CSC 226

Algorithms and Data Structures: II
Shortest Paths - Dijkstra

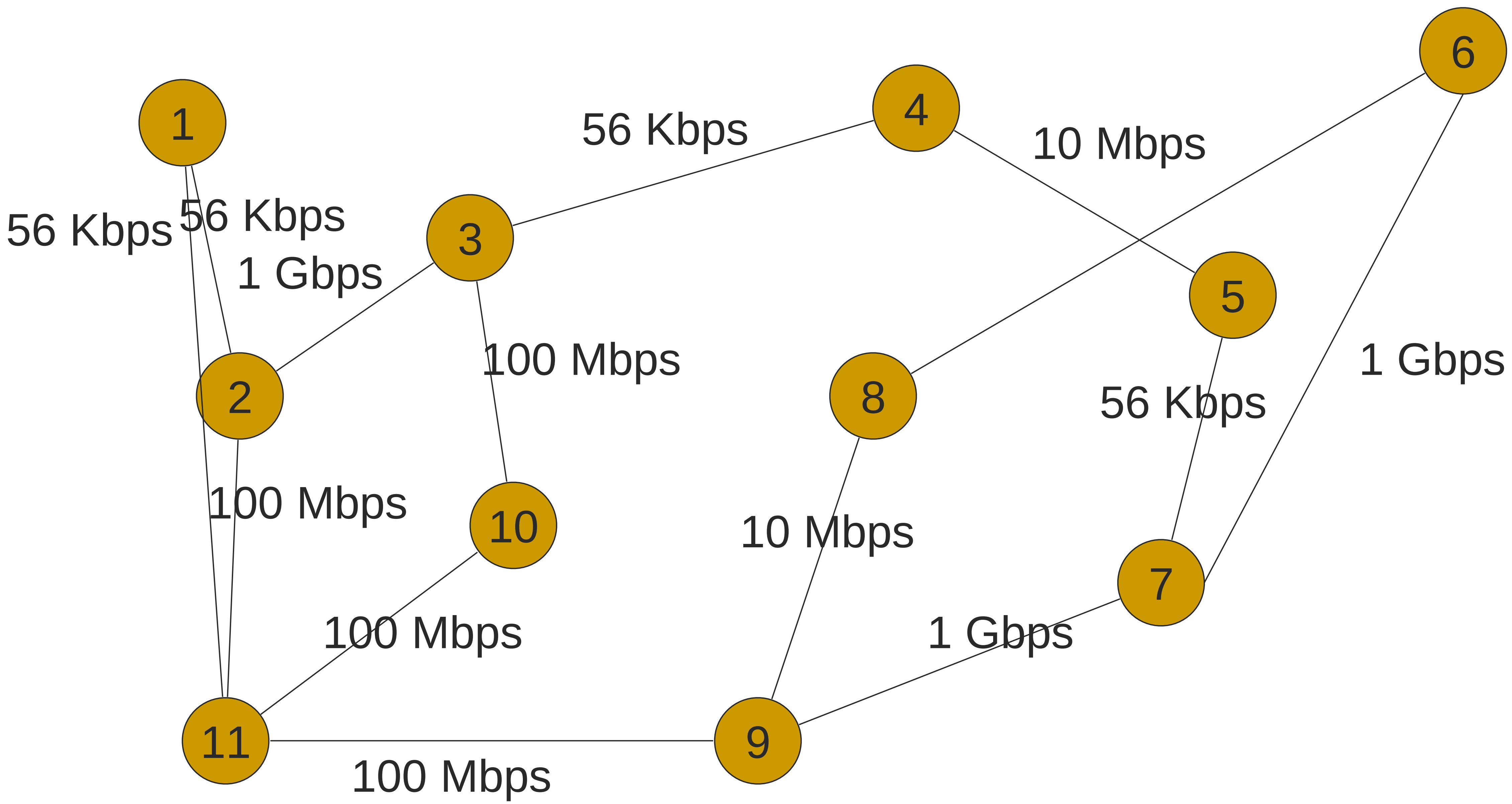
Tianming Wei
twei@uvic.ca
ECS 466

Shortest Paths

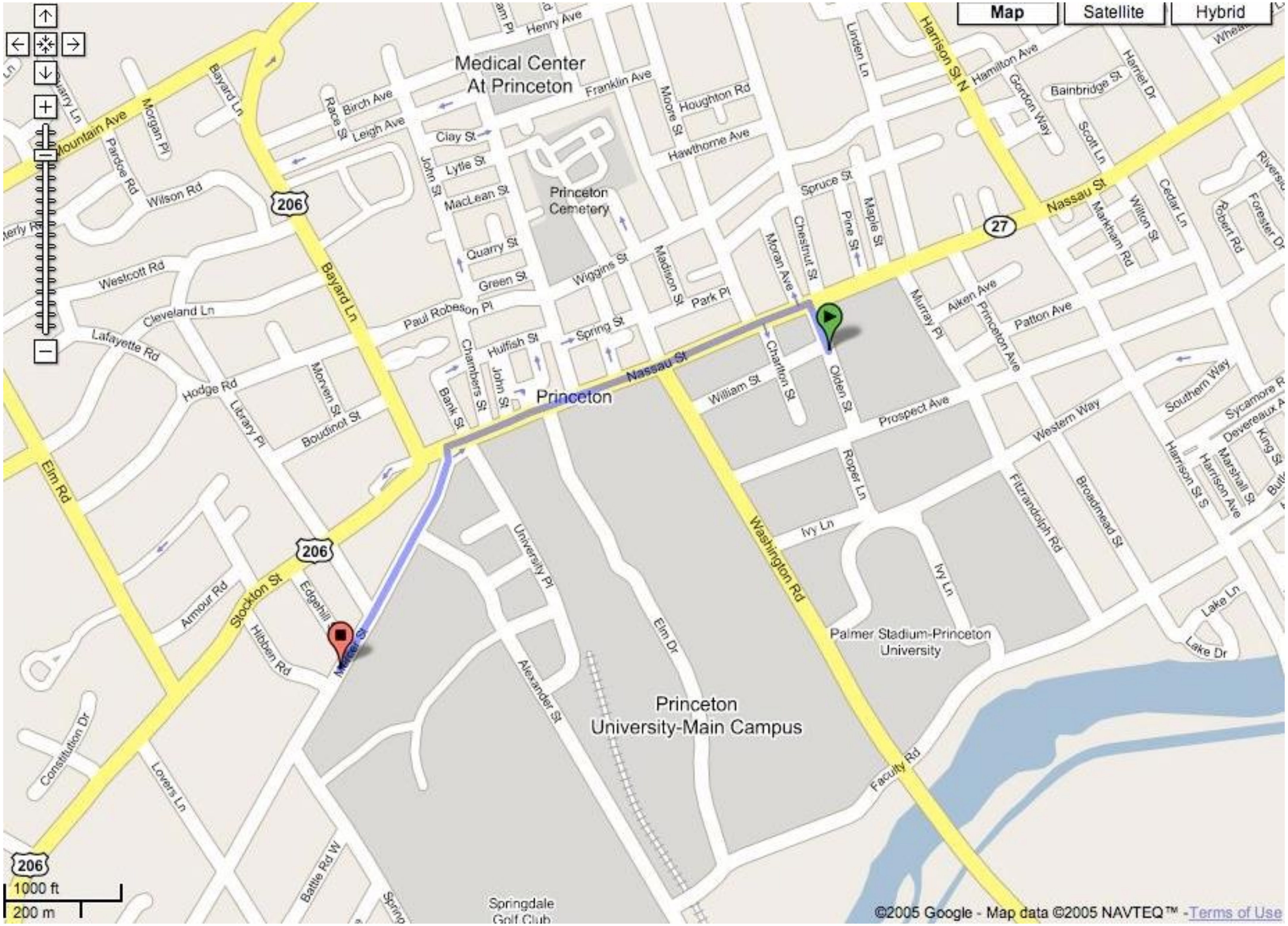
in edge-weighted graphs

Communication Speeds in a Computer Network

Find fastest way to route a data packet between two computers



Google maps



Map
Satellite
Hybrid
Medical Center
At Princeton
Henry Ave
Franklin Ave
Birch Ave
Leigh Ave
Race St
Clay St
Lytle St
MacLean St
John St
Houghton Rd
Moore St
Hawthorne Ave
Princeton
Cemetery
Mountain Ave
Morgan Pl
Bayard Ln
Pardoe Rd
Wilson Rd
206
Westcott Rd
Cleveland Ln
Lafayette Rd
Hodge Rd
Library Pl
Mercer St
Boudinot St
Paul Robeson Pl
Quarry St
Green St
Wiggins St
Madison St
Park Pl
Spring St
Hulfish St
Chambers St
Bank St
Princeton
Nassau St
William St
Charlton St
Olden St
Prospect Ave
Spruce St
Chestnut St
Moran Ave
Pine St
Maple St
Linden Ln
Harrison St N
Hamilton Ave
Gordon Way
Bainbridge St
Harriet Dr
Scott Ln
Nassau St
27
Markham Rd
Wilton St
Cedar Ln
Robert Rd
Forester Dr
Murray Pl
Aiken Ave
Princeton Ave
Patton Ave
Western Way
Southern Way
Sycamore Rd
Devereaux Ave
King St
Harrison St S
Harrison Ave
Marshall St
Roper Ln
Ivy Ln
Fitzrandolph Rd
Broadmead St
Washington Rd
Palmer Stadium-Princeton
University
Lake Ln
Lake Dr
Faculty Rd
University Pl
Elm Dr
Alexander St
Princeton
University-Main Campus
Elm Rd
Armour Rd
Stockton St
Edgehill St
Hibben Rd
Constitution Dr
Lovers Ln
Battle Rd W
Springdale
Golf Club
1000 ft
200 m
©2005 Google - Map data ©2005 NAVTEQ™ - Terms of Use

Shortest path applications

- PERT/CPM.
- Map routing.
- Seam carving.
- Texture mapping.
- Robot navigation.
- Typesetting in TeX.
- Urban traffic planning.
- Optimal pipelining of VLSI chip.
- Telemarketer operator scheduling.
- Routing of telecommunications messages.
- Network routing protocols (OSPF, BGP, RIP).
- Exploiting arbitrage opportunities in currency exchange.
- Optimal truck routing through given traffic congestion pattern.

http://en.wikipedia.org/wiki/Seam_carving

Reference: Network Flows: Theory, Algorithms, and Applications, R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, Prentice Hall, 1993.

Shortest Path problems

- Find a shortest path between two given vertices
- Single source shortest paths
- Single sink shortest paths
- All pair shortest paths

Single Source Shortest Path problems

- Undirected graphs with non-negative edge weights
- Directed graphs with non-negative edge weights
- Directed graphs with arbitrary weights

Single Source Shortest Paths

- If graph is not weighted (all edge-weights are unit-weight): BFS works
- Now assume: graph is edge-weighted
  - Every edge is associated with a positive number
    - Possible weights: integers, real numbers, rational numbers
    - Edge-weights can represent: distance, connection cost, affinity

Single Source Shortest Paths

- Input: An edge-weighted undirected graph and a source node v with: for every edge e edge-weight w(e) > 0
- Output: All single-source shortest paths (and their weight) for v in G: for every node $w \neq v$ in G a shortest path from v to w.
  - Here, a *path* p from v to w consisting of edges $e_0, e_1, \dots, e_{k-1}$ is shortest in G, if its length

$$w(p) = \sum_{i=0}^{k-1} w(e_i)$$

is minimum (i.e., there is no path from v to w in G that is shorter).

`Algorithm`
`DijkstraShortestPaths(G,v)`

Input: A simple undirected graph G with nonnegative edge-weights, a distinguished vertex v in G

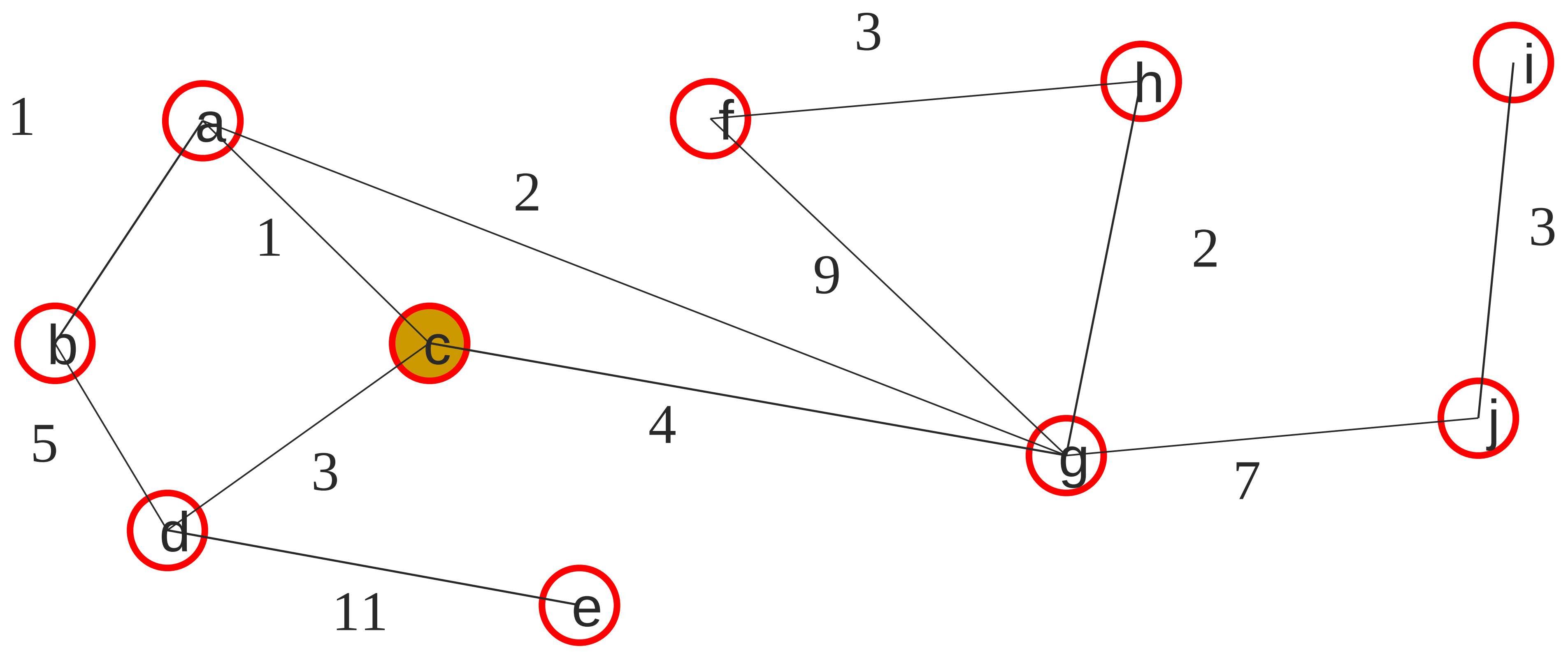
Output: A label D[u] for each vertex u in G such that D[u] is the shortest distance from v to u in G.

Algorithm DijkstraShortestPaths(G,v)

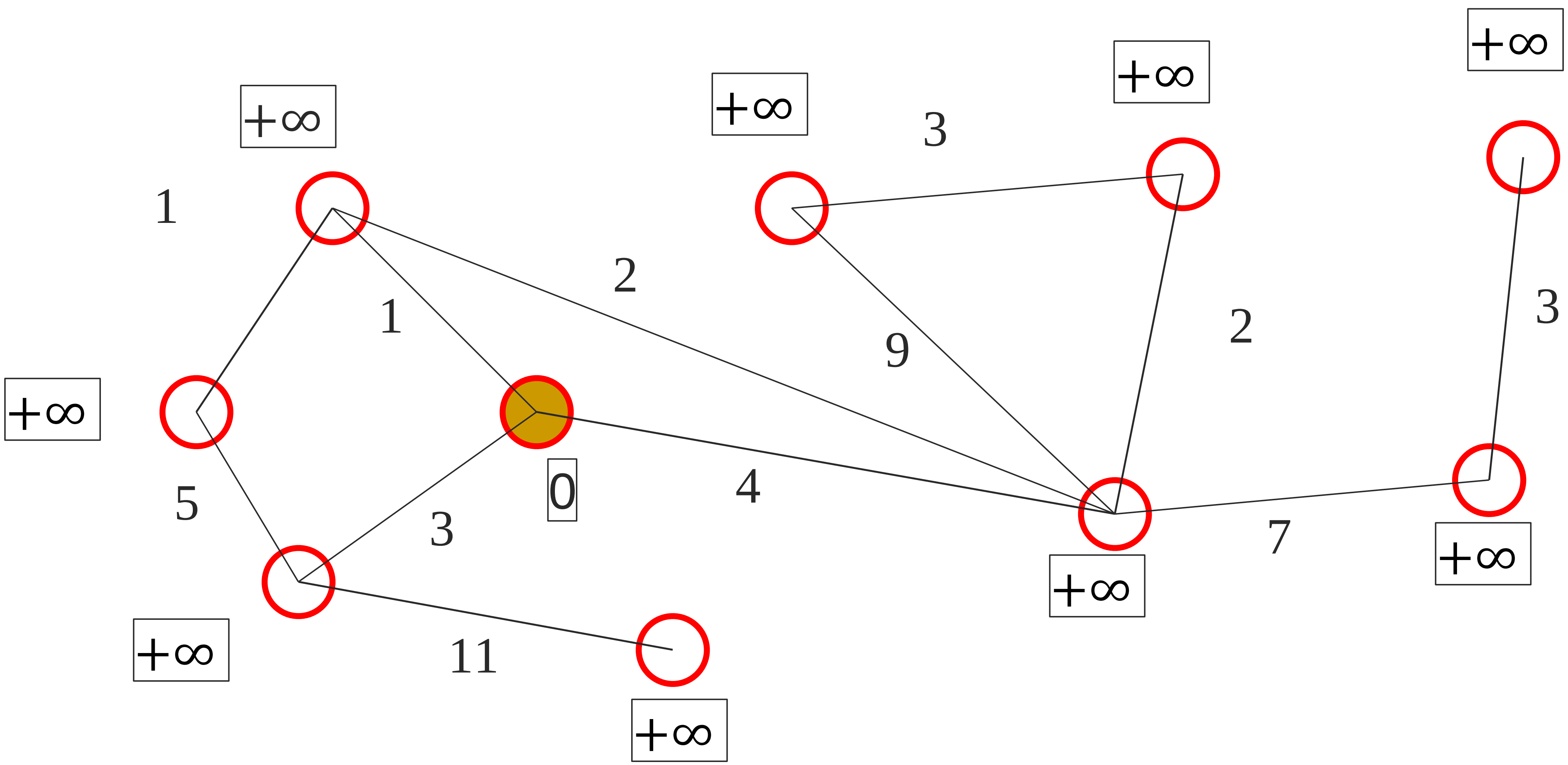
```
D[v]←0
for each vertex u≠v of G do
   D[u]← +∞
Let Q be a priority queue containing all
   vertices of G using D[.] as keys
while Q is not empty do
   u←Q.removeMin() //u is added to cloud
   for each vertex z∈N(u) with z∈Q do
     if D[u]+w((u,z)) < D[z] then
         D[z]←D[u]+w((u,z))
         update z's key in Q to D[z]
return D
```

Relaxation

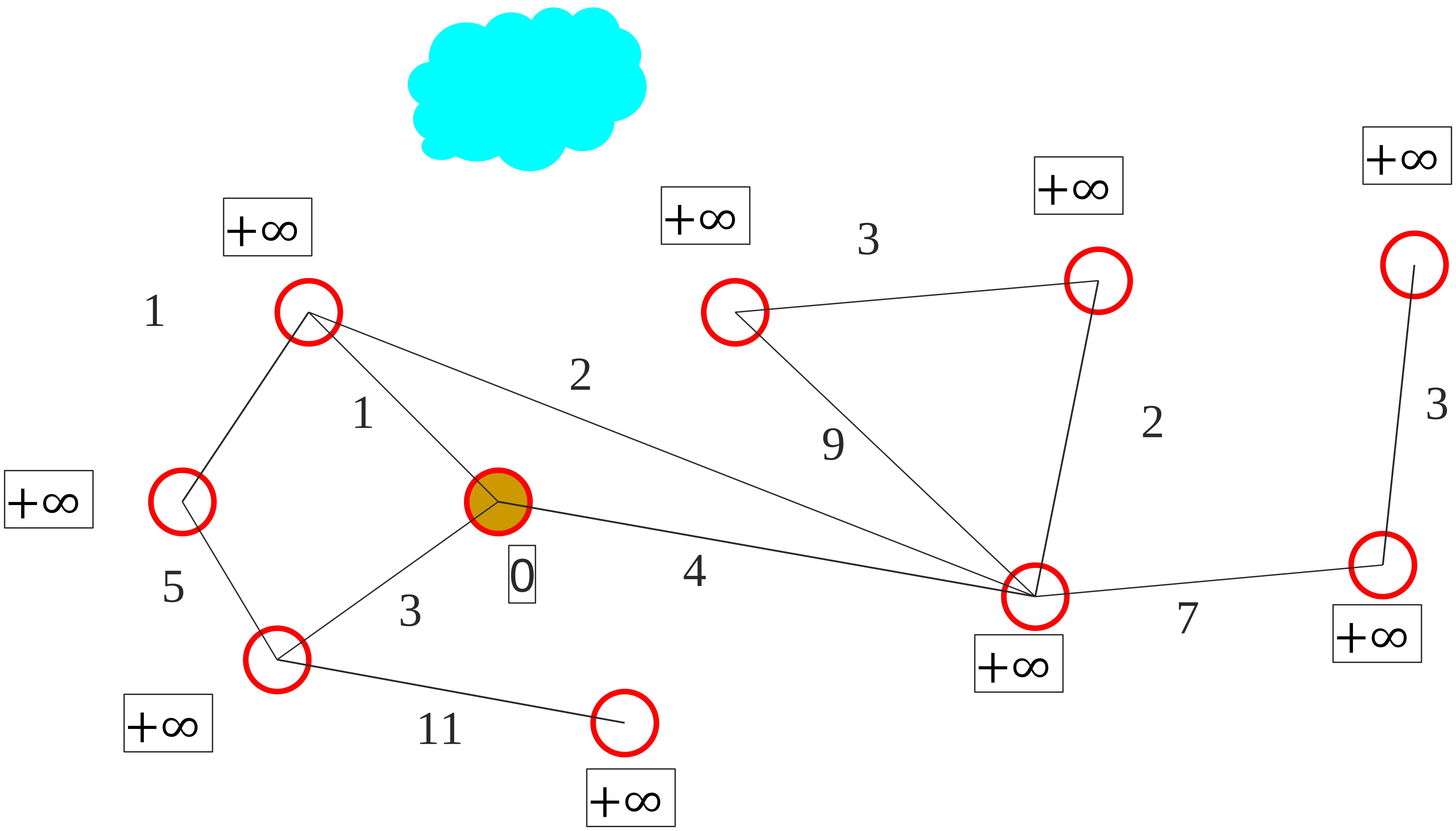
Dijkstra's algorithm: a greedy algorithm



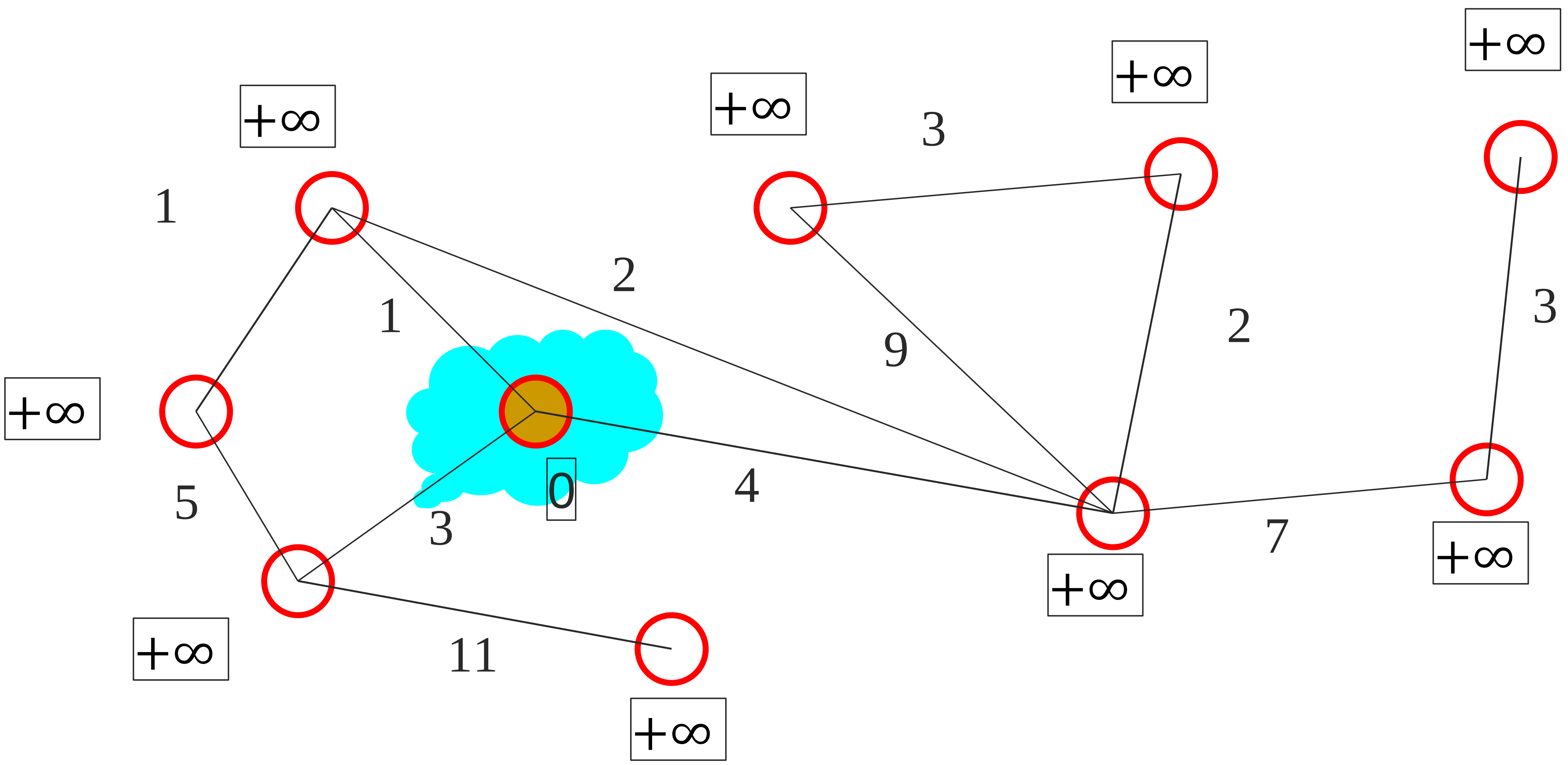
Dijkstra's algorithm: Initializing



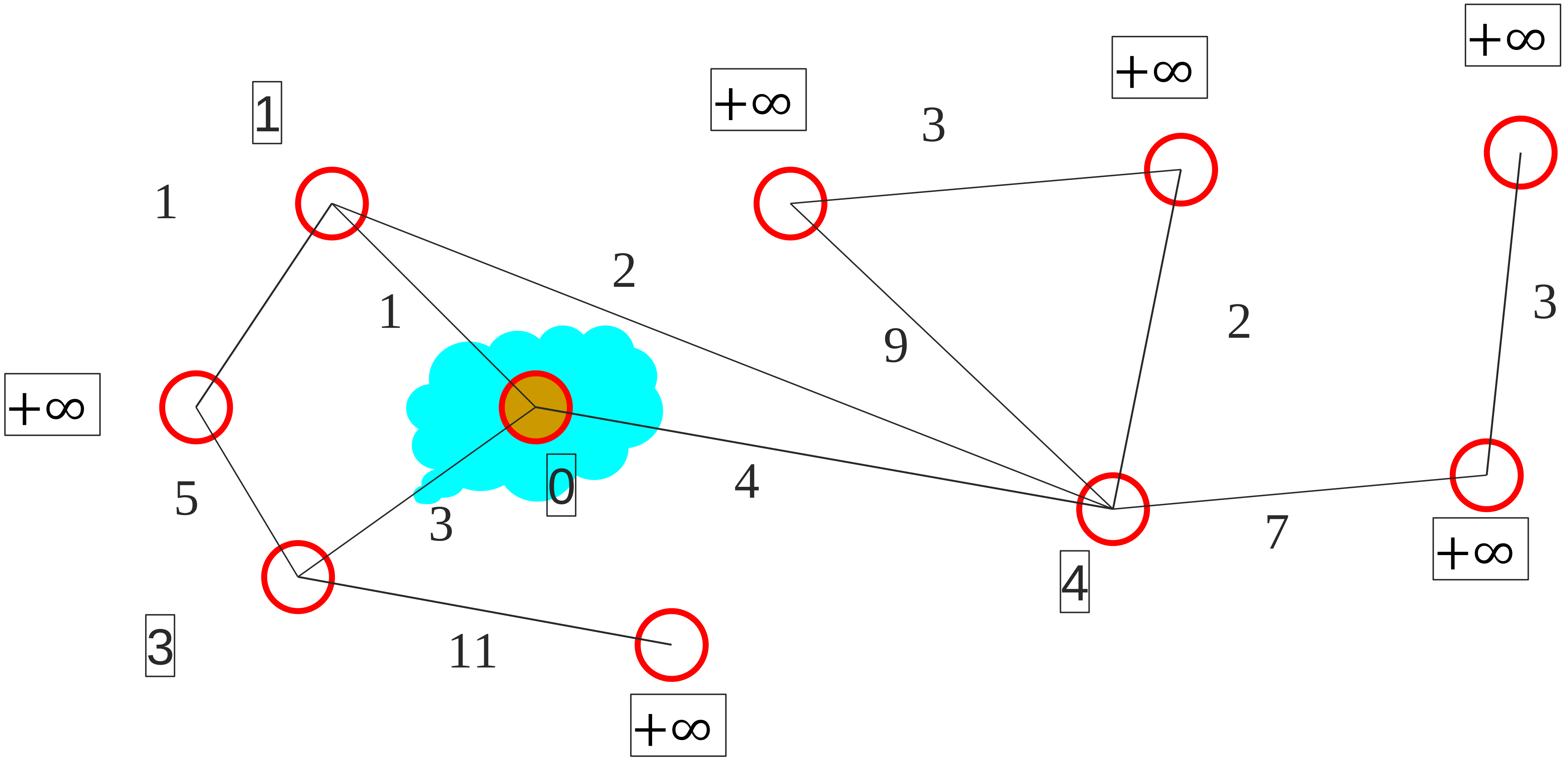
Dijkstra's algorithm: Initializing Cloud C (consisting of "solved" subgraph)



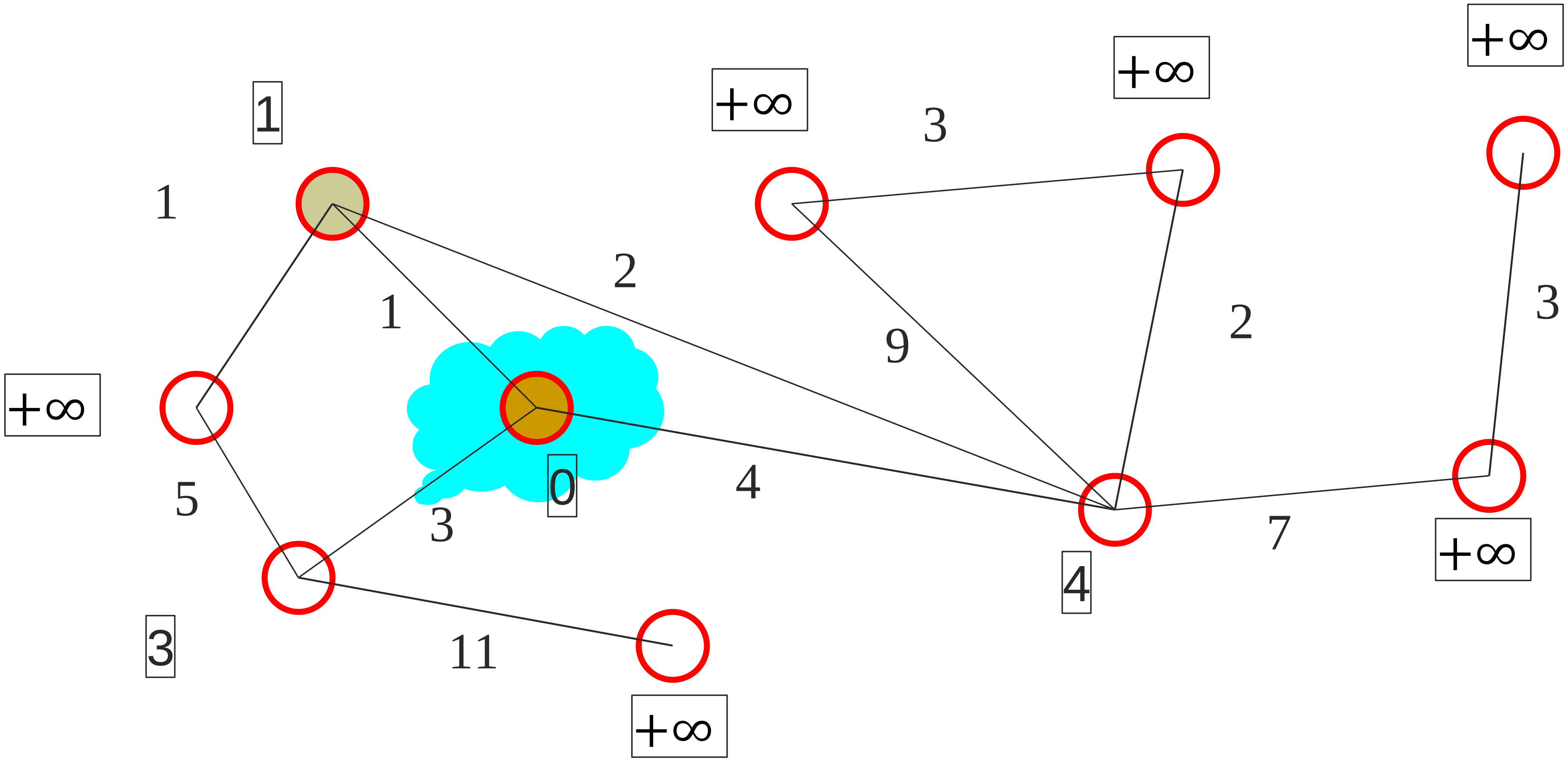
Dijkstra's algorithm: pull v into C



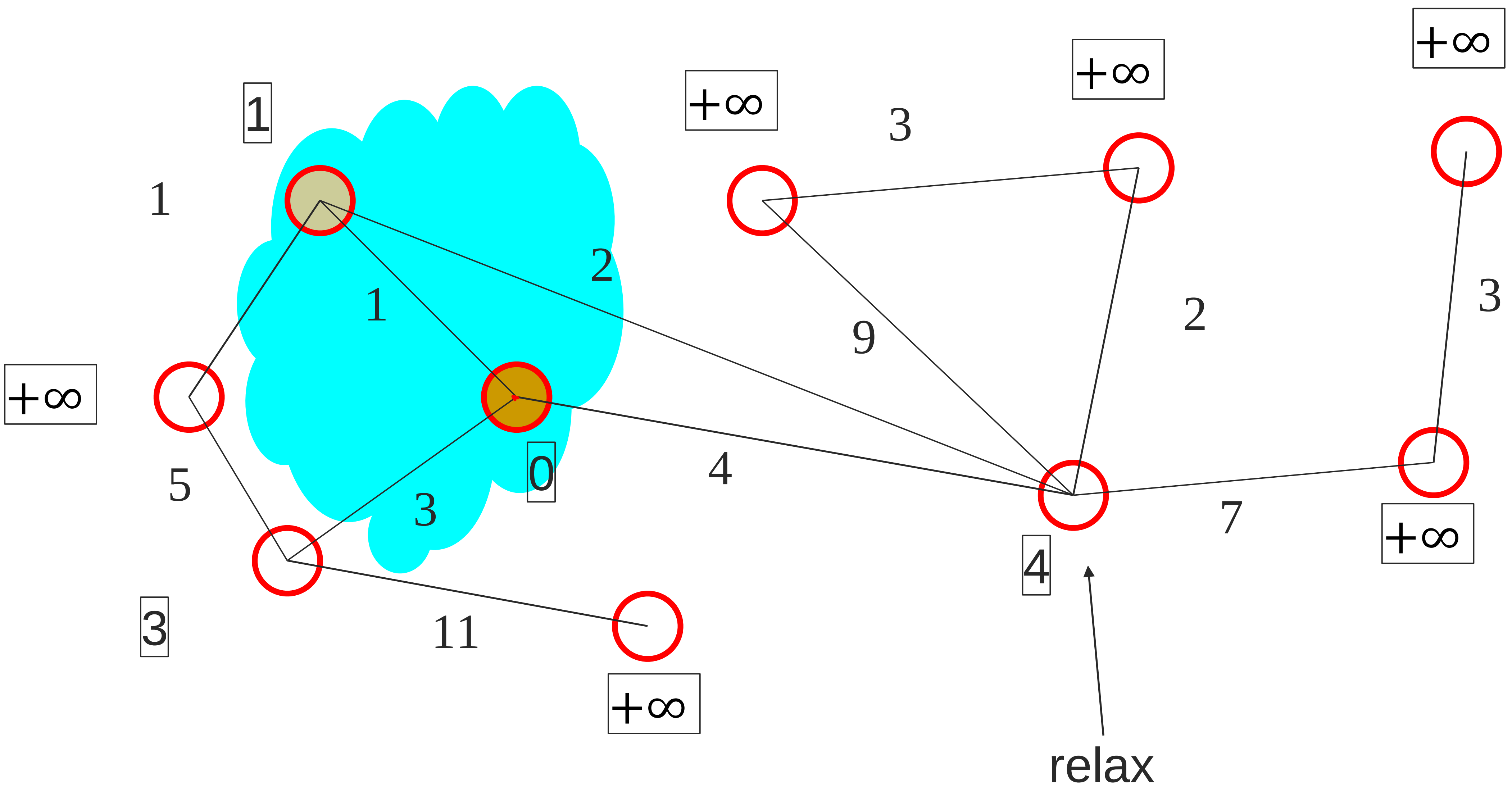
Dijkstra's algorithm: update *C's* neighborhood



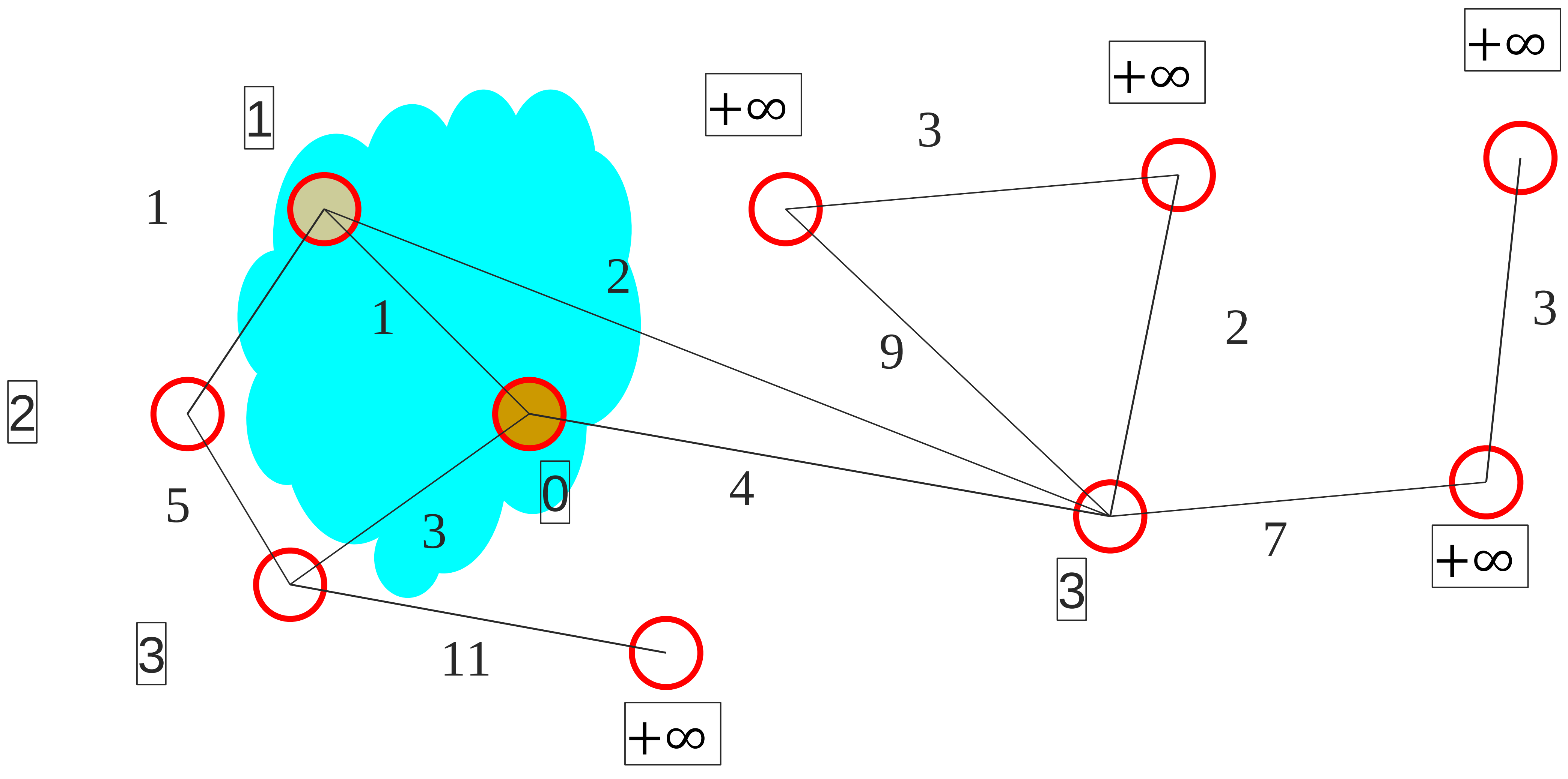
Dijkstra's algorithm: pick closest vertex u outside C



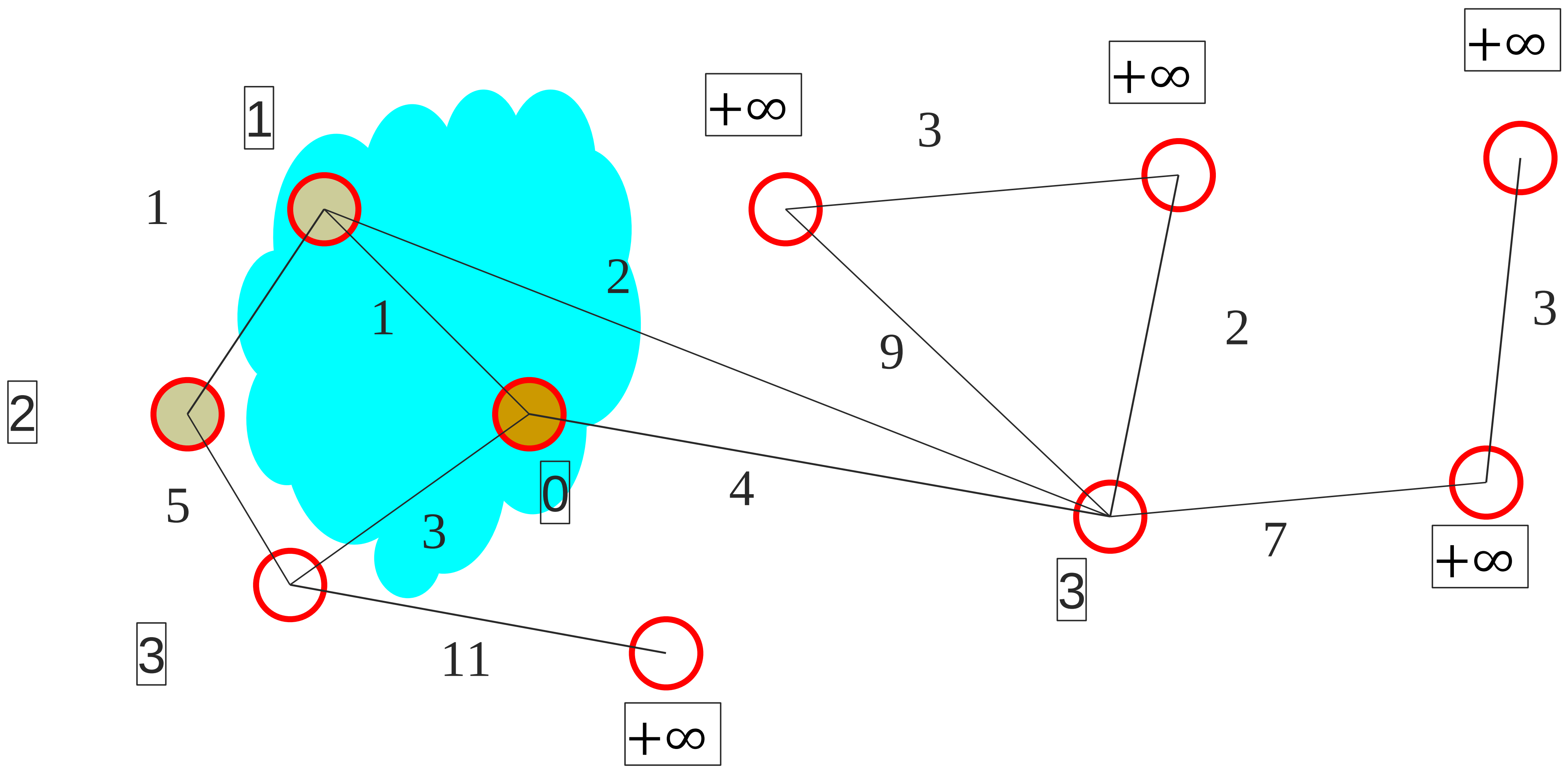
Dijkstra's algorithm: pull u into C



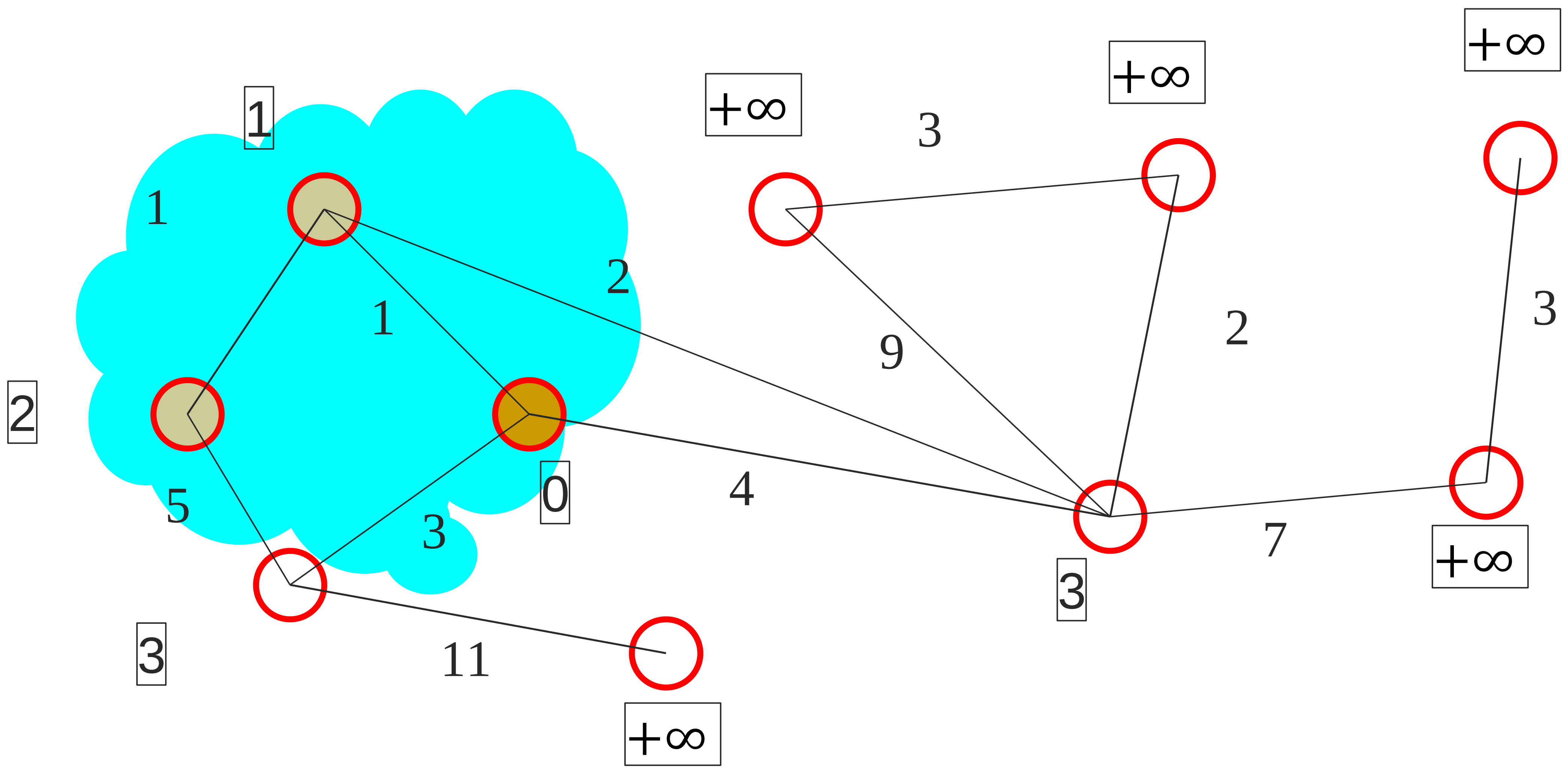
Dijkstra's algorithm: update *C*'s neighborhood



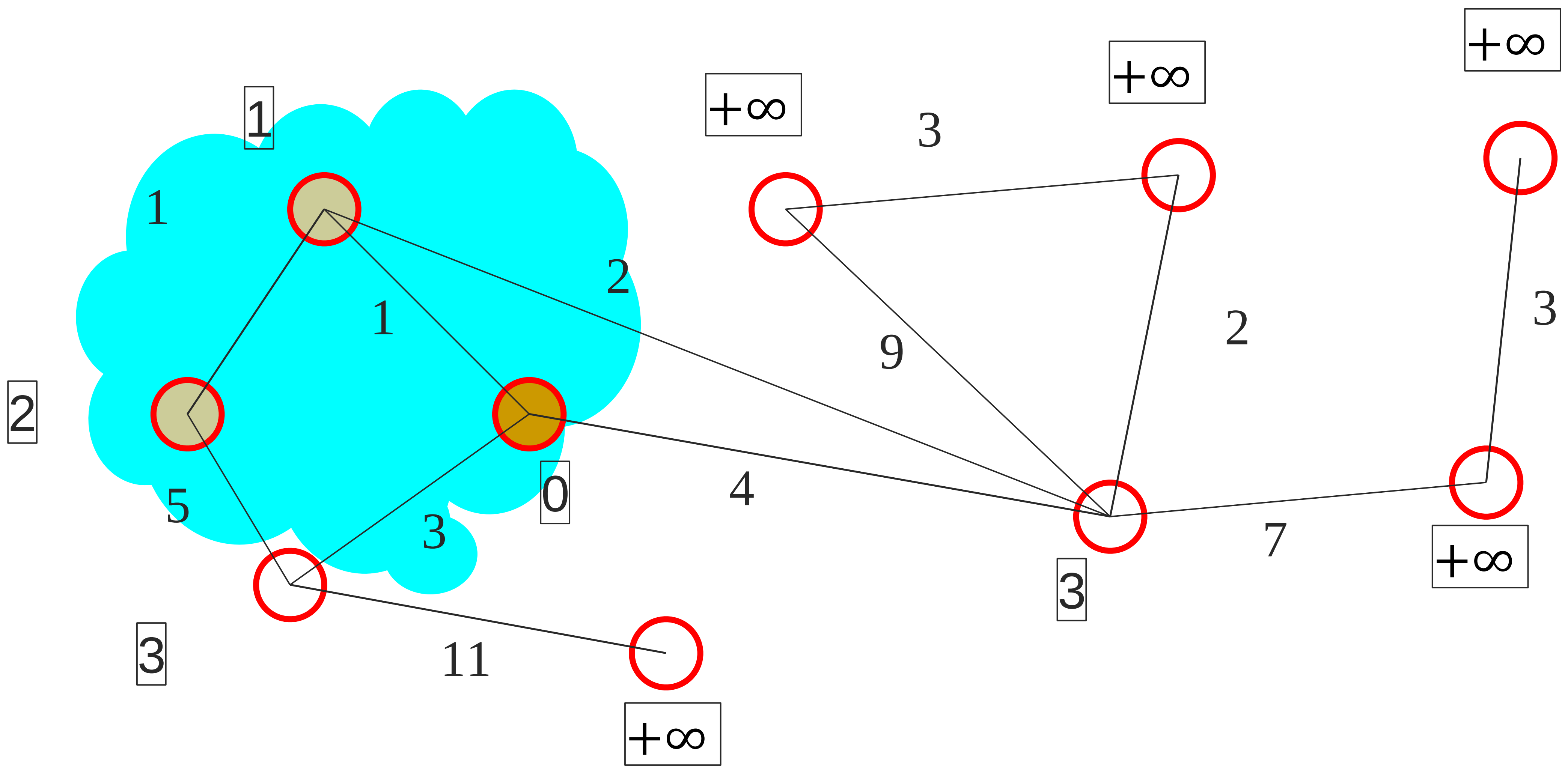
Dijkstra's algorithm: pick closest vertex u outside C



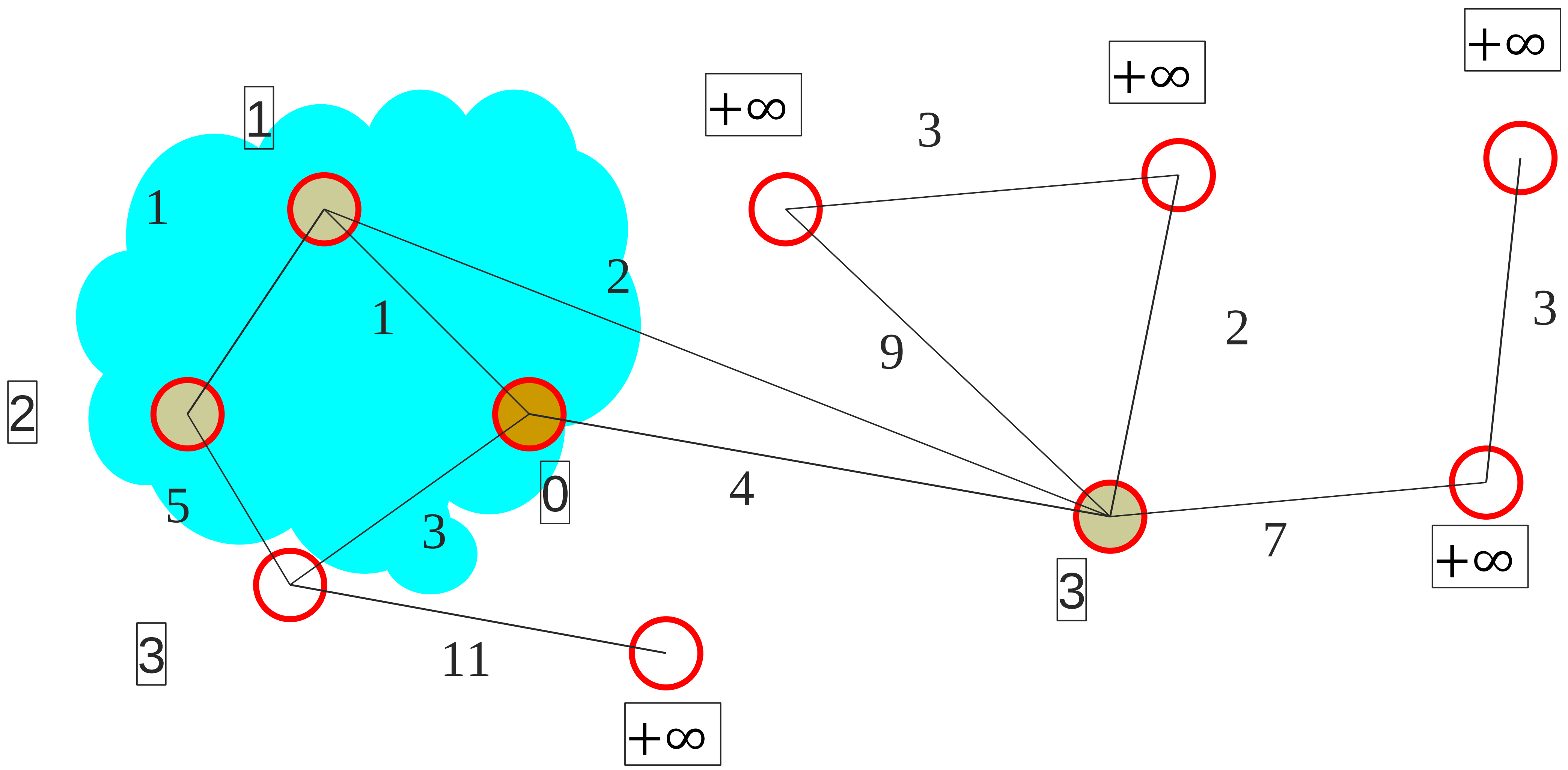
Dijkstra's algorithm: pull u into C



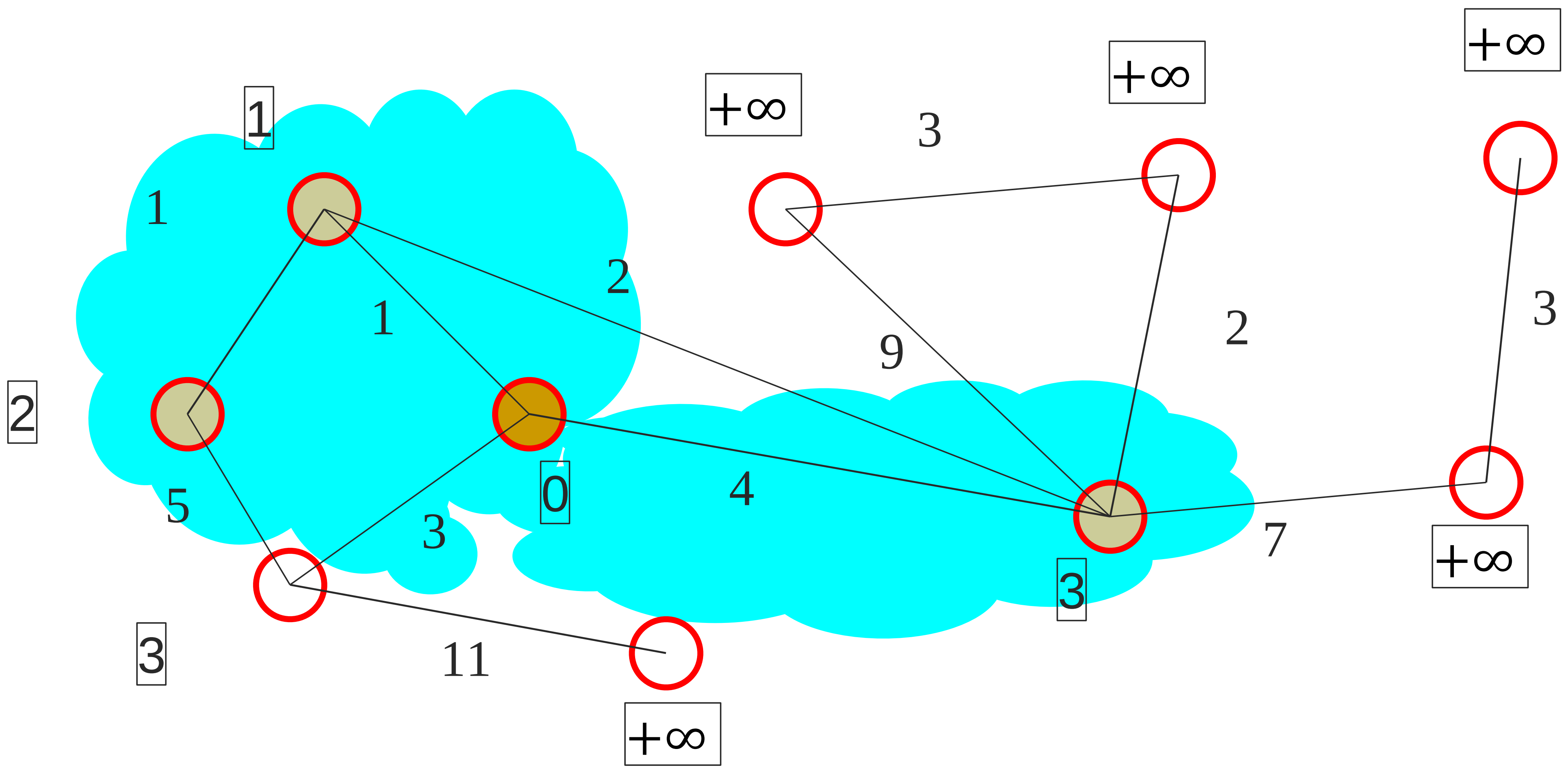
Dijkstra's algorithm: update *C's* neighborhood



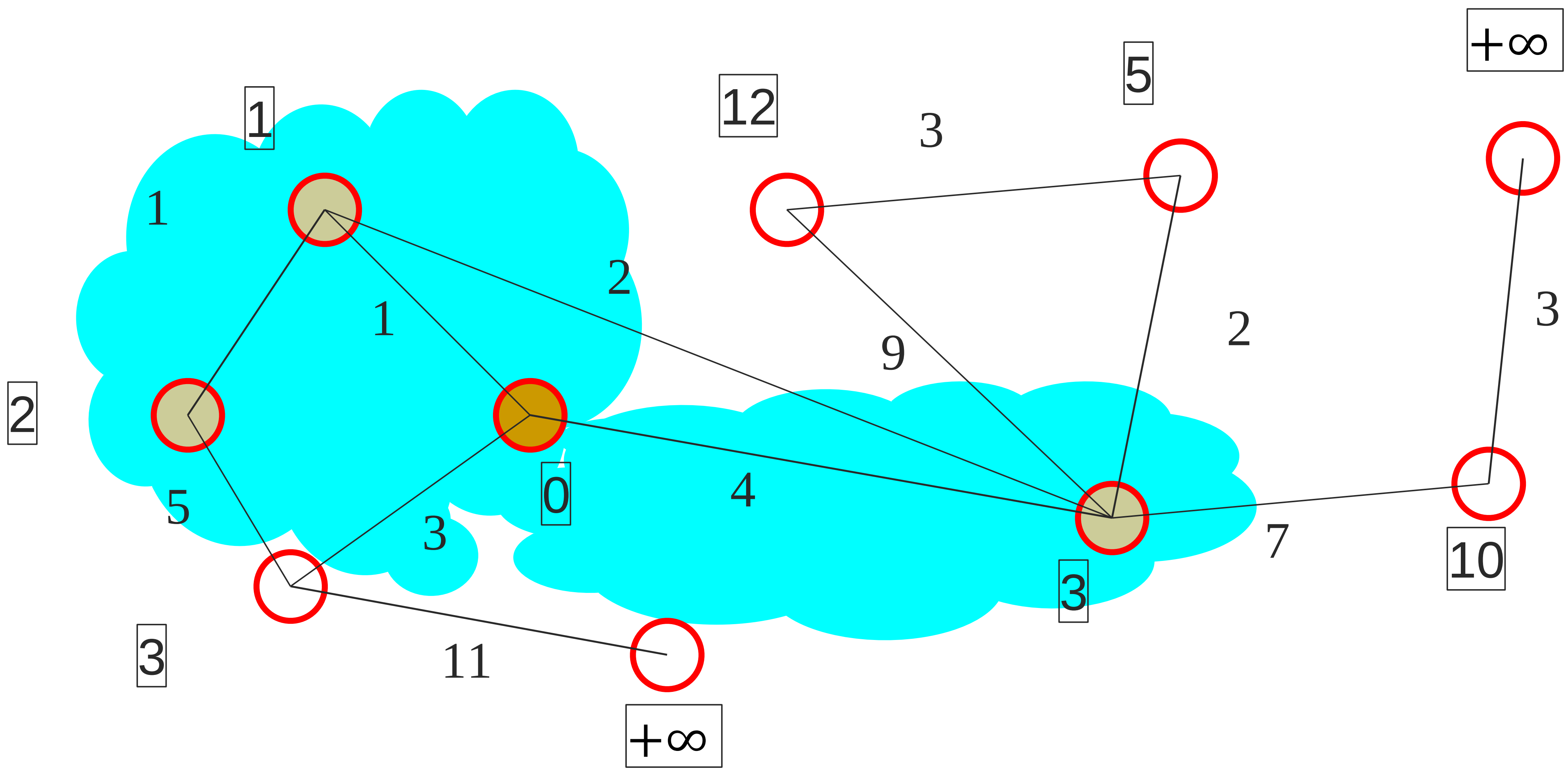
Dijkstra's algorithm: pick closest vertex u outside C



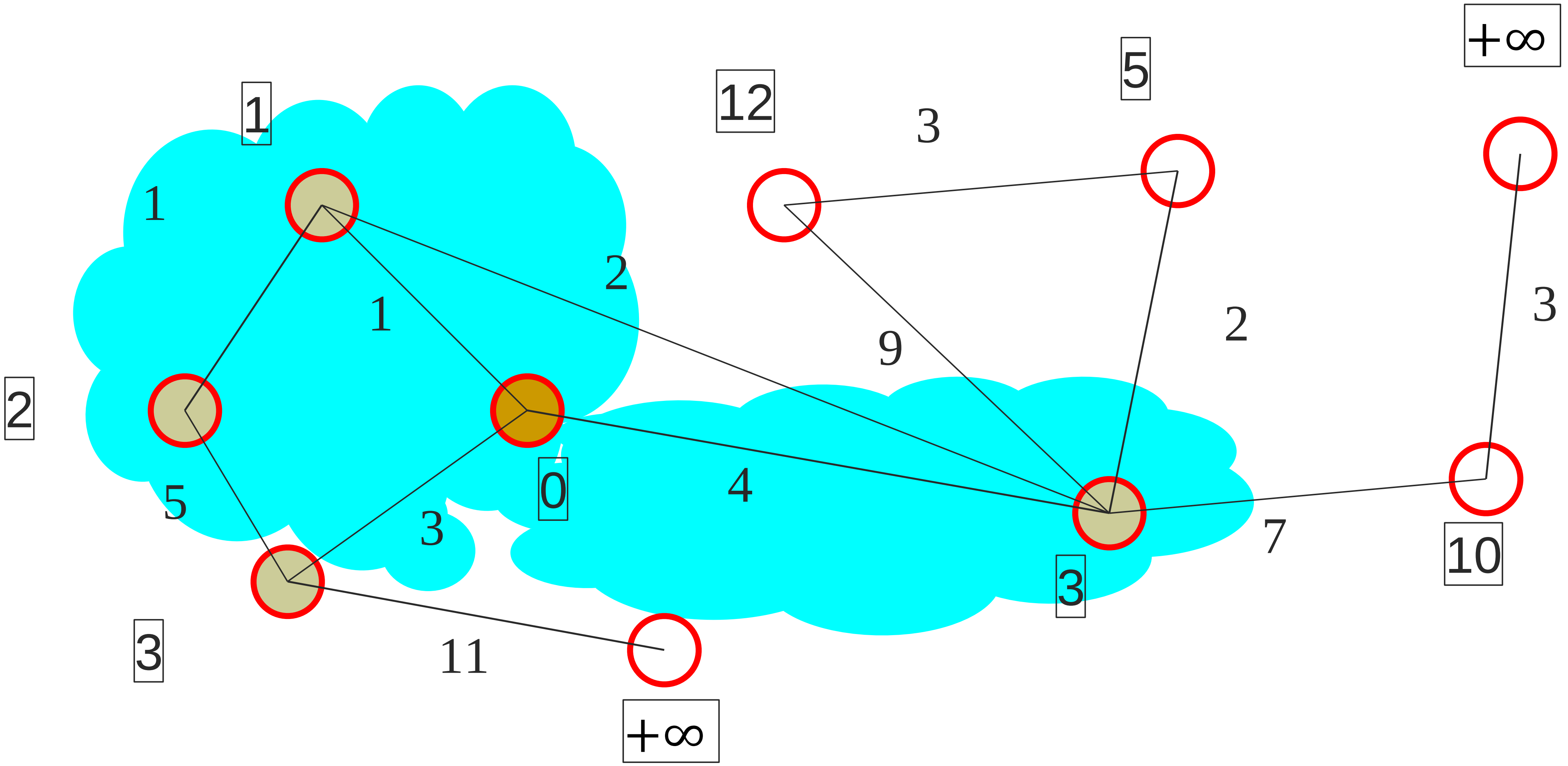
Dijkstra's algorithm: pull u into C



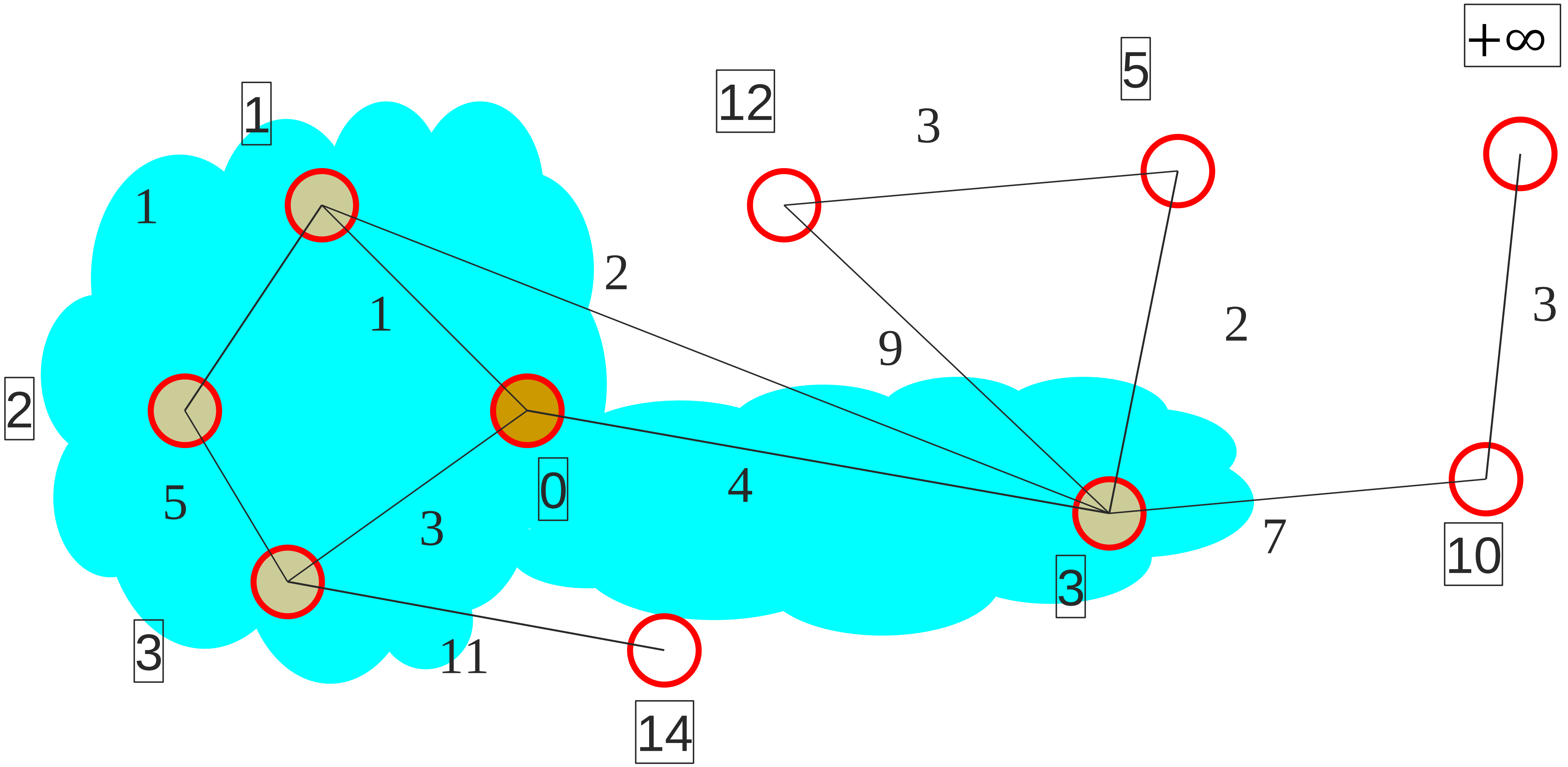
Dijkstra's algorithm: update *C*'s neighborhood



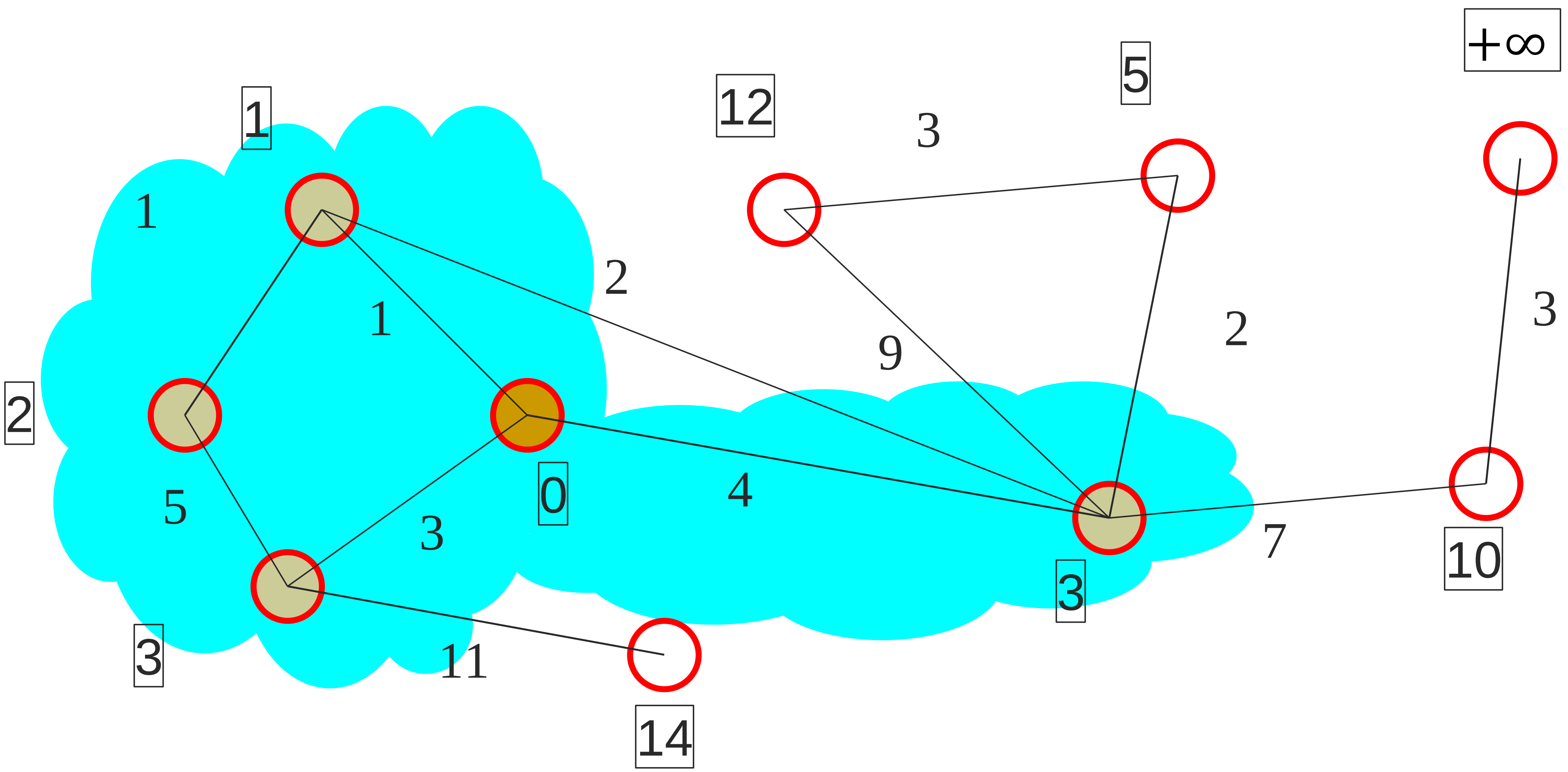
Dijkstra's algorithm: pick closest vertex u outside C



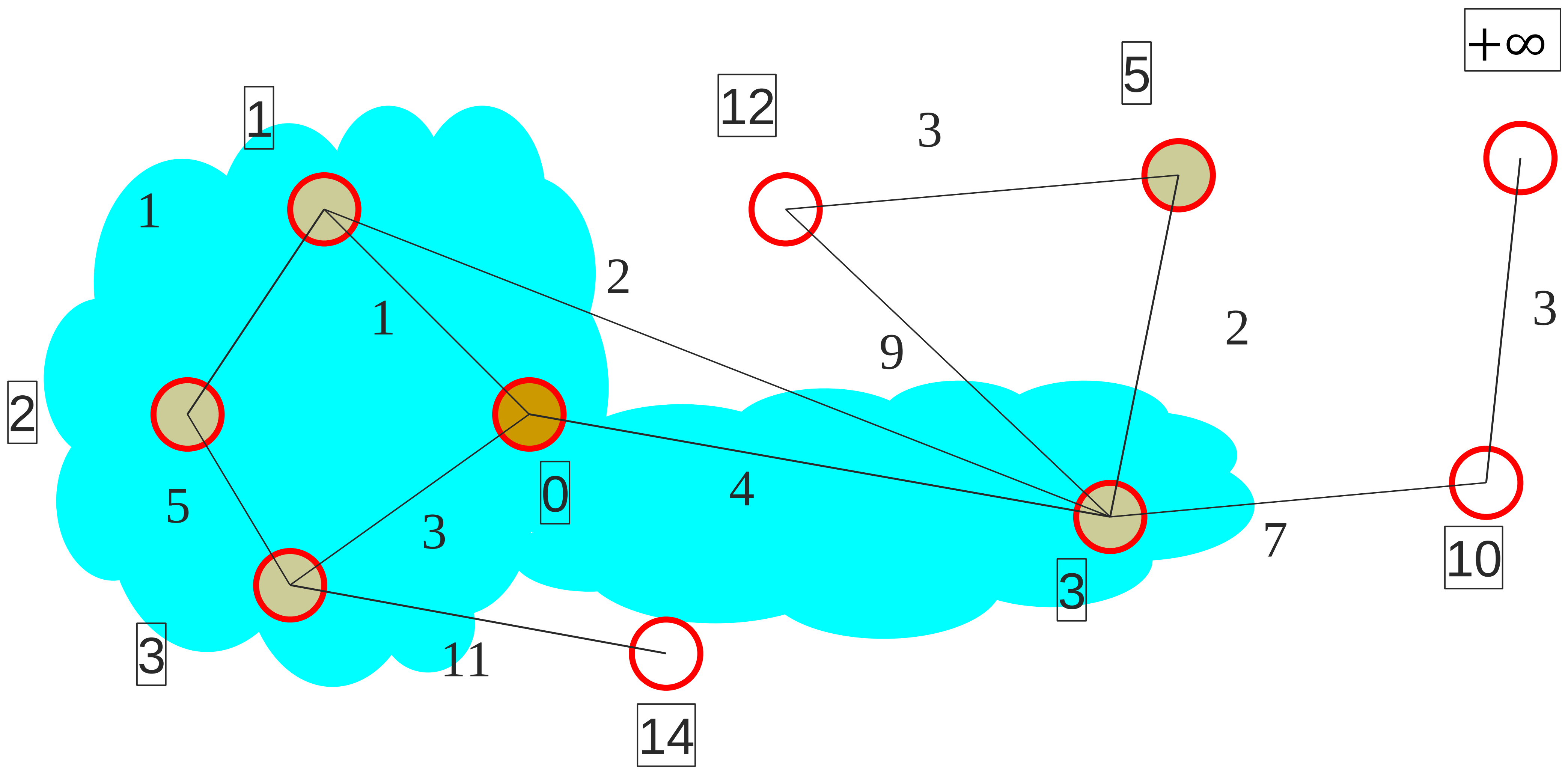
Dijkstra's algorithm: pull u into C



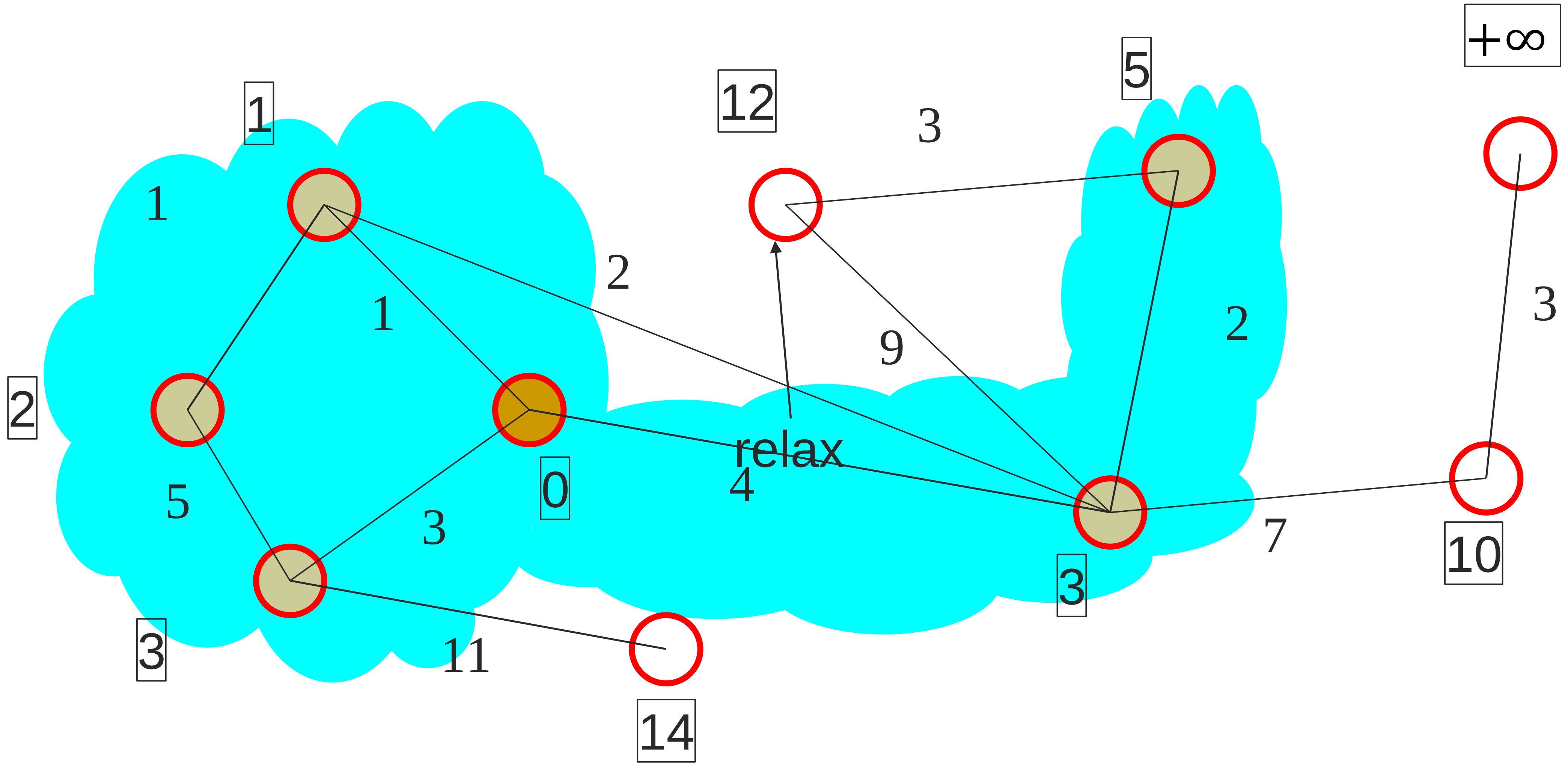
Dijkstra's algorithm: update *C's* neighborhood



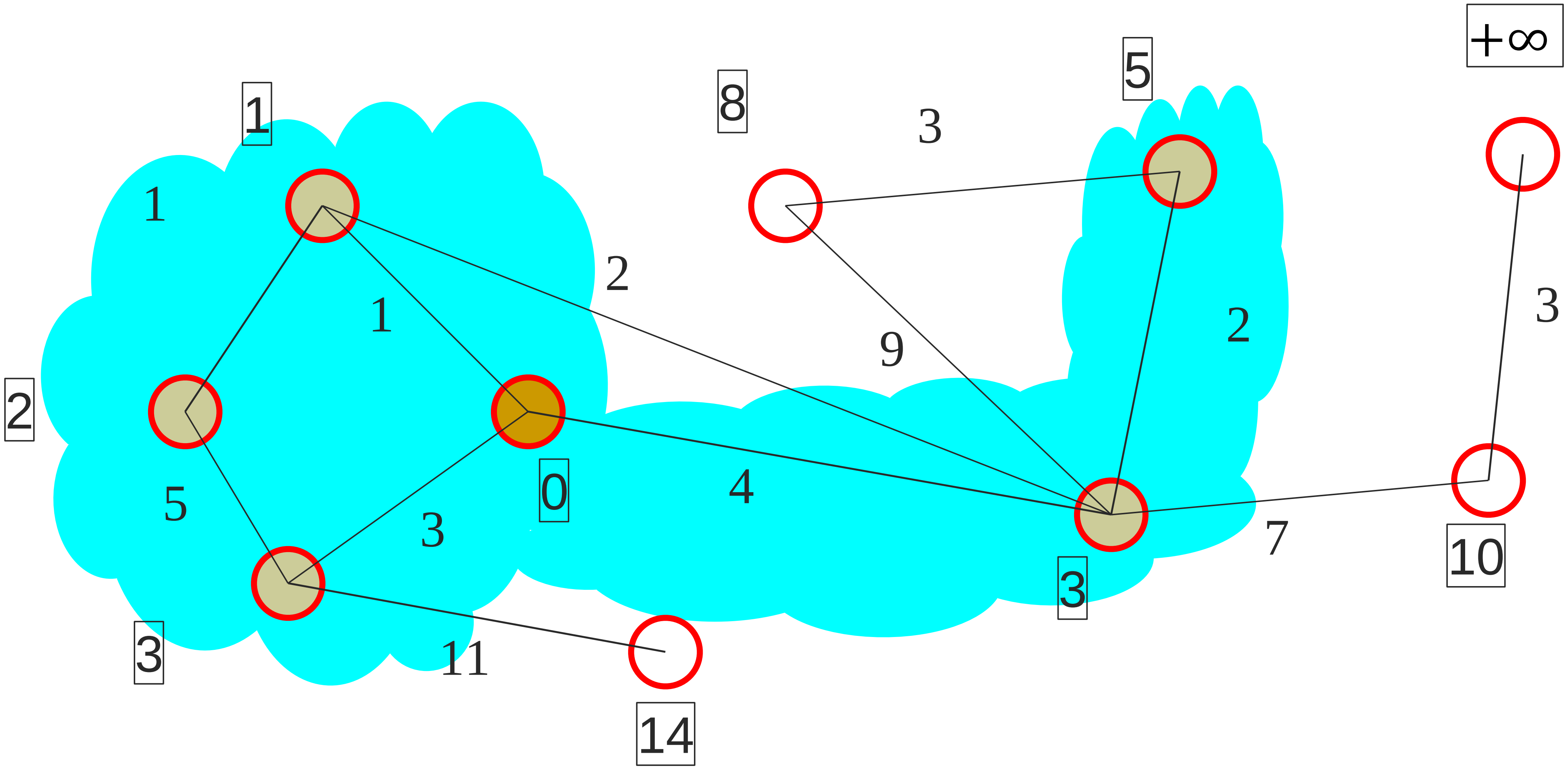
Dijkstra's algorithm: pick closest vertex u outside C



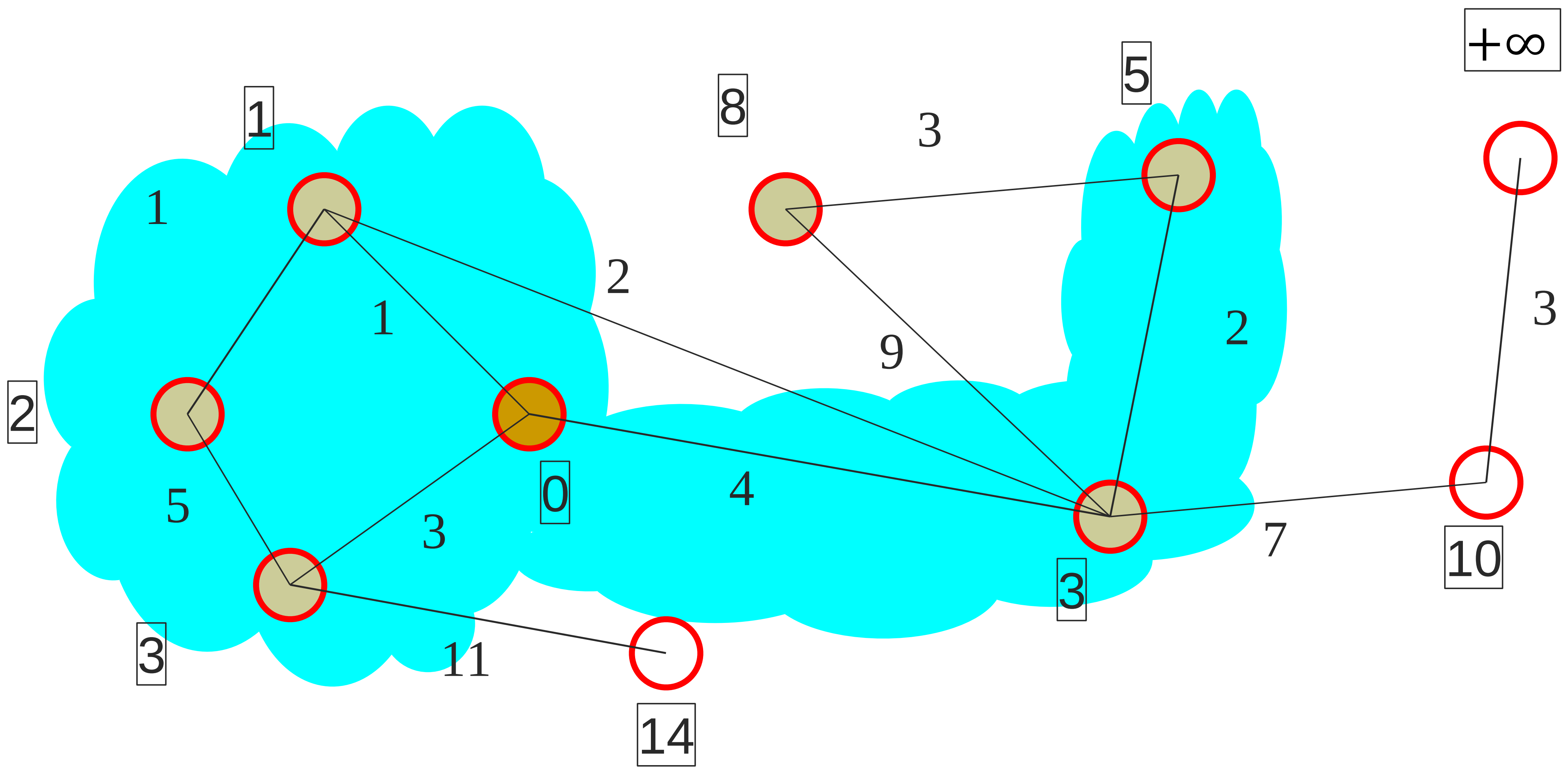
Dijkstra's algorithm: pull u into C



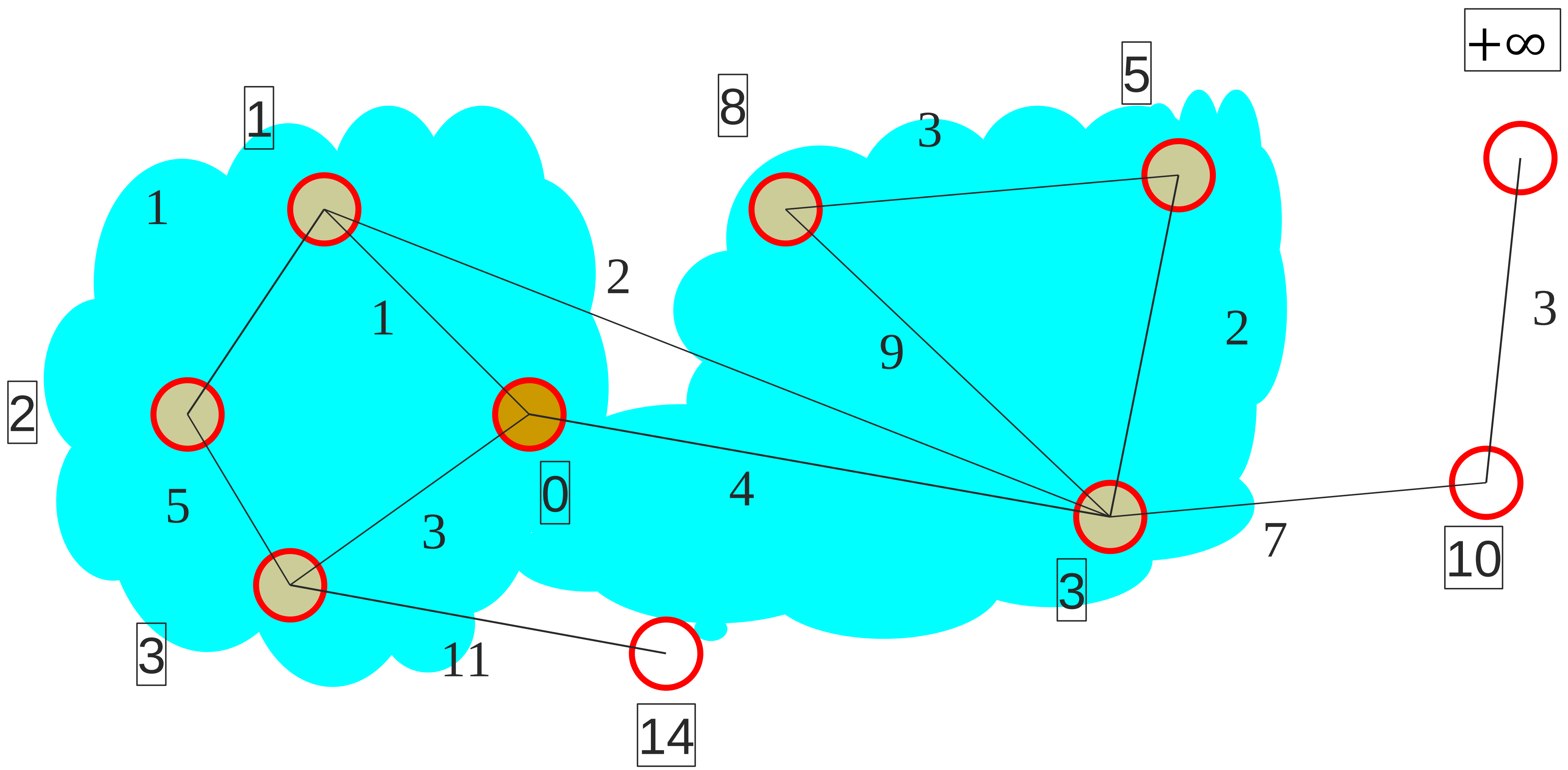
Dijkstra's algorithm: update C's neighborhood



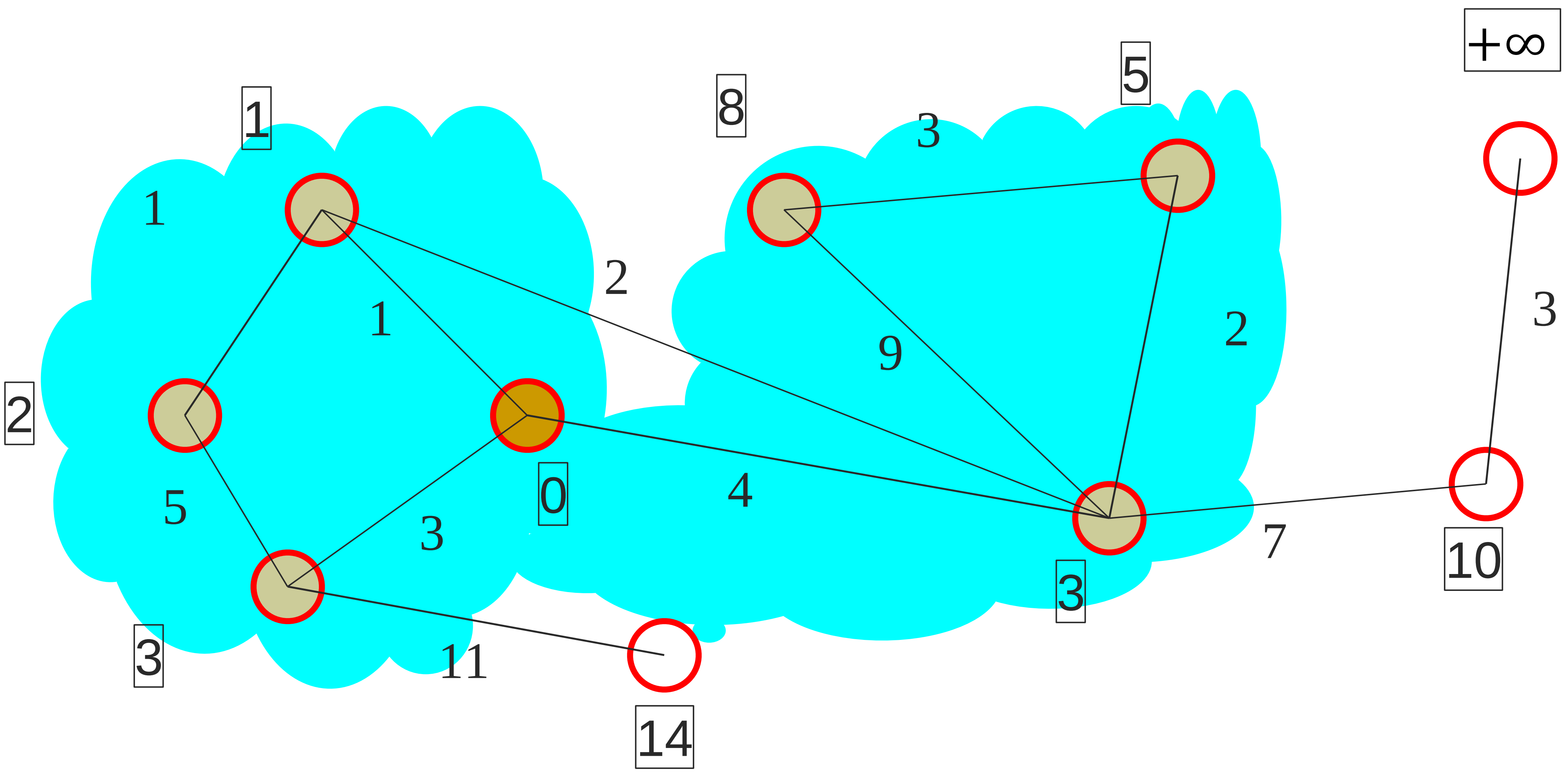
Dijkstra’s algorithm: pick closest vertex u outside C



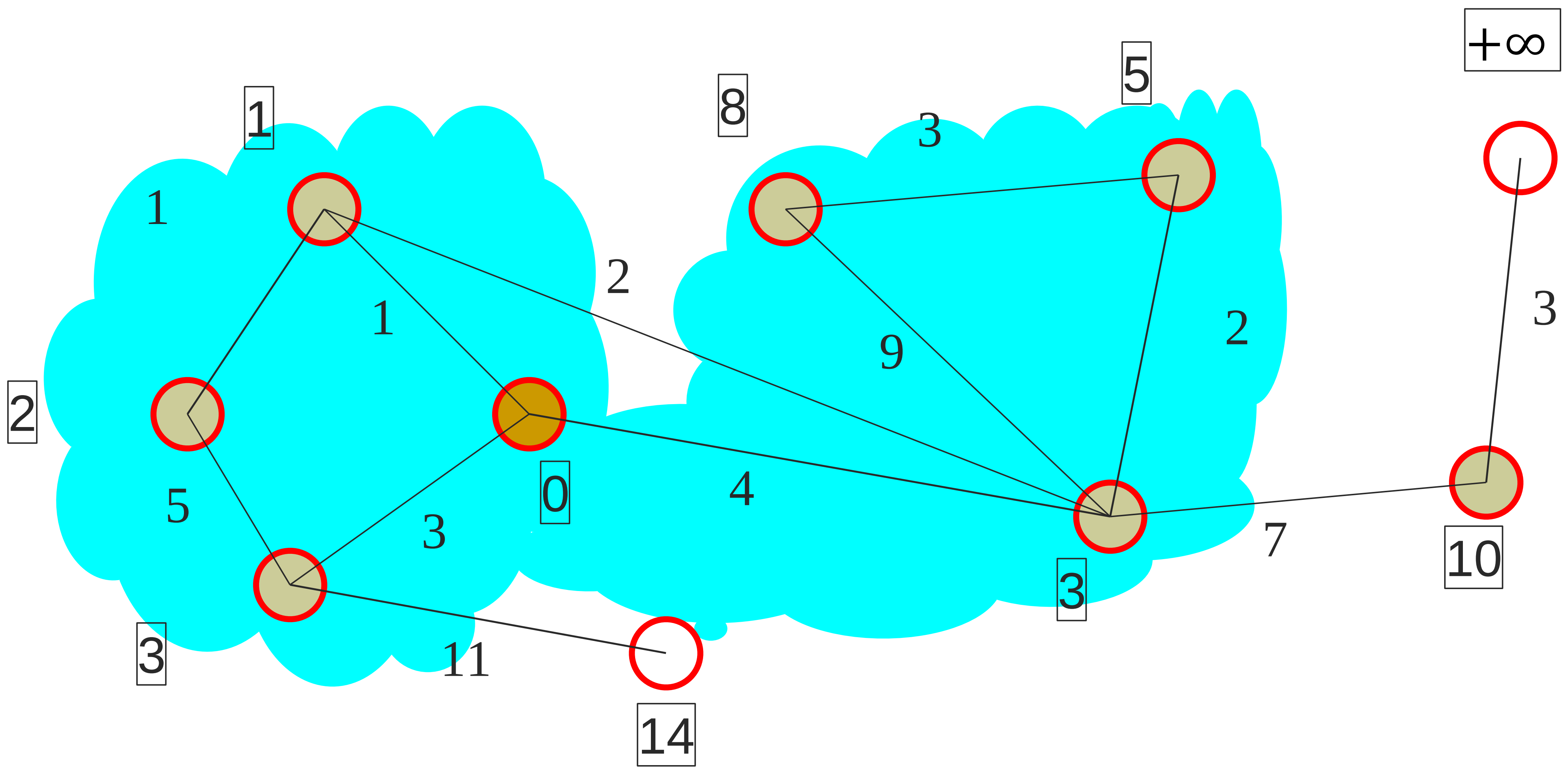
Dijkstra's algorithm: pull u into C



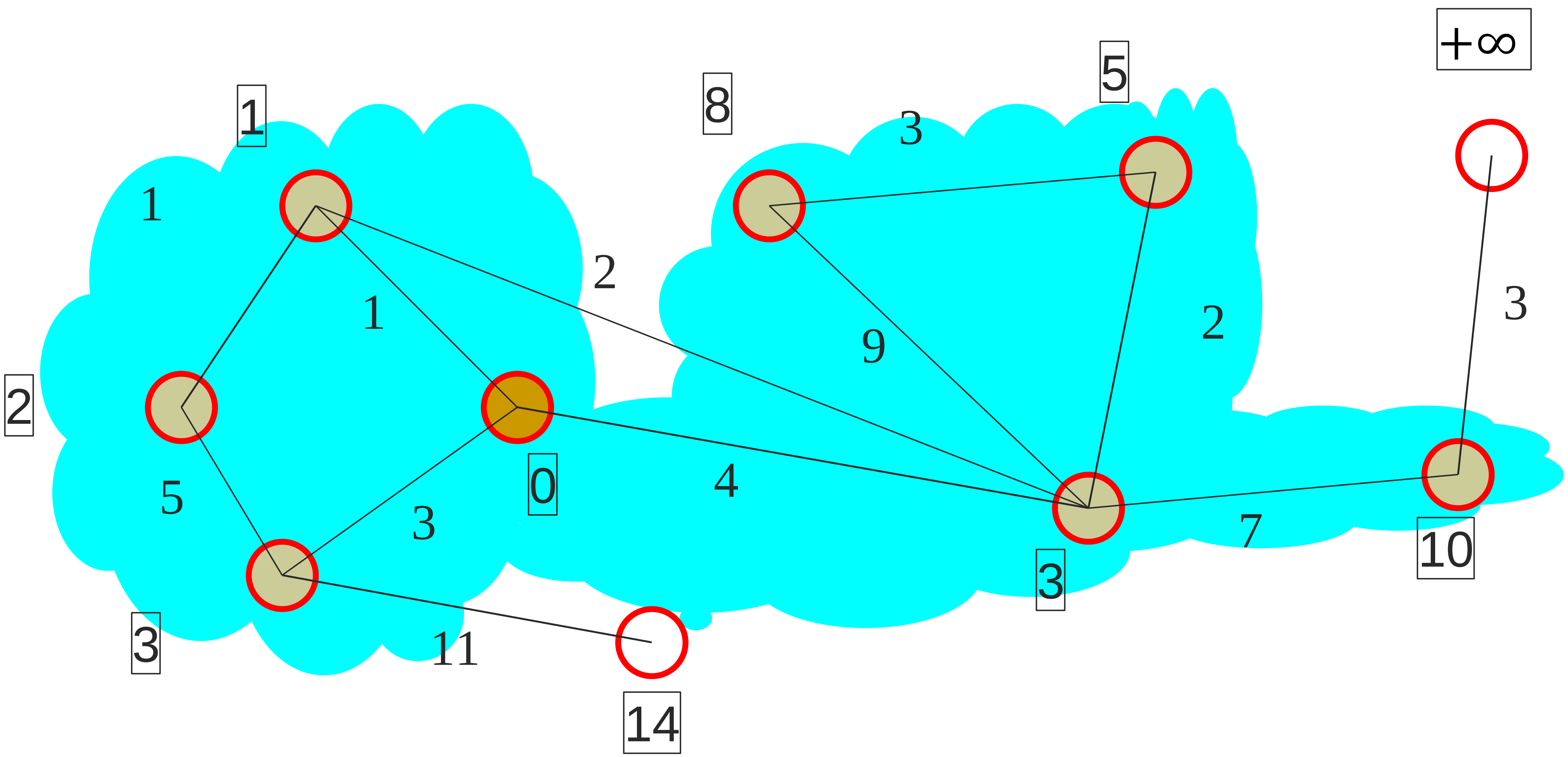
Dijkstra's algorithm: update *C's* neighborhood



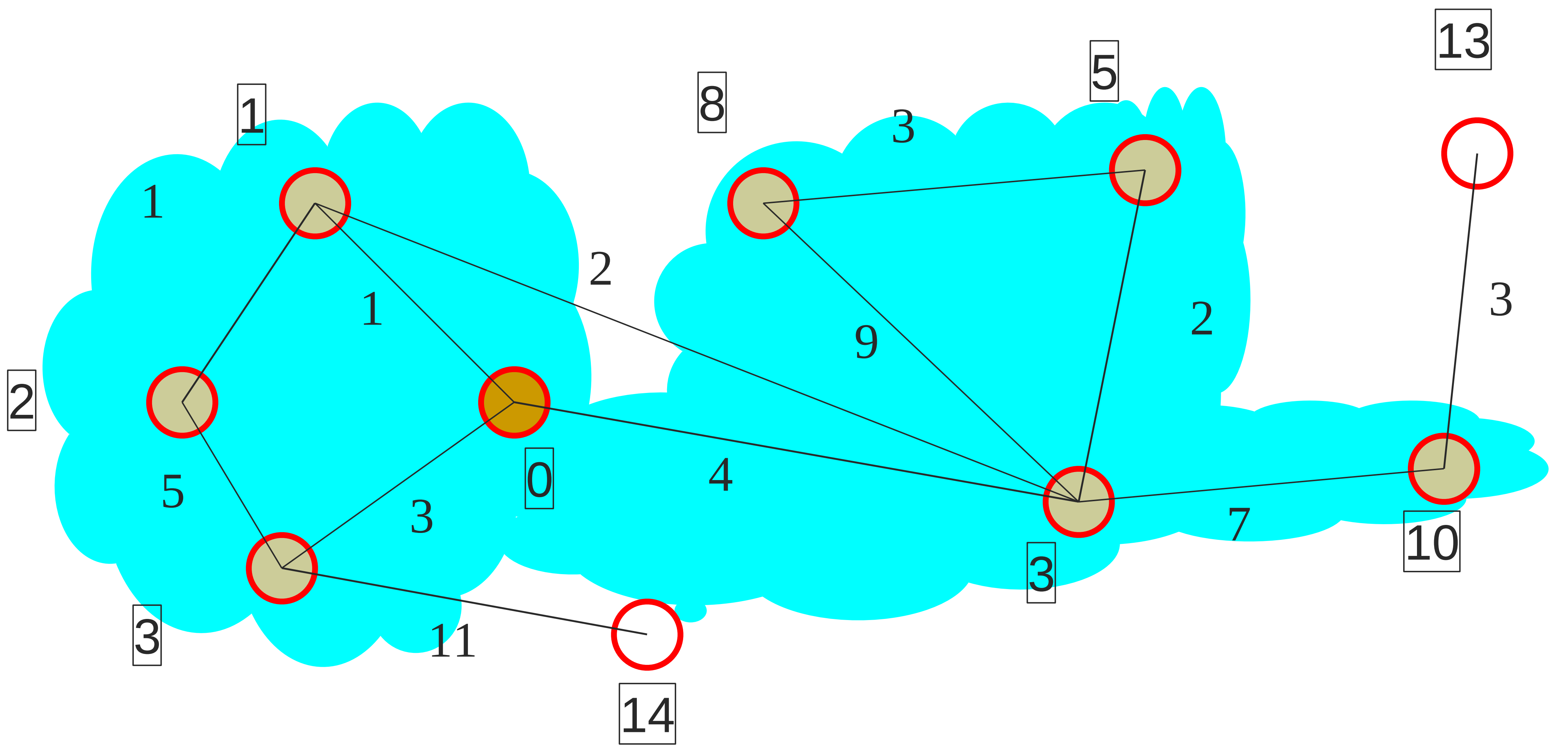
Dijkstra’s algorithm: pick closest vertex u outside C



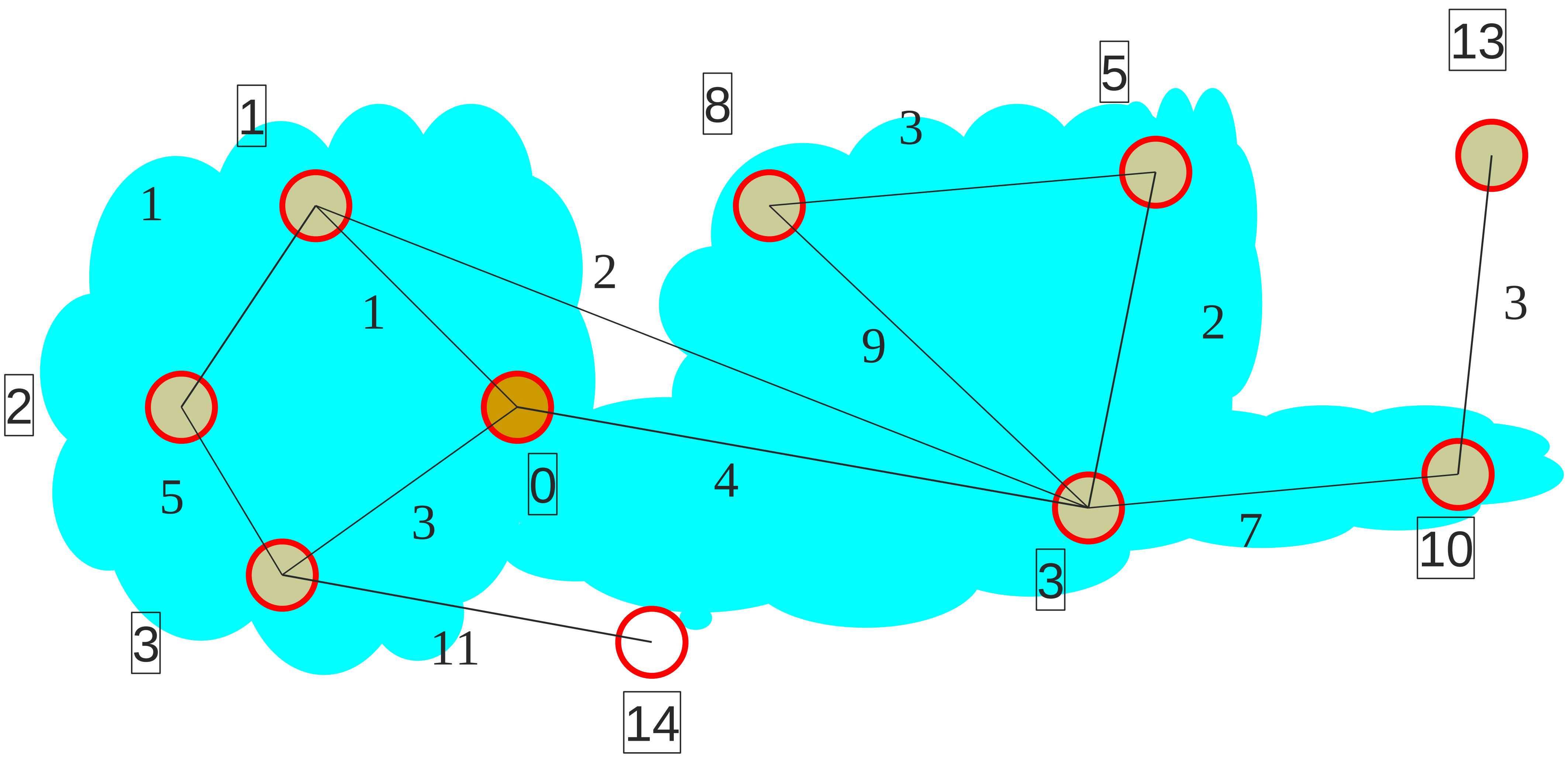
Dijkstra's algorithm: pull u into C



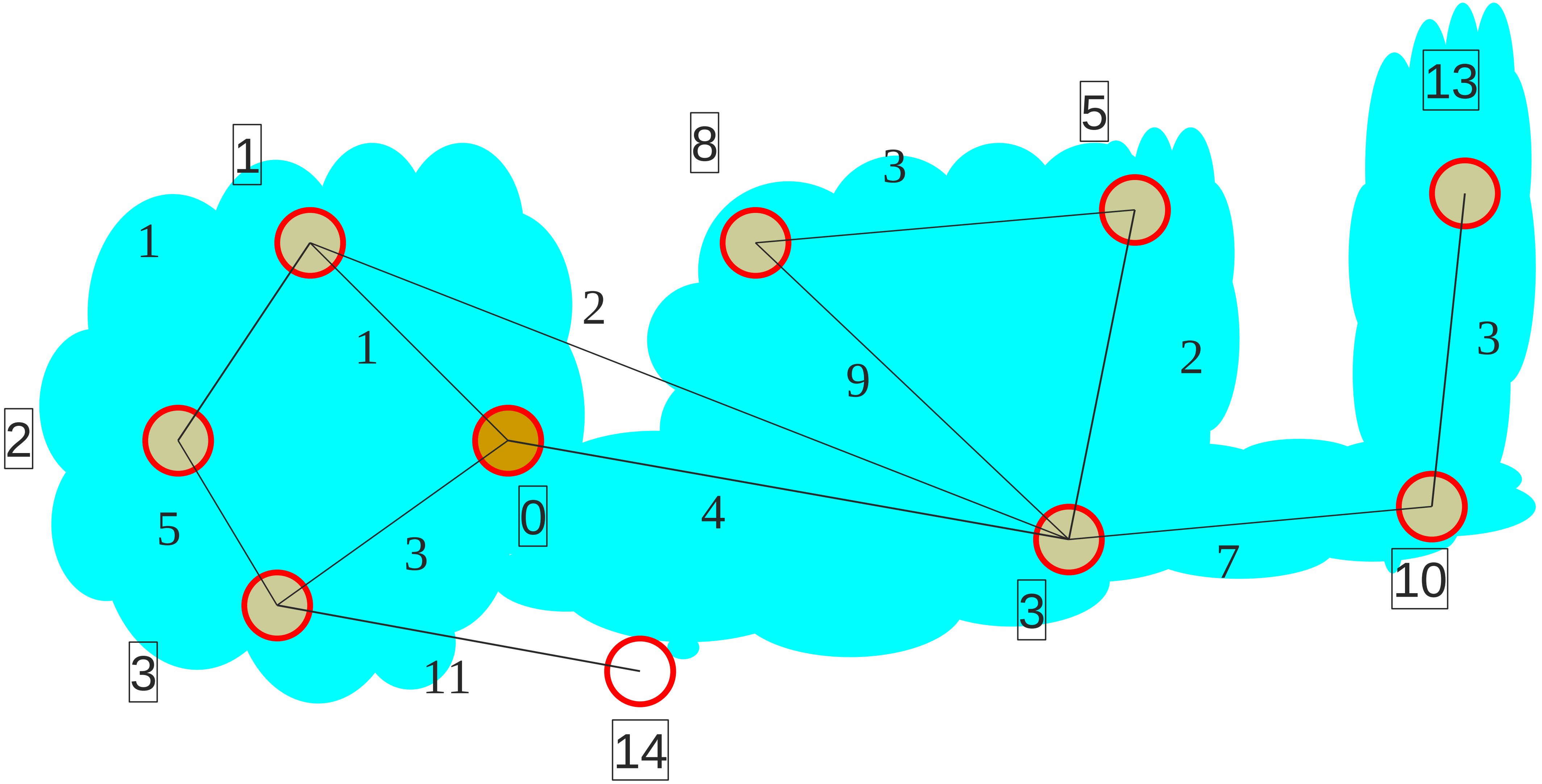
Dijkstra's algorithm: update *C's* neighborhood



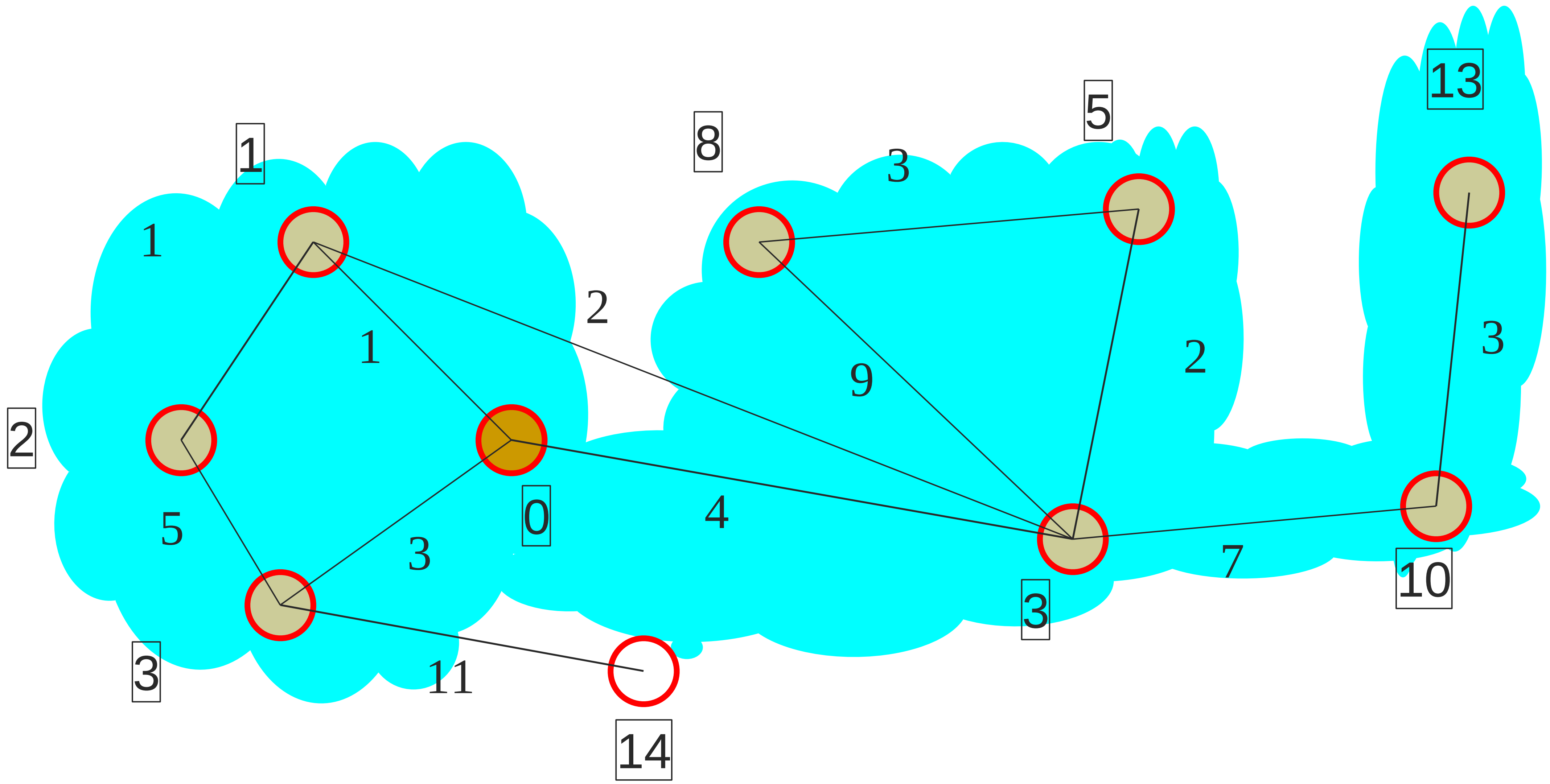
Dijkstra's algorithm: pick closest vertex u outside C



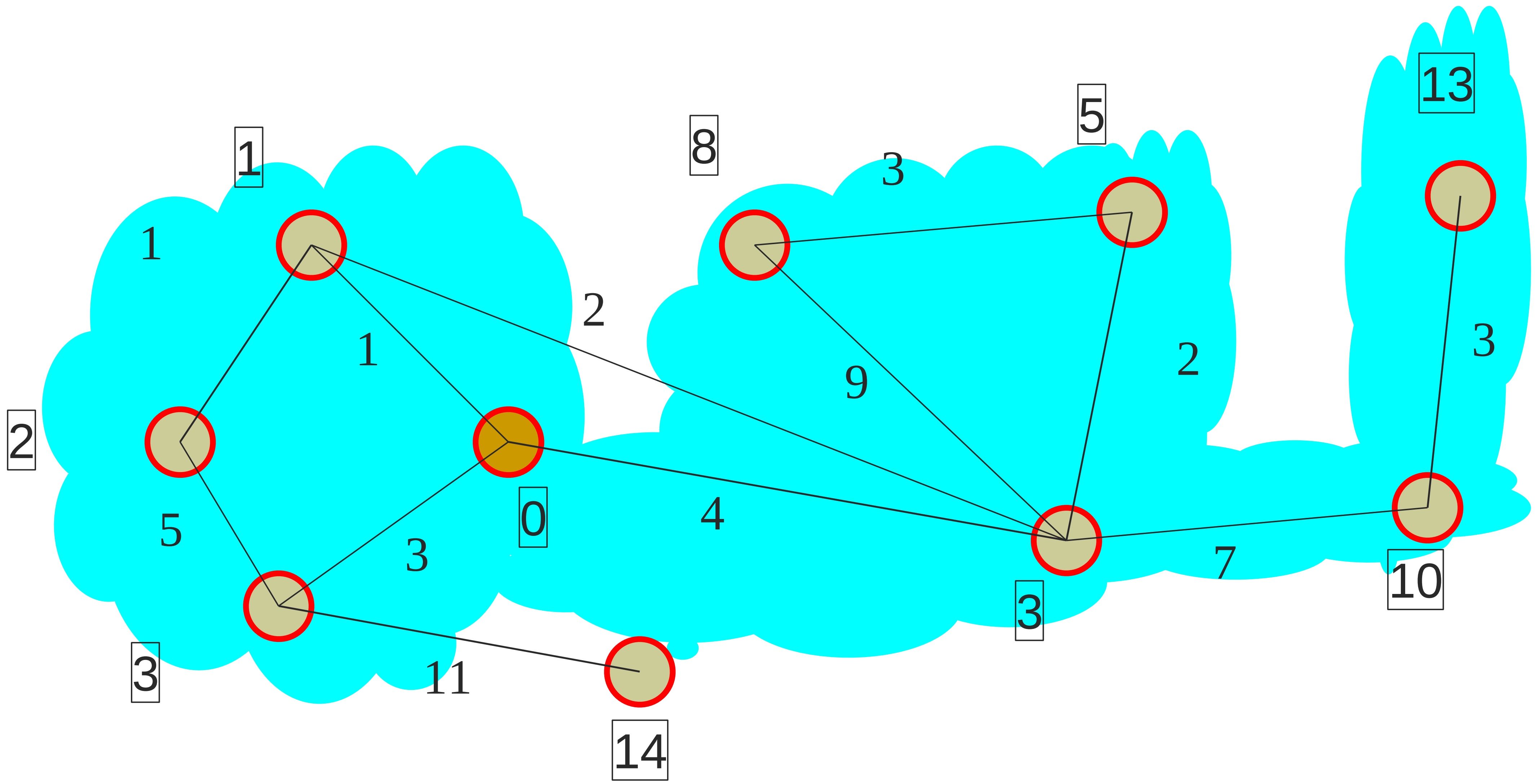
Dijkstra's algorithm: pull u into C



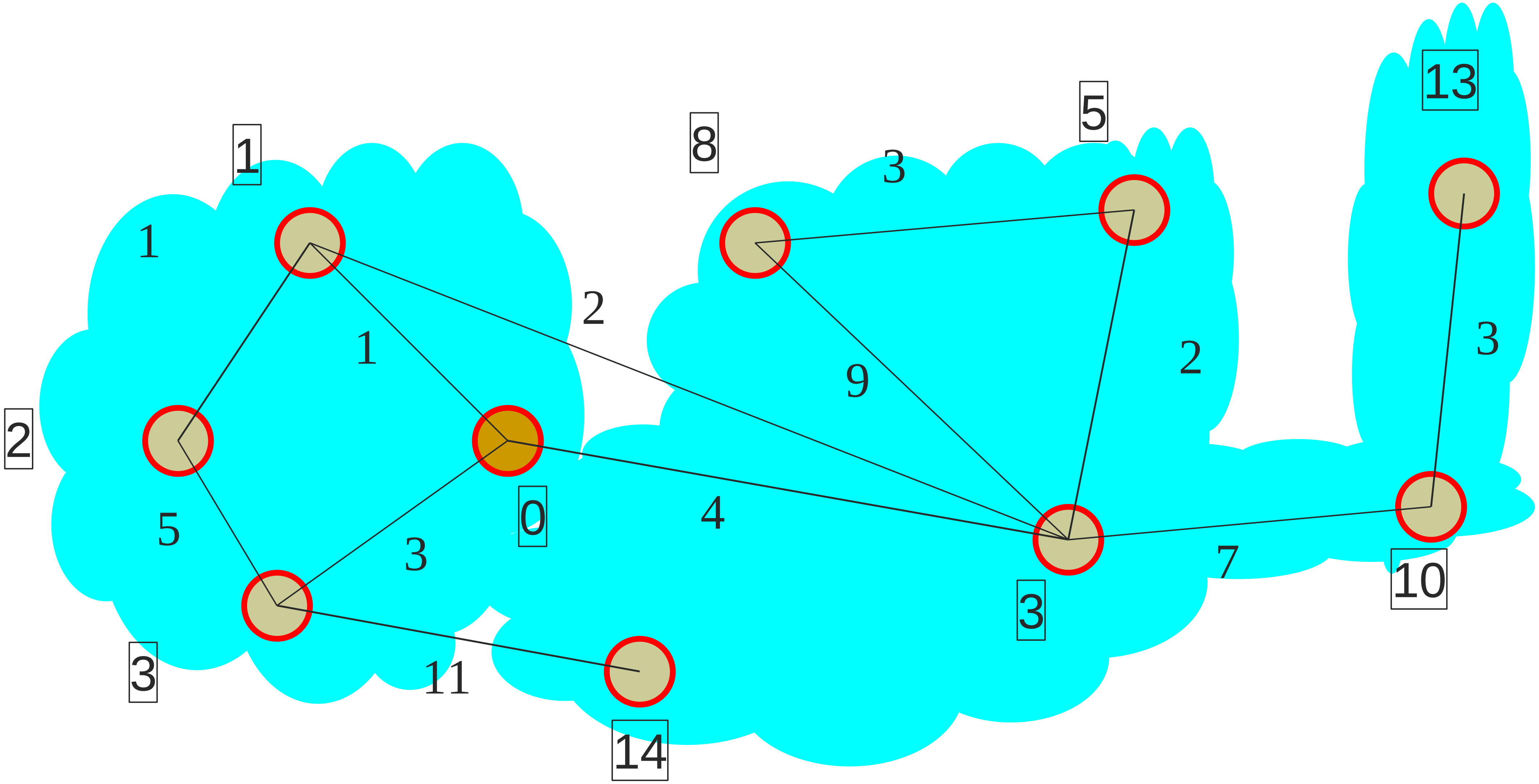
Dijkstra's algorithm: update *C's* neighborhood



Dijkstra's algorithm: pick closest vertex u outside C



Dijkstra’s algorithm: pull u into C



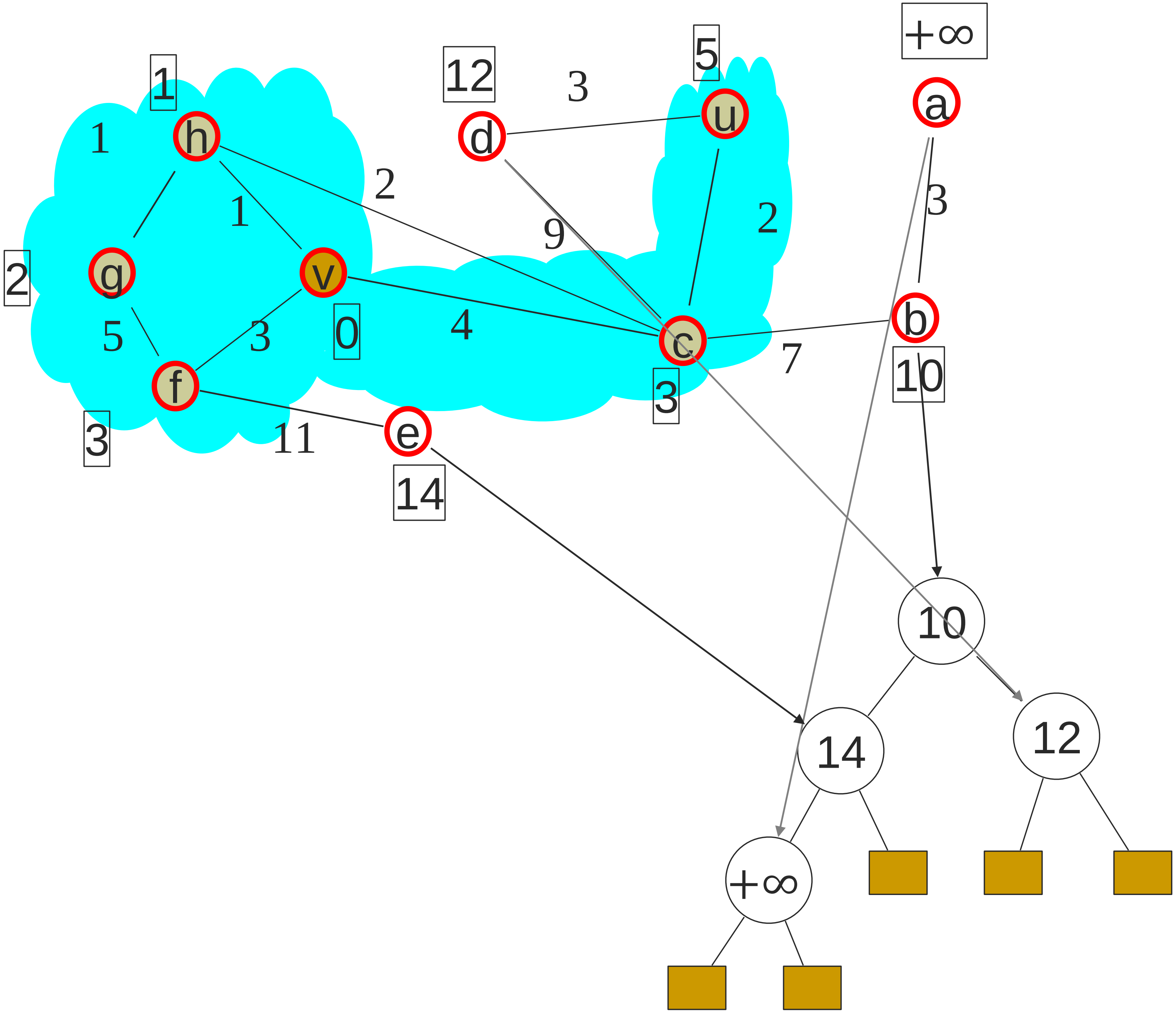
When pulling a neighbour u of C into C

- The value associated with u denotes the length of a shortest path from v to u
- For any vertex x not in the cloud
  - the value associated with x denotes a shortest path from v to x without the use of other vertices outside of the cloud
  - $+\infty$ denotes that the vertex cannot be reached yet from v via cloud vertices only

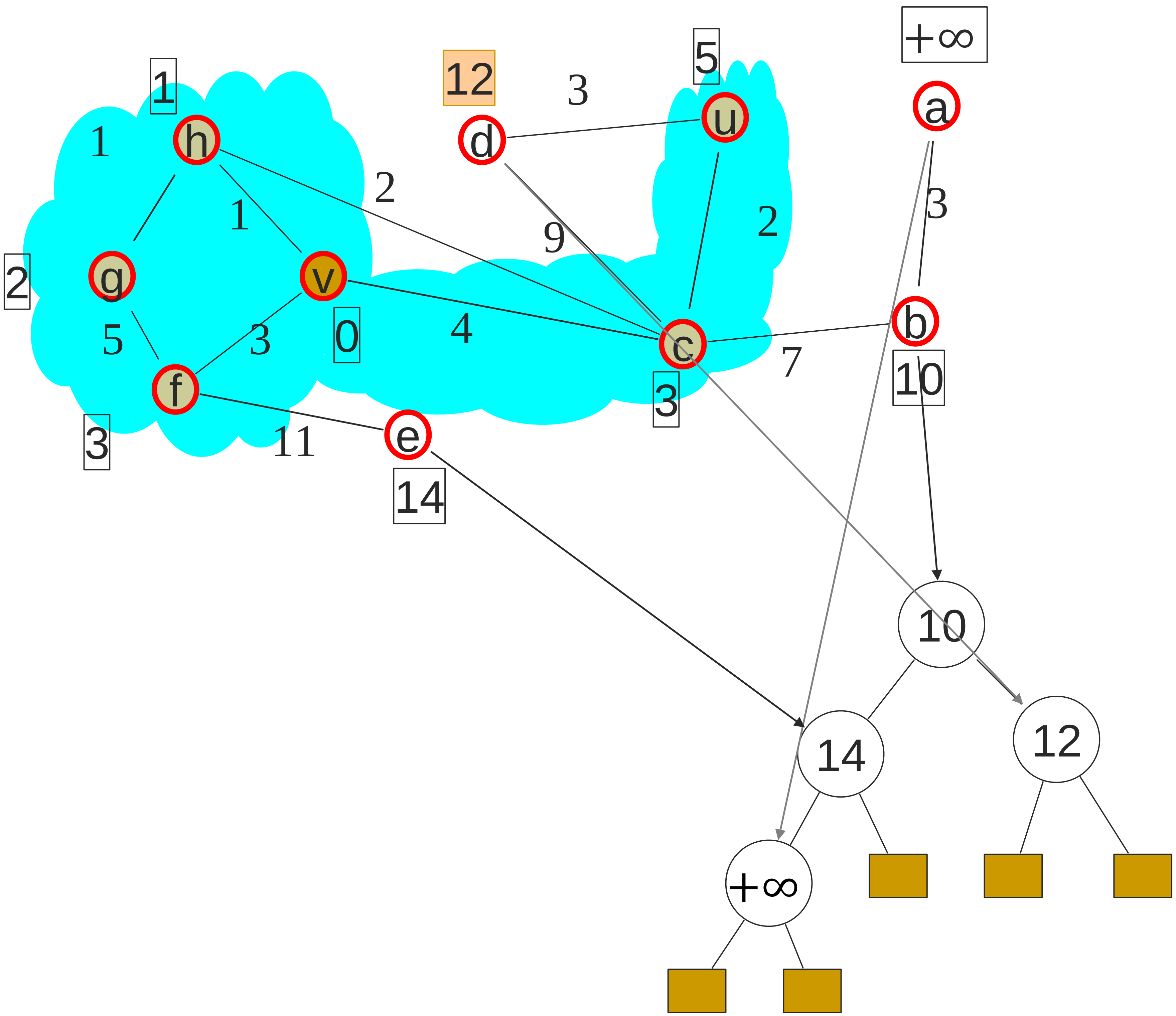
Algorithm DijkstraShortestPaths(G,v)

```
D[v]←0
for each vertex u≠v of G do
   D[u]← +∞
Let Q be a priority queue containing all
   vertices of G using D[.] as keys
while Q is not empty do
   u←Q.removeMin() //u is added to cloud
   for each vertex z∈N(u) with z∈Q do
     if D[u]+w((u,z)) < D[z] then
Relaxation  D[z]←D[u]+w((u,z))
            update z's key in Q to D[z]
return D
```

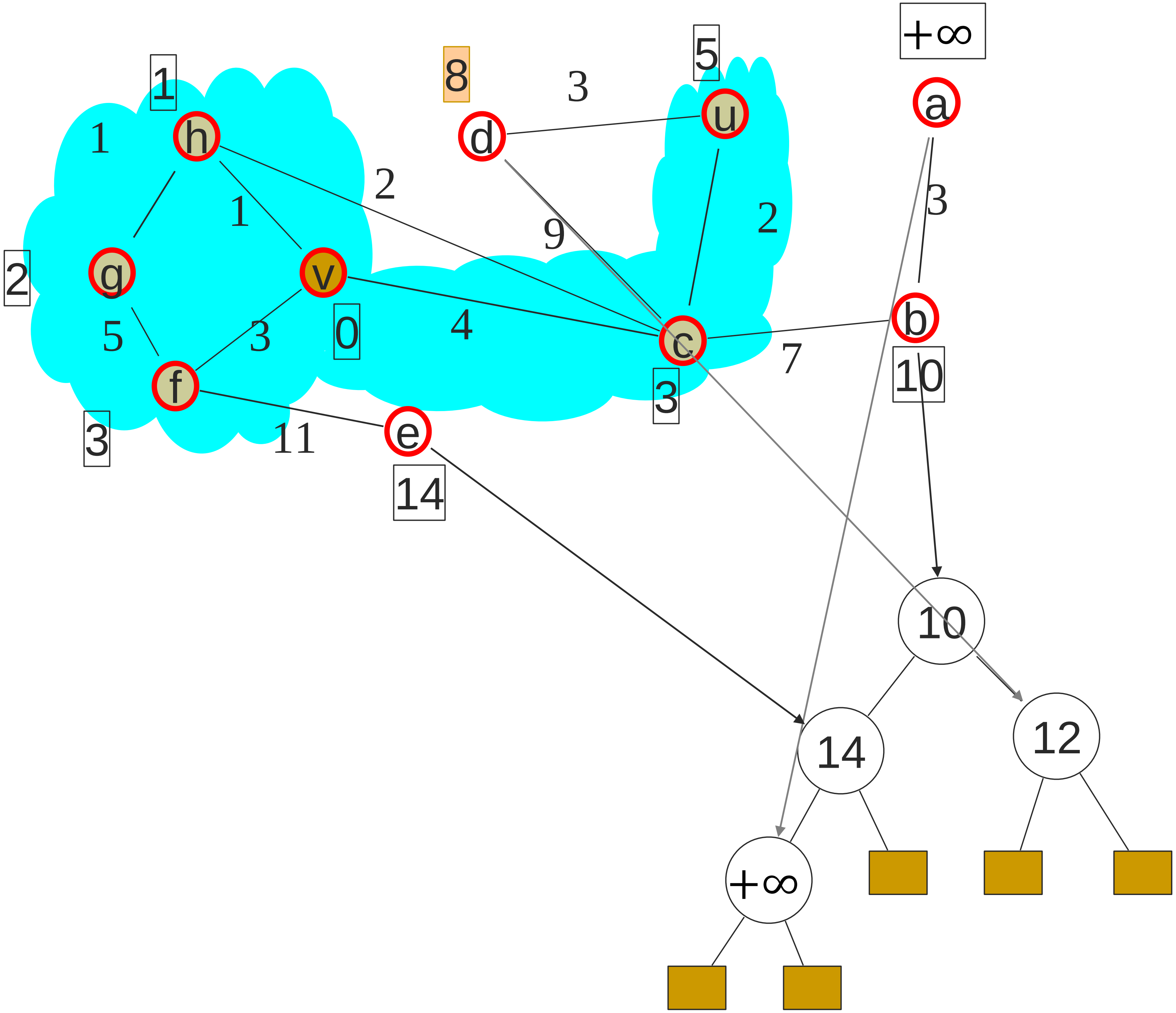
Dijkstra’s algorithm using heaps



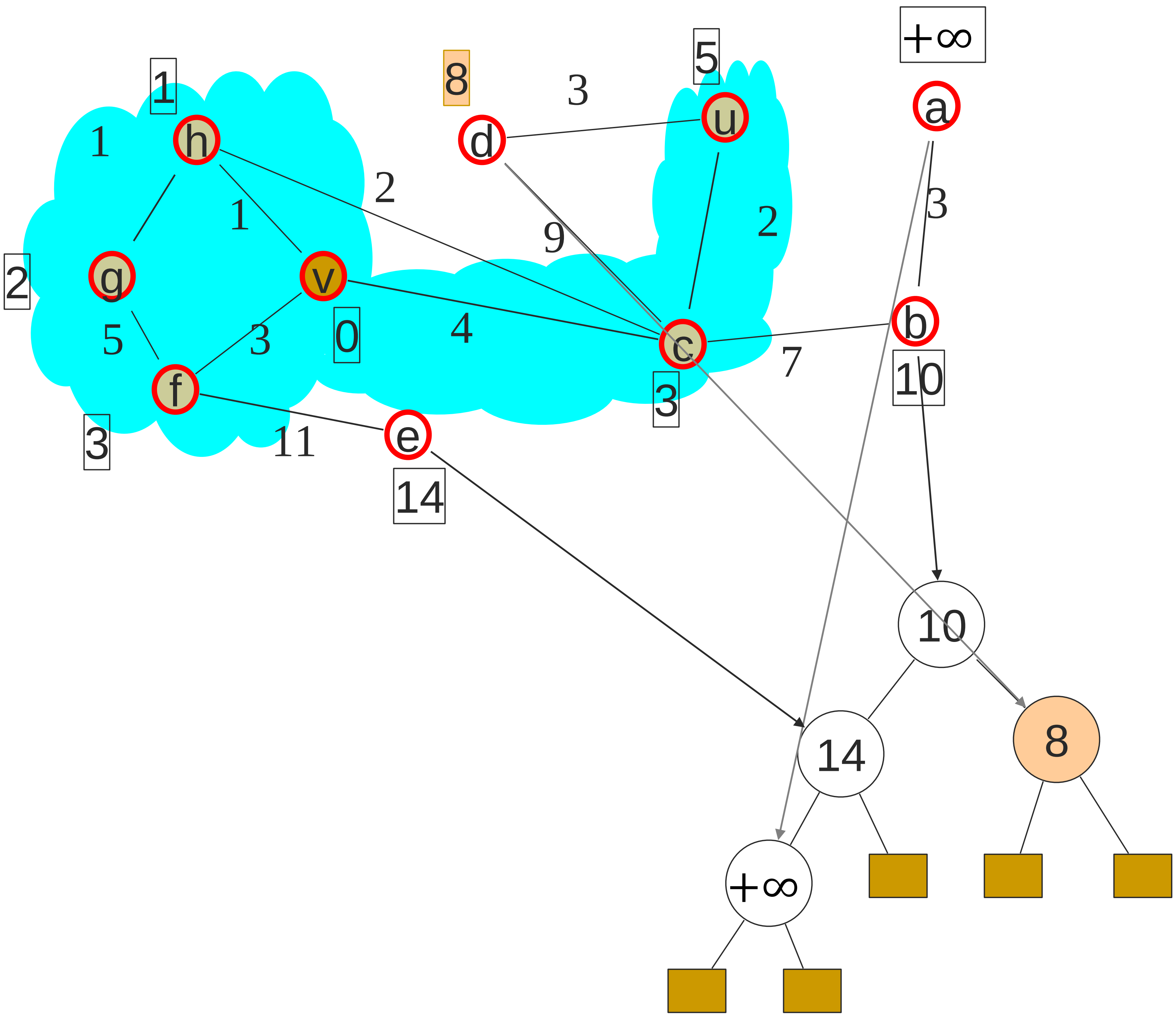
Dijkstra’s algorithm using heaps



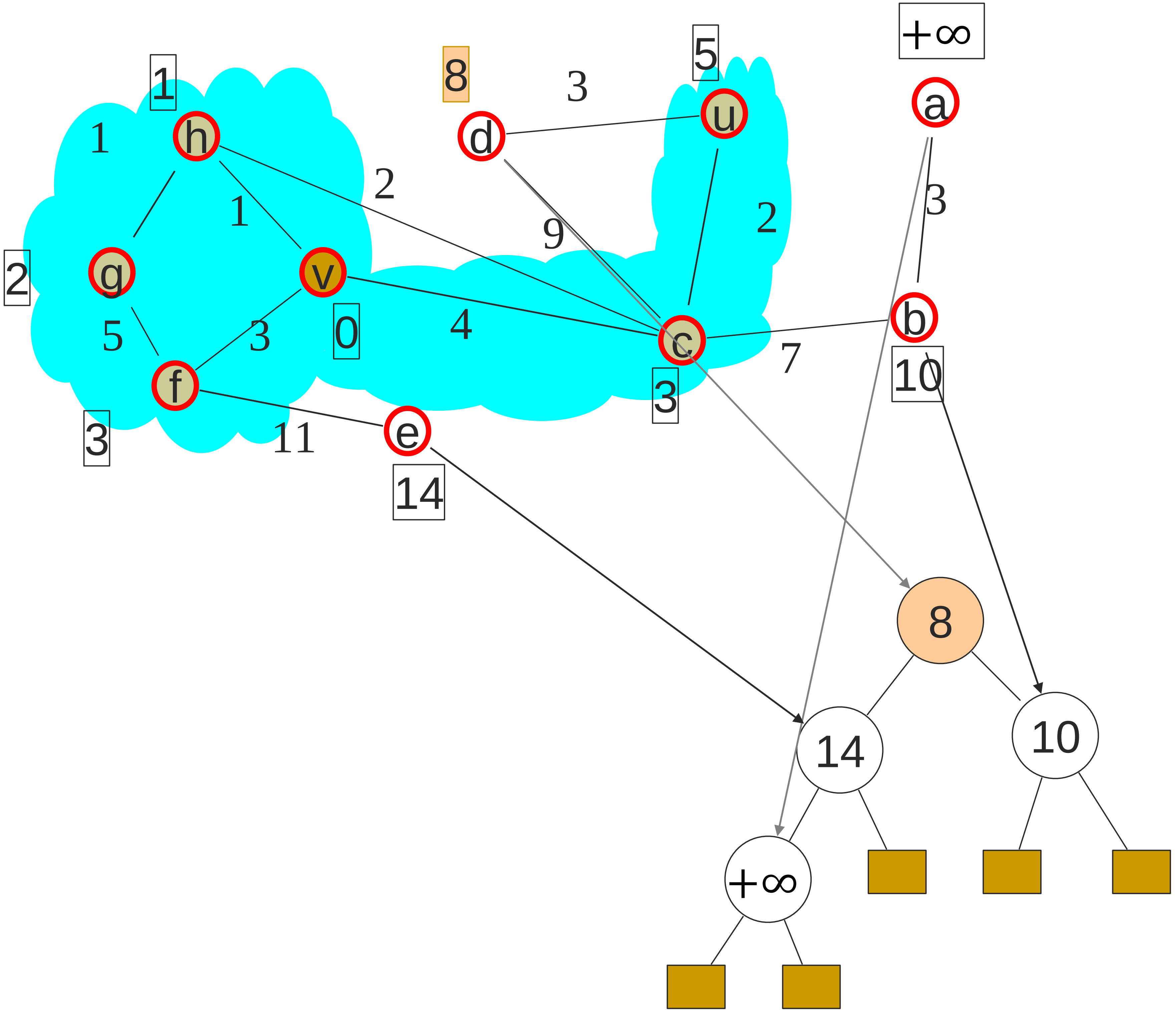
Dijkstra's algorithm using heaps



Dijkstra's algorithm using heaps



Dijkstra’s algorithm using heaps



Running time

```
D[v]←0
for each vertex u≠v of G do
   D[u]← +∞
Let Q be a priority queue containing all
   vertices of G using D[.] as keys
while Q is not empty do
   u←Q.removeMin() //u is added to cloud
   for each vertex z∈N(u) with z∈Q do
     if D[u]+w((u,z)) < D[z] then
         D[z]←D[u]+w((u,z))
         update z's key in Q to D[z]
return D
```

Relaxation

Running time for $G=(V,E)$ with $|V|=n$ and $|E|=m$

- Insertion of vertices in priority queue Q
  - $O(n)$ when using bottom-up heap construction
- While loop:
  - Per iteration:
    - Remove vertex from Q $O(\log n)$
    - Relaxation $O(\deg(u) \log(n))$
  - $\sum_{u \in G} \big(1 + \deg(u)\big) \log n$ is $O((n+m)\log n)$

- Overall running time: $O(m \log n)$

In real life applications

- Often the graphs are <u>sparse</u>
- Then $O(m \log n)$ may be $O(n \log n)$

Algorithm DijkstraShortestPaths(G,v)

Correctness

```
D[v]←0
for each vertex u≠v of G do
   D[u]← +∞
Let Q be a priority queue containing all
   vertices of G using D[.] as keys
while Q is not empty do
   u←Q.removeMin() //u is added to cloud
   for each vertex z∈adj(u) with z∈Q do
     if D[u]+w((u,z)) < D[z] then
        D[z]←D[u]+w((u,z))
        update z's key in Q to D[z]
return D
```

Relaxation

Correctness of Dijkstra's algorithm

- To show: whenever u is pulled into cloud C, D[u] stores the length of a shortest path from v (the starting vertex) to u
- Definition: For vertices u and v in G, we denote with $d(v,u)$ the length of a shortest path from v to u.

Whenever u is pulled into cloud C, $D[u]$ stores the length of a shortest path from v to u

Lemma:

Let v_m be an outside vertex in $V - C$ such that $D[m]$ is minimum. Then $D[m] \leq D[j]$, for all $j \in V - C$.

That is, <u>**$D[m]$ is a lower bound on the length of the shortest path from v to any vertex in $V - C$**</u>.

Proof for Lemma:

Assume for the purpose of contradiction that there is some vertex $v_j \in V - C$, with $d_j < D[m]$. Since $D[m] \leq D[j]$, we have $d_j < D[j]$. So any true shortest path P from v to v_x is shorter than the length of a shortest path using only vertices from C as intermediates.

Then P must use at least one vertex from $V - C$ as an intermediate. Let v_x be the first vertex from $V - C$ along P, as we go from v to v_j, so that the predecessor of v_x along the path belongs to C. Since v_x comes before v_j, $D[x] \leq d_j < D[m]$. But v_m was defined to be a vertex of $V - C$ such that $D[m]$ is minimum. This is a contradiction.

So our assumption that there is some v_j such that $d_j < D[m]$ has to be wrong.

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<Double> 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;                                    add v to T
   for (Edge e : G.adj(v)) {                            for each edge e = v–w, add to
      int w = e.other(v);                               PQ if w not already in T
      if (marked[w]) continue;
      if (e.weight() < distTo[w]) {
          edgeTo[w] = e;                                add edge e to tree
          distTo[w] = e.weight();
          if (pq.contains(w)) pq.changeKey(w, distTo[w]);   Update distance to w or
          else pq.insert(w, distTo[w]);                     Insert distance to w
       }
   }
}
public Iterable<Edge> edges(){                          Create the mst
   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; }
```

Djikstra's algorithm: eager implementation

```
public class DijkstraUndirectedSP {
   private Edge[] edgeTo;                      // last edge from on path to v
   private double[] distTo;                    // distance to v from s
   private IndexMinPQ<Double> pq;     // eligible crossing edges

    public DijkstraUndirectedSP(WeightedGraph G, int s) {
        edgeTo = new Edge[G.V()];
        distTo = new double[G.V()];

        for(int v = 0; v < G.V(); v++)
              distTo[v] = Double.POSITIVE_INFINITY;
        pq = new IndexMinPQ<Double>(G.V());
        distTo[s] = 0.0;
        pq.insert(s, distTo[s]);
        while(!pq.isEmpty())
                relax(G, pq.delMin());
   }
```

assume G is connected

repeatedly delete the edge e = v–w from PQ that is closest to s.

Dijkstra's algorithm: eager implementation

```
private void relax(WeightedGraph G, int v) {

   for (Edge e : G.adj(v)) {
      int w = e.other(v);

      if (distTo[v] + e.weight() < distTo[w]) {
           edgeTo[w] = e;
           distTo[w] = distTo[v] + e.weight();
           if (pq.contains(w)) pq.changeKey(w, distTo[w]);
           else pq.insert(w, distTo[w]);
       }
   }
}
public Iterable<Edge> edges(){
   Queue<Edge> spt = new Queue<Edge>();
   for (int v = 0; v < edgeTo.length; v++)
      Edge e = edgeTo[v];
      if (e != null) {
         spt.enqueue(e);
      }
   }
   return spt; }
```

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 spt

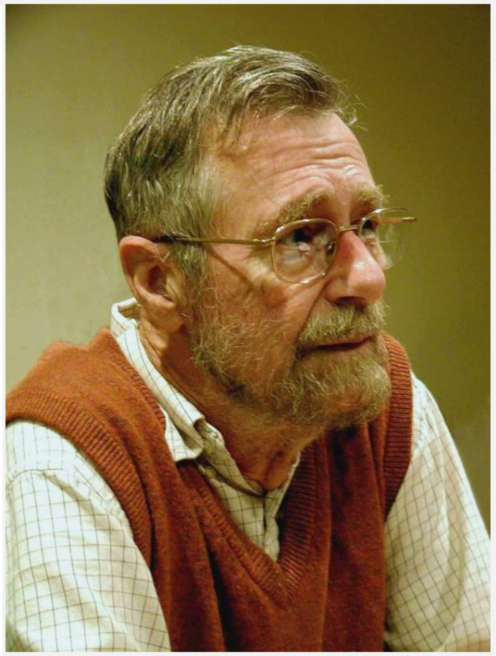
" Do only what only you can do. "

" In their capacity as a tool, computers will be but a ripple on the surface of our culture. In their capacity as intellectual challenge, they are without precedent in the cultural history of mankind. "

" The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence. "

" It is practically impossible to teach good programming to students that have had a prior exposure to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration. "

" APL is a mistake, carried through to perfection. It is the language of the future for the programming techniques of the past: it creates a new generation of coding bums. "



Edsger W. Dijkstra
Turing award 1972

http://www.zazzle.com/dijkstra_on_object_oriented_programming_and_cali_tshirt-235725459155842241