CSC 226

Algorithms and Data Structures: II

Rich Little
rlittle@uvic.ca

Two basic facts about trees

- Let T be a connected graph with no cycles, that is a tree.

  - What happens if we add a new edge to T, without adding a new vertex?

  - What happens if we remove an edge from T, without removing any vertices?

Two fundamental properties of minimum spanning trees

- Cycle property

- Cut property

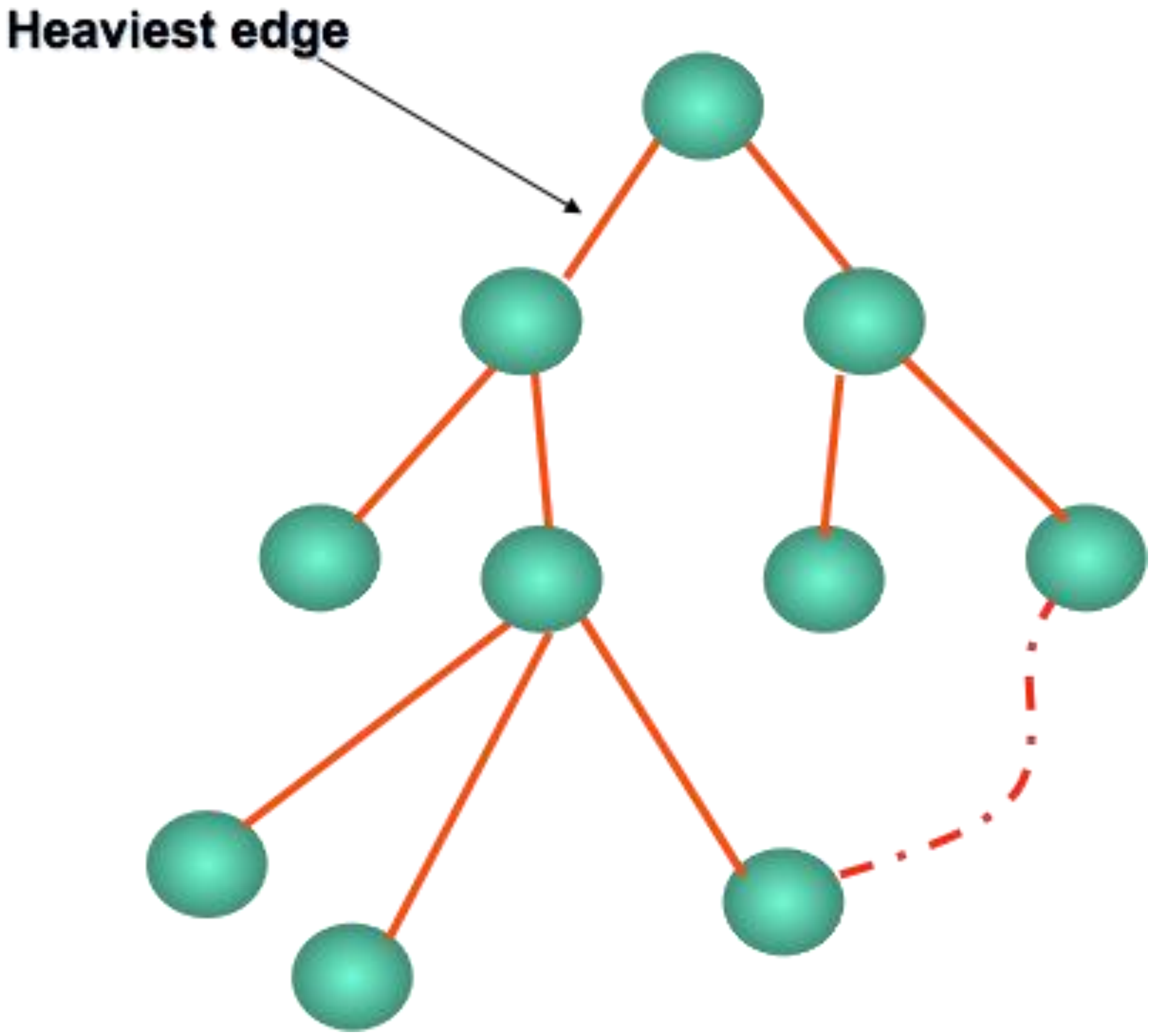
Cycle property

- ***Theorem – Cycle Property***:
  Let C be any cycle in a 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



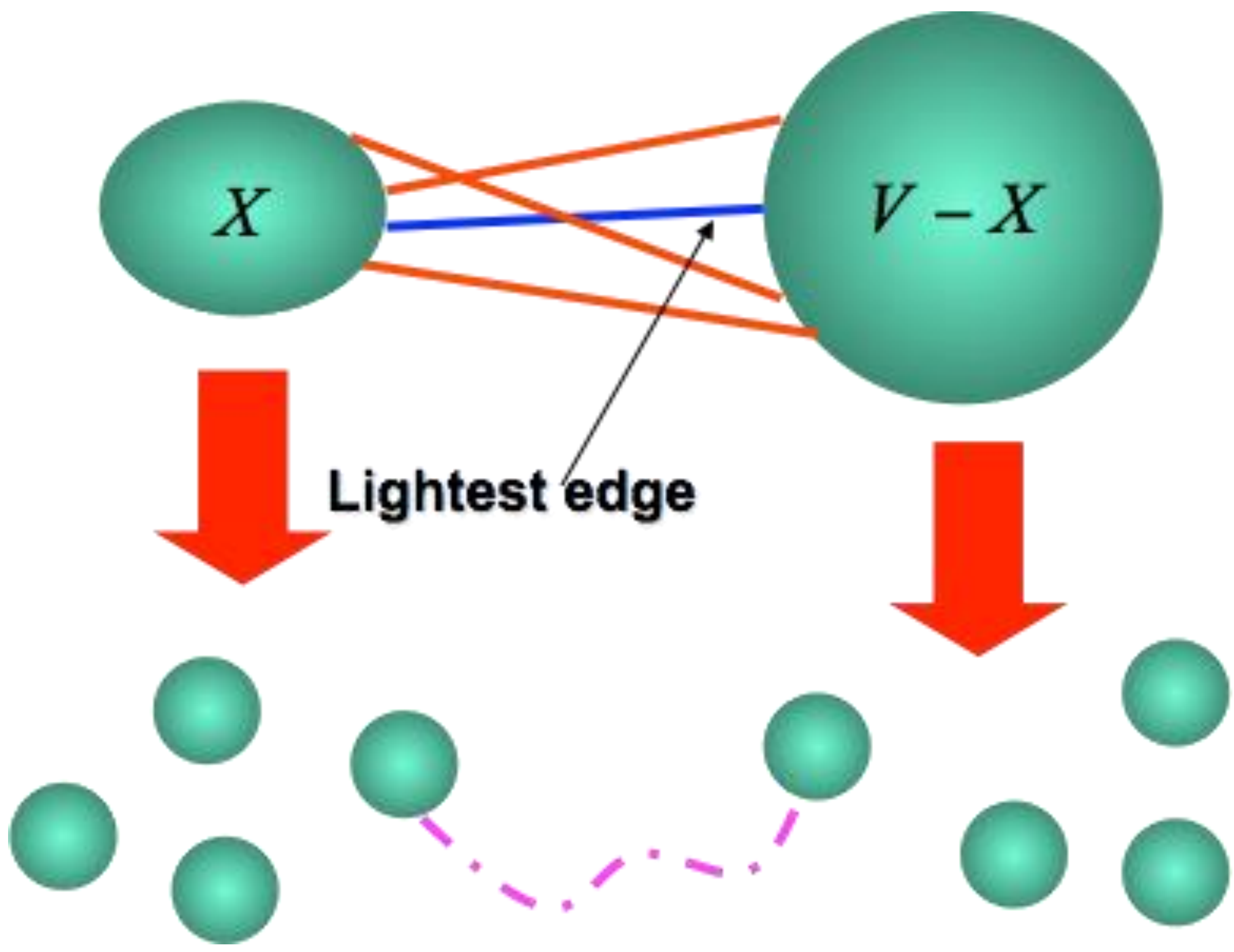
Cut Property

- ***Theorem – Cut Property:***
  Let X be any proper subset of vertices in a weighted graph $G = (V, E)$, and let e be the lightest edge that has exactly one endpoint in X. Then, the minimum spanning tree T for G must contain 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 X
- 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*

Prim's Correctness Proof

- ***Theorem:*** If $G = (V, E)$ is a connected, weighted graph with distinct edge weights, then Prim's algorithm correctly finds the MST for G.

- **Proof:** Let T be the MST for G. Let S be the spanning tree created by Prim's algorithm. We want to show that $T = S$.

- We will use induction on the number of edges added to S.

- That is, we show that for every edge e that Prim's adds to S, then e must be in T.

Prim's Correctness Proof

- *Base case:* When $m = 1$, or after 1 edge has been added to S. Let v_0 denote the starting vertex. Let $e_1 = \{v_0, v_1\}$ be that edge.

- At this point $S = \left(\{v_0, v_1\}, \left\{\{v_0, v_1\}\right\}\right)$. Thus, by Prim, e_1 is the lightest edge incident upon v_0.

- Let $X = \{v_0\}$ be a cut of graph $G = (V, E)$. By the cut property the minimum weight edge from X to $V - X$ must be in the MST T. That edge is e_1, thus $e_1 \in T$.

Prim's Correctness Proof

- *Induction:* Let $m = k$ and let

$$S = (\{v_0, v_1, \dots, v_k\}, \{e_1, e_2, \dots, e_k\})$$

  be the current state of the tree built by Prim's algorithm after k iterations.

- Assume that $e_1, e_2, \dots, e_k$ are all in T, the MST for G.
- Now, run the next iteration of Prim's, adding vertex v_{k+1} and edge e_{k+1}, i.e. let $m = k + 1$.
- Let $X = \{v_0, v_1, \dots, v_k\}$ be a cut of the vertex set V, in G. By the cut property, the lightest edge from X to V-X must be in the MST for G. That edge is e_{k+1}, by Prim's which chooses the lightest edge out of $\{v_0, v_1, \dots, v_k\}$ which does not create a cycle.
- Therefore, e_{k+1} is in T and we are done. Every edge that Prim's adds to tree S is in the minimum spanning tree T and Prim's adds exactly $n - 1$ edges.

Pseudocode: Prim's Algorithm

Algorithm PrimJarníkMST(G):

Input*:* A weighted connected graph G with n vertices and m edges

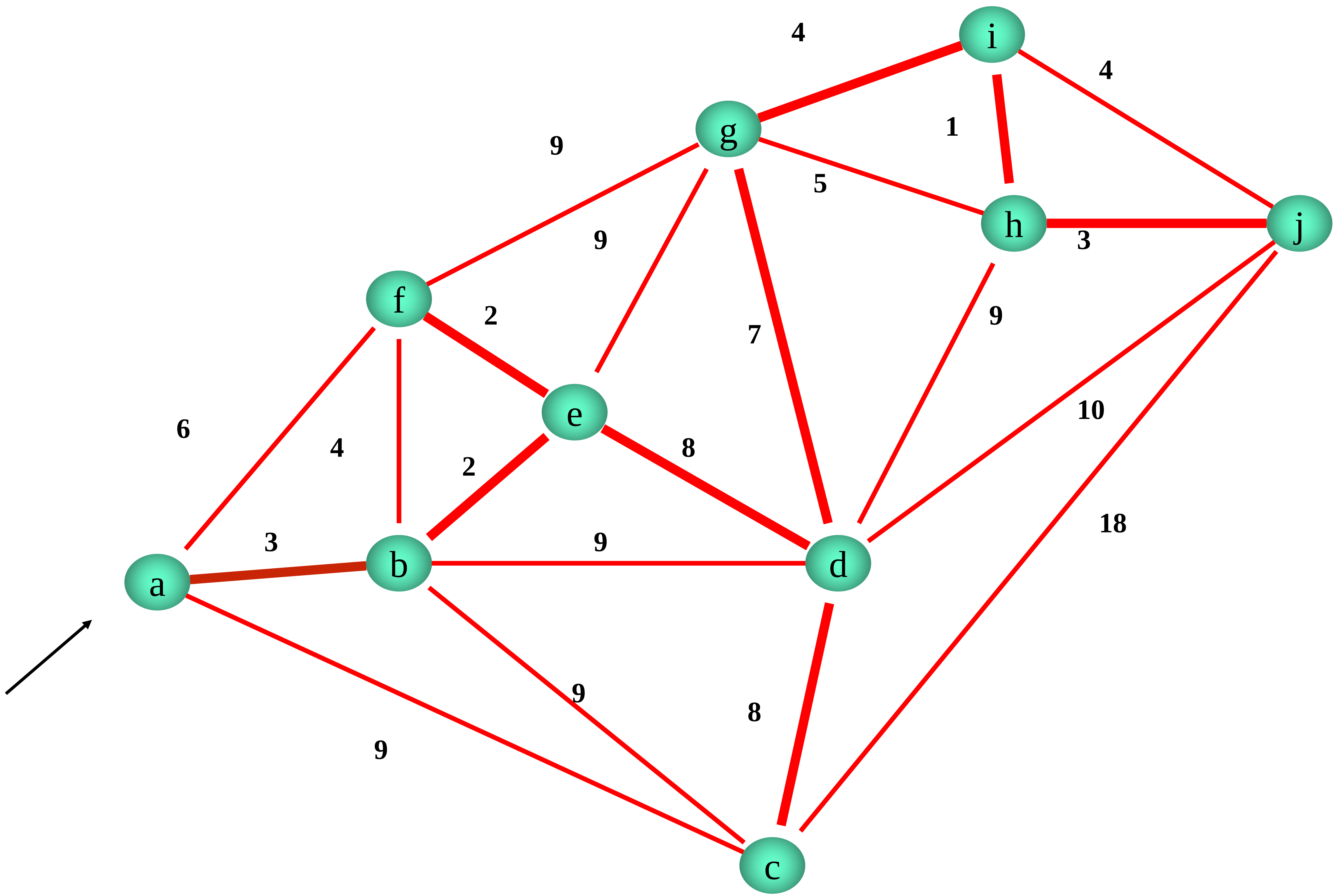
Output*:* an MST T for G

Data structures*:* Array D; Priority Queue Q*;* and tree T

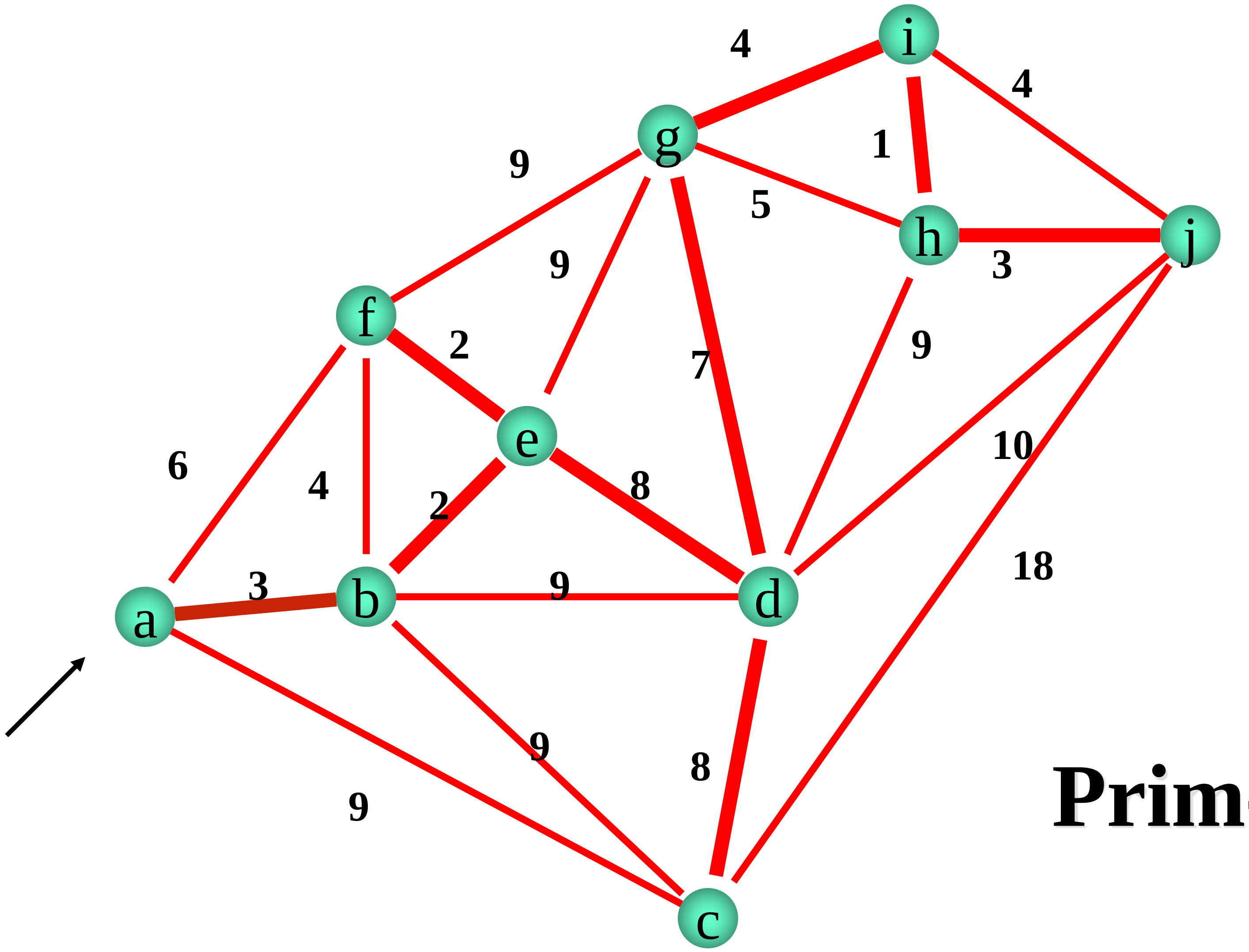
Pick an arbitrary vertex v in G
$D[v] \leftarrow 0$
for each vertex $u \neq v$ **do**
    $D[u] \leftarrow +\infty$
for each vertex u **do**
    Add ((u, null), $D[u]$) to Q // including v; here $D[u]$ is the key
while Q is not empty **do**
    $(u, e) \leftarrow Q$.removeMin()
    Add vertex u and edge e to T
    for each vertex z adjacent to u such that z is in Q **do**
        if $w((u, z)) < D[z]$ **then**
            $D[z] \leftarrow w((u, z))$
            Change z entry in Q to ((z, (u, z)), $D[z]$)
return T

D :distance vector, maintains reachable vertices

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



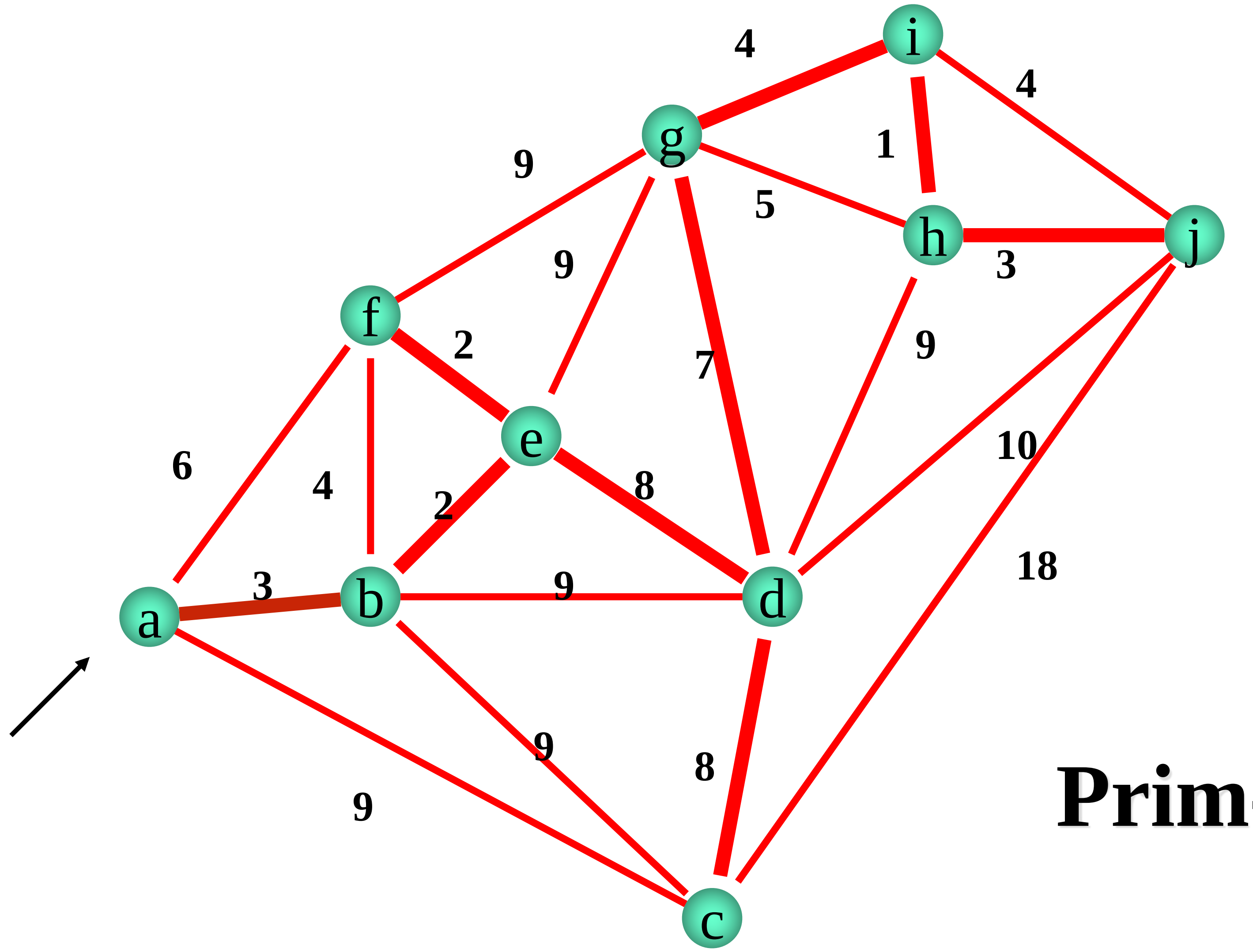
a
b
c
d
e
f
g
h
i
j
3
6
9
4
2
9
9
2
9
8
9
7
8
4
5
1
4
3
9
10
18



Prim-Jarník-Dijkstra

D

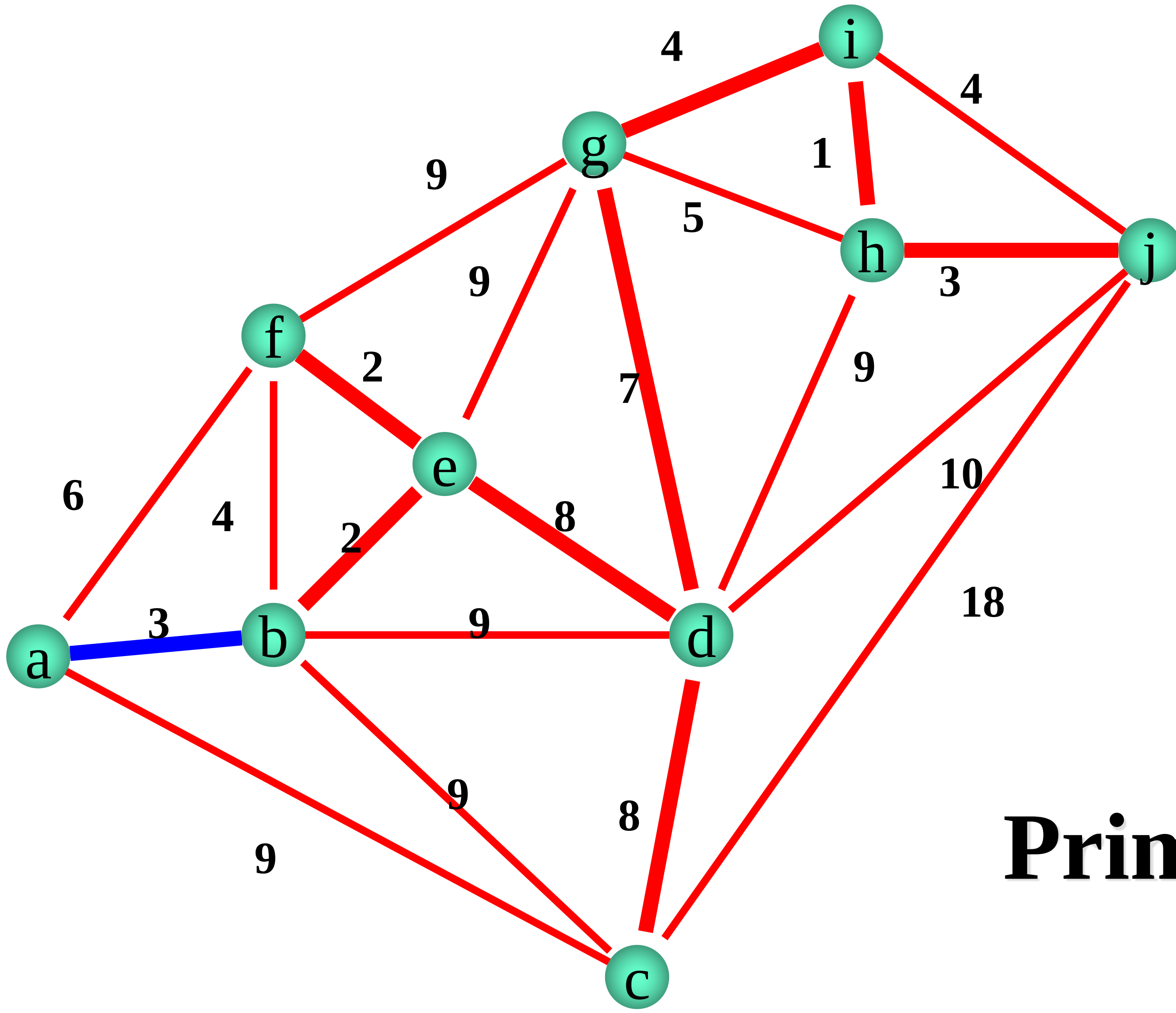
a	b	c	d	e	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞



Prim-Jarník-Dijkstra

D

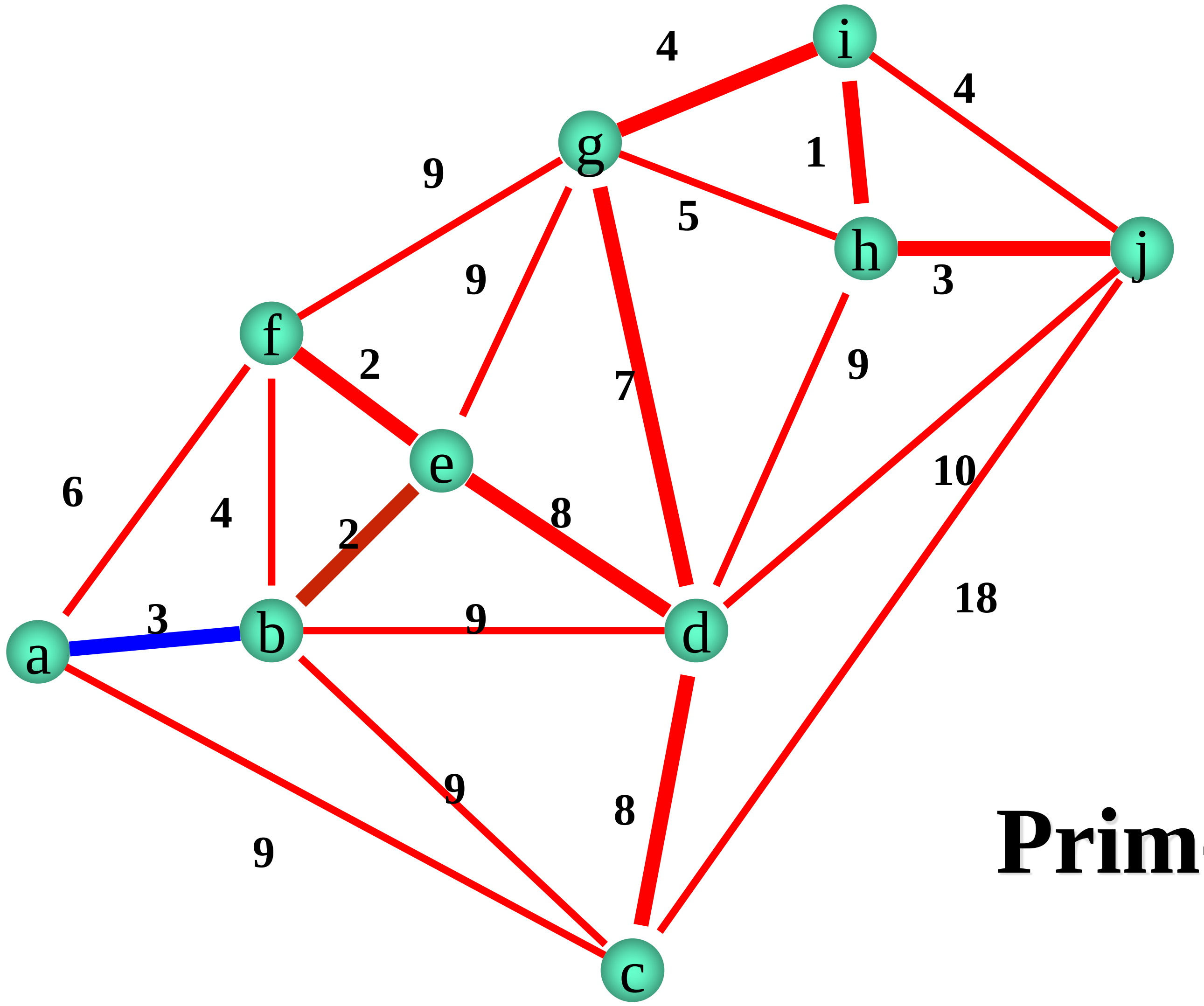
a	b	c	d	e	f	g	h	i	j
0	3	9	$+\infty$	$+\infty$	6	$+\infty$	$+\infty$	$+\infty$	$+\infty$



Prim-Jarník-Dijkstra

D

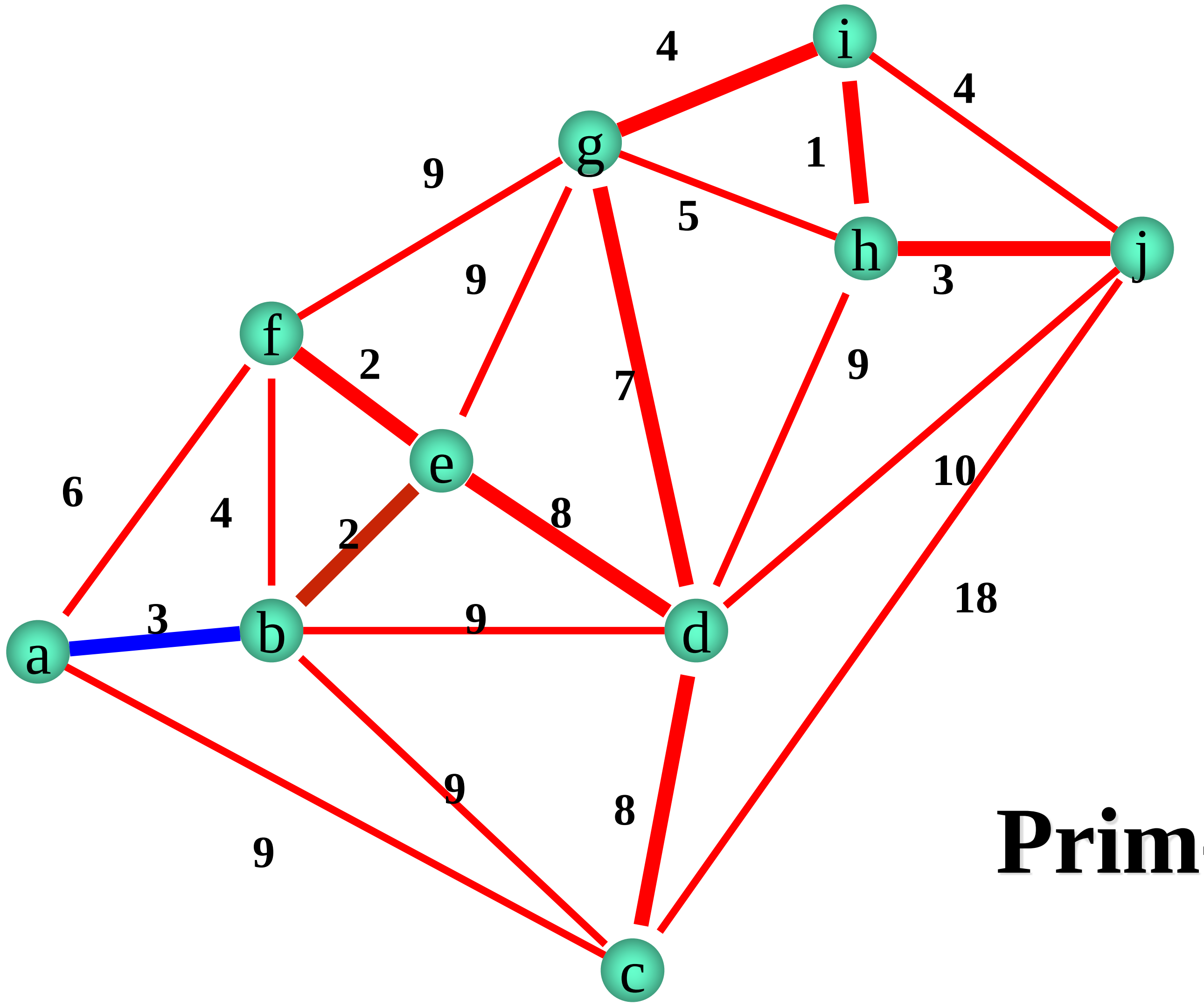
a	b	c	d	e	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞



Prim-Jarník-Dijkstra

D

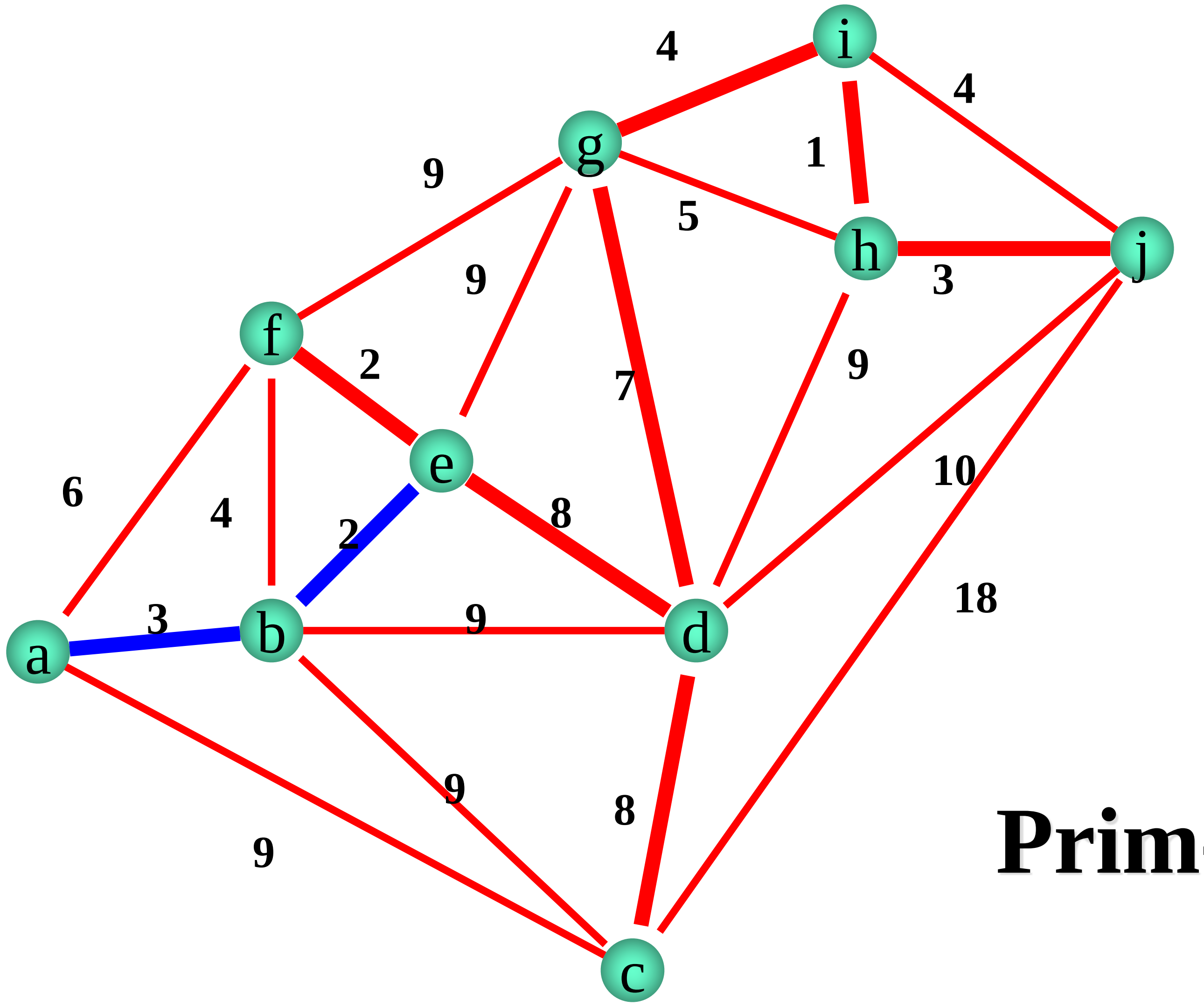
a	**b**	c	d	e	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞



Prim-Jarník-Dijkstra

D

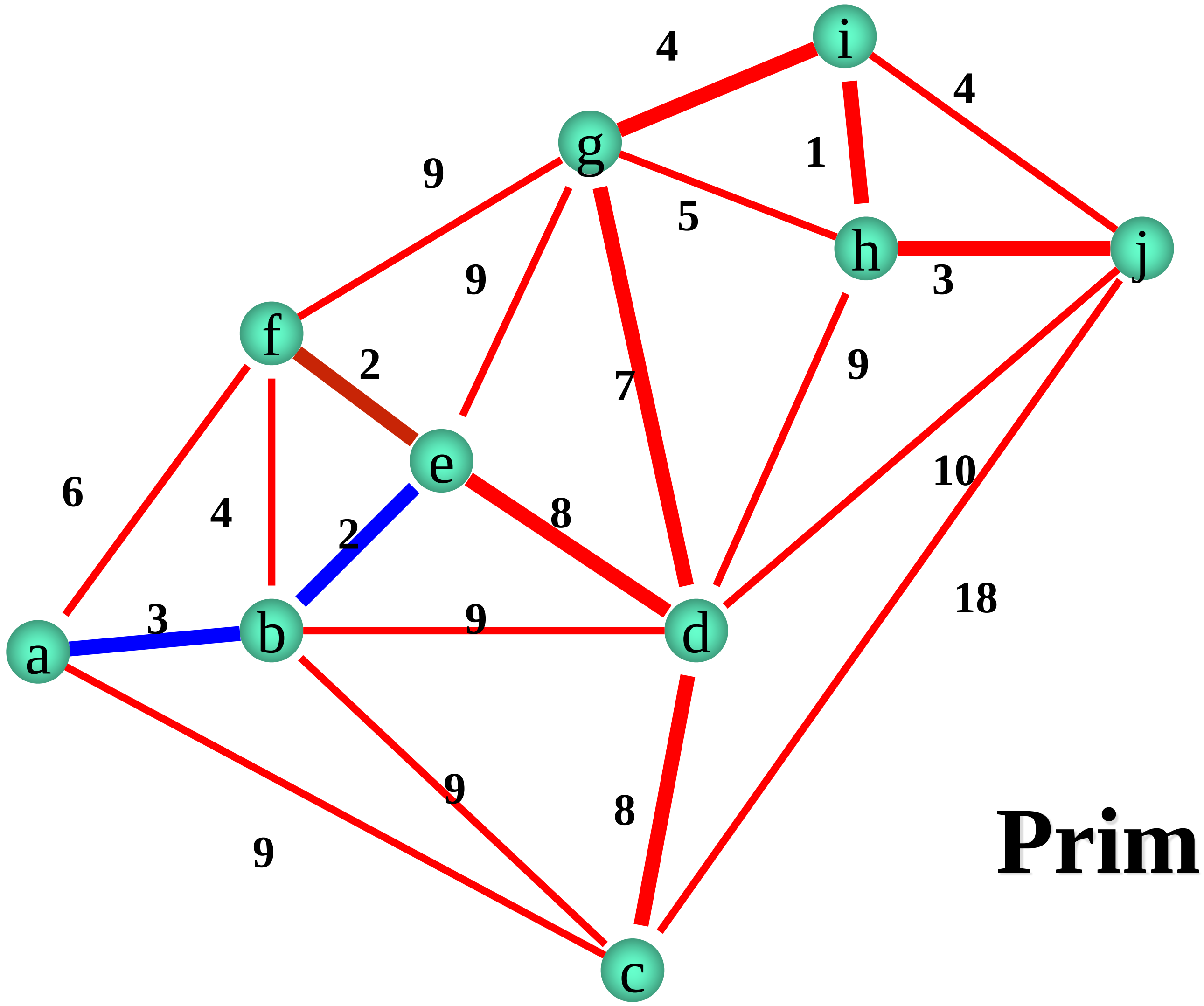
a	**b**	c	d	e	f	g	h	i	j
0	3	9	$+\infty$	$+\infty$	6	$+\infty$	$+\infty$	$+\infty$	$+\infty$
0	3	9	9	2	4	$+\infty$	$+\infty$	$+\infty$	$+\infty$



Prim-Jarník-Dijkstra

D

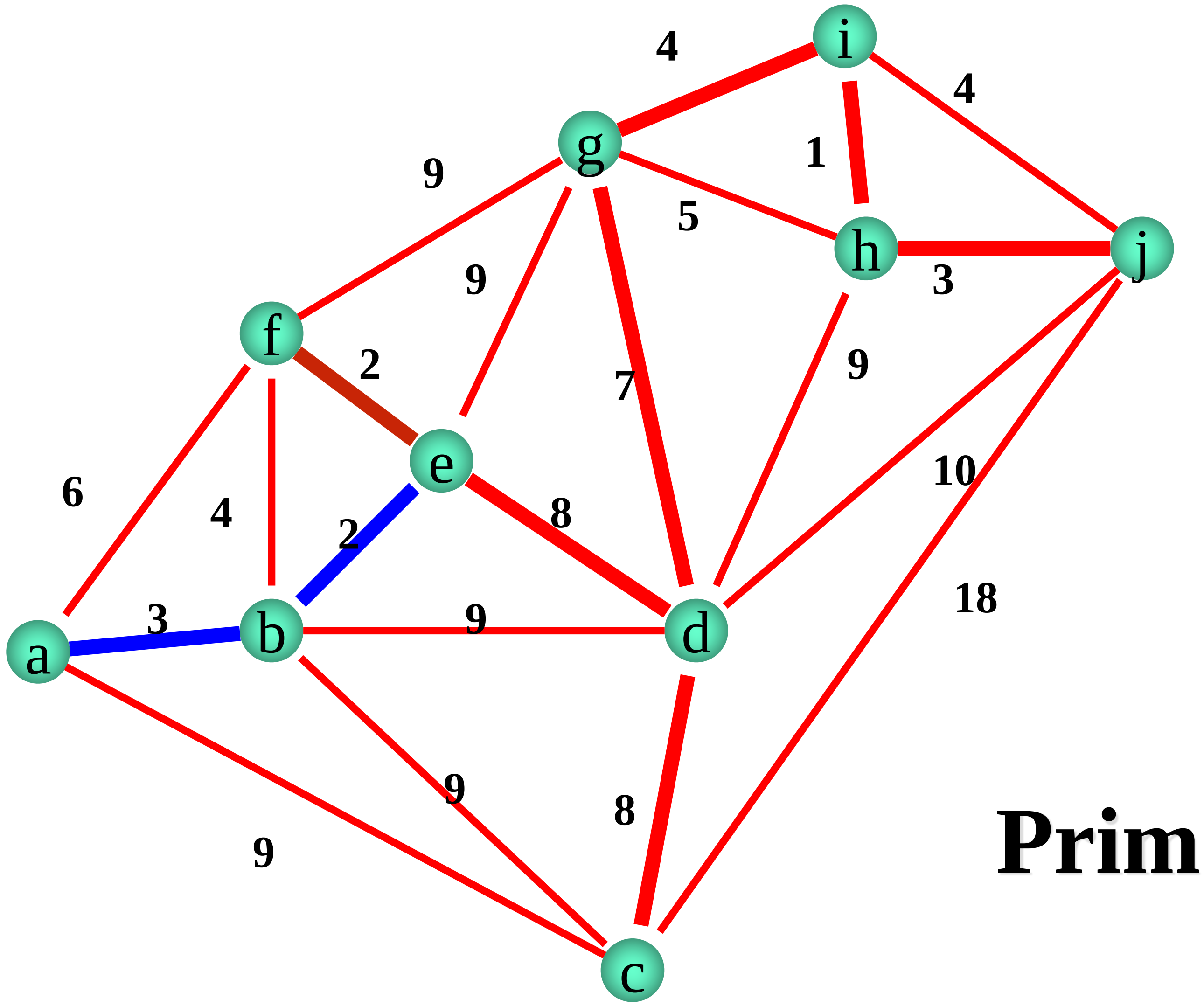
a	**b**	c	d	e	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞



Prim-Jarník-Dijkstra

D

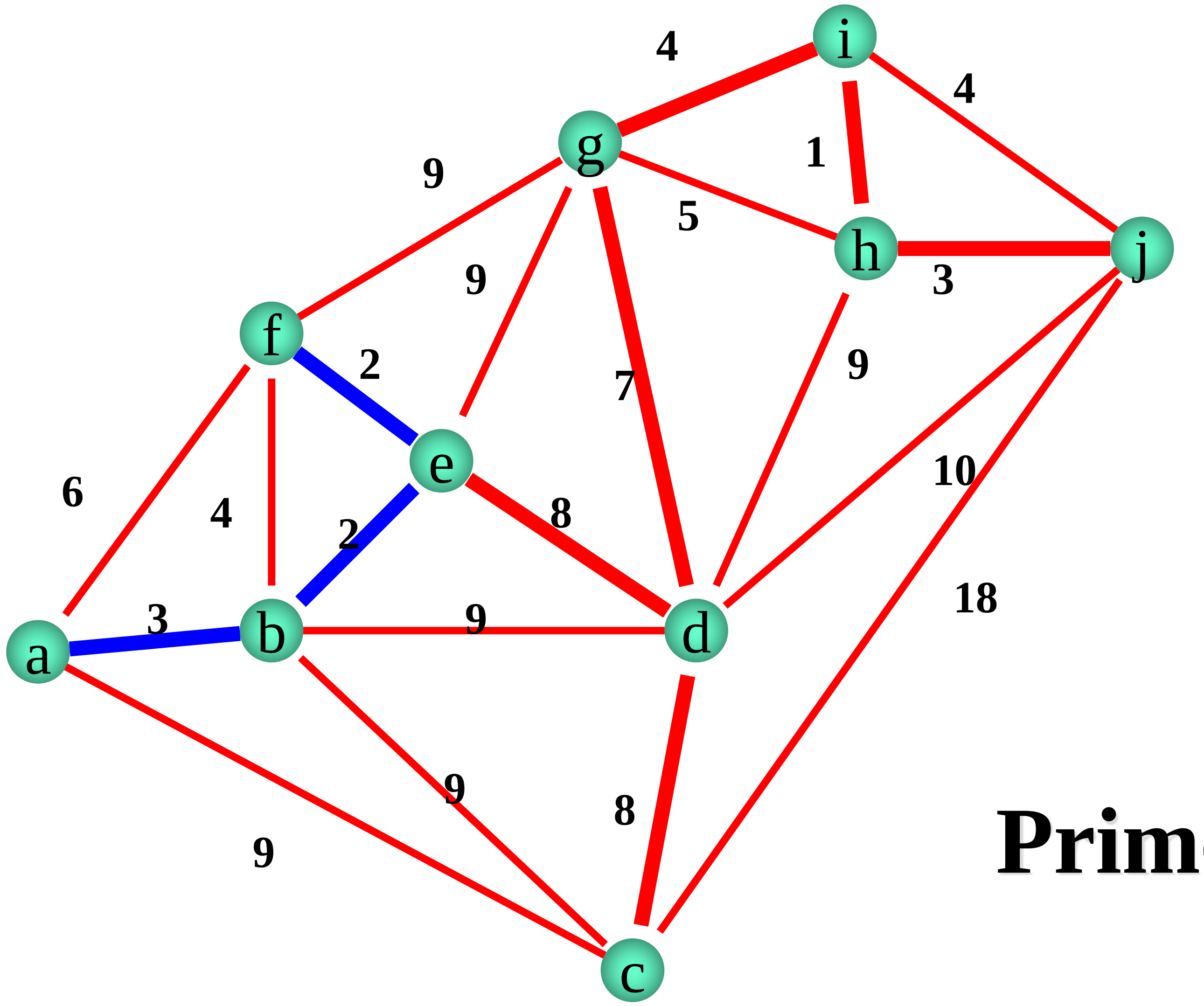
a	**b**	c	d	**e**	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞



Prim-Jarník-Dijkstra

D

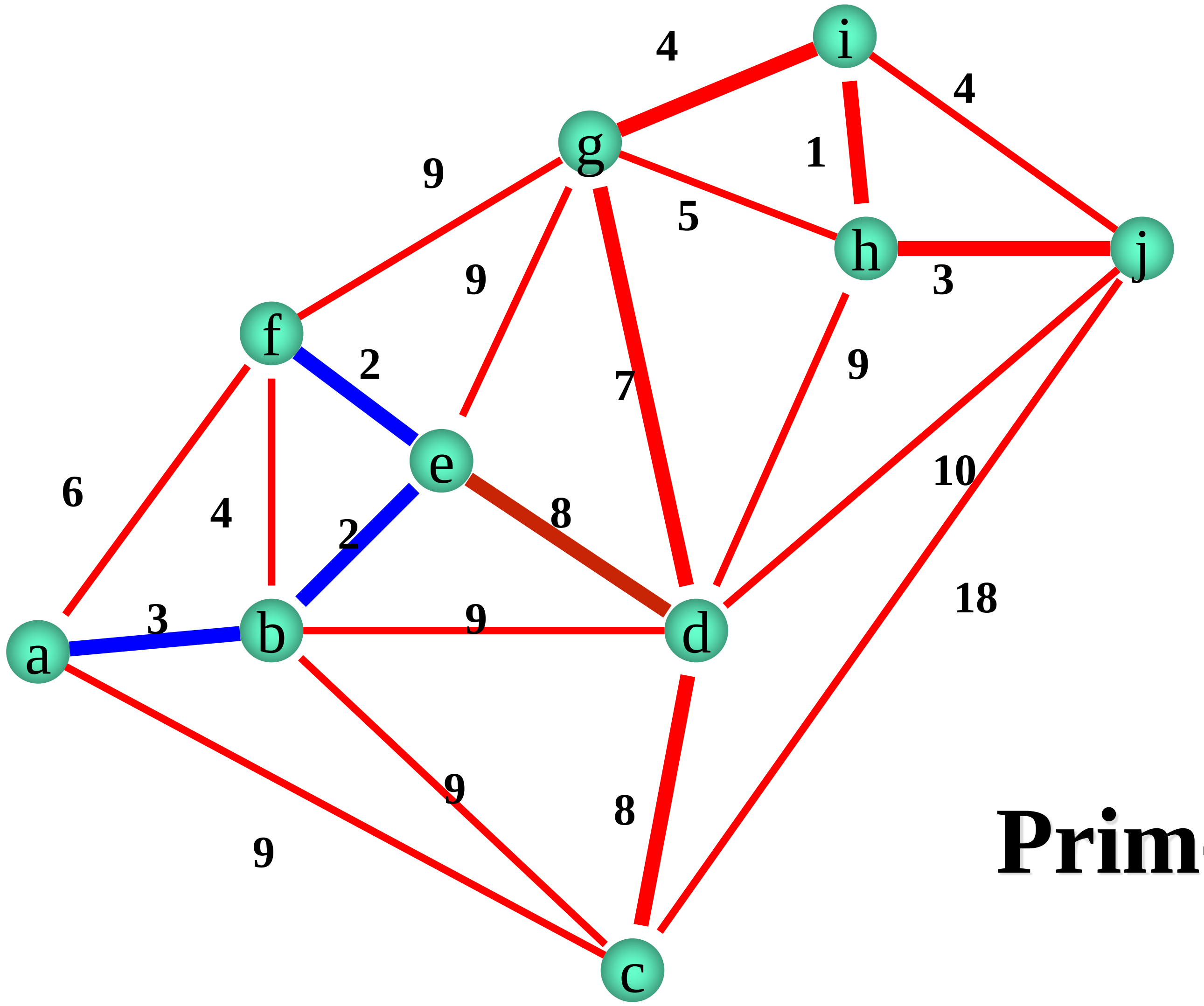
a	**b**	c	d	**e**	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞



Prim-Jarník-Dijkstra

D

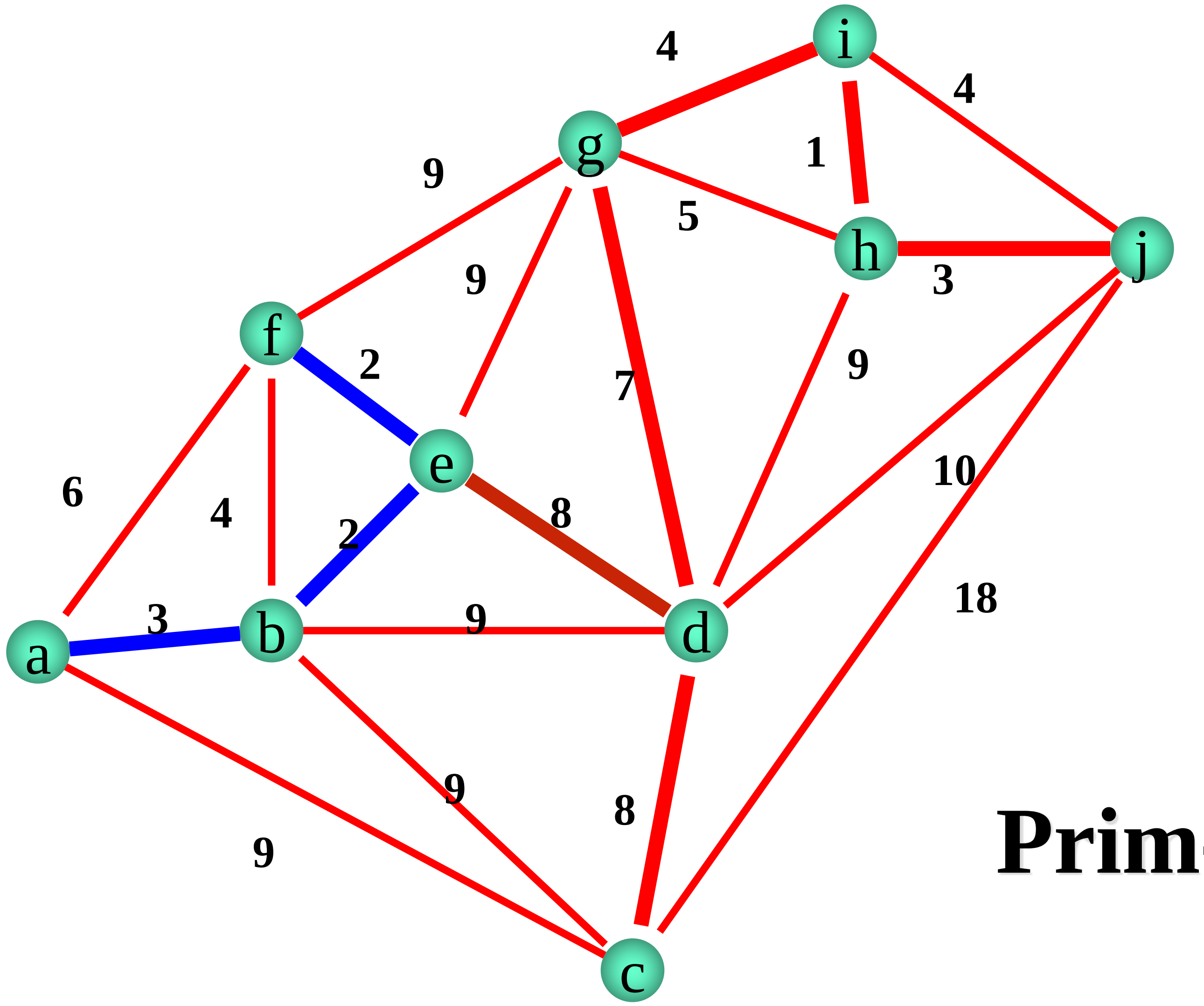
a	**b**	c	d	**e**	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞



Prim-Jarník-Dijkstra

D

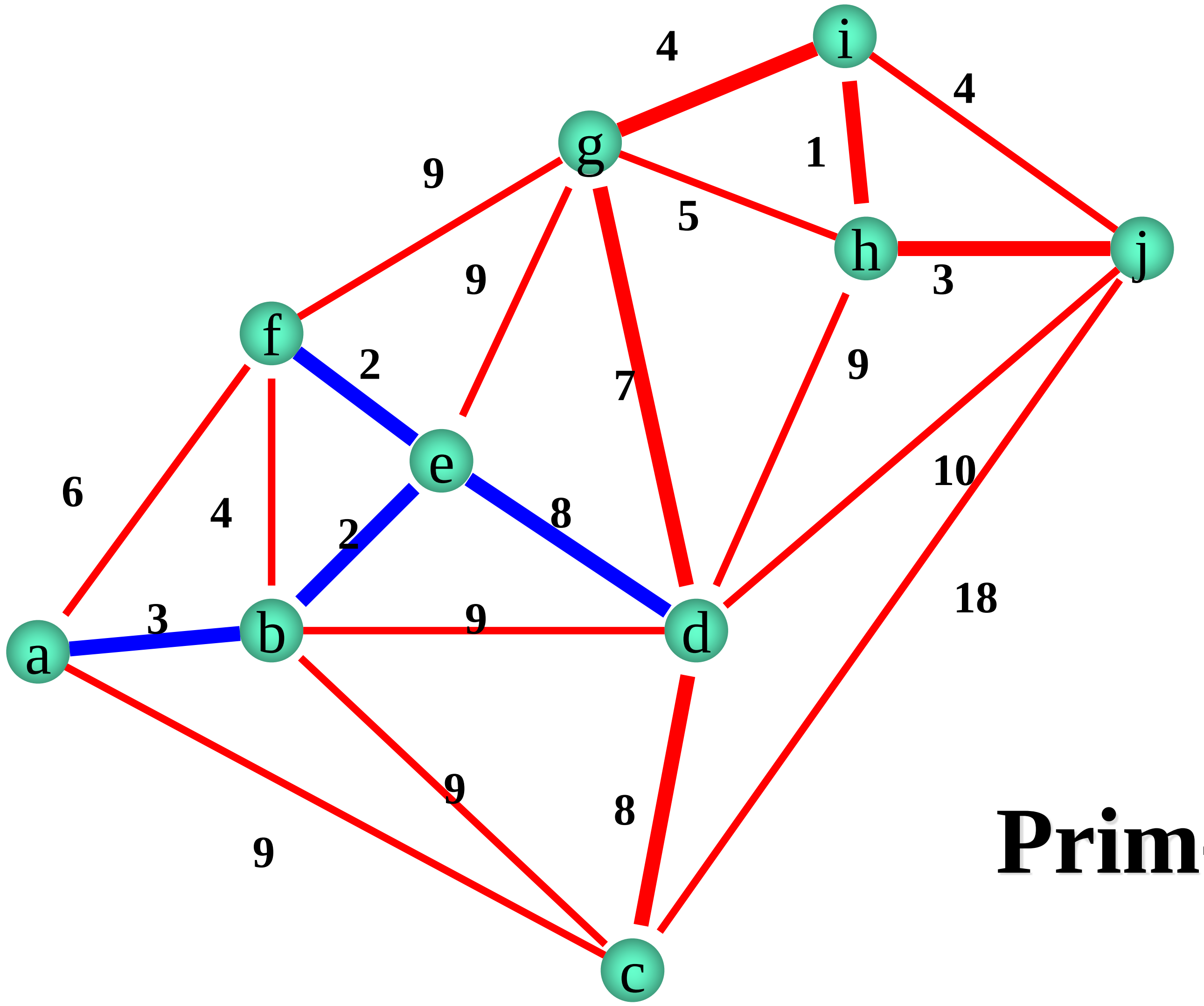
a	b	c	d	e	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞



Prim-Jarník-Dijkstra

D

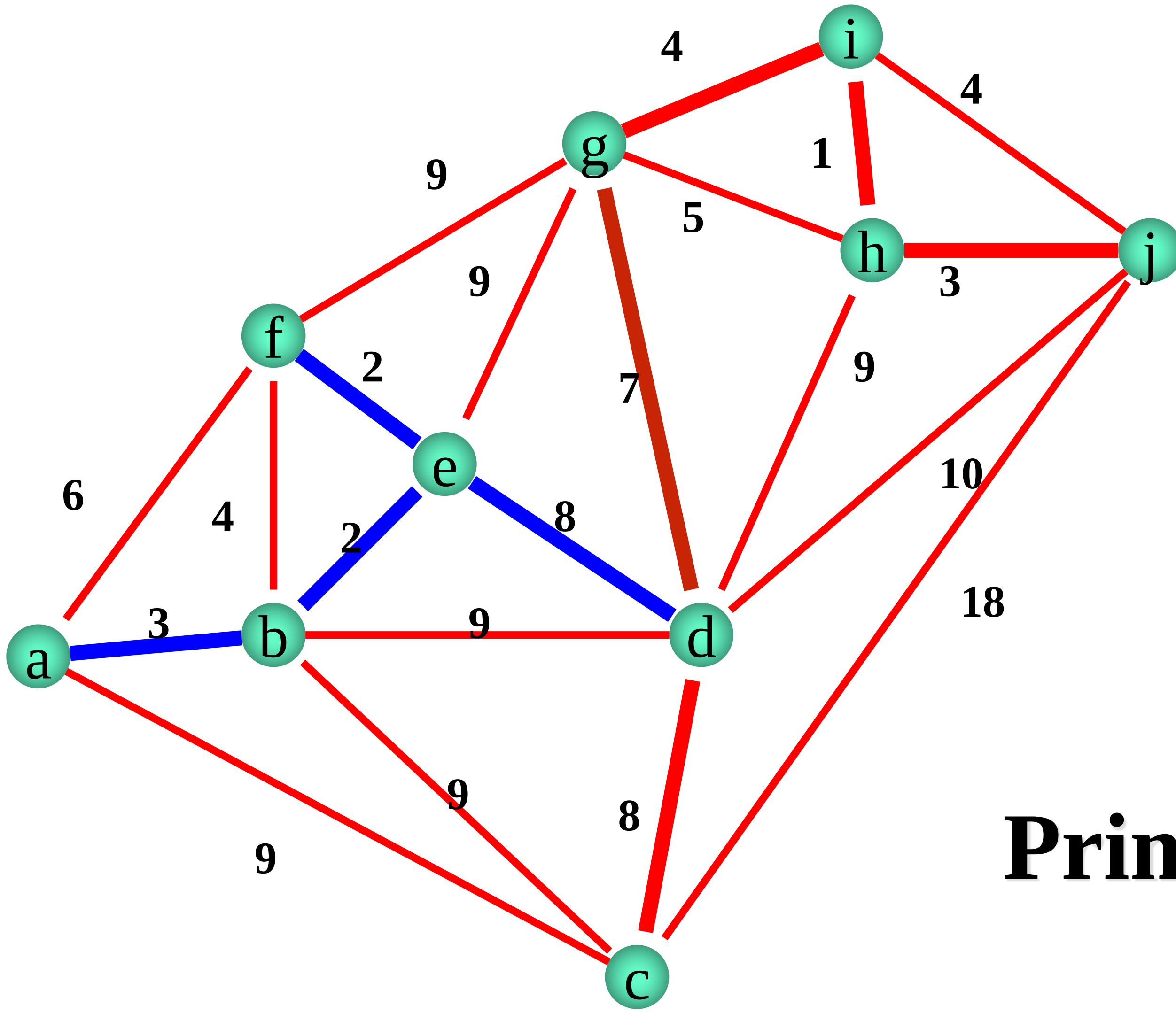
a	**b**	c	d	**e**	**f**	g	h	i	j
0	3	9	$+\infty$	$+\infty$	6	$+\infty$	$+\infty$	$+\infty$	$+\infty$
0	3	9	9	2	4	$+\infty$	$+\infty$	$+\infty$	$+\infty$
0	3	9	8	2	2	9	$+\infty$	$+\infty$	$+\infty$
0	3	9	8	2	2	9	$+\infty$	$+\infty$	$+\infty$



Prim-Jarník-Dijkstra

D

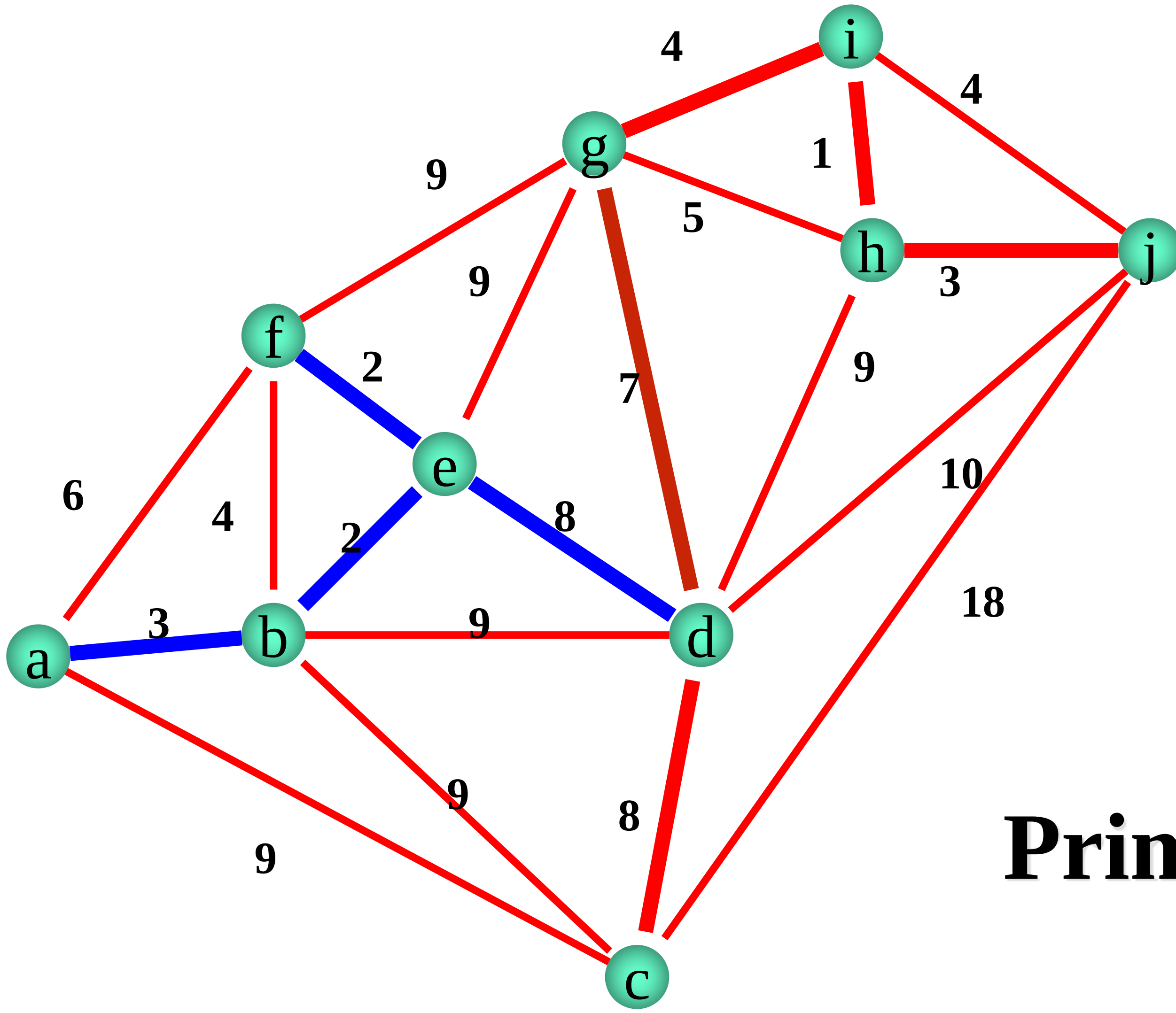
a	**b**	c	d	**e**	**f**	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞



Prim-Jarník-Dijkstra

D

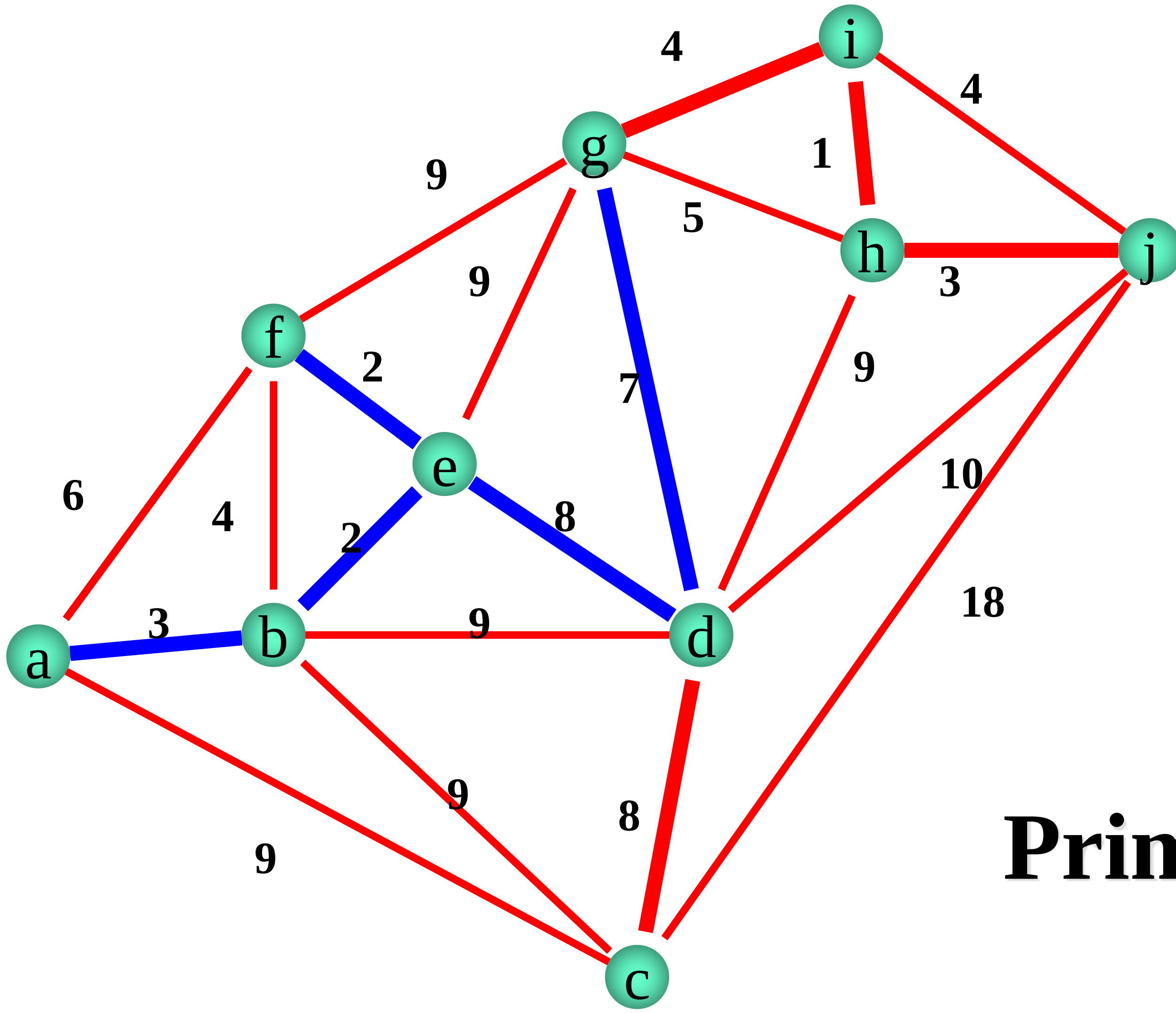
a	**b**	c	**d**	**e**	**f**	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10



Prim-Jarník-Dijkstra

D

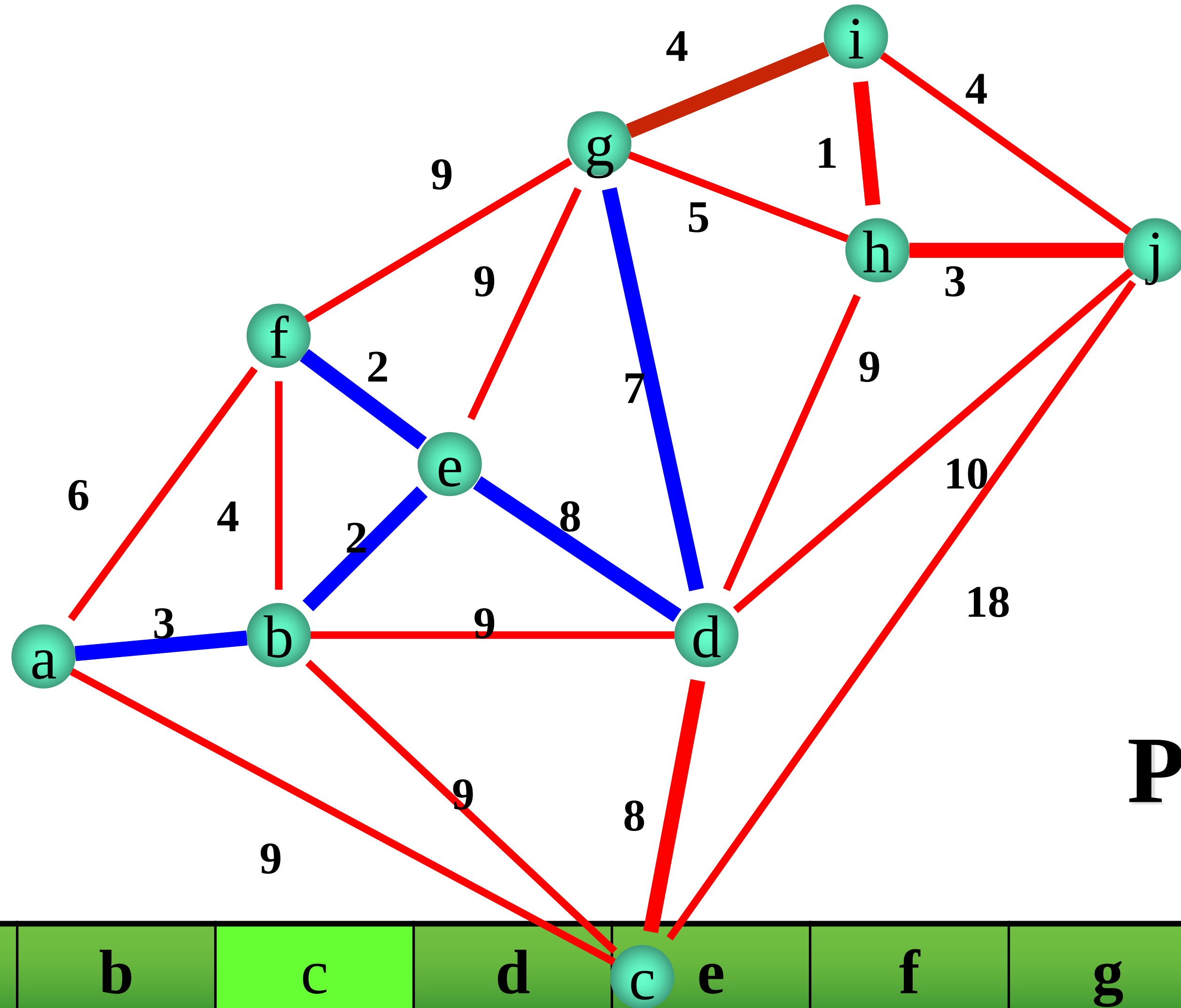
a	**b**	c	**d**	**e**	**f**	g	h	i	j
0	3	9	$+\infty$	$+\infty$	6	$+\infty$	$+\infty$	$+\infty$	$+\infty$
0	3	9	9	2	4	$+\infty$	$+\infty$	$+\infty$	$+\infty$
0	3	9	8	2	2	9	$+\infty$	$+\infty$	$+\infty$
0	3	9	8	2	2	9	9	$+\infty$	$+\infty$
0	3	8	8	2	2	7	9	$+\infty$	10



Prim-Jarník-Dijkstra

D

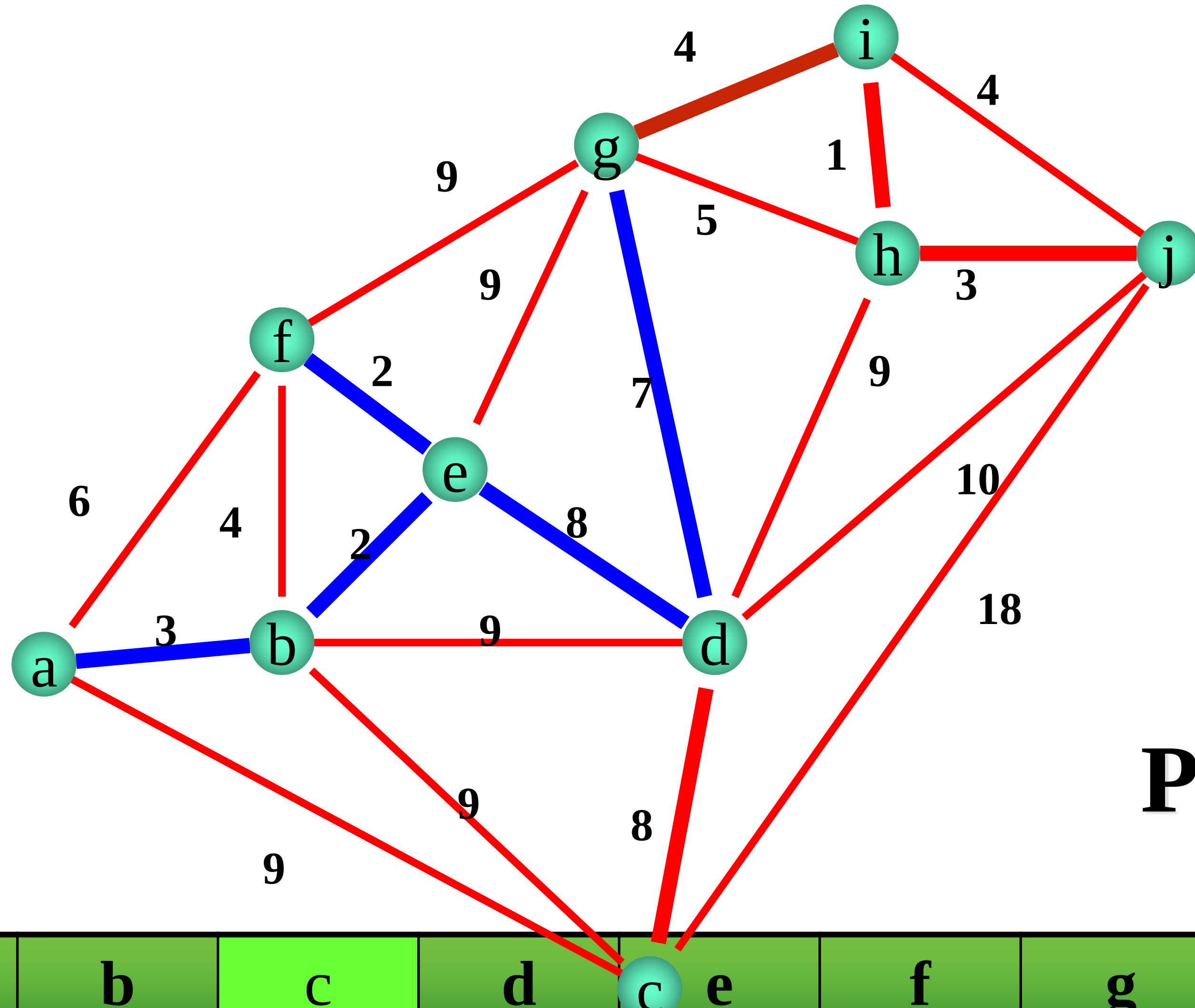
a	**b**	c	**d**	**e**	**f**	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10



Prim-Jarník-Dijkstra

D

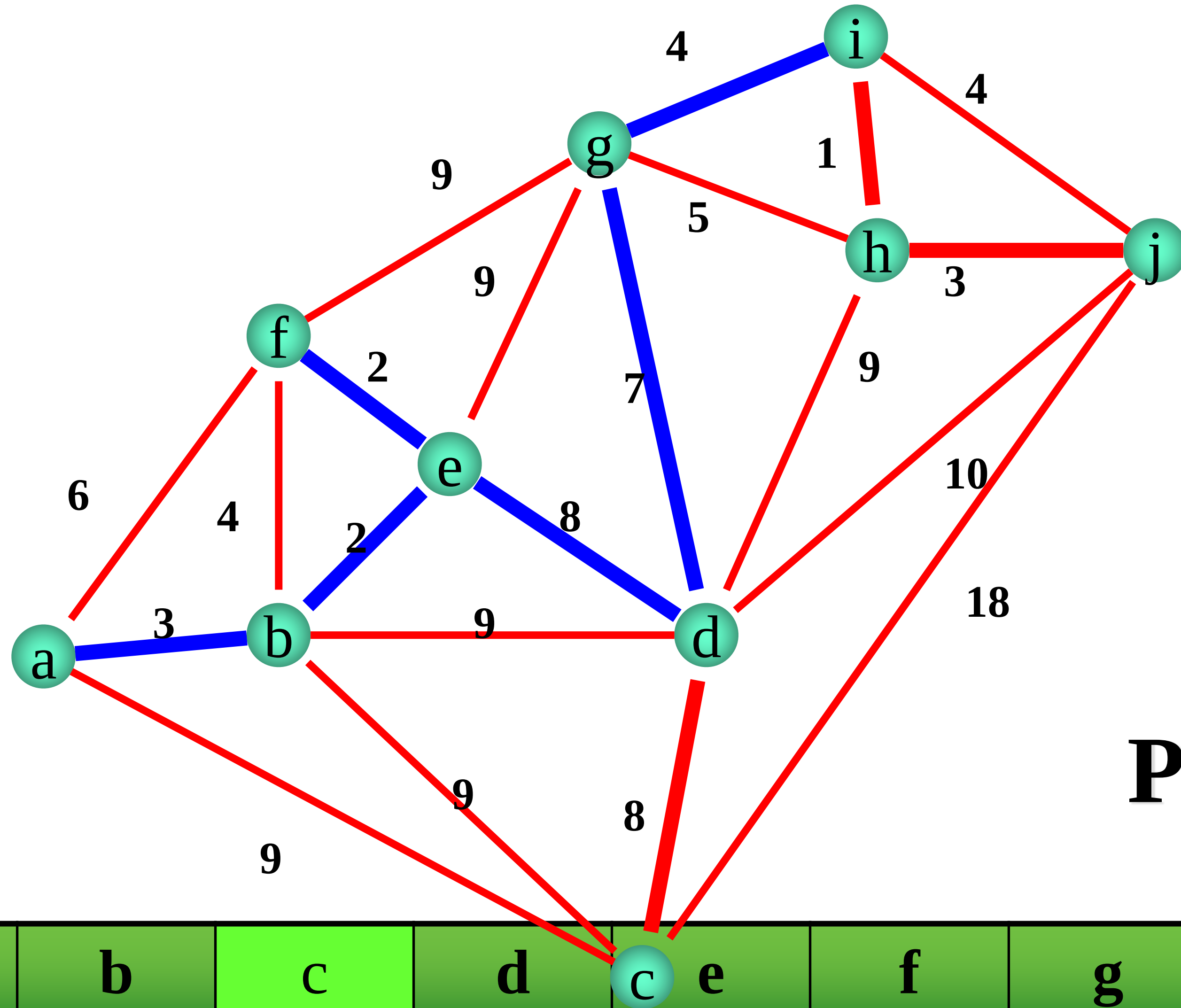
a	b	c	d	e	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10
0	3	8	8	2	2	7	5	4	10



Prim-Jarník-Dijkstra

D

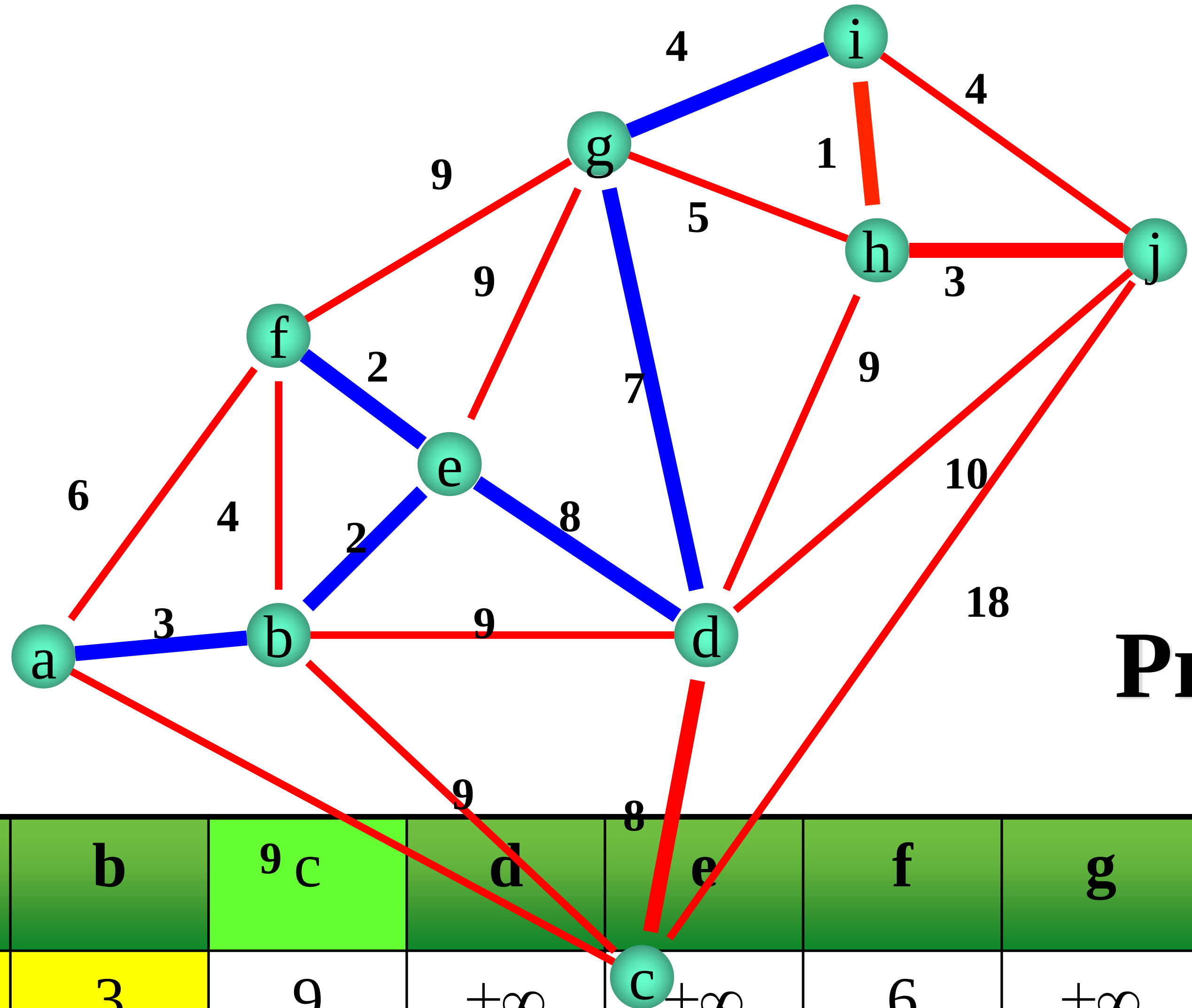
a	b	c	d	e	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10
0	3	8	8	2	2	7	5	4	10



Prim-Jarník-Dijkstra

D

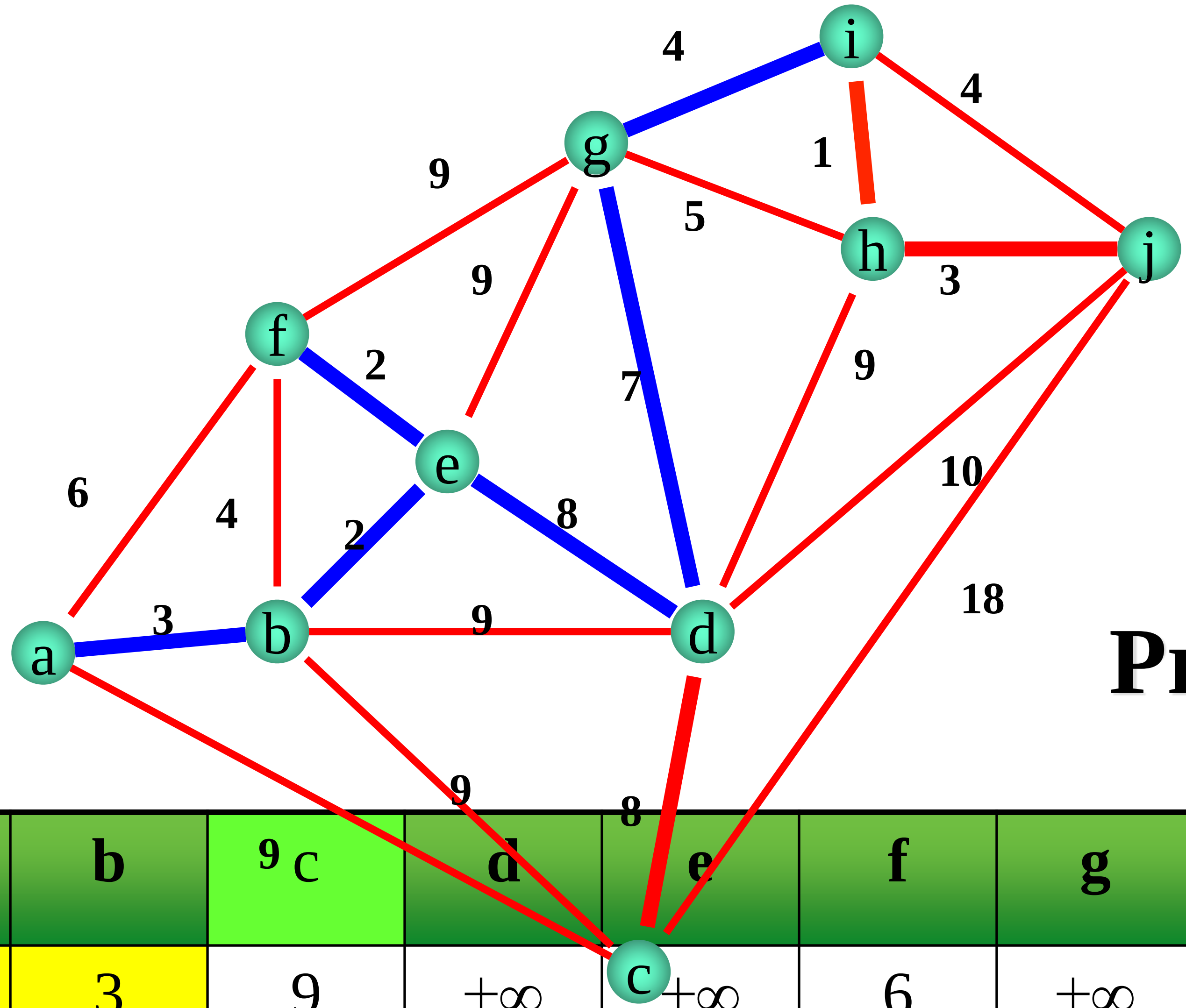
a	**b**	c	**d**	**e**	**f**	**g**	h	**i**	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10
0	3	8	8	2	2	7	5	4	10



Prim-Jarník-Dijkstra

D

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

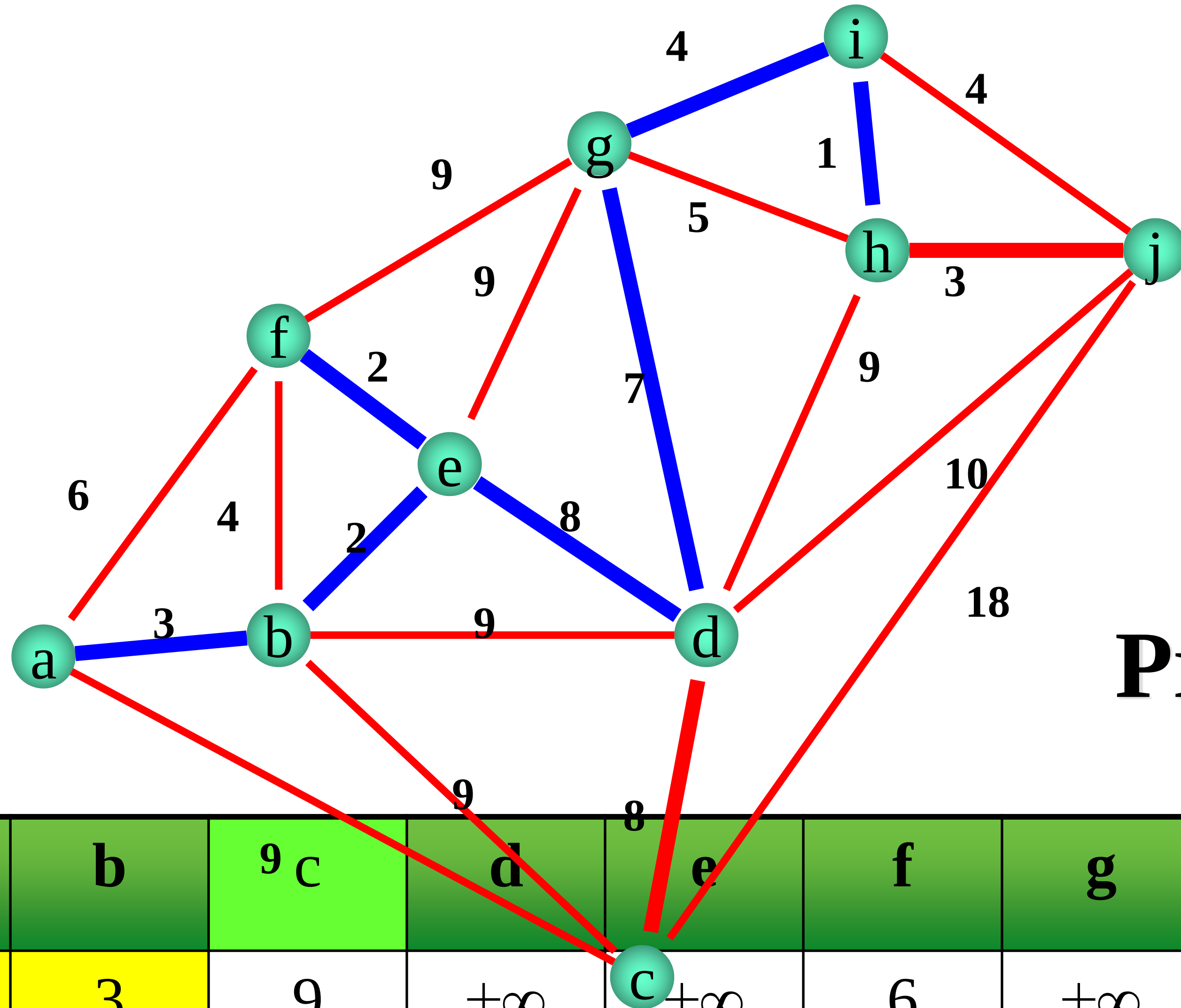


Prim-Jarník-Dijkstra

D

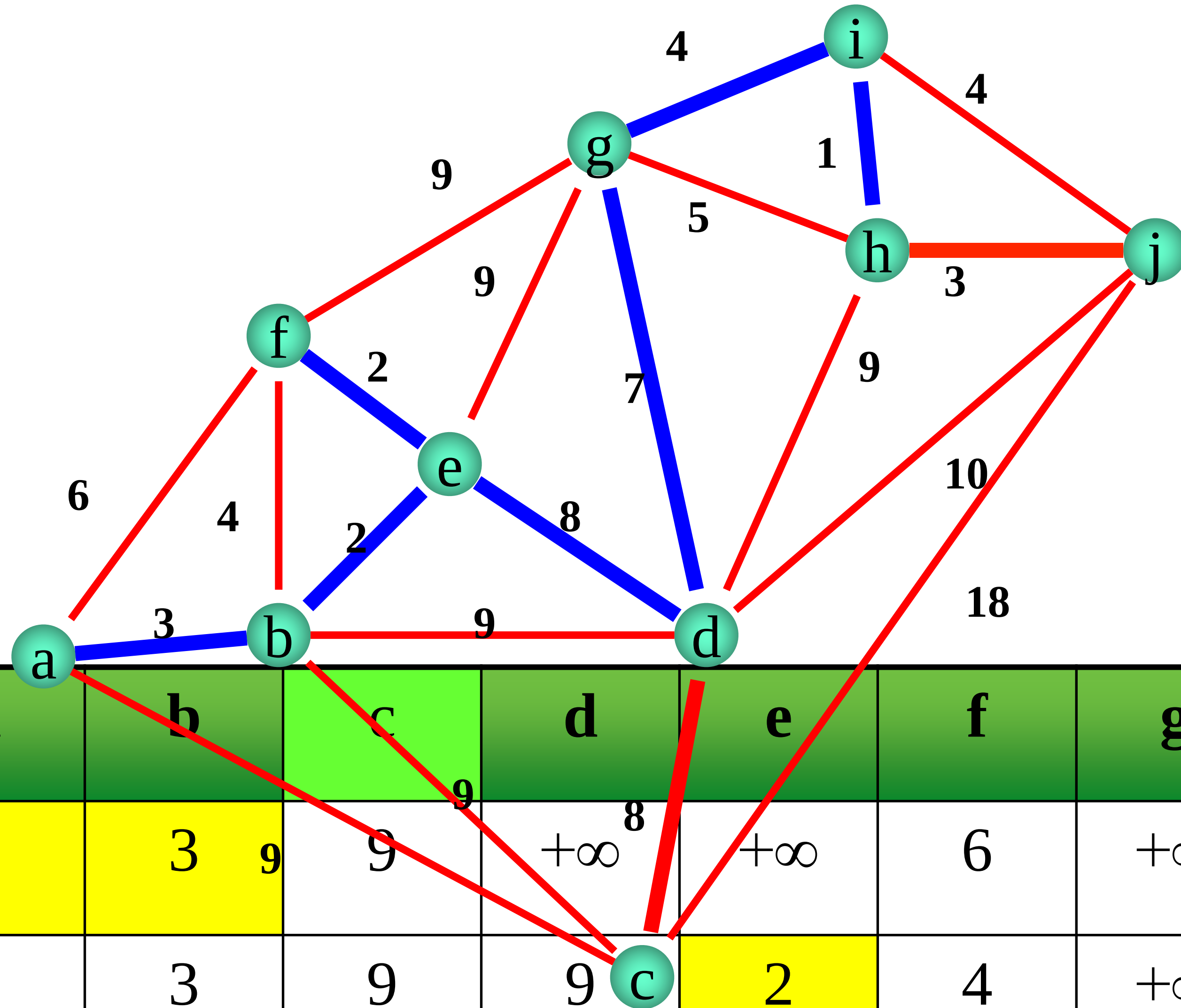
a	**b**	c	**d**	**e**	**f**	**g**	h	**i**	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10
0	3	8	8	2	2	7	5	4	10
0	3	8	8	2	2	7	1	4	4

Prim-Jarník-Dijkstra



D

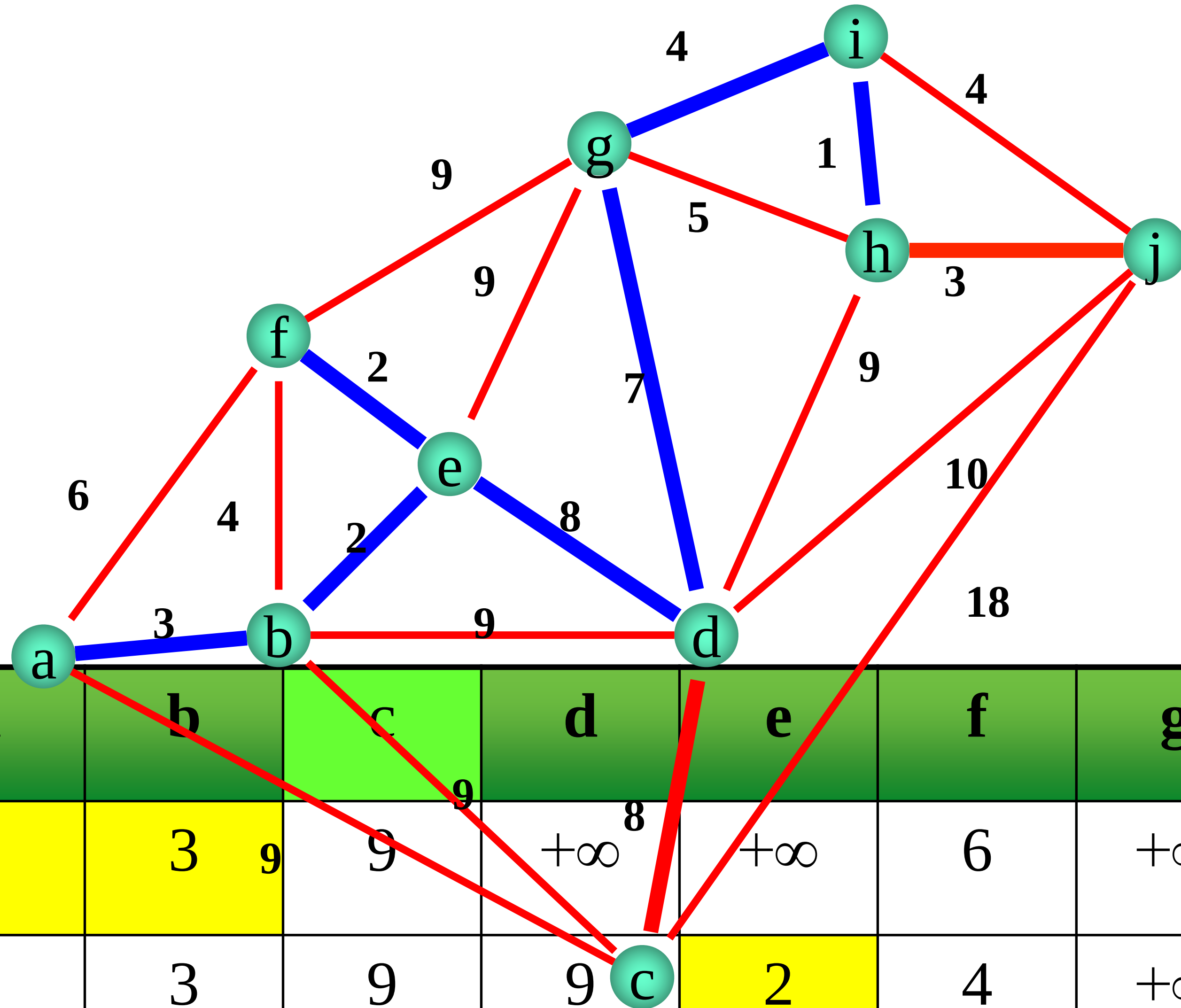
a	b	c	d	e	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10
0	3	8	8	2	2	7	5	4	10
0	3	8	8	2	2	7	1	4	4



Prim-Jarník-Dijkstra

D

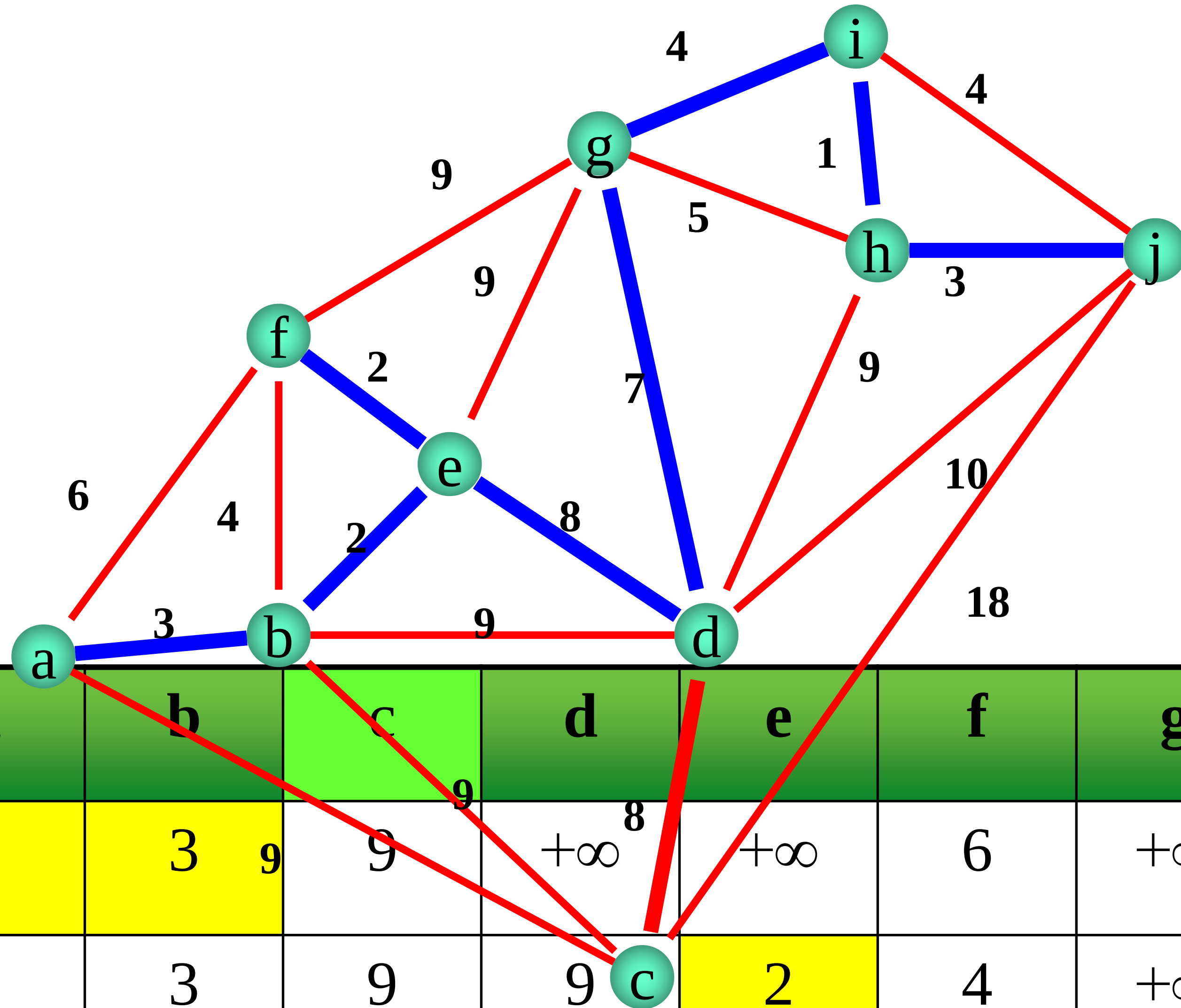
a	b	c	d	e	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10
0	3	8	8	2	2	7	1	4	4
0	3	8	8	2	2	7	1	4	4
0	3	8	8	2	2	7	1	4	3



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

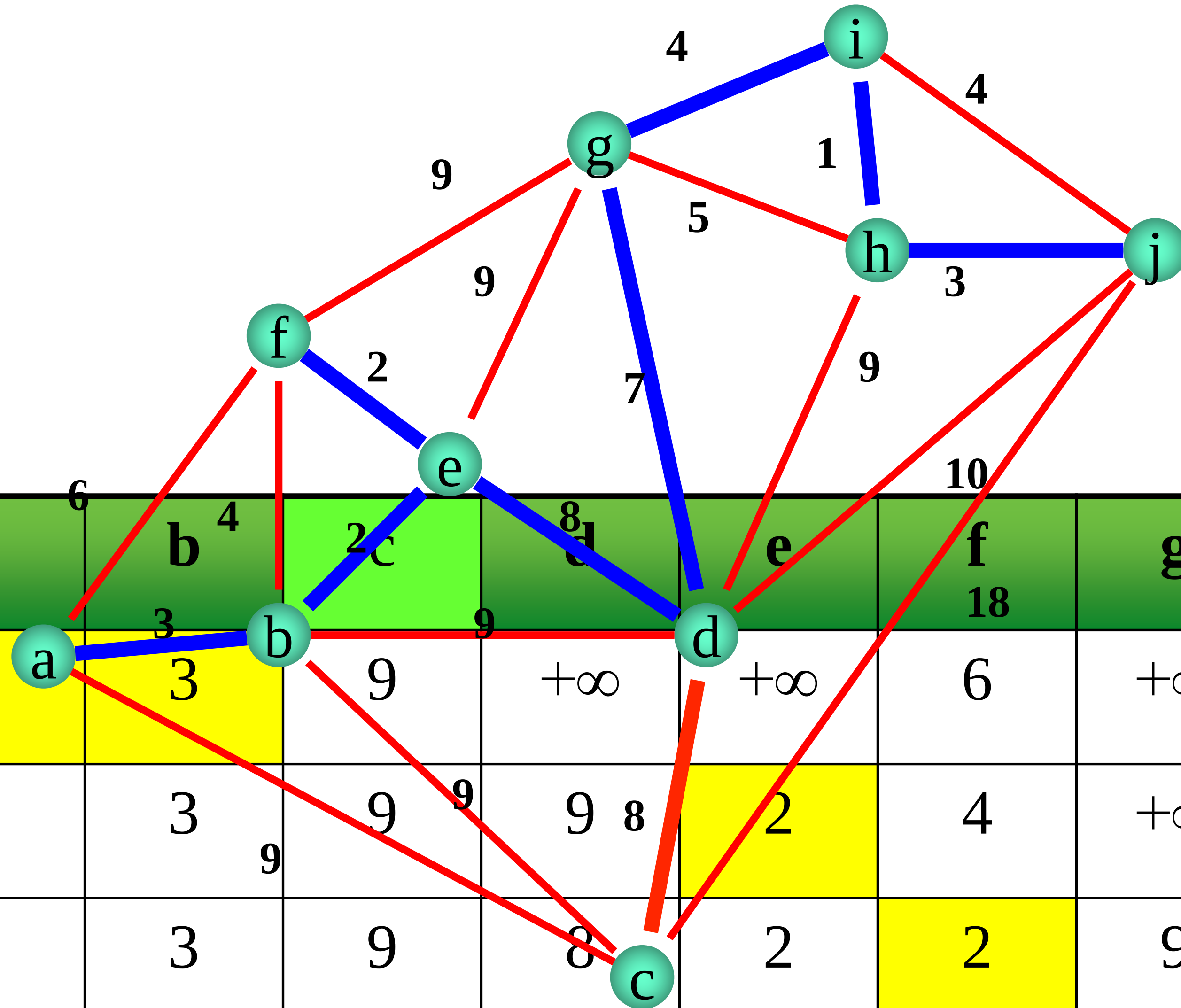


Prim-Jarník-Dijkstra

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

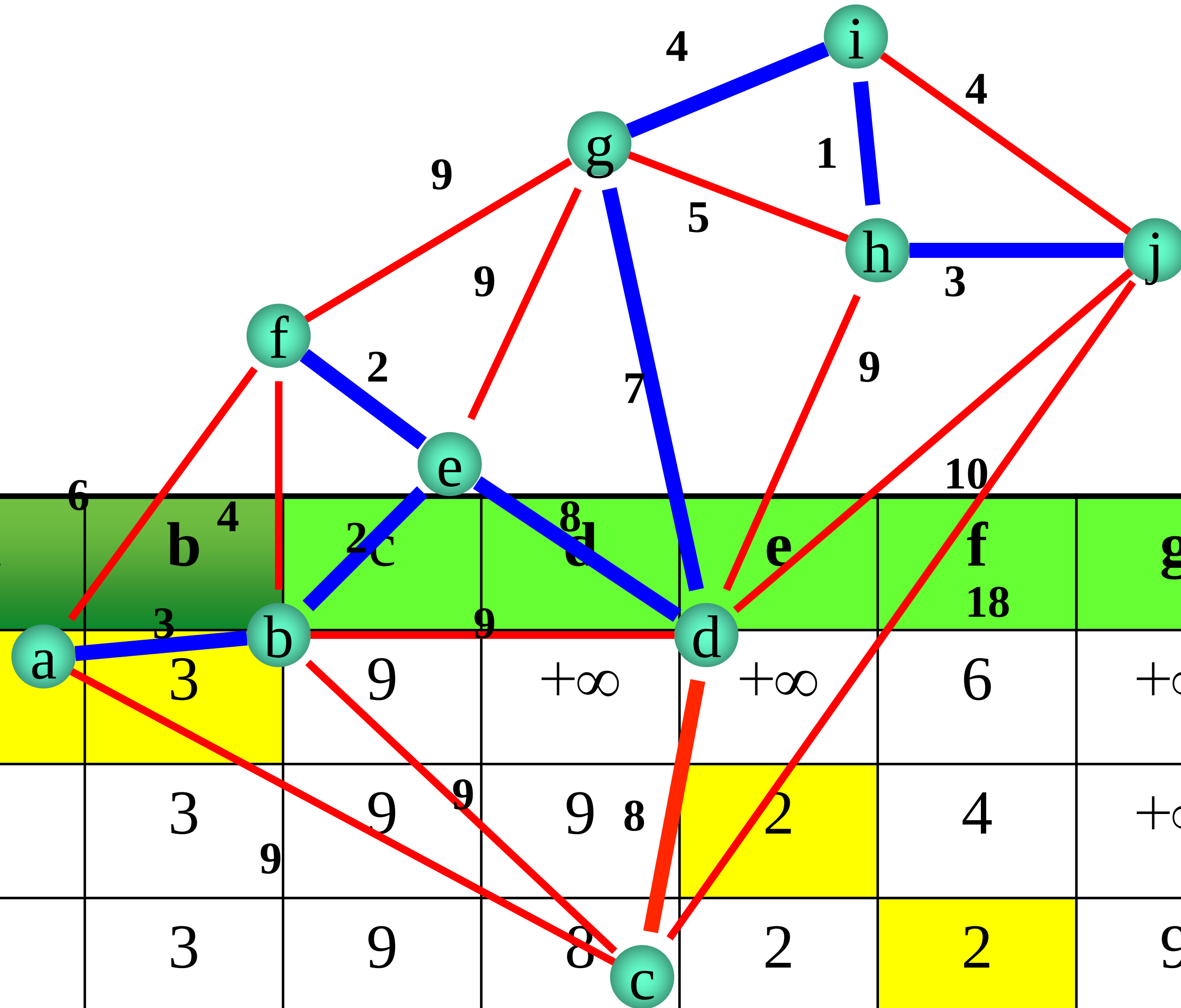
Prim-Jarník-Dijkstra



D

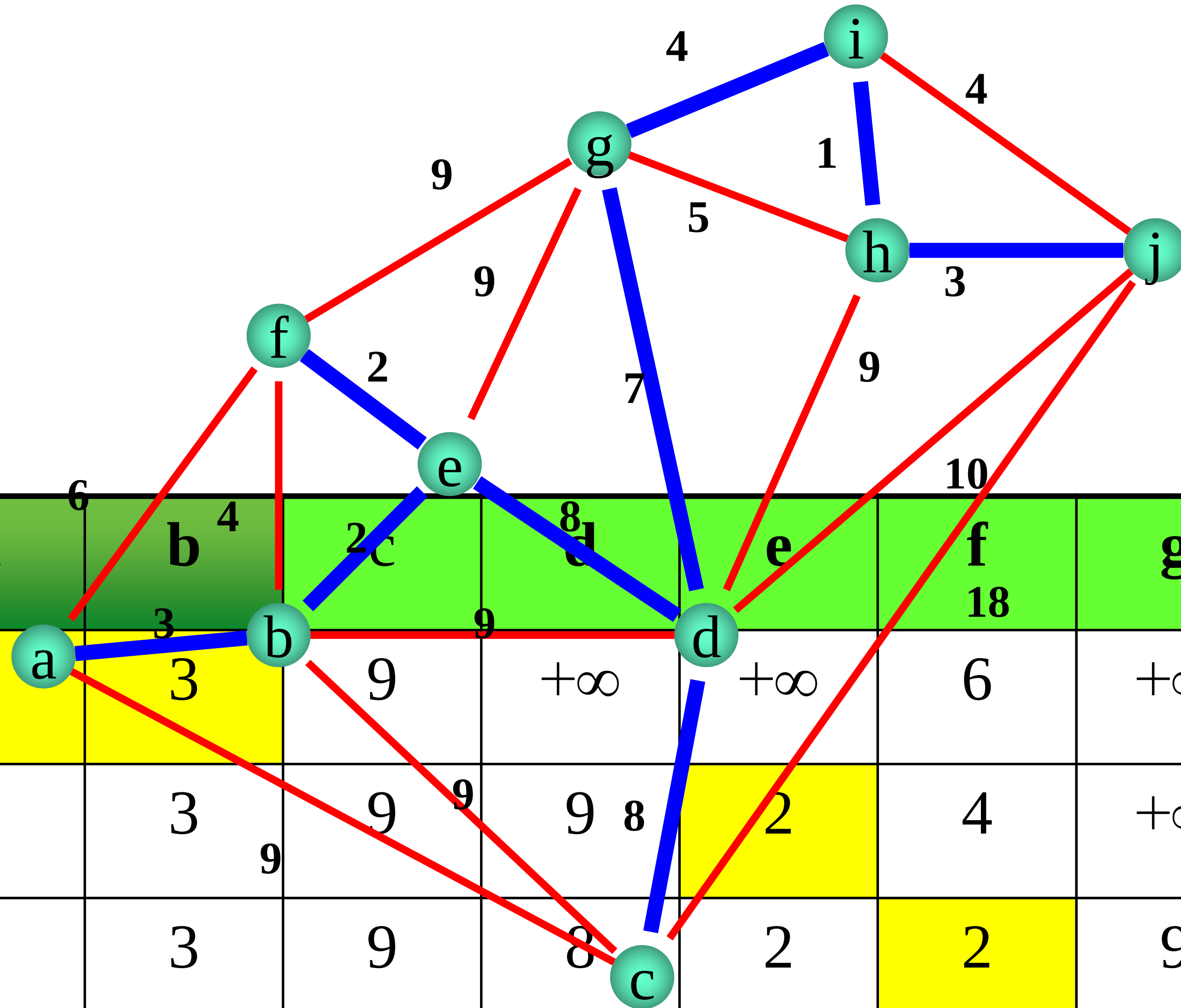
a	b	c	d	e	f	g	h	i	j
0	3	9	+∞	+∞	6	+∞	+∞	+∞	+∞
0	3	9	9	2	4	+∞	+∞	+∞	+∞
0	3	9	8	2	2	9	+∞	+∞	+∞
0	3	9	8	2	2	9	9	+∞	+∞
0	3	8	8	2	2	7	9	+∞	10
0	3	8	8	2	2	7	1	4	4
0	3	8	8	2	2	7	1	4	4
0	3	8	8	2	2	7	1	4	3
0	3	8	8	2	2	7	1	4	3

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

Algorithm PrimJarníkMST(G):
Input: A weighted connected graph G with n vertices and m edges
Output: an MST T for G
Data structures: Array D; Priority Queue Q; and tree T

```
Pick an arbitrary vertex v in G
D[v] ← 0
for each vertex u ≠ v do
    D[u] ← +∞
for each vertex u do
    Add ((u, null), D[u]) to Q // including v; here D[u] is the key
while Q is not empty do
    (u, e) ← Q.removeMin()
    Add vertex u and edge e to T
    for each vertex z adjacent to u such that z is in Q do
        if w((u, z)) < D[z] then
            D[z] ← w((u, z))
            Change z entry in Q to ((z, (u, z)), D[z])
return T
```

D :distance vector, maintains reachable vertices

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

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