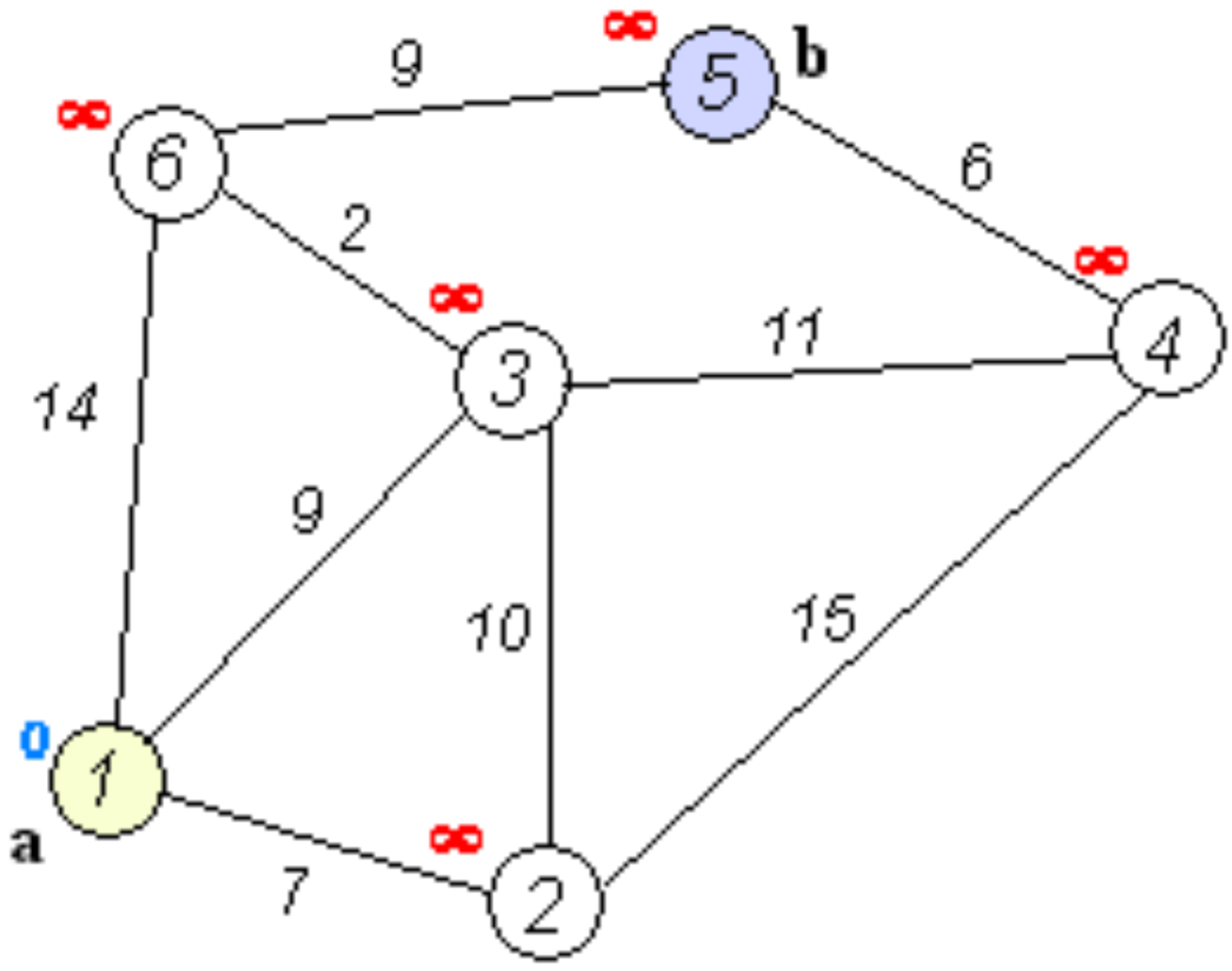
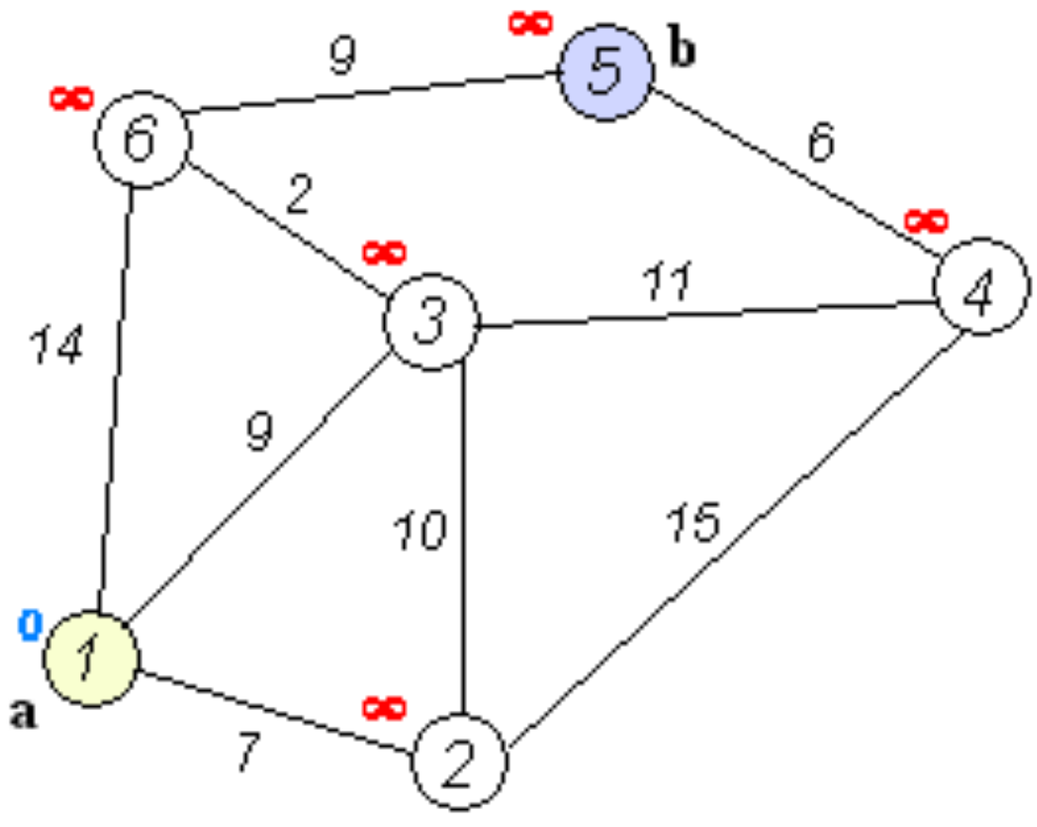
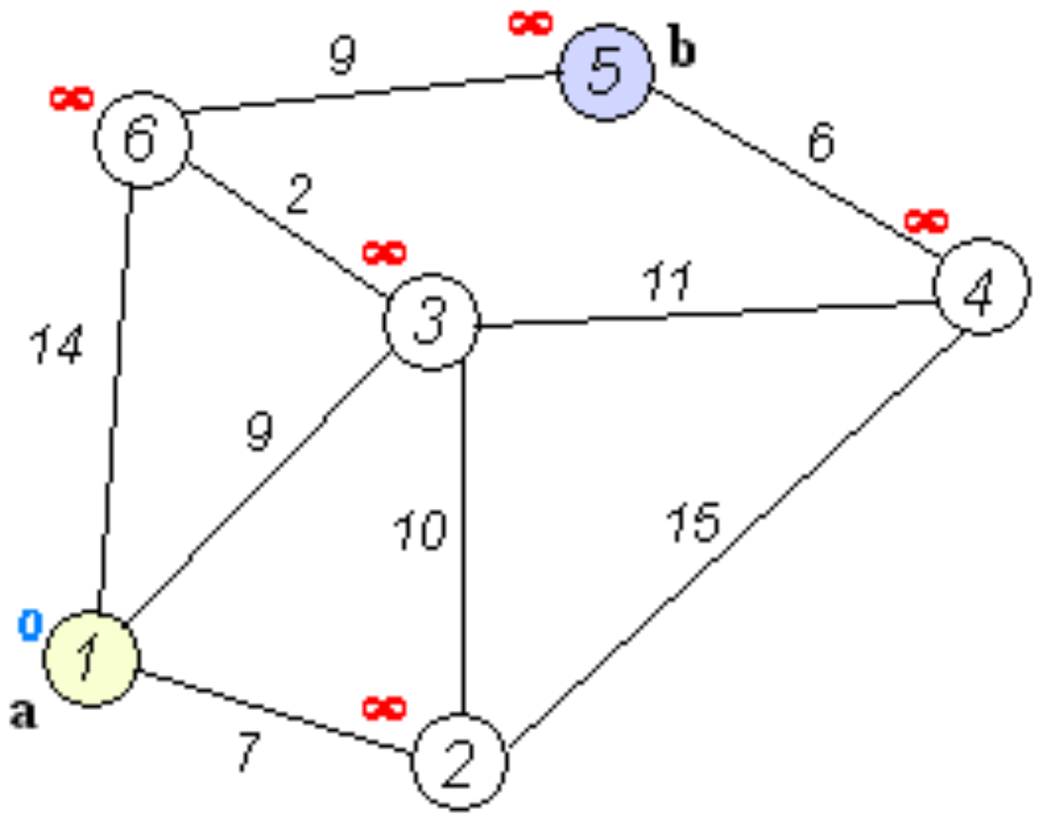

∞
9
5
b
∞
6
6
2
∞
∞
11
4
3
14
9
10
15
0
1
∞
a
7
2

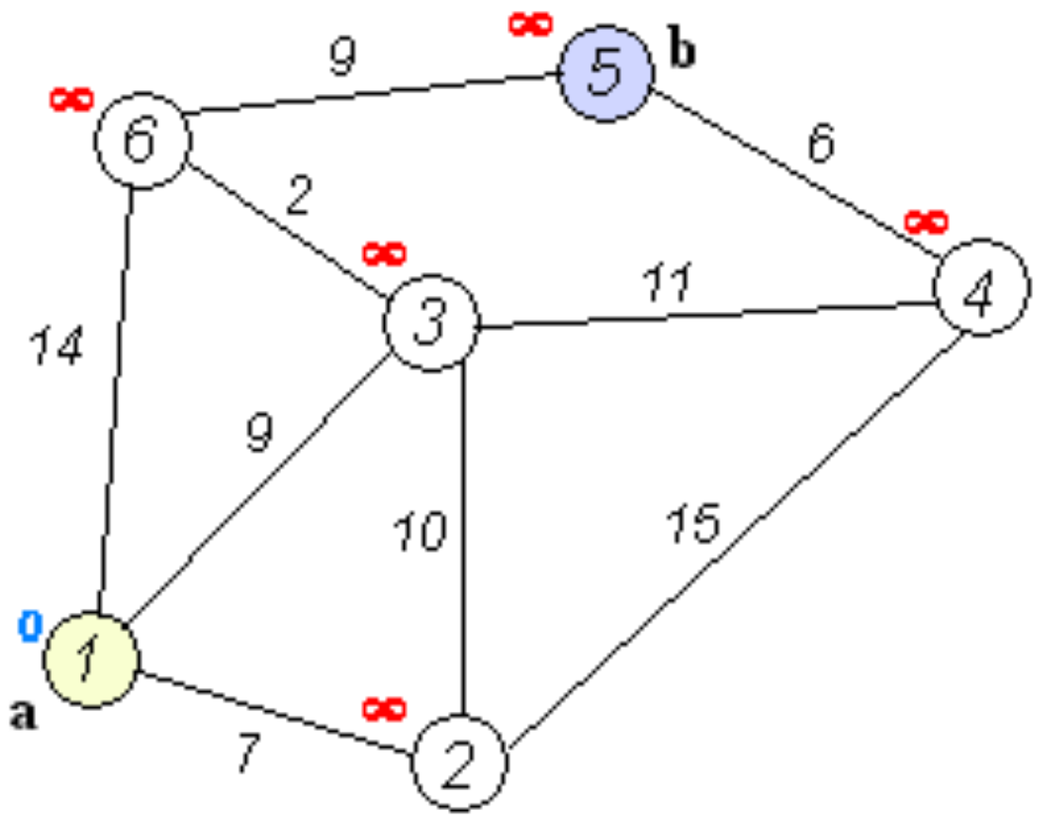
```
1   function Dijkstra(Graph, source):
2       dist[source] ← 0                                          // Initialization
3
4       create vertex set Q
5
6       for each vertex v in Graph:
7           if v ≠ source
8               dist[v] ← INFINITY                                // Unknown distance from so
9               prev[v] ← UNDEFINED                               // Predecessor of v
10
11          Q.add_with_priority(v, dist[v])
```



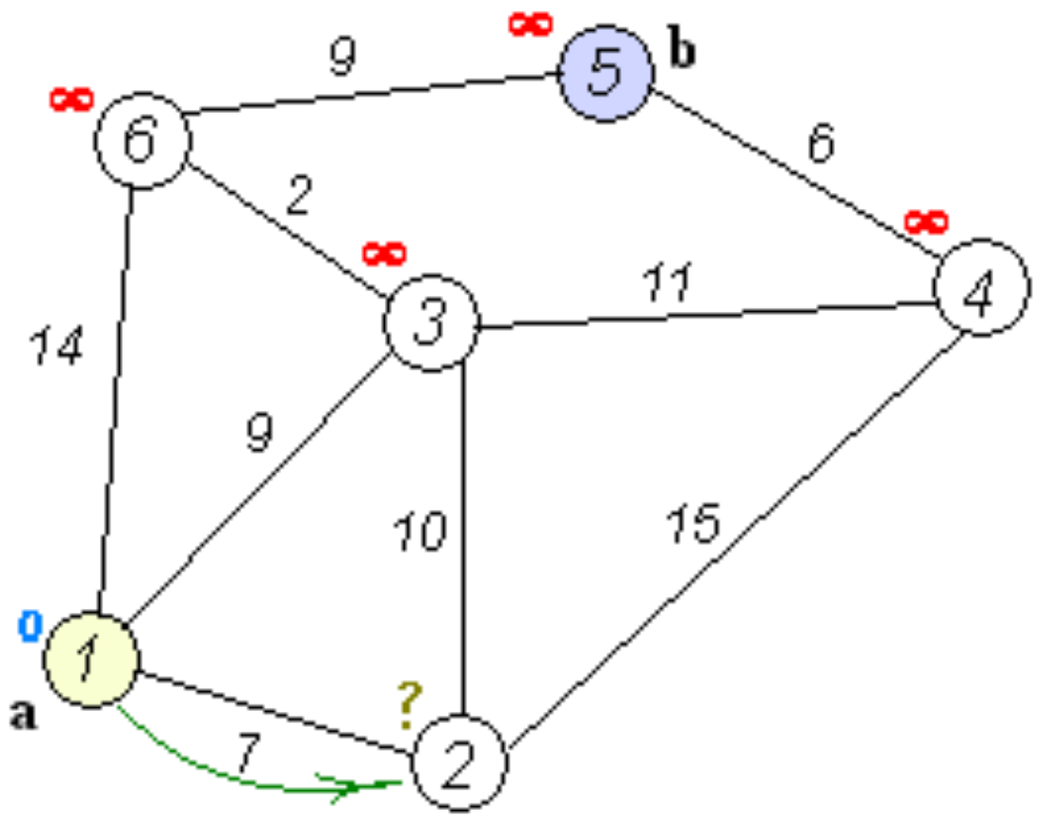
```
1   function Dijkstra(Graph, source):
2       dist[source] ← 0                                      // Initialization
3
4       create vertex set Q
5
6       for each vertex v in Graph:
7           if v ≠ source
8               dist[v] ← INFINITY                            // Unknown distance from so
9               prev[v] ← UNDEFINED                           // Predecessor of v
10
11          Q.add_with_priority(v, dist[v])
```



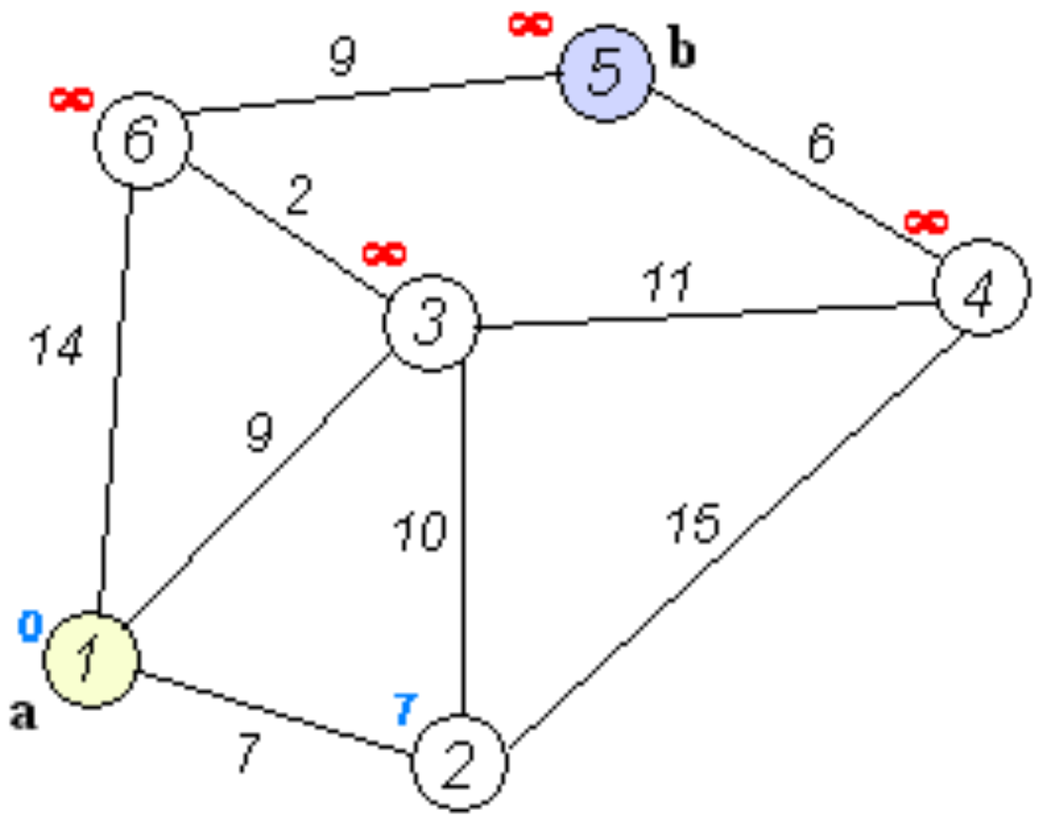
```
13
14      while Q is not empty:                                  // The main loop
15          u ← Q.extract_min()                                 // Remove and return best verte
16          for each neighbor v of u:                          // only v that is still in Q
17              alt ← dist[u] + length(u, v)
18              if alt < dist[v]
19                  dist[v] ← alt
20                  prev[v] ← u
21                  Q.decrease_priority(v, alt)
22
23      return dist[], prev[]
```



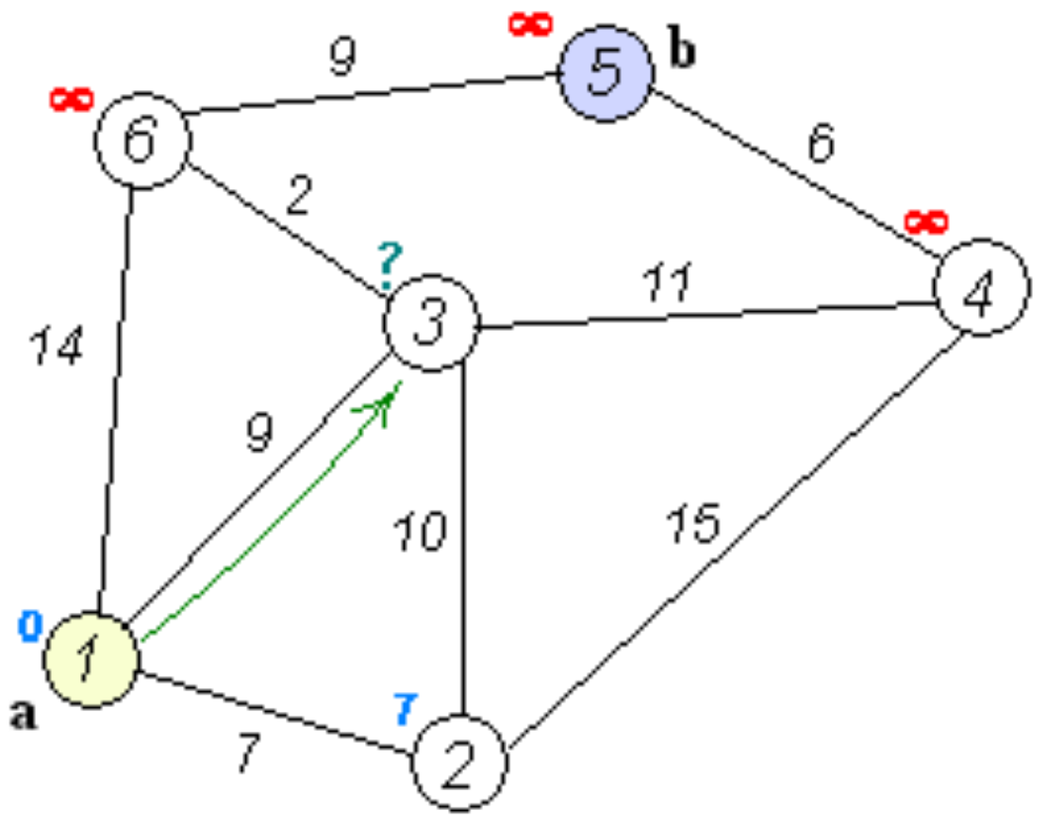
```
13
14      while Q is not empty:                                  // The main loop
15          u ← Q.extract_min()                                 // Remove and return best verte
16          for each neighbor v of u:                          // only v that is still in Q
17              alt ← dist[u] + length(u, v)
18              if alt < dist[v]
19                  dist[v] ← alt
20                  prev[v] ← u
21                  Q.decrease_priority(v, alt)
22
23      return dist[], prev[]
```



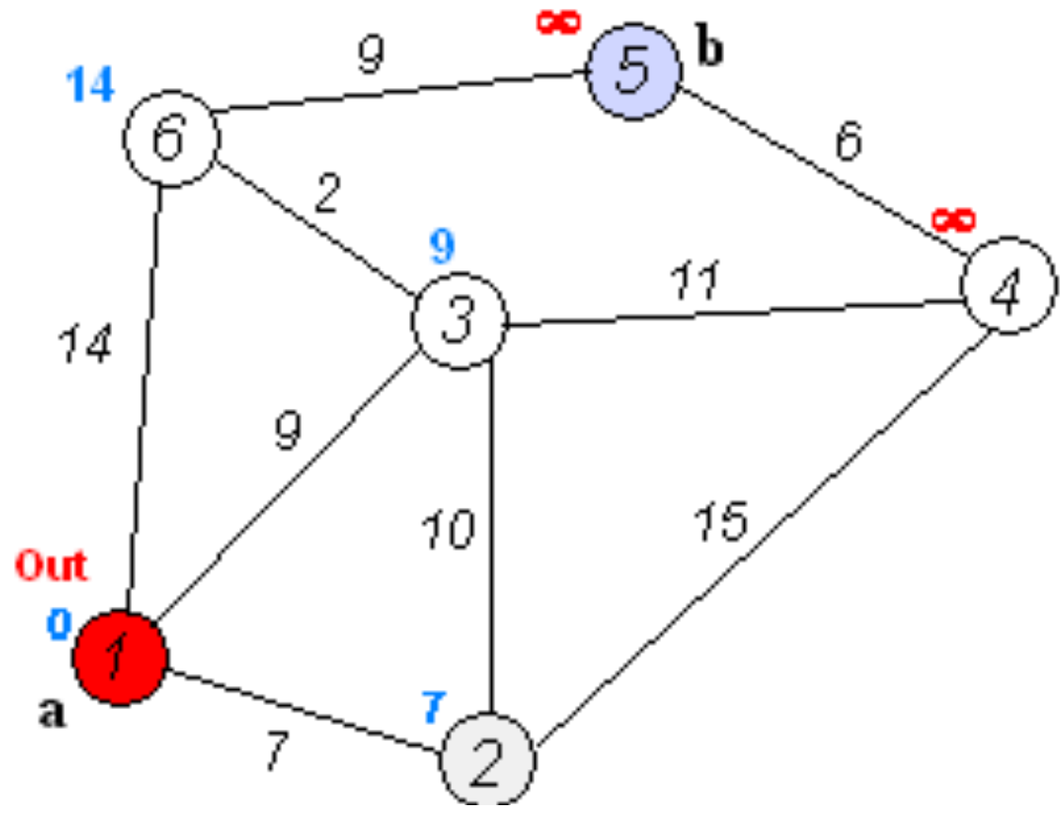
```
13
14      while Q is not empty:                                  // The main loop
15          u ← Q.extract_min()                                 // Remove and return best verte
16          for each neighbor v of u:                          // only v that is still in Q
17              alt ← dist[u] + length(u, v)
18              if alt < dist[v]
19                  dist[v] ← alt
20                  prev[v] ← u
21                  Q.decrease_priority(v, alt)
22
23      return dist[], prev[]
```



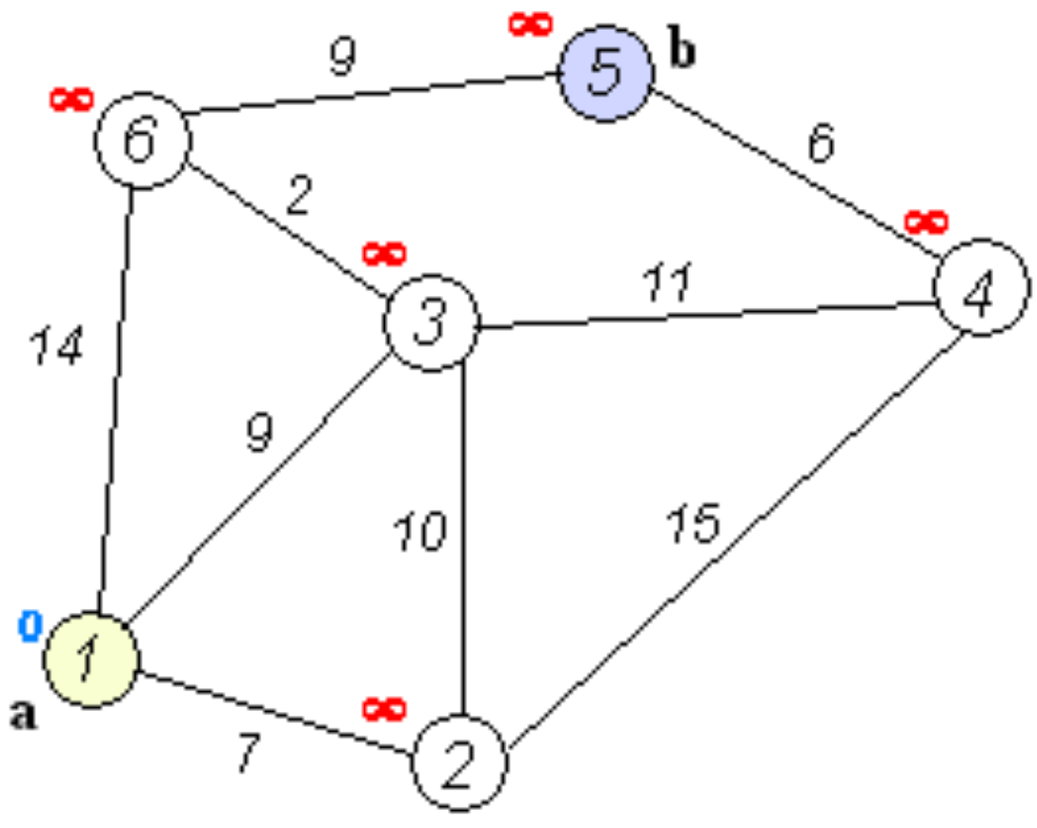
```
13
14      while Q is not empty:                                  // The main loop
15          u ← Q.extract_min()                                 // Remove and return best verte
16          for each neighbor v of u:                          // only v that is still in Q
17              alt ← dist[u] + length(u, v)
18              if alt < dist[v]
19                  dist[v] ← alt
20                  prev[v] ← u
21                  Q.decrease_priority(v, alt)
22
23      return dist[], prev[]
```



```
13
14      while Q is not empty:                                  // The main loop
15          u ← Q.extract_min()                                 // Remove and return best verte
16          for each neighbor v of u:                          // only v that is still in Q
17              alt ← dist[u] + length(u, v)
18              if alt < dist[v]
19                  dist[v] ← alt
20                  prev[v] ← u
21                  Q.decrease_priority(v, alt)
22
23      return dist[], prev[]
```

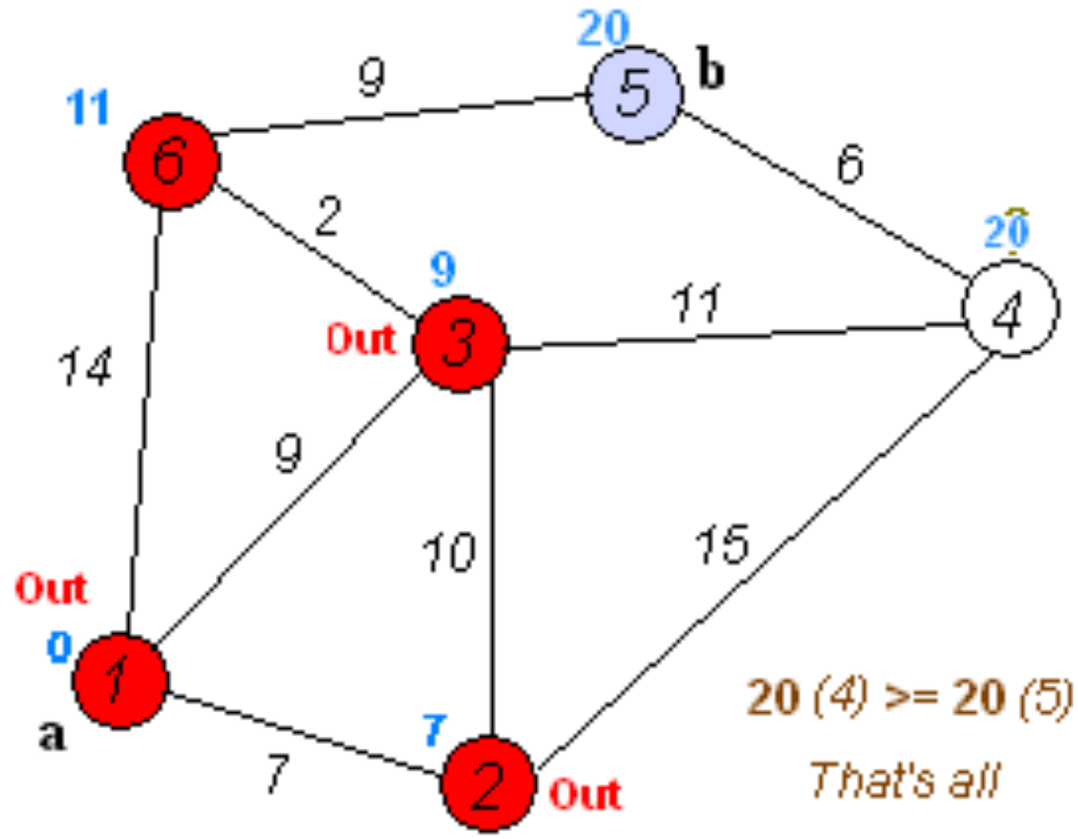


```
13
14      while Q is not empty:                                  // The main loop
15          u ← Q.extract_min()                                 // Remove and return best verte
16          for each neighbor v of u:                          // only v that is still in Q
17              alt ← dist[u] + length(u, v)
18              if alt < dist[v]
19                  dist[v] ← alt
20                  prev[v] ← u
21                  Q.decrease_priority(v, alt)
22
23      return dist[], prev[]
```



```
13
14      while Q is not empty:                                  // The main loop
15          u ← Q.extract_min()                                 // Remove and return best verte
16          for each neighbor v of u:                          // only v that is still in Q
17              alt ← dist[u] + length(u, v)
18              if alt < dist[v]
19                  dist[v] ← alt
20                  prev[v] ← u
21                  Q.decrease_priority(v, alt)
22
23      return dist[], prev[]
```

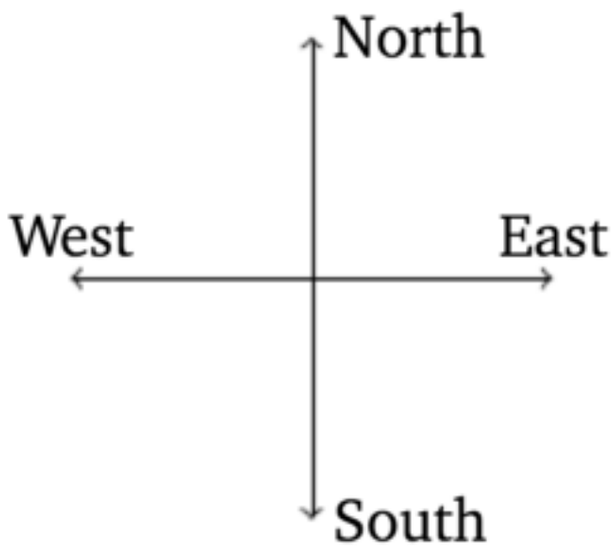
The algorithm ends when the destination is extracted from the queue.



But we don't have a graph...

But we don't have a graph...

But we have a grid



88	89	90	91	92	93	94	95	96	97	98	99	100	101	102
109	110	111	112	113	114	115	116	117	118	119	120	121	122	123
130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
151	152	153	154	155	156	157	158	159	160	161	162	163	164	165
172	173	174	175	176	177	178	179	180	181	182	183	184	185	186
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
214	215	216	217	218	219	220	221	222	223	224	225	226	227	228
235	236	237	238	239	240	241	242	243	244	245	246	247	248	249
256	257	258	259	260	261	262	263	264	265	266	267	268	269	270
277	278	279	280	281	282	283	284	285	286	287	288	289	290	291
298	299	300	301	302	303	304	305	306	307	308	309	310	311	312
319	320	321	322	323	324	325	326	327	328	329	330	331	332	333
340	341	342	343	344	345	346	347	348	349	350	351	352	353	354

And what about the priority Queue?

See code example