CSC 226

Algorithms and Data Structures: II

Rich Little
rlittle@uvic.ca
ECS 516

Two basic properties for minimum spanning trees

- Cycle property
- Cut property

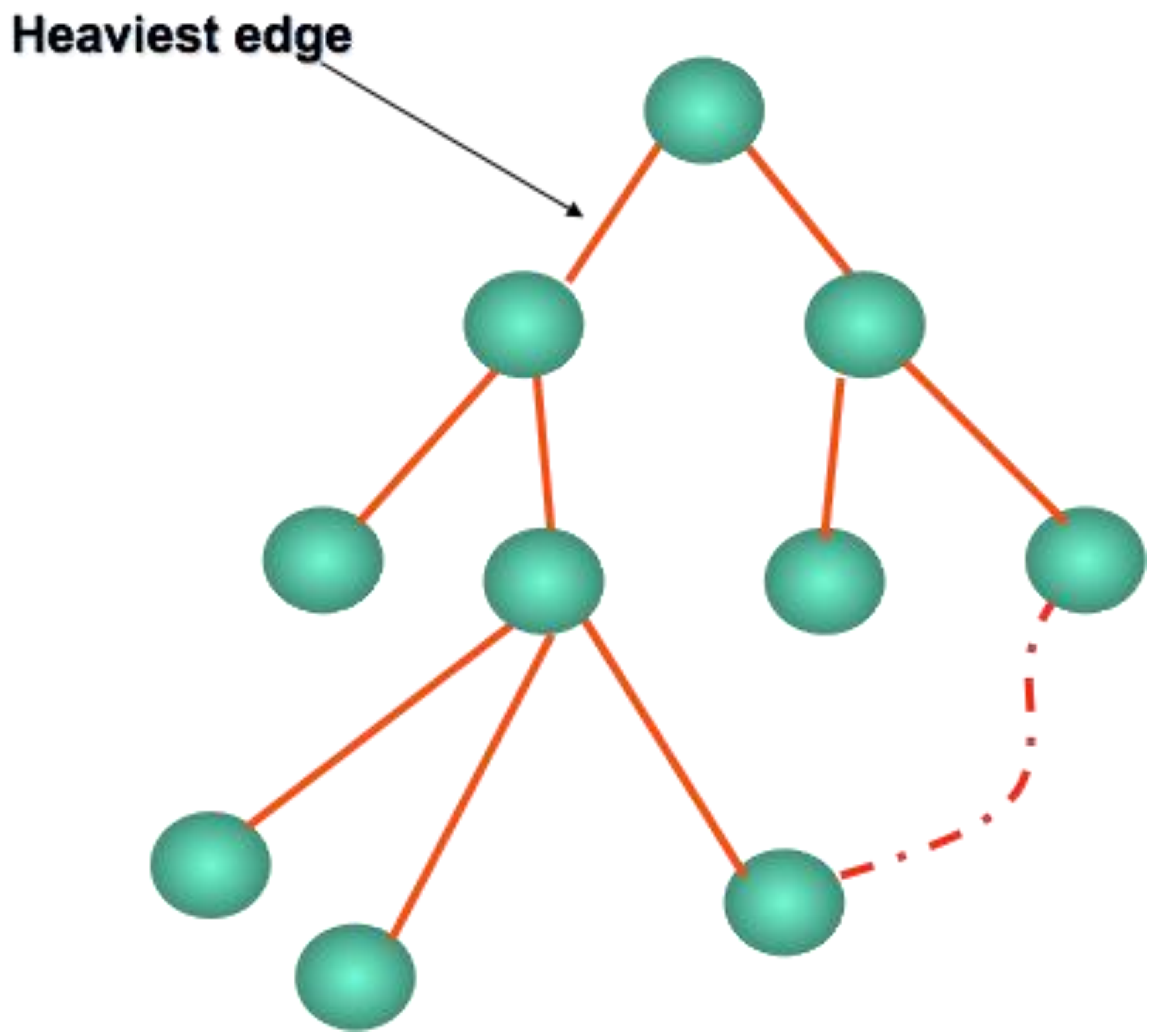
Cycle property

- Let C be any cycle in weighted graph G with distinct edge weights. Let e be the heaviest edge in the cycle.
- Then the minimum spanning tree for G does not contain e.

Proof. (Cycle property)

- Assume that all edges in the graph are of distinct weight
- We proof by contradiction: the MST T for G does not contain edge e
- Assume e does belong to MST T. Then deleting e from T disconnects T into two trees, T_1 and T_2.
- Consider cycle C. C consists of some vertices that belong to T_1 and the other vertices of C belong to T_2.
- There is an edge in C, say f, that connects a vertex from T_1 to a vertex T_2.
- Merge T_1 and T_2 using f to spanning tree T^*. The new tree, T^*, is lighter than T. A contradiction.

Cycle property



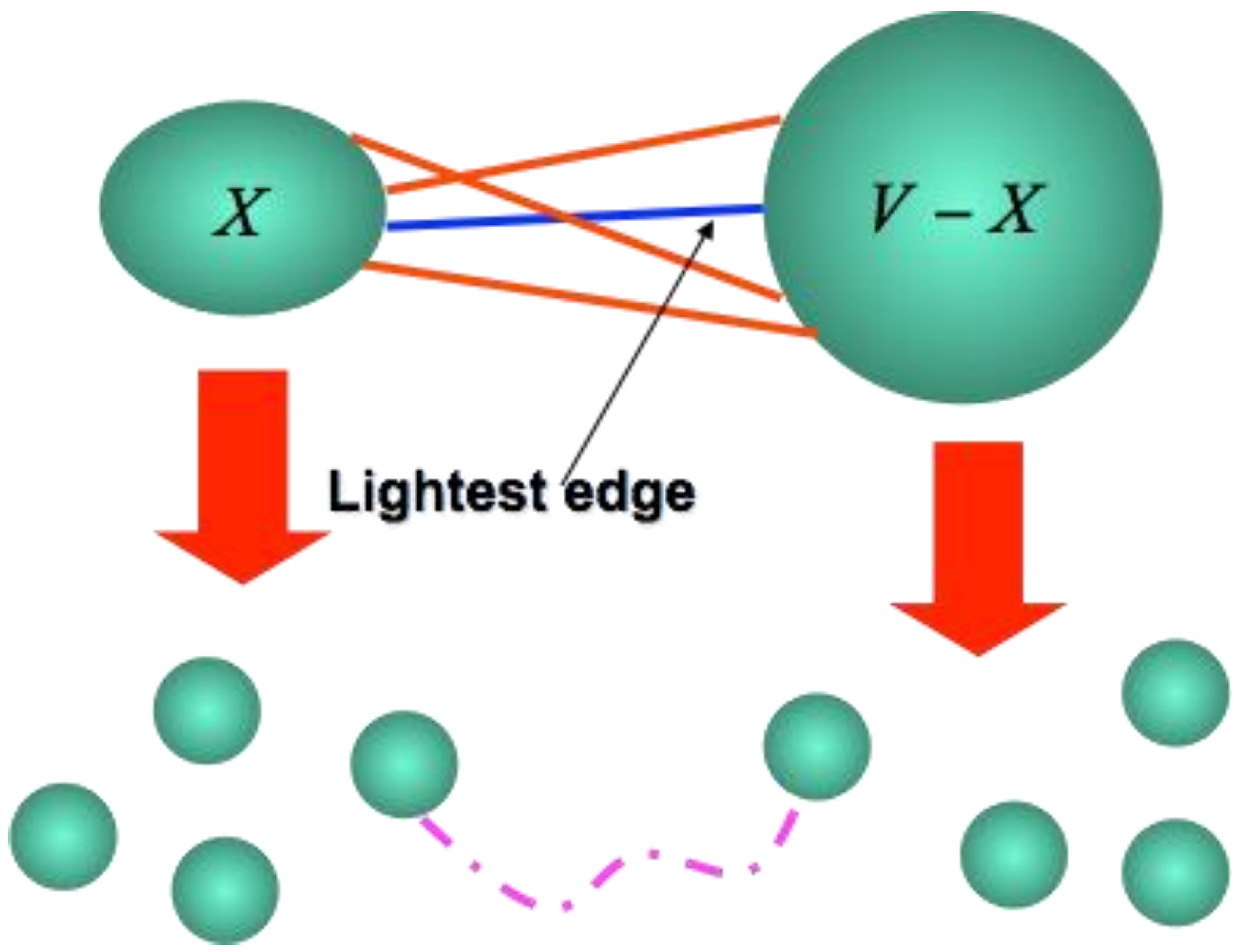
Cut Property

- Let V' be any proper subset of vertices in weighted graph $G = (V, E)$, and let e be the lightest edge that has exactly one endpoint in V'
- Then the minimum spanning tree T for G contains e.

Proof (Cut property)

- Assume that all edges in the graph are of distinct weight
- We prove by contradiction: MST T for G contains edge e
- Assume it does not
- Add e to T creating cycle C
- Consider edge f in C that has exactly one endpoint in V'
- Create spanning tree T^* by replacing e with f, but T^* is lighter than T. Contradiction.

Cut Property



Prim's Algorithm Correctness

Cut property

- Initialize tree with single chosen vertex
- Grow tree by finding lightest edge not yet in tree and expanding the tree, and connect it to tree; repeat until all vertices are in the tree
- *Example of greedy algorithm*

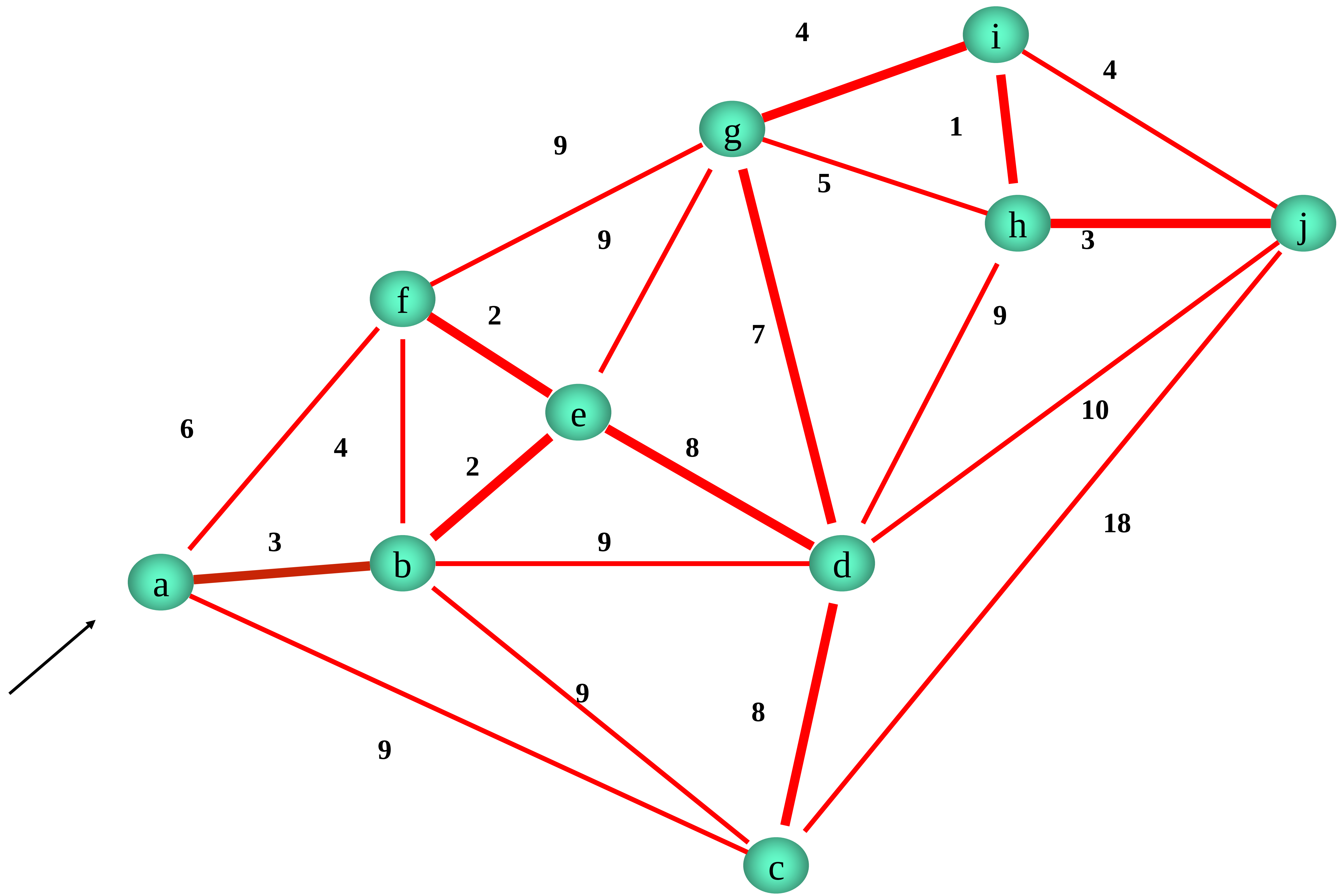
Pseudocode: Prim's Algorithm

Algorithm Prim-Jarník-Dijkstra
Input: a weighted connected graph $G = (V, E)$
Output: an MST T for G
Data structure: array D; Priority Queue PQ; and tree T

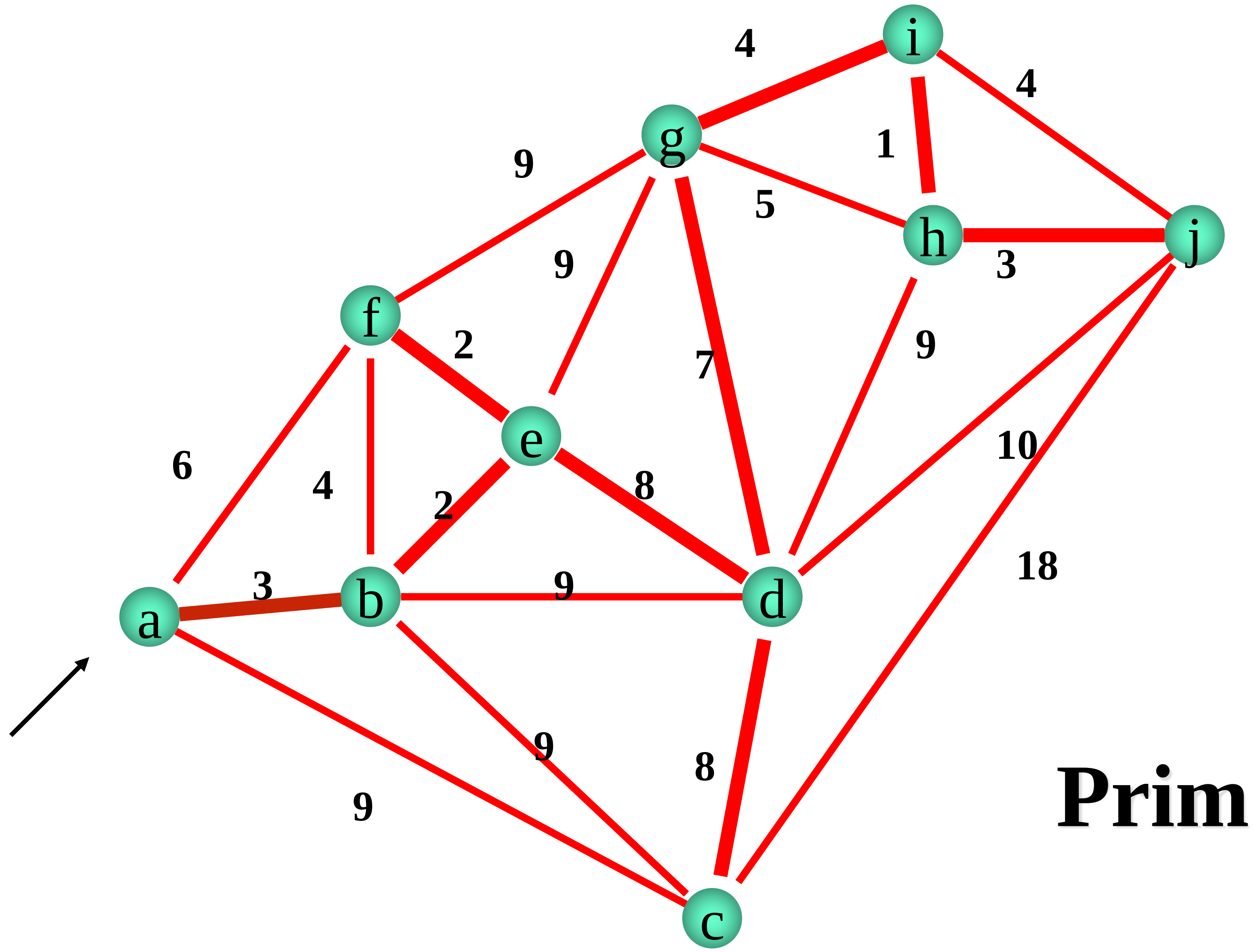
pick an arbitrary vertex v in G; $D[v] \leftarrow 0$
for each vertex $u \neq v$ **do** $D[u] \leftarrow +\infty$ **end**
$T \leftarrow \emptyset$
for each vertex u **do** PQ.insert({(u, (null), $D[u]$}) **end** // including v
// for each vertex u, (u, edge) is the element and $D[u]$ is the key in PQ
while not PQ.empty() **do**
  $(u, e) \leftarrow PQ$.deleteMin()
  add vertex u and edge e to T
  for each vertex z adjacent to u such that z is in PQ **do**
    if weight((u, z)) $< D[z]$ **then**
      $D[z] \leftarrow$ weight((u, z))
      in PQ, change element and key of z to $\{z, (u, z), D[z]\}$
      update PQ
    end
  end
end
return T

D :distance vector, maintains reachable vertices

PQ :a priority queue for the edges according to values in D



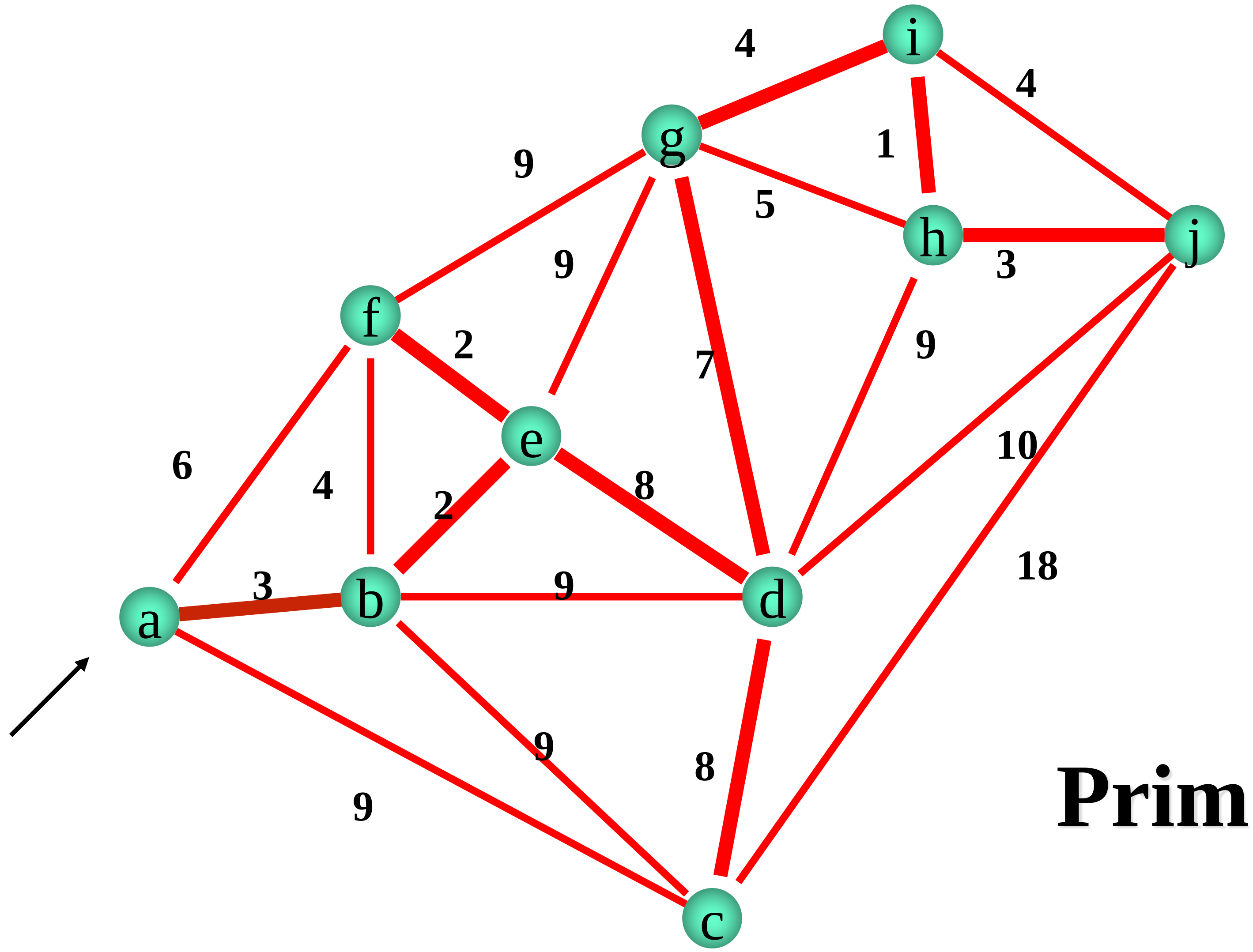
a
b
c
d
e
f
g
h
i
j
3
6
9
4
2
2
9
9
8
7
9
5
4
1
4
3
9
10
18
8
9



Prim-Jarník-Dijkstra

D

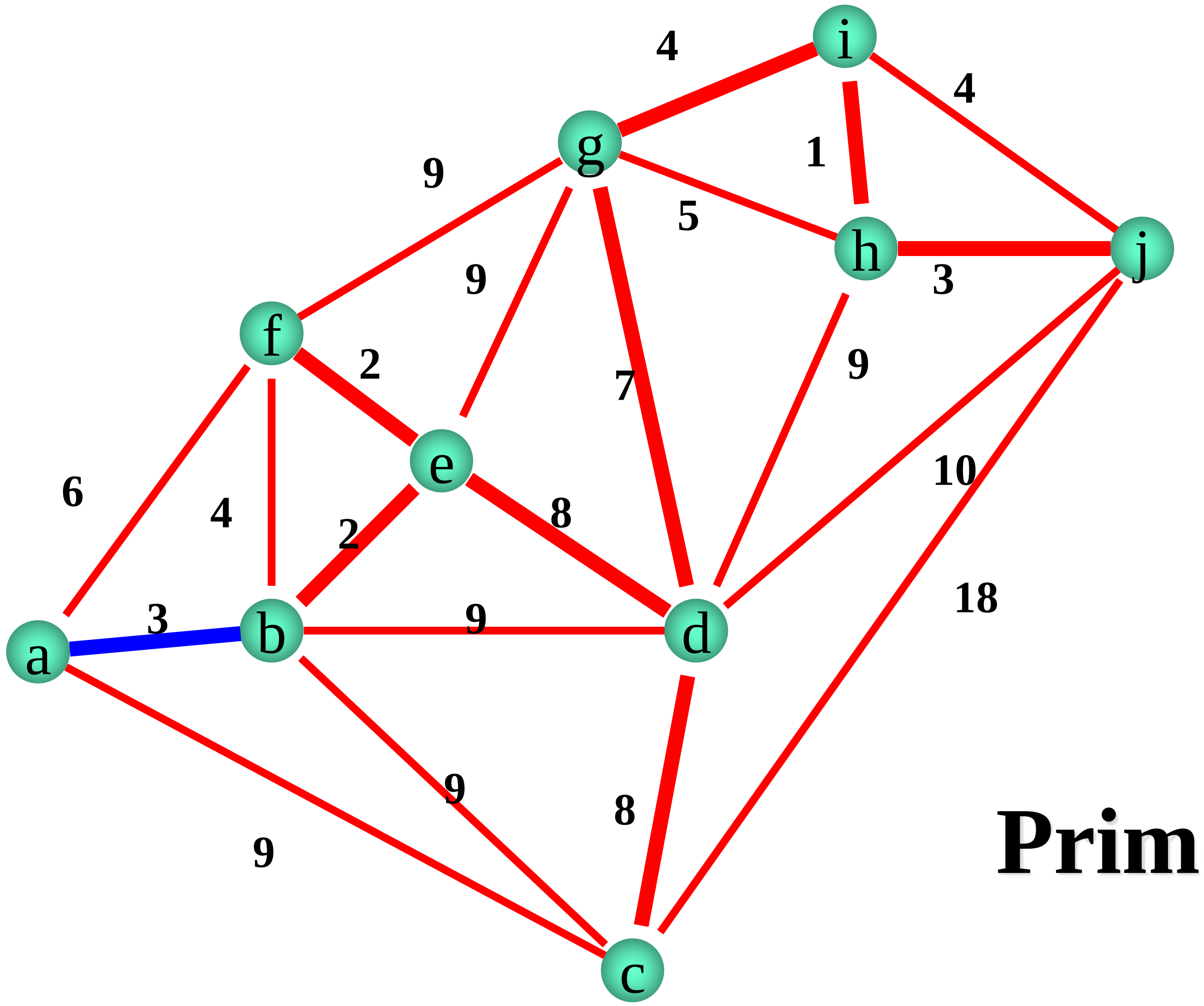
a	b	c	d	e	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞



Prim-Jarník-Dijkstra

D

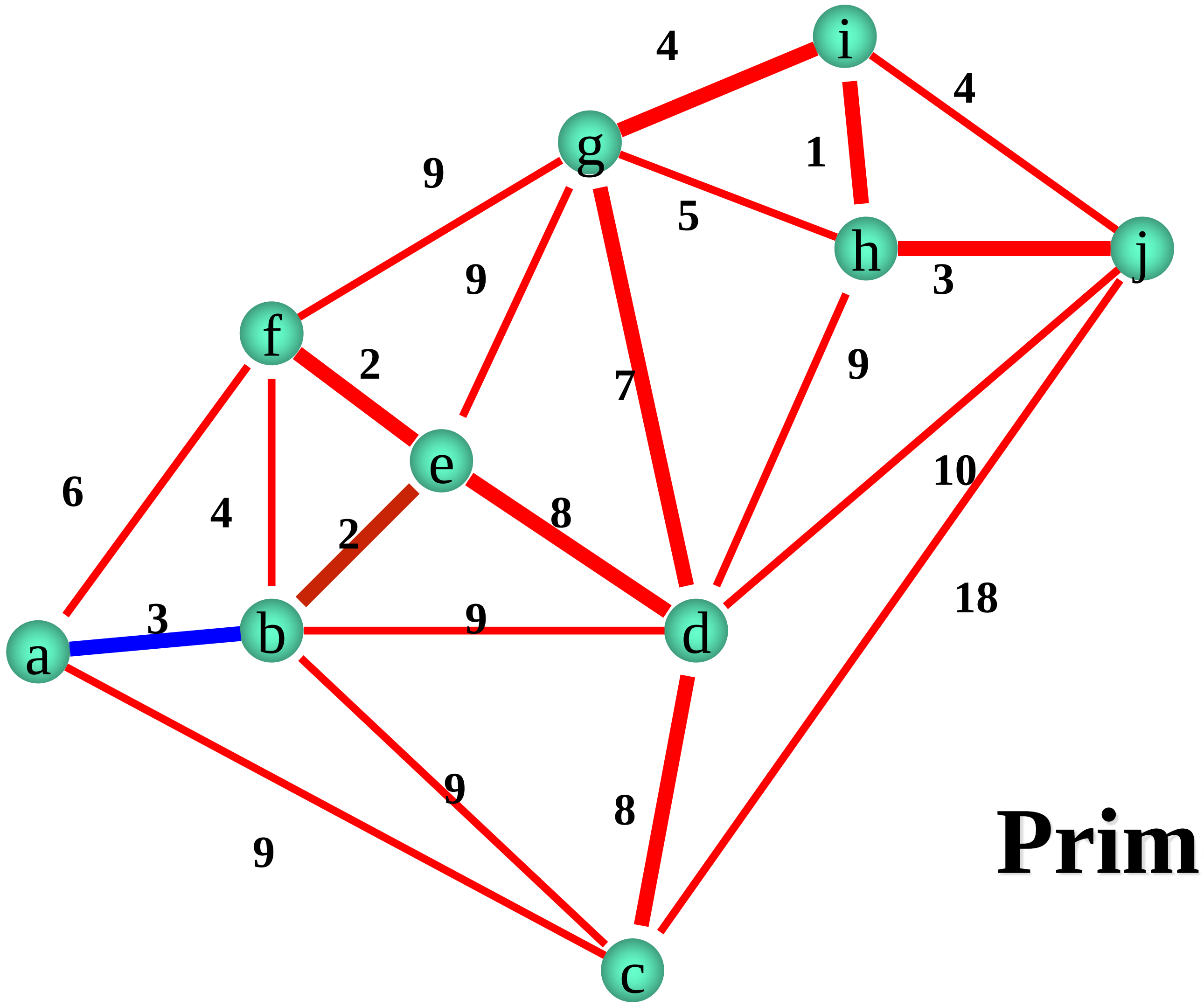
a	b	c	d	e	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞



Prim-Jarník-Dijkstra

D

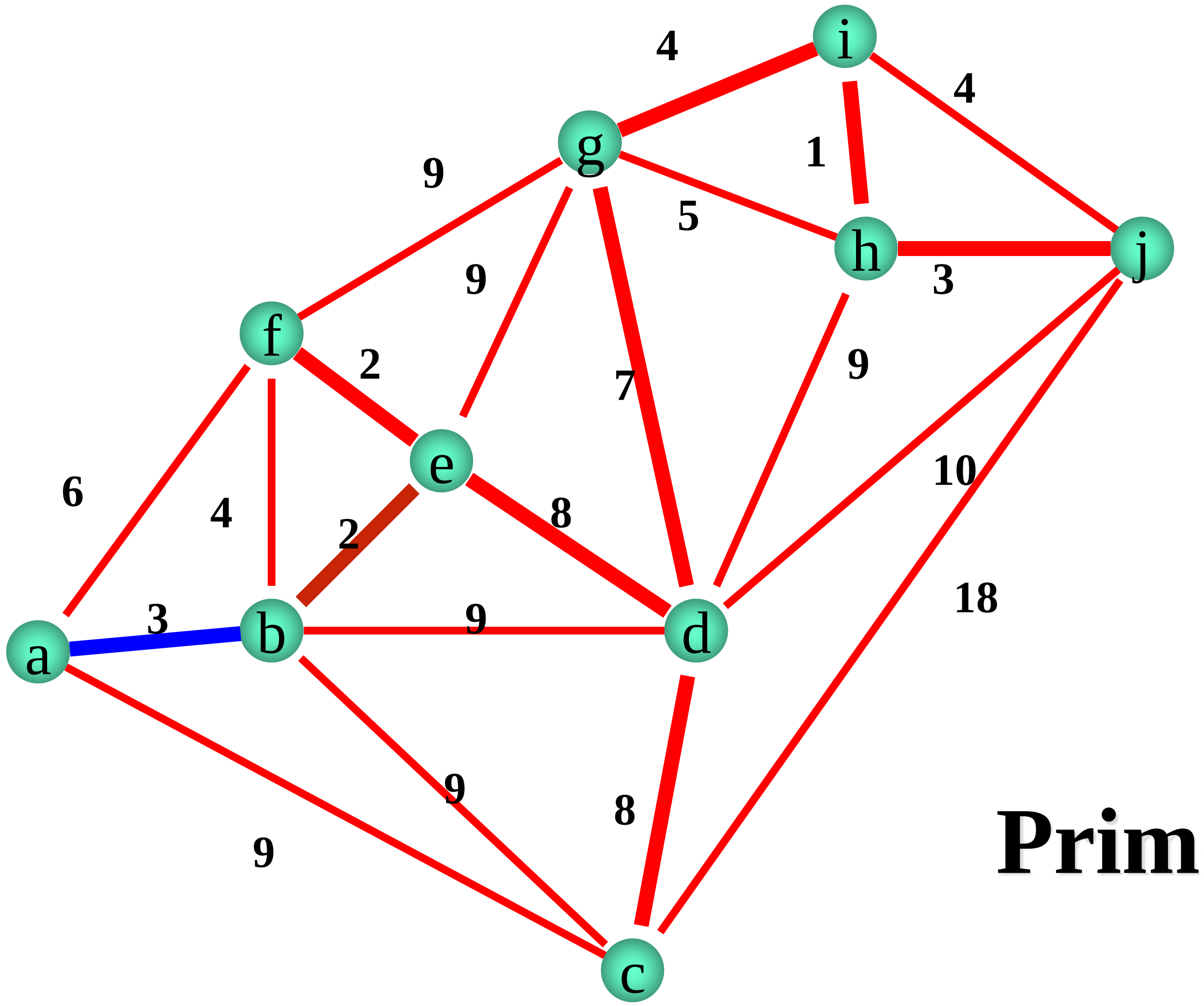
a	b	c	d	e	f	g	h	i	j
0	3	9	$+\infty$	$+\infty$	6	$+\infty$	$+\infty$	$+\infty$	$+\infty$



Prim-Jarník-Dijkstra

D

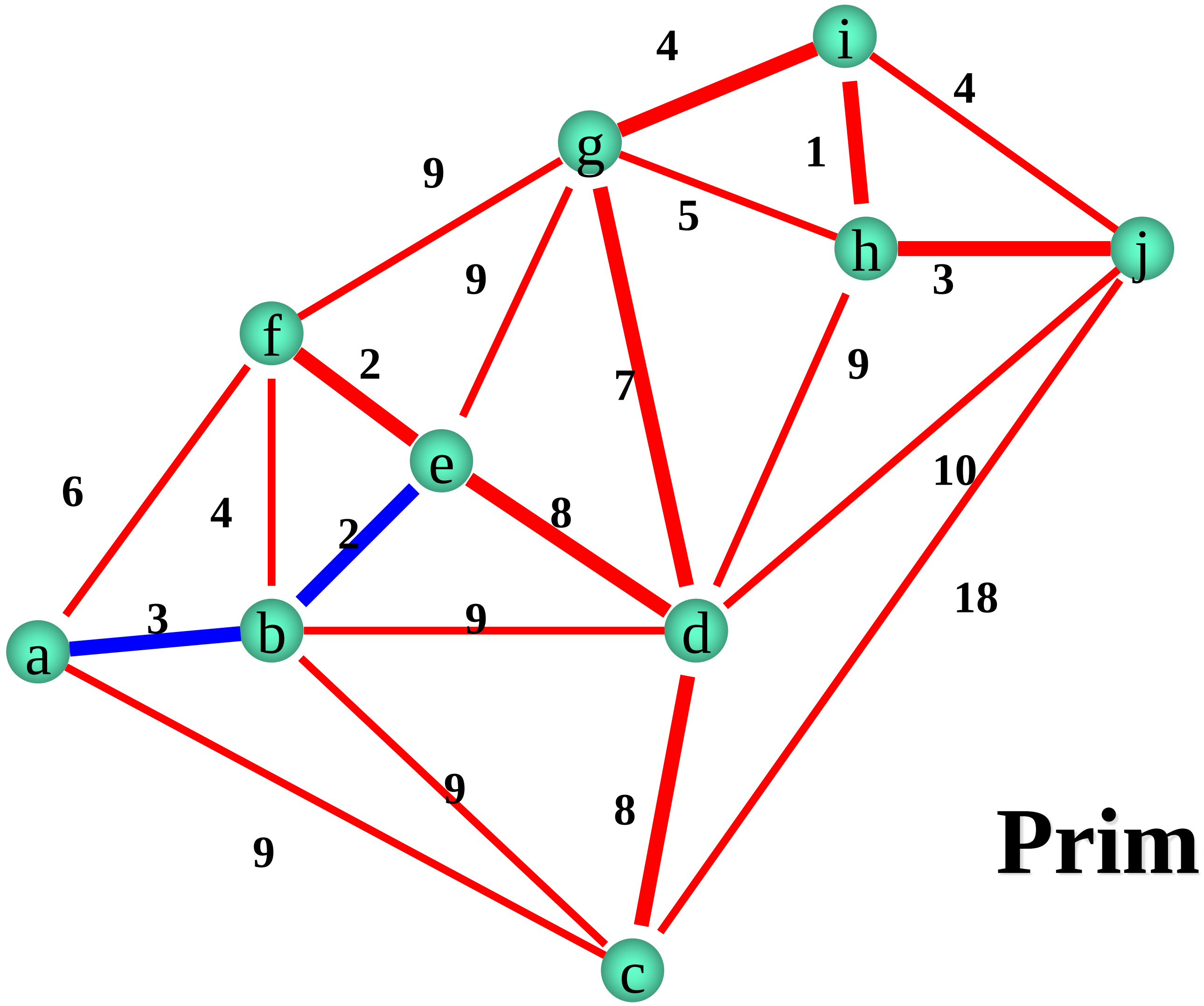
a	**b**	c	d	e	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞



Prim-Jarník-Dijkstra

D

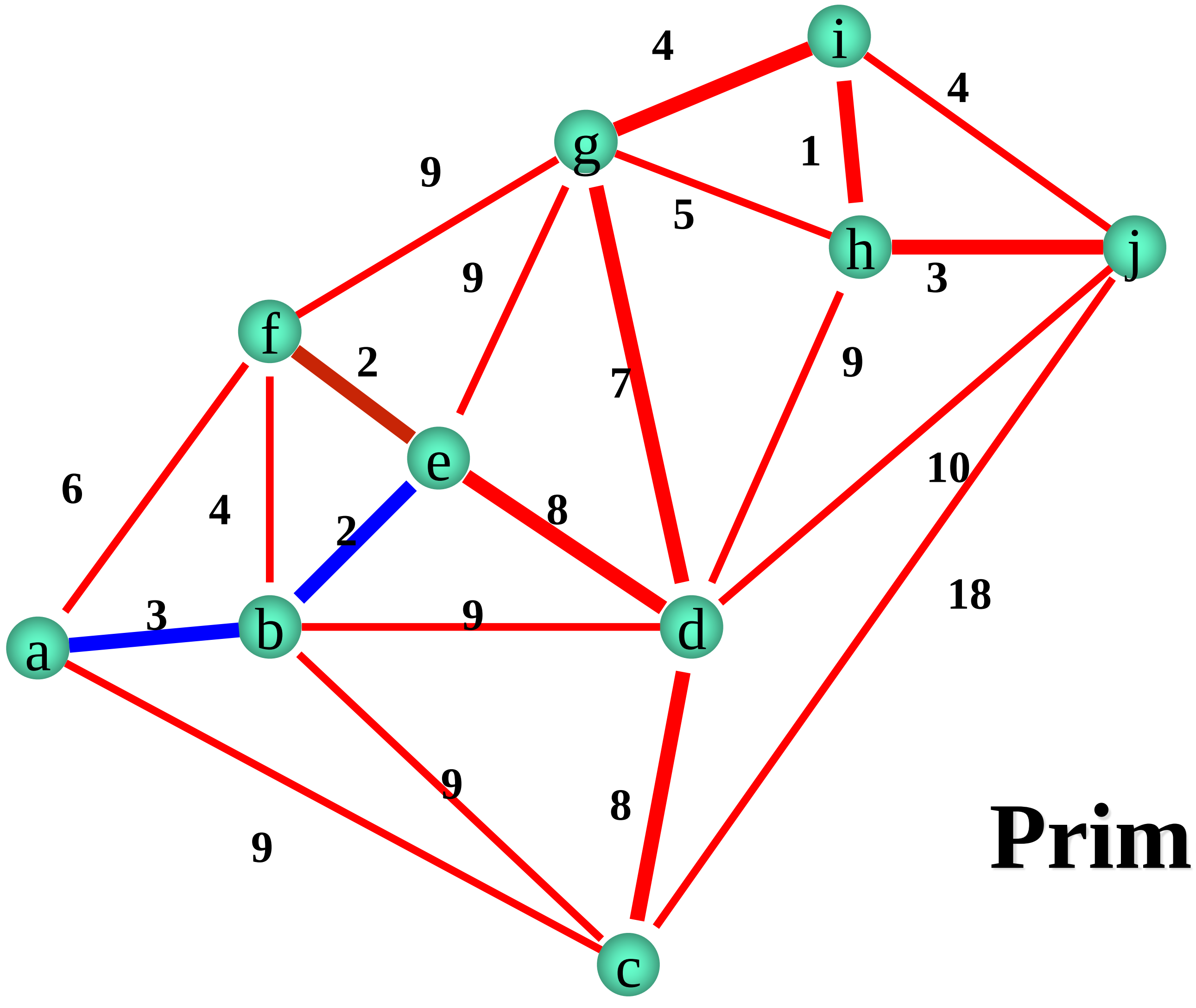
a	**b**	c	d	e	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞



Prim-Jarník-Dijkstra

D

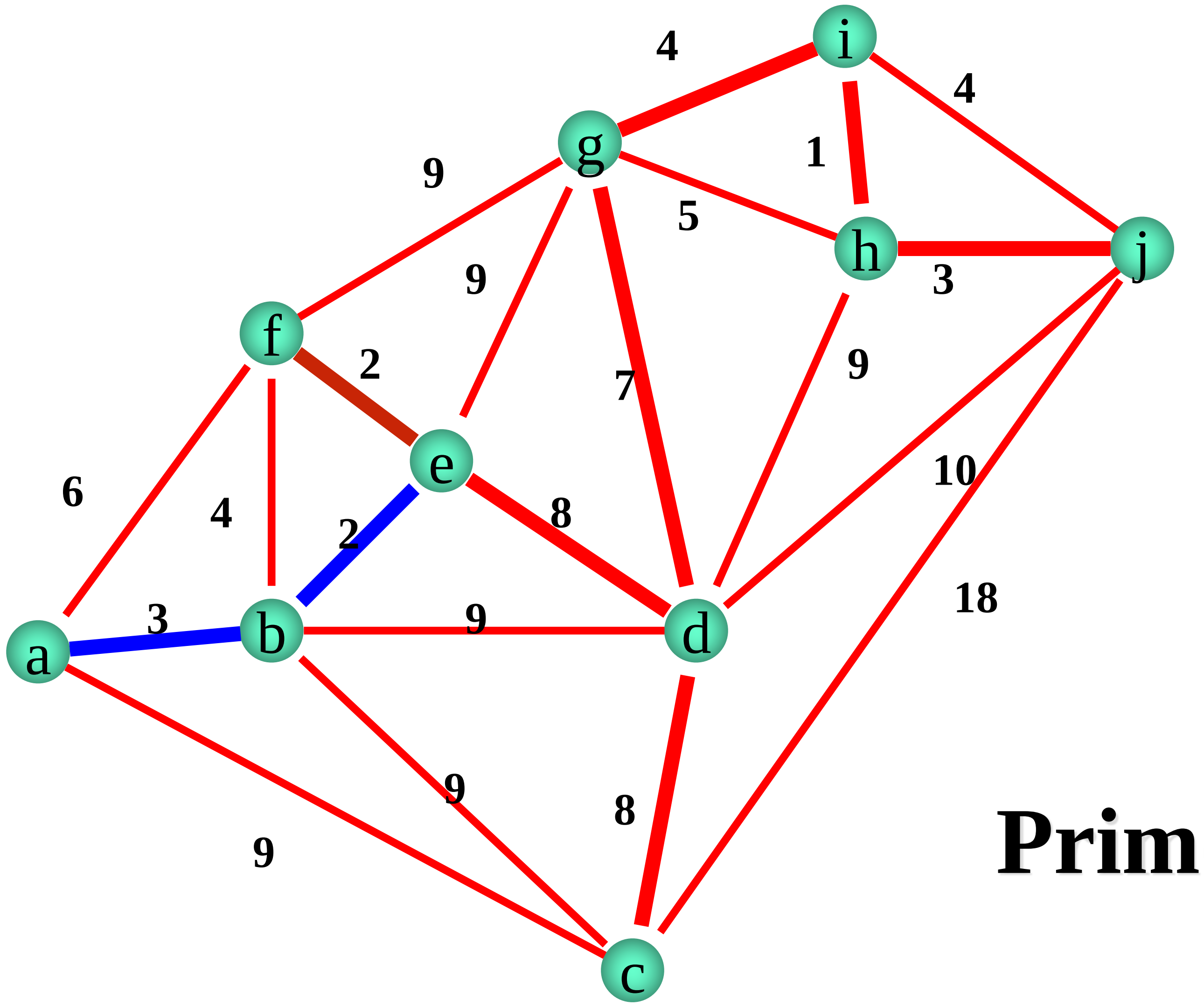
a	**b**	c	d	**e**	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞



Prim-Jarník-Dijkstra

D

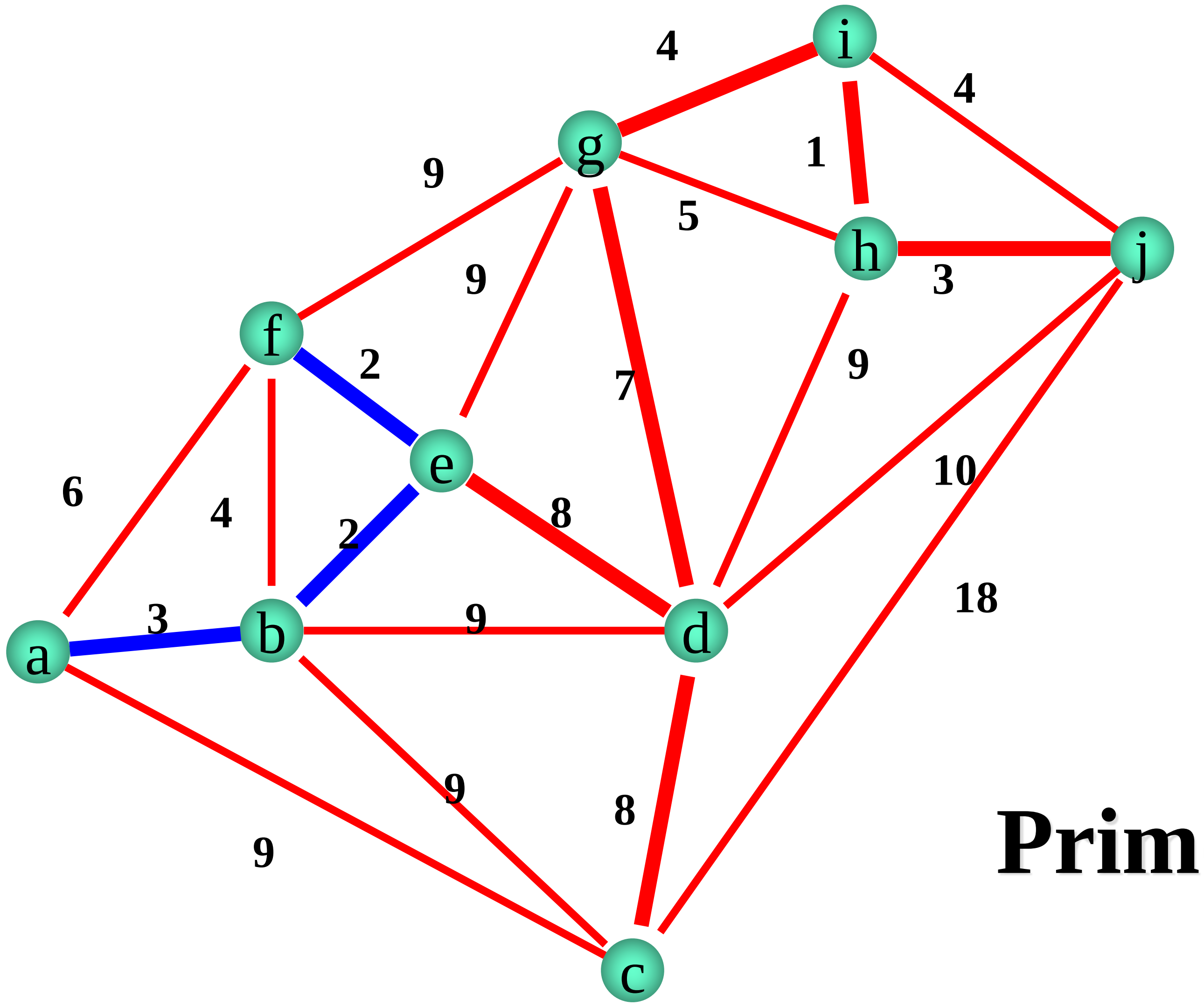
a	**b**	c	d	**e**	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞



Prim-Jarník-Dijkstra

D

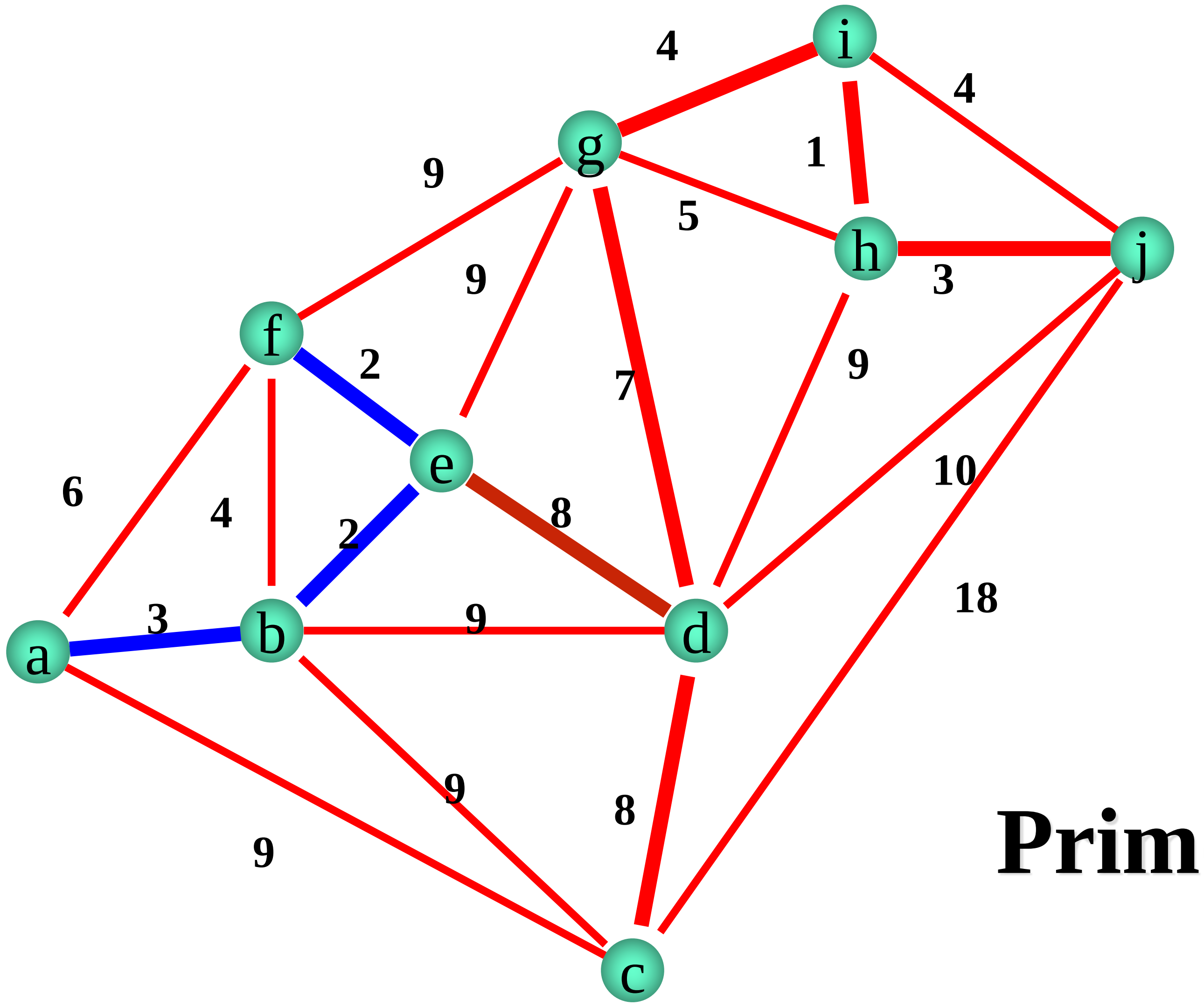
a	**b**	c	d	**e**	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞



Prim-Jarník-Dijkstra

D

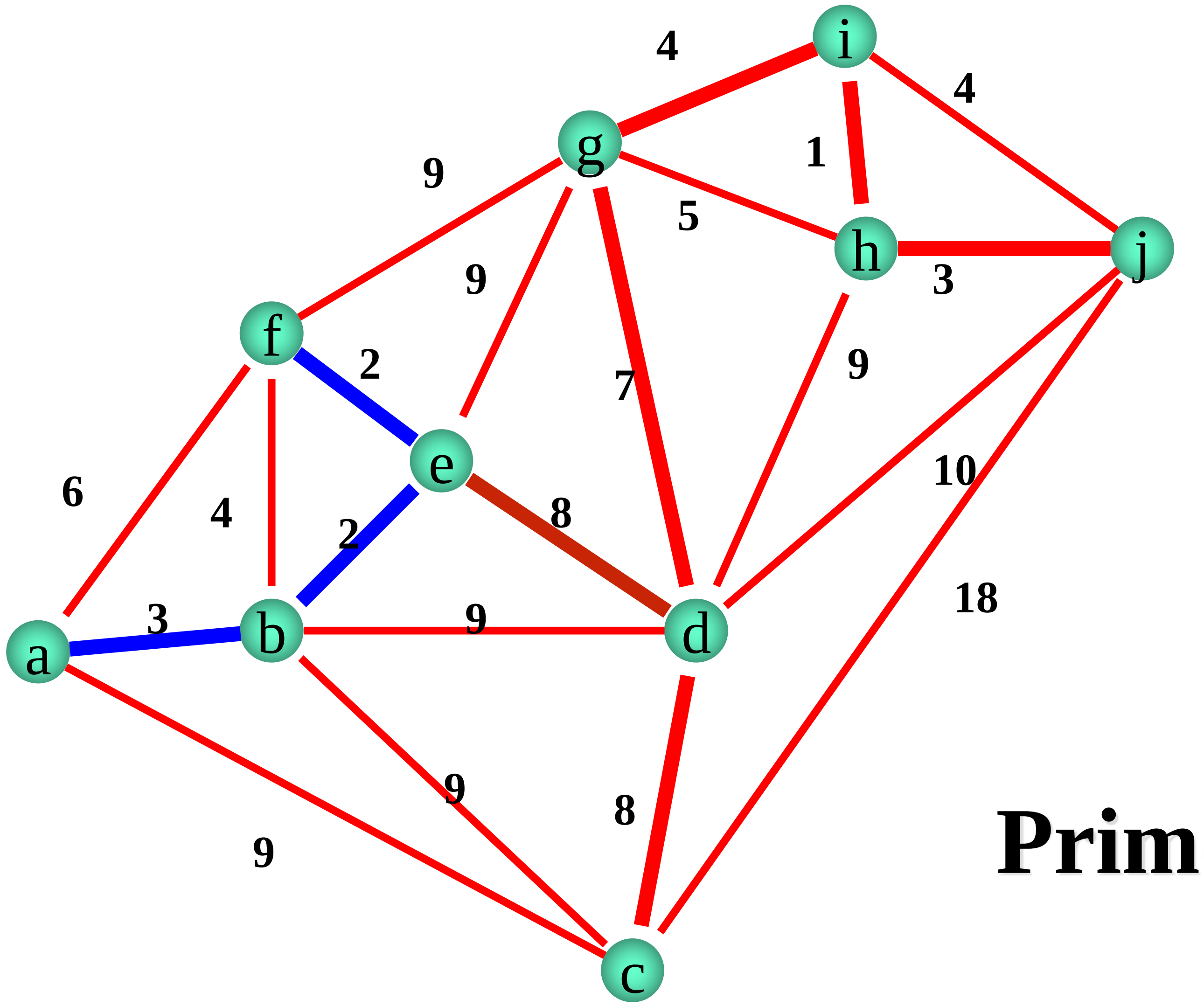
a	**b**	c	d	**e**	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞



Prim-Jarník-Dijkstra

D

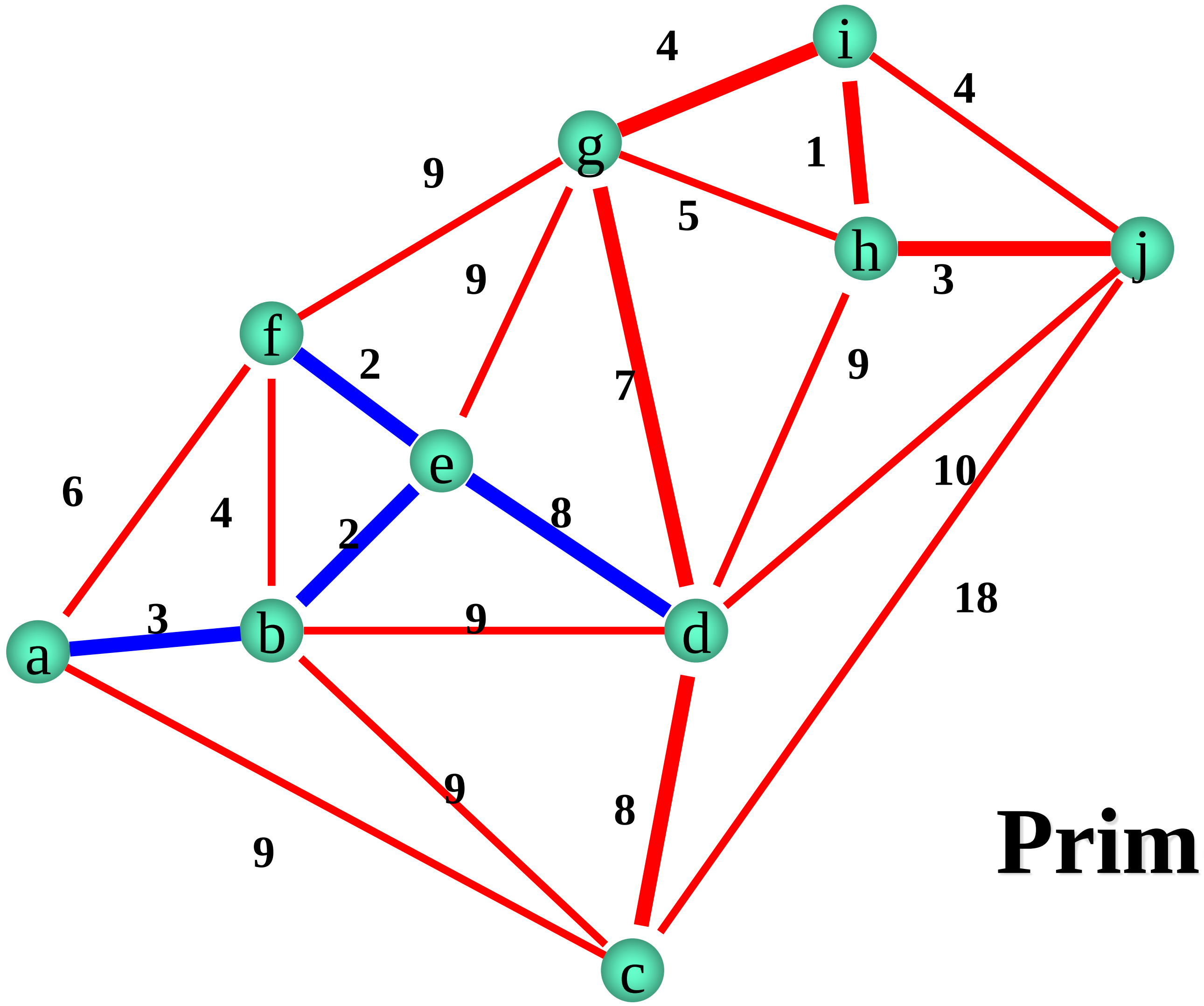
a	**b**	c	d	**e**	**f**	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞



Prim-Jarník-Dijkstra

D

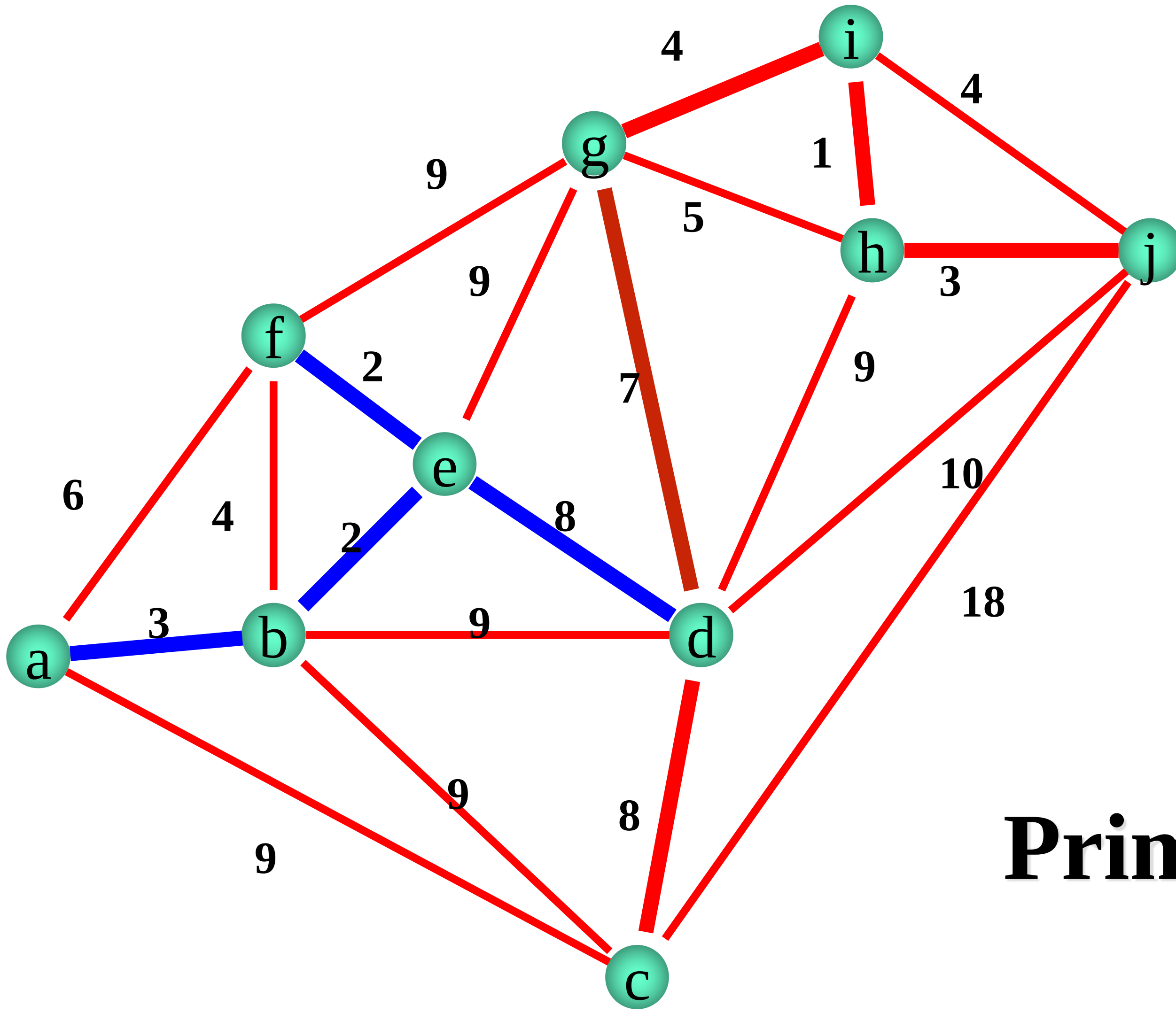
a	**b**	c	d	**e**	**f**	g	h	i	j
0	3	9	$+\infty$	$+\infty$	6	$+\infty$	$+\infty$	$+\infty$	$+\infty$
0	3	9	9	2	4	$+\infty$	$+\infty$	$+\infty$	$+\infty$
0	3	9	8	2	2	9	$+\infty$	$+\infty$	$+\infty$
0	3	9	8	2	2	9	$+\infty$	$+\infty$	$+\infty$



Prim-Jarník-Dijkstra

D

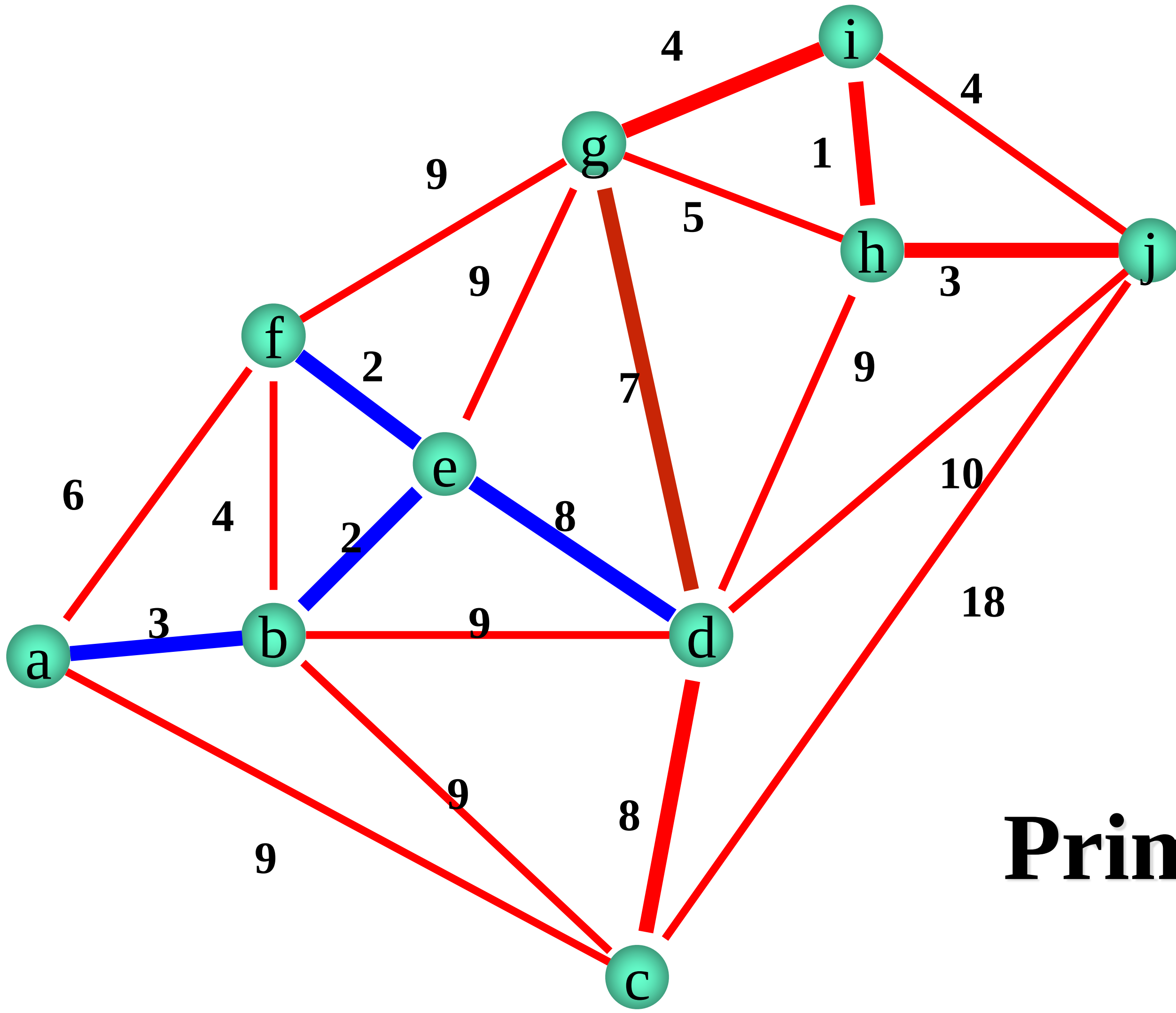
a	**b**	c	d	**e**	**f**	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞



Prim-Jarník-Dijkstra

D

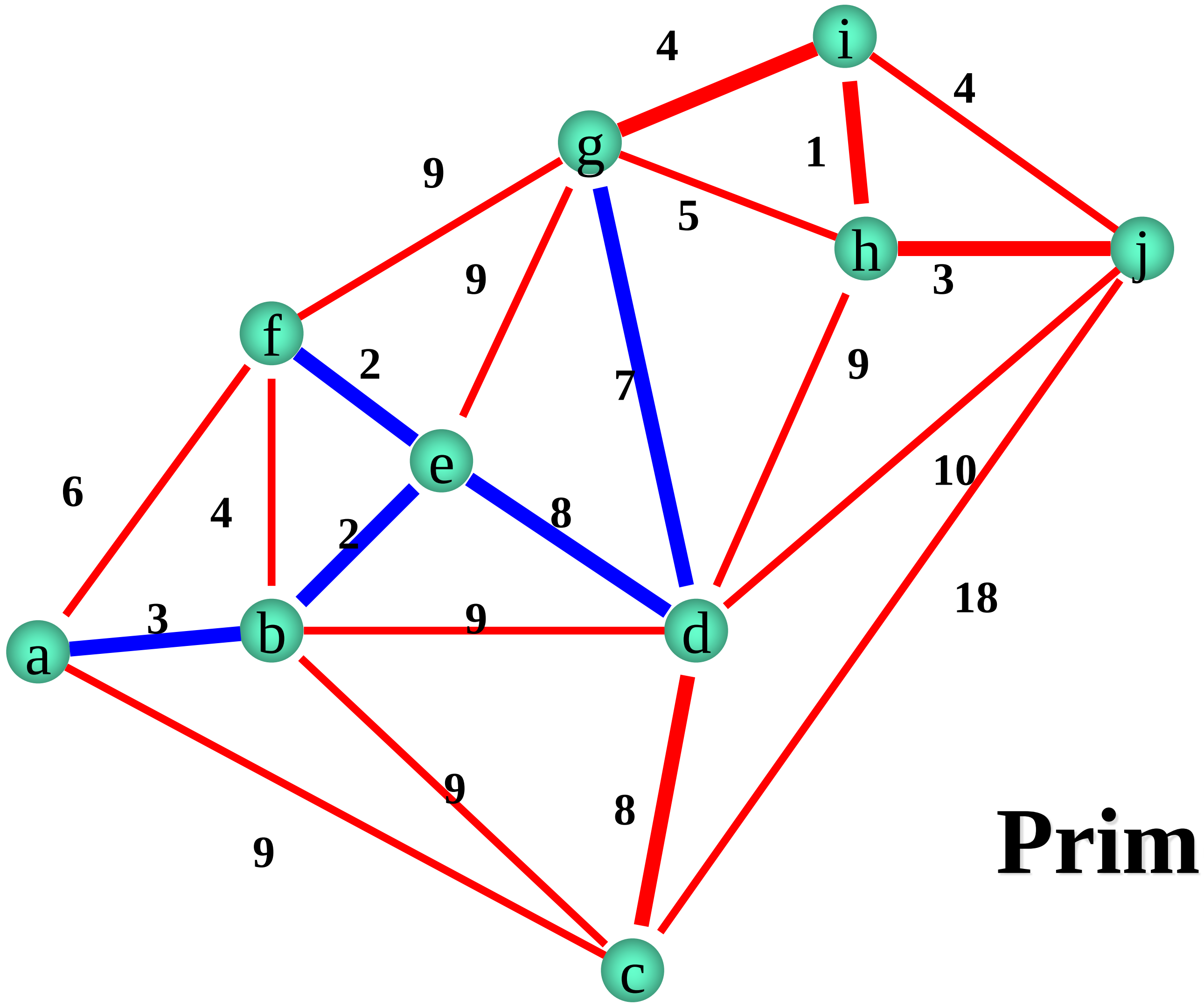
a	**b**	c	**d**	**e**	**f**	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10



Prim-Jarník-Dijkstra

D

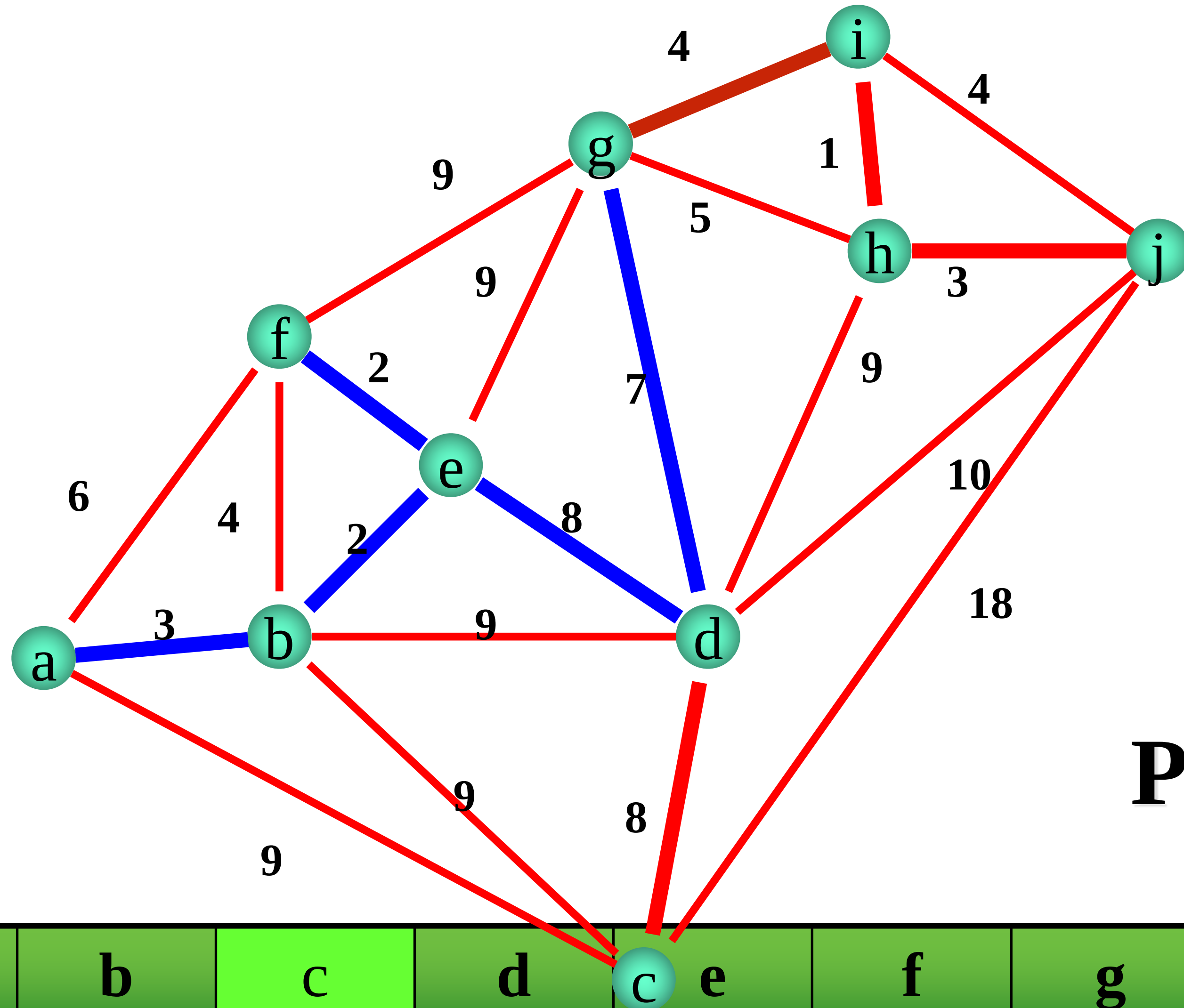
a	**b**	c	**d**	**e**	**f**	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10



Prim-Jarník-Dijkstra

D

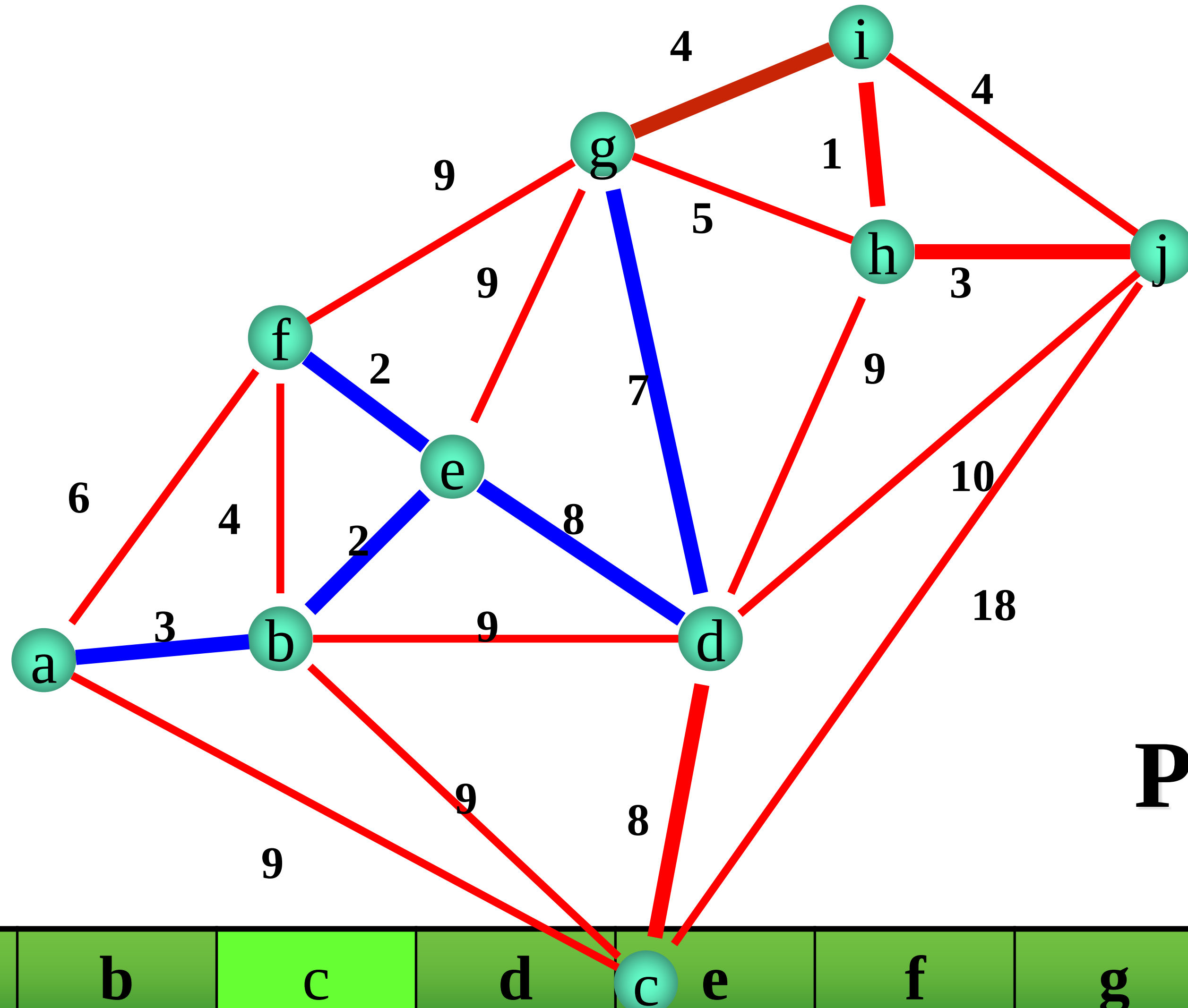
a	b	c	d	e	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10



Prim-Jarník-Dijkstra

D

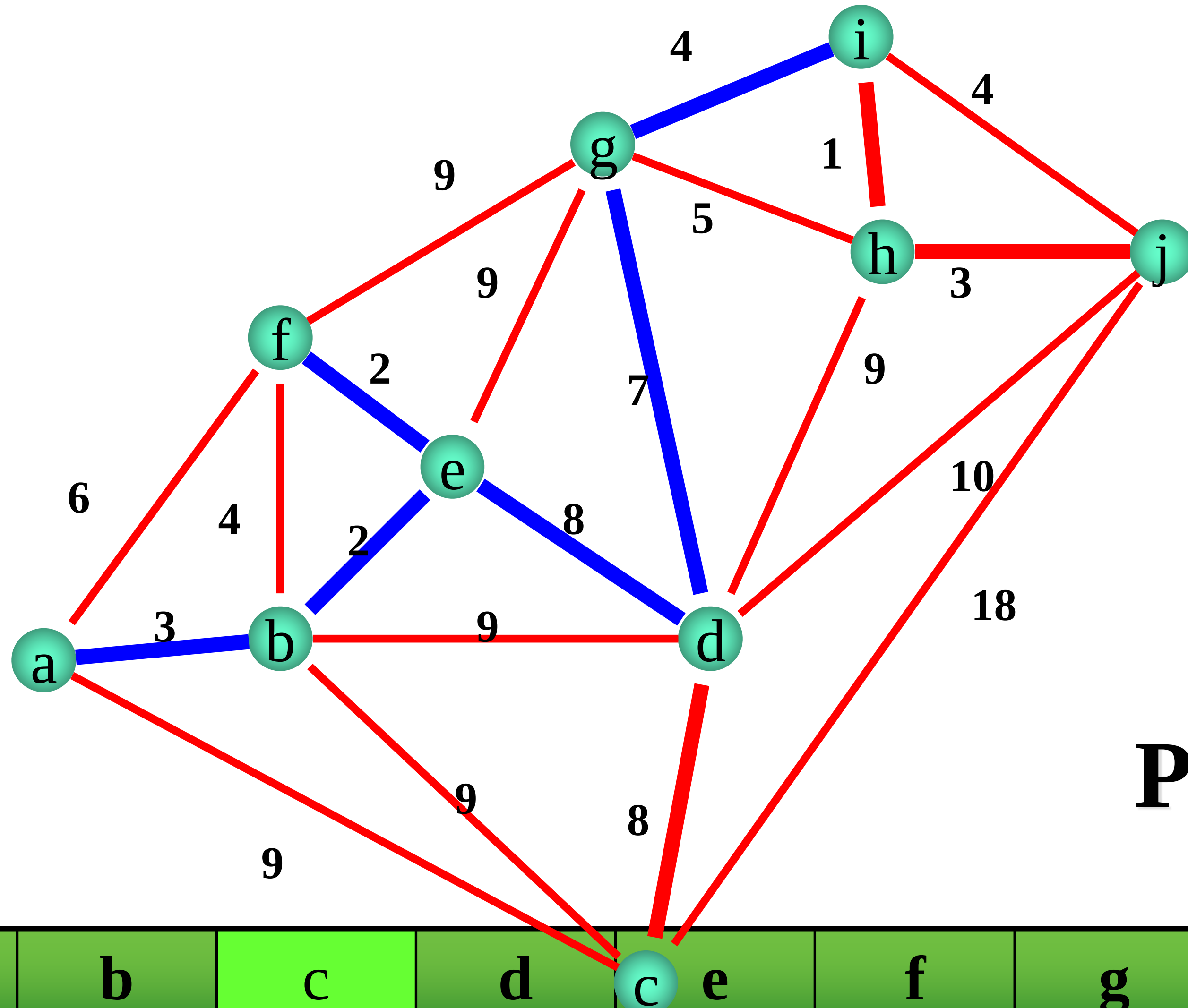
a	**b**	c	**d**	**e**	**f**	**g**	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10
0	3	8	8	2	2	7	5	4	10



Prim-Jarník-Dijkstra

D

a	b	c	d	e	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10
0	3	8	8	2	2	7	5	4	10

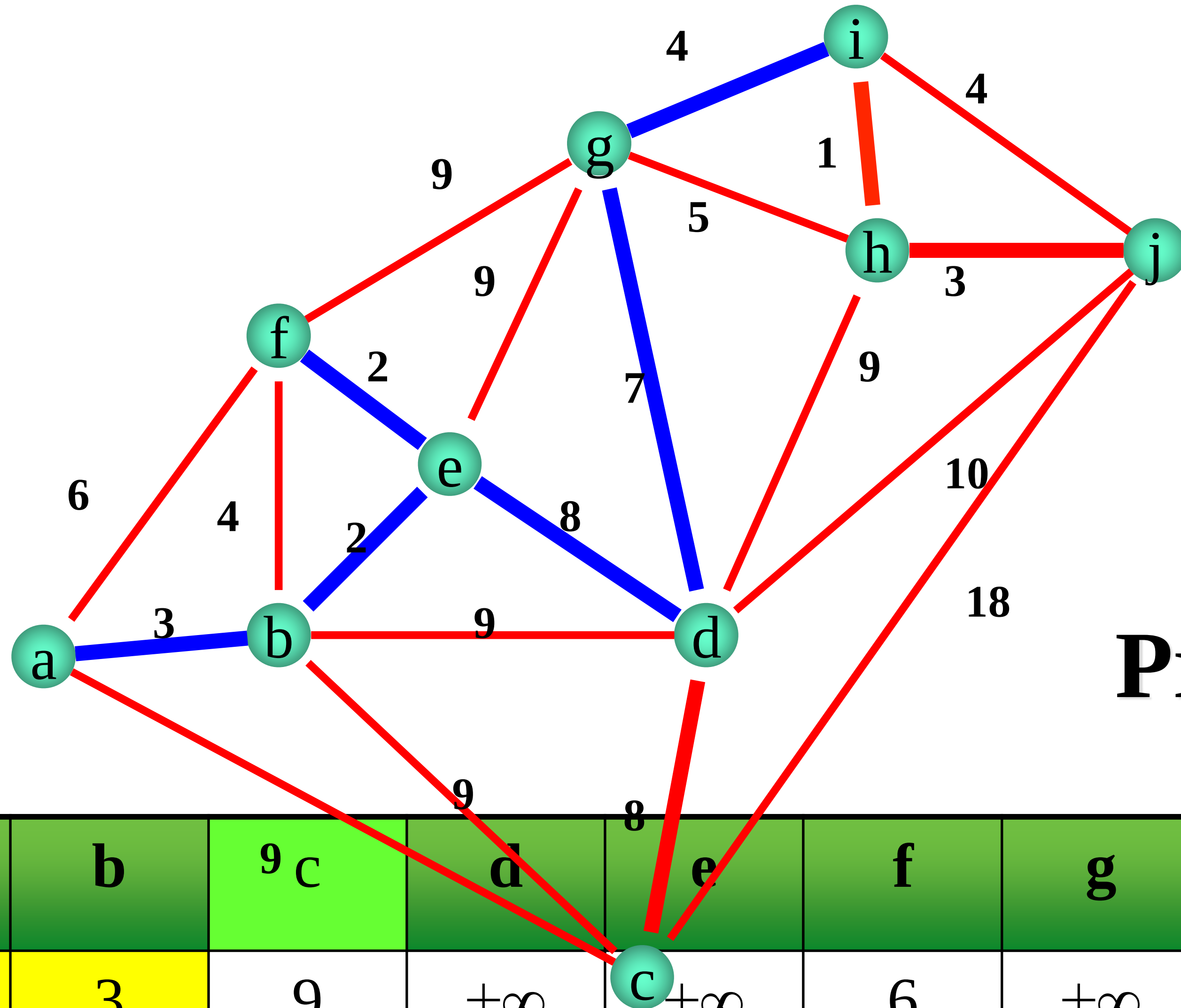


Prim-Jarník-Dijkstra

D

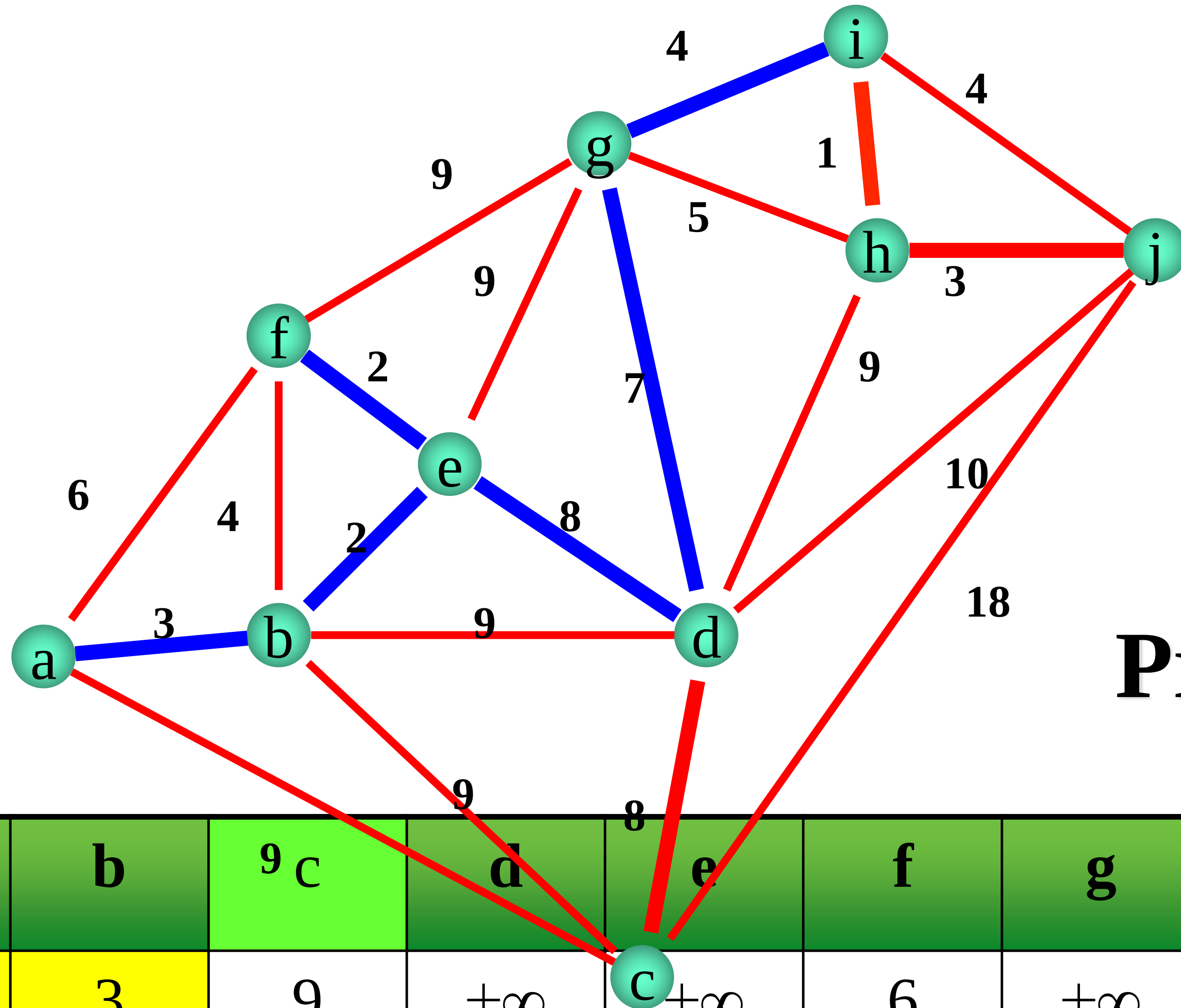
a	b	c	d	e	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10
0	3	8	8	2	2	7	5	4	10

Prim-Jarník-Dijkstra



D

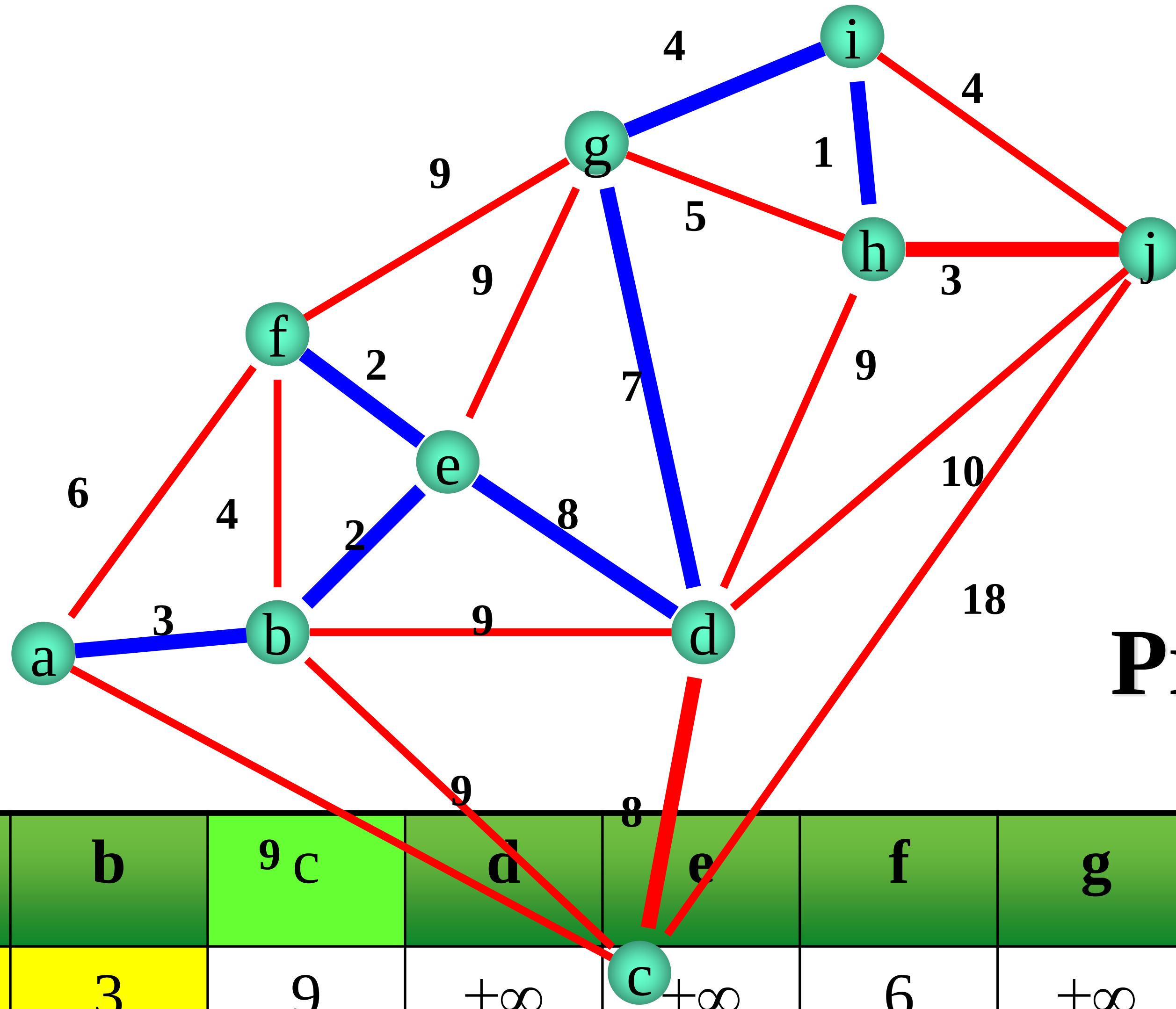
a	b	c	d	e	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10
0	3	8	8	2	2	7	5	4	10
0	3	8	8	2	2	7	1	4	4



Prim-Jarník-Dijkstra

D

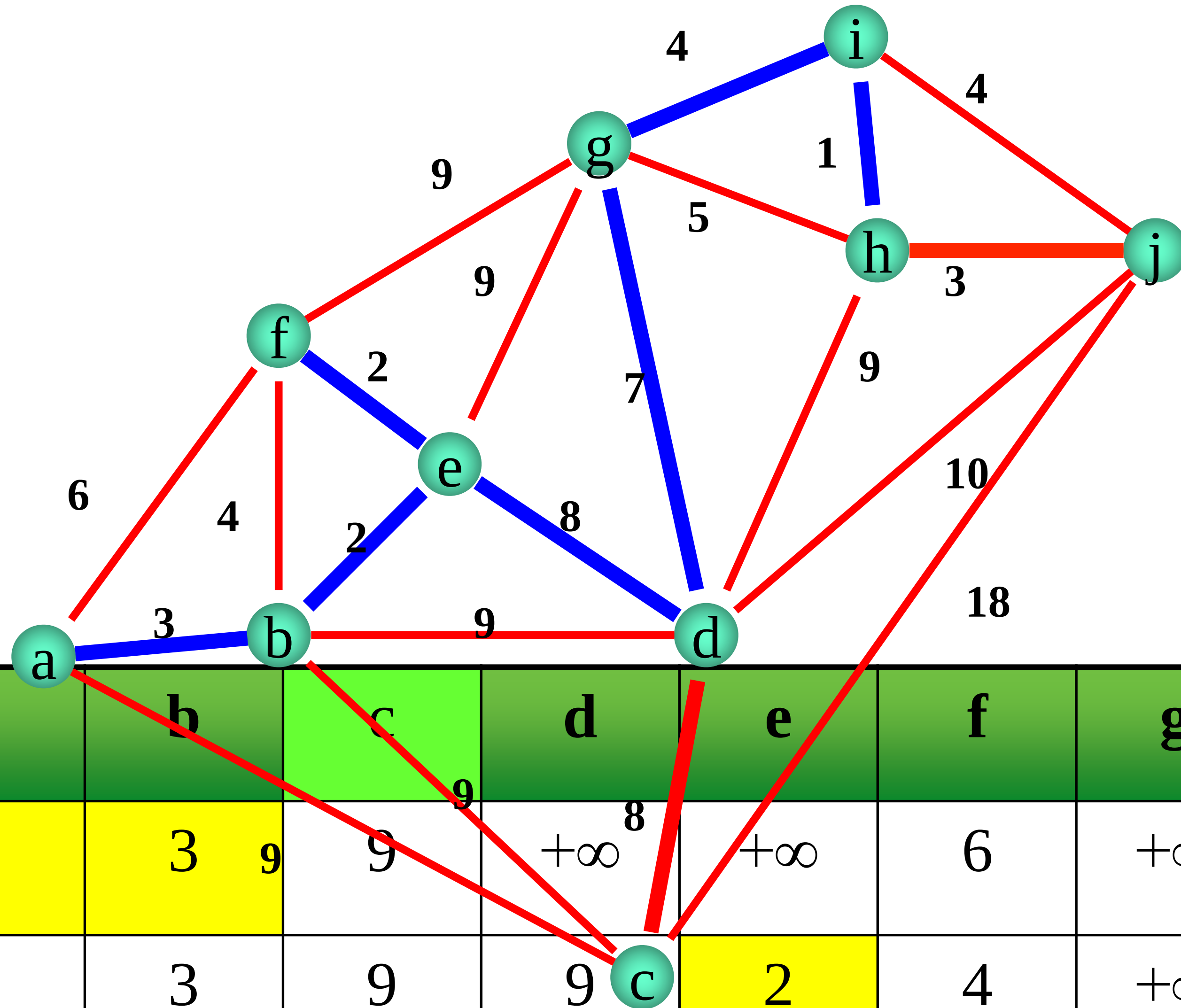
a	**b**	c	**d**	**e**	**f**	**g**	h	**i**	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10
0	3	8	8	2	2	7	5	4	10
0	3	8	8	2	2	7	1	4	4



Prim-Jarník-Dijkstra

D

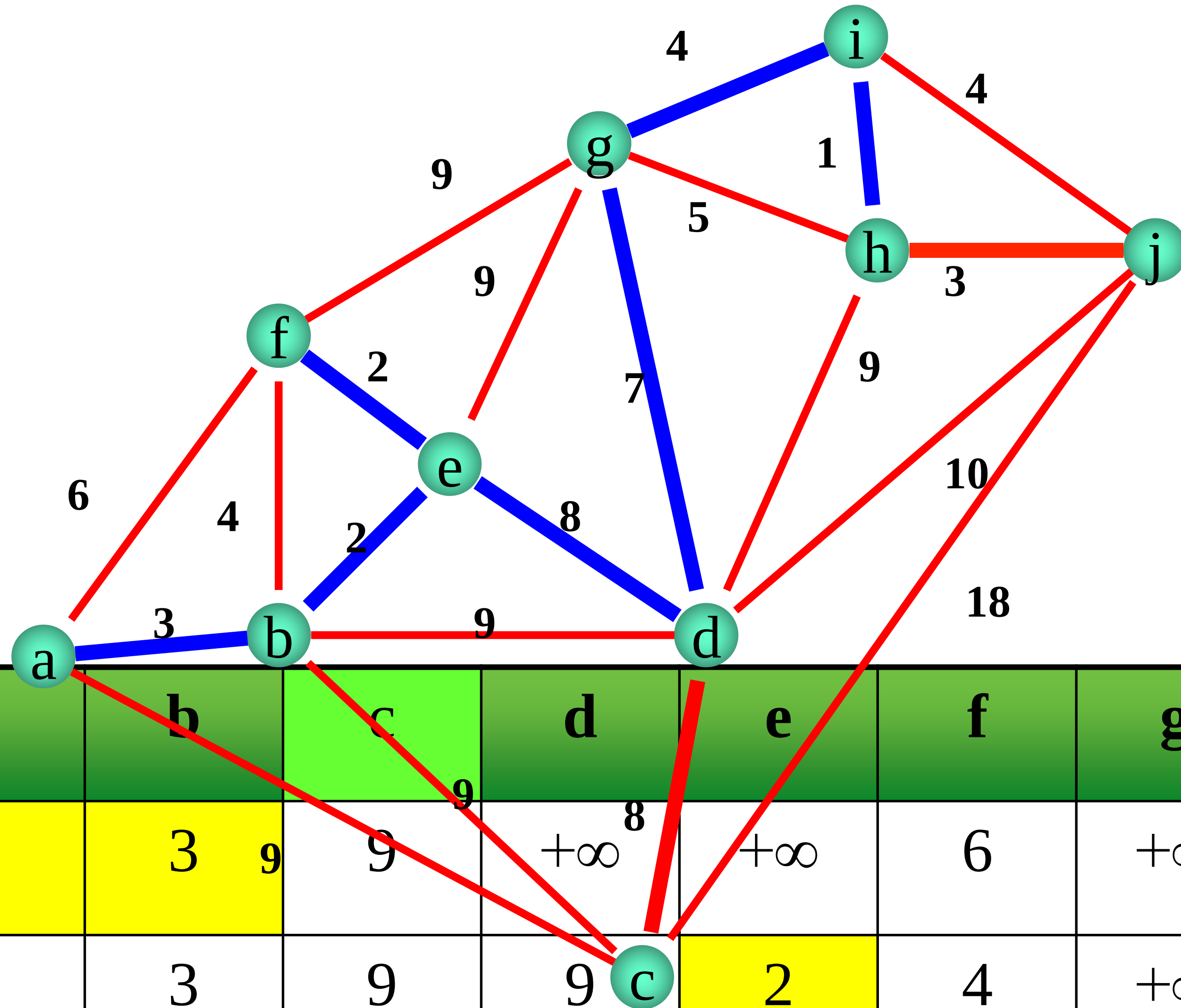
a	**b**	c	**d**	**e**	**f**	**g**	h	**i**	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10
0	3	8	8	2	2	7	5	4	10
0	3	8	8	2	2	7	1	4	4



Prim-Jarník-Dijkstra

D

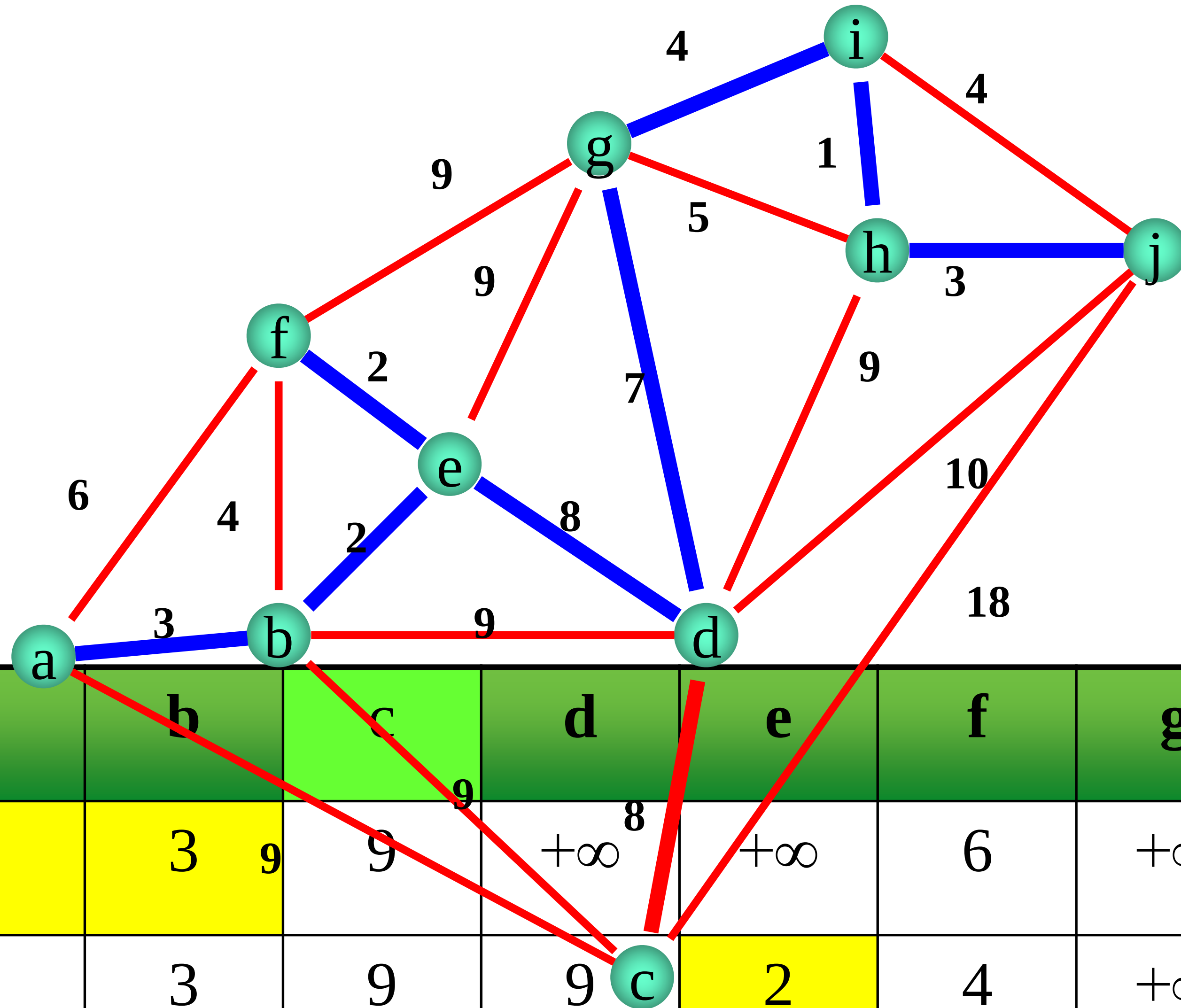
a	b	c	d	e	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10
0	3	8	8	2	2	7	1	4	4
0	3	8	8	2	2	7	1	4	4
0	3	8	8	2	2	7	1	4	3



Prim-Jarník-Dijkstra

D

a	b	c	d	e	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10
0	3	8	8	2	2	7	1	4	4
0	3	8	8	2	2	7	1	4	4
0	3	8	8	2	2	7	1	4	3

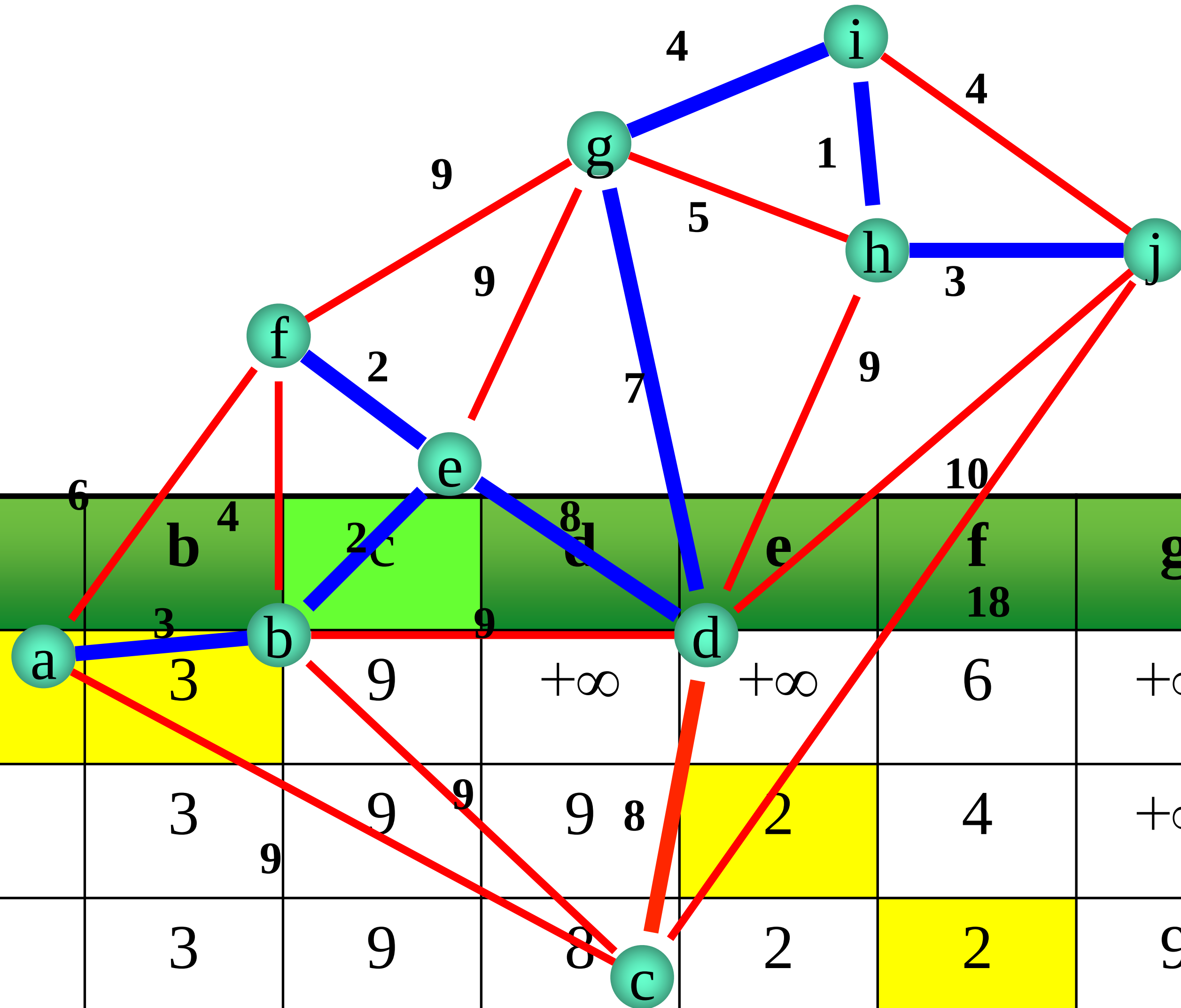


Prim-Jarník-Dijkstra

D

a	b	c	d	e	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10
0	3	8	8	2	2	7	1	4	4
0	3	8	8	2	2	7	1	4	4
0	3	8	8	2	2	7	1	4	3

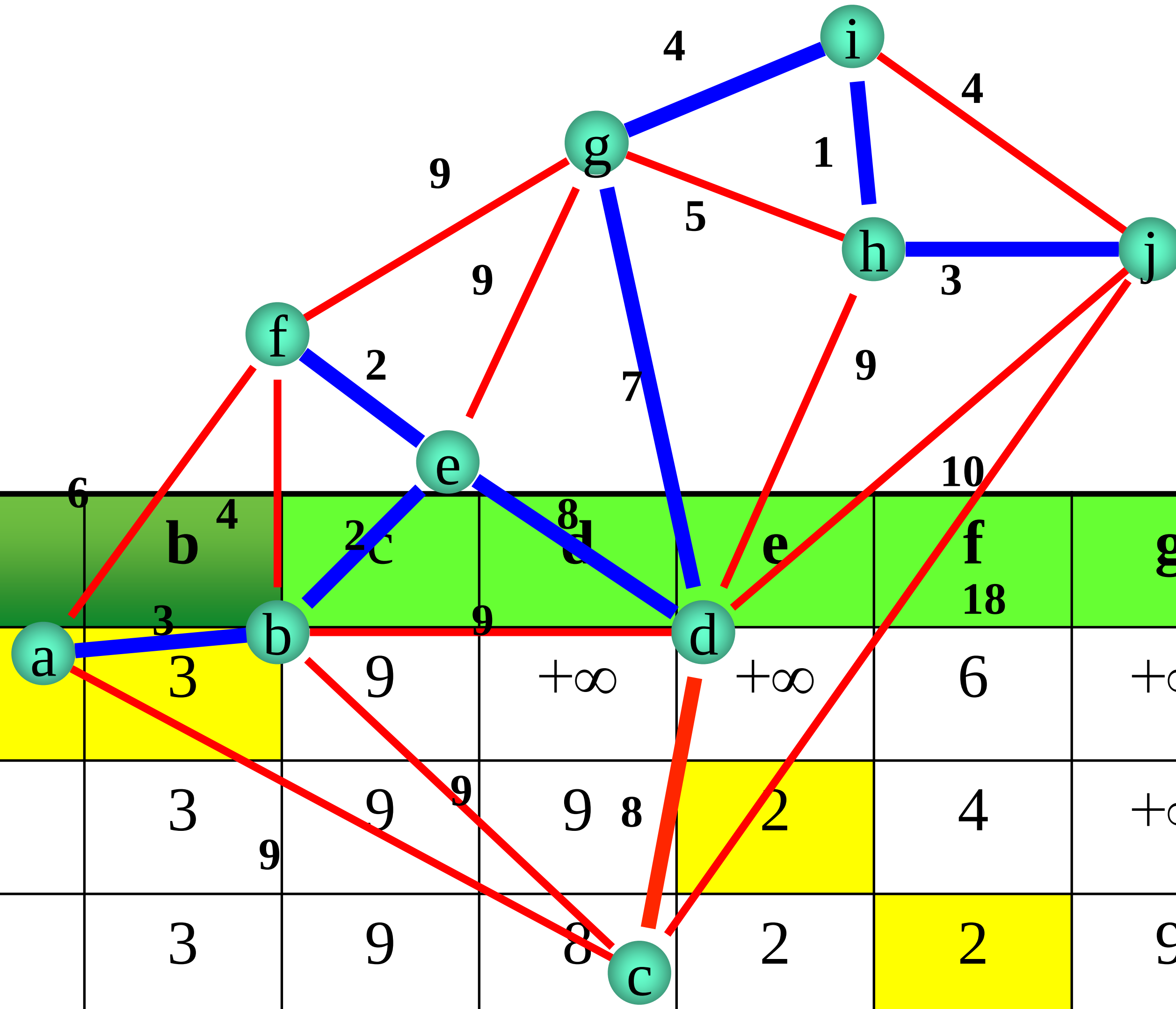
Prim-Jarník-Dijkstra



D

a	**b**	c	d	**e**	**f**	**g**	h	**i**	**j**
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10
0	3	8	8	2	2	7	1	4	4
0	3	8	8	2	2	7	1	4	4
0	3	8	8	2	2	7	1	4	3
0	3	8	8	2	2	7	1	4	3

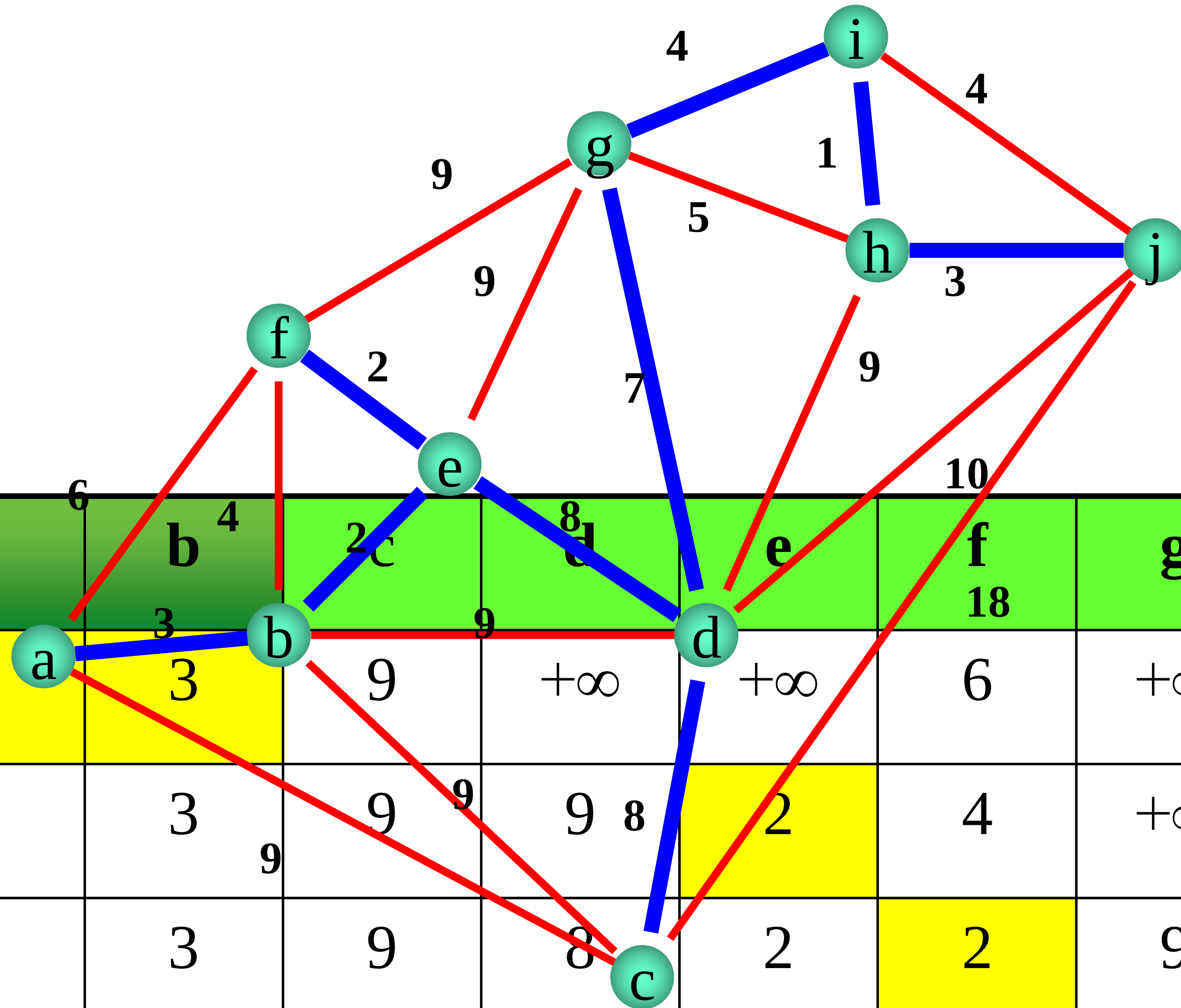
Prim-Jarník-Dijkstra



D

a	b	c	d	e	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10
0	3	8	8	2	2	7	1	4	4
0	3	8	8	2	2	7	1	4	4
0	3	8	8	2	2	7	1	4	3
0	3	8	8	2	2	7	1	4	3

Prim-Jarník-Dijkstra



D

a	**b**	c	d	**e**	**f**	**g**	h	**i**	**j**
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10
0	3	8	8	2	2	7	1	4	4
0	3	8	8	2	2	7	1	4	4
0	3	8	8	2	2	7	1	4	3
0	3	8	8	2	2	7	1	4	3

Pseudocode: Prim's Algorithm

D: distance vector, maintains reachable vertices

PQ: priority queue (heap) for the edges, according to their values in D

Algorithm Prim-Jarník-Dijkstra
Input: a weighted connected graph $G = (V, E)$
Output: an MST T for G
Data structure: array D; Priority Queue PQ; tree T

pick an arbitrary vertex v in G; $D[v] \leftarrow 0$
for each vertex $u \neq v$ **do** $D[u] \leftarrow +\infty$ **end**
$T \leftarrow \emptyset$
for each vertex u **do** PQ.insert({(u, (null), $D[u]$}) **end** // including v
// for each vertex u, (u, edge) is the element and $D[u]$ is the key in PQ
while not PQ.empty() **do**
  $(u, e) \leftarrow PQ$.deleteMin()
  add vertex u and edge e to T
  for each vertex z adjacent to u such that z is in PQ **do**
    if weight((u, z)) $< D[z]$ **then**
      $D[z] \leftarrow$ weight((u, z))
      in PQ, change element and key of z to $\{z, (u,z), D[z]\}$
      update PQ
    end
  end
end
return T

Prim-Jarník Time Complexity

Theorem. The Prim-Jarník algorithm constructs a minimum spanning tree for a connected weighted graph $G = (V, E)$ with n vertices and m edges in $O(m \log(n))$ time.

Prim's algorithm: eager implementation

```
public class PrimMST {
  private Edge[] edgeTo;              // shortest edge from tree to vertex
  private double[] distTo;            // distTo[w] = edgeTo[w].weight()
  private boolean[] marked;           // true if v in mst
  private IndexMinPQ<Edge> pq;        // eligible crossing edges

   public PrimMST(WeightedGraph G) {
     edgeTo = new Edge[G.V()];
     distTo = new double[G.V()];
     marked = new boolean[G.V()];
     for(int v = 0; v < G.V(); v++)
        distTo[v] = Double.POSITIVE_INFINITY;
     pq = new IndexMinPQ<Double>(G.V());
     distTo[0[ = 0.0;
     pq.insert(0, 0.0);
     while(!pq.isEmpty())
         visit(G, pq.delMin());
  }
```

assume G is connected

repeatedly delete the
min weight edge e = v–w from PQ

Prim's algorithm: eager implementation

```
private void visit(WeightedGraph G, int v) {
   marked[v] = true;
   for (Edge e : G.adj(v)) {
      int w = e.other(v);
      if (marked[w]) continue;
      if (e.weight() < distTo[w]) {
           edgeTo[w] = e;
           distTo[w] = e.weight();
           if (pq.contains(w)) pq.changeKey(w, distTo[w]);
           else pq.insert(w, distTo[w]);
       }
   }
}
public Iterable<Edge> edges(){
   Queue<Edge> mst = new Queue<Edge>();
   for (int v = 0; v < edgeTo.length; v++)
      Edge e = edgeTo[v];
      if (e != null) {
         mst.enqueue(e);
      }
   }
   return mst; }
```

add v to T

for each edge e = v–w, add to PQ if w not already in T

add edge e to tree

Update distance to w or Insert distance to w

Create the mst