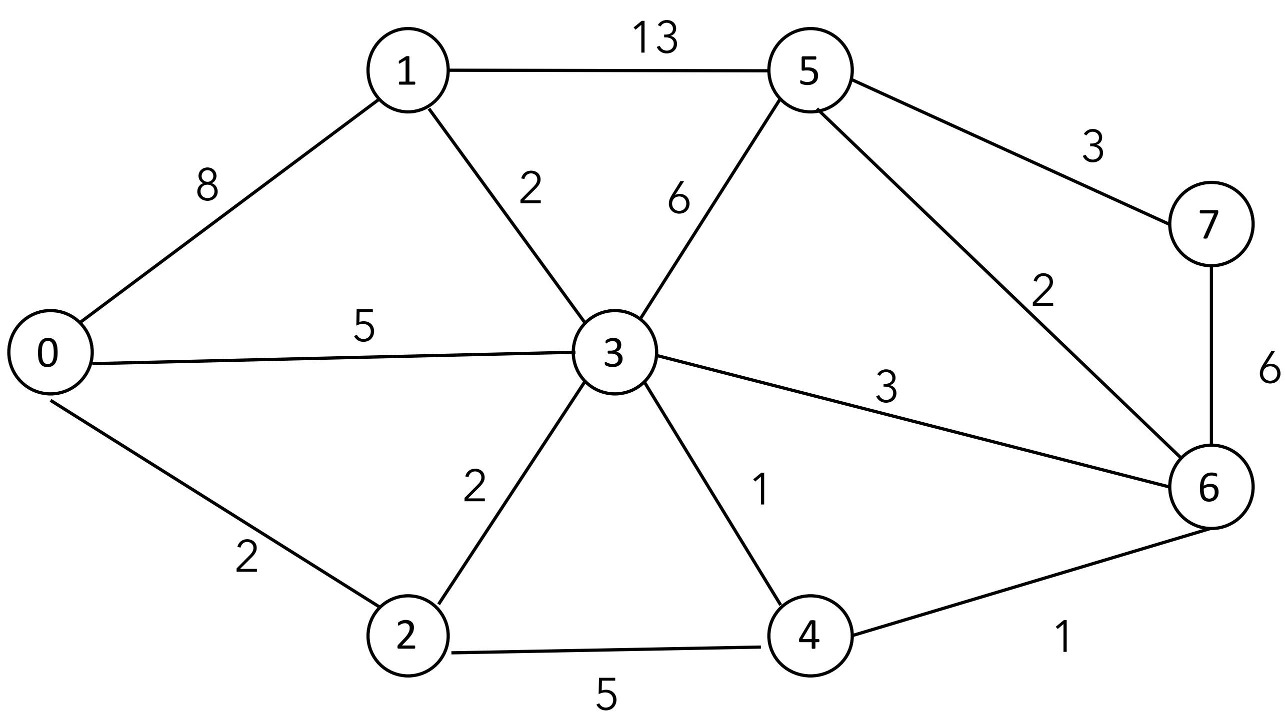
```
public class DijkstraSP {
    private double[] distTo;          // distTo[v] = distance  of shortest s->v path
    private DirectedEdge[] edgeTo;    // edgeTo[v] = last edge on shortest s->v path
    private IndexMinPQ<Double> pq;    // priority queue of vertices


    public DijkstraSP(EdgeWeightedDigraph G, int s) {
        distTo = new double[G.V()];
        edgeTo = new DirectedEdge[G.V()];

        for (int v = 0; v < G.V(); v++)
            distTo[v] = Double.POSITIVE_INFINITY;
        distTo[s] = 0.0;

        // relax vertices in order of distance from s
        pq = new IndexMinPQ<Double>(G.V());
        pq.insert(s, distTo[s]);
        while (!pq.isEmpty()) {
            int v = pq.delMin();
            for (DirectedEdge e : G.adj(v))
                relax(e);
        }
    }

    // relax edge e and update pq if changed
    private void relax(DirectedEdge e) {
        int v = e.from(), w = e.to();
        if (distTo[w] > distTo[v] + e.weight()) {
            distTo[w] = distTo[v] + e.weight();
            edgeTo[w] = e;
            if (pq.contains(w)) pq.decreaseKey(w, distTo[w]);
            else                pq.insert(w, distTo[w]);
        }
    }
```

Running time depends on PQ implementation

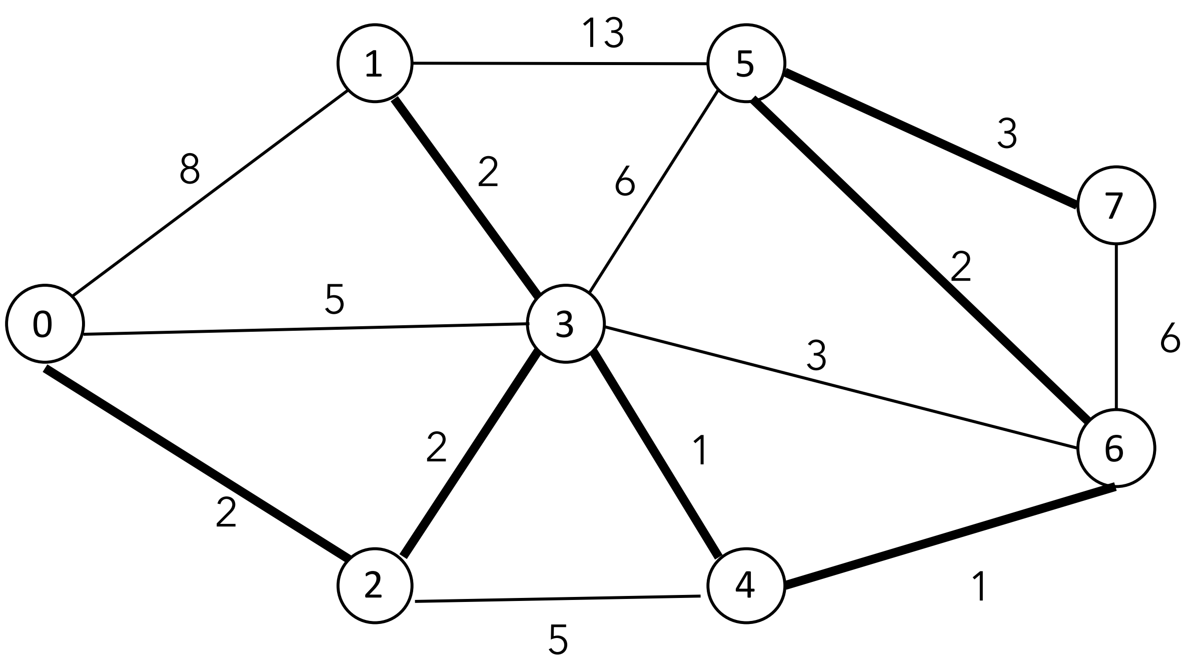
- Many variations. Assuming binary heap, running time is proportional to $|E|\log|V|$ and $|V|$ extra space.
  - Cost of insert, delete-min, decrease-key are all $\log V$.
- More complicated version with a Fibonacci heap (CS140…) takes $O(|E| + |V|\log|V|)$ time but in practice it's not worth implementing.

Practice Time

- Run Dijkstra's algorithm on the following graph with 0 being the starting vertex.



Answer



v	distTo[]	edgeTo[]
0	0	-
1	6	3->1
2	2	0->2
3	4	2->3
4	5	3->4
5	8	6->5
6	6	4->6
7	11	5->7

Lecture 27: Shortest Paths

- Introduction to Shortest Paths
- API
- Properties
- Dijkstra's Algorithm
- Belman-Ford Algorithm

Framework for shortest-paths algorithm

- Generic algorithm to compute a SPT from s
  - `distTo[v]`=∞ for each vertex v.
  - `edgeTo[v]=null` for each vertex v.
  - `distTo[s]=0`.
  - Repeat until done:
    - Relax any edge.
- `distTo[v]` is the length of a simple path from `s` to `v`.
- `distTo[v]` does not increase.

Bellman-Ford algorithm

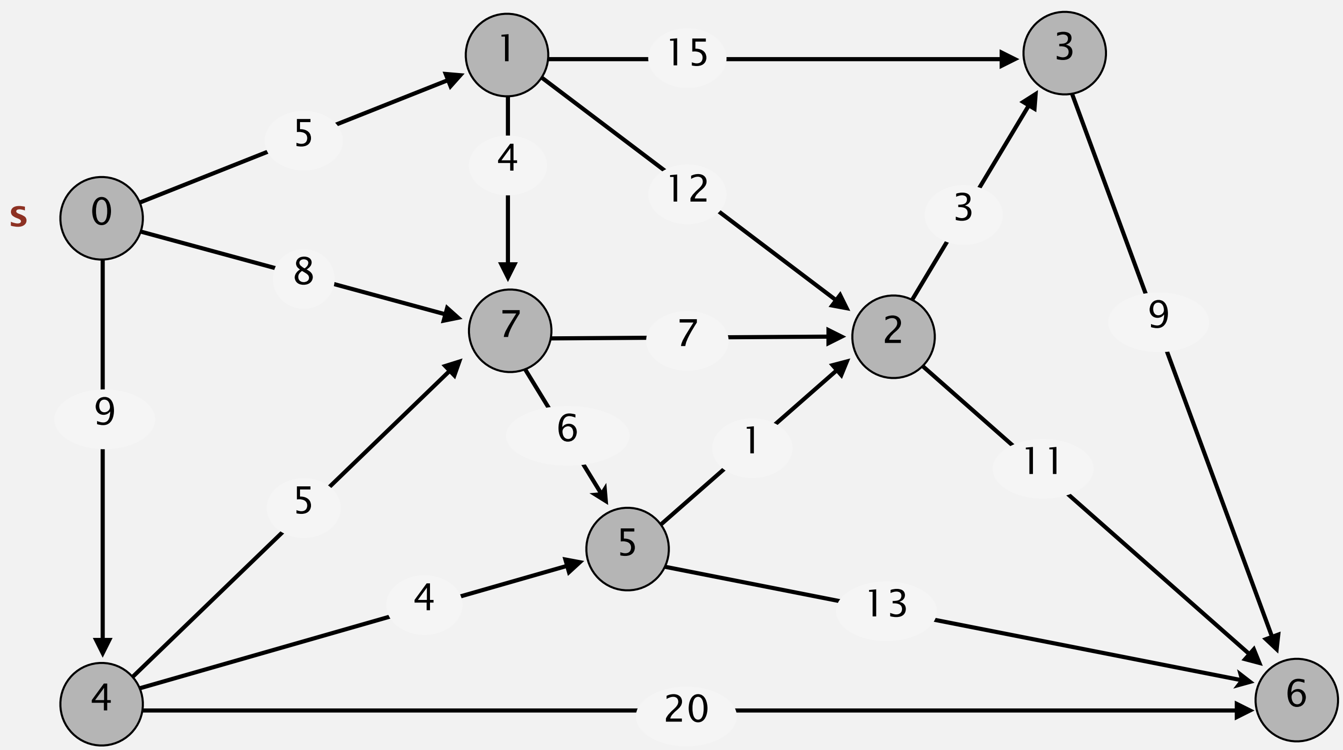
- `distTo[v]`=∞ for each vertex v.
- `edgeTo[v]=null` for each vertex v.
- `distTo[s]=0.`
- Repeat |V|-1 times:
  - Relax all edges.

http://algs4.cs.princeton.edu

BELLMAN-FORD DEMO

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.

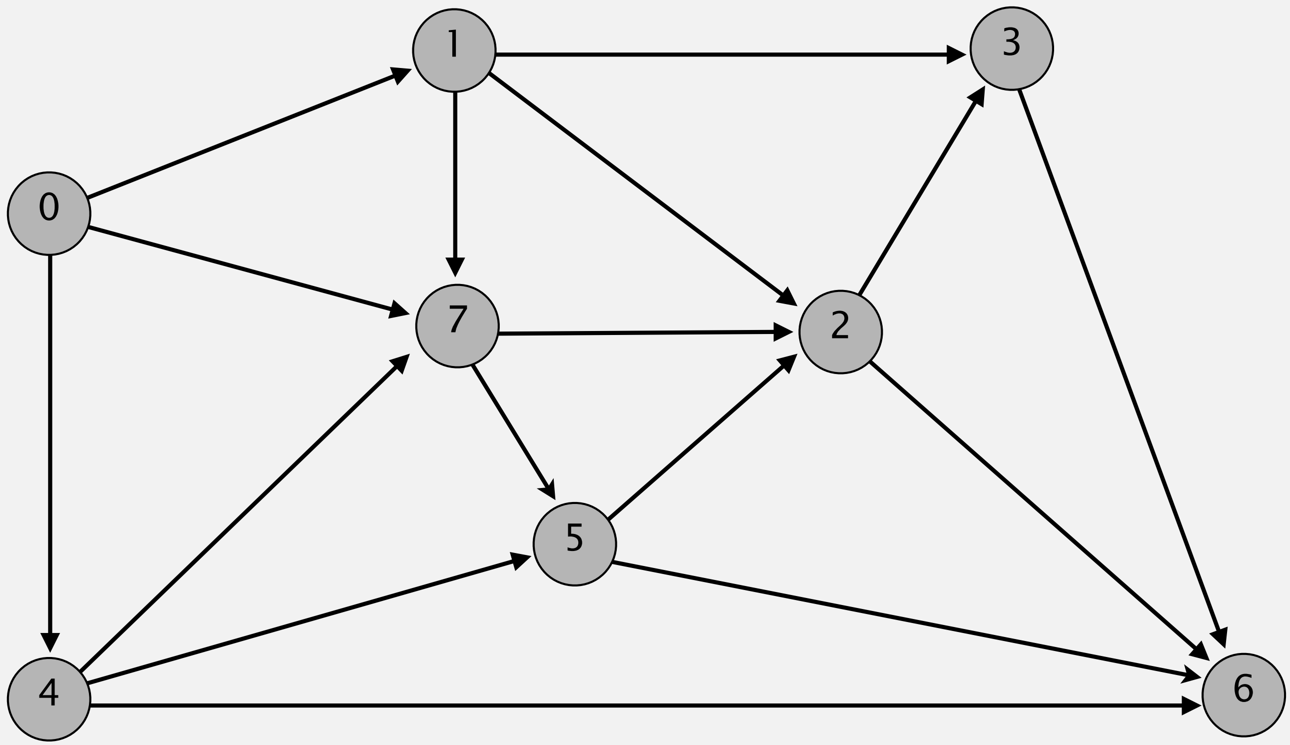


an edge-weighted digraph

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

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.

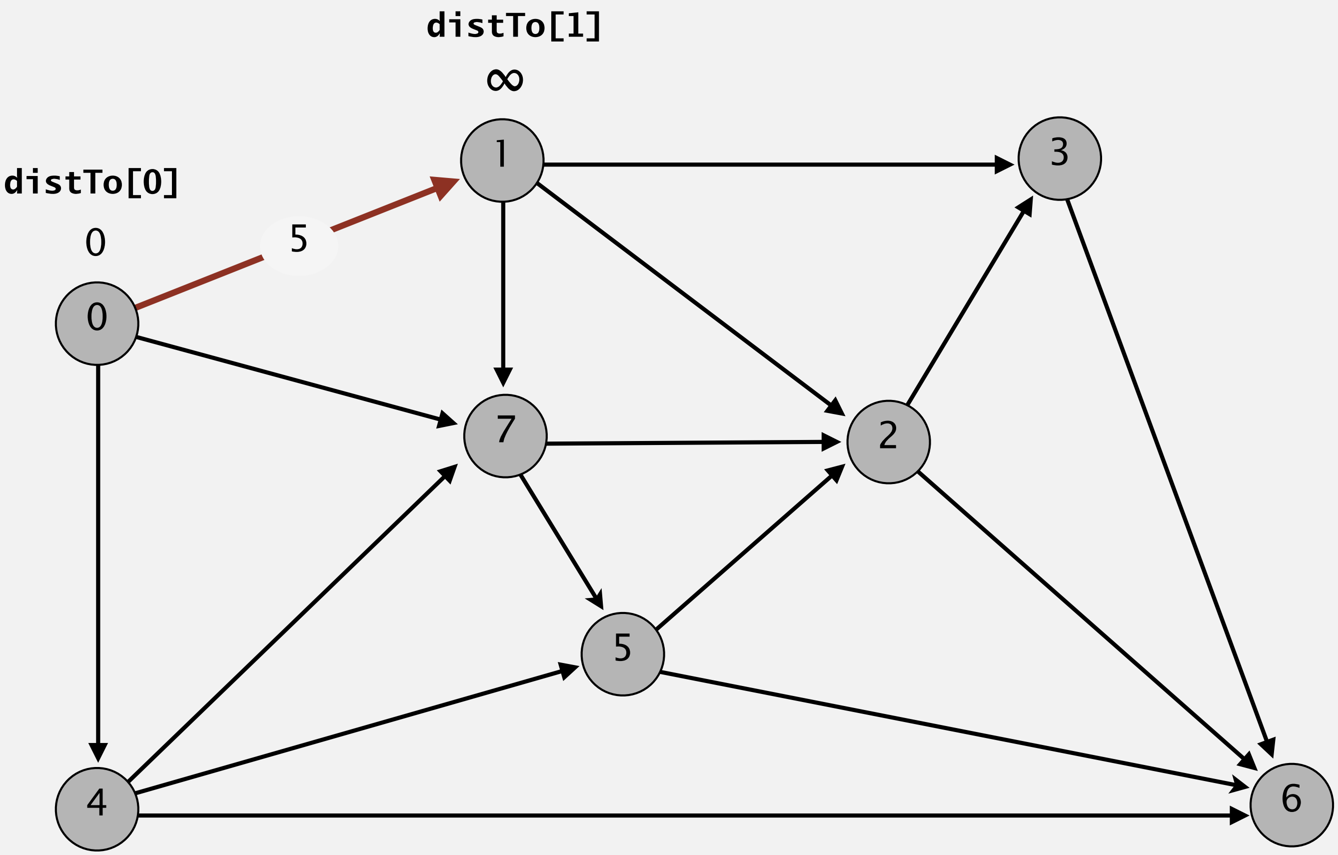


v	distTo[]	edgeTo[]
0	0.0	-
1		
2		
3		
4		
5		
6		
7		

initialize

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



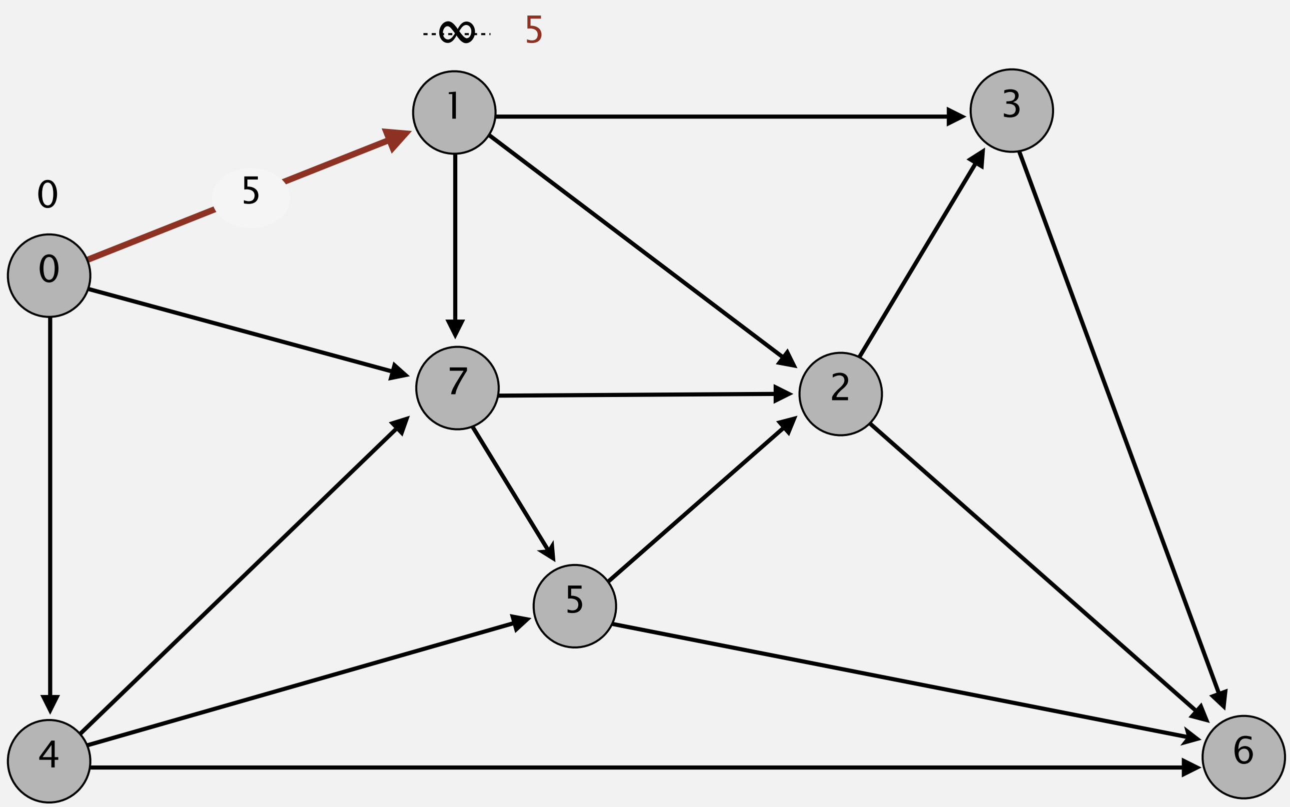
v	distTo[]	edgeTo[]
0	0.0	-
1		
2		
3		
4		
5		
6		
7		

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



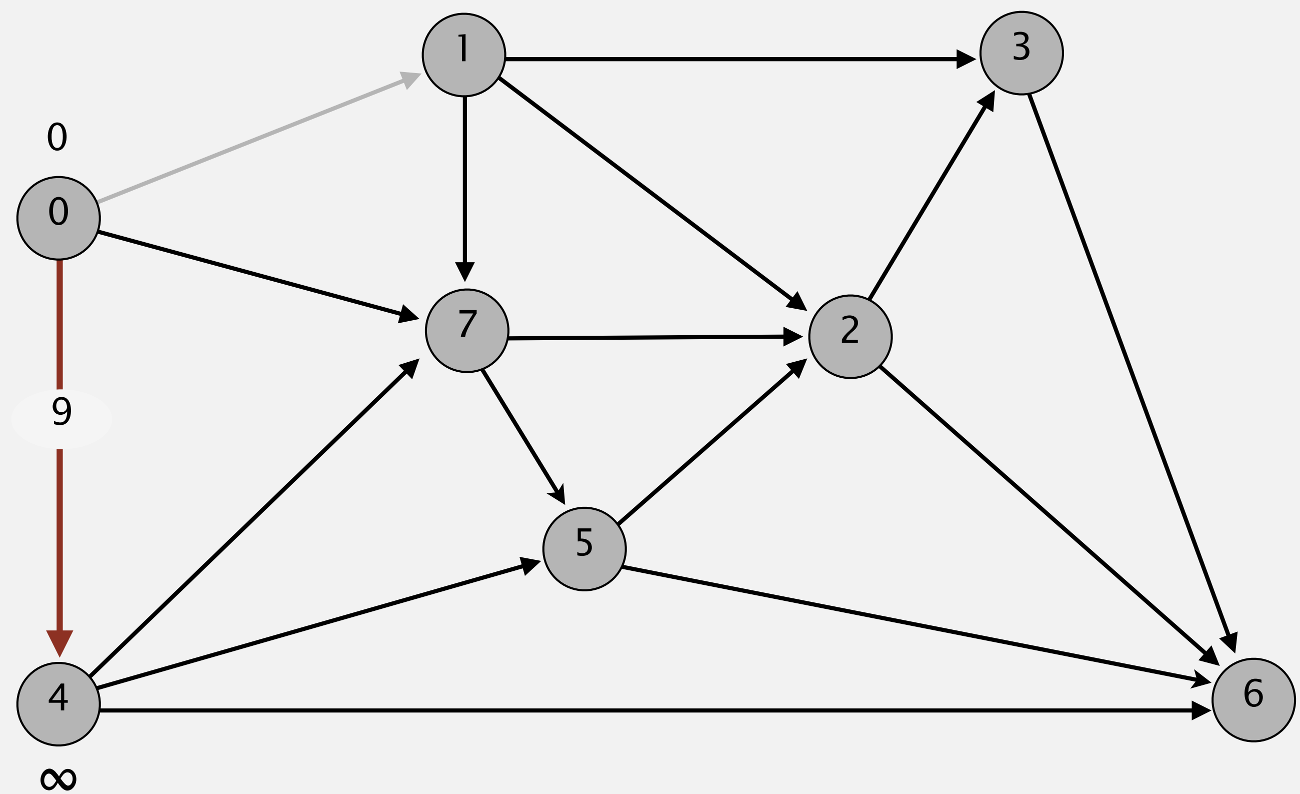
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2		
3		
4		
5		
6		
7		

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



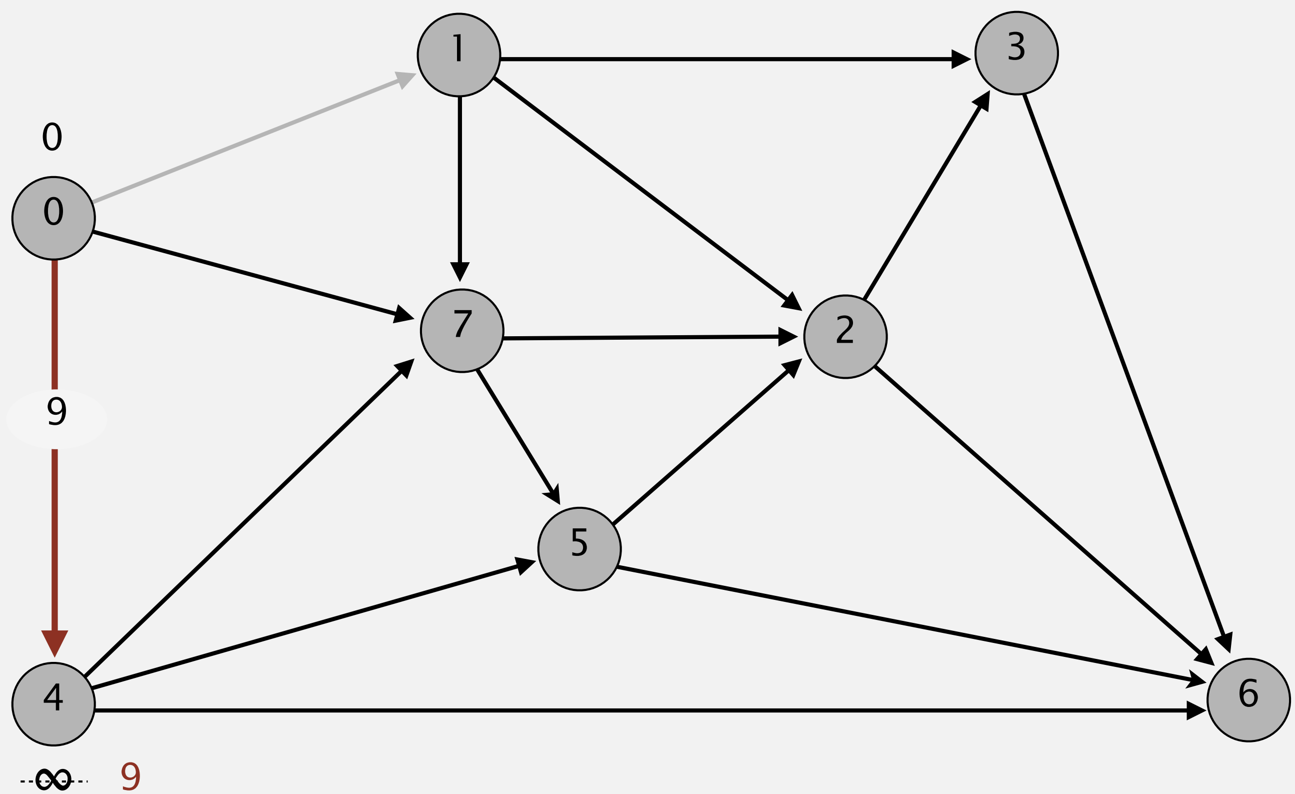
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2		
3		
4		
5		
6		
7		

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



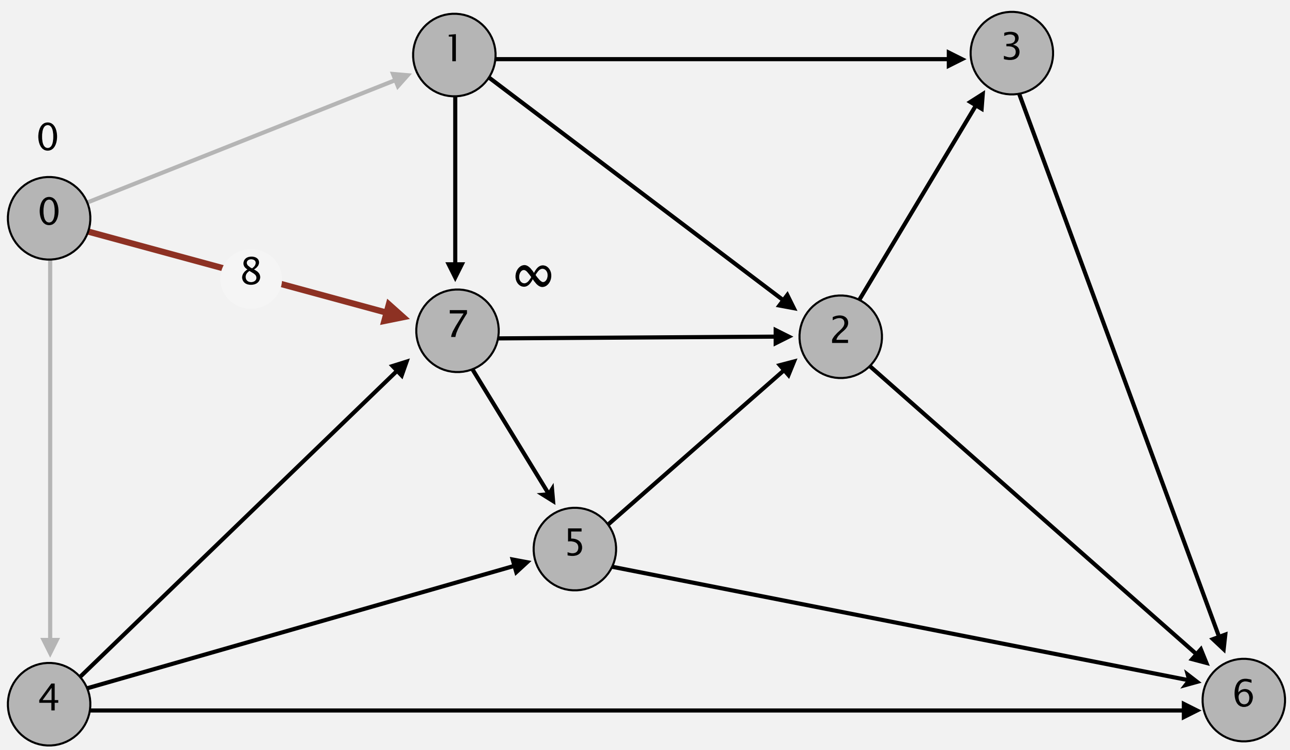
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2		
3		
4	9.0	0→4
5		
6		
7		

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2		
3		
4	9.0	0→4
5		
6		
7		

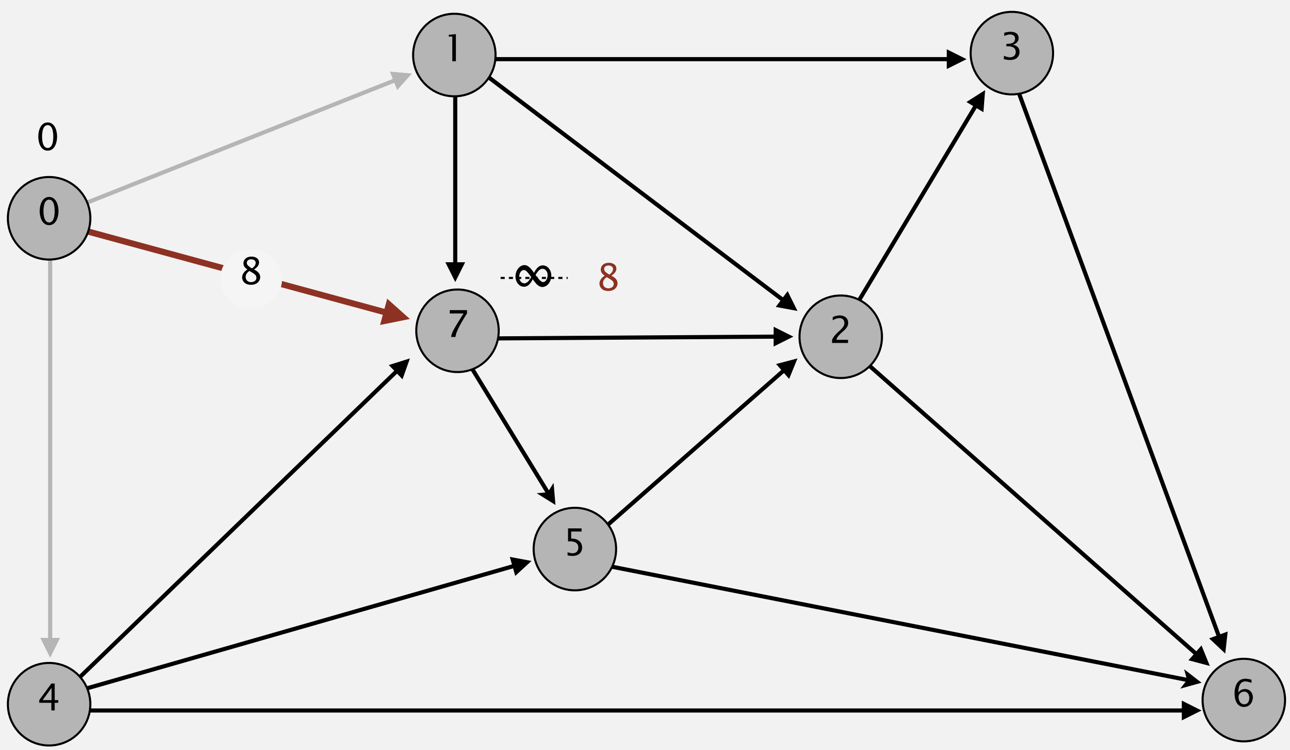
pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

↑

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



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

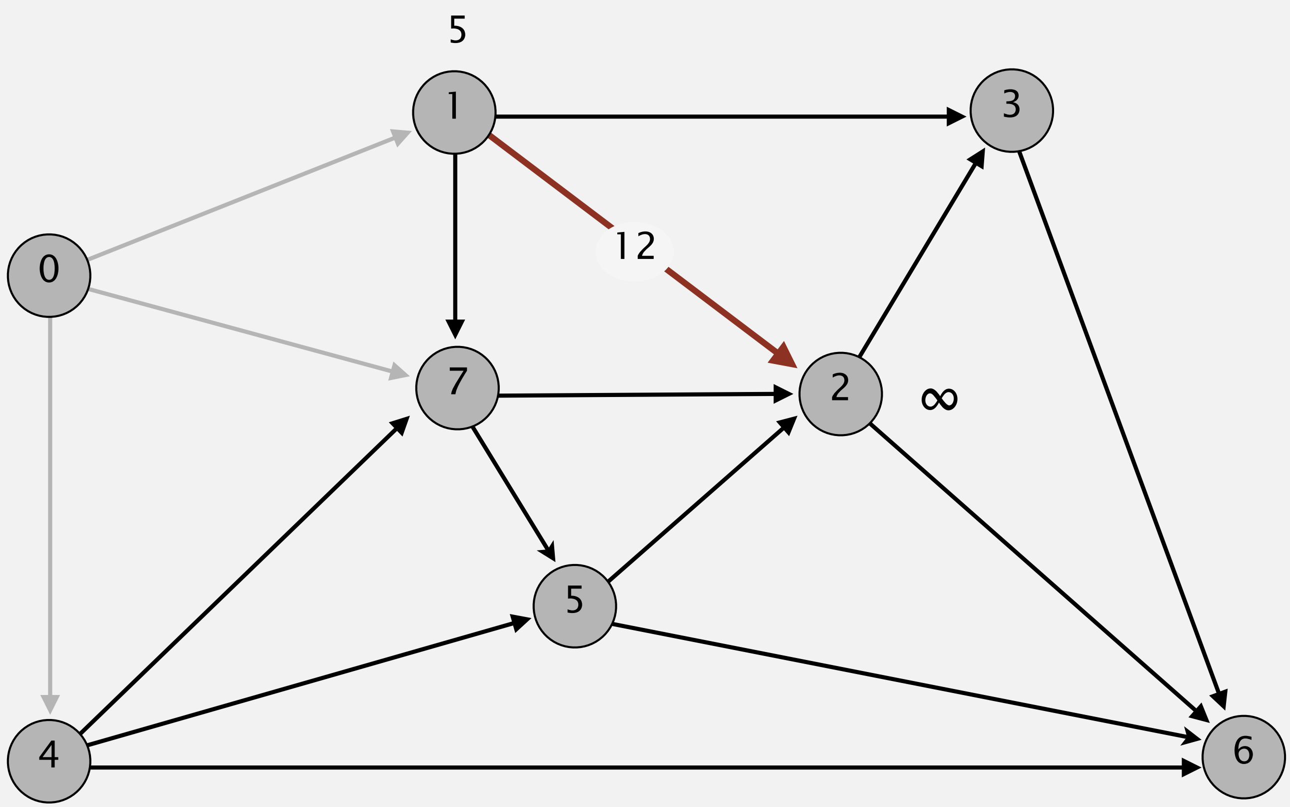
pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

↑

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



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

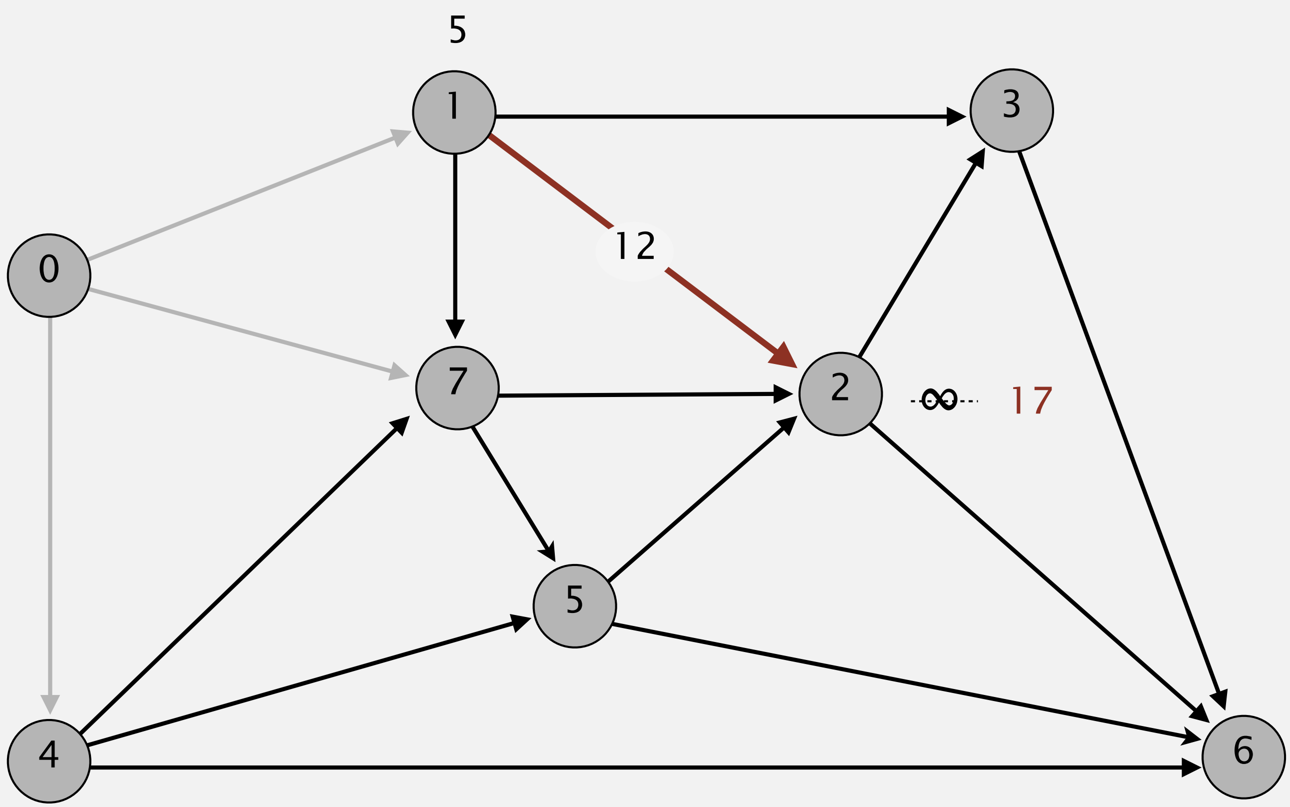
pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

↑

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



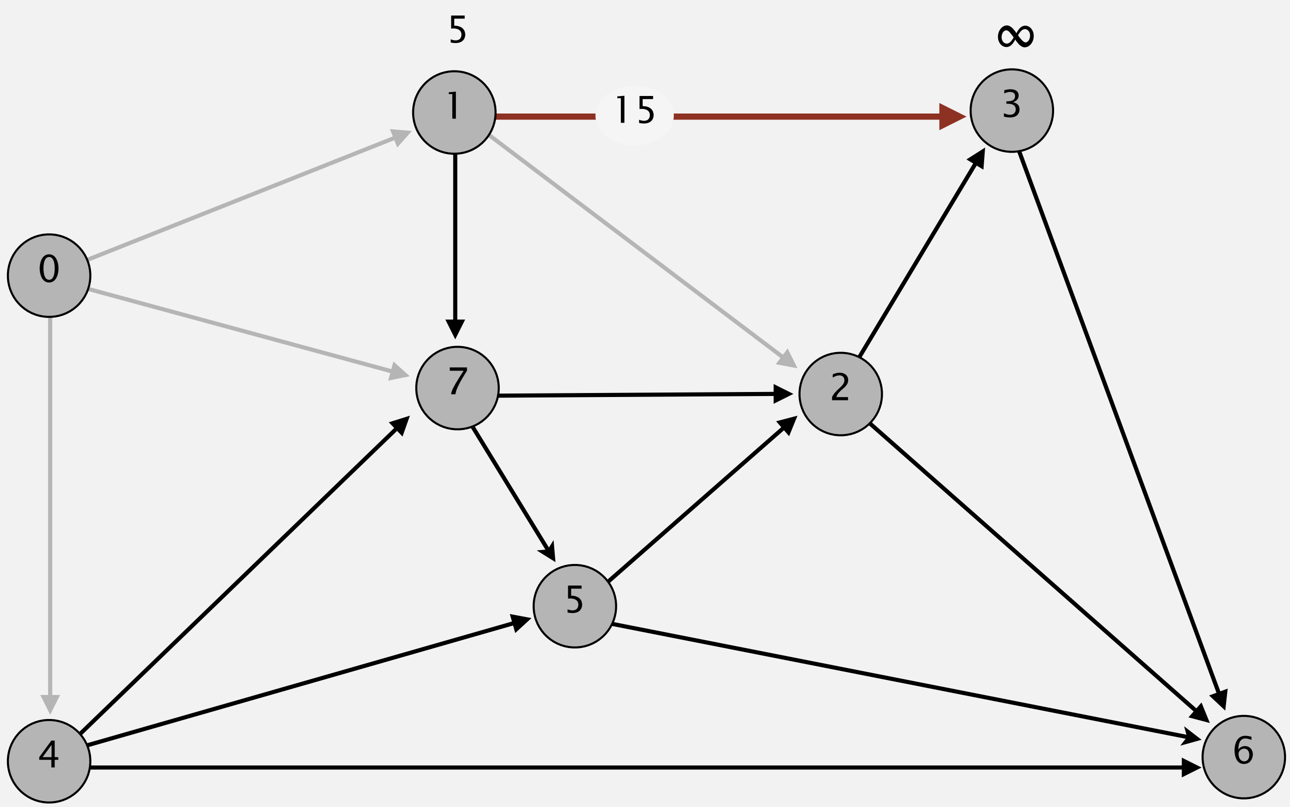
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3		
4	9.0	0→4
5		
6		
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



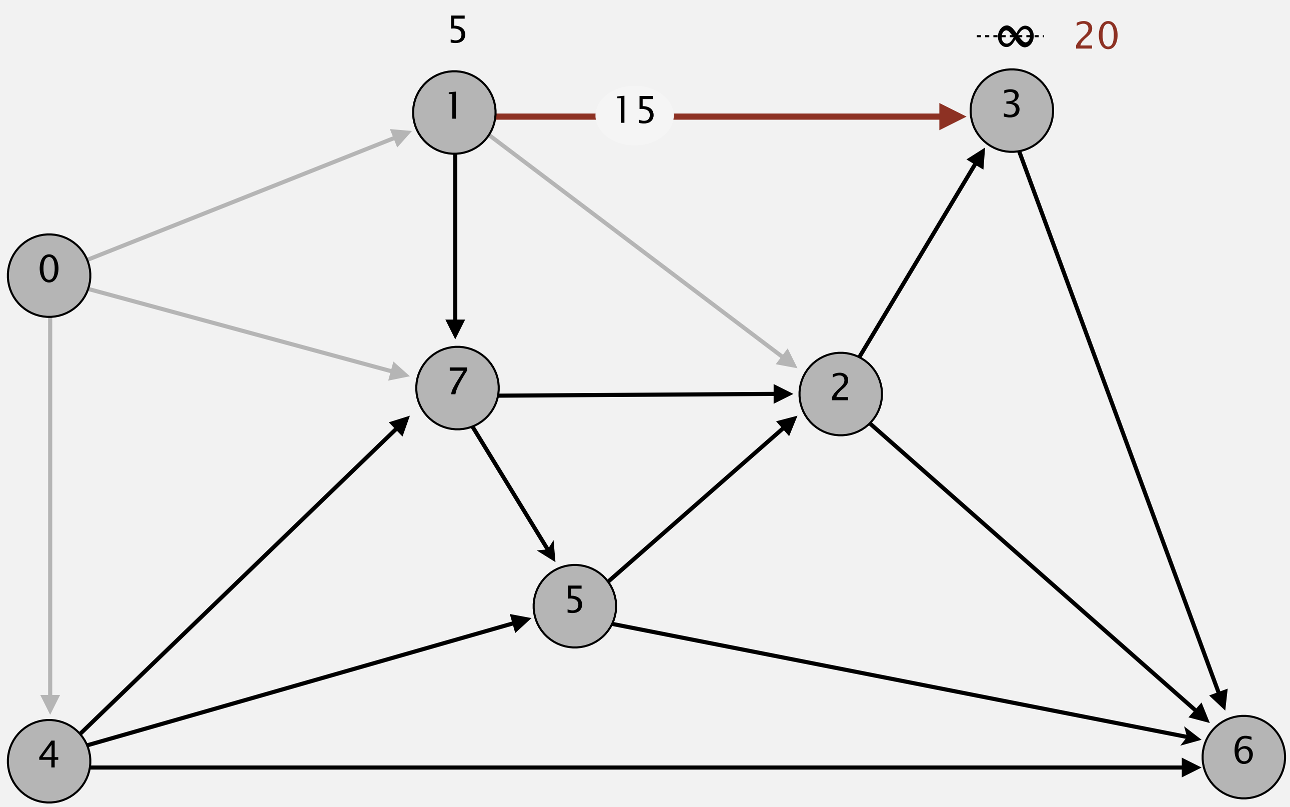
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3		
4	9.0	0→4
5		
6		
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



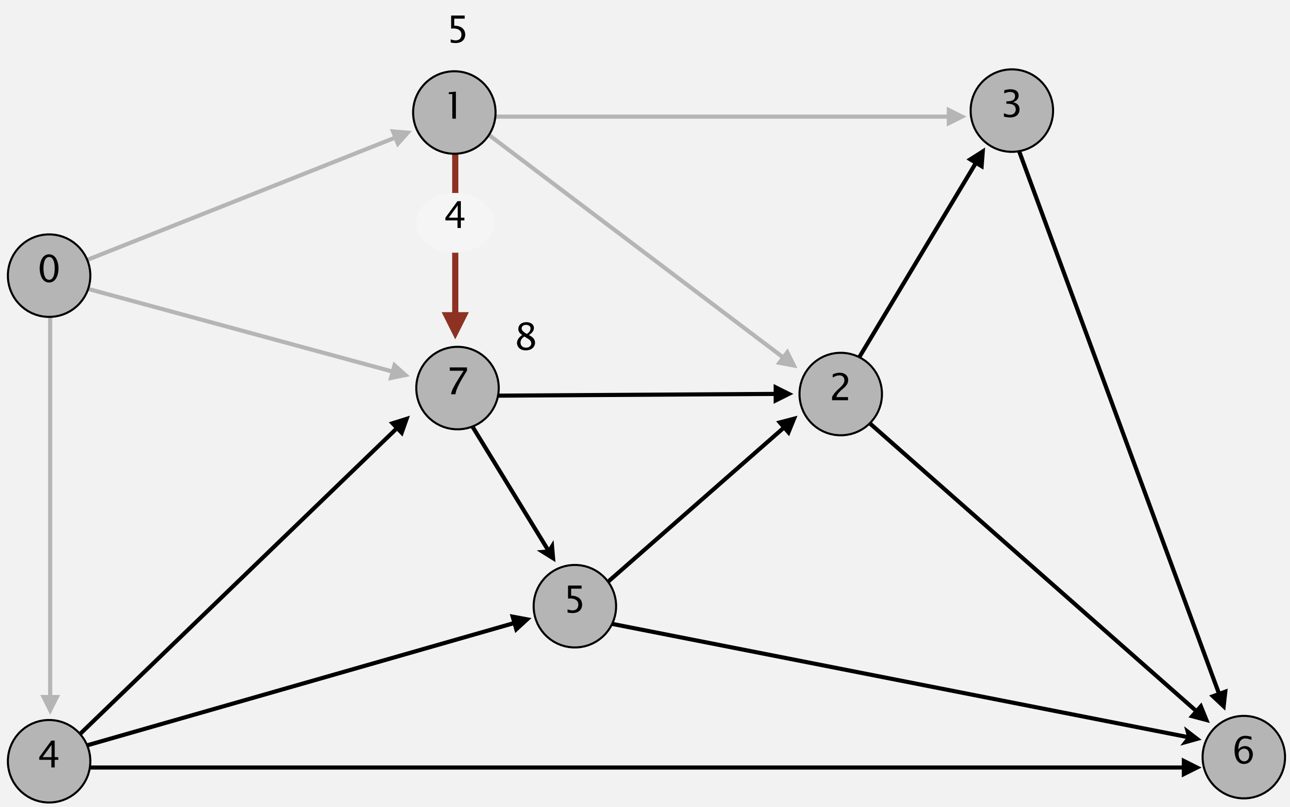
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5		
6		
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



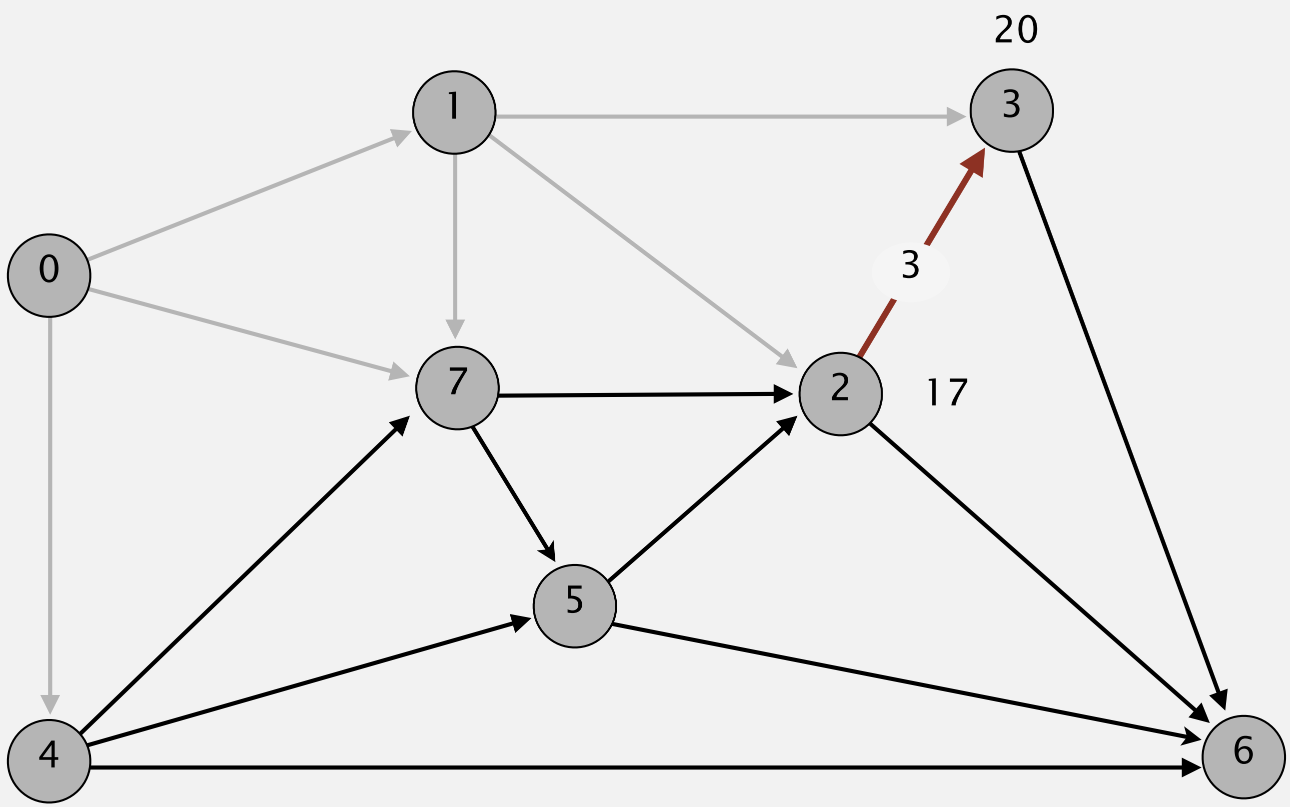
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5		
6		
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



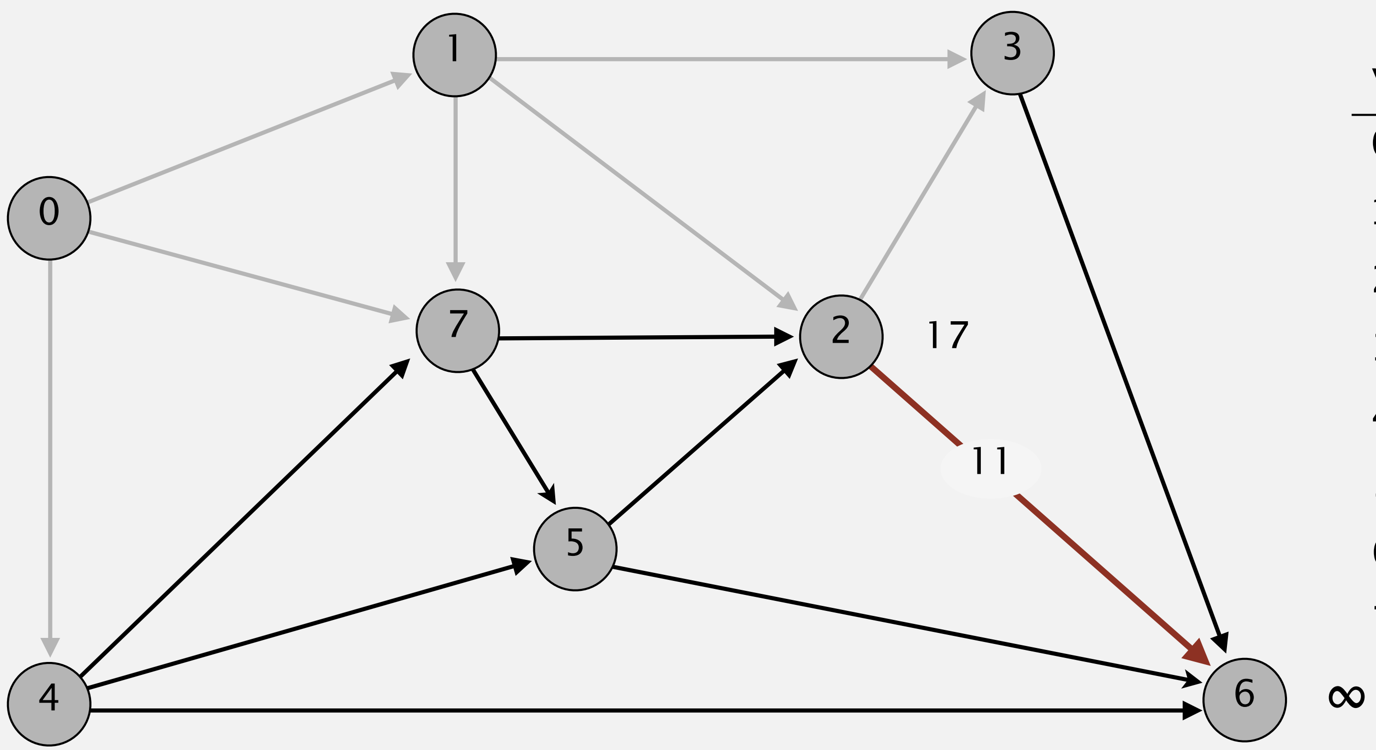
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5		
6		
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



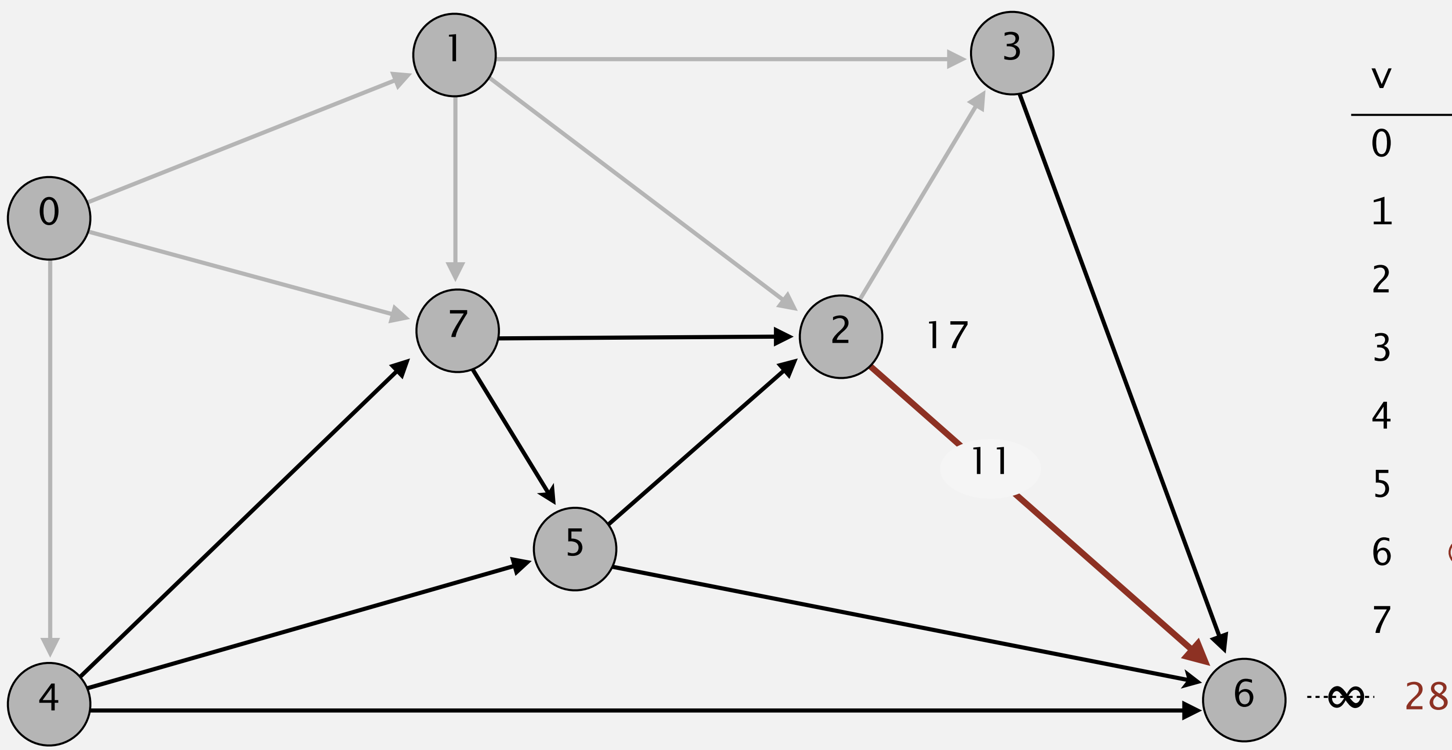
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5		
6		
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



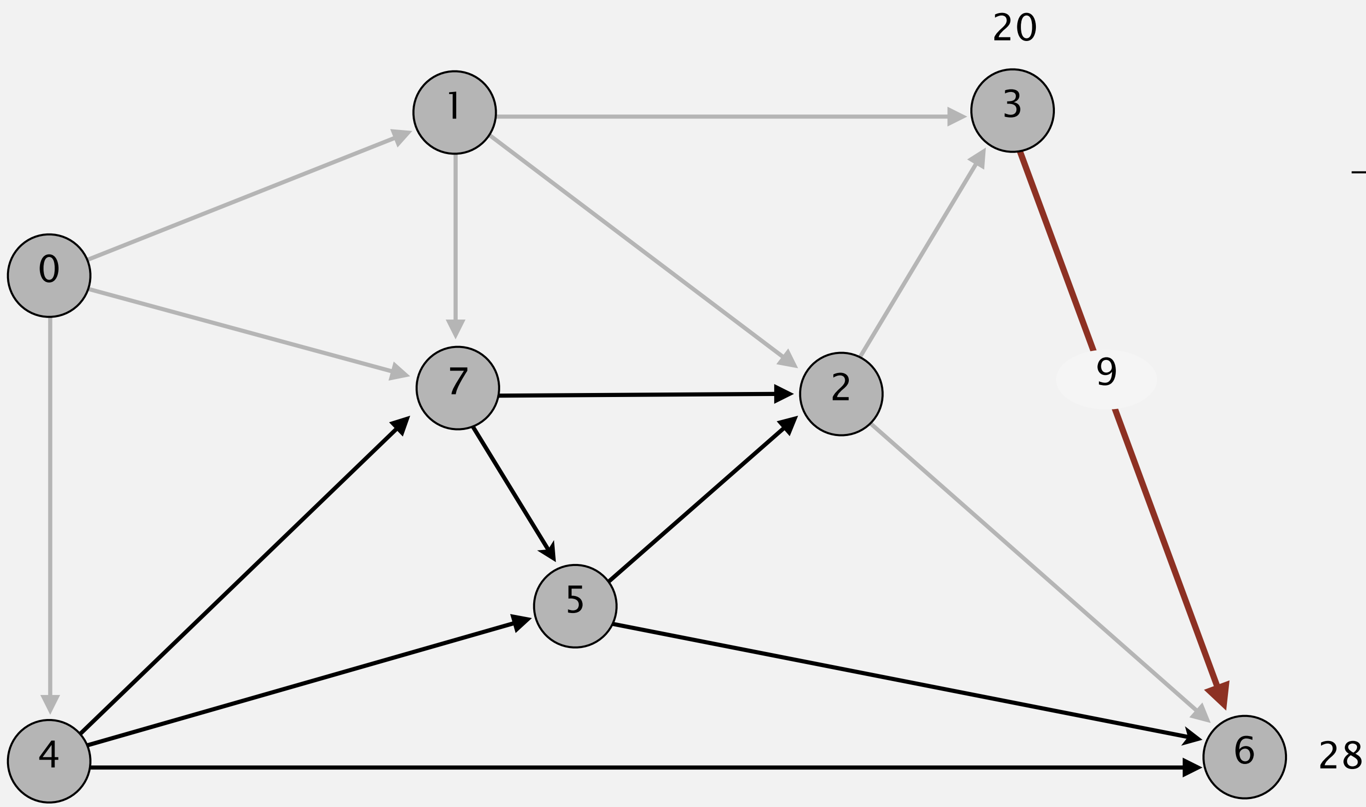
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5		
6	28.0	2→6
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5		
6	28.0	2→6
7	8.0	0→7

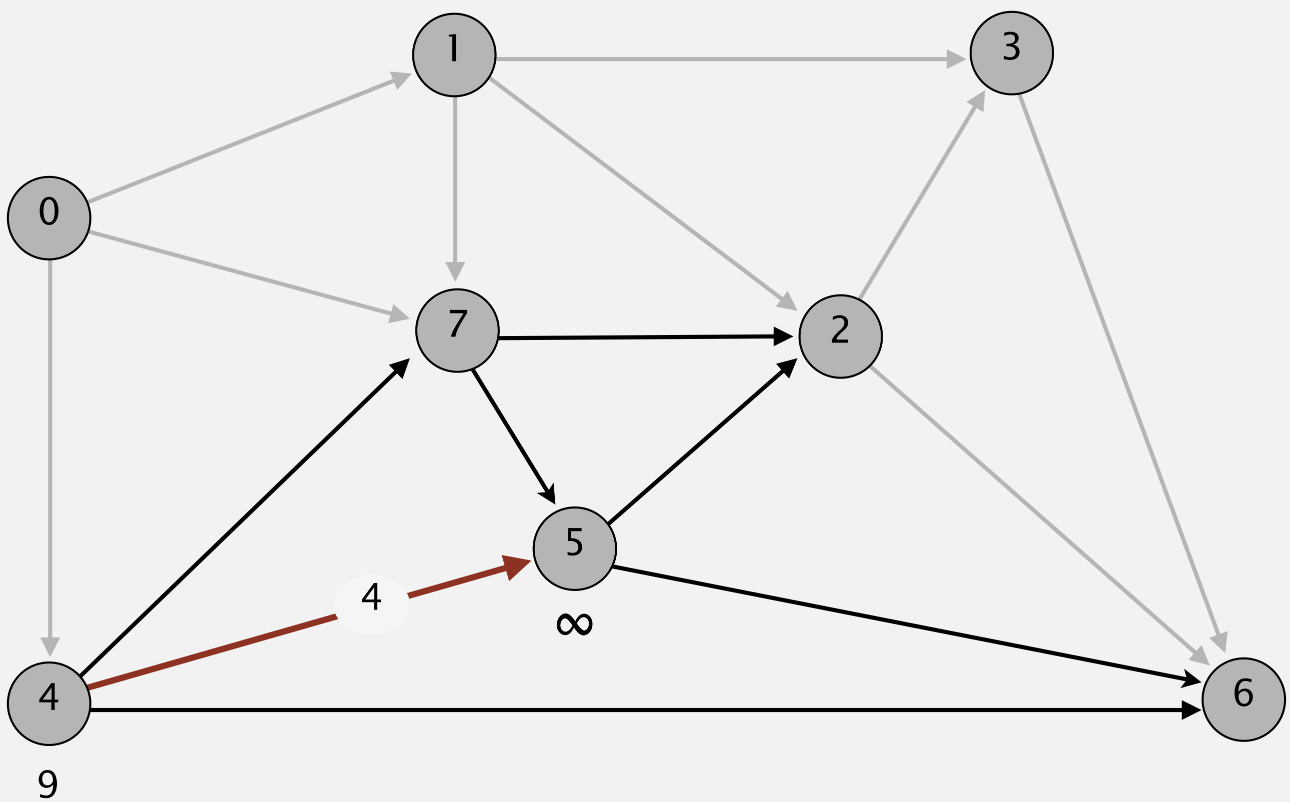
pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

↑

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



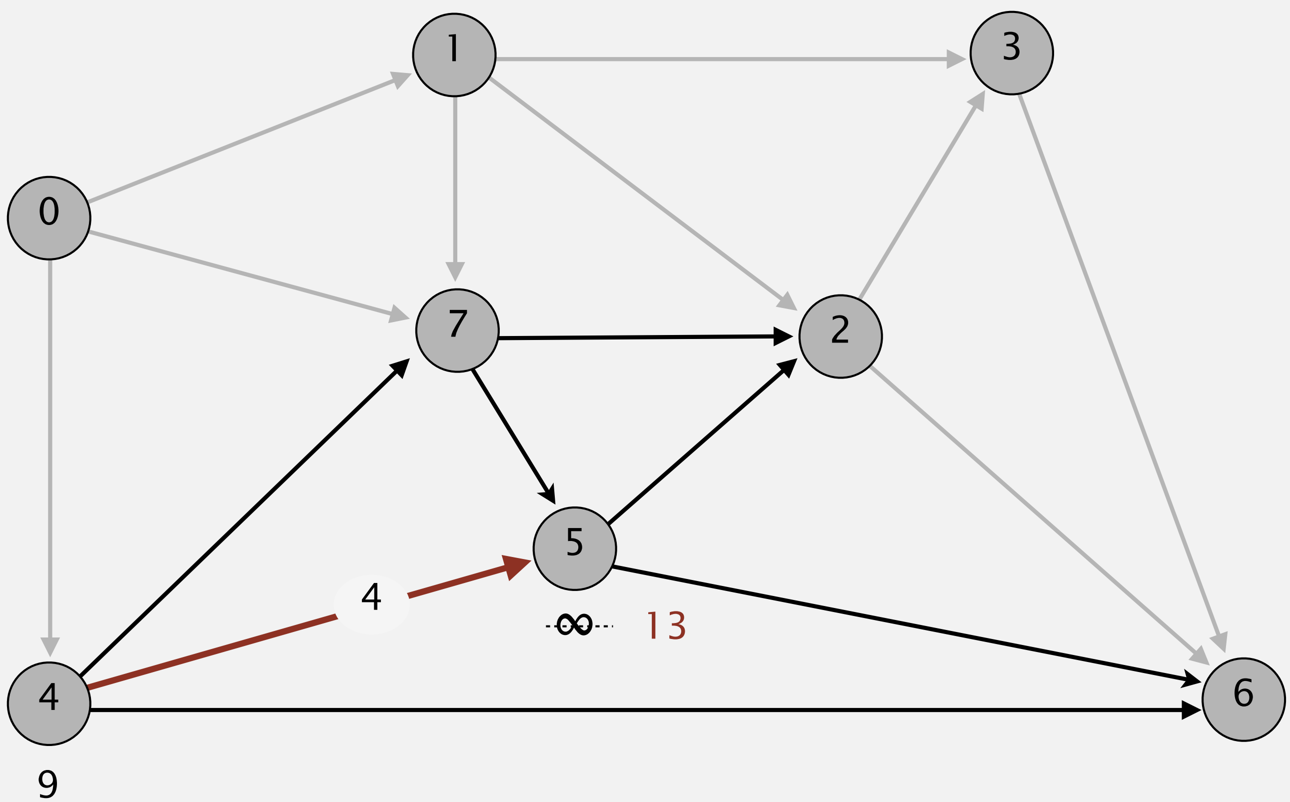
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5		
6	28.0	2→6
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



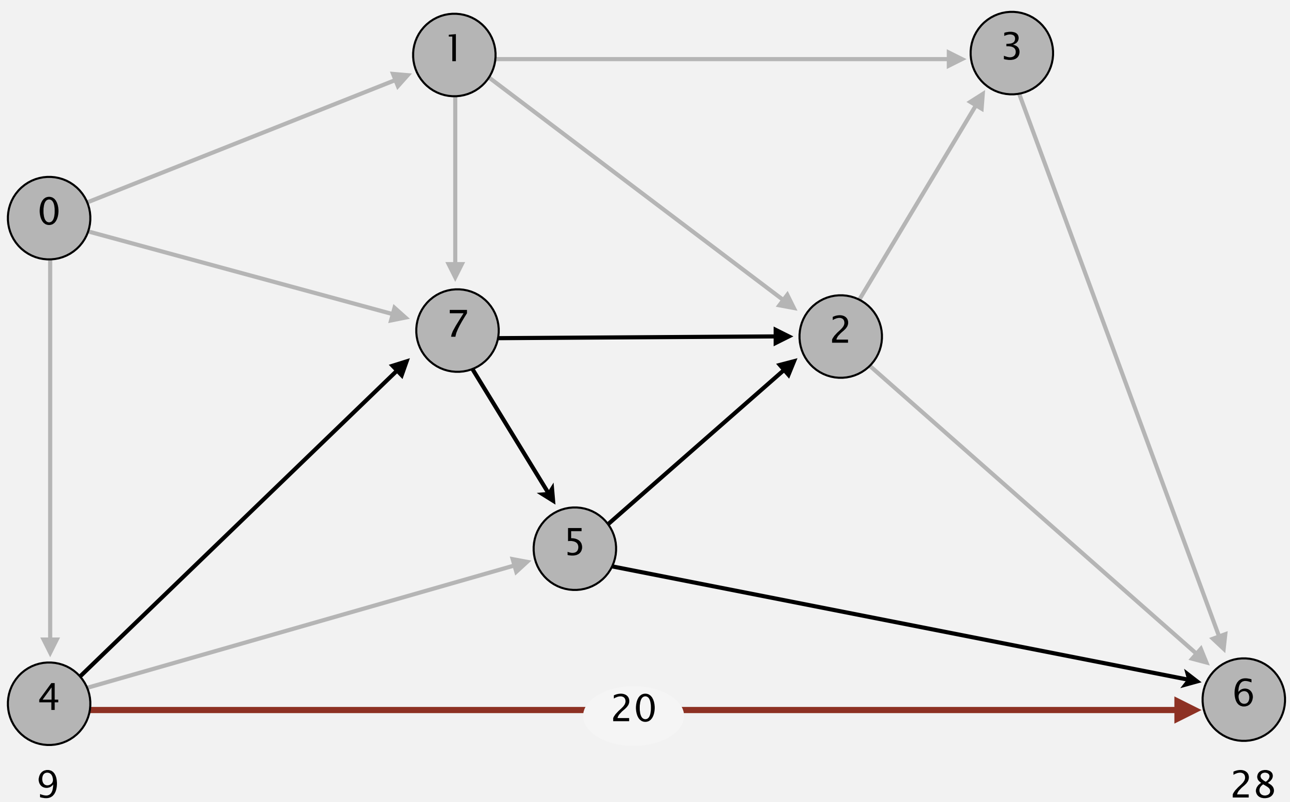
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	28.0	2→6
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	28.0	2→6
7	8.0	0→7

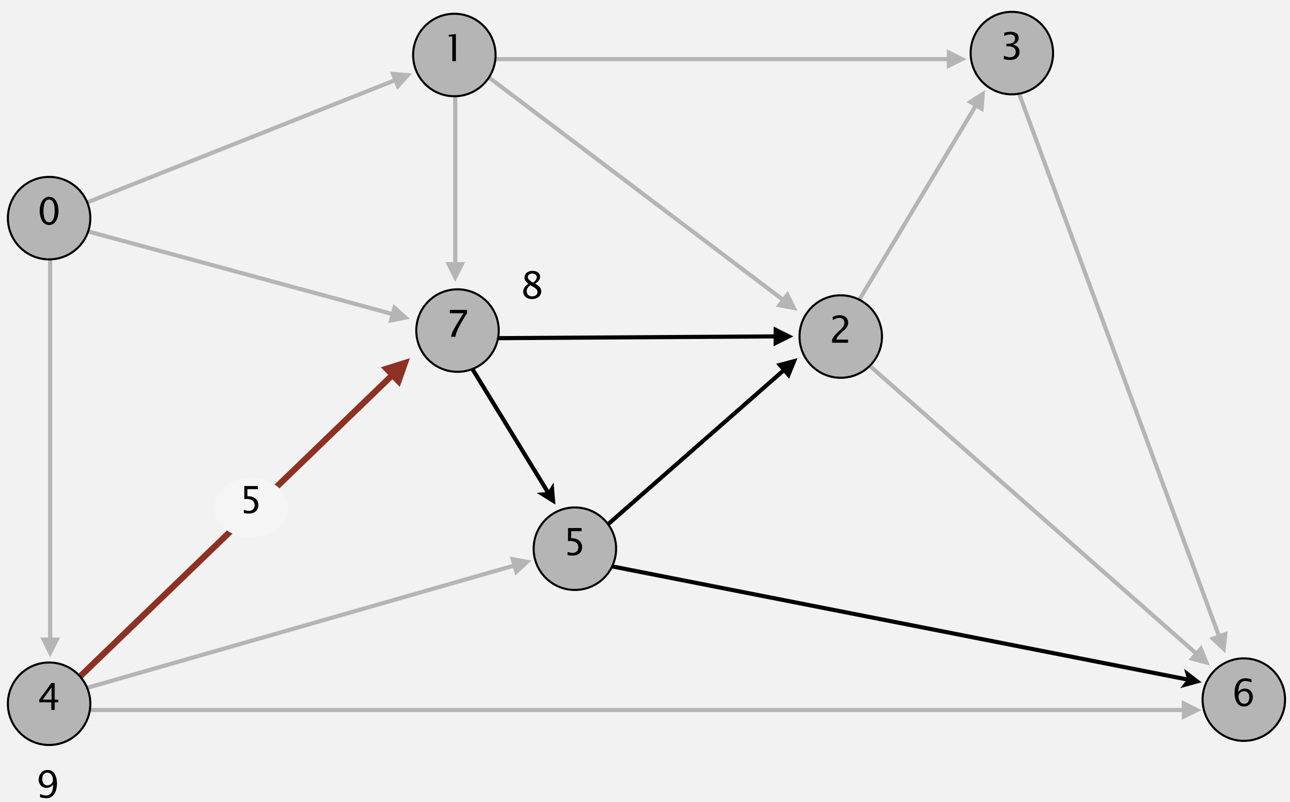
pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

↑

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



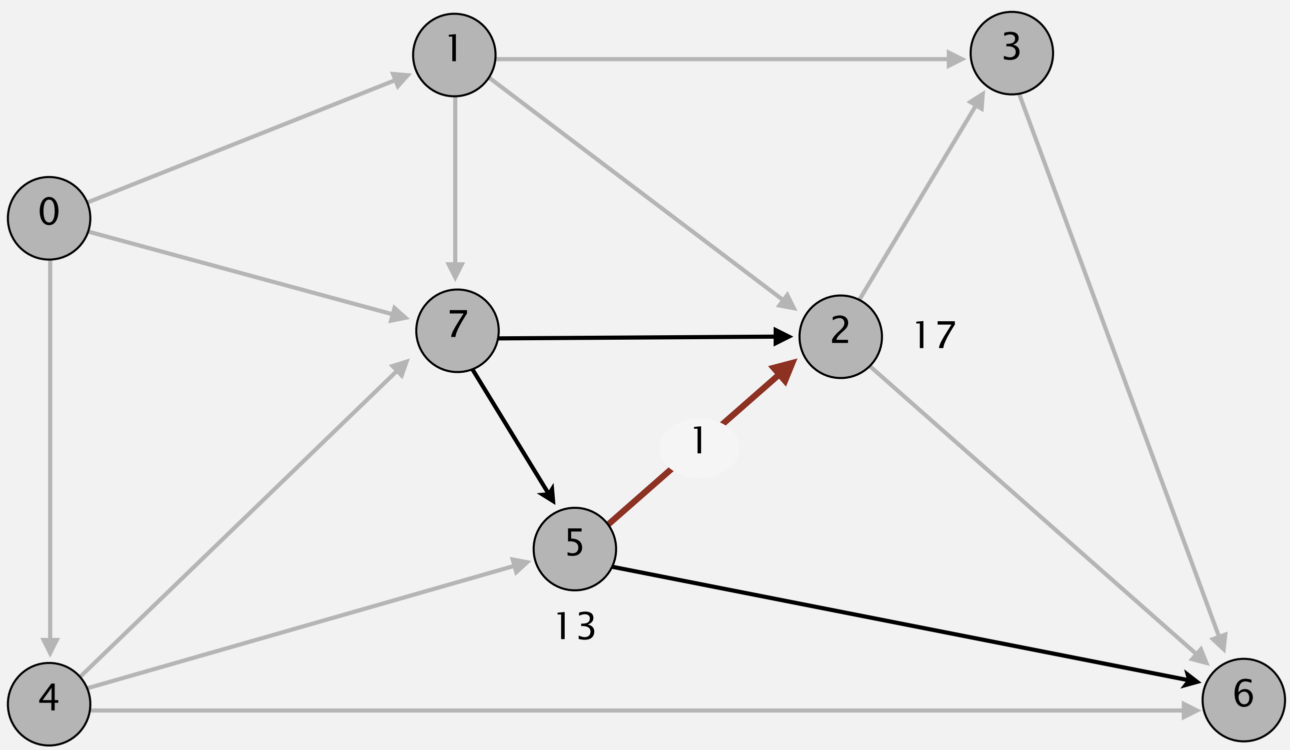
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	28.0	2→6
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



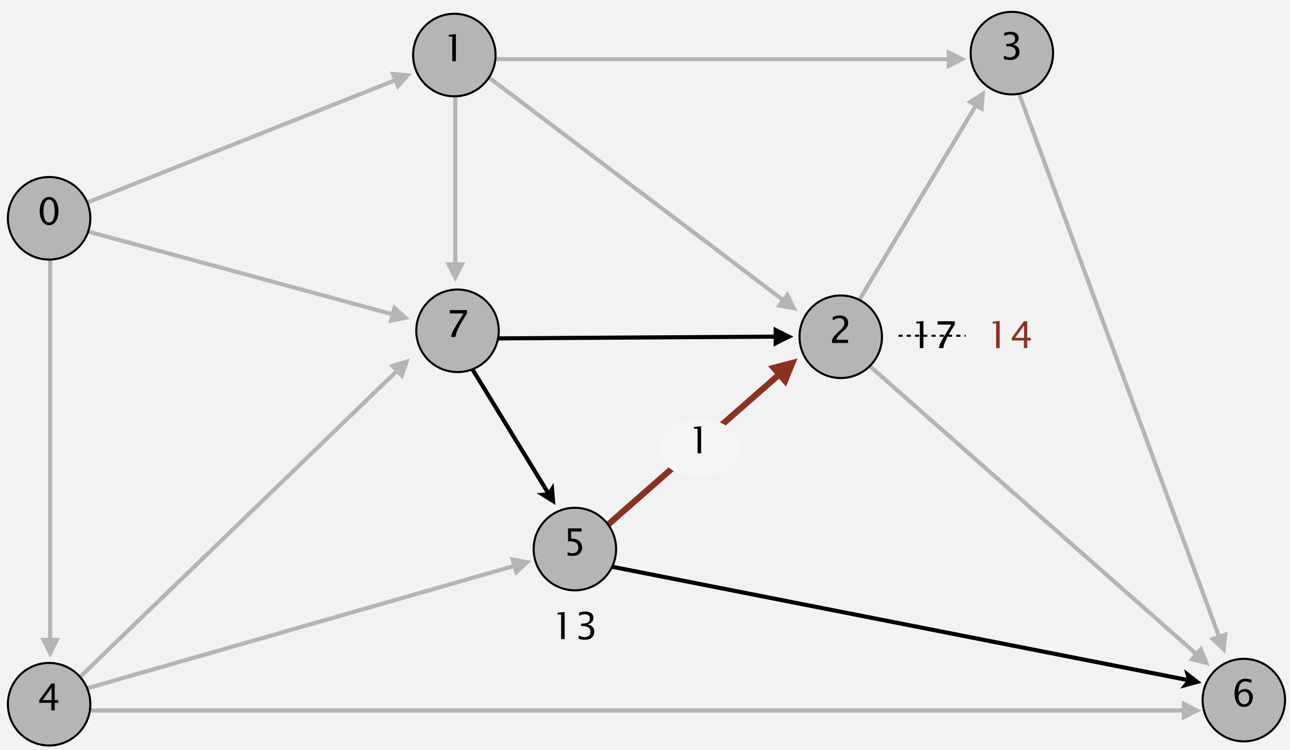
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	17.0	1→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	28.0	2→6
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



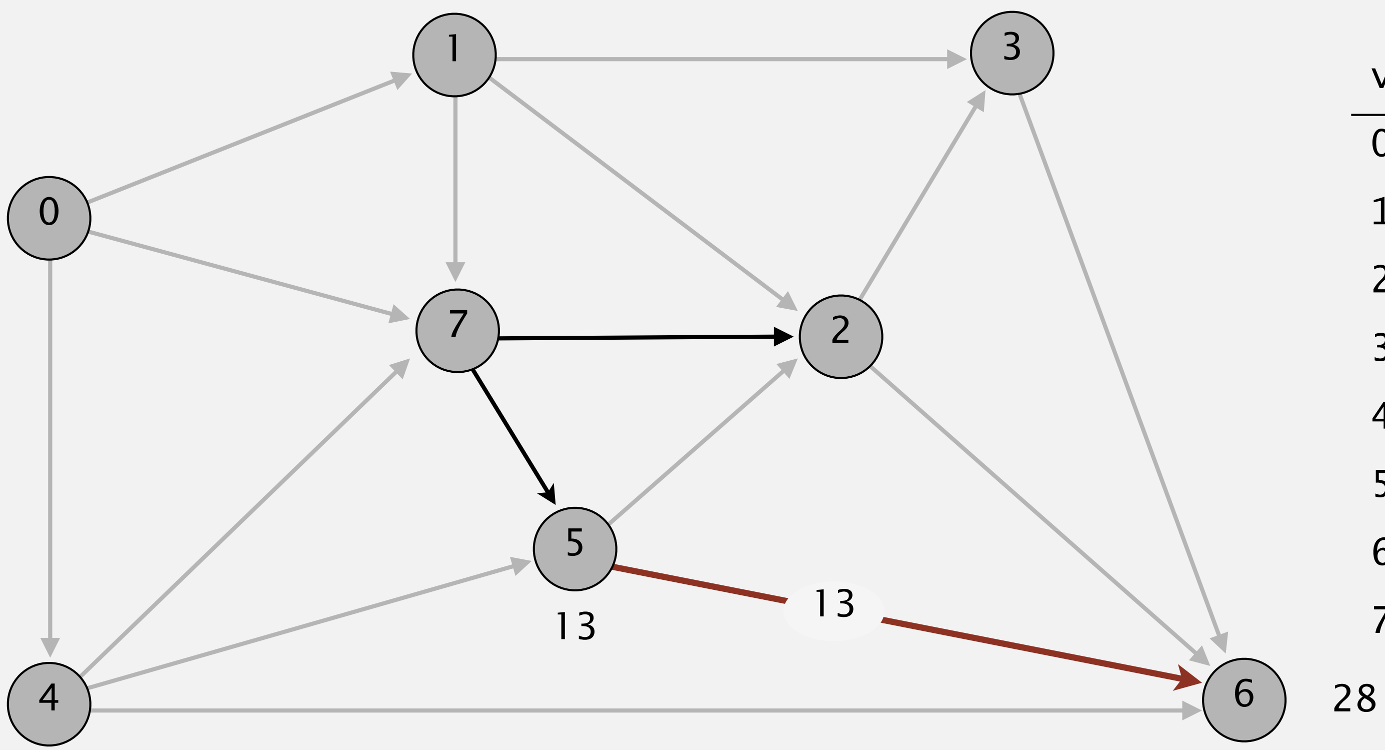
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	28.0	2→6
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



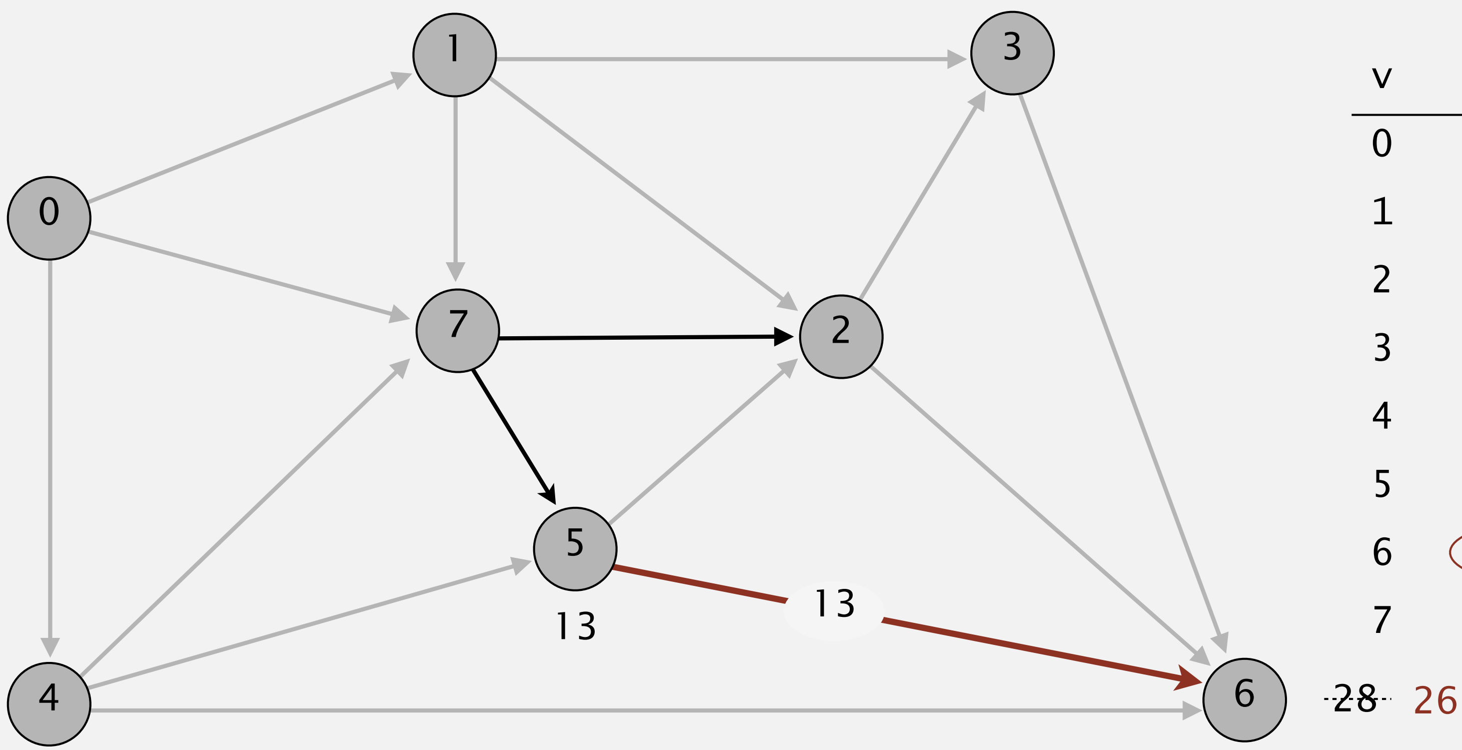
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	28.0	2→6
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



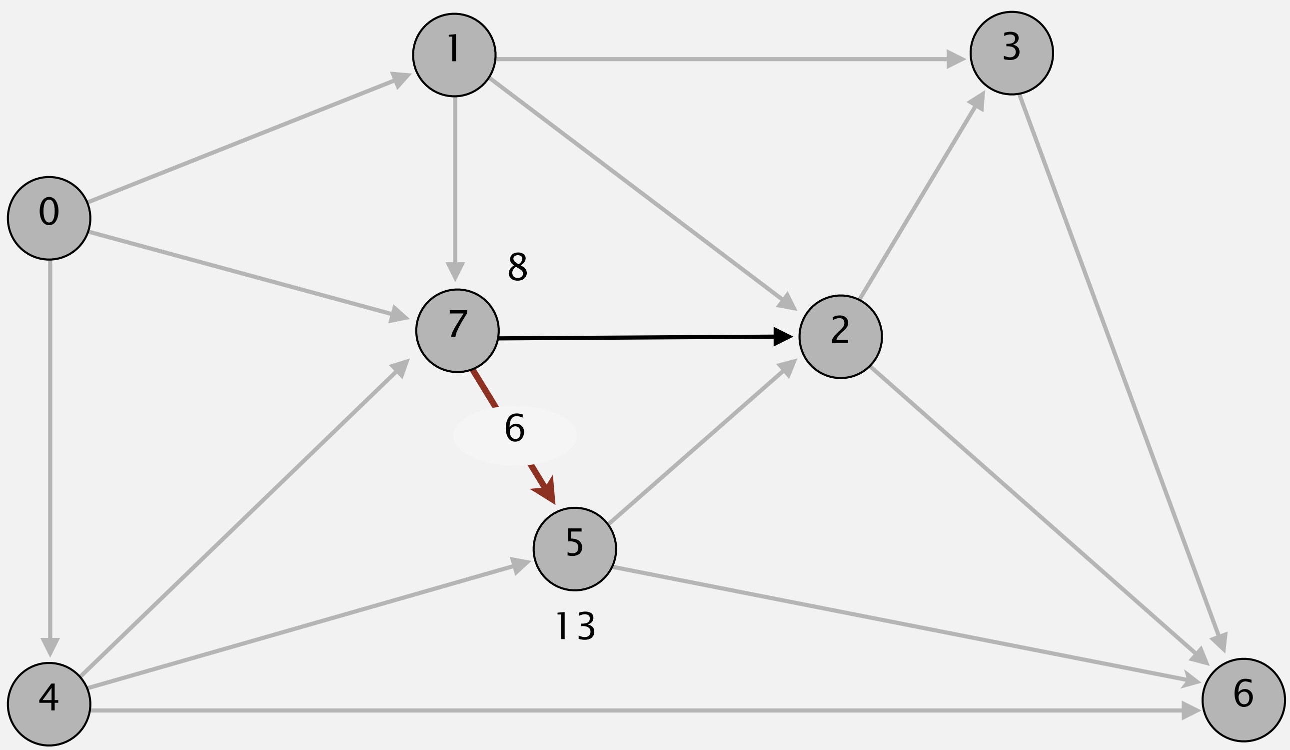
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

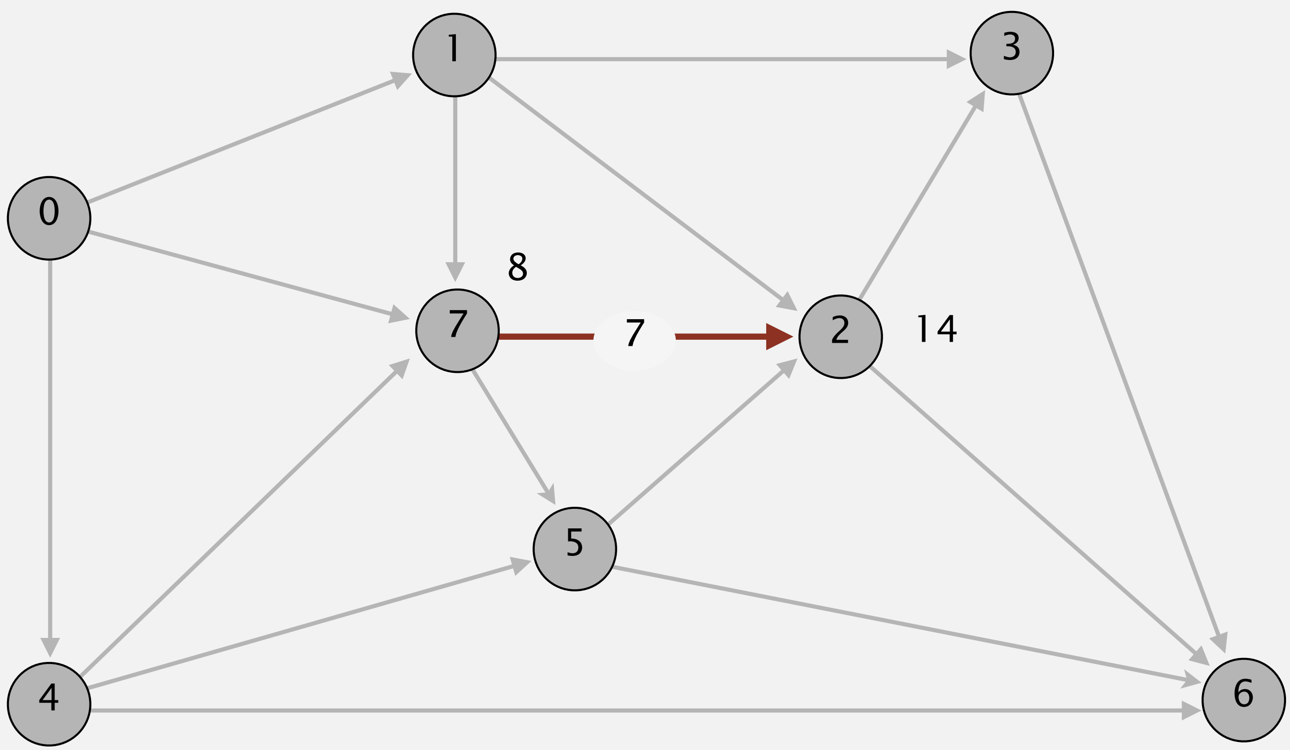
pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

↑

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



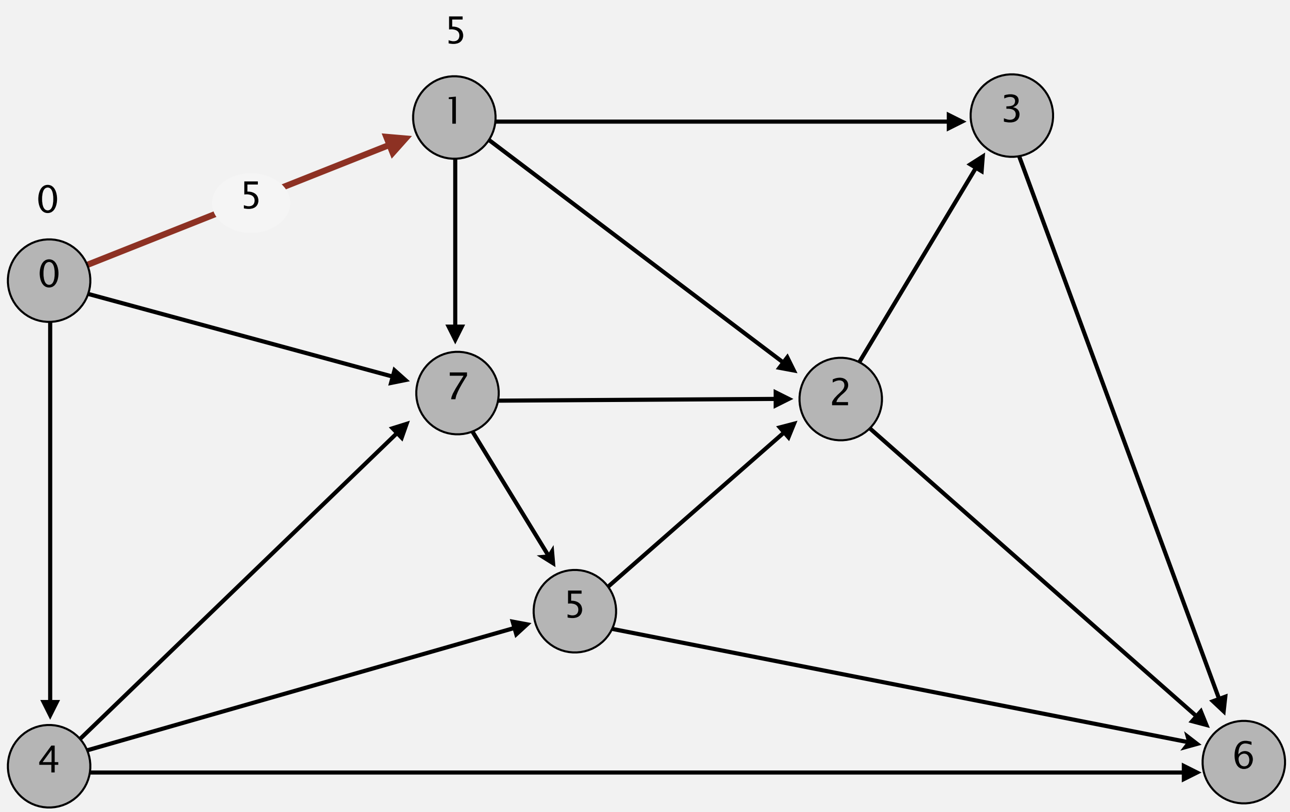
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

pass 0

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



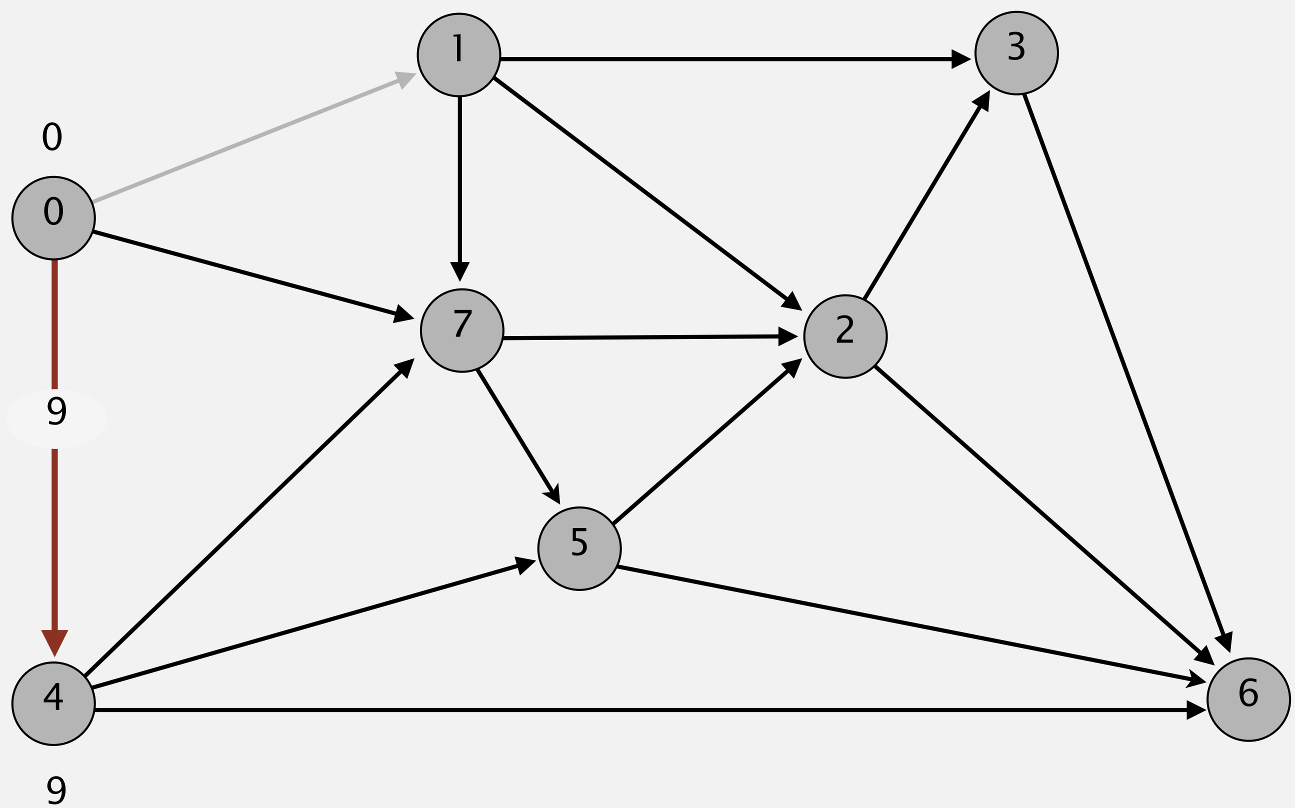
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



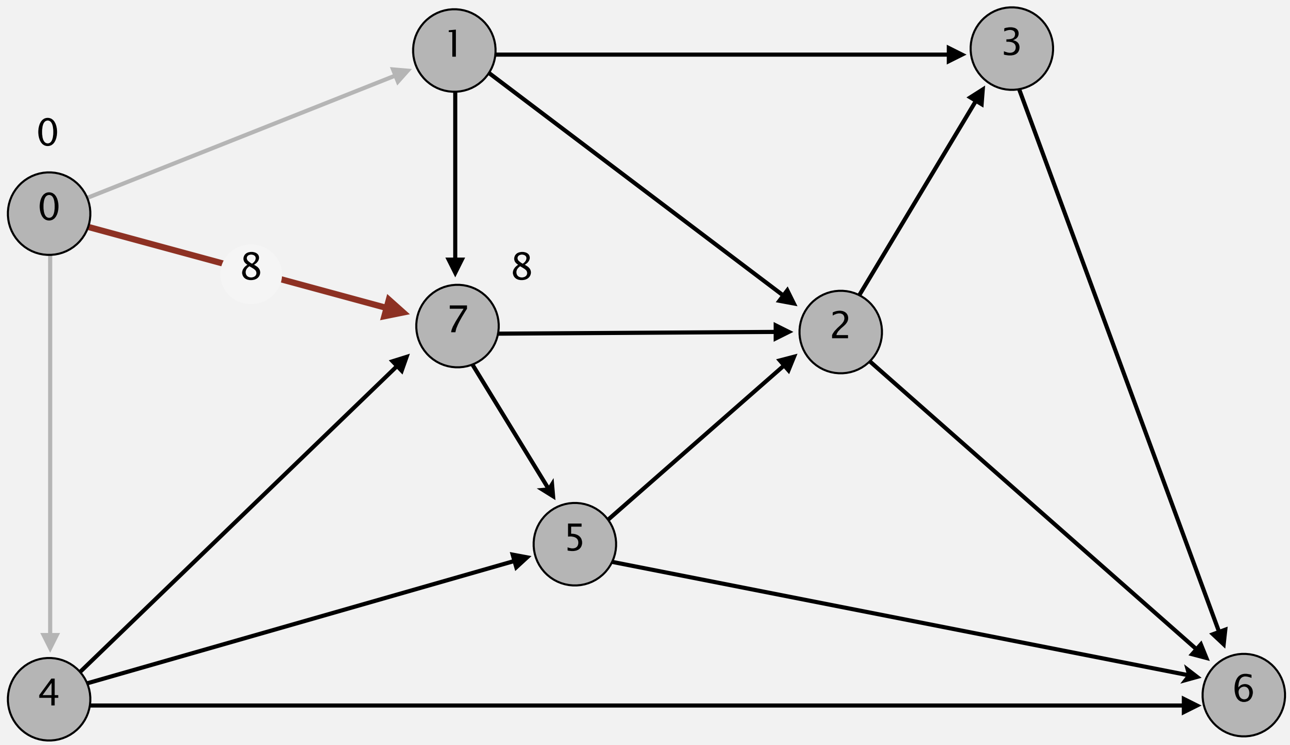
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



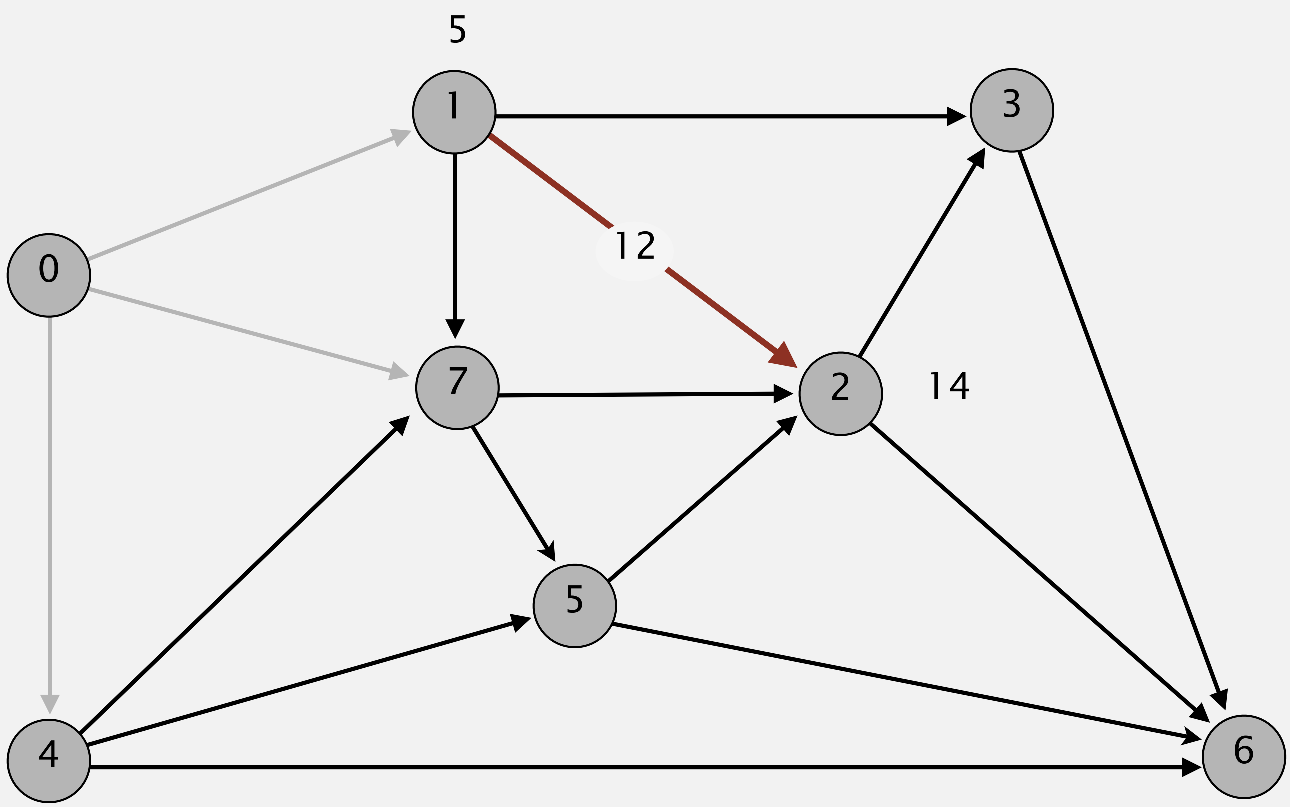
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



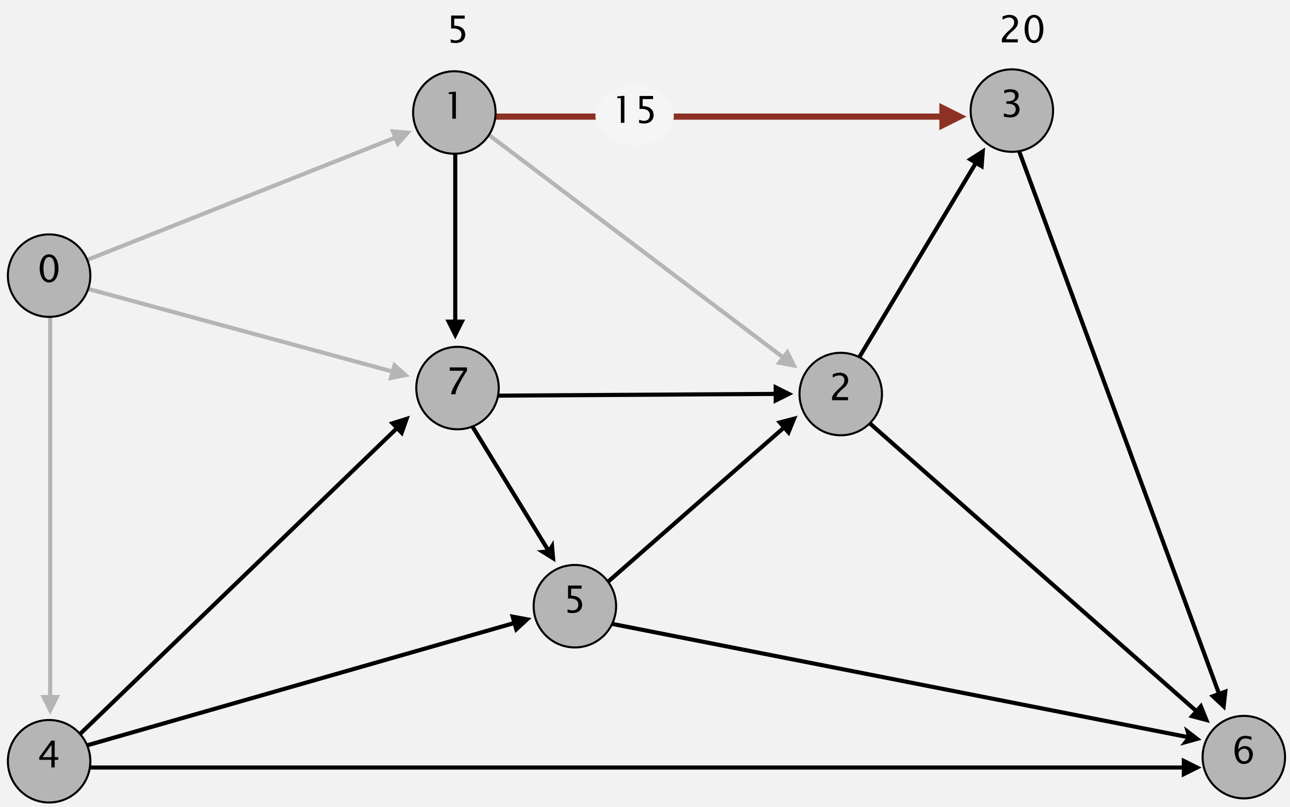
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



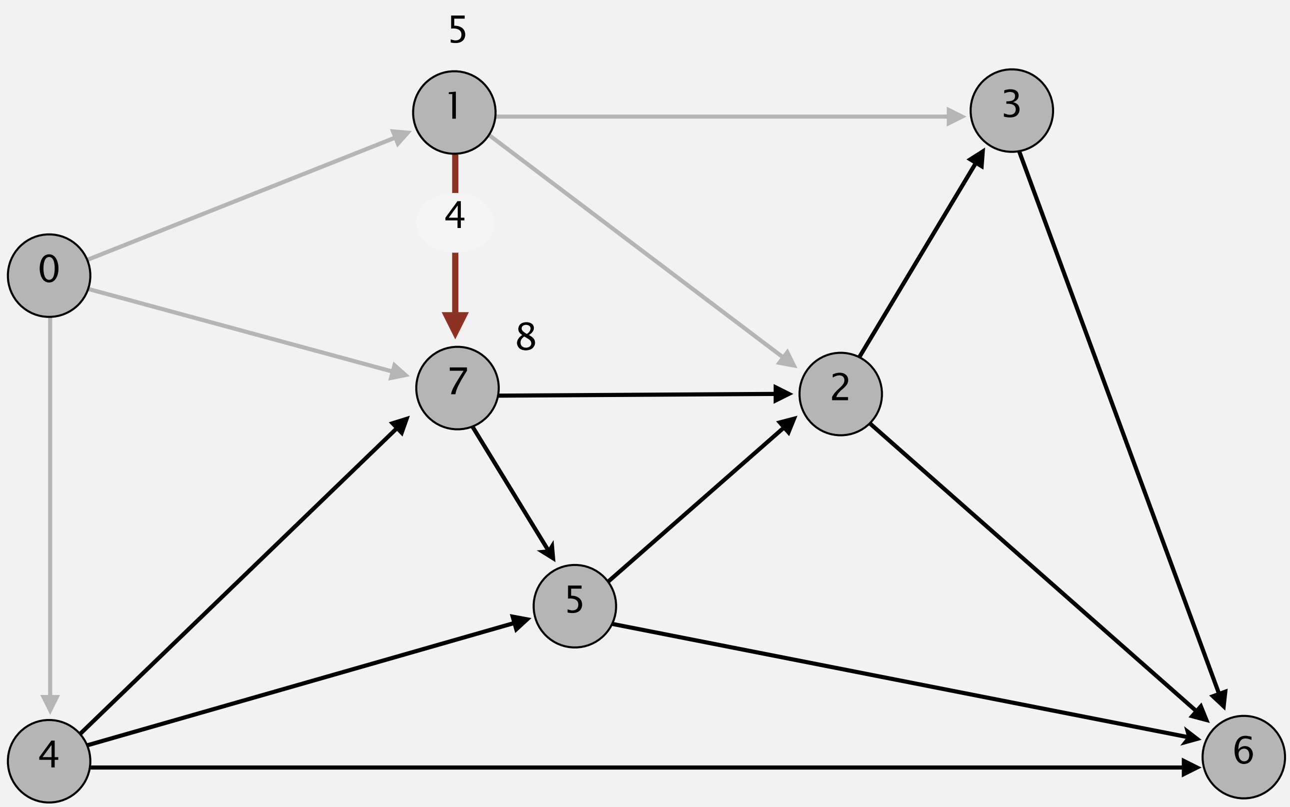
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



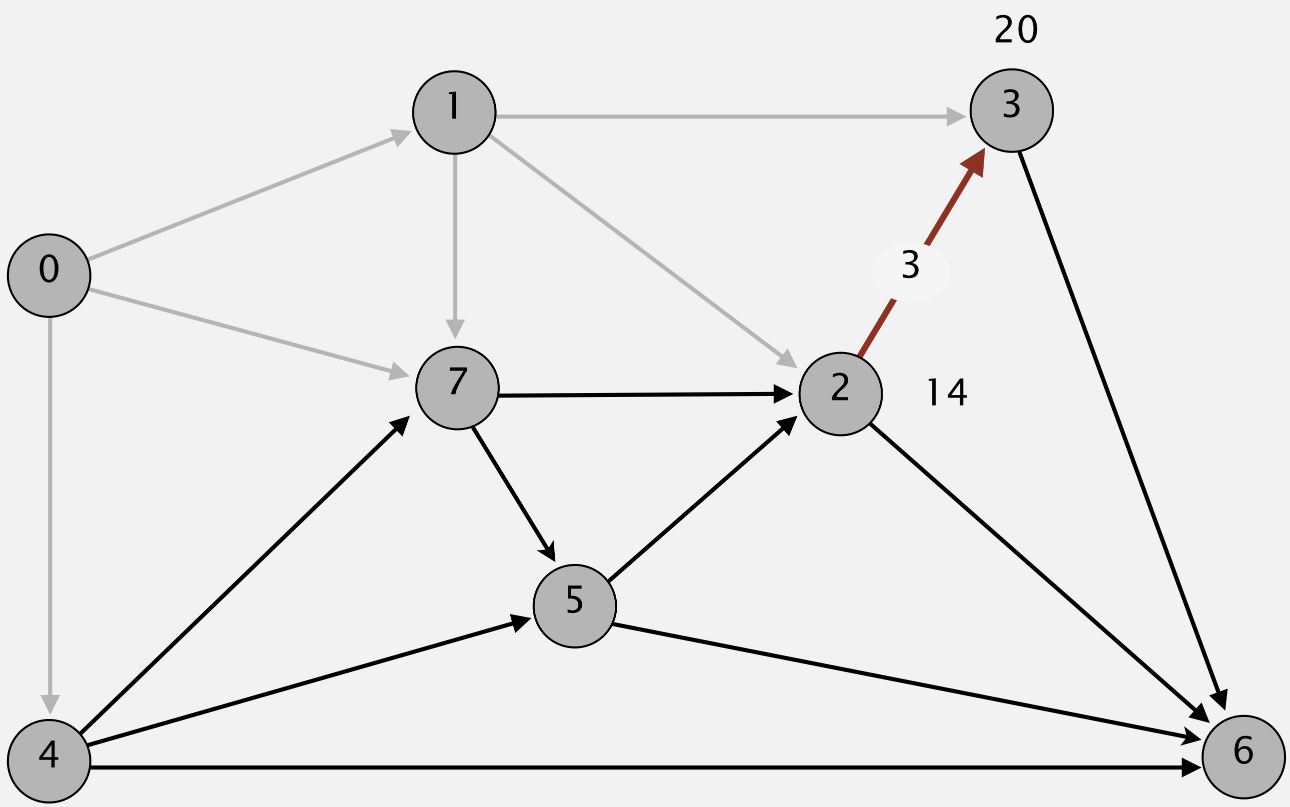
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



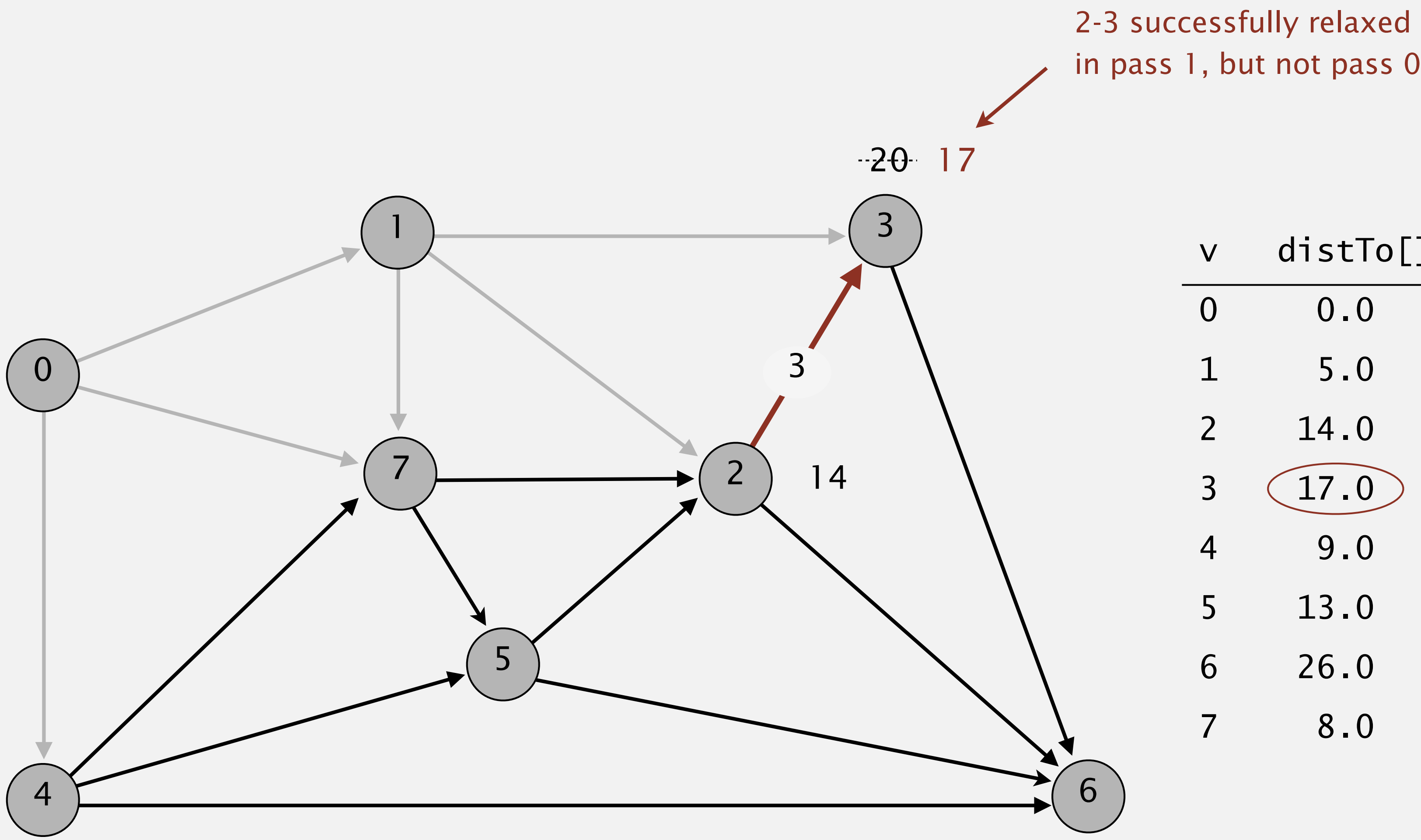
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	20.0	1→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



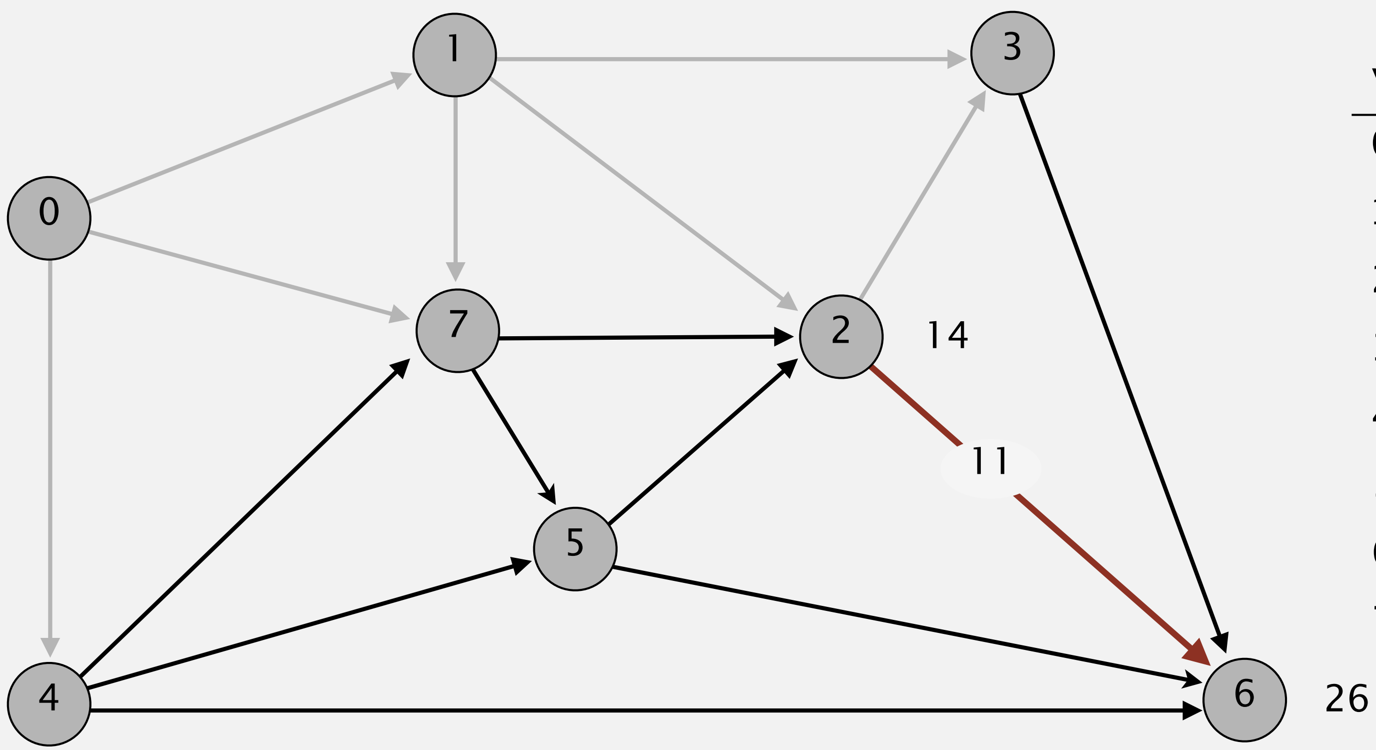
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	26.0	5→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



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	26.0	5→6
7	8.0	0→7

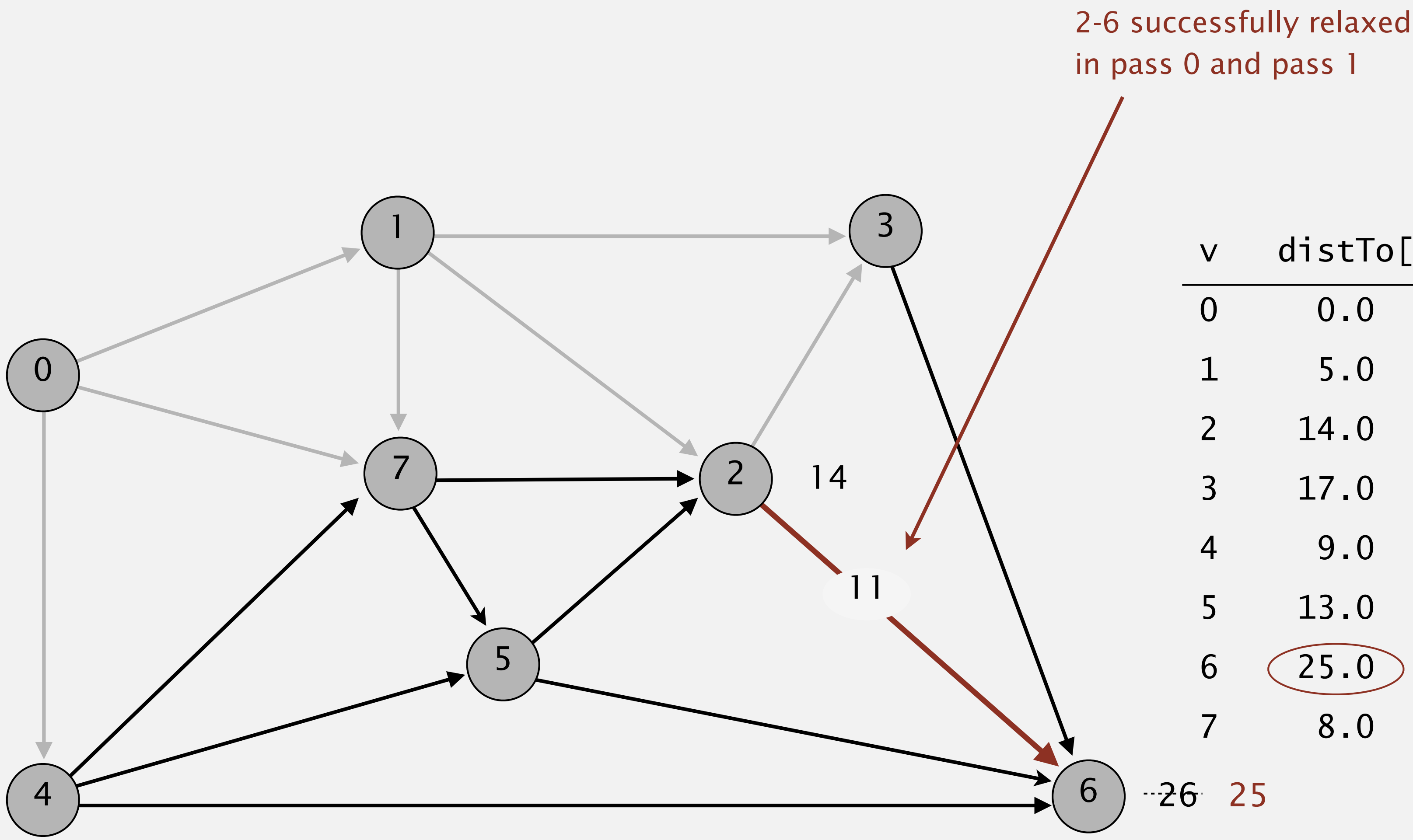
pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

↑

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



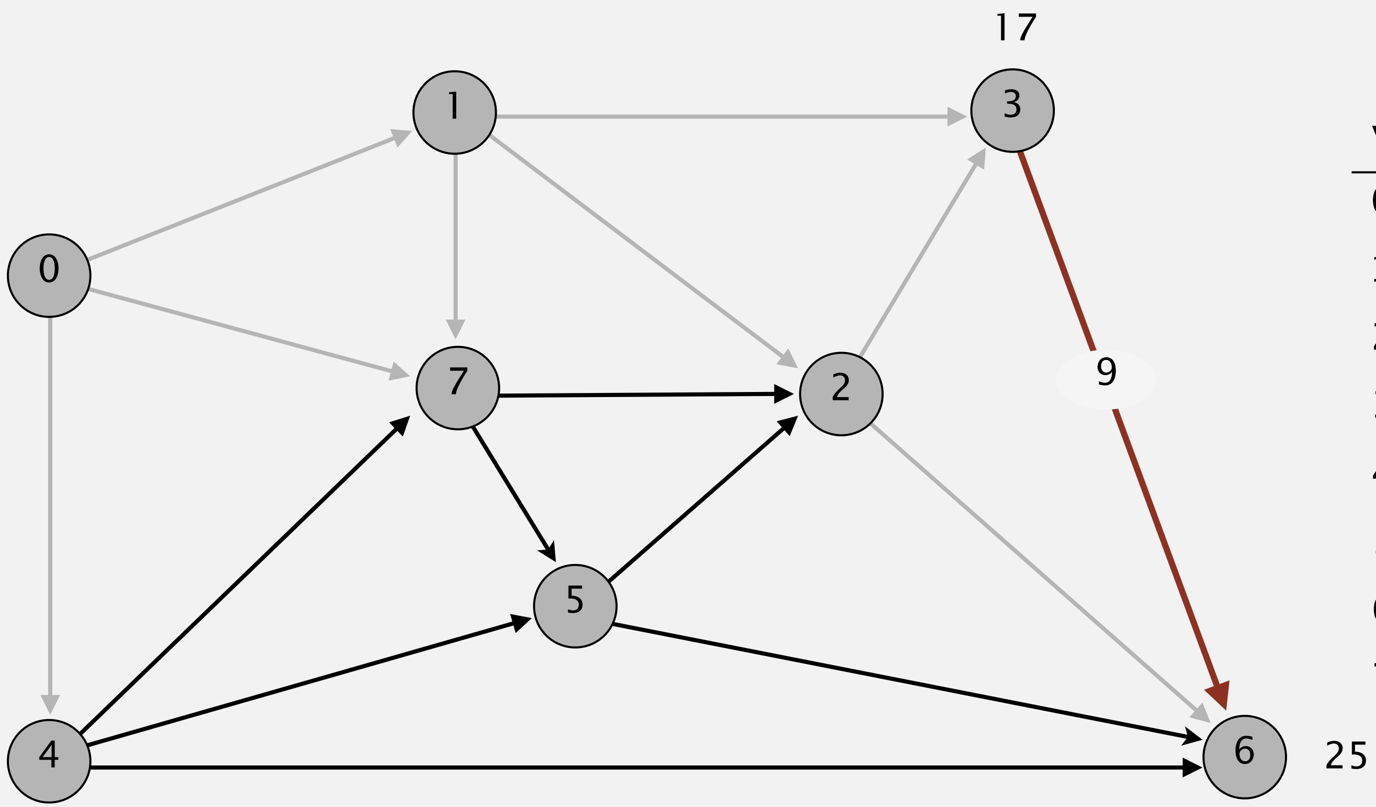
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



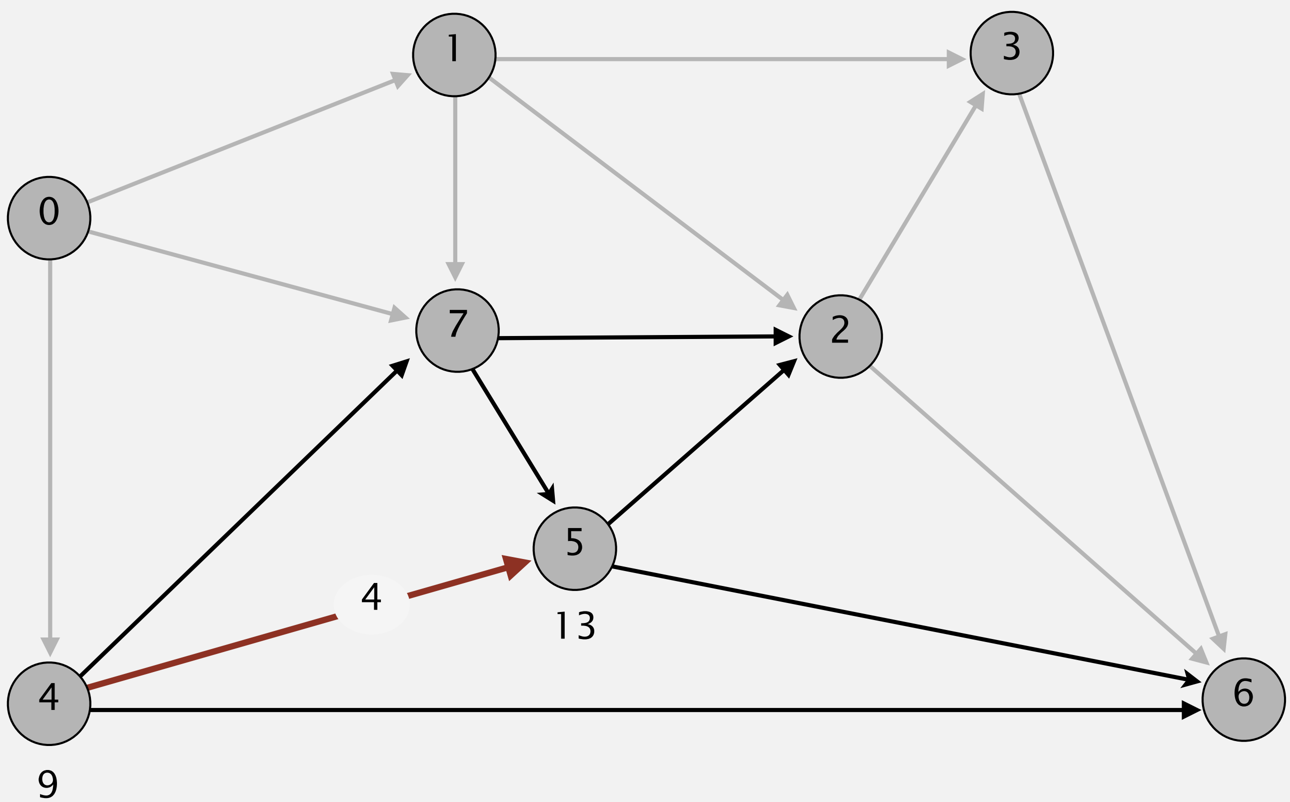
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



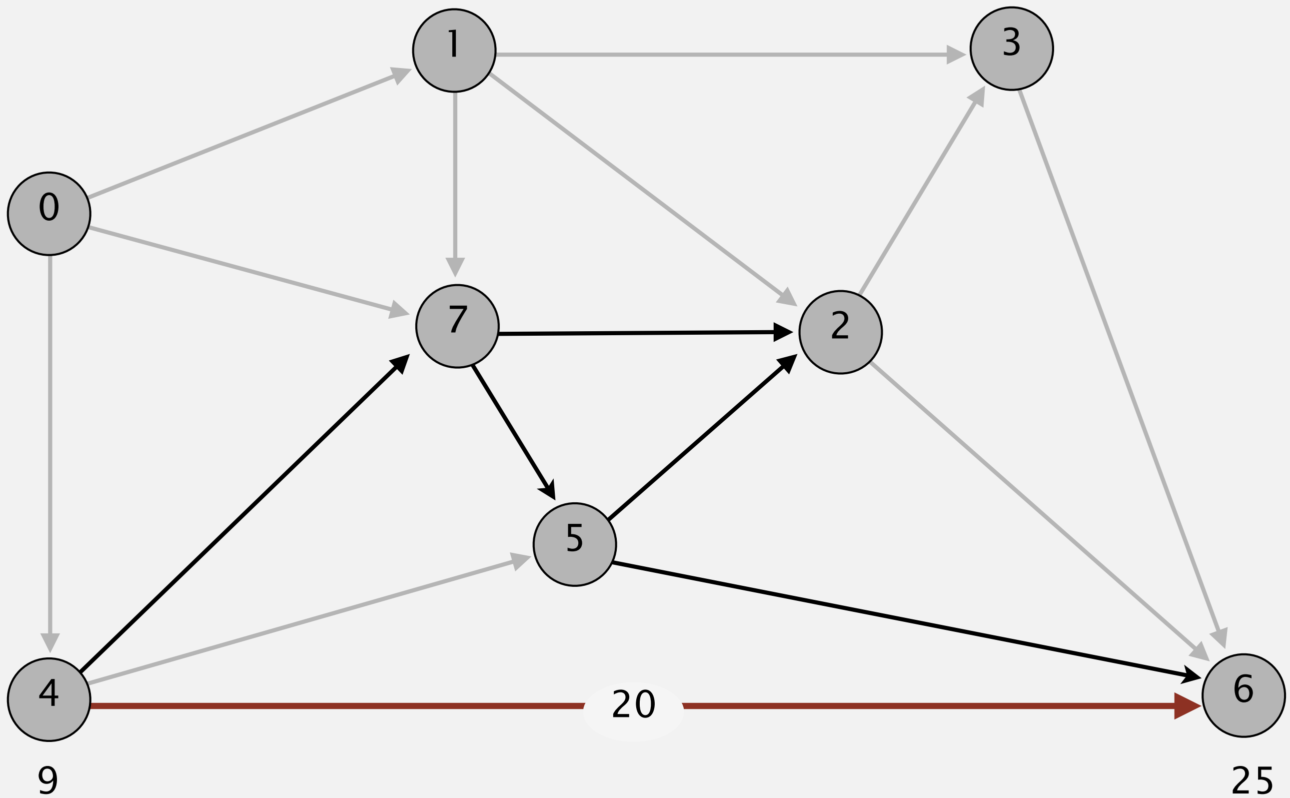
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



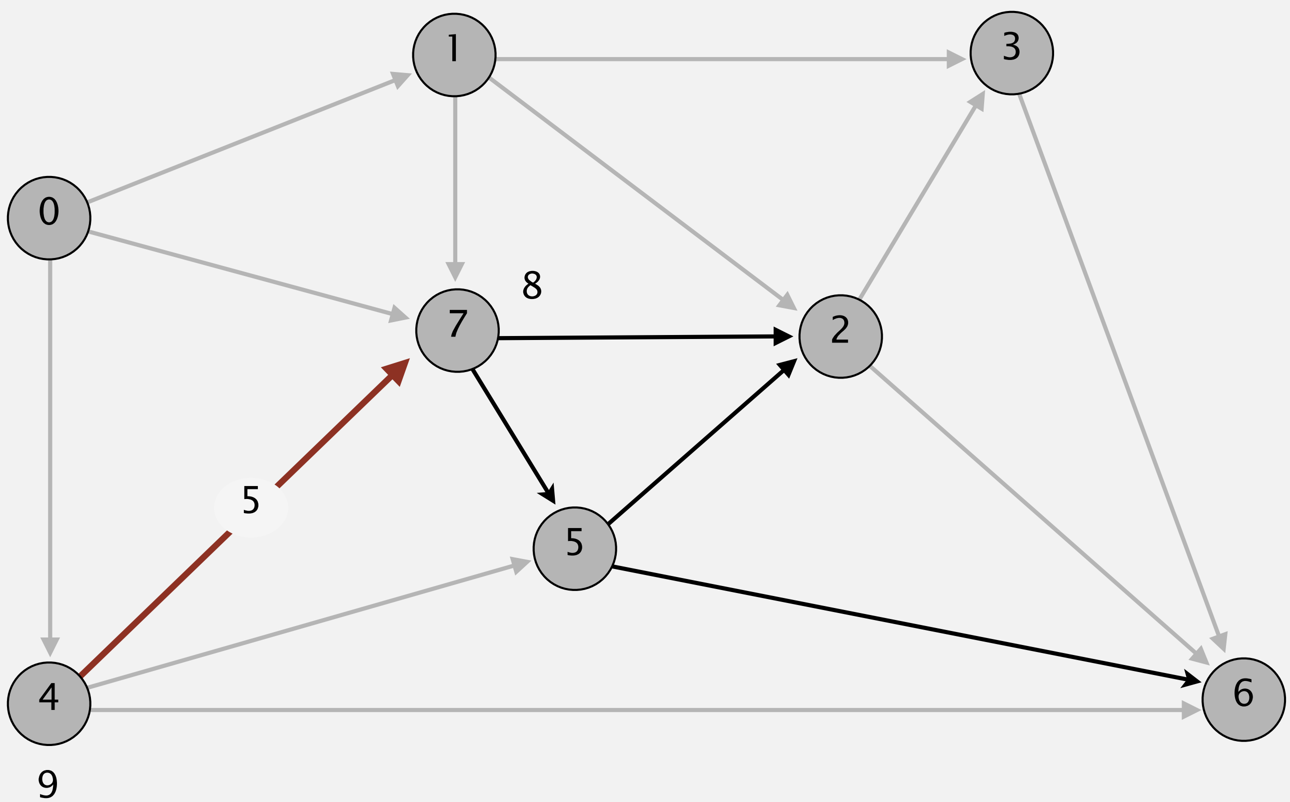
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



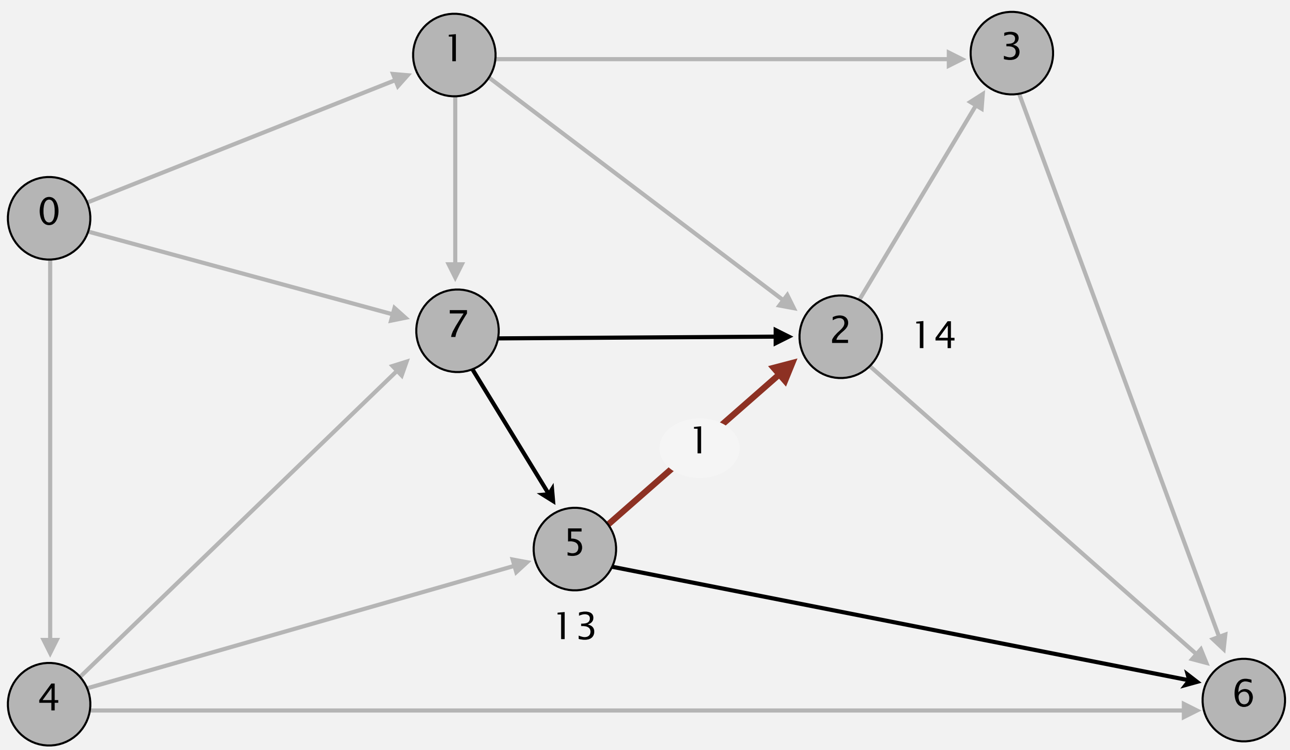
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



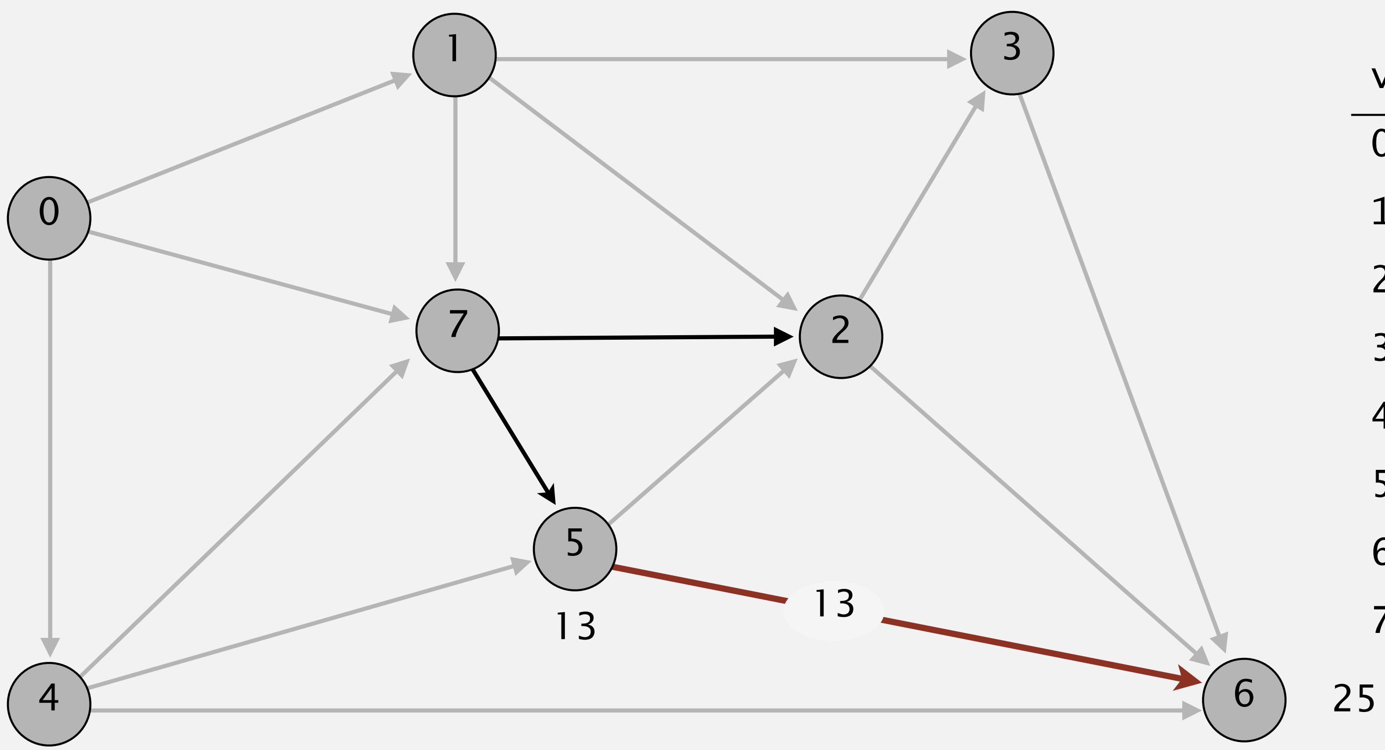
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



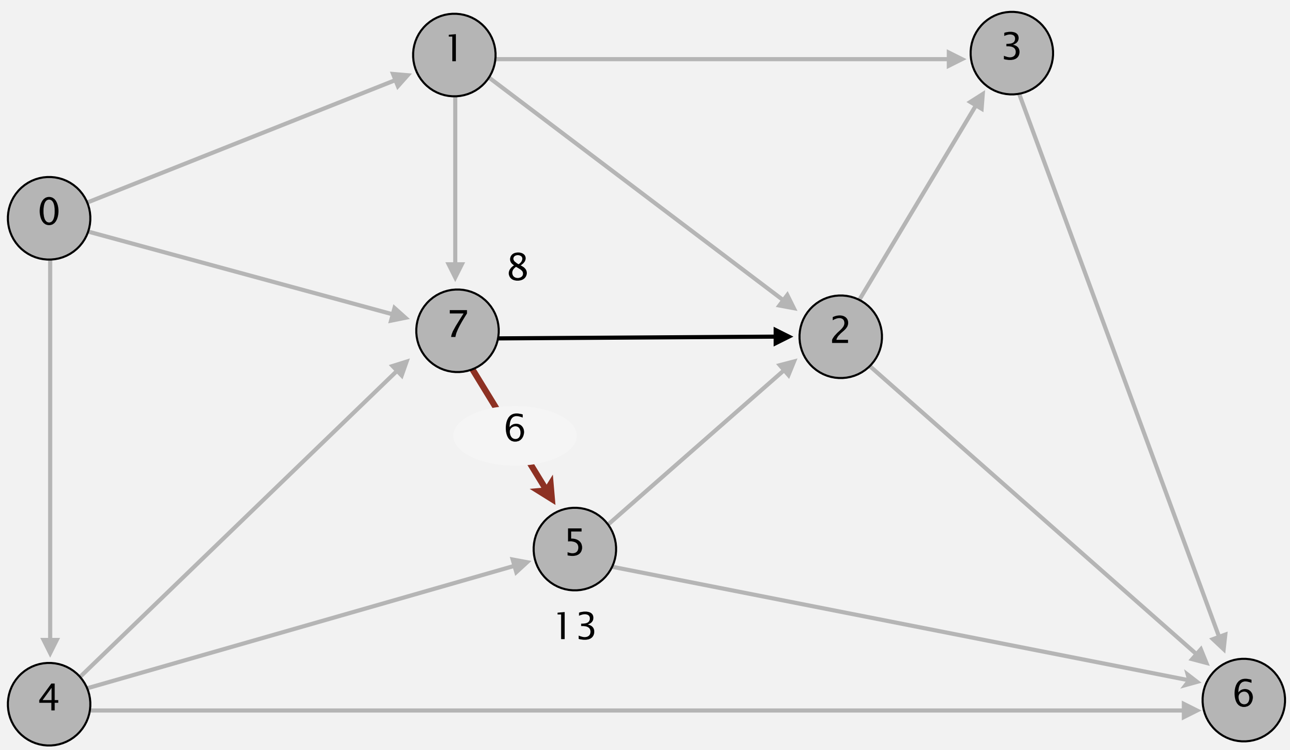
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



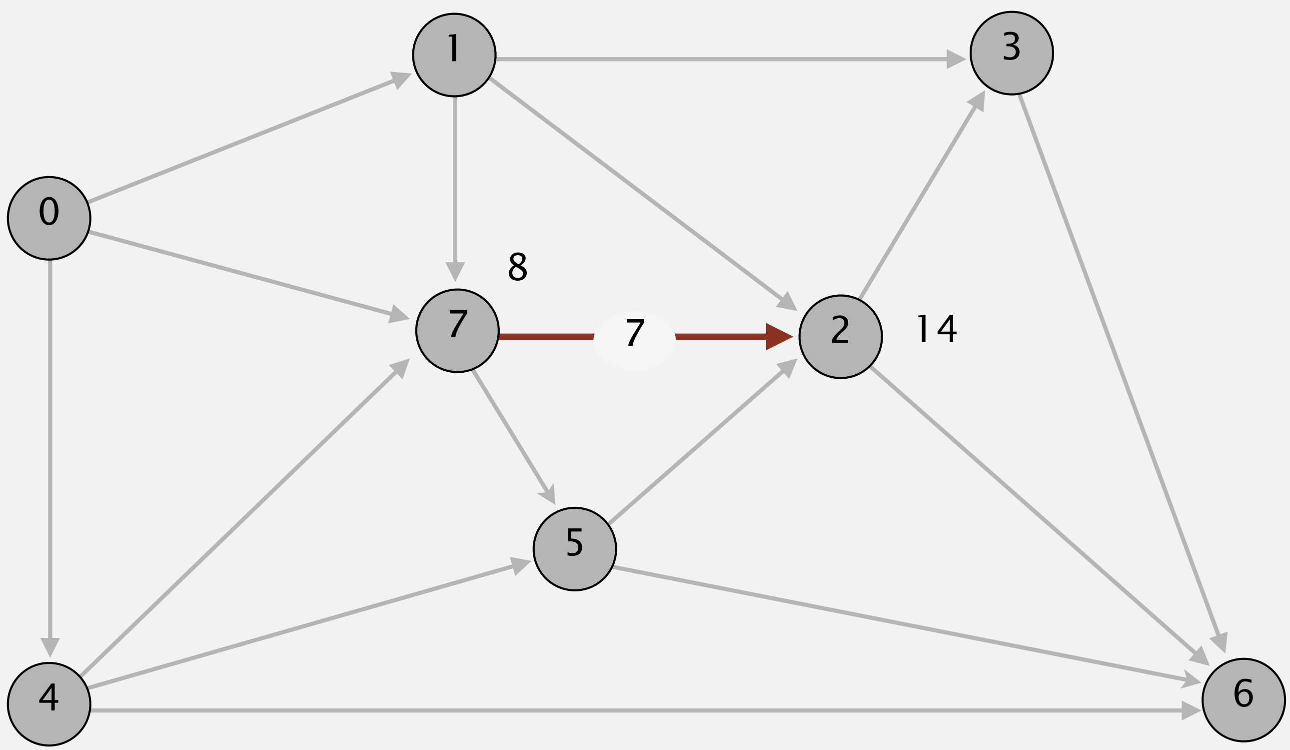
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



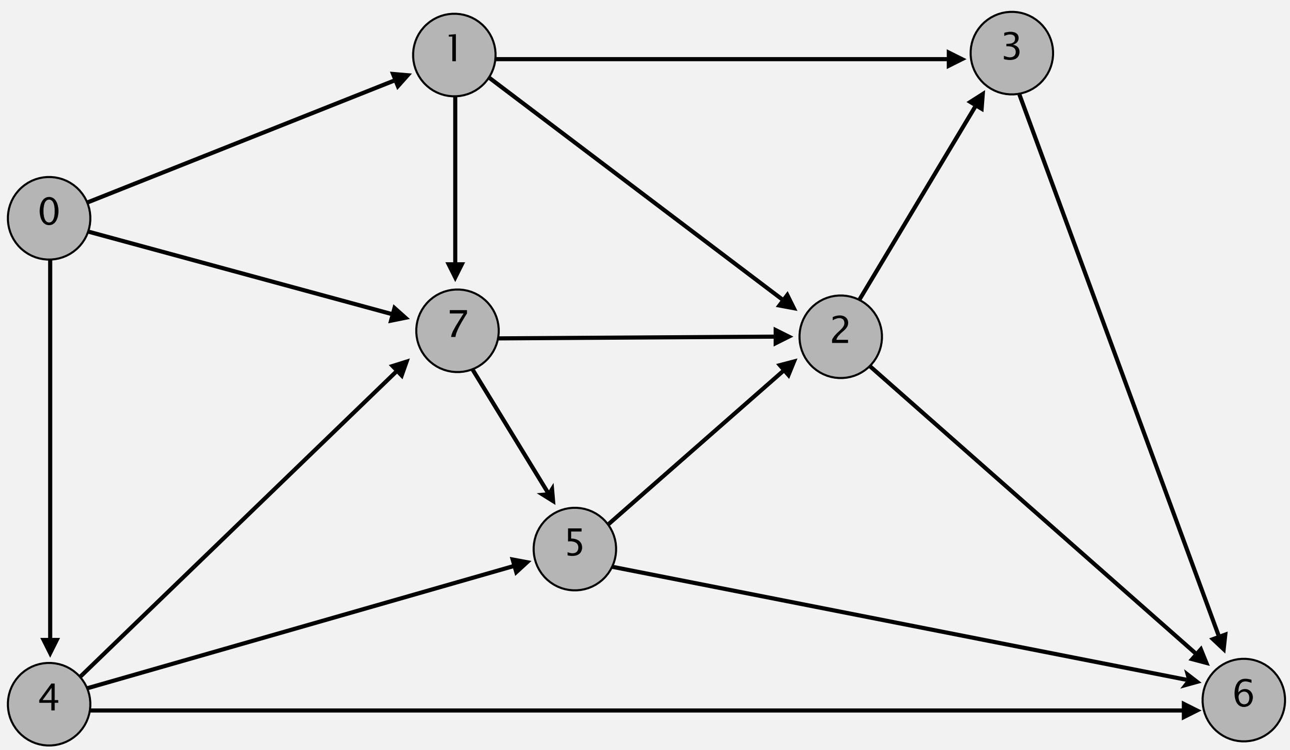
v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

pass 1

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



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

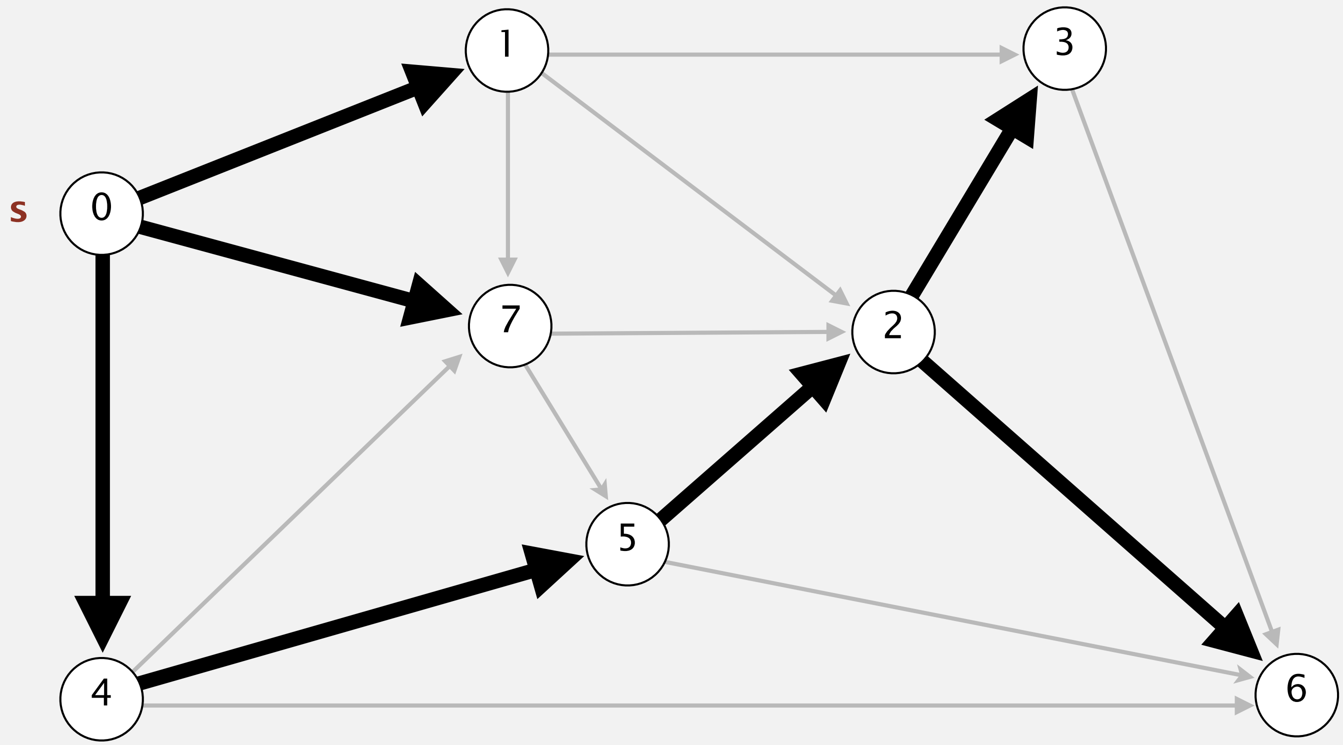
pass 2, 3, 4, 5, 6, 7 (no further changes)

0→1 0→4 0→7 1→2 1→3 1→7 2→3 2→6 3→6 4→5 4→6 4→7 5→2 5→6 7→5 7→2

↑

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



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

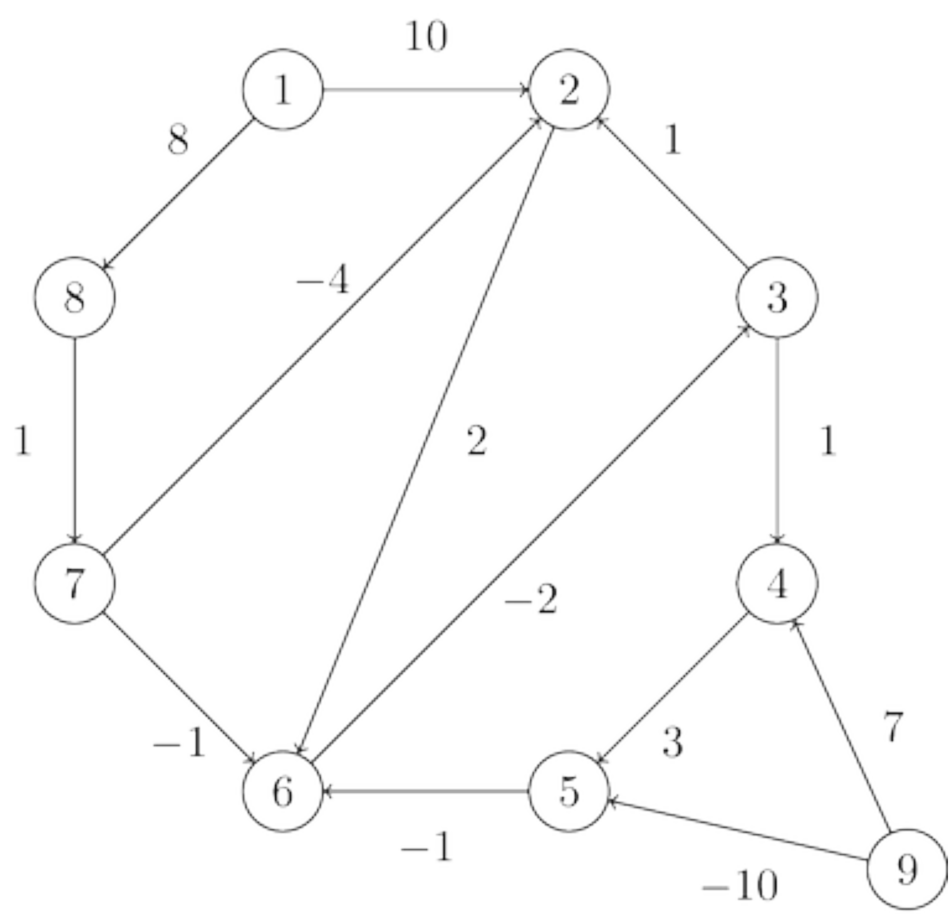
shortest-paths tree from vertex s

Bellman-Ford vs Dijkstra

- Bellman-Ford's worst-case running time is $|E||V|$ vs Dijkstra's $|E|\log|V|$.
  - Bellman-Ford's algorithm is queue-based.
- Both require $|V|$ extra space and can handle graphs with cycles.
- Only Bellman-Ford can handle negative weights, as long as there are no cycles that sum to a negative weight.

Practice Time

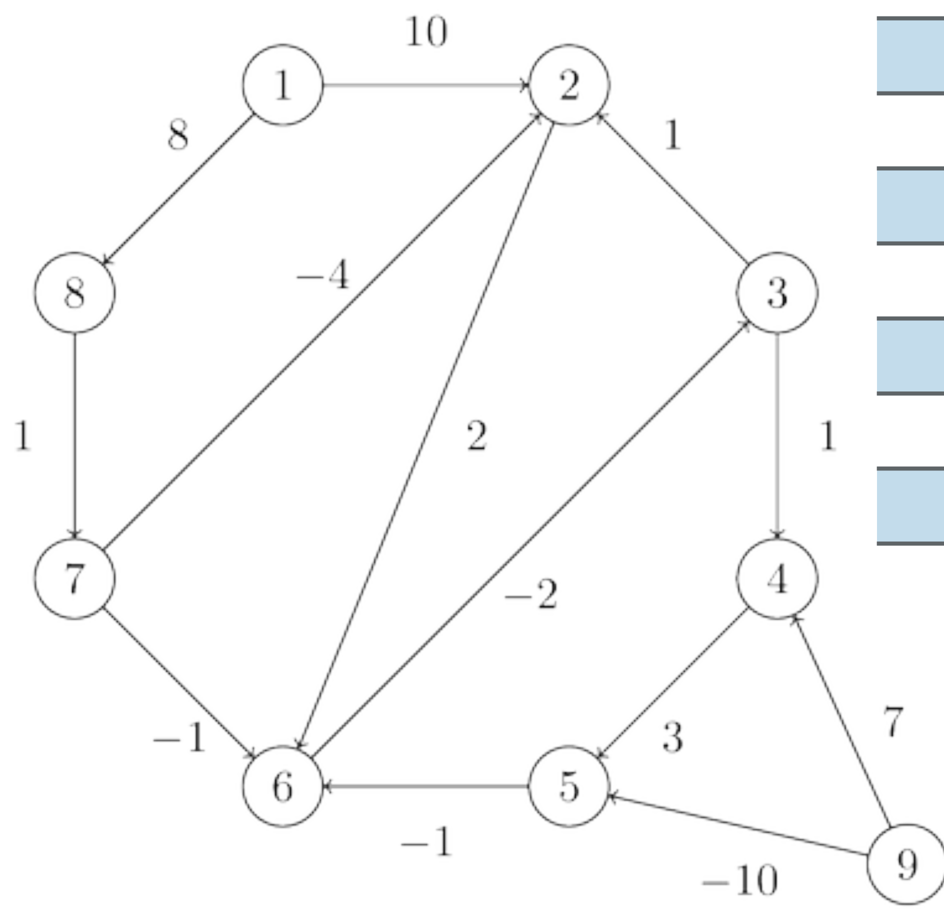
1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7



http://rosalind.info/problems/bf/

Practice Time

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

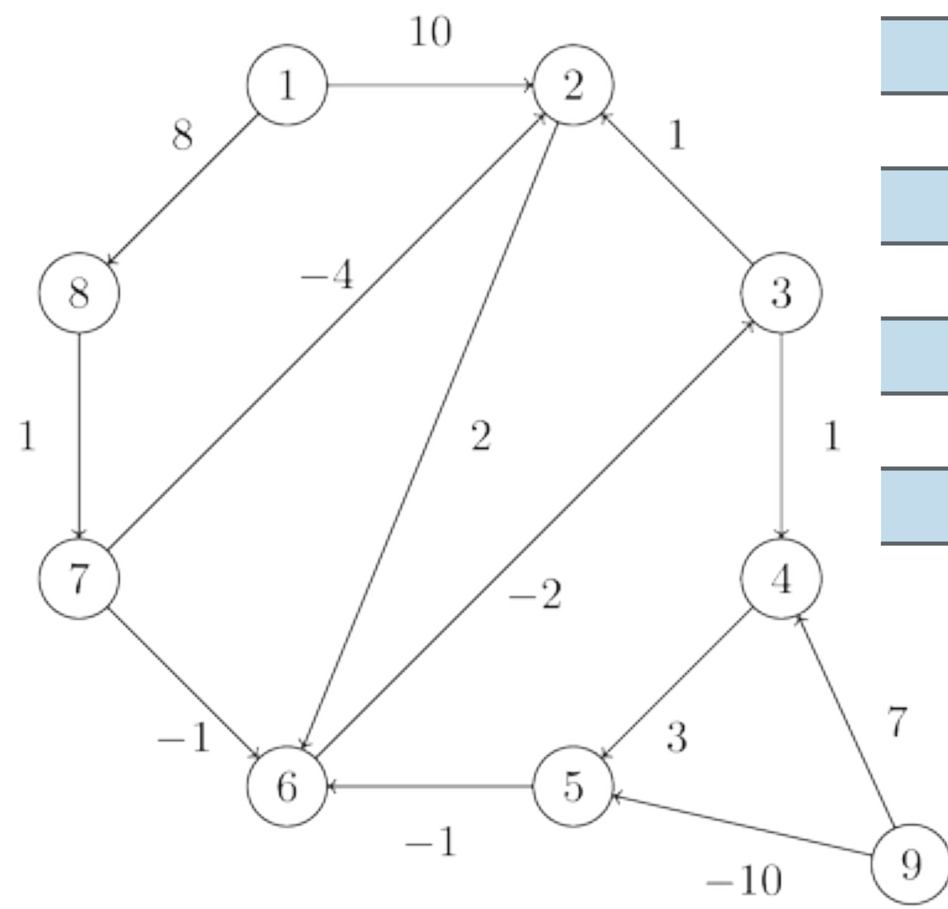


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	Inf	null
3	Inf	null
4	Inf	null
5	Inf	null
6	Inf	null
7	Inf	null
8	Inf	null
9	Inf	null

Practice Time - Pass 0

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

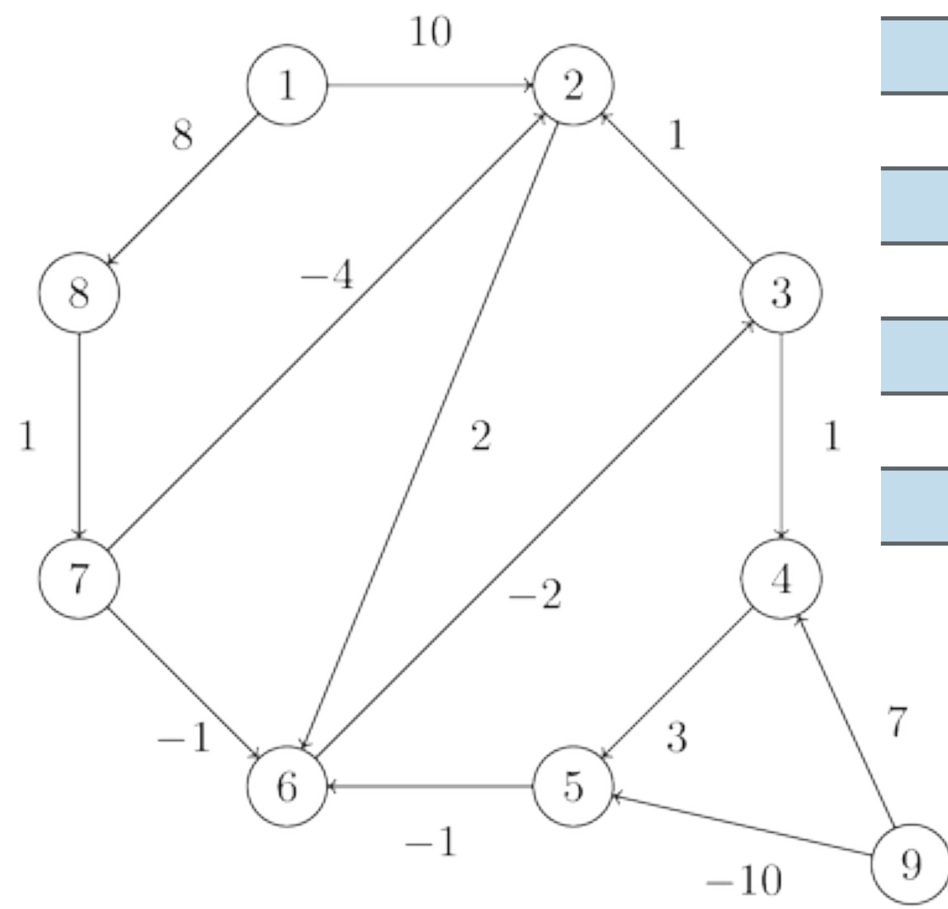


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	10	1–> 2
3	Inf	null
4	Inf	null
5	Inf	null
6	Inf	null
7	Inf	null
8	Inf	null
9	Inf	null

Practice Time - Pass 0

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

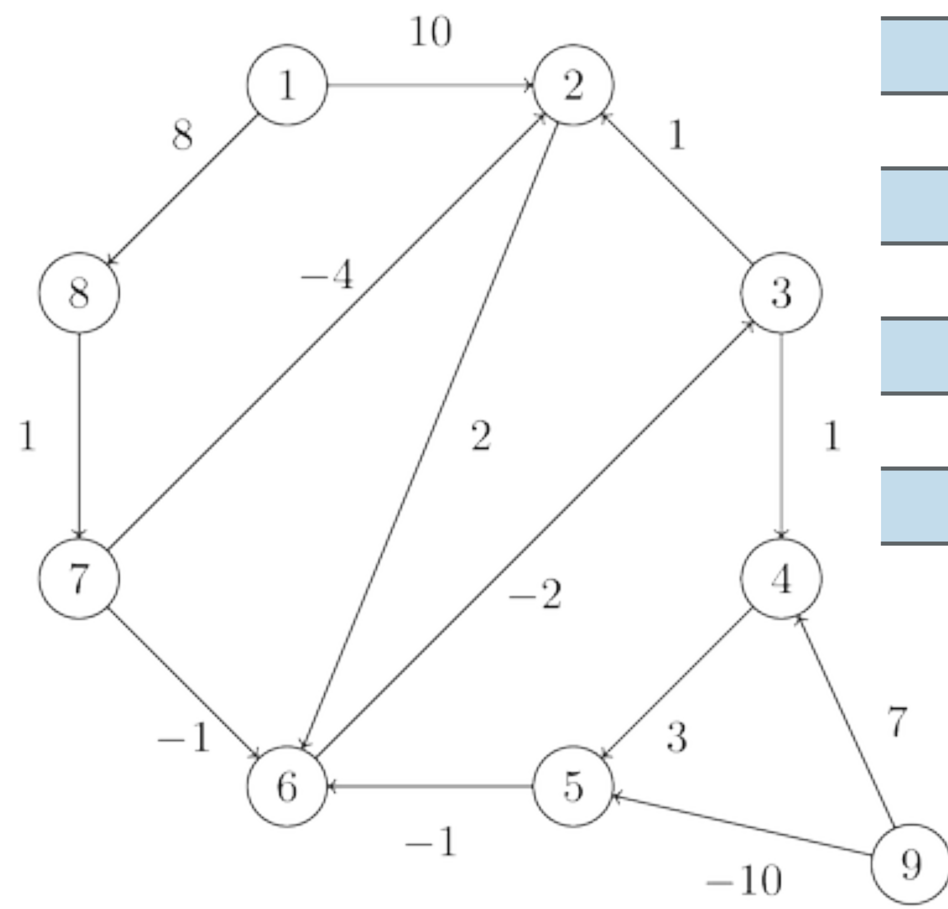


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	10	1–> 2
3	Inf	null
4	Inf	null
5	Inf	null
6	Inf	null
7	Inf	null
8	Inf	null
9	Inf	null

Practice Time - Pass 0

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

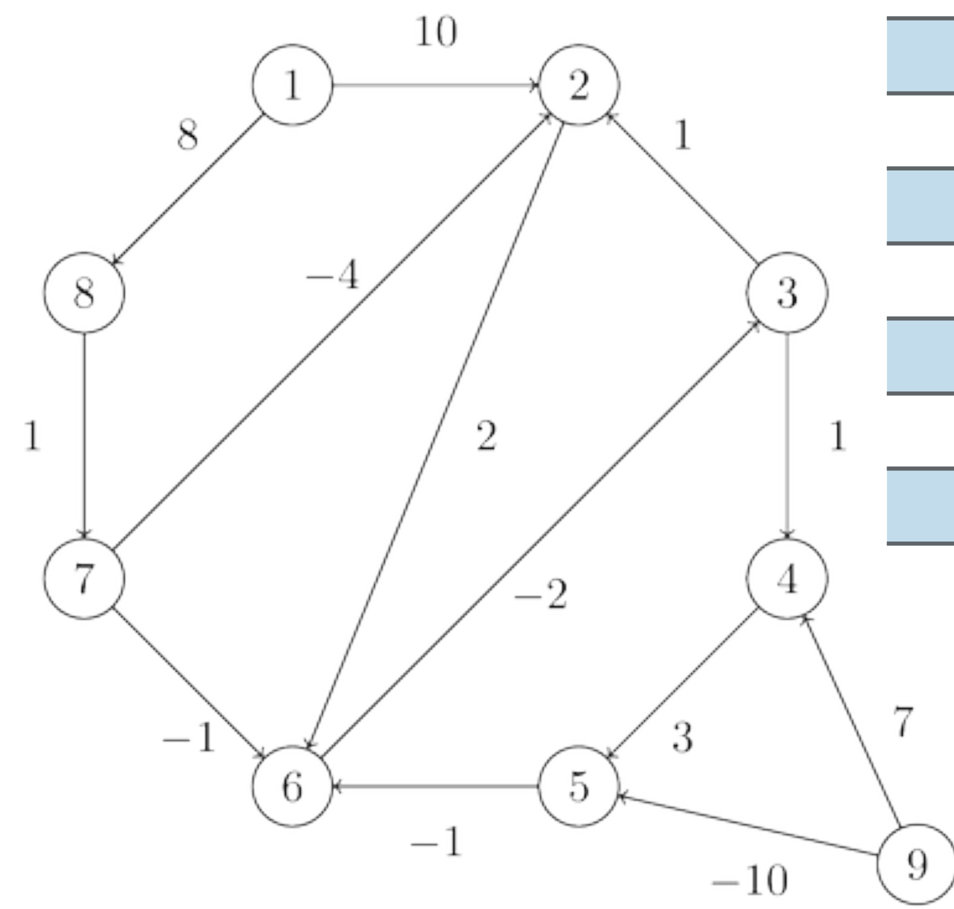


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	10	1–> 2
3	Inf	null
4	Inf	null
5	Inf	null
6	Inf	null
7	Inf	null
8	Inf	null
9	Inf	null

Practice Time - Pass 0

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

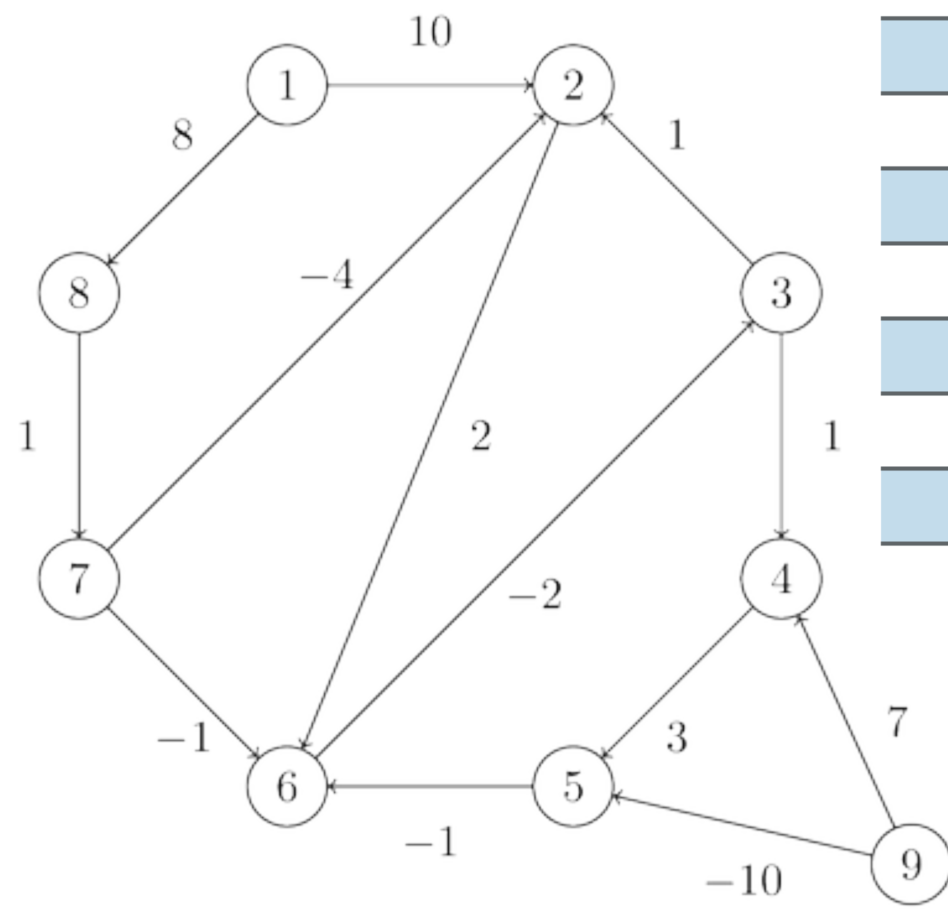


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	10	1–> 2
3	Inf	null
4	Inf	null
5	Inf	null
6	Inf	null
7	Inf	null
8	Inf	null
9	Inf	null

Practice Time - Pass 0

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

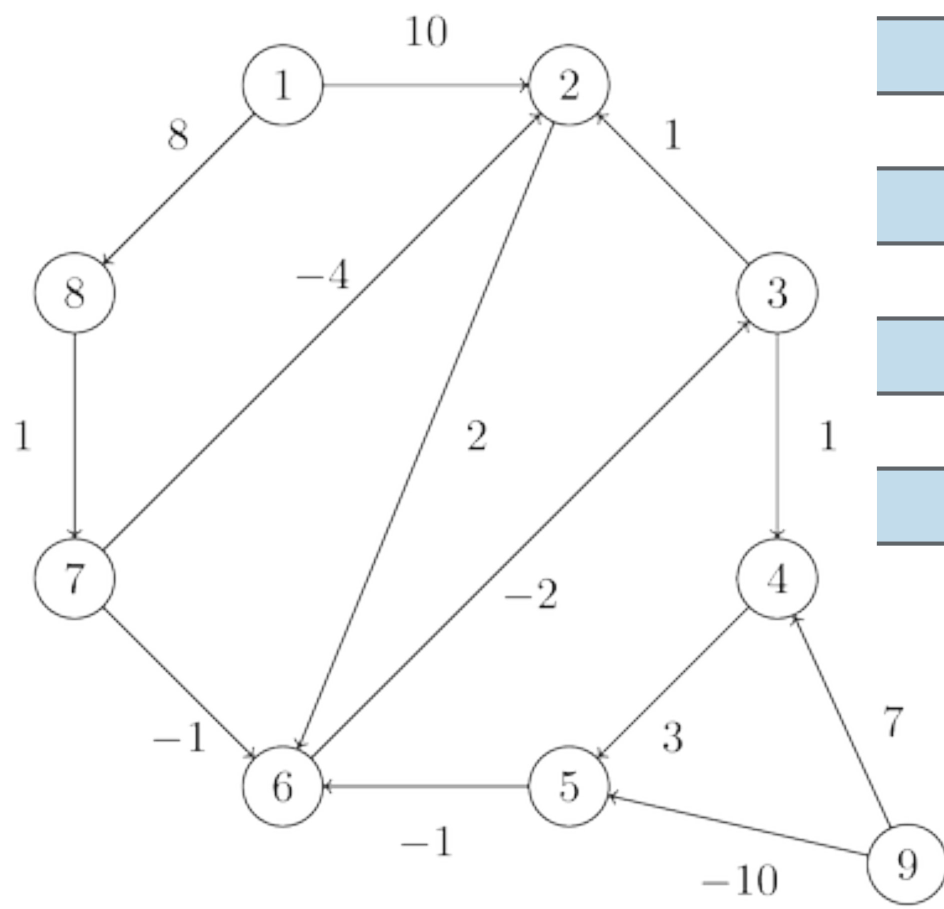


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	10	1–> 2
3	Inf	null
4	Inf	null
5	Inf	null
6	Inf	null
7	Inf	null
8	Inf	null
9	Inf	null

Practice Time - Pass 0

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

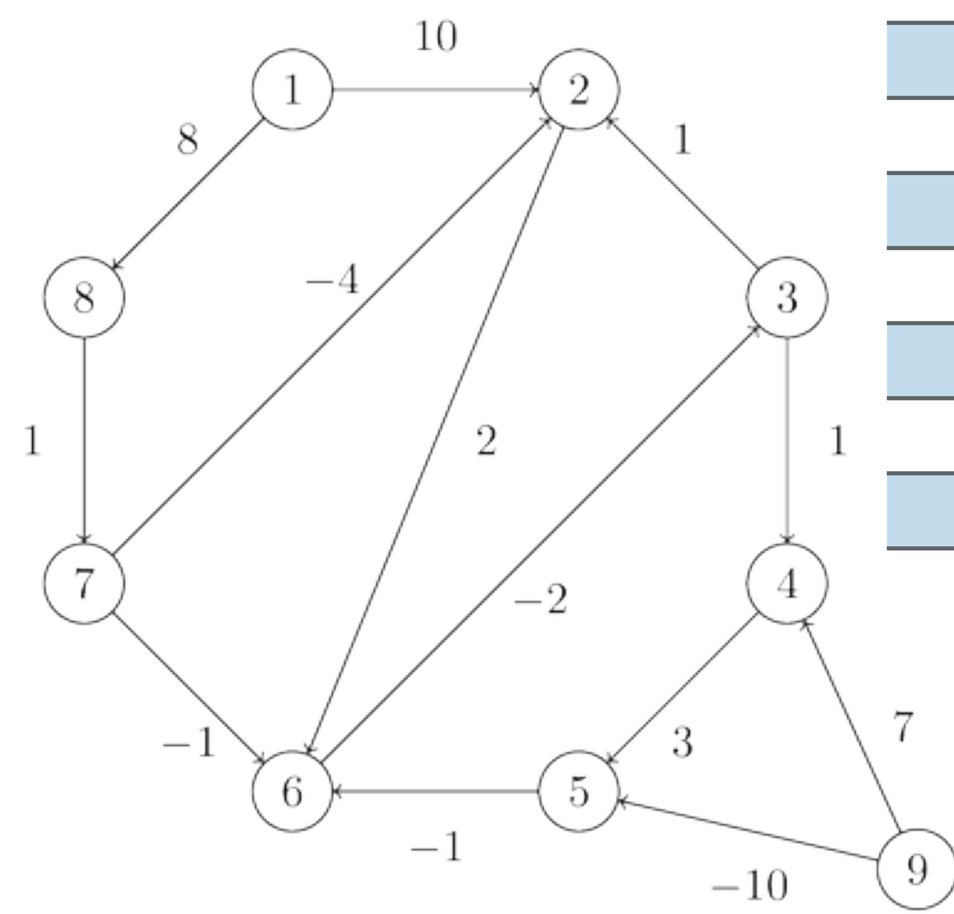


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	10	1–> 2
3	Inf	null
4	Inf	null
5	Inf	null
6	Inf	null
7	Inf	null
8	Inf	null
9	Inf	null

Practice Time - Pass 0

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

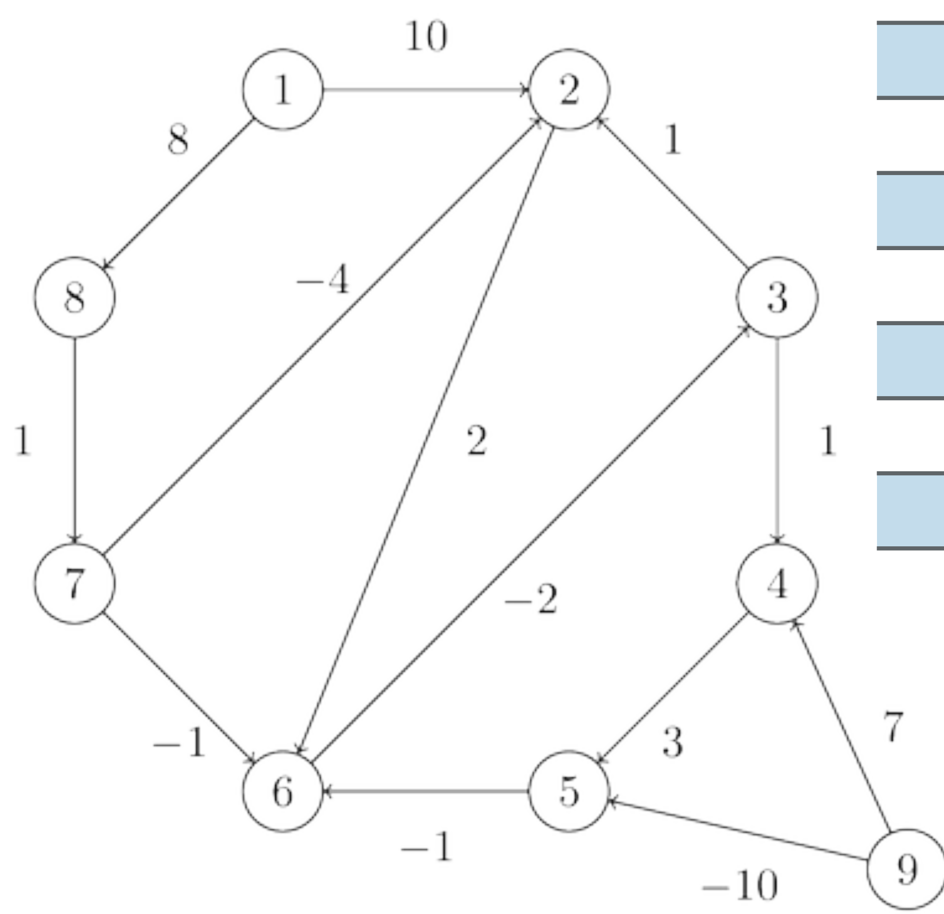


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	10	1–> 2
3	Inf	null
4	Inf	null
5	Inf	null
6	Inf	null
7	Inf	null
8	Inf	null
9	Inf	null

Practice Time - Pass 0

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

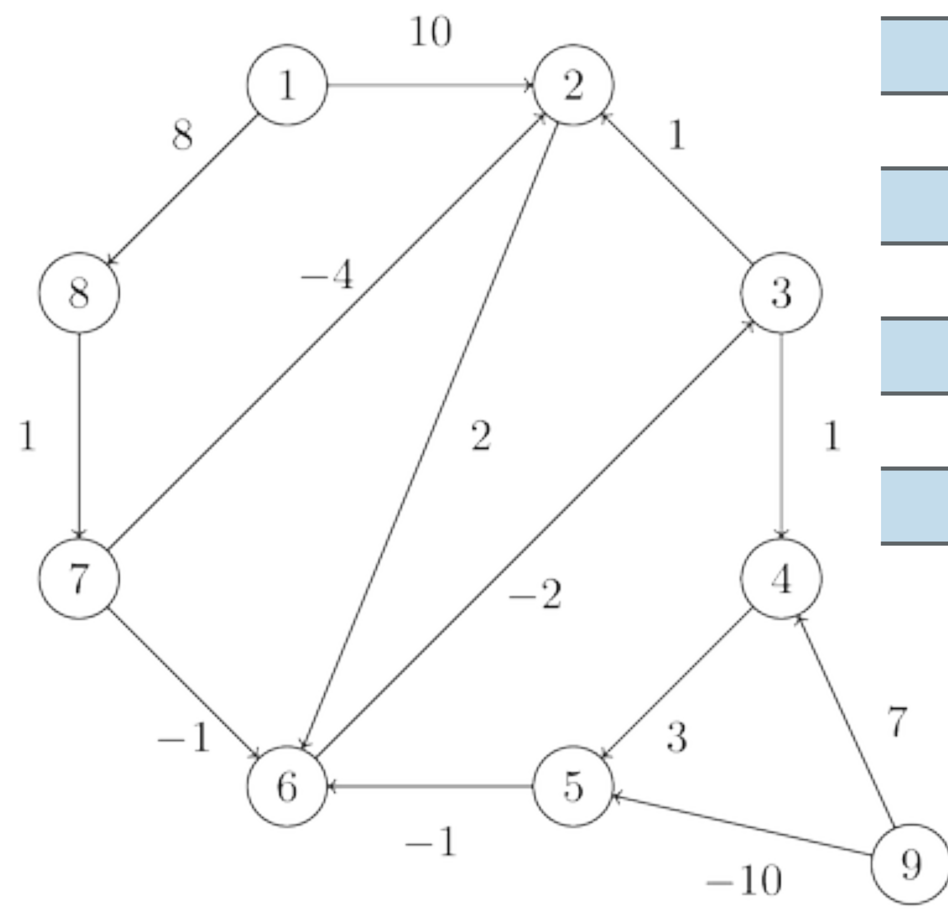


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	10	1–> 2
3	Inf	null
4	Inf	null
5	Inf	null
6	Inf	null
7	Inf	null
8	8	1–>8
9	Inf	null

Practice Time - Pass 0

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

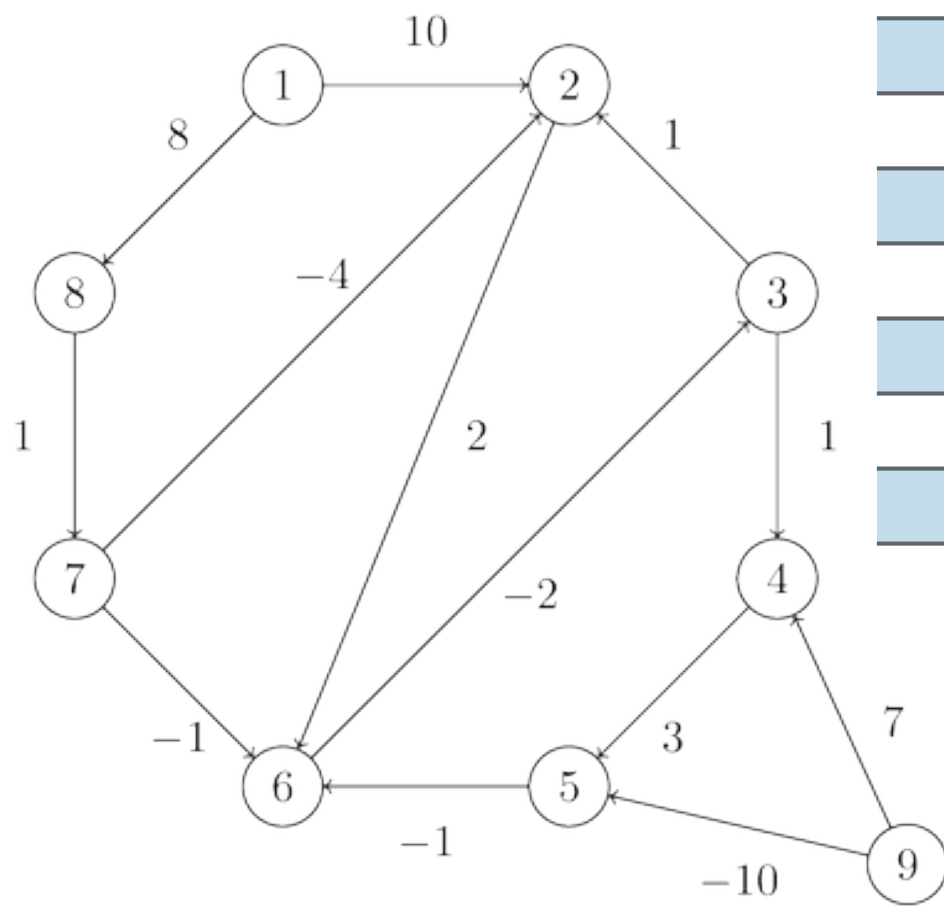


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	10	1–> 2
3	Inf	null
4	Inf	null
5	Inf	null
6	Inf	null
7	Inf	null
8	8	1–>8
9	Inf	null

Practice Time - Pass 0

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

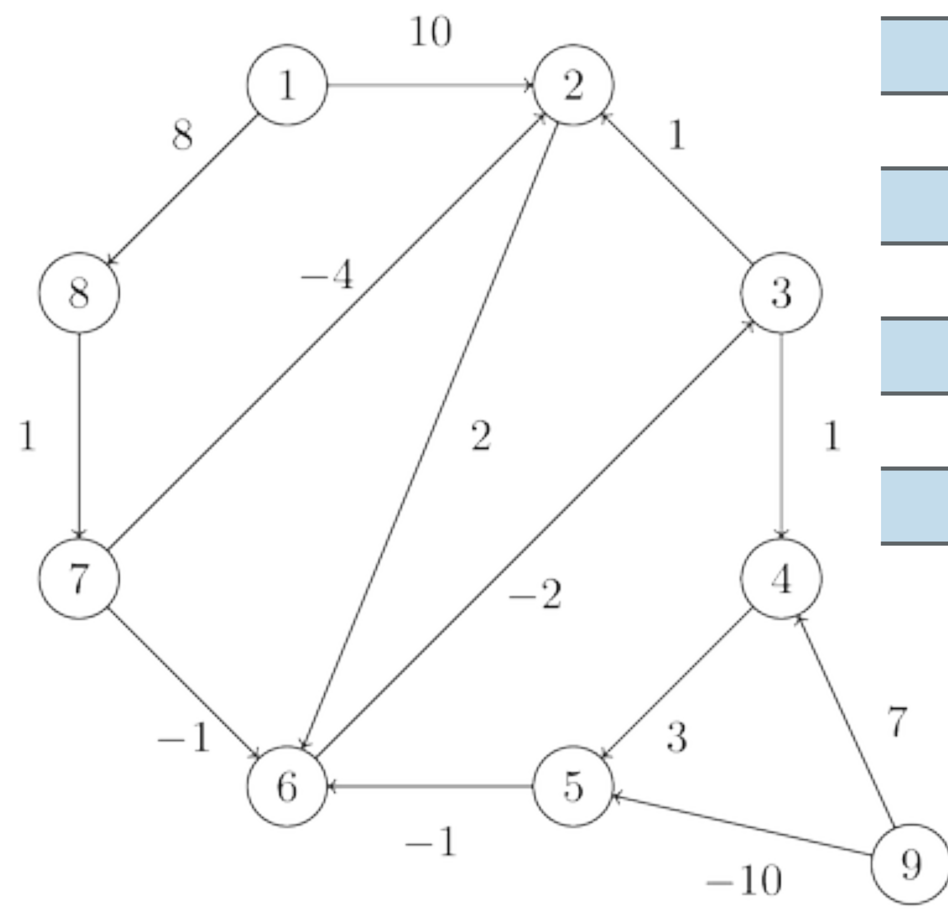


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	10	1–> 2
3	Inf	null
4	Inf	null
5	Inf	null
6	12	2–>6
7	Inf	null
8	8	1–>8
9	Inf	null

Practice Time - Pass 0

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

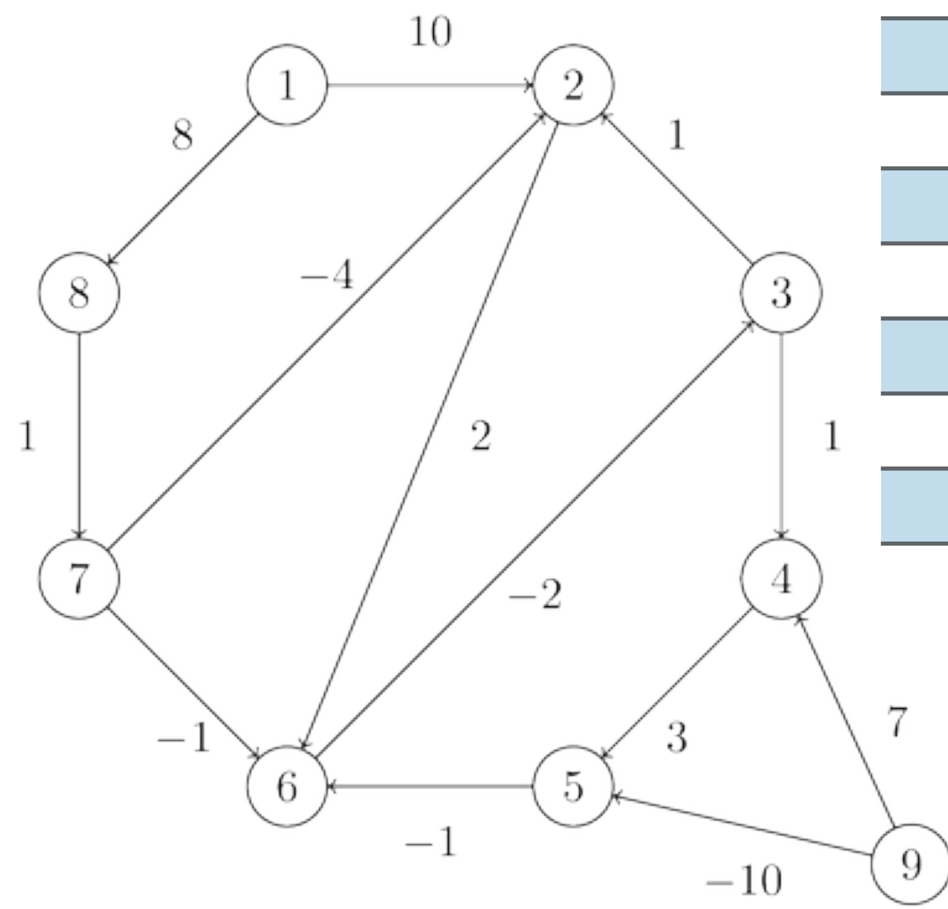


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	10	1–> 2
3	10	6–>3
4	Inf	null
5	Inf	null
6	12	2–>6
7	Inf	null
8	8	1–>8
9	Inf	null

Practice Time - Pass 0

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

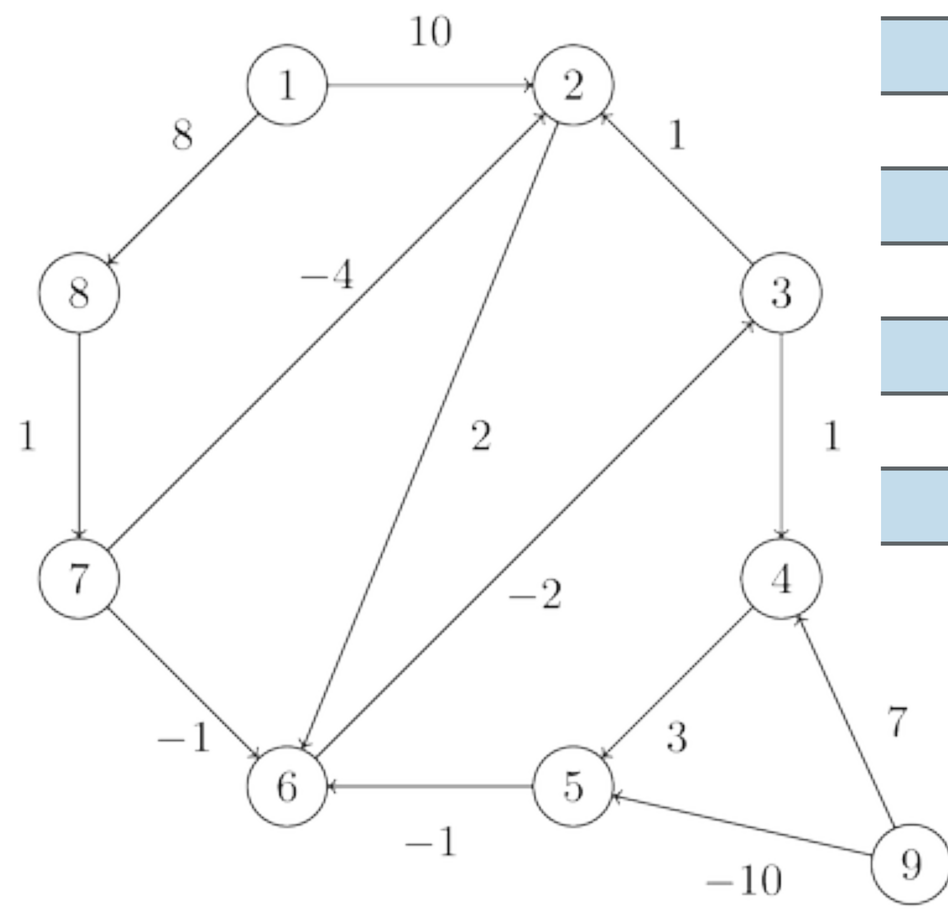


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	10	1–> 2
3	10	6–>3
4	Inf	null
5	Inf	null
6	12	2–>6
7	Inf	null
8	8	1–>8
9	Inf	null

Practice Time - Pass 0

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

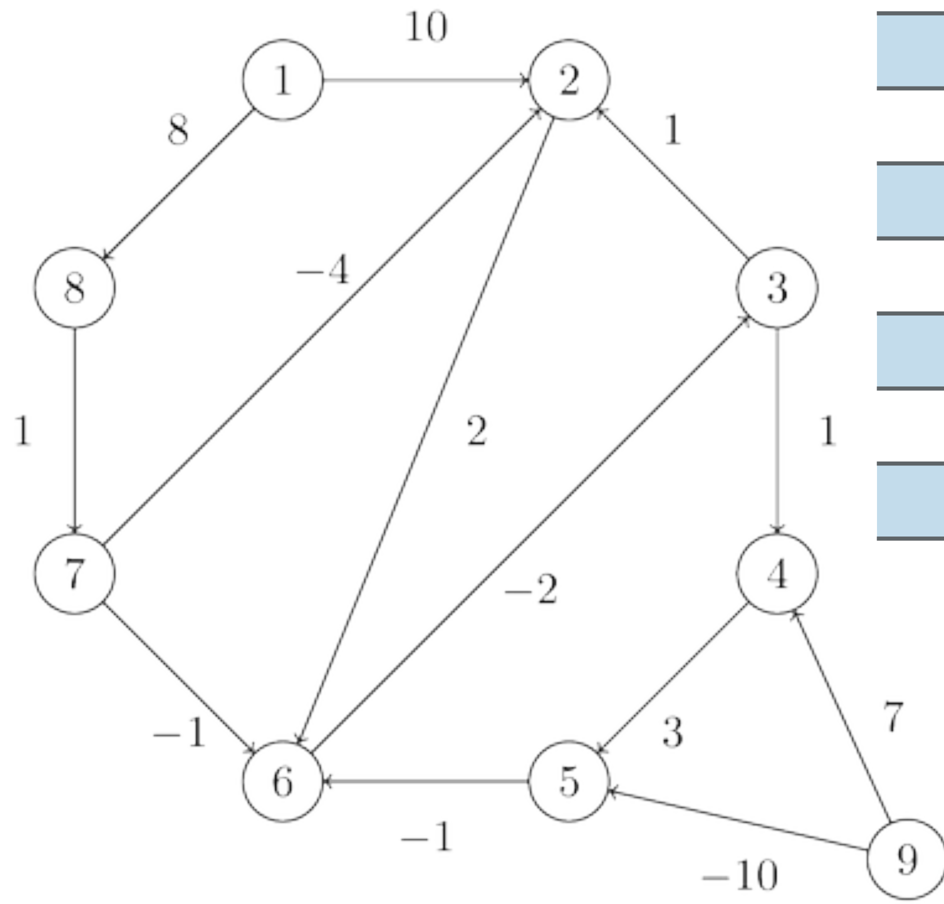


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	10	1–> 2
3	10	6–>3
4	Inf	null
5	Inf	null
6	12	2–>6
7	Inf	null
8	8	1–>8
9	Inf	null

Practice Time - Pass 1

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

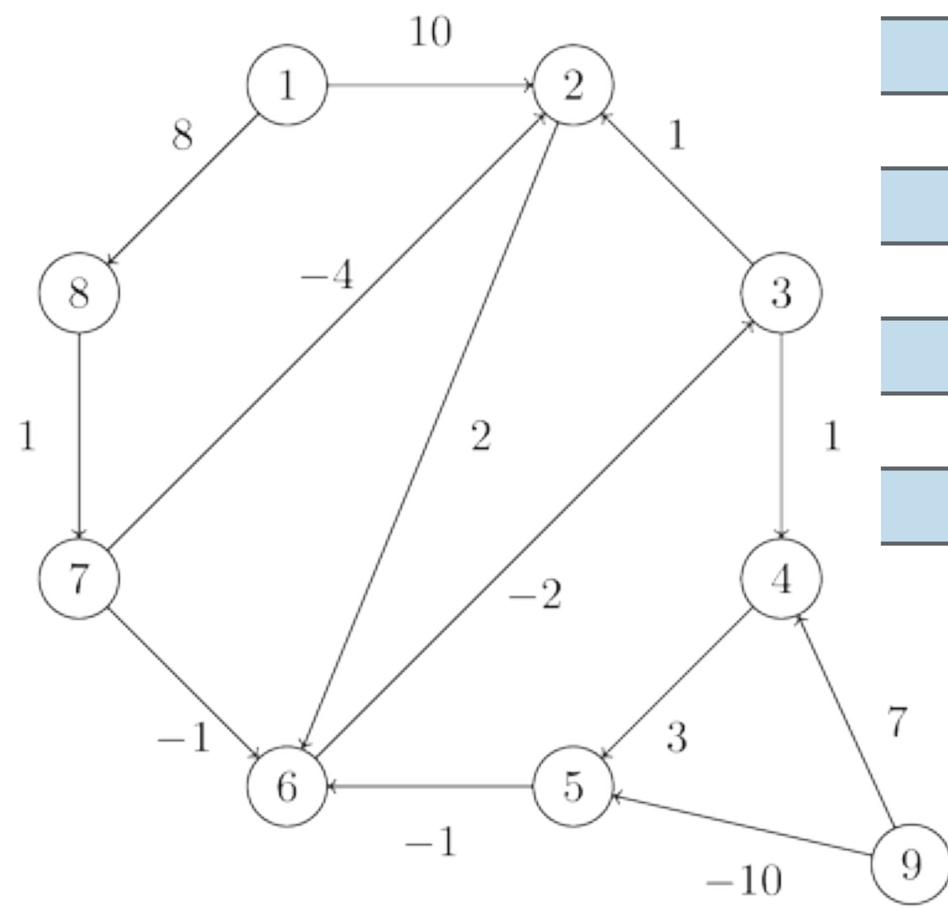


v	distTo[]	edgeTo[]
1	0	-
2	10	1–> 2
3	10	6–>3
4	Inf	null
5	Inf	null
6	12	2–>6
7	Inf	null
8	8	1–>8
9	Inf	null

http://rosalind.info/problems/bf/

Practice Time - Pass 1

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

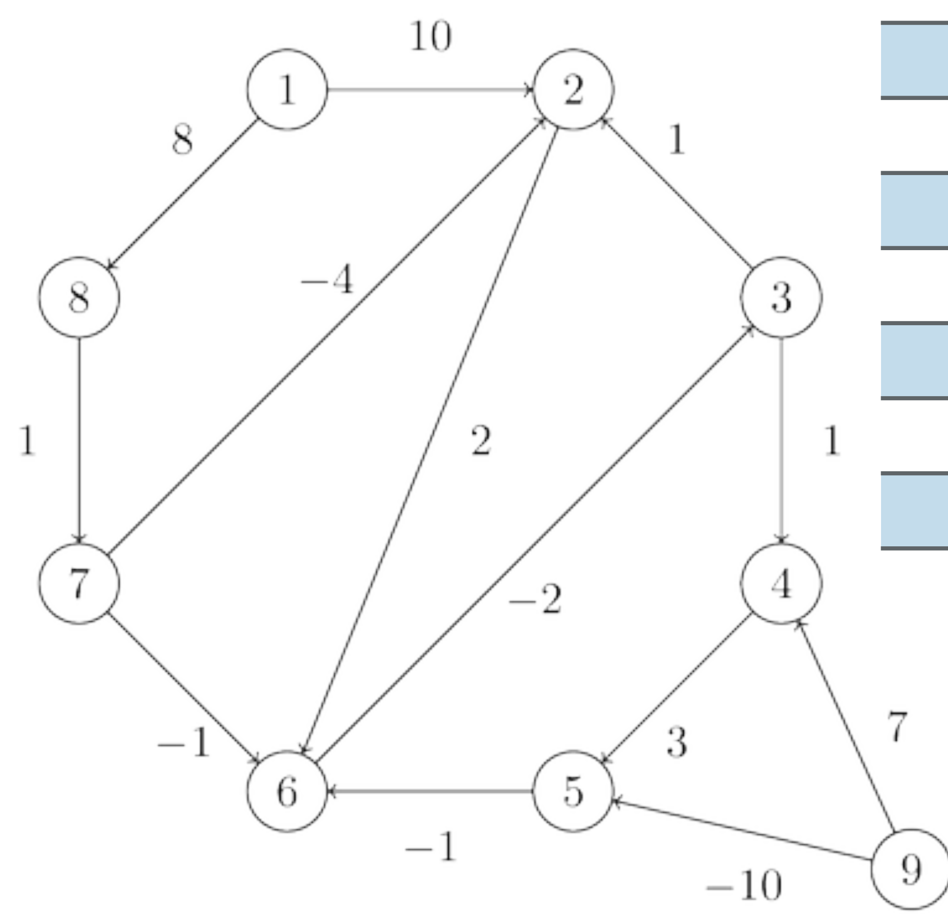


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	10	1–> 2
3	10	6–>3
4	Inf	null
5	Inf	null
6	12	2–>6
7	Inf	null
8	8	1–>8
9	Inf	null

Practice Time - Pass 1

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

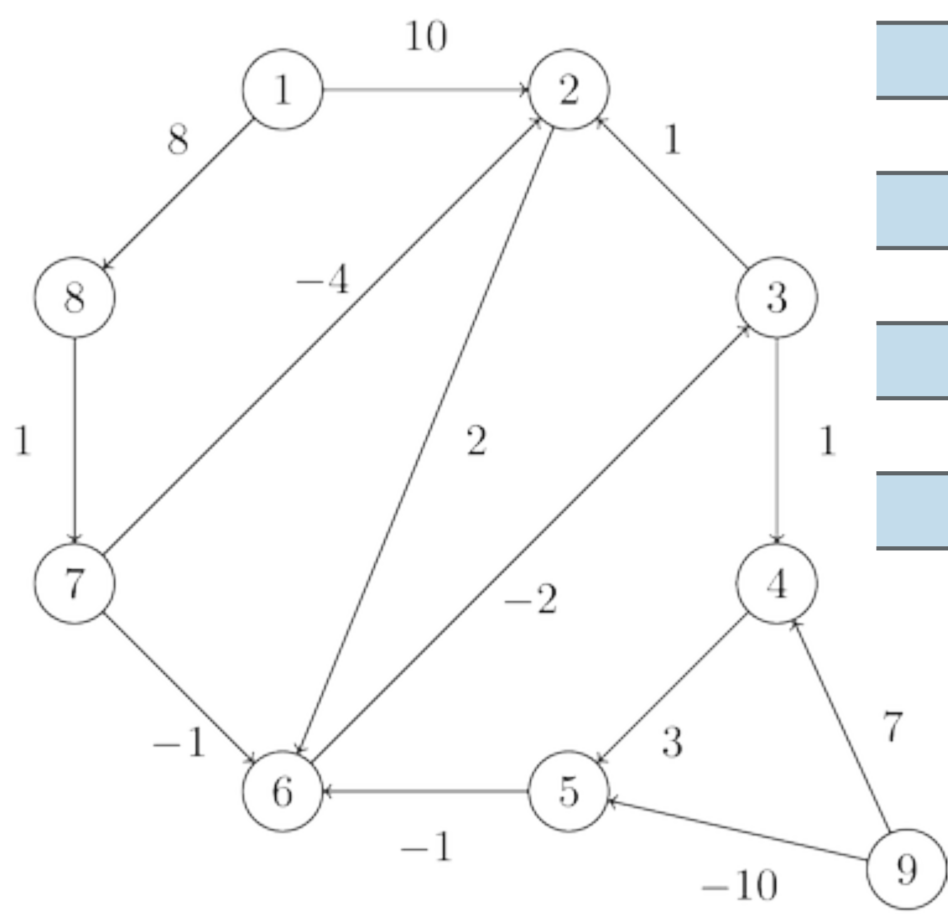


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	10	1–> 2
3	10	6–>3
4	11	3–>4
5	Inf	null
6	12	2–>6
7	Inf	null
8	8	1–>8
9	Inf	null

Practice Time - Pass 1

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

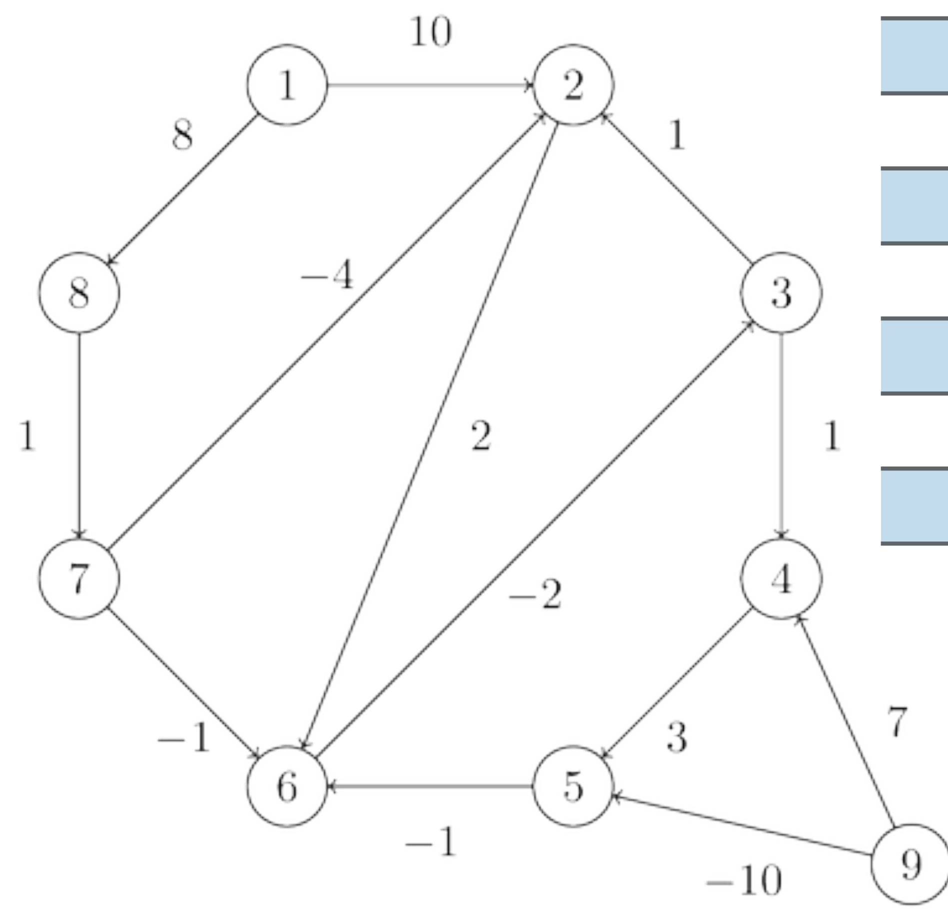


v	distTo[]	edgeTo[]
1	0	-
2	10	1–> 2
3	10	6–>3
4	11	3–>4
5	14	4–>5
6	12	2–>6
7	Inf	null
8	8	1–>8
9	Inf	null

http://rosalind.info/problems/bf/

Practice Time - Pass 1

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

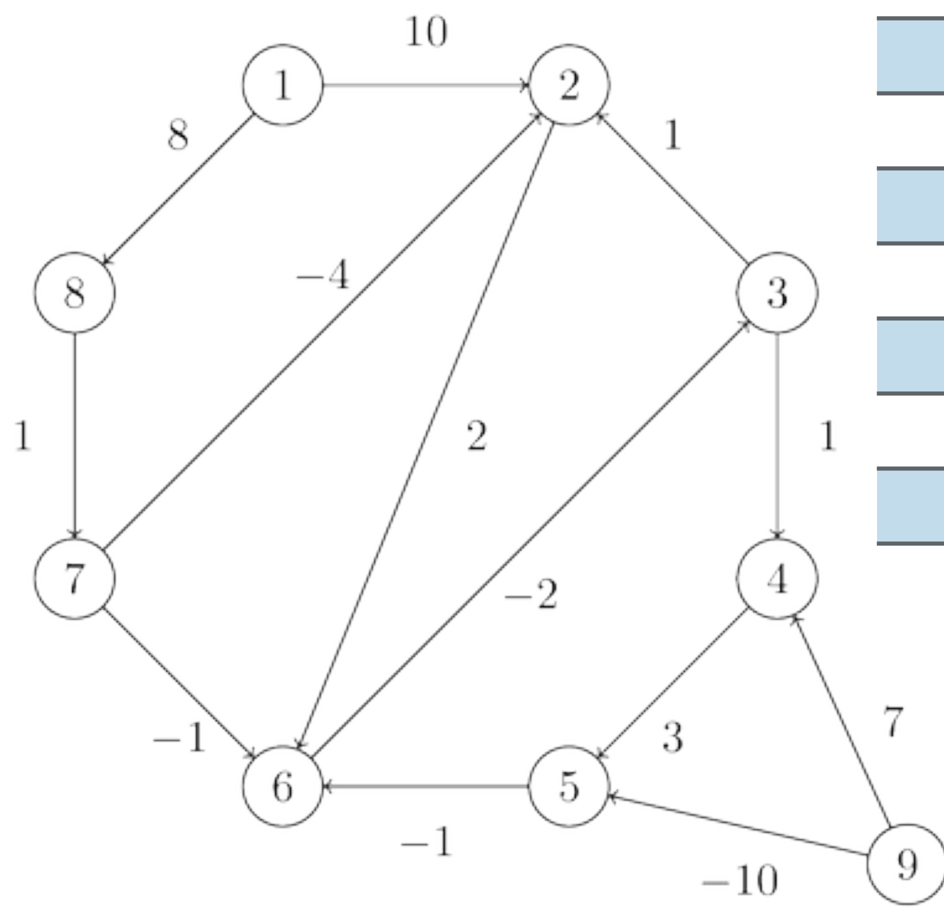


v	distTo[]	edgeTo[]
1	0	-
2	10	1–> 2
3	10	6–>3
4	11	3–>4
5	14	4–>5
6	12	2–>6
7	Inf	null
8	8	1–>8
9	Inf	null

http://rosalind.info/problems/bf/

Practice Time - Pass 1

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

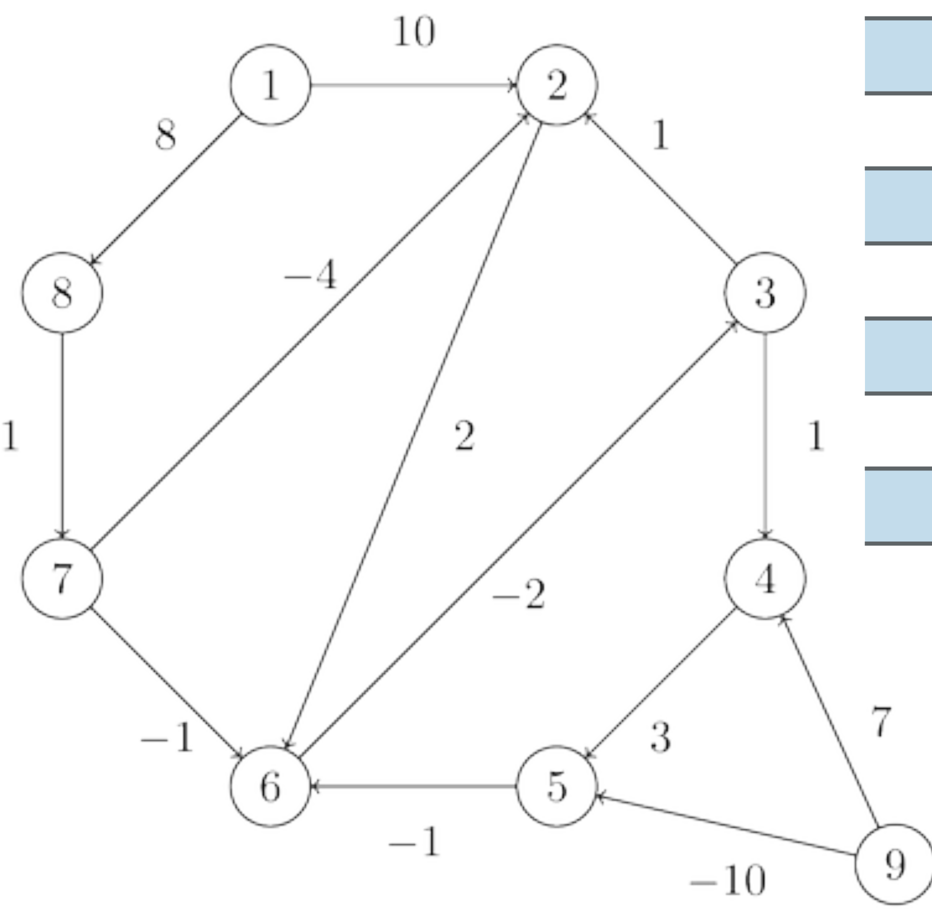


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	10	1–> 2
3	10	6–>3
4	11	3–>4
5	14	4–>5
6	12	2–>6
7	Inf	null
8	8	1–>8
9	Inf	null

Practice Time - Pass 1

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

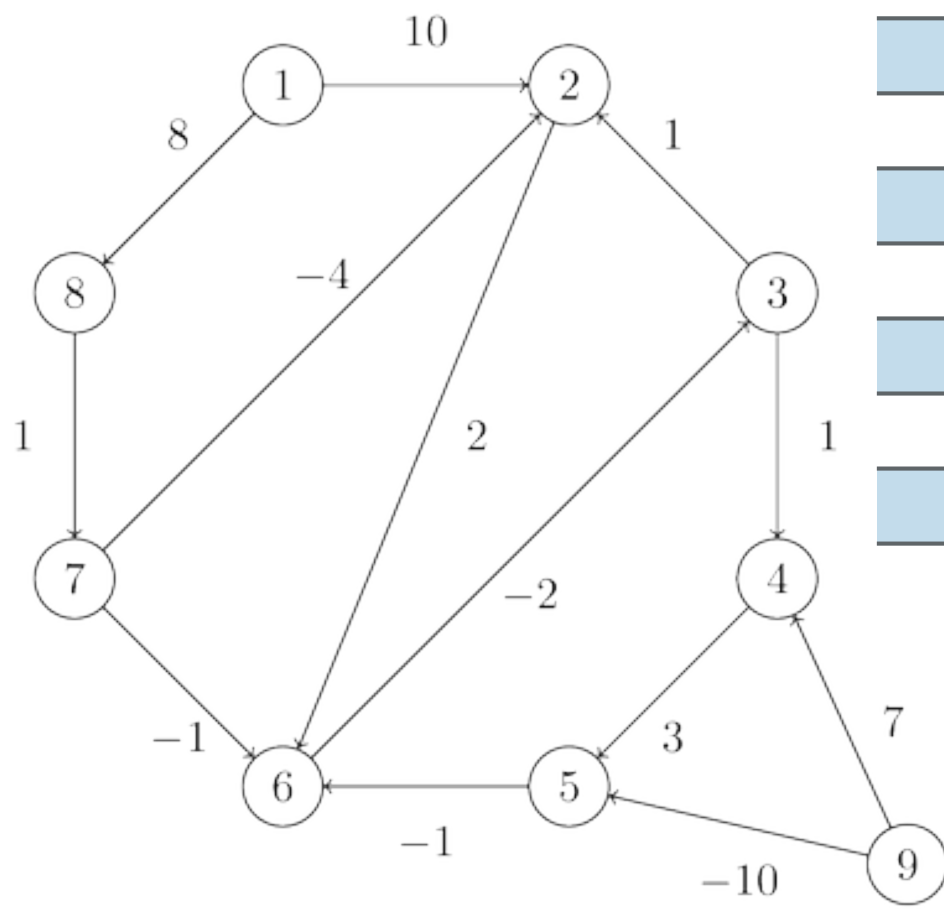


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	10	1–> 2
3	10	6–>3
4	11	3–>4
5	14	4–>5
6	12	2–>6
7	9	8–>7
8	8	1–>8
9	Inf	null

Practice Time - Pass 1

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

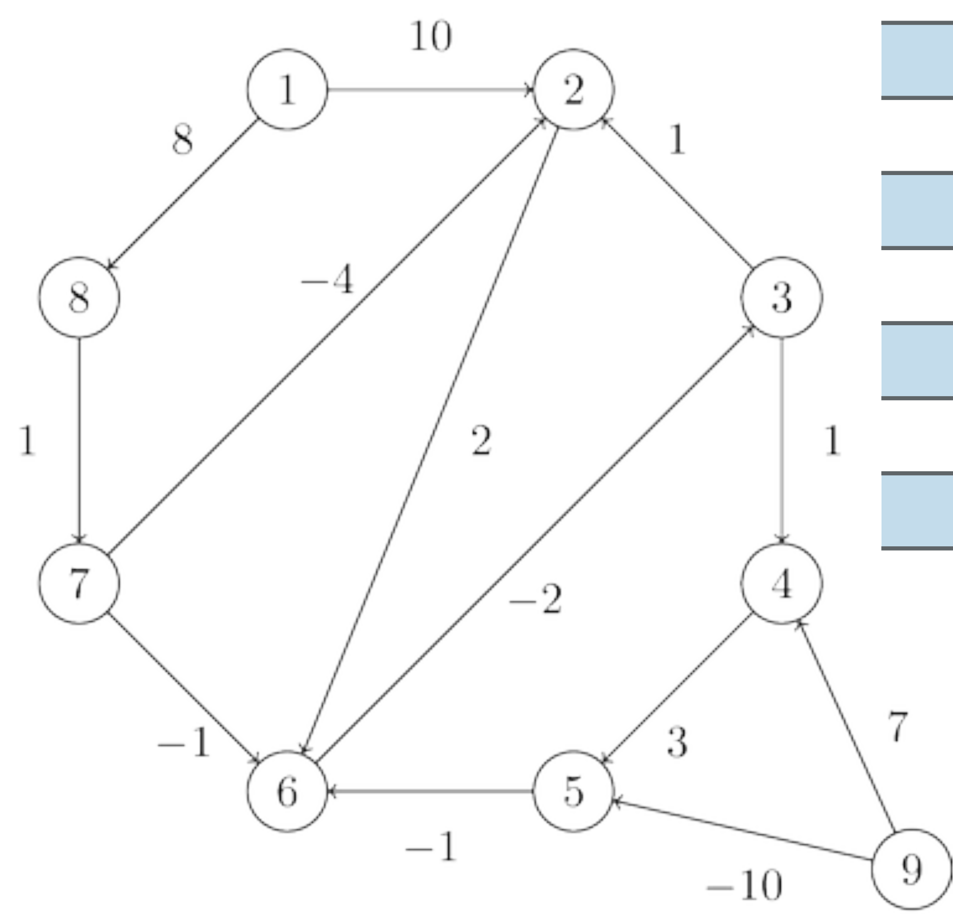


v	distTo[]	edgeTo[]
1	0	-
2	10	1–> 2
3	10	6–>3
4	11	3–>4
5	14	4–>5
6	12	2–>6
7	9	8–>7
8	8	1–>8
9	Inf	null

http://rosalind.info/problems/bf/

Practice Time - Pass 1

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

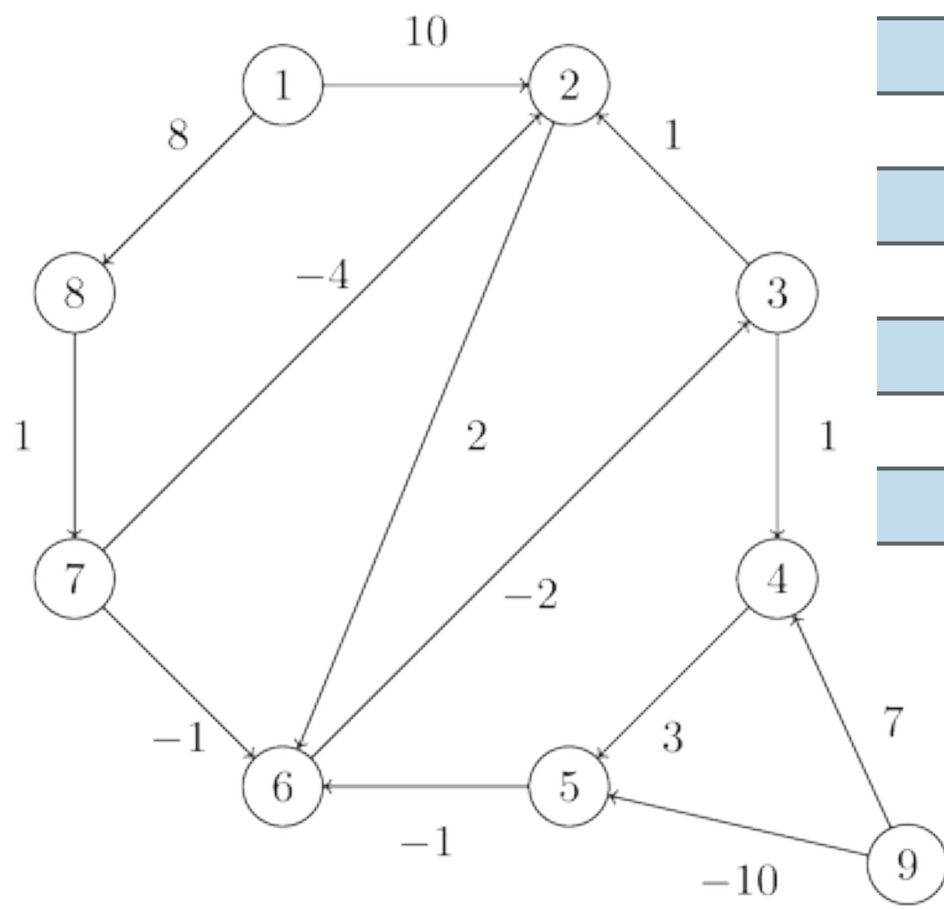


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	5	7–>2
3	10	6–>3
4	11	3–>4
5	14	4–>5
6	12	2–>6
7	9	8–>7
8	8	1–>8
9	Inf	null

Practice Time - Pass 1

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

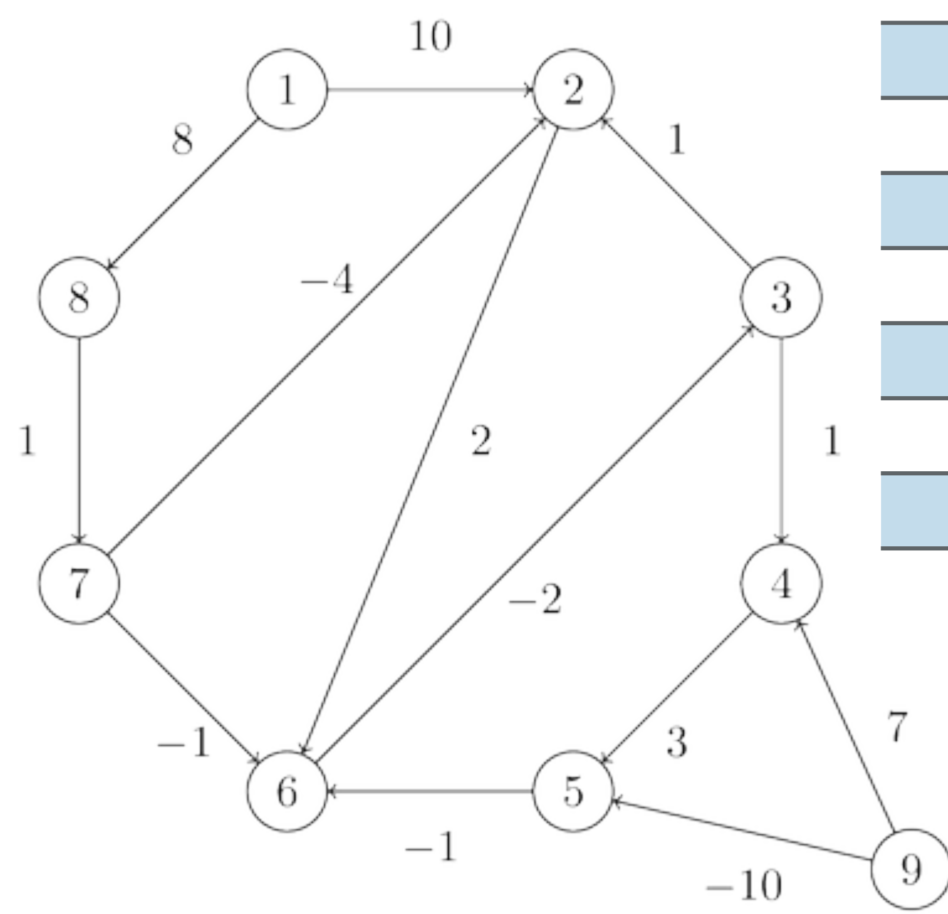


v	distTo[]	edgeTo[]
1	0	-
2	5	7–>2
3	10	6–>3
4	11	3–>4
5	14	4–>5
6	7	2–>6
7	9	8–>7
8	8	1–>8
9	Inf	null

http://rosalind.info/problems/bf/

Practice Time - Pass 1

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

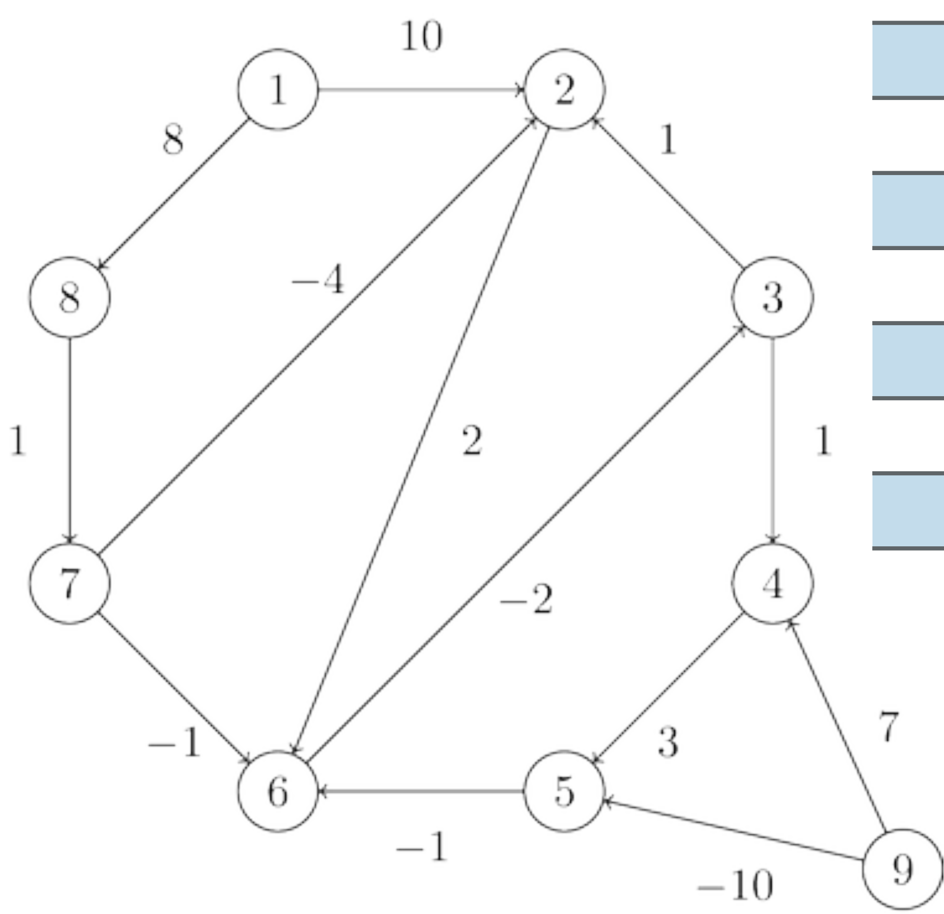


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	5	7–>2
3	5	6–>3
4	11	3–>4
5	14	4–>5
6	7	2–>6
7	9	8–>7
8	8	1–>8
9	Inf	null

Practice Time - Pass 1

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

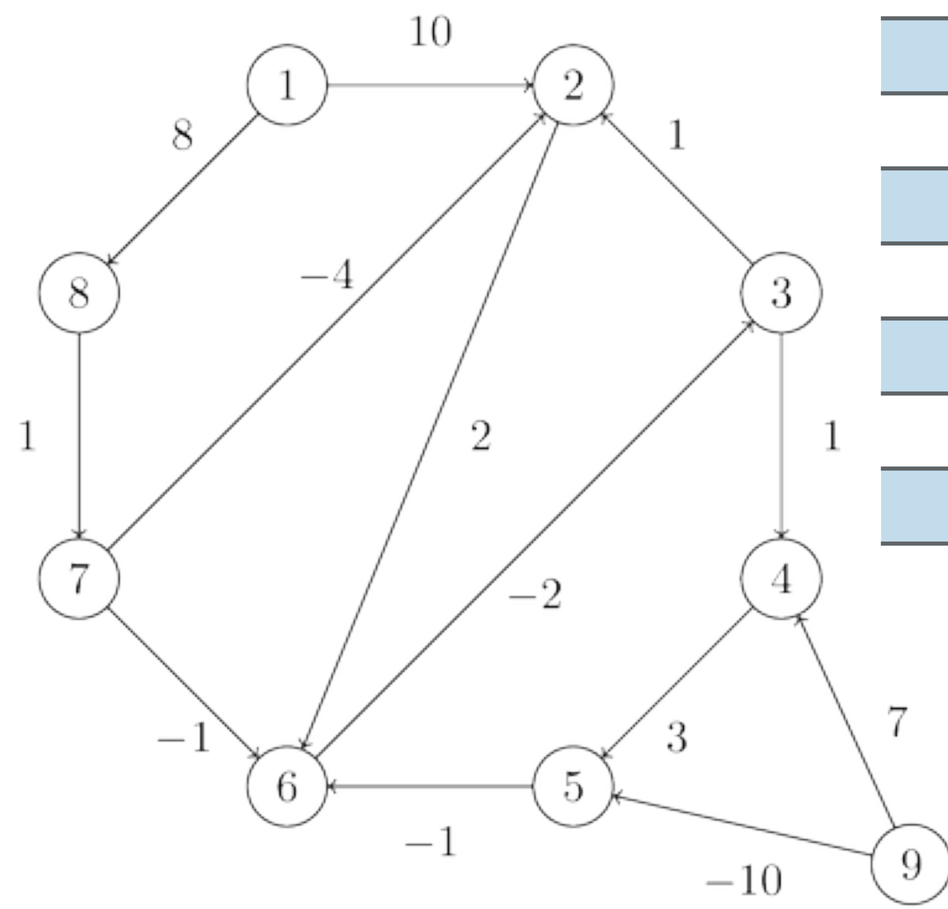


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	5	7–>2
3	5	6–>3
4	11	3–>4
5	14	4–>5
6	7	2–>6
7	9	8–>7
8	8	1–>8
9	Inf	null

Practice Time - Pass 1

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

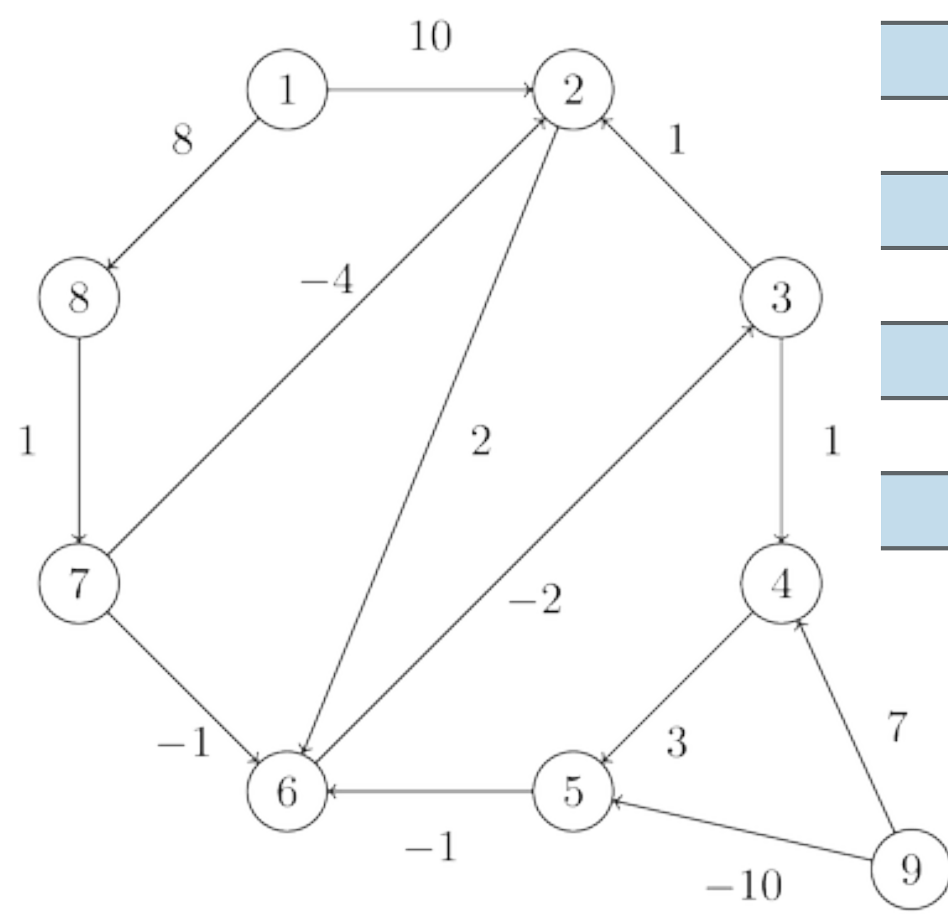


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	5	7–>2
3	5	6–>3
4	11	3–>4
5	14	4–>5
6	7	2–>6
7	9	8–>7
8	8	1–>8
9	Inf	null

Practice Time - Pass 2

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

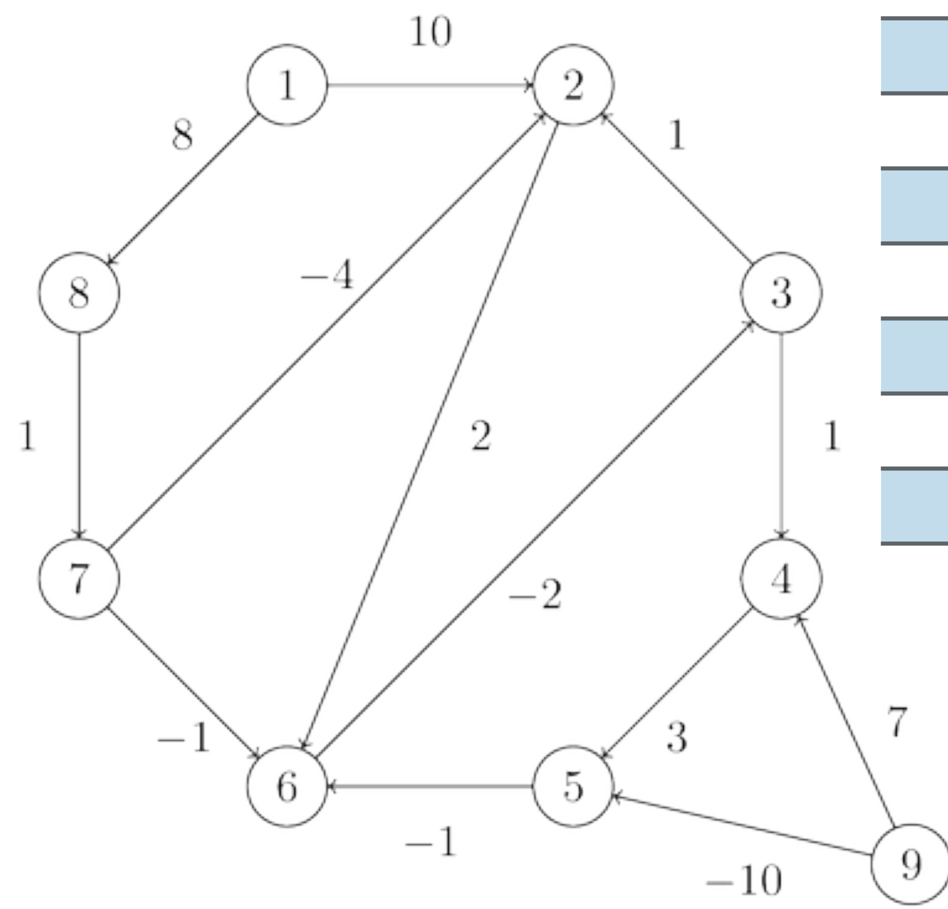


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	5	7–>2
3	5	6–>3
4	11	3–>4
5	14	4–>5
6	7	2–>6
7	9	8–>7
8	8	1–>8
9	Inf	null

Practice Time - Pass 2

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

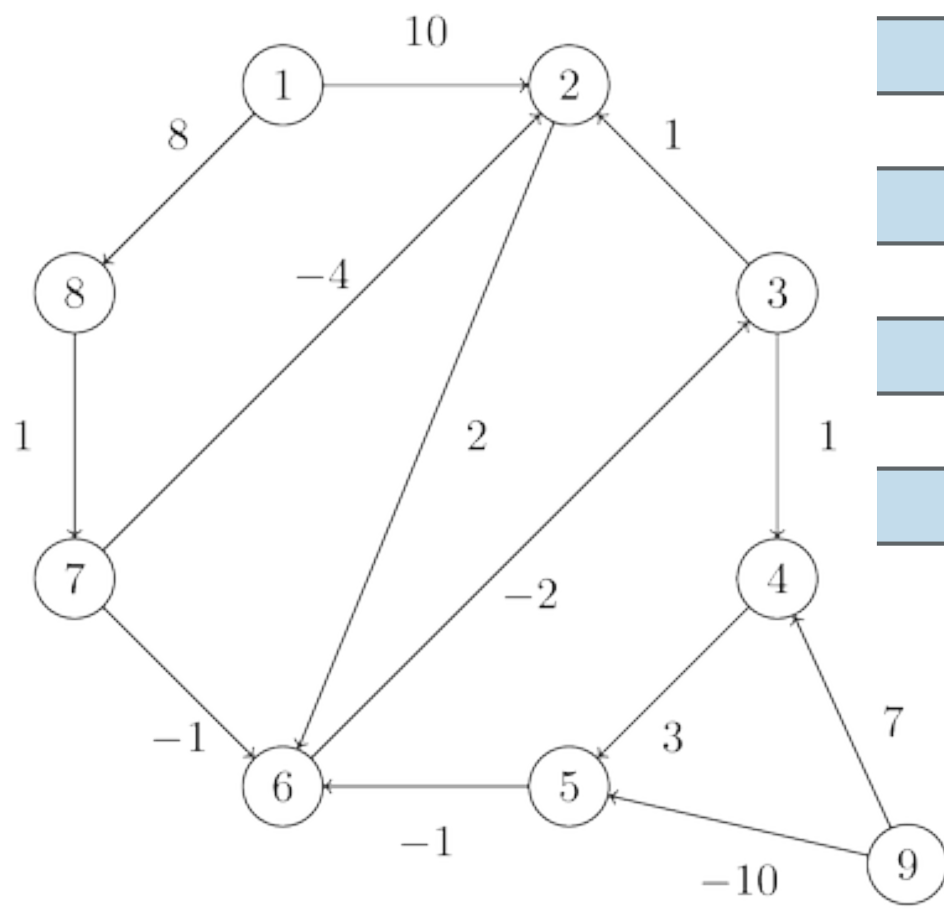


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	5	7–>2
3	5	6–>3
4	11	3–>4
5	14	4–>5
6	7	2–>6
7	9	8–>7
8	8	1–>8
9	Inf	null

Practice Time - Pass 2

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

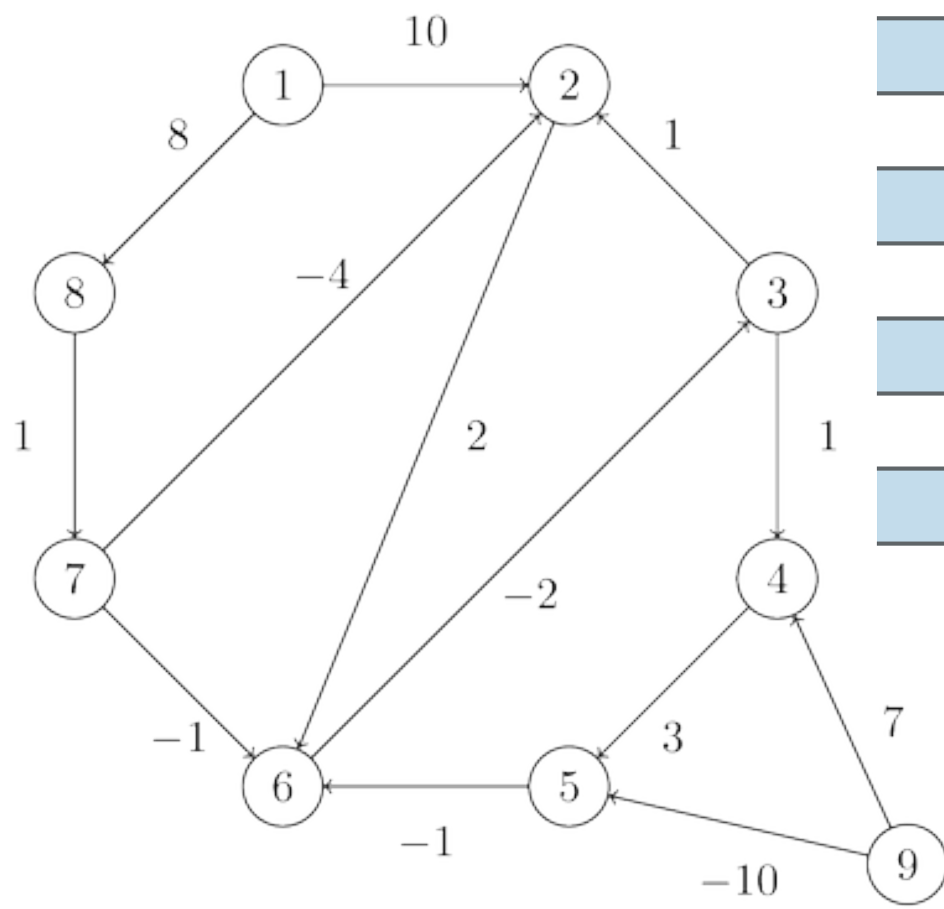


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	5	7–>2
3	5	6–>3
4	6	3–>4
5	14	4–>5
6	7	2–>6
7	9	8–>7
8	8	1–>8
9	Inf	null

Practice Time - Pass 2

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

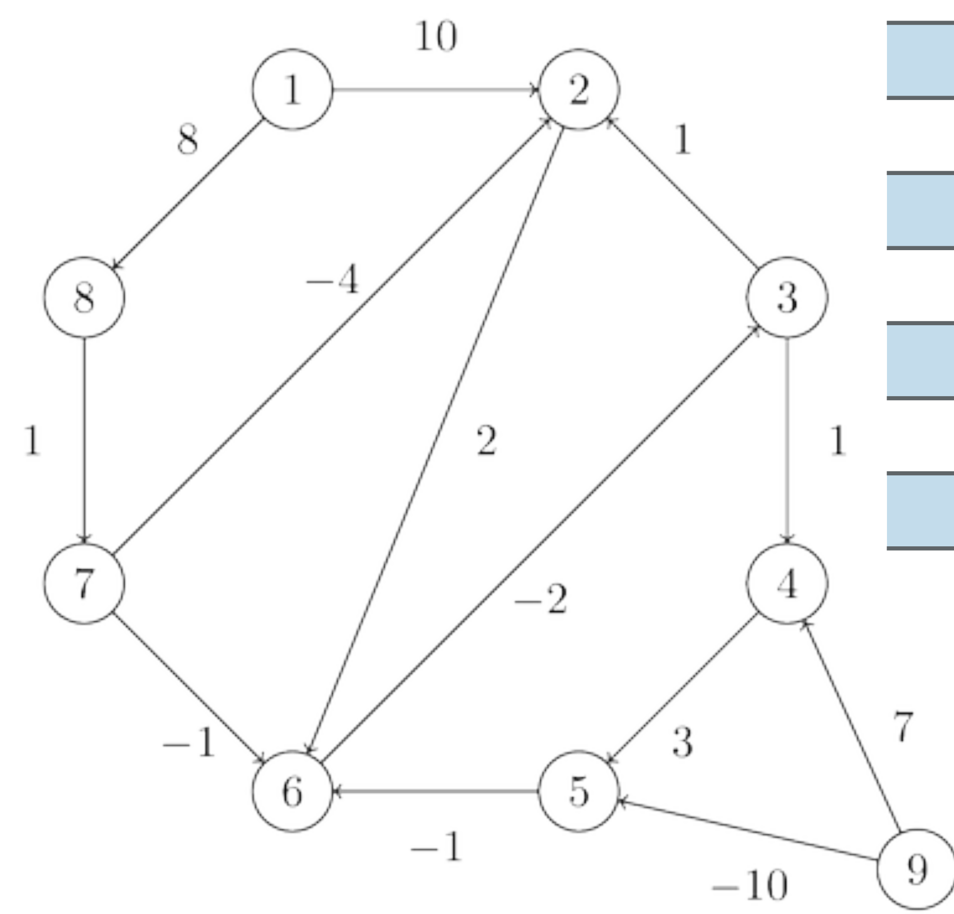


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	5	7–>2
3	5	6–>3
4	6	3–>4
5	9	4–>5
6	7	2–>6
7	9	8–>7
8	8	1–>8
9	Inf	null

Practice Time - Pass 2

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

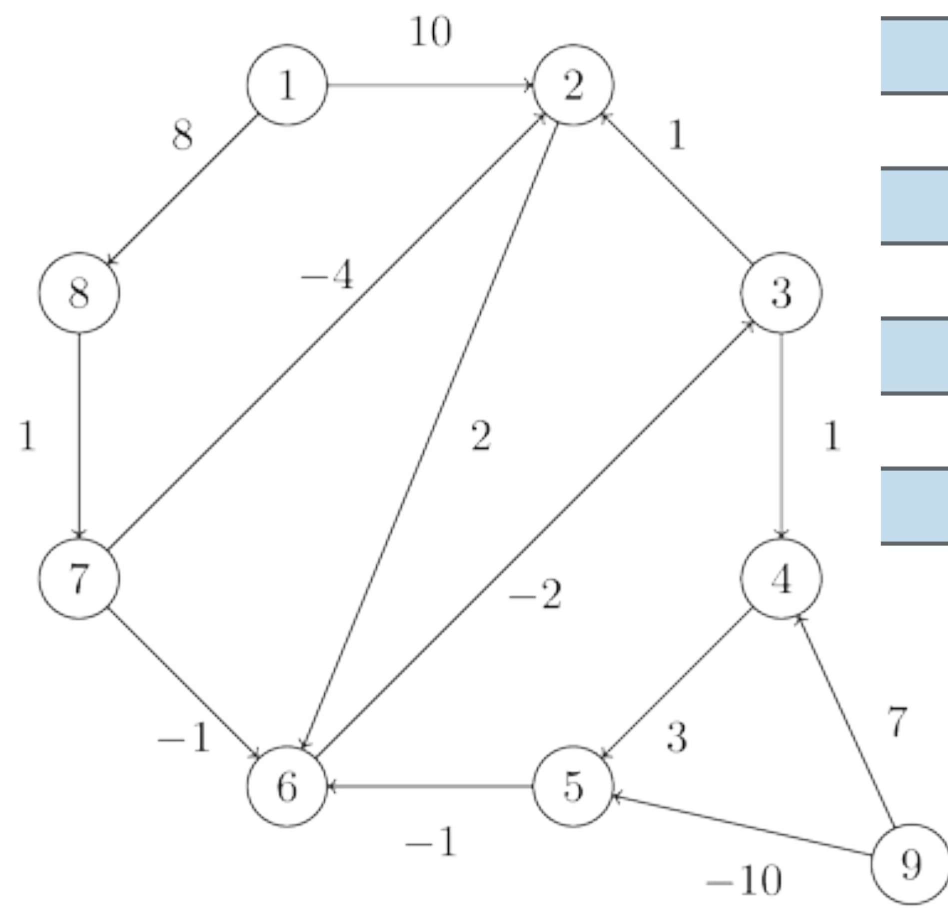


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	5	7–>2
3	5	6–>3
4	6	3–>4
5	9	4–>5
6	7	2–>6
7	9	8–>7
8	8	1–>8
9	Inf	null

Practice Time - Pass 2

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

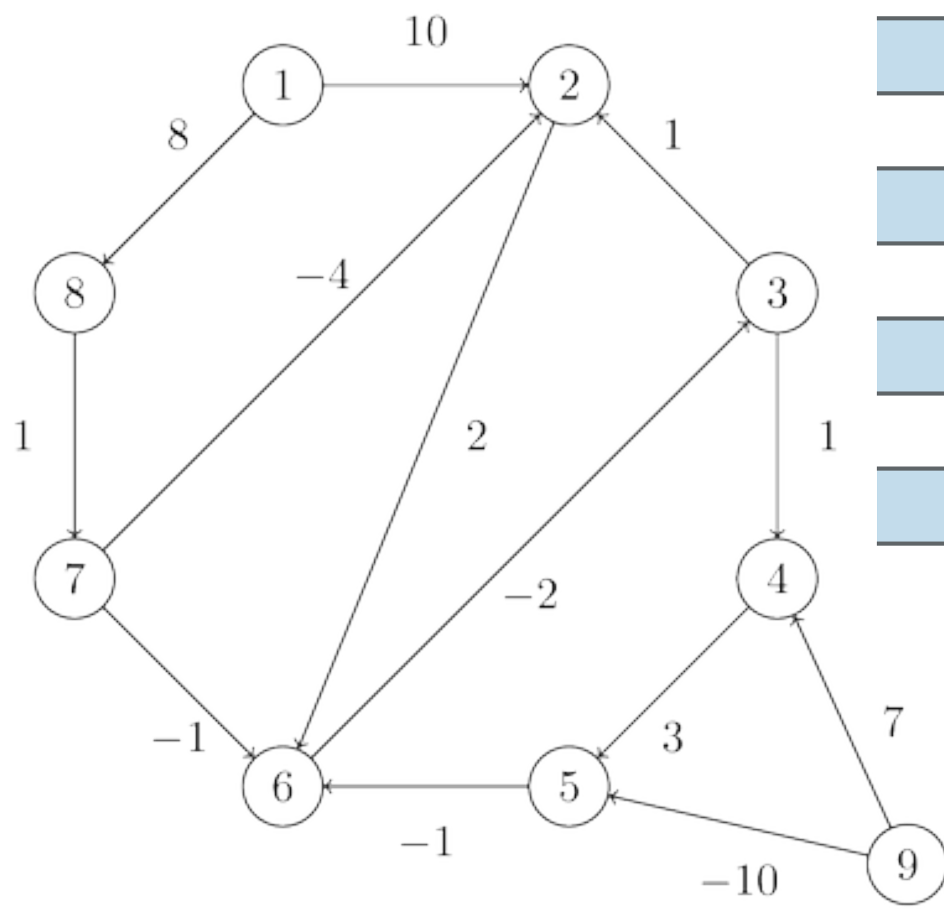


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	5	7–>2
3	5	6–>3
4	6	3–>4
5	9	4–>5
6	7	2–>6
7	9	8–>7
8	8	1–>8
9	Inf	null

Practice Time - Pass 2

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

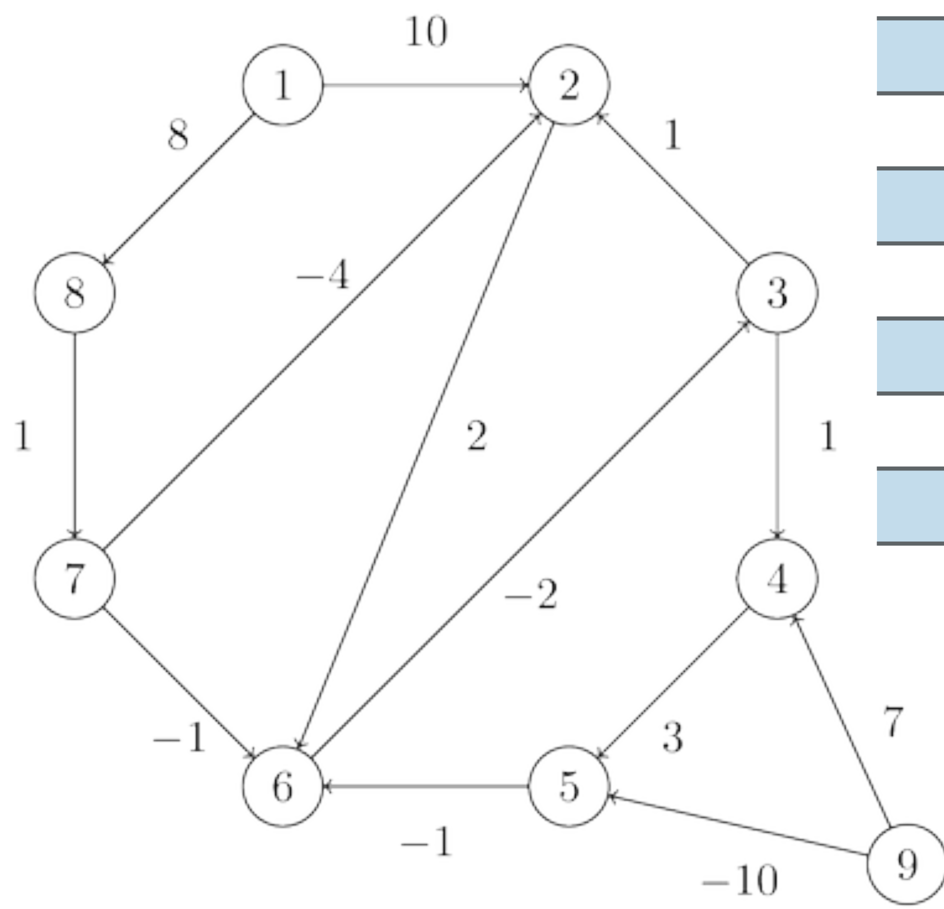


v	distTo[]	edgeTo[]
1	0	-
2	5	7–>2
3	5	6–>3
4	6	3–>4
5	9	4–>5
6	7	2–>6
7	9	8–>7
8	8	1–>8
9	Inf	null

http://rosalind.info/problems/bf/

Practice Time - Pass 2

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

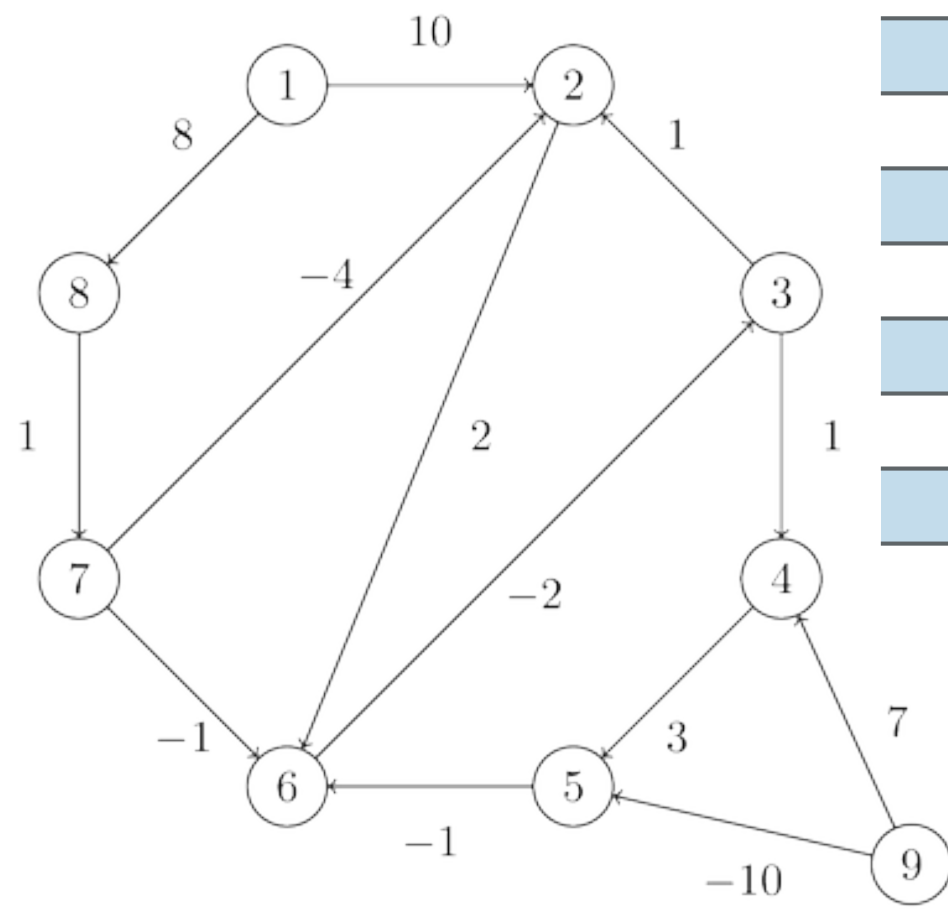


v	distTo[]	edgeTo[]
1	0	-
2	5	7–>2
3	5	6–>3
4	6	3–>4
5	9	4–>5
6	7	2–>6
7	9	8–>7
8	8	1–>8
9	Inf	null

http://rosalind.info/problems/bf/

Practice Time - Pass 2

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

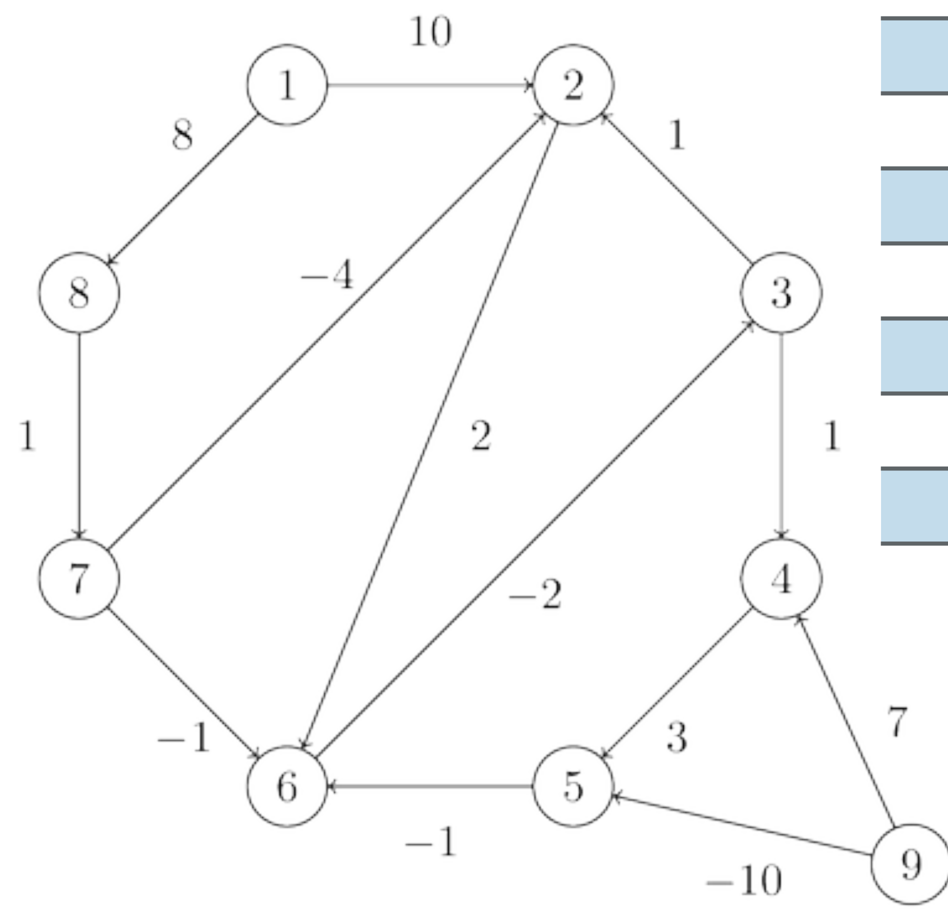


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	5	7–>2
3	5	6–>3
4	6	3–>4
5	9	4–>5
6	7	2–>6
7	9	8–>7
8	8	1–>8
9	Inf	null

Practice Time - Pass 2

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

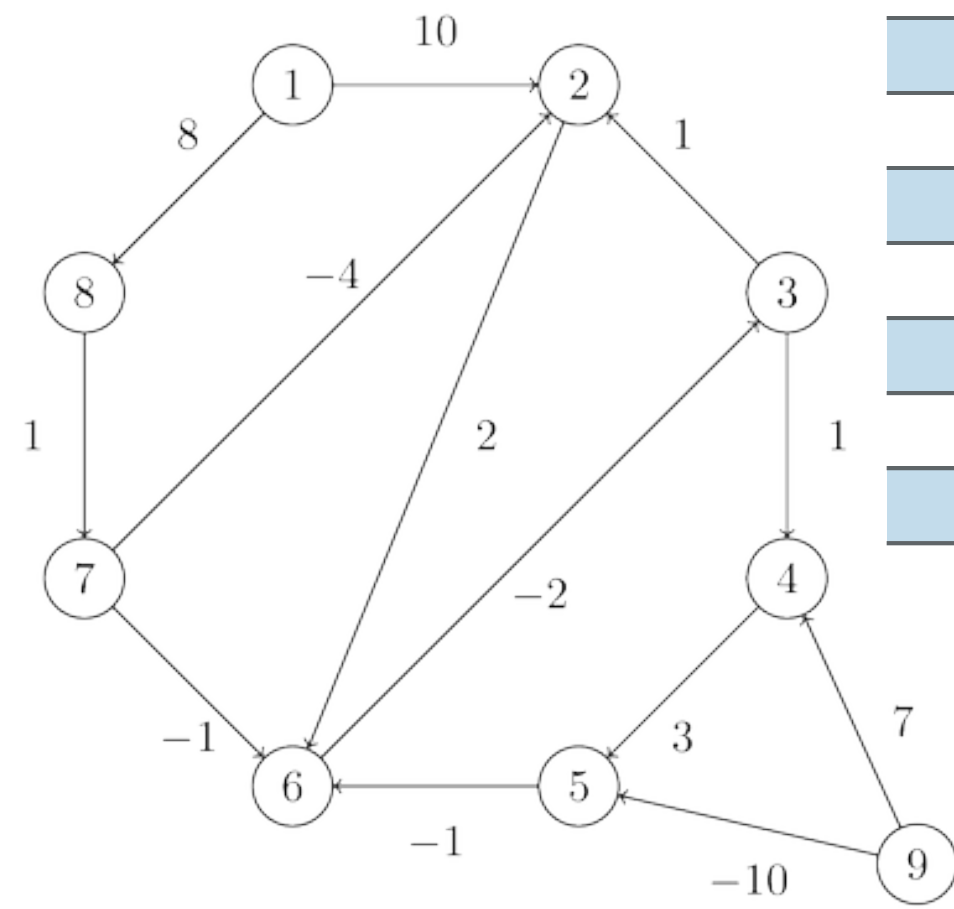


v	distTo[]	edgeTo[]
1	0	-
2	5	7–>2
3	5	6–>3
4	6	3–>4
5	9	4–>5
6	7	2–>6
7	9	8–>7
8	8	1–>8
9	Inf	null

http://rosalind.info/problems/bf/

Practice Time - Pass 2

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

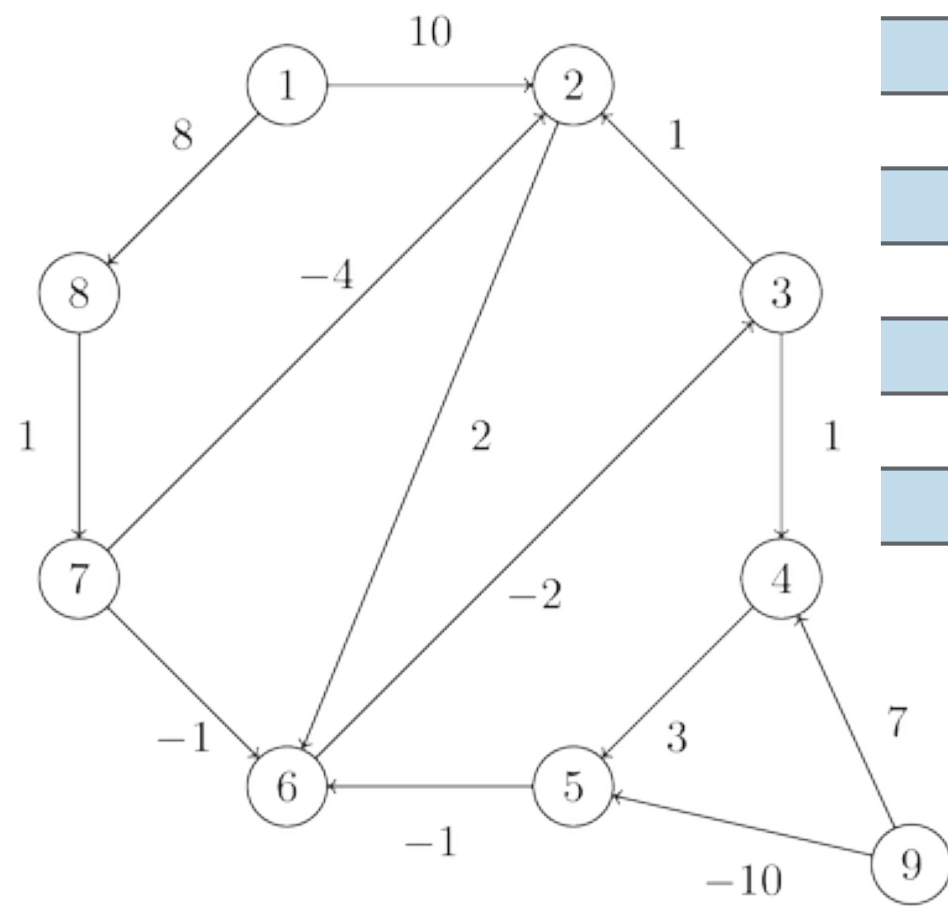


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	5	7–>2
3	5	6–>3
4	6	3–>4
5	9	4–>5
6	7	2–>6
7	9	8–>7
8	8	1–>8
9	Inf	null

Practice Time - Pass 2

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

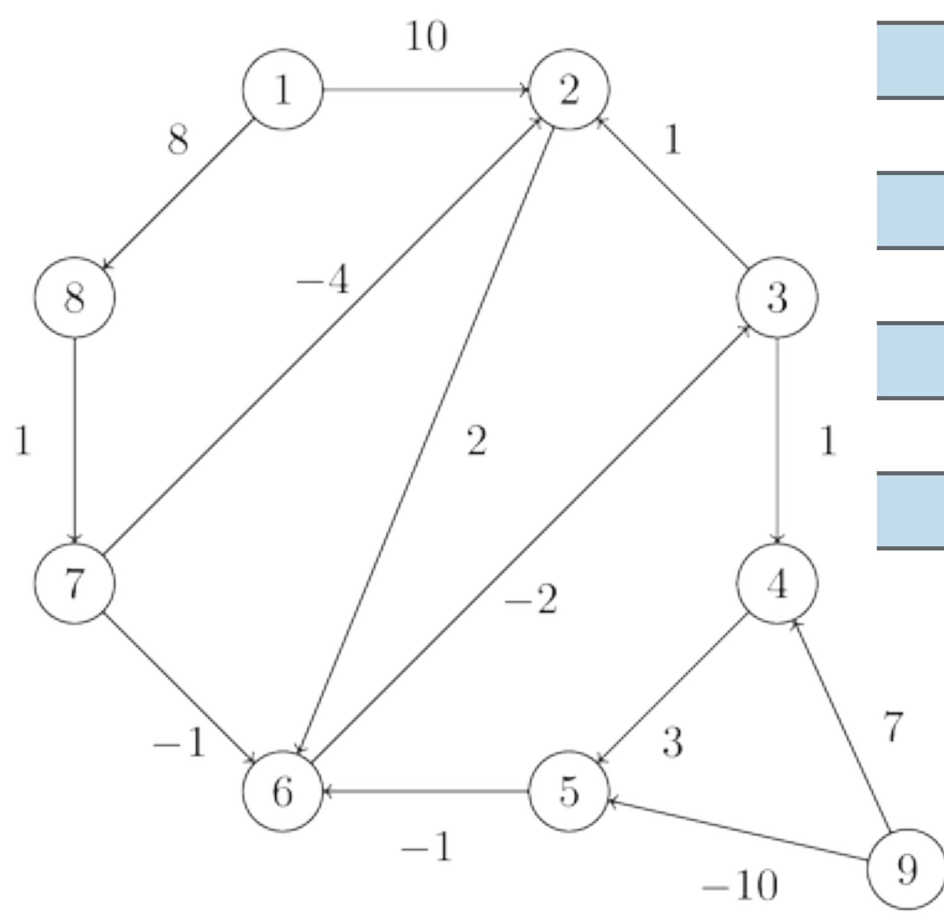


v	distTo[]	edgeTo[]
1	0	-
2	5	7–>2
3	5	6–>3
4	6	3–>4
5	9	4–>5
6	7	2–>6
7	9	8–>7
8	8	1–>8
9	Inf	null

http://rosalind.info/problems/bf/

Practice Time - Pass 2

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

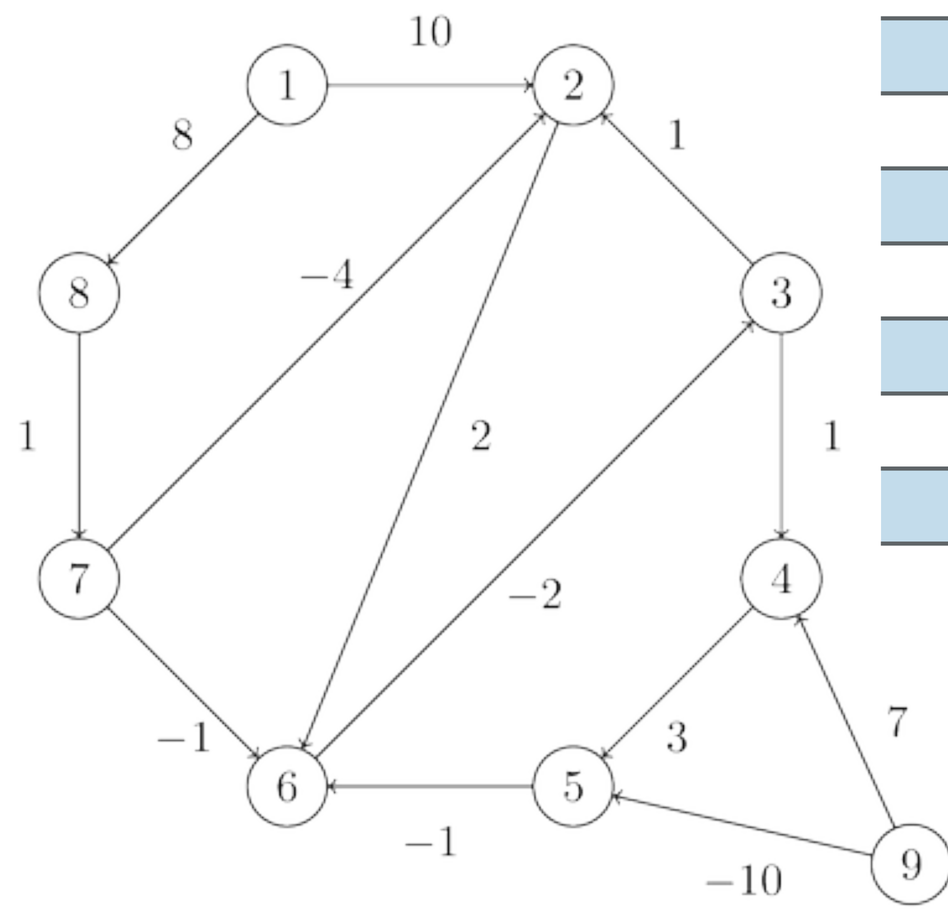


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	5	7–>2
3	5	6–>3
4	6	3–>4
5	9	4–>5
6	7	2–>6
7	9	8–>7
8	8	1–>8
9	Inf	null

Practice Time - No changes in passes 3-8

1 2 10
3 2 1
3 4 1
4 5 3
5 6 -1
7 6 -1
8 7 1
1 8 8
7 2 -4
2 6 2
6 3 -2
9 5 -10
9 4 7

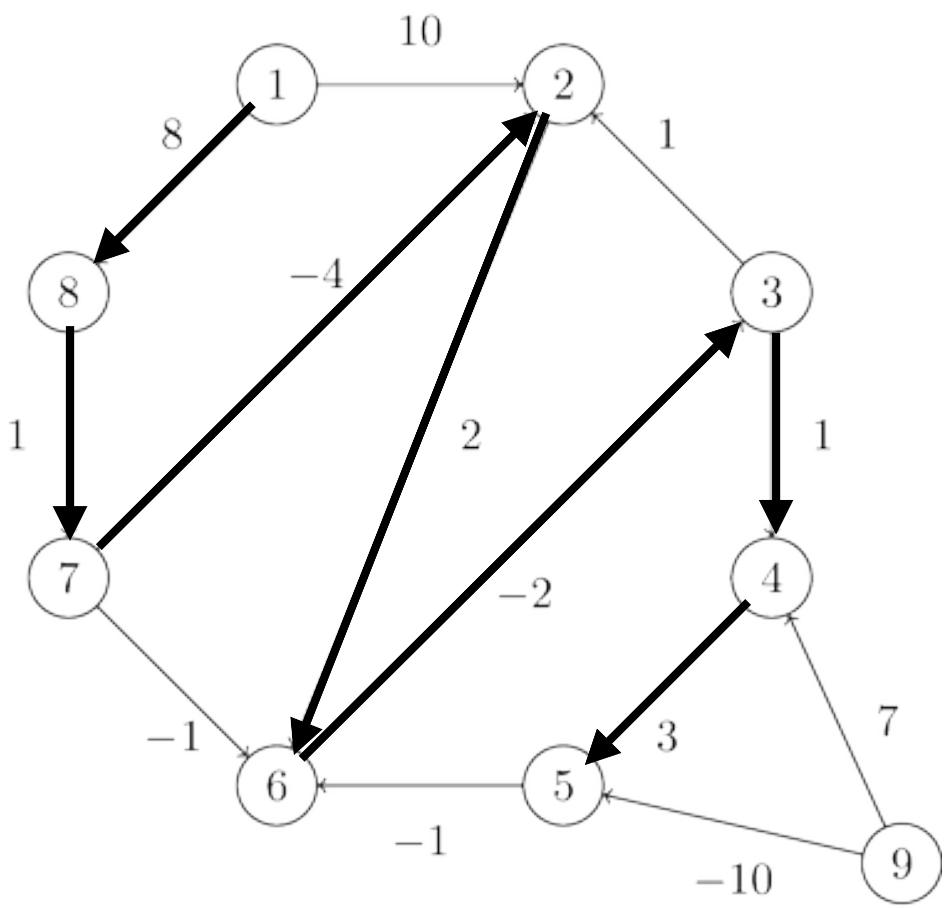


http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	5	7–>2
3	5	6–>3
4	6	3–>4
5	9	4–>5
6	7	2–>6
7	9	8–>7
8	8	1–>8
9	Inf	null

Answer

1	2	10
3	2	1
3	4	1
4	5	3
5	6	-1
7	6	-1
8	7	1
1	8	8
7	2	-4
2	6	2
6	3	-2
9	5	-10
9	4	7



http://rosalind.info/problems/bf/

v	distTo[]	edgeTo[]
1	0	-
2	5	7->2
3	5	6->3
4	6	3->4
5	9	4->5
6	7	2->6
7	9	8->7
8	8	1->8
9	Inf	null

Lecture 27: Shortest Paths

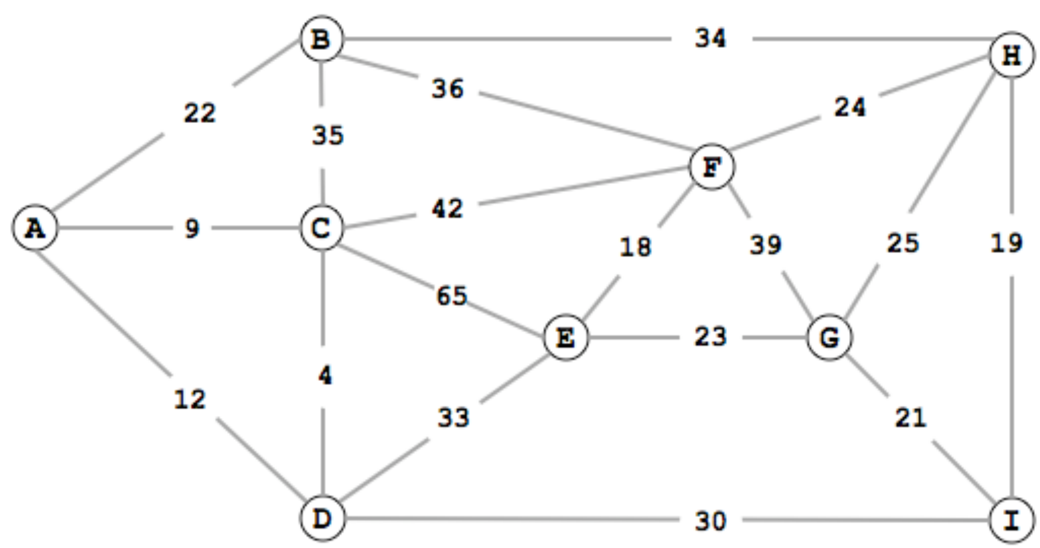
- Introduction to Shortest Paths
- API
- Properties
- Dijkstra's Algorithm
- Belman-Ford Algorithm

Readings:

- Textbook: Chapter 4.4 (Pages 638-676)
- Website:
  - https://algs4.cs.princeton.edu/44sp/

Practice Problems:

Run Dijkstra's algorithm on the graph on the right with A being the starting vertex.



Practice Problems:

Run the Bellman-Ford algorithm on this directed graph using vertex z as the source. In each pass show the values d and pi. In the graph, V = (s, t, v, x, z} and the weighted, directed edges are E = {(s, t, 6), (s, v, 7), (t, v, 8), (t, z, -4), (t, x, 5), (v, x, -3), (v, z, 9), (x, t, -2), (z, s, 2), (z, x, 4)}.

https://www.chegg.com/homework-help/questions-and-answers/run-bellman-ford-algorithm-following-directed-graph-using-vertex-z-source-pass-show-values-q17182493