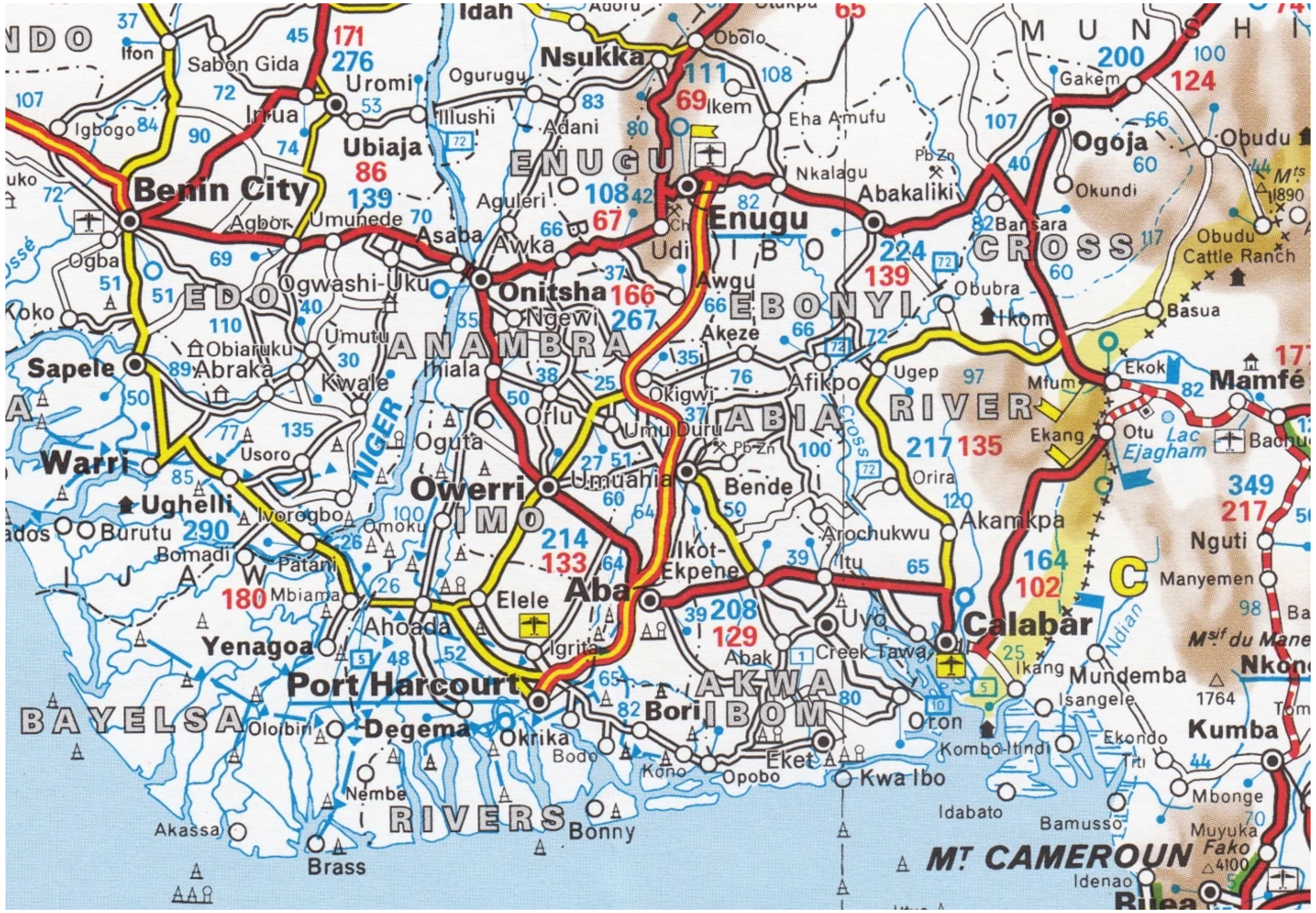
THE UNIVERSITY OF VERMONT
COLLEGE OF ENGINEERING &
MATHEMATICAL SCIENCES

Shortest Path

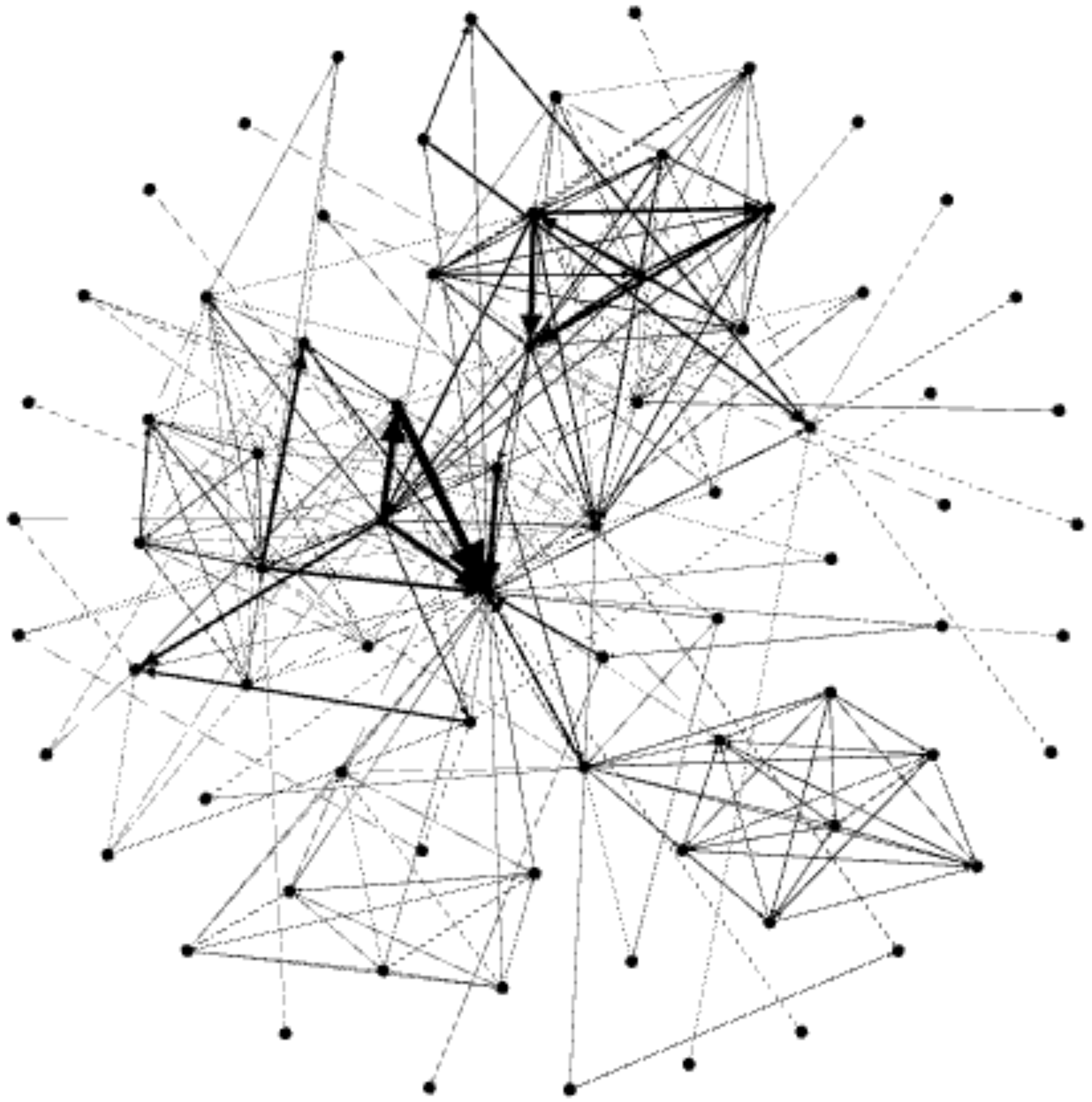


CS 124 / Department of Computer Science

Image copyright © Michelin (map #741, Africa North and West)

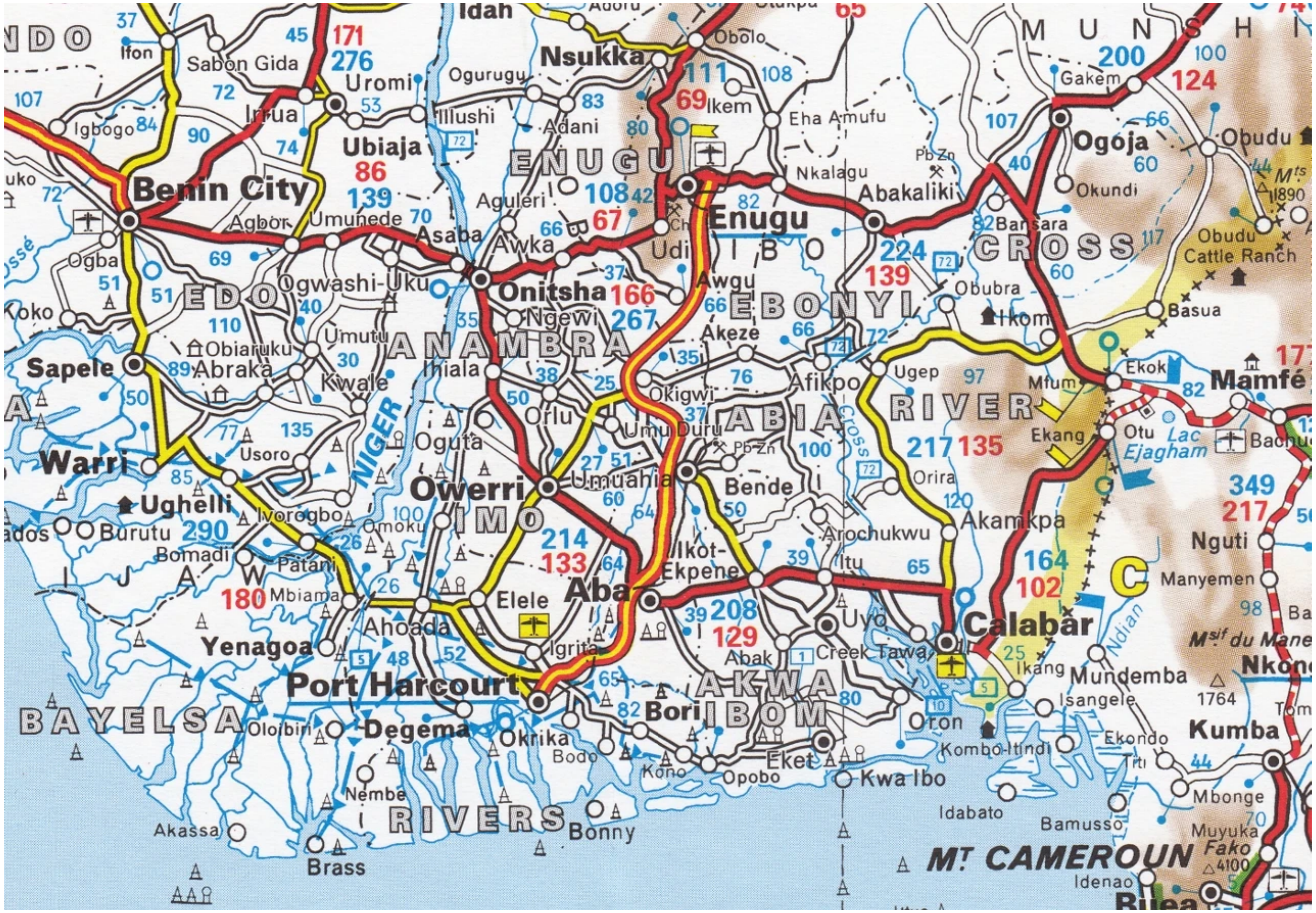
Shortest path

- Vehicle routing
- Network design
- Telecommunications
- Optimization

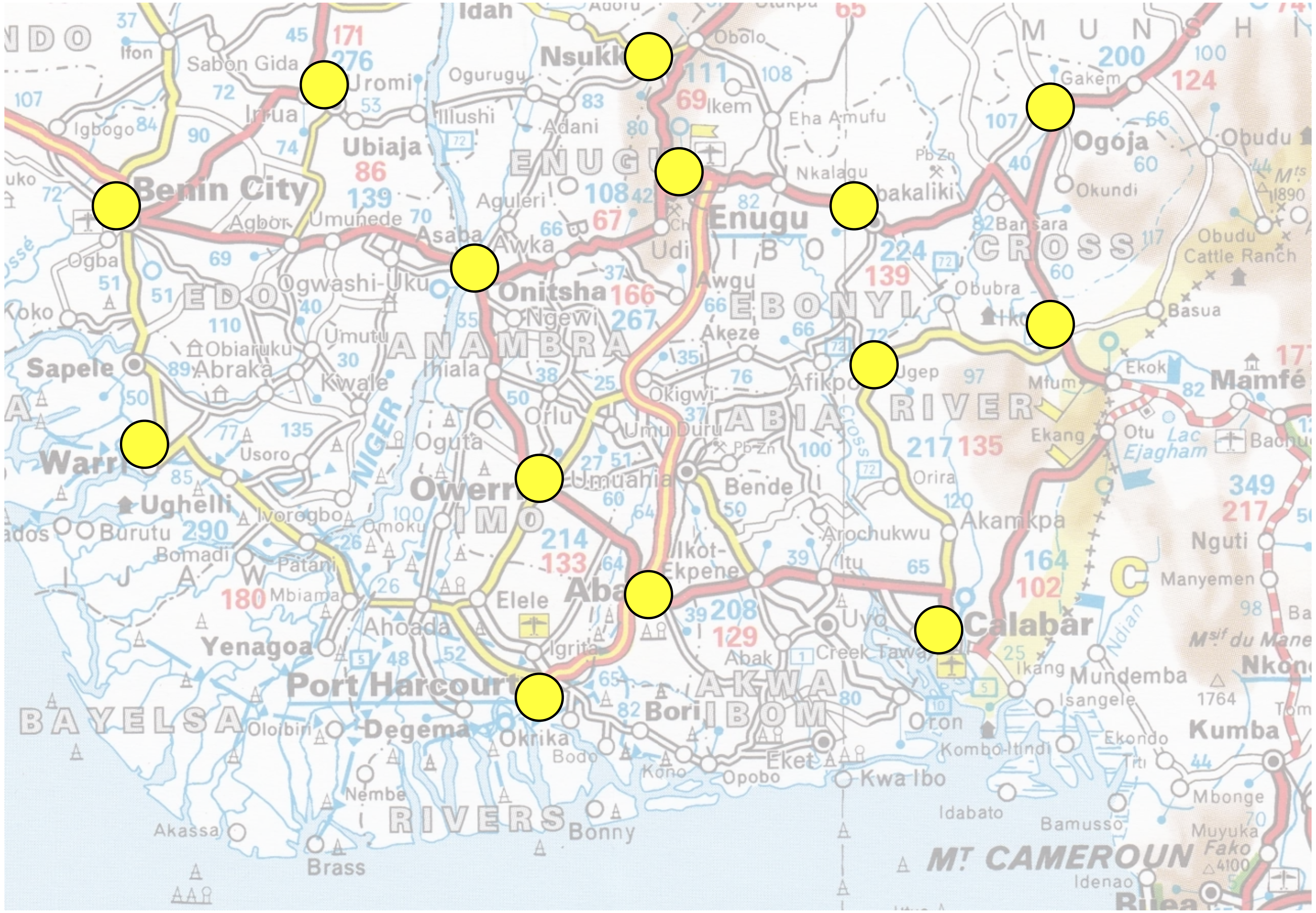


Weighted network of characters in Victor Hugo's *Les Misérables*, from the Stanford GraphBase, Knuth, 1993. Image generated with Gephi.

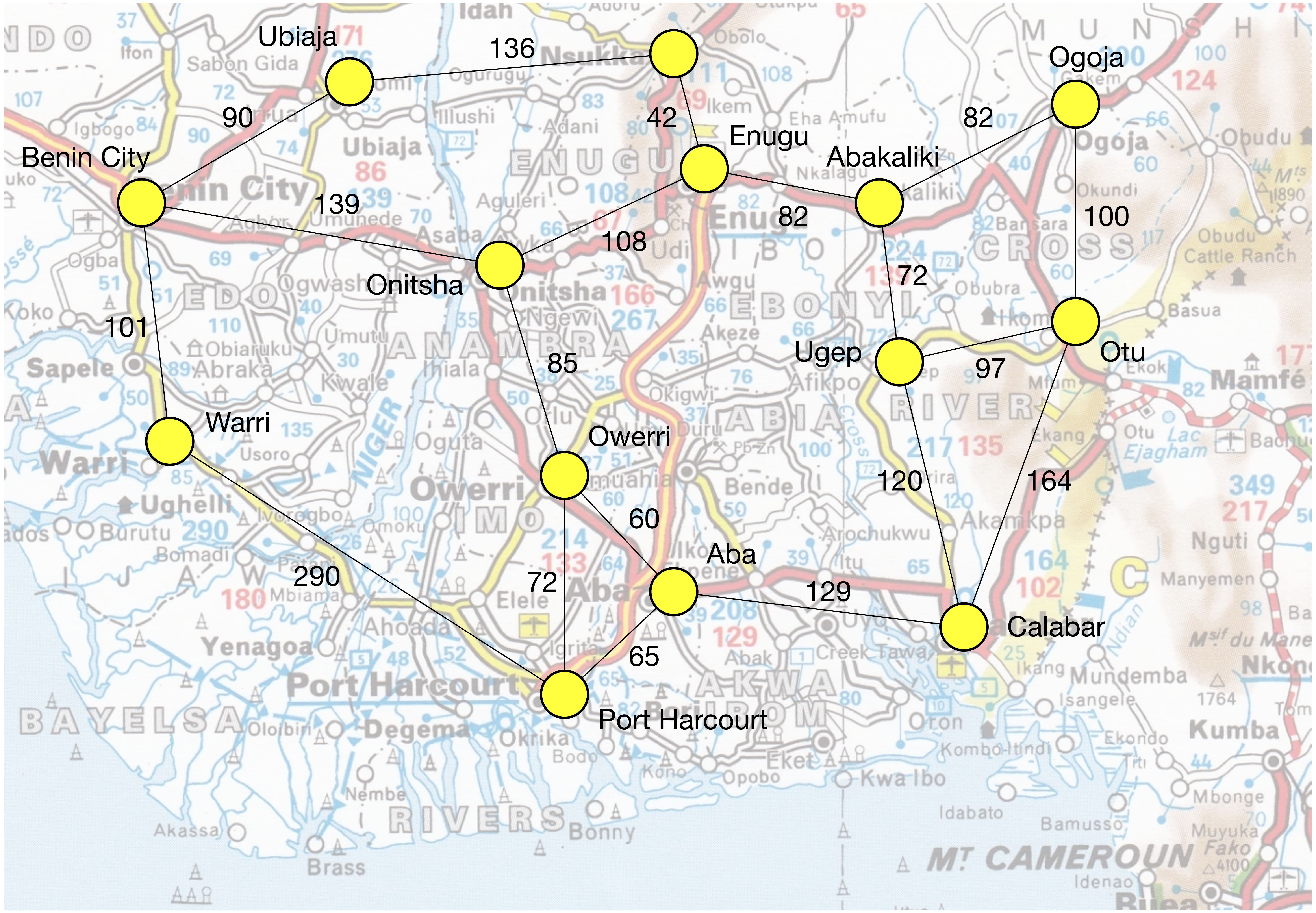
Shortest path



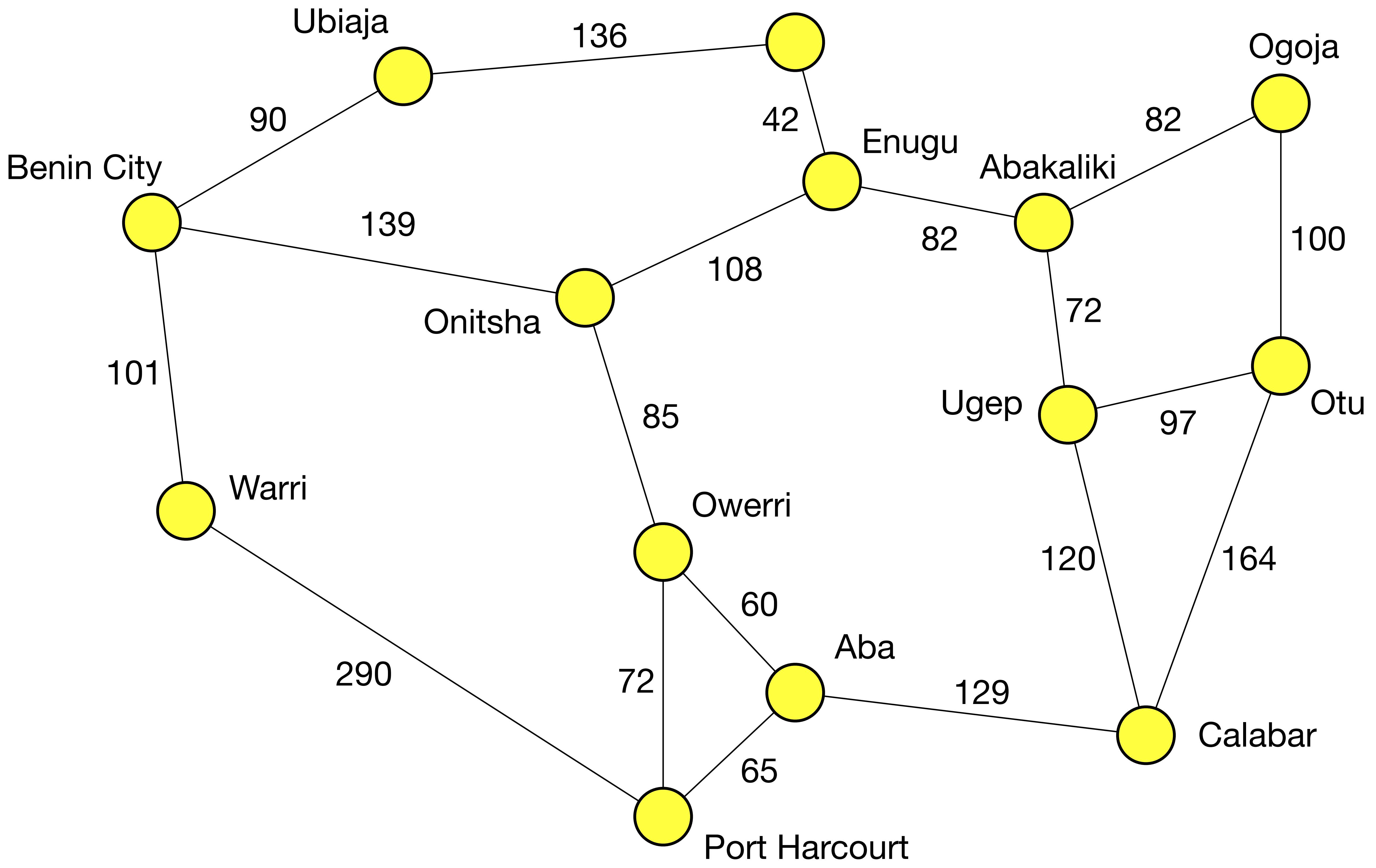
Shortest path



Shortest path



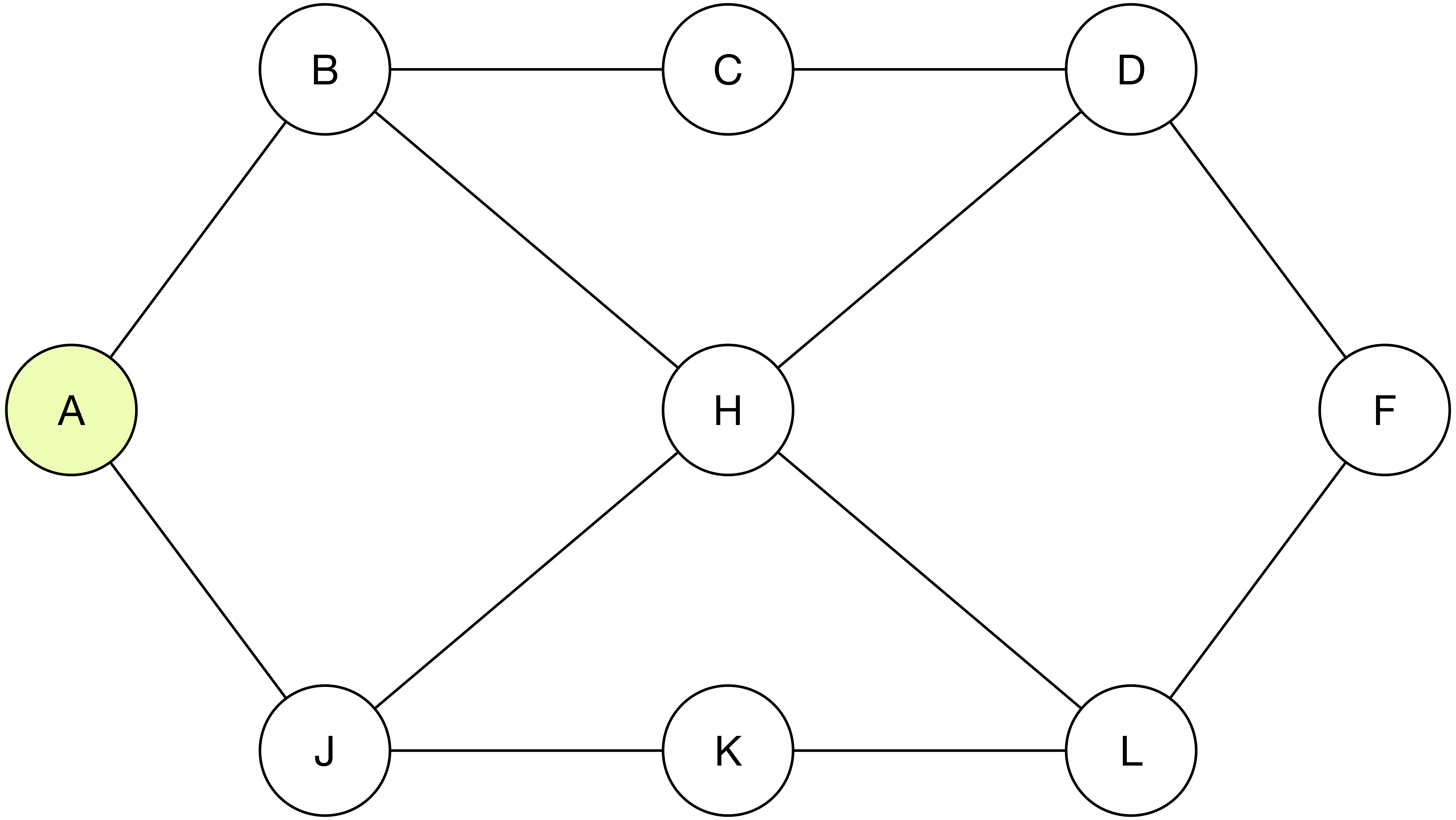
Shortest path



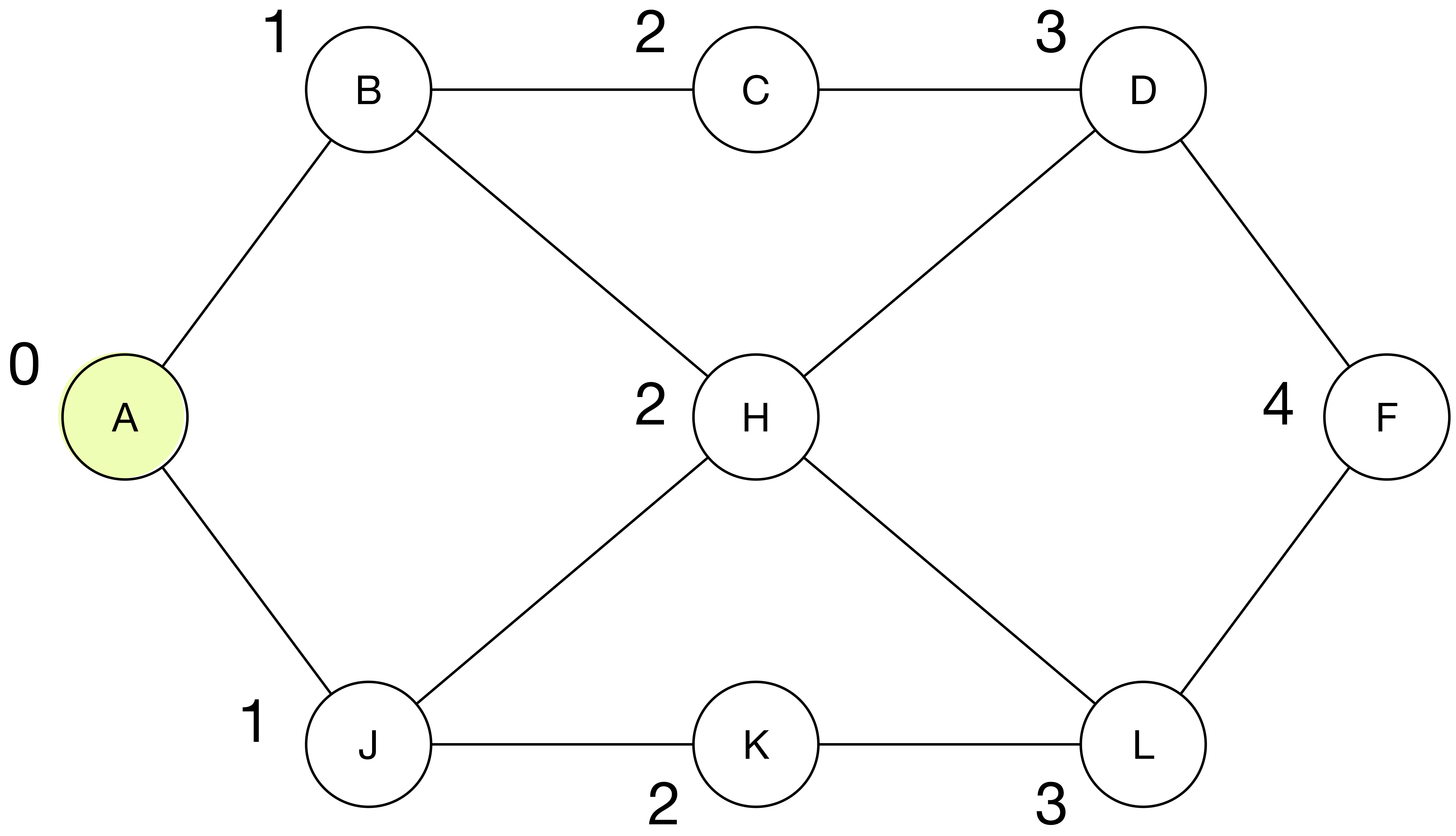
Shortest path

- Shortest paths from some node V_0 to all other nodes
- Shortest path from some node V_0 to one other node, V_1

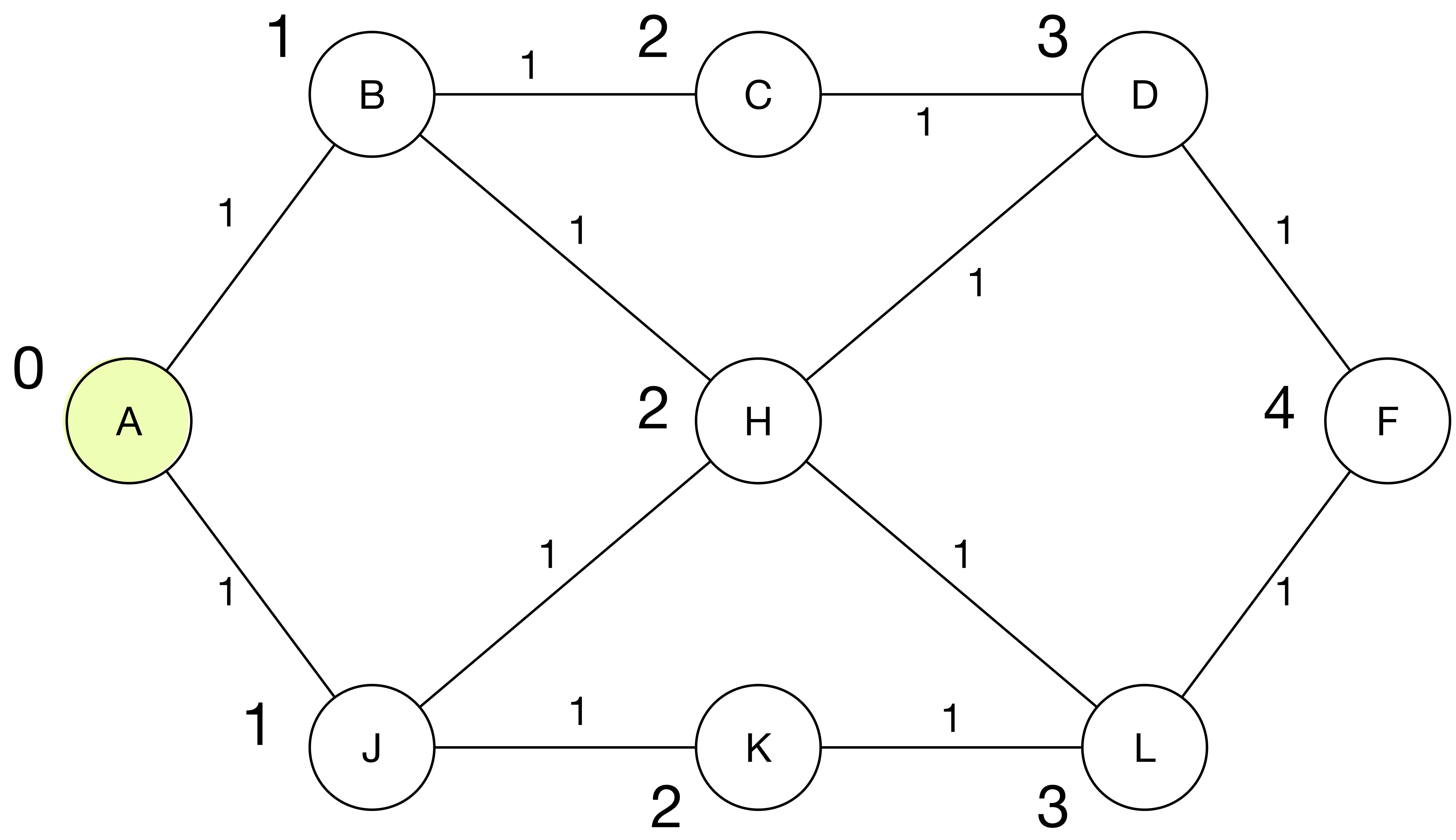
Shortest path, undirected, unweighted



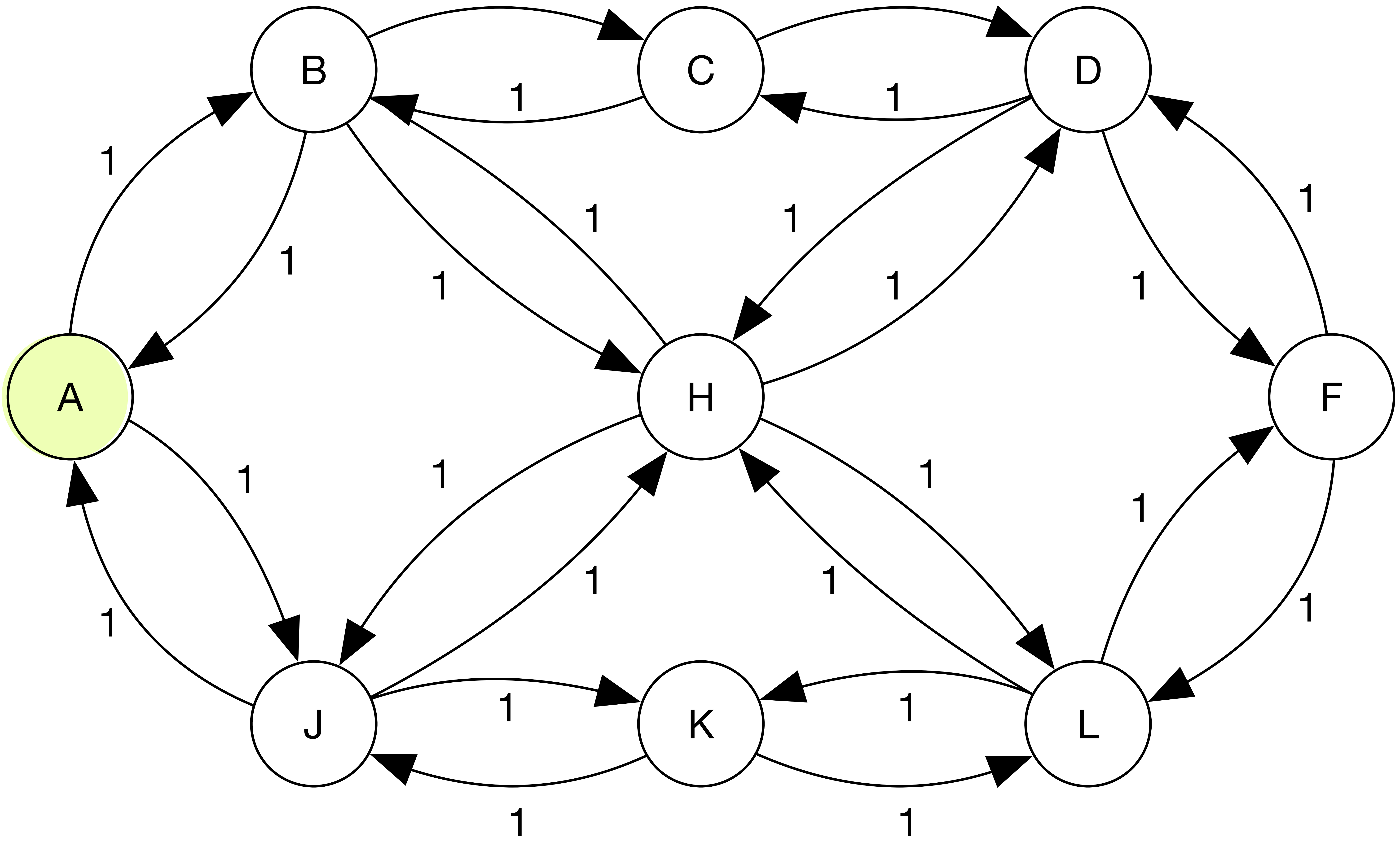
Shortest path, undirected, unweighted



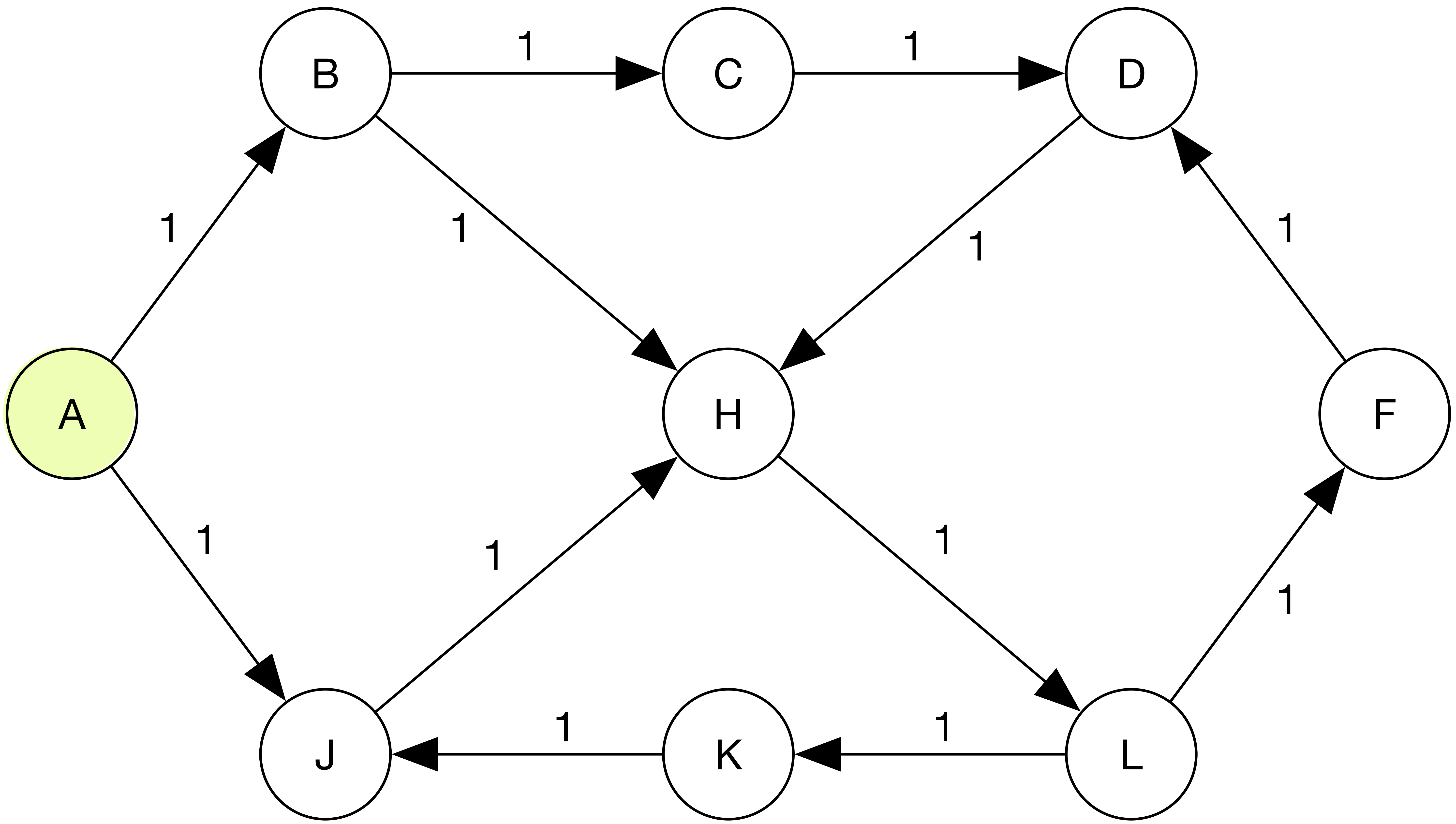
Shortest path, undirected



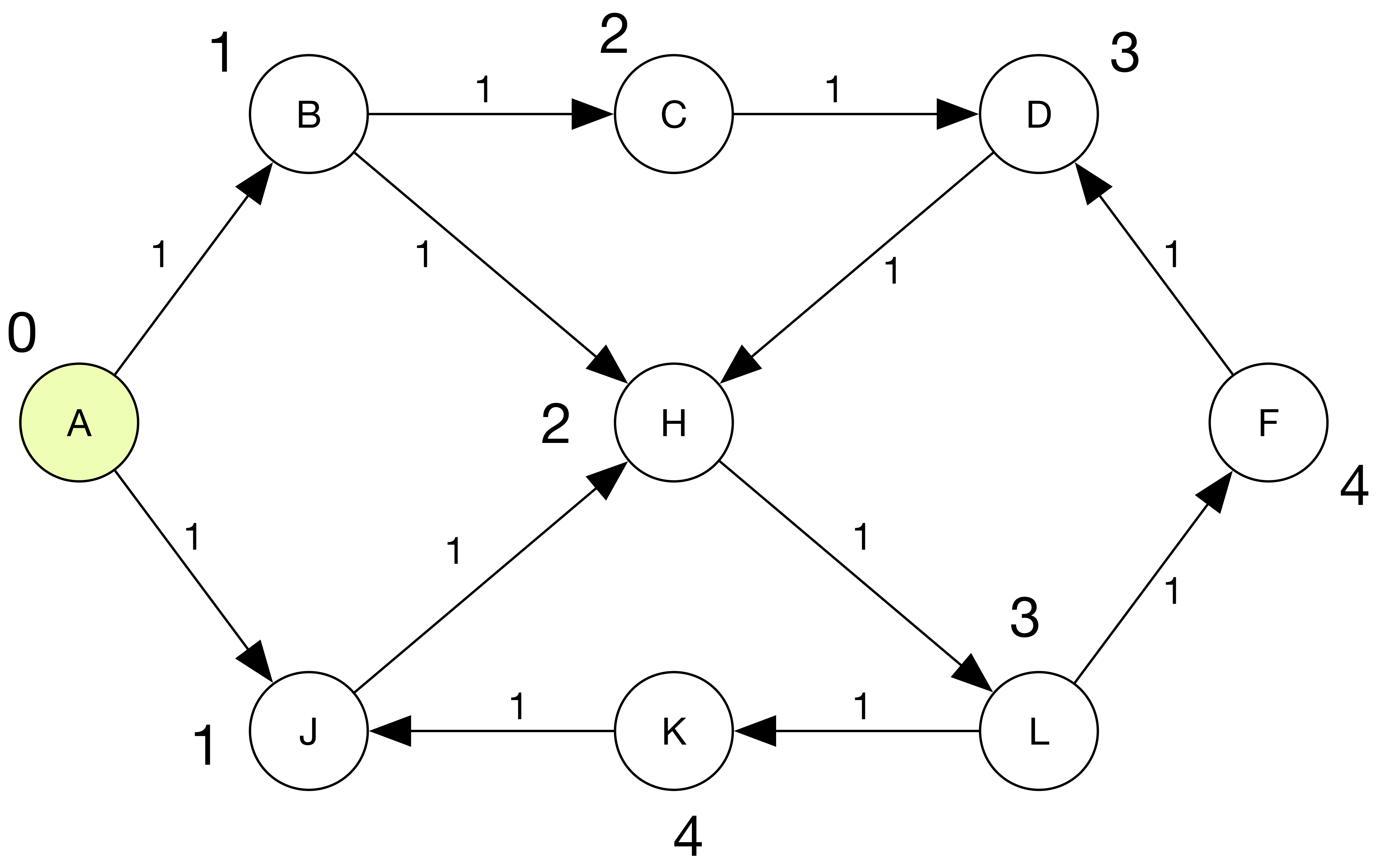
Shortest path



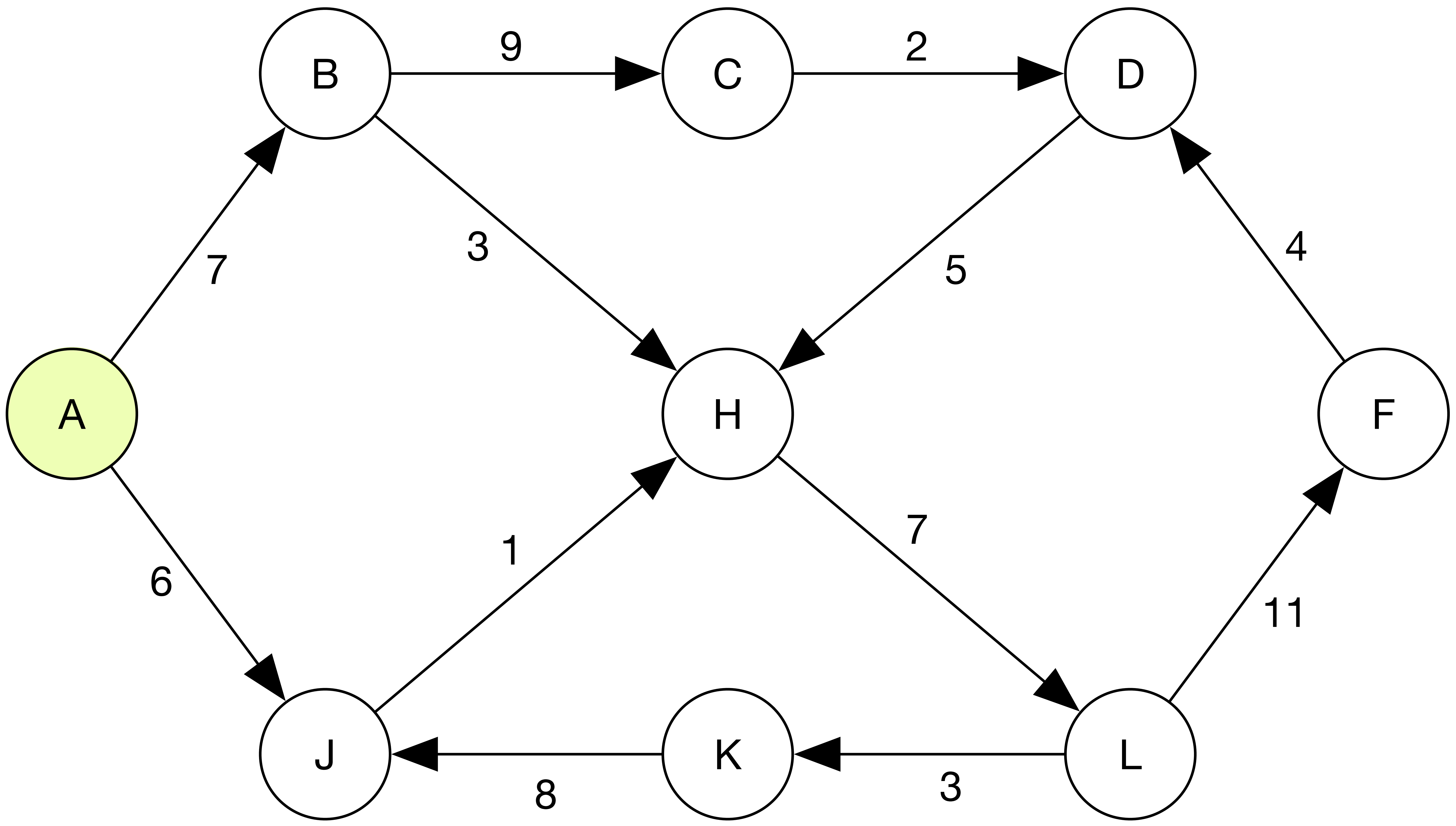
Shortest path



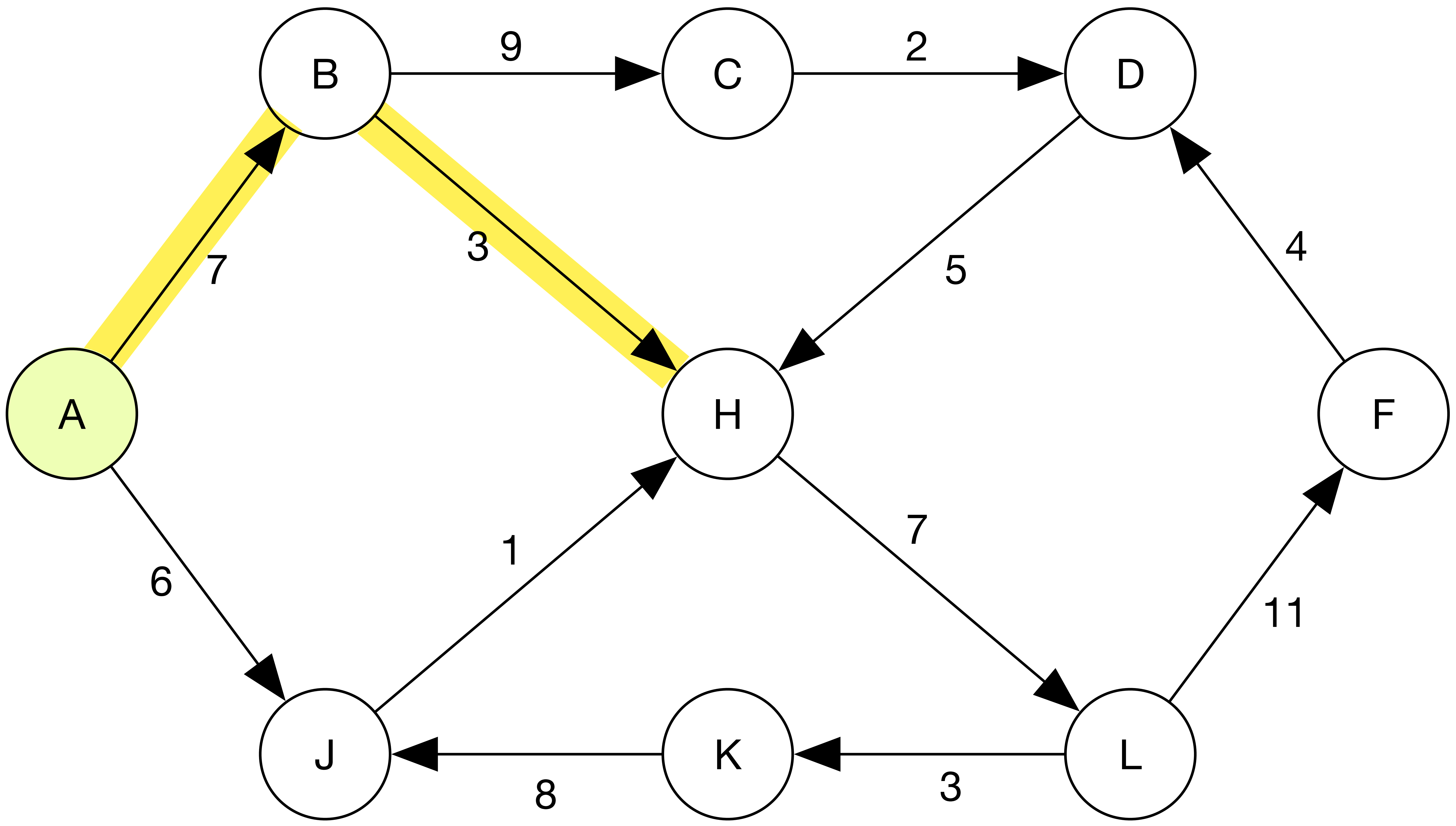
Shortest path



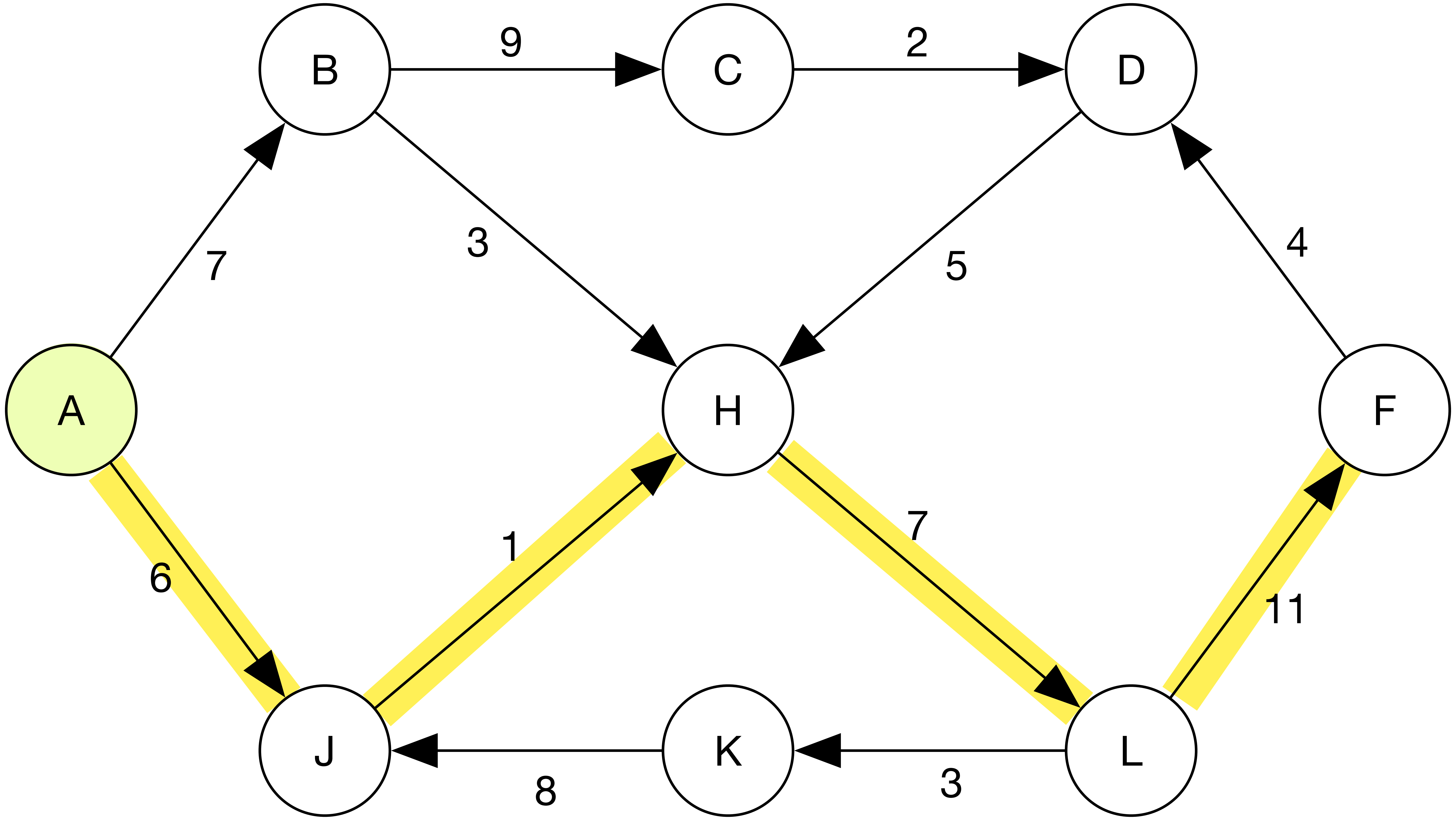
Shortest path



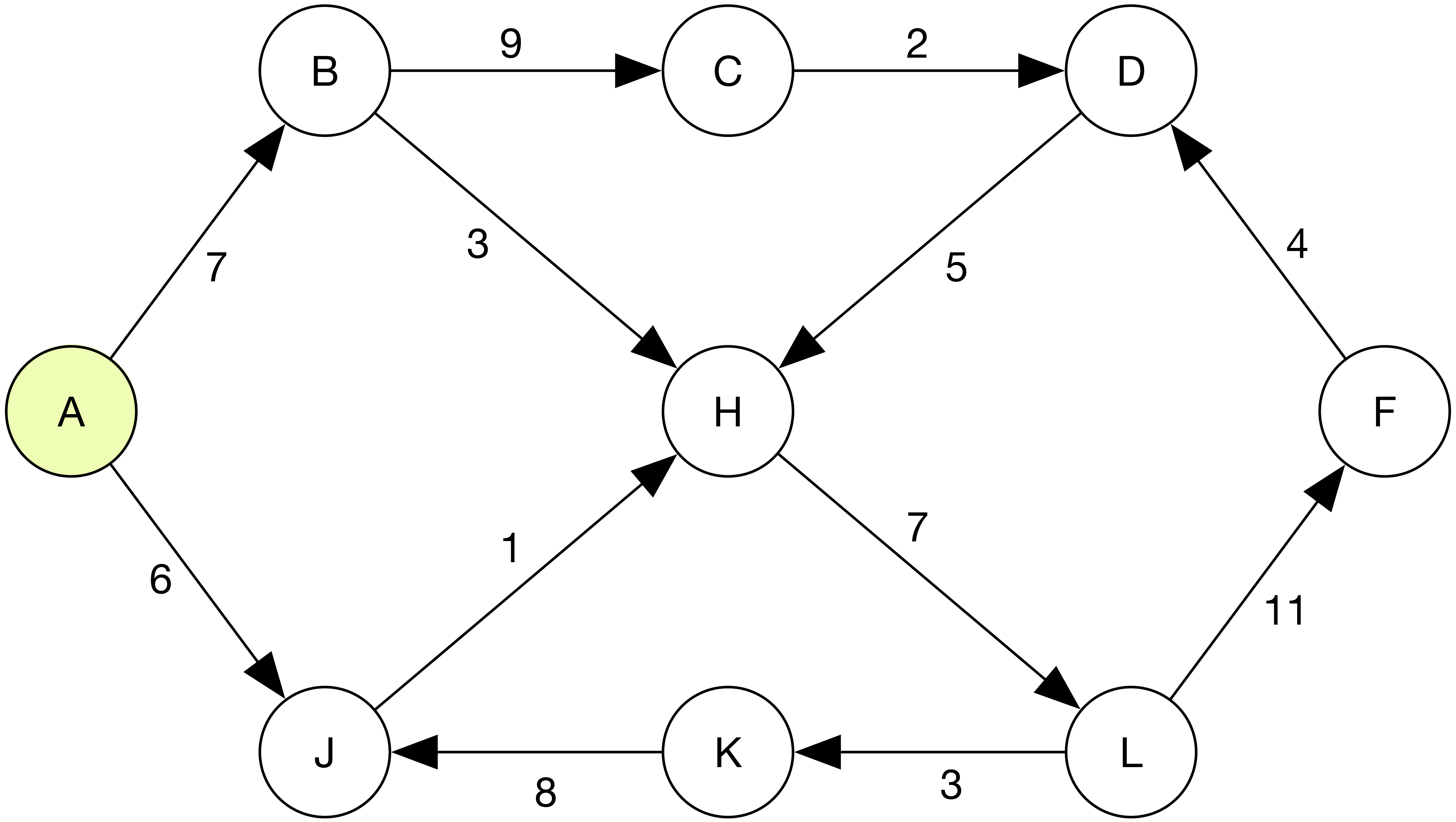
Shortest path



Shortest path



Shortest path



Shortest path: Dijkstra's algorithm

Dijkstra's algorithm

Given some graph, $G = (V, E)$, and some starting node $S \in V$, Dijkstra's algorithm will find the shortest paths (or paths with minimum weight) from S to all other nodes in V.

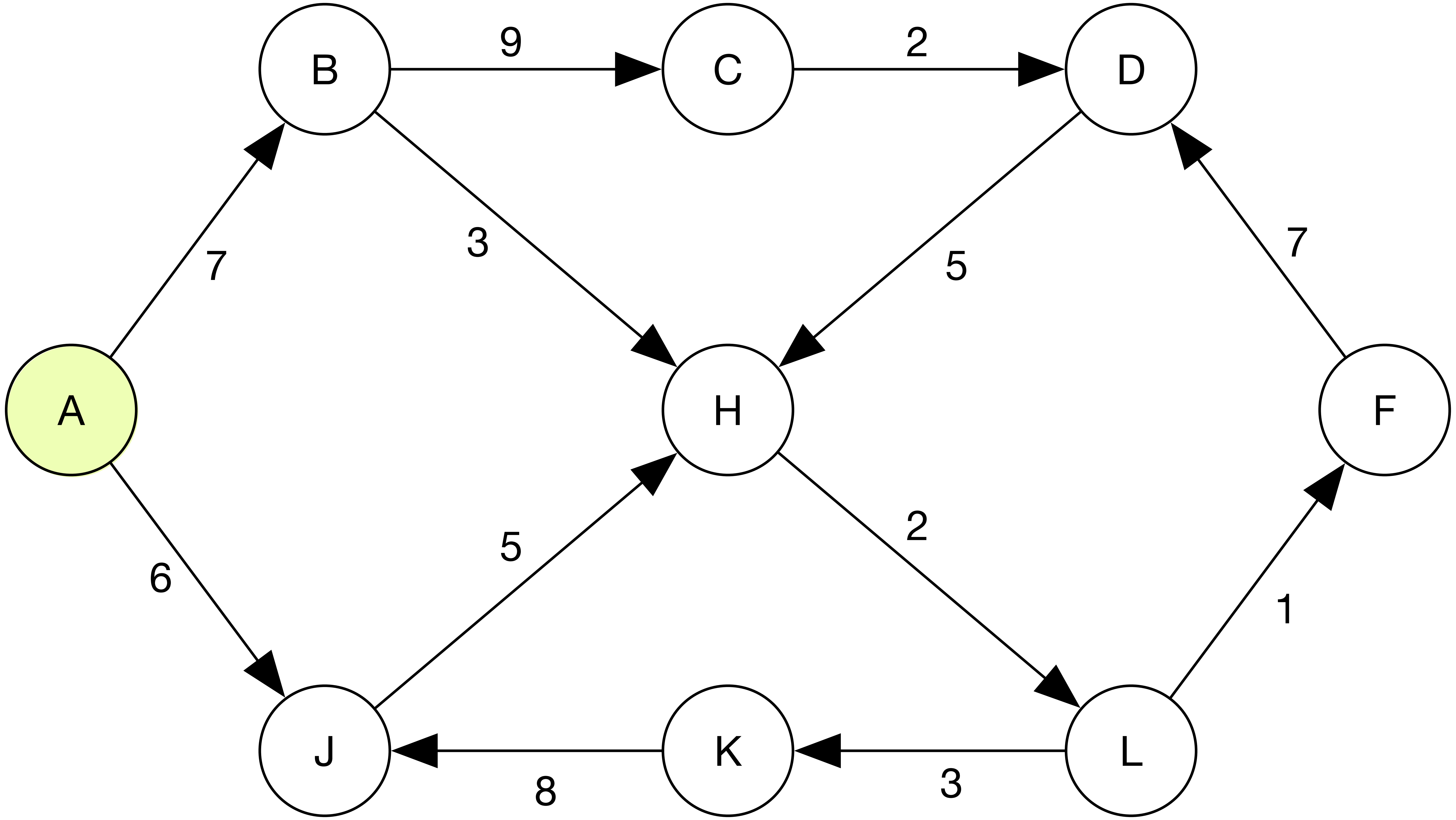
Note that G must not contain any negative weight edges.

Dijkstra

Mark all nodes as unvisited (add to set U)

Unvisited set

U = {A, B, C, D,
F, H, J, K, L}

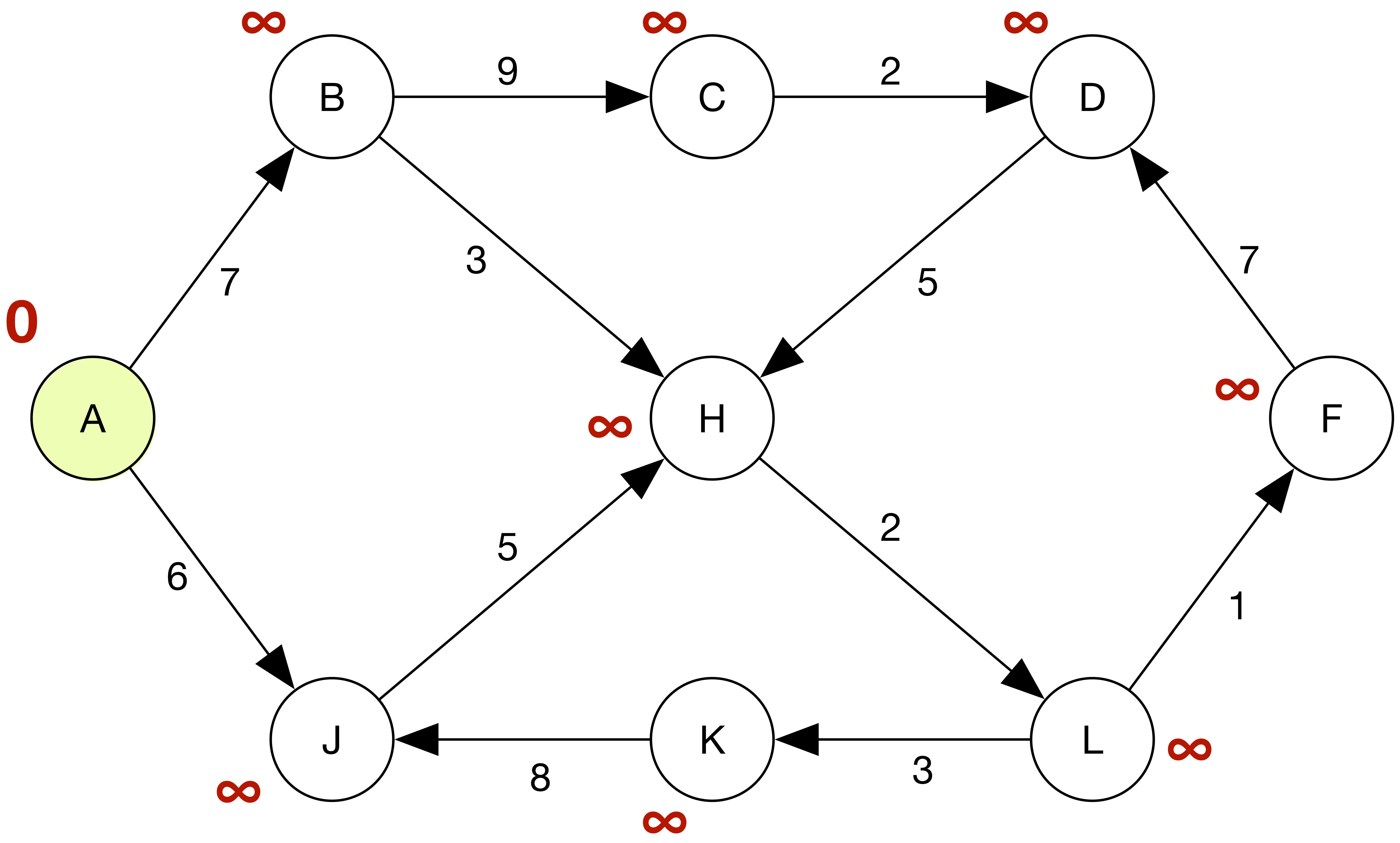


Dijkstra

Initialize distances

Unvisited set

U = {A, B, C, D,
F, H, J, K, L}

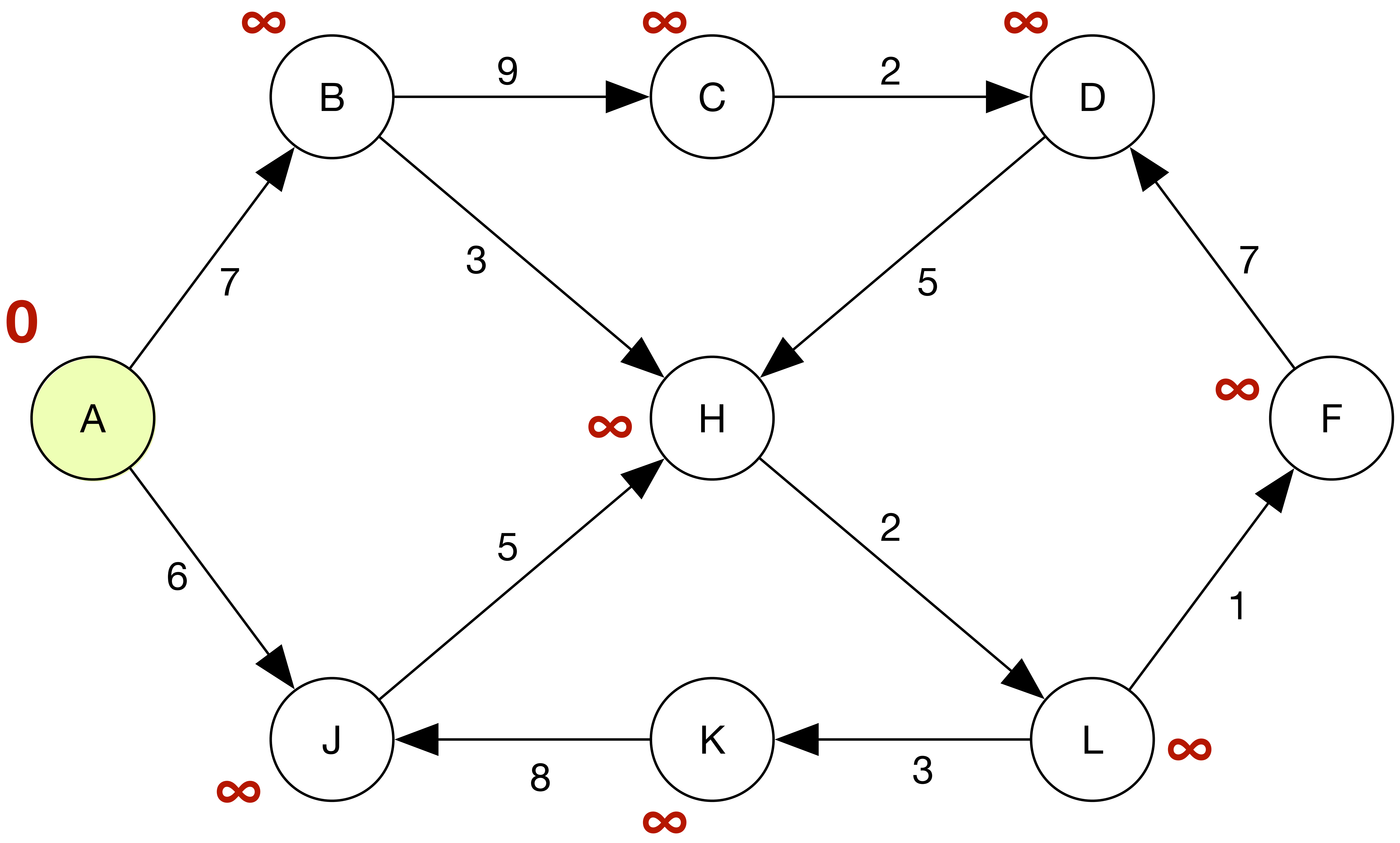


Dijkstra

Calculate distances to unvisited neighbors

Unvisited set

U = {A, B, C, D,
F, H, J, K, L}

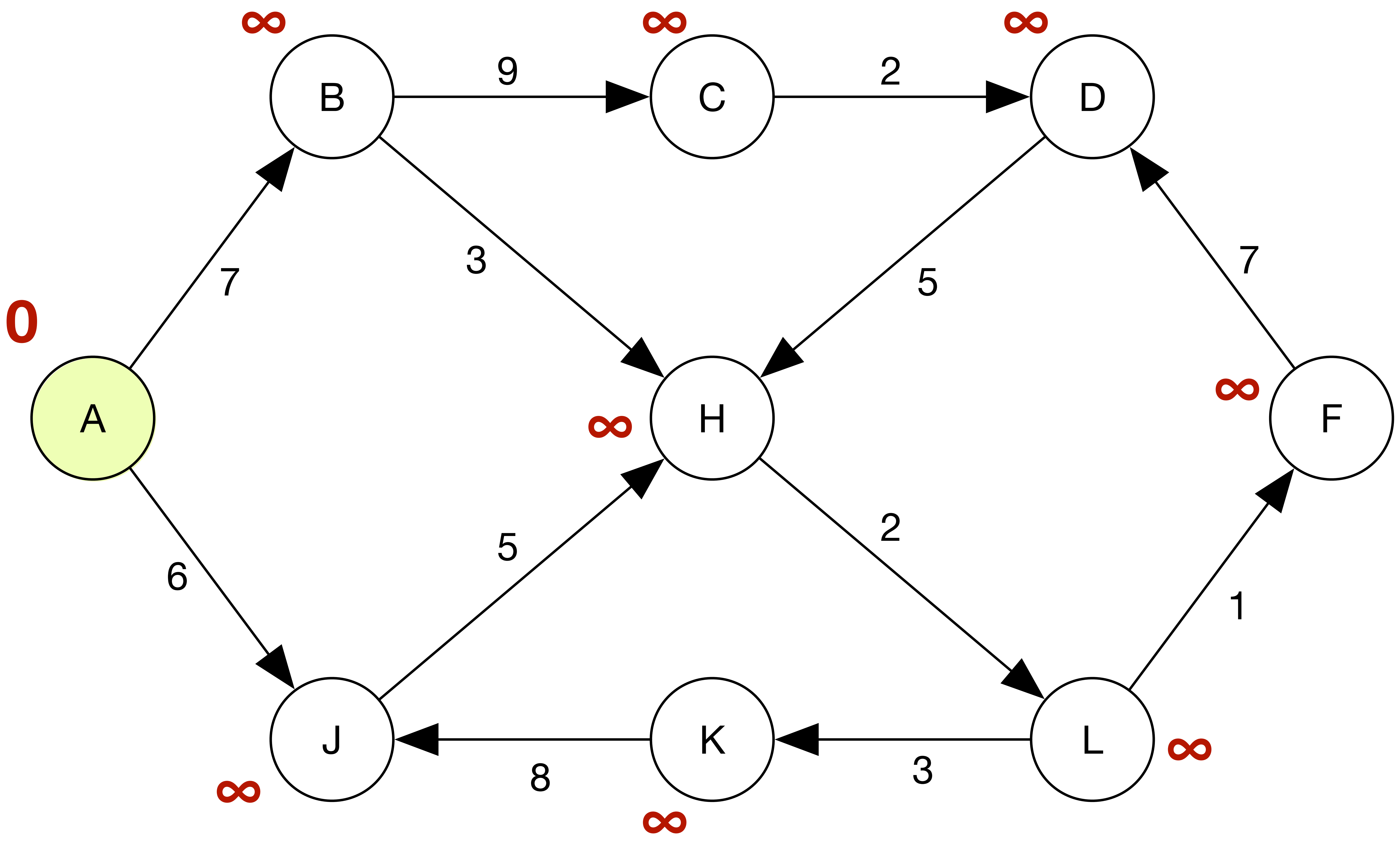


Dijkstra

Calculate distances to unvisited neighbors

Unvisited set

U = {A, B, C, D,
F, H, J, K, L}

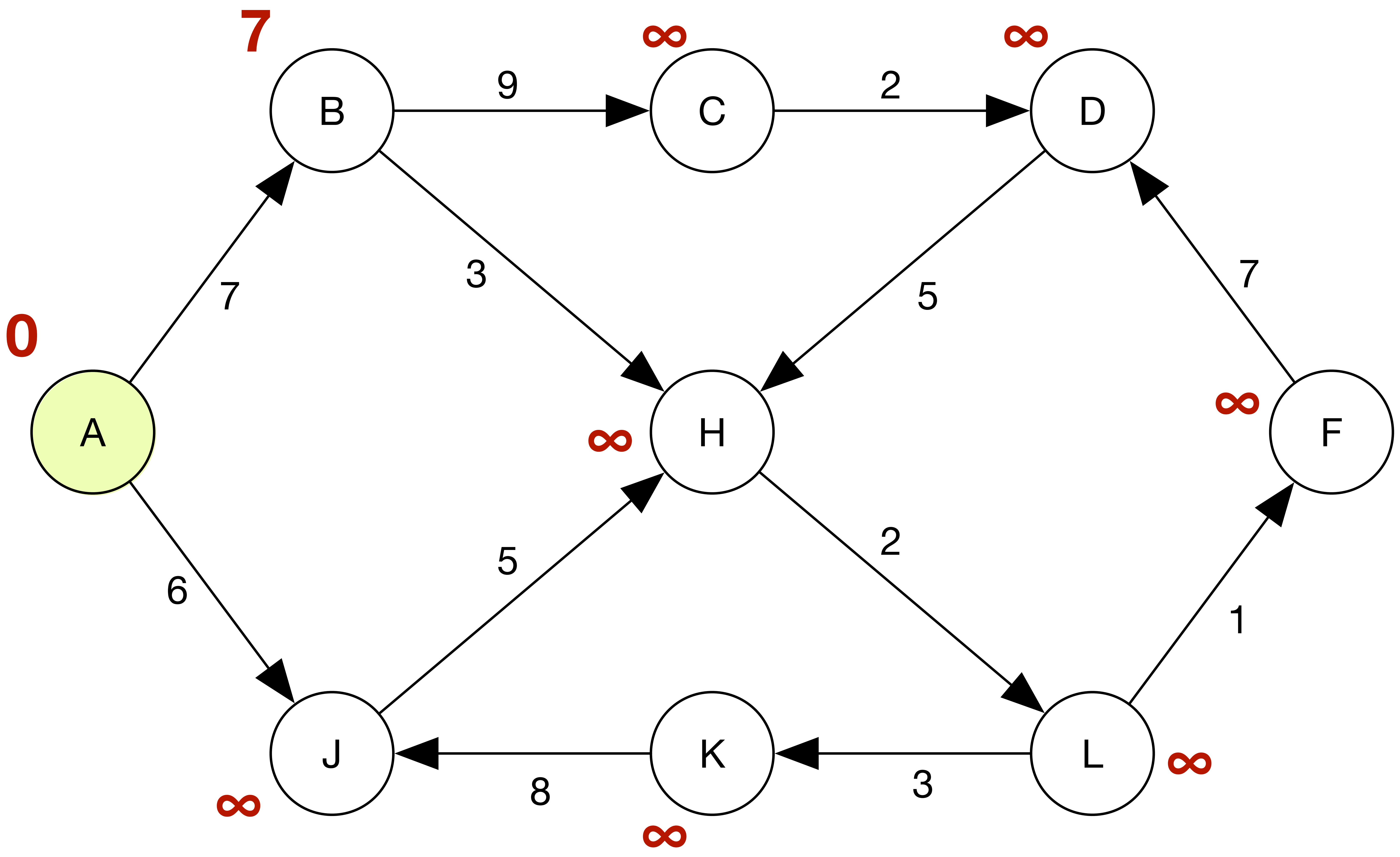


Dijkstra

Calculate distances to unvisited neighbors

Unvisited set

U = {A, B, C, D,
F, H, J, K, L}

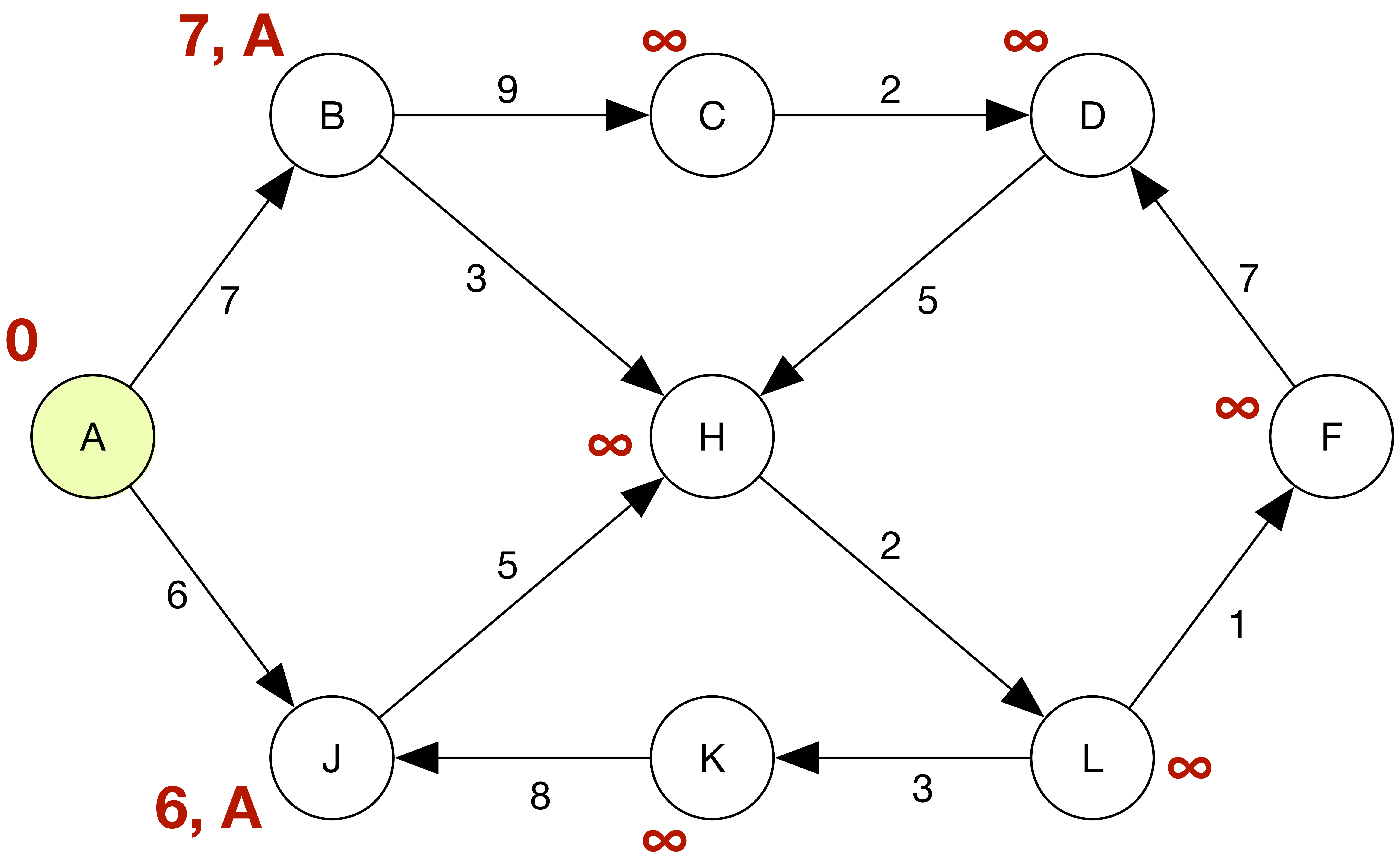


Dijkstra

Calculate distances to unvisited neighbors

Unvisited set

U = {A, B, C, D,
F, H, J, K, L}

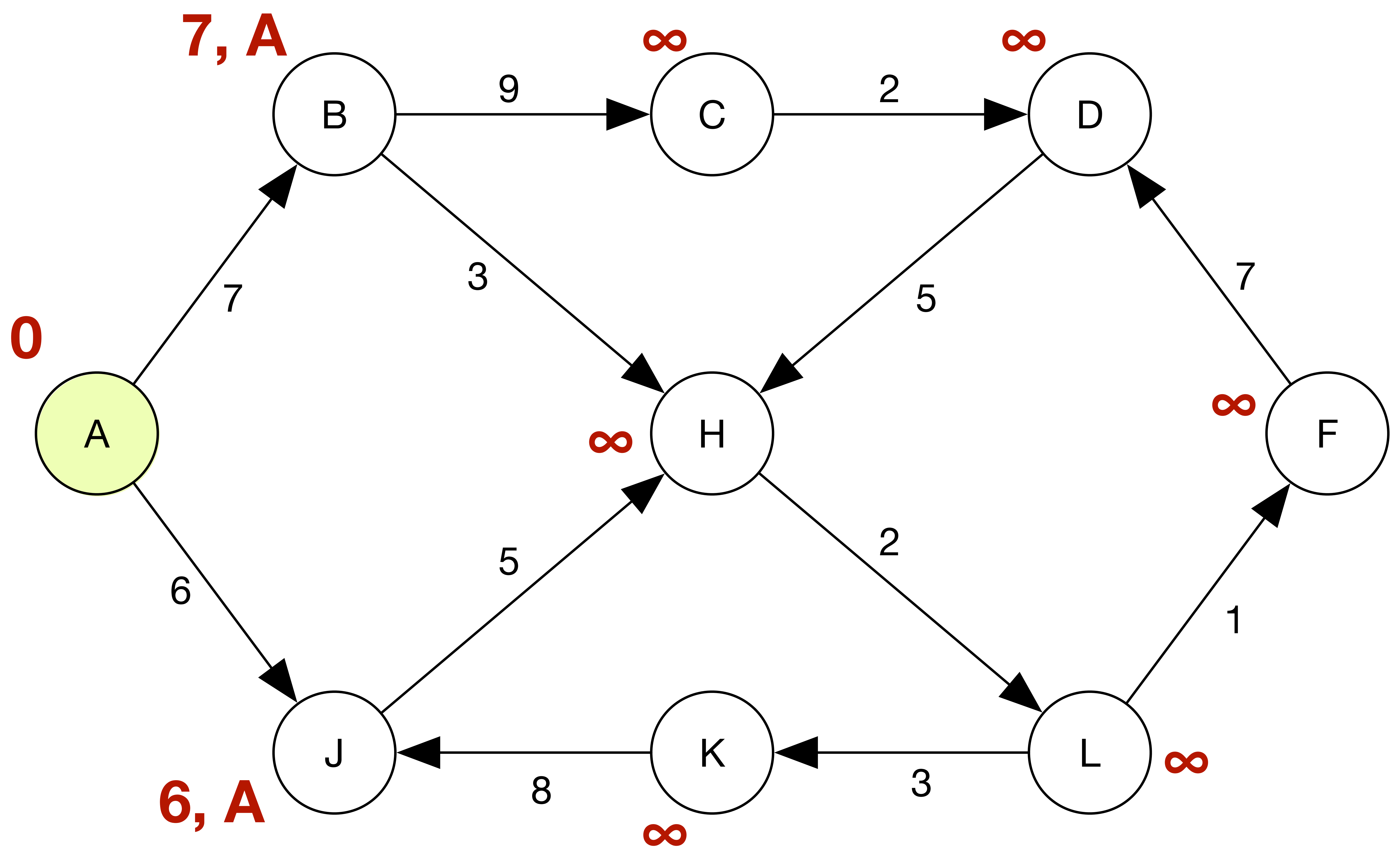


Dijkstra

Mark A as visited (remove from U)

Unvisited set

U = {B, C, D,
F, H, J, K, L}

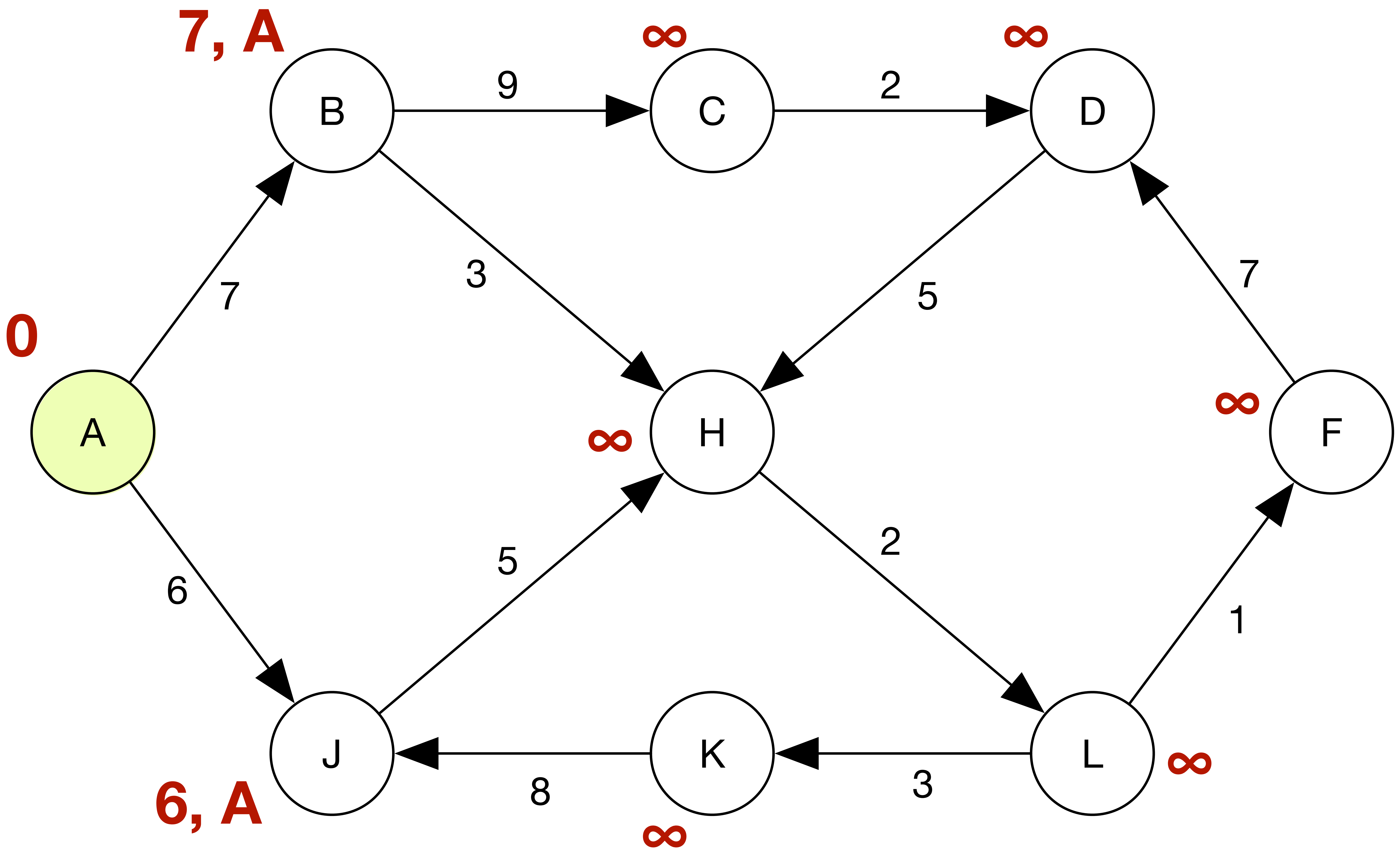


Dijkstra

Choose next node from which to explore

Unvisited set

U = {B, C, D,
F, H, J, K, L}

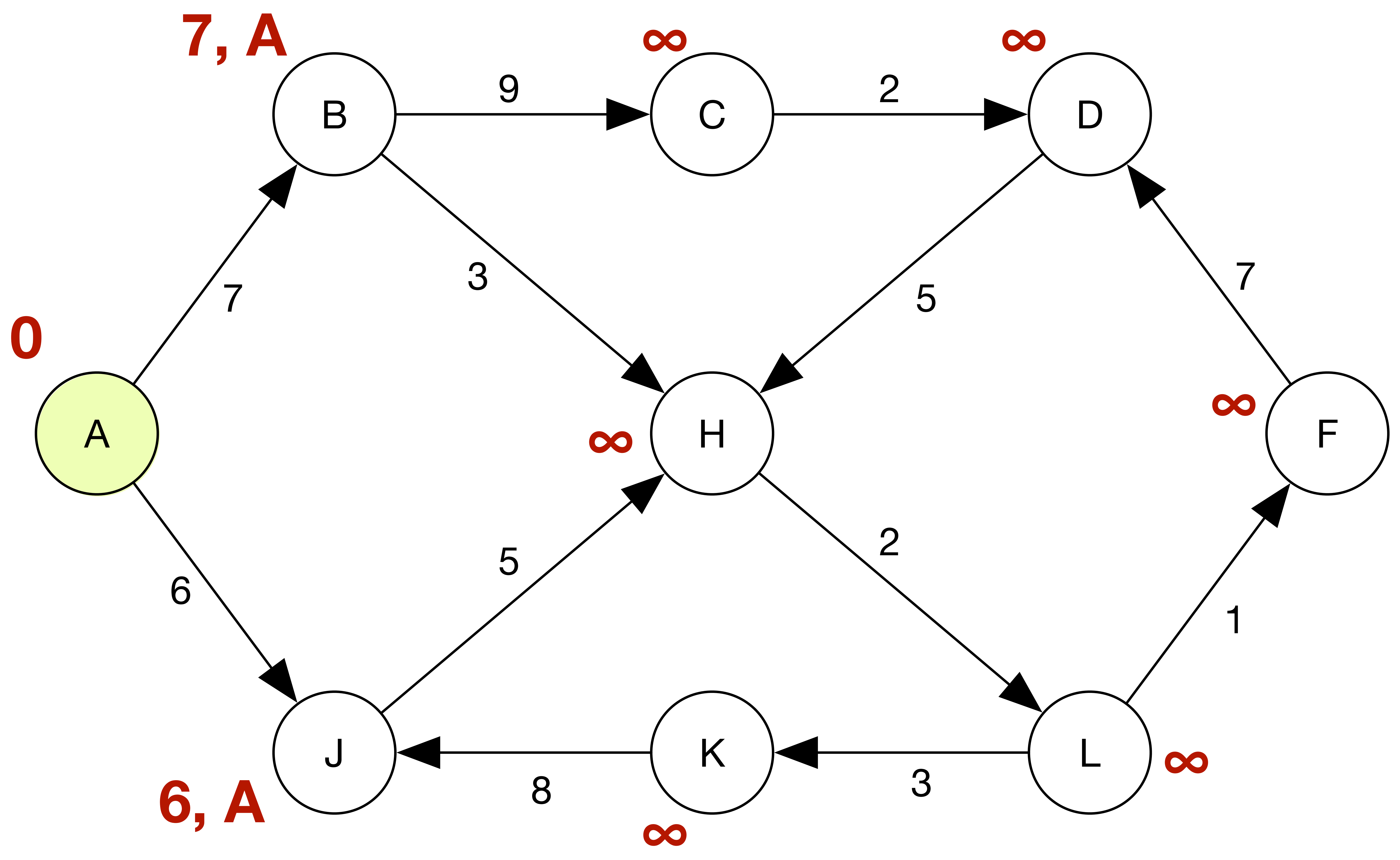


Dijkstra

Explore from J, and calculate distances

Unvisited set

U = {B, C, D,
F, H, J, K, L}

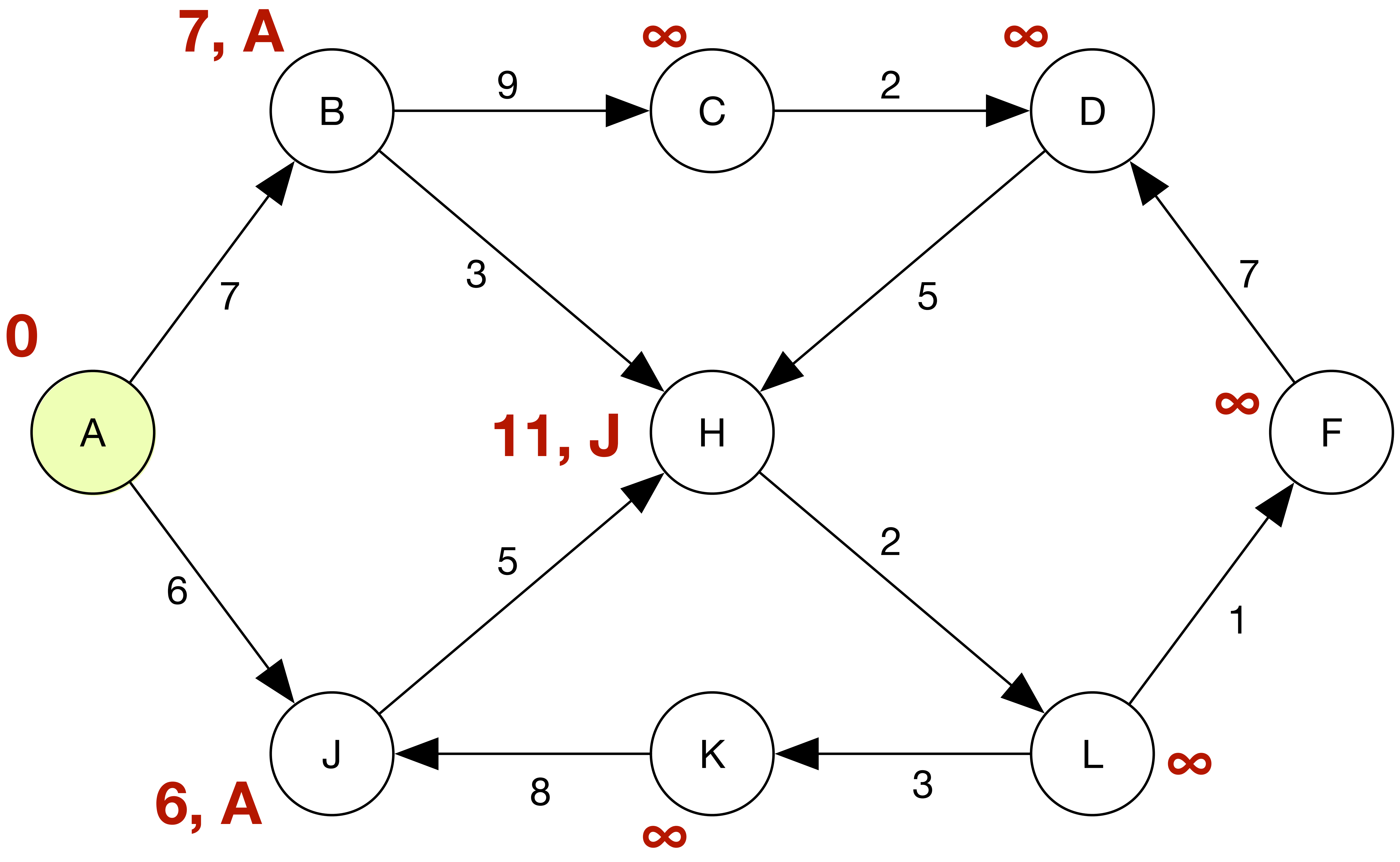


Dijkstra

Explore from J, and calculate distances

Unvisited set

U = {B, C, D,
F, H, J, K, L}

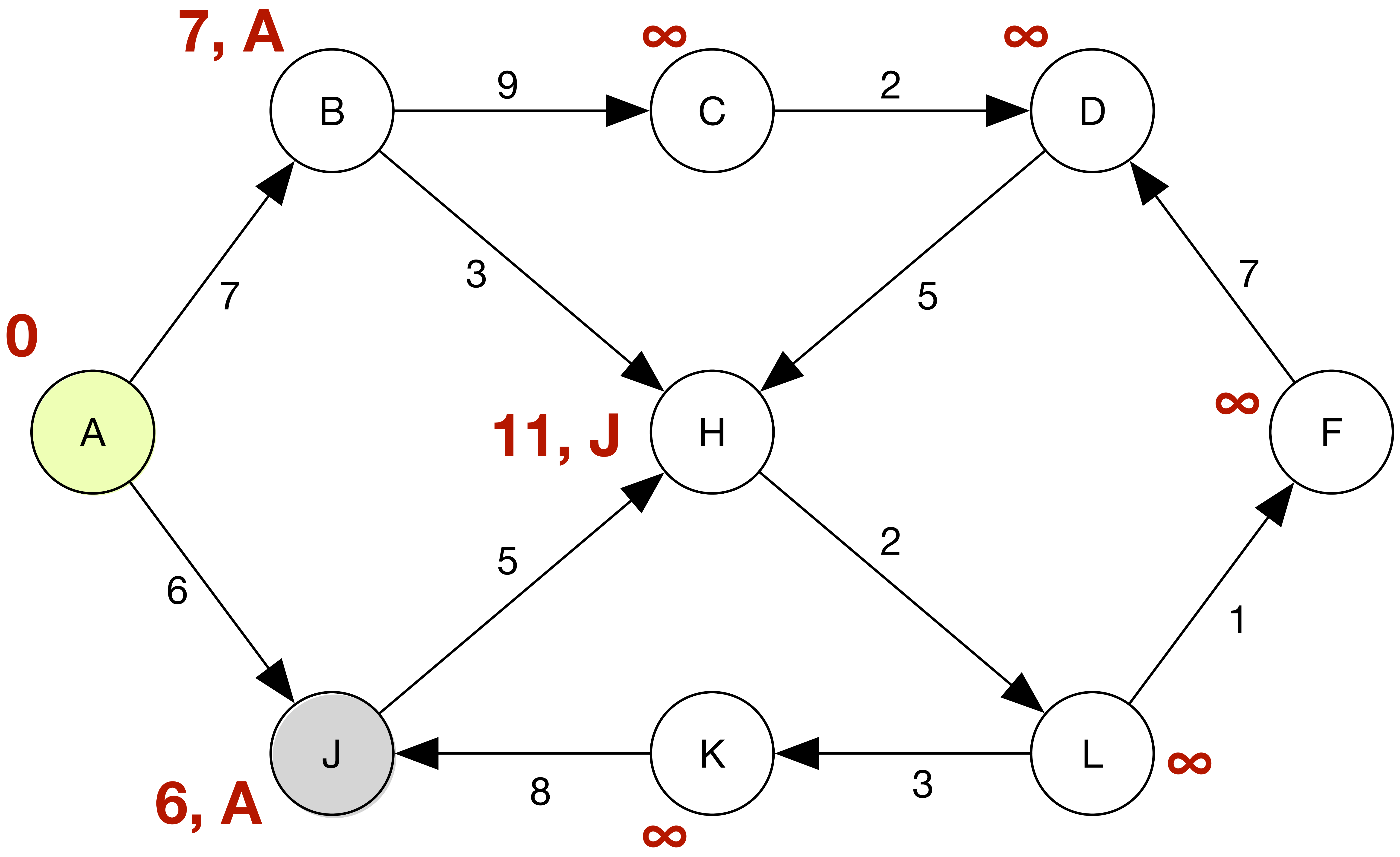


Dijkstra

Mark J as visited (remove from set U)

Unvisited set

U = {B, C, D,
F, H, K, L}

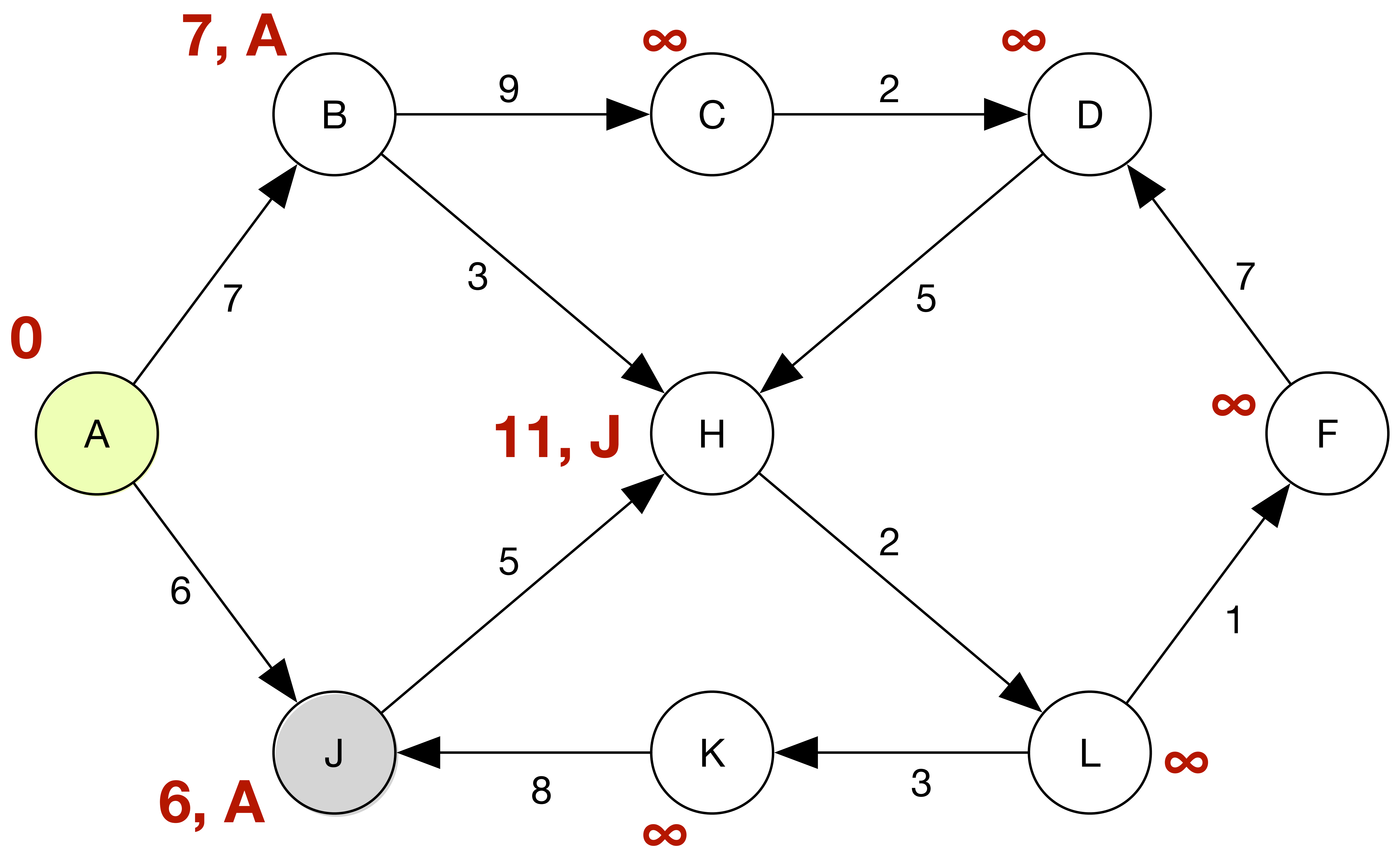


Dijkstra

Choose next node from which to explore

Unvisited set

U = {B, C, D,
F, H, K, L}

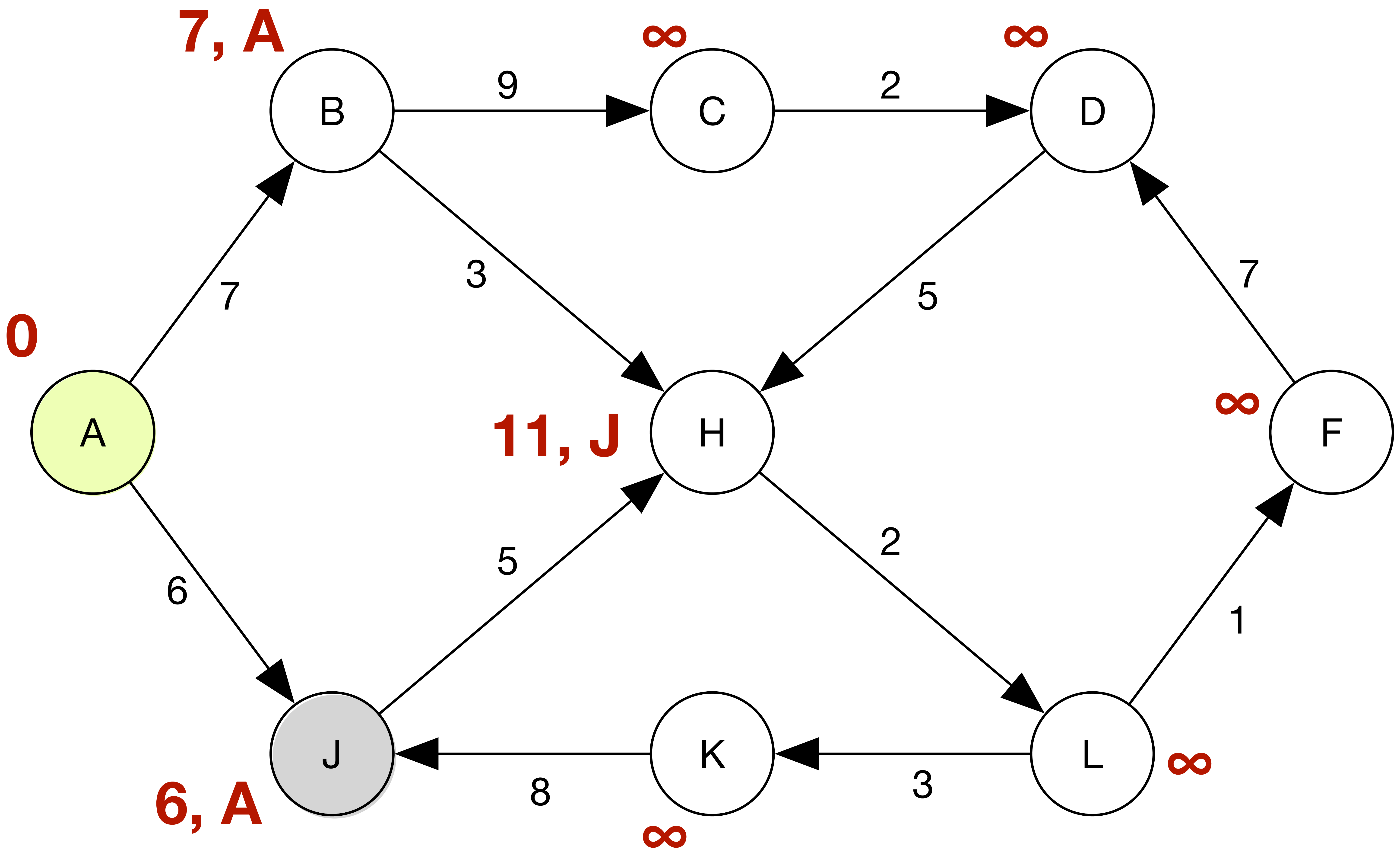


Dijkstra

Explore from B and calculate distances

Unvisited set

U = {B, C, D,
F, H, K, L}

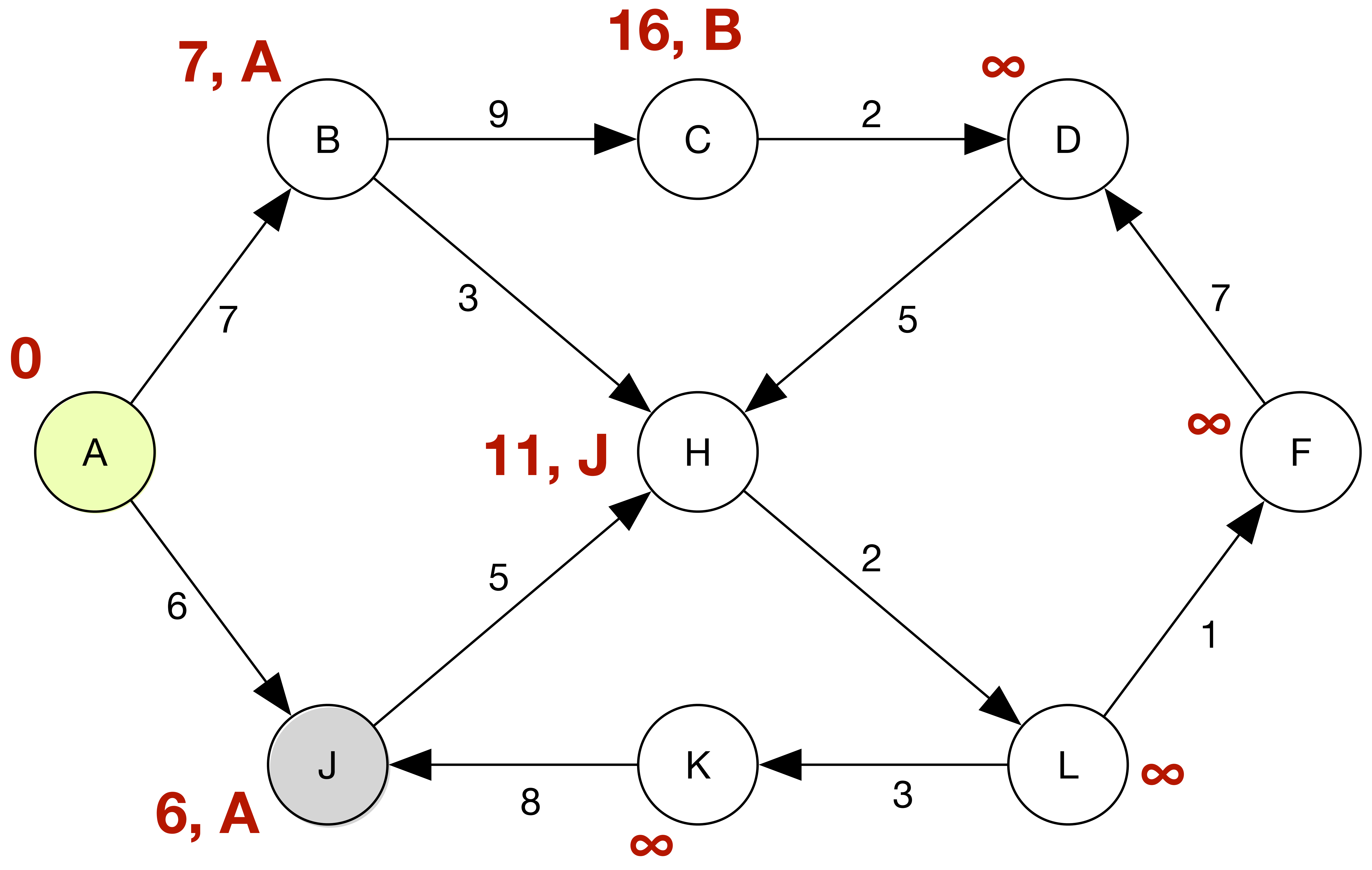


Dijkstra

Explore from B and calculate distances

Unvisited set

U = {B, C, D,
F, H, K, L}

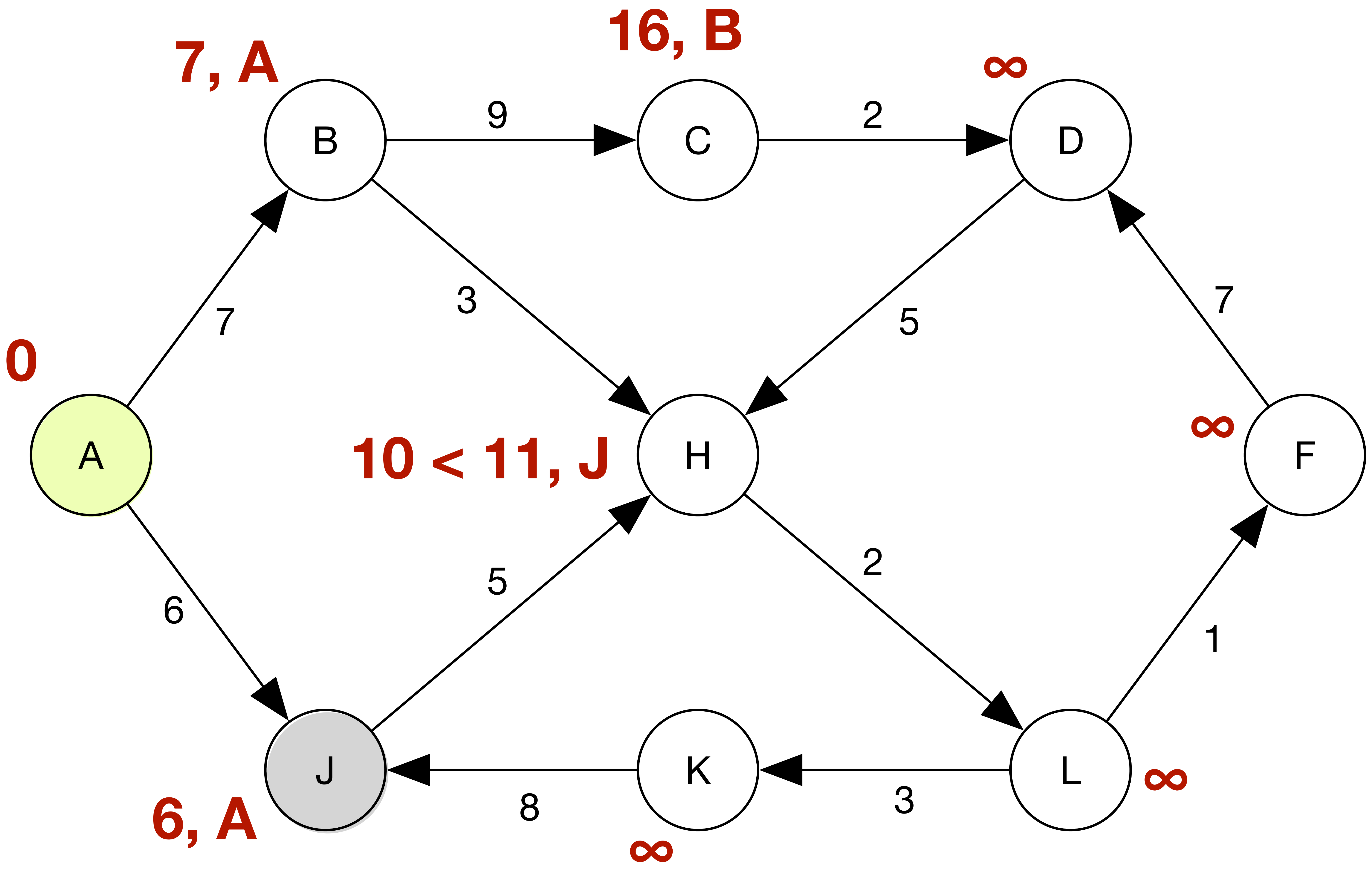


Dijkstra

Explore from B and calculate distances

Unvisited set

U = {B, C, D,
F, H, K, L}

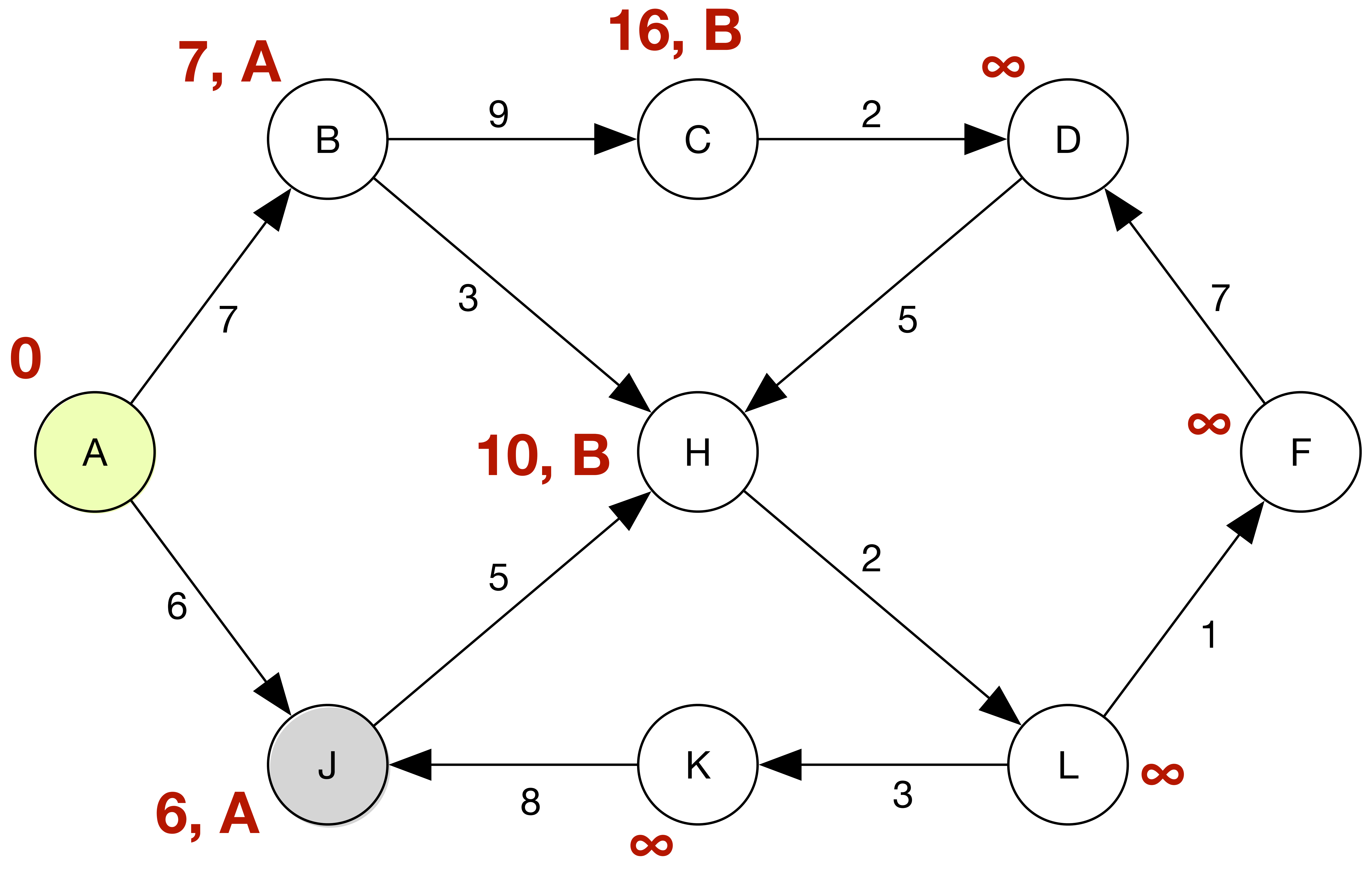


Dijkstra

Explore from B and calculate distances

Unvisited set

U = {B, C, D,
F, H, K, L}

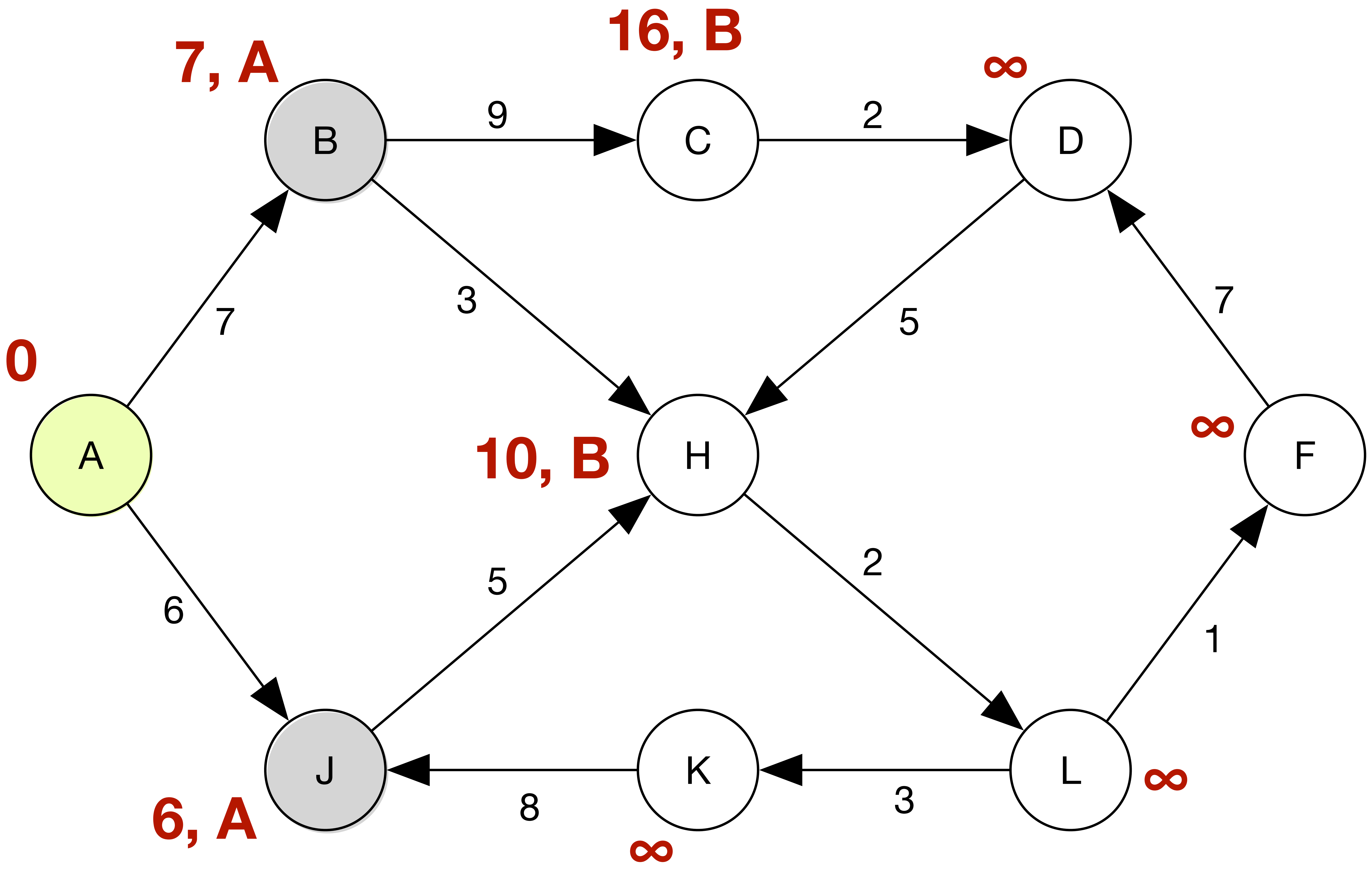


Dijkstra

Mark B as visited (remove from set U)

Unvisited set

U = {C, D,
F, H, K, L}

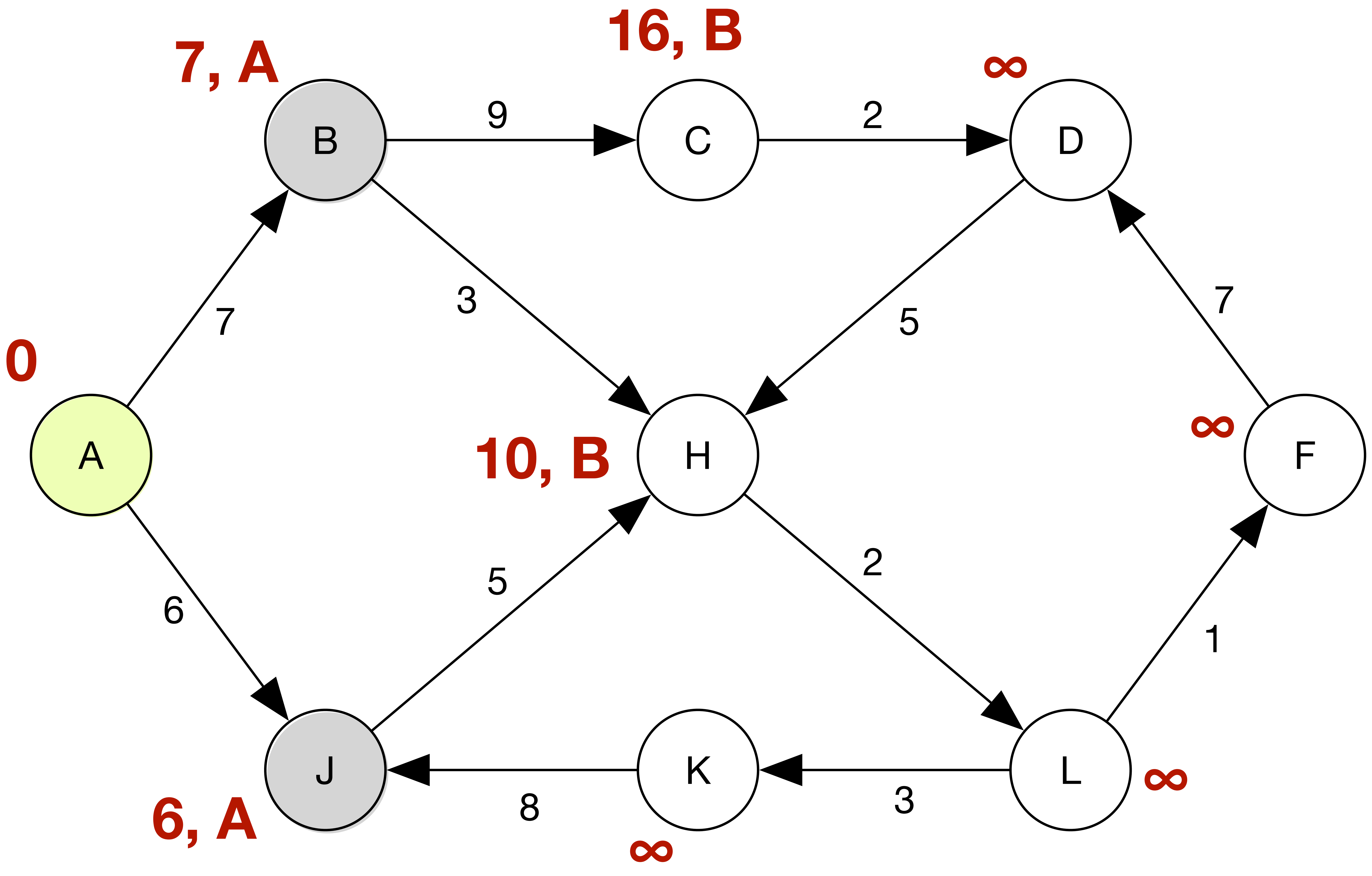


Dijkstra

Choose the next node from which to explore

Unvisited set

U = {C, D,
F, H, K, L}

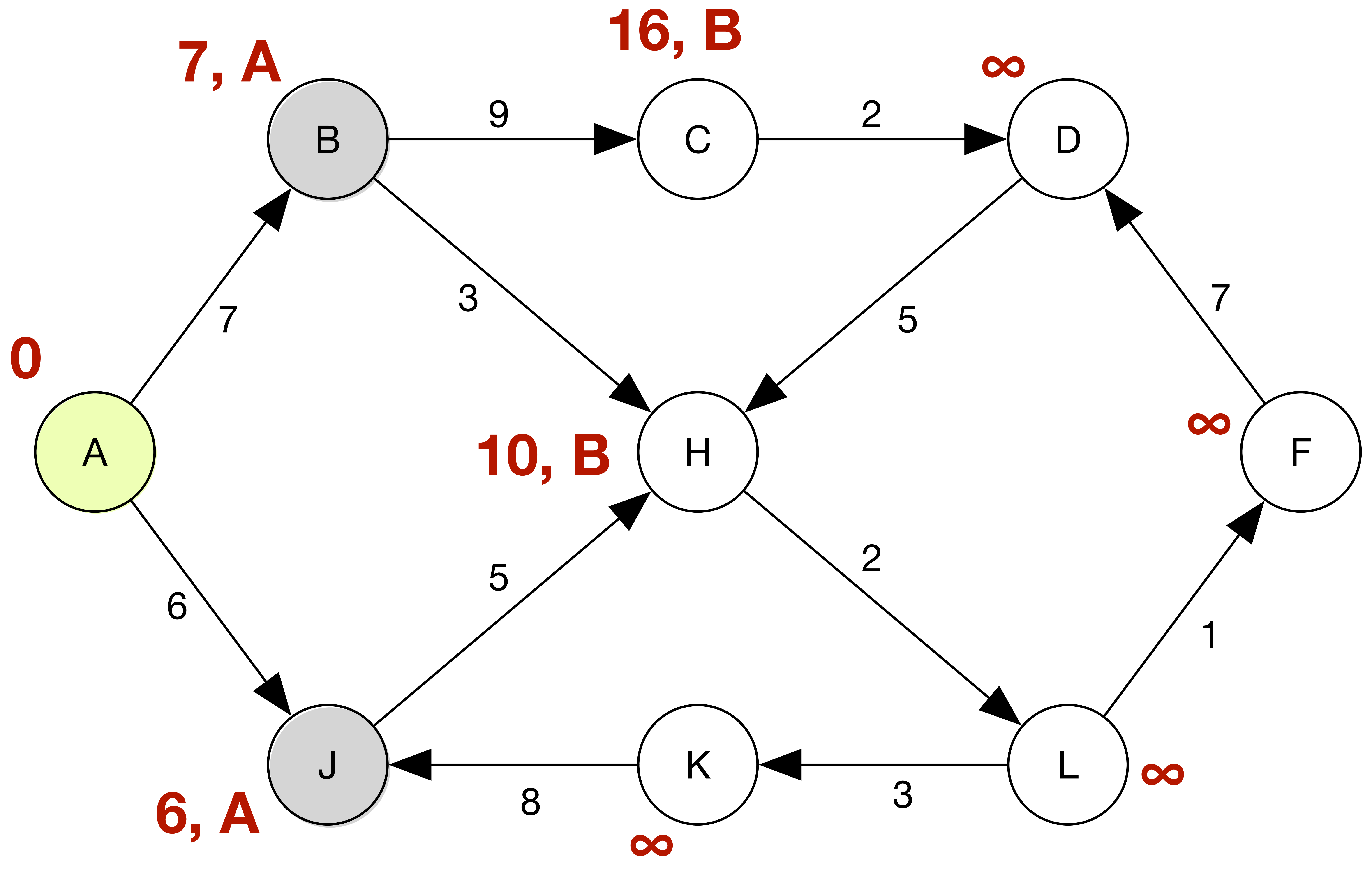


Dijkstra

Explore from H and calculate distances

Unvisited set

U = {C, D,
F, H, K, L}

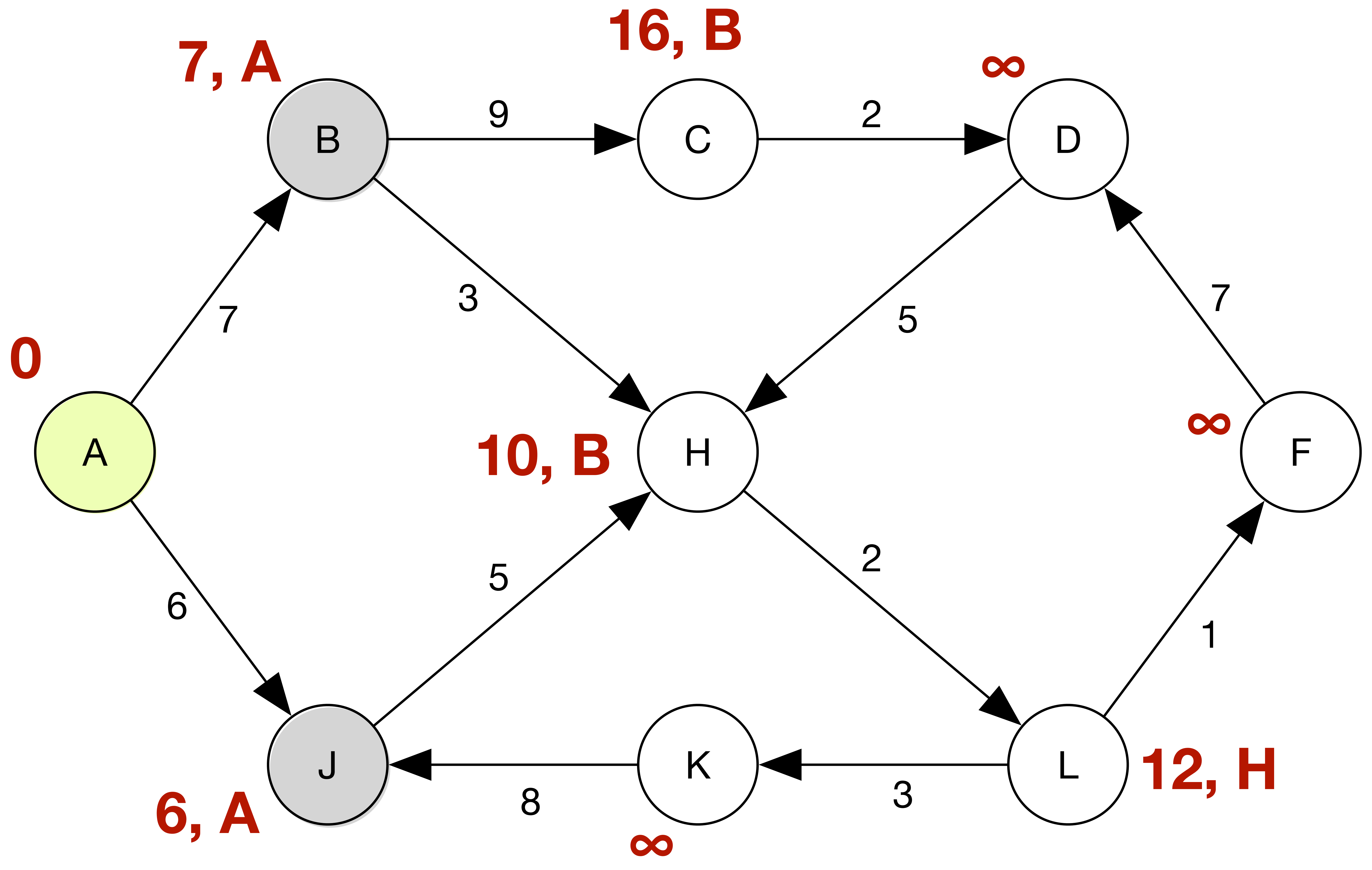


Dijkstra

Explore from H and calculate distances

Unvisited set

U = {C, D,
F, H, K, L}

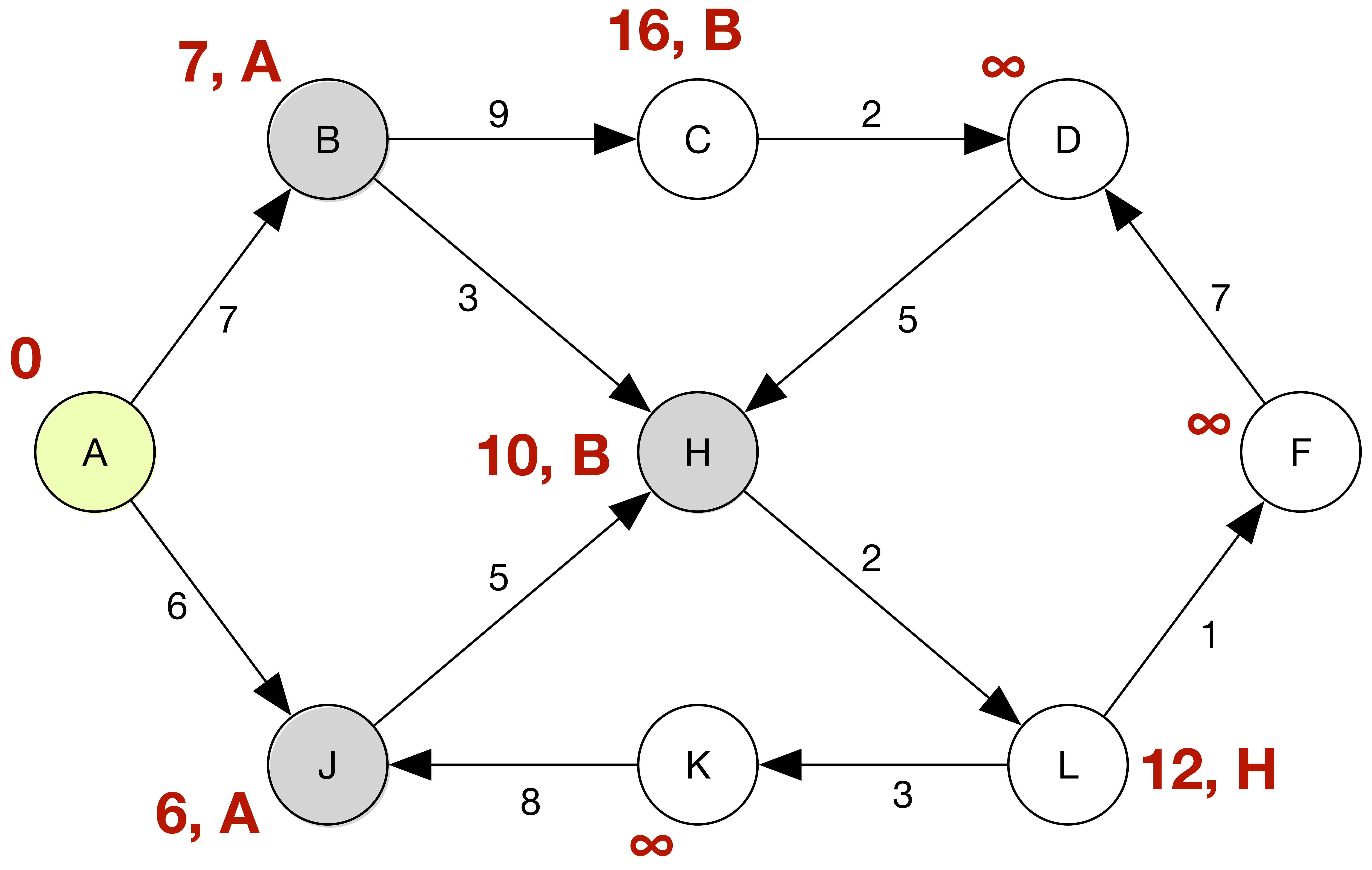


Dijkstra

Mark H as visited (remove from set U)

Unvisited set

U = {C, D,
F, K, L}

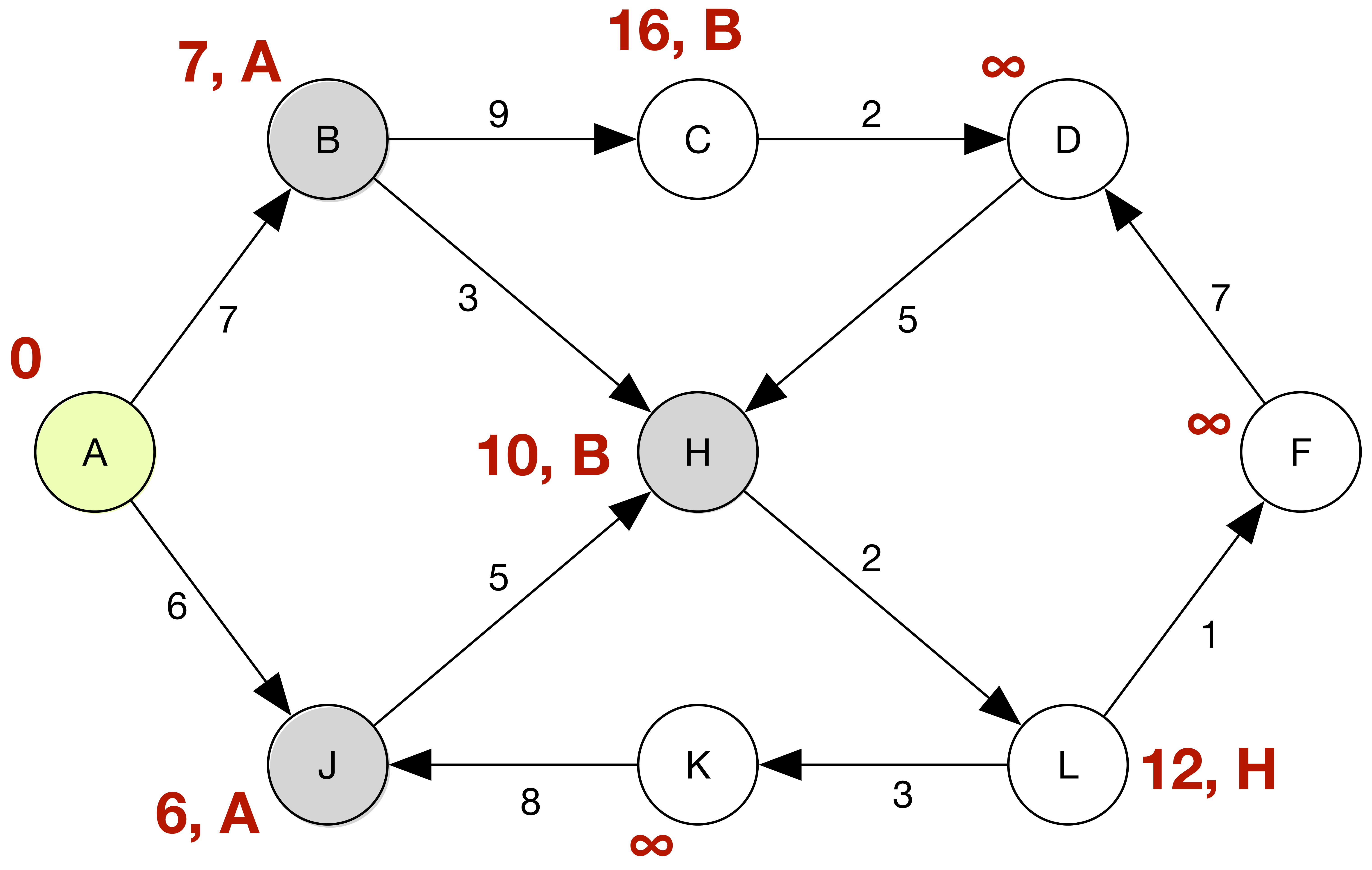


Dijkstra

Choose next node from which to explore

Unvisited set

U = {C, D,
F, K, L}

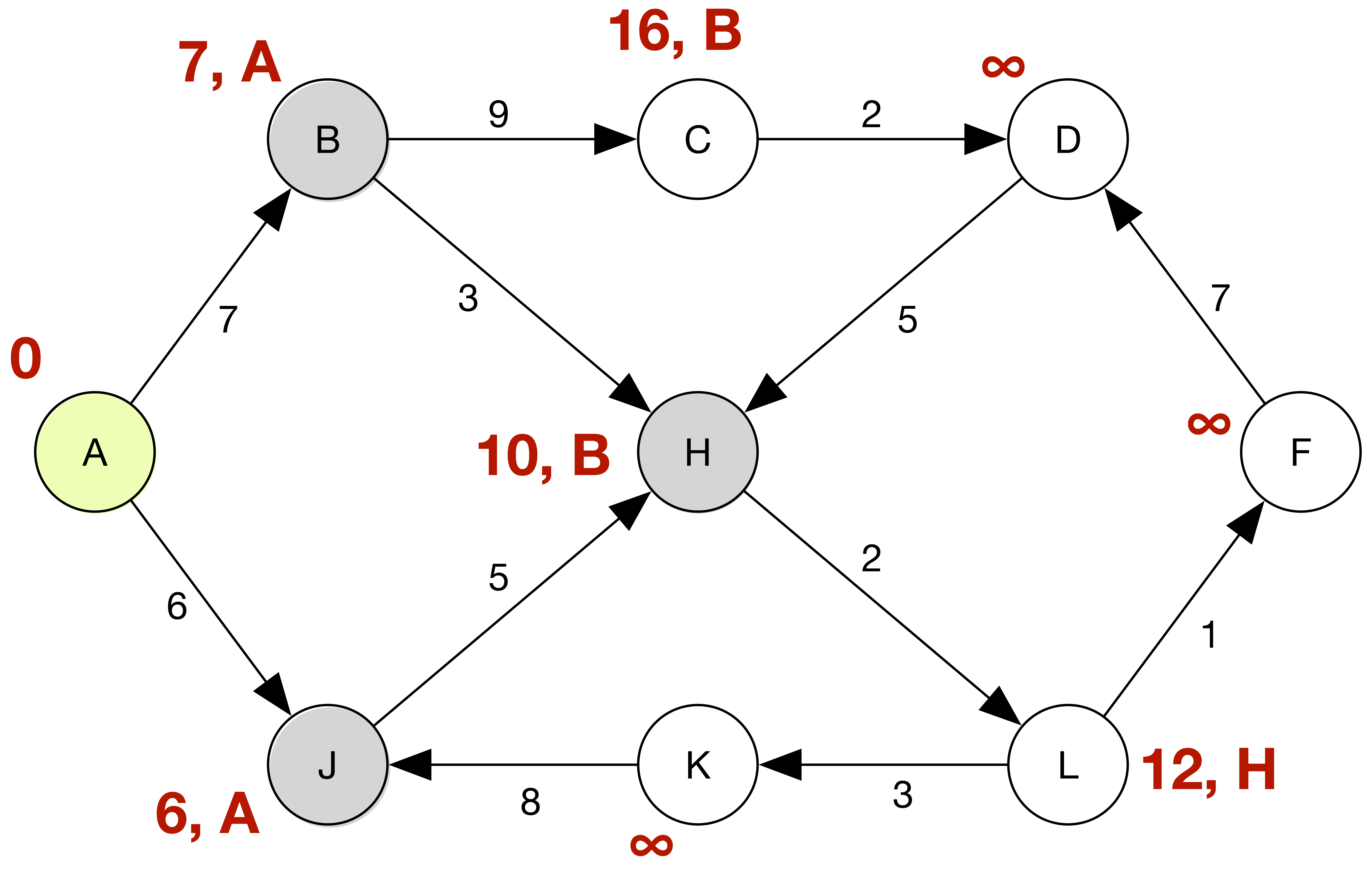


Dijkstra

Unvisited set

U = {C, D, F, K, L}

Explore from L and calculate distances

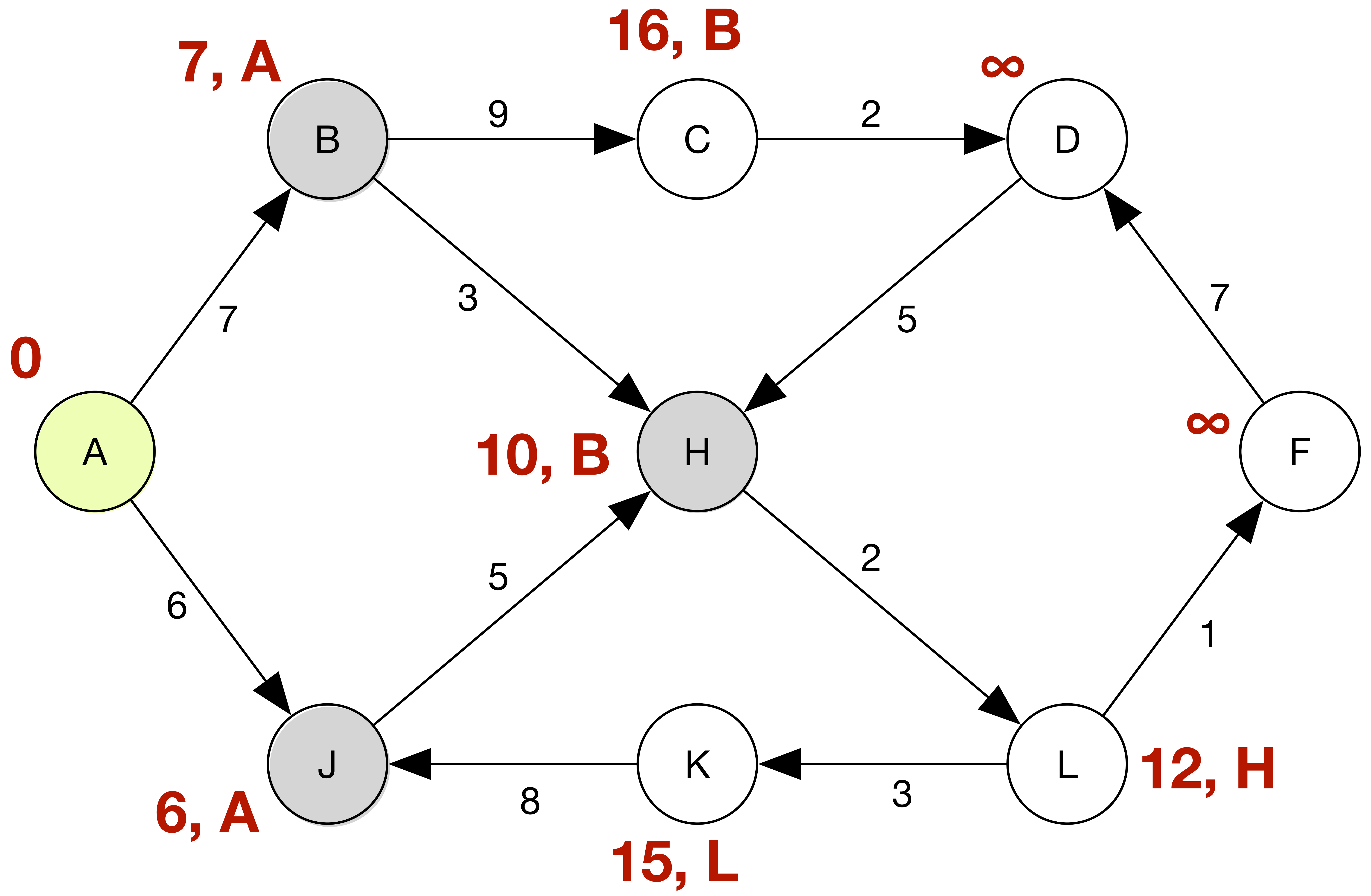


Dijkstra

Explore from L and calculate distances

Unvisited set

U = {C, D,
F, K, L}

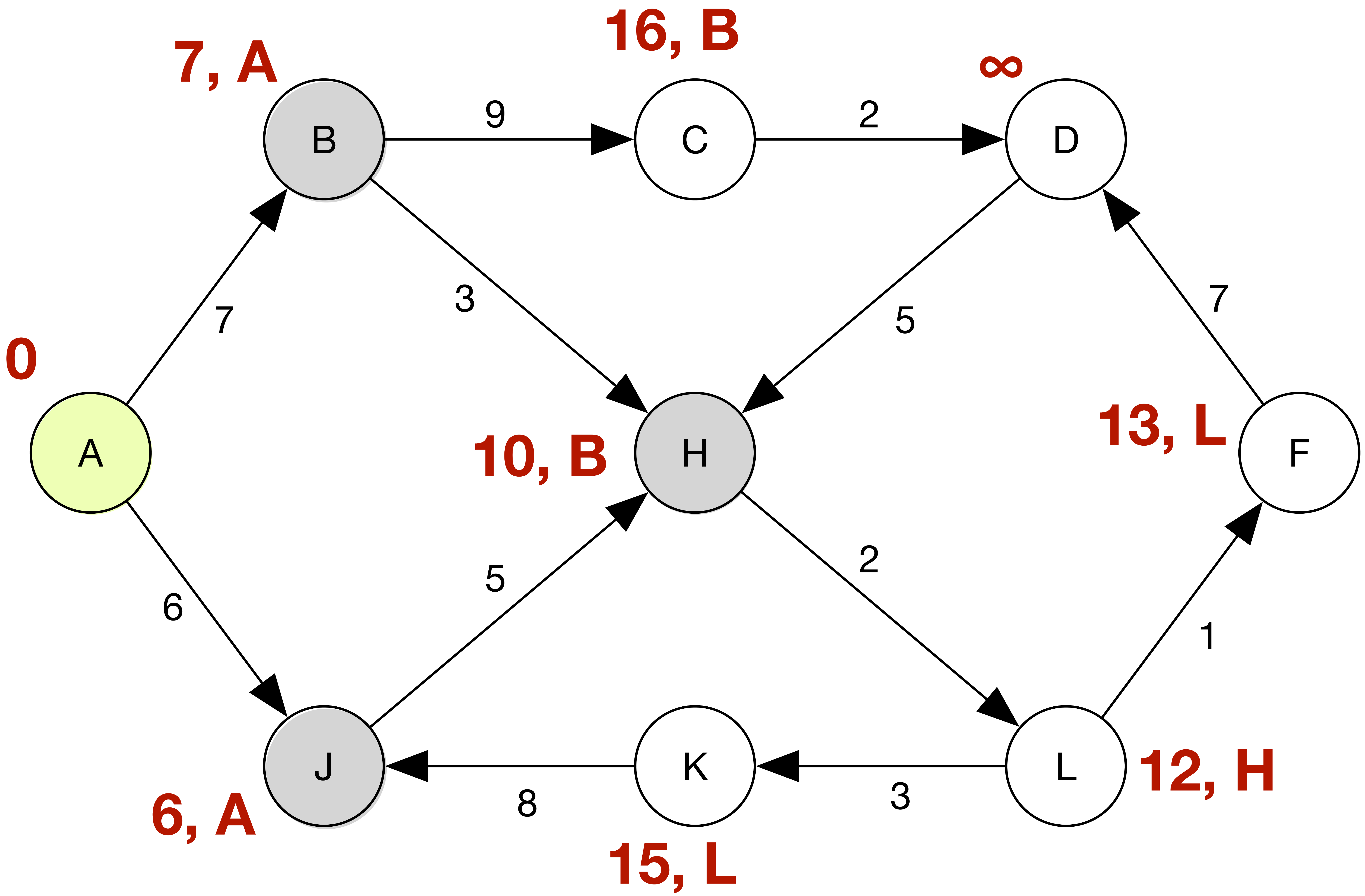


Dijkstra

Unvisited set

U = {C, D,
F, K, L}

Explore from L and calculate distances

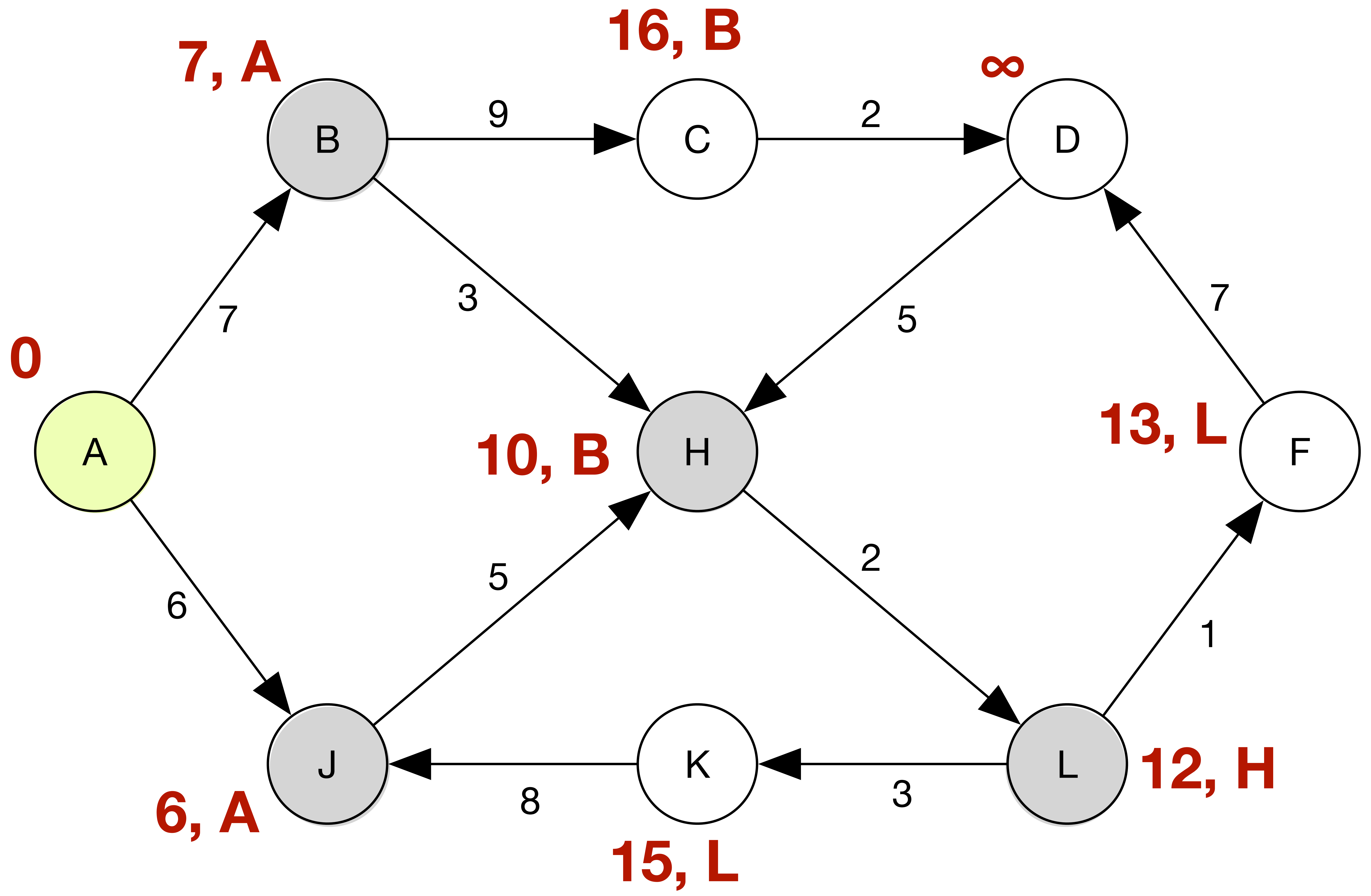


Dijkstra

Mark L as visited (remove from set U)

Unvisited set

U = {C, D,
F, K}

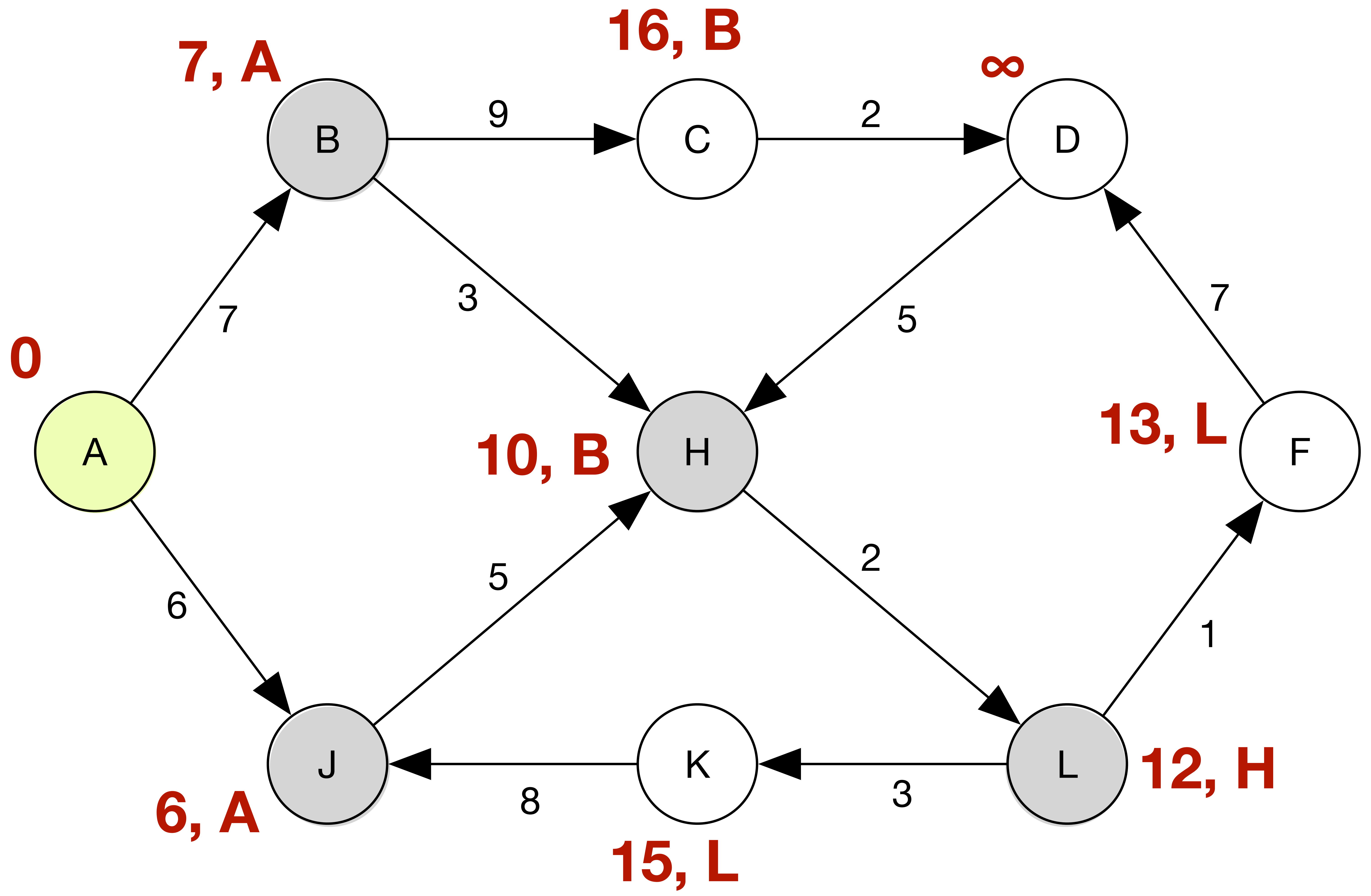


Dijkstra

Choose next node from which to explore

Unvisited set

U = {C, D,
F, K}

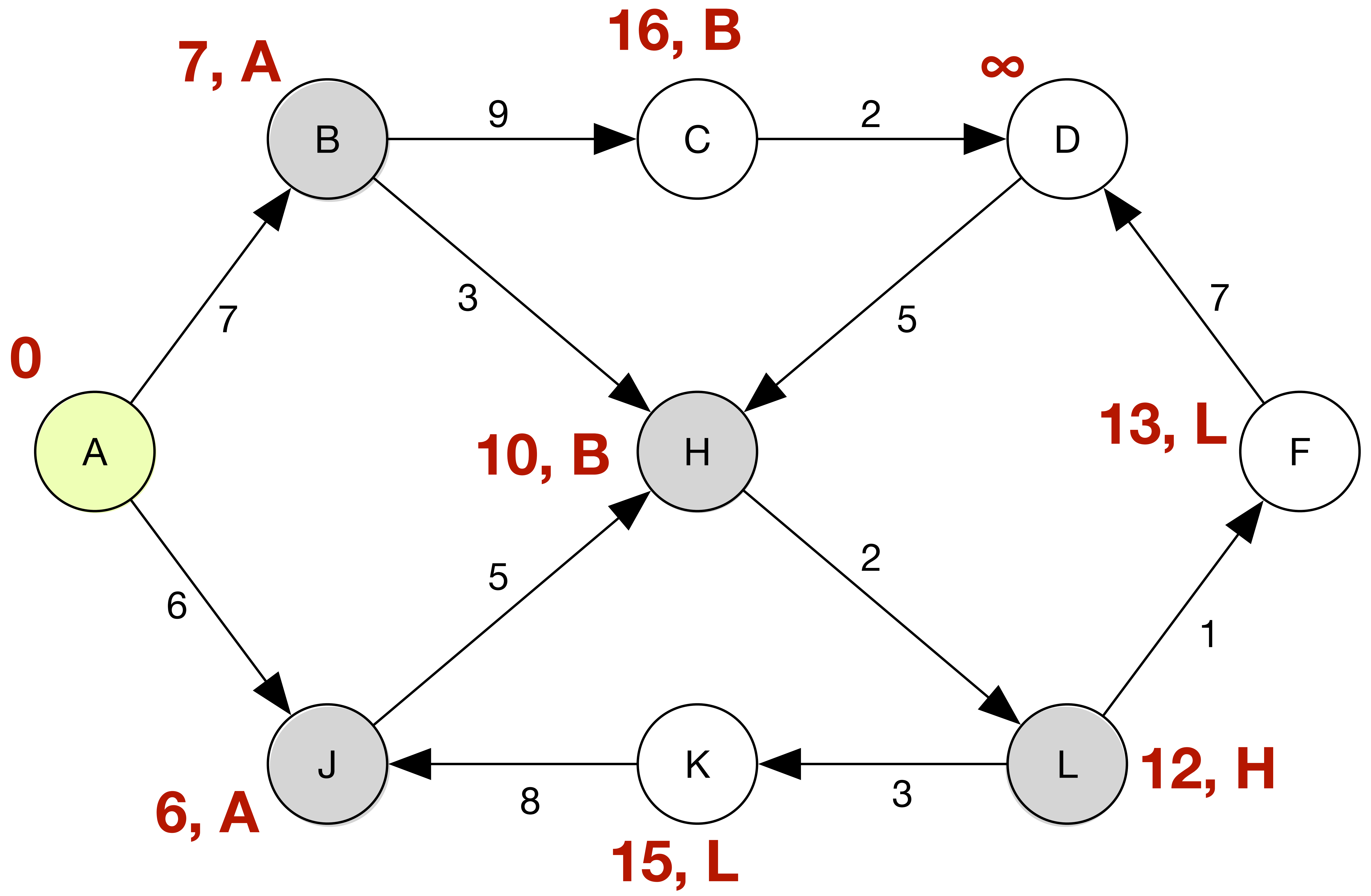


Dijkstra

Unvisited set

U = {C, D, F, K}

Explore from F and calculate distances

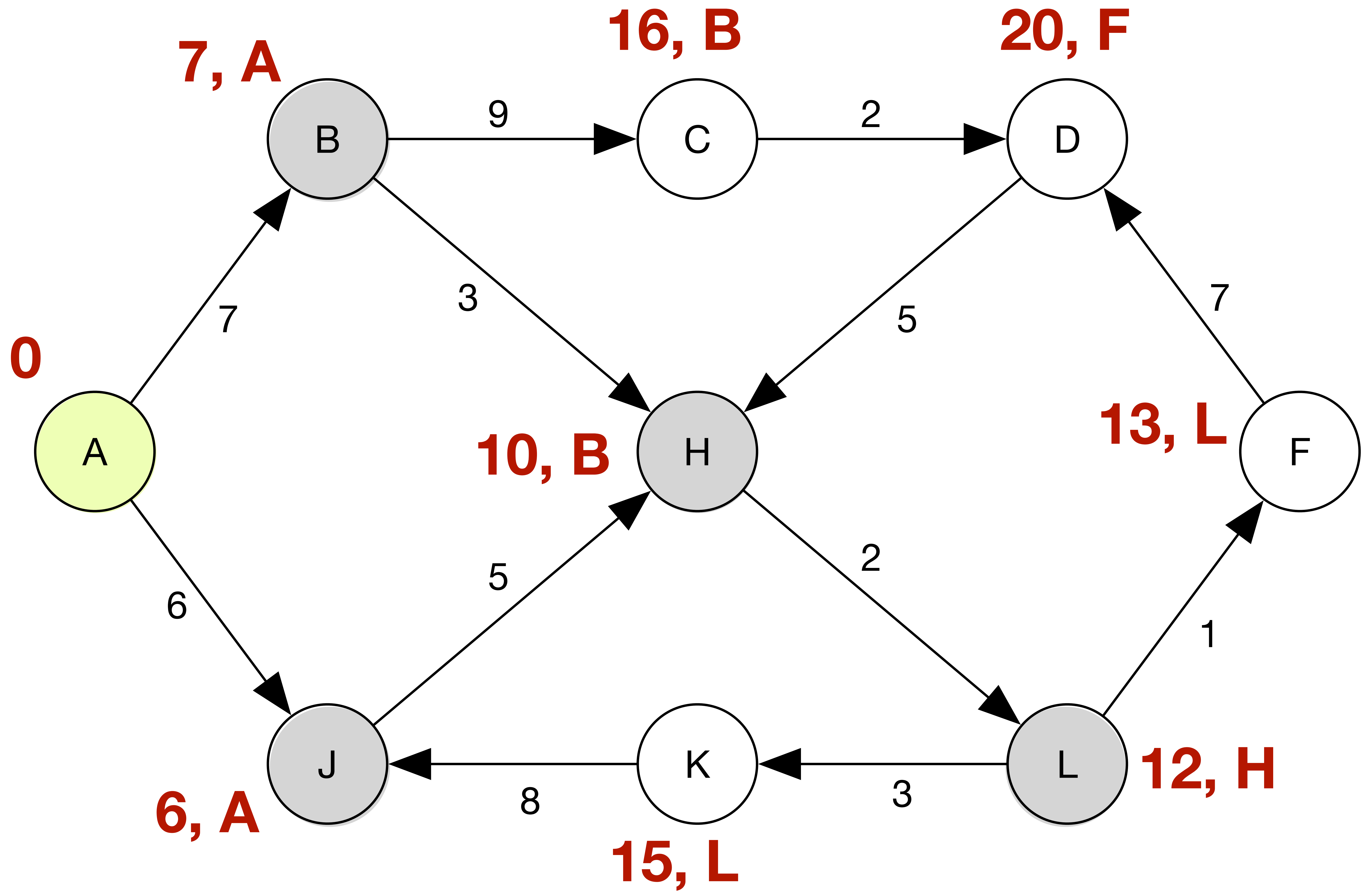


Dijkstra

Unvisited set

U = {C, D, F, K}

Explore from F and calculate distances

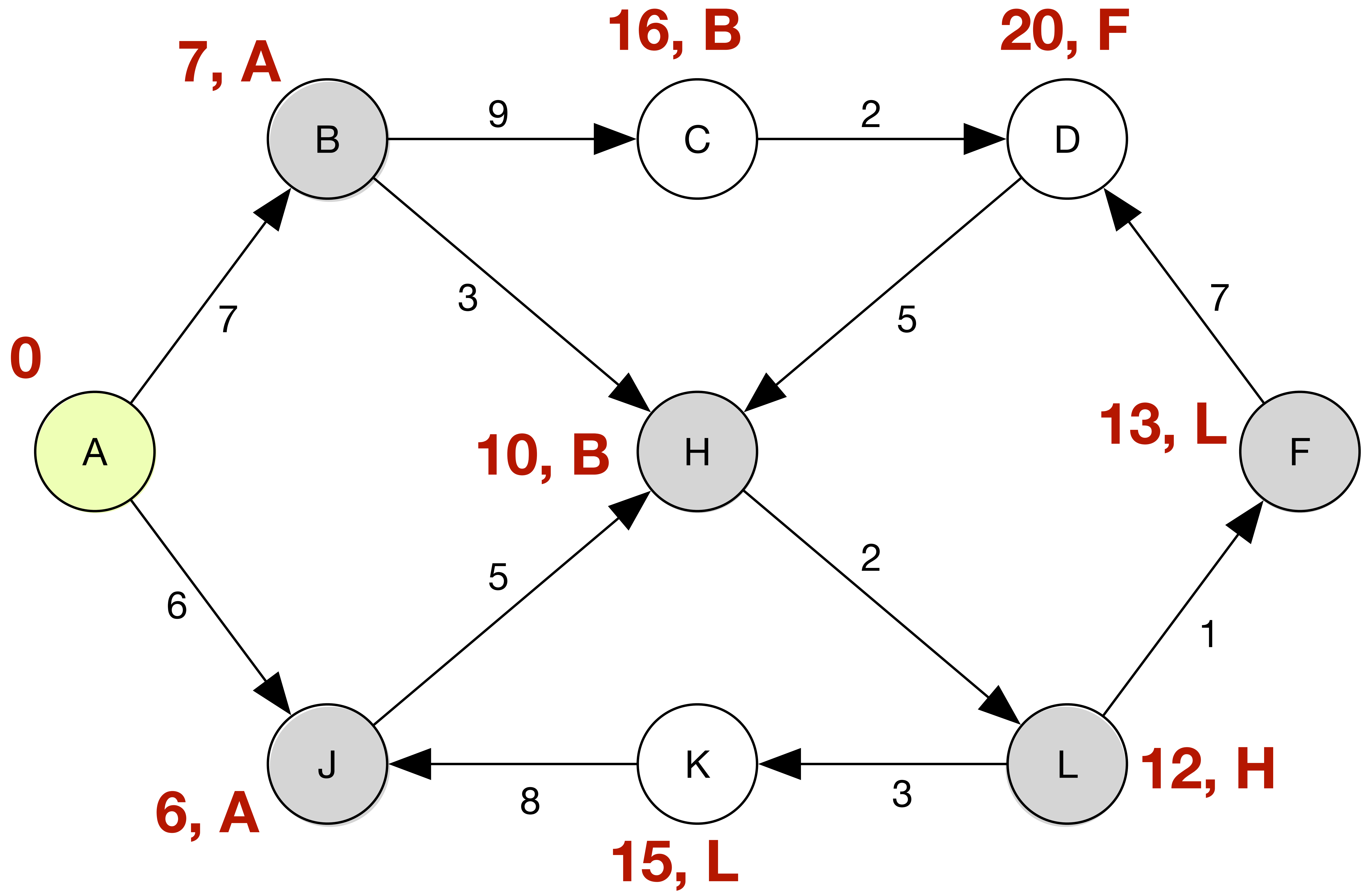


Dijkstra

Unvisited set

U = {C, D, K}

Mark F as visited

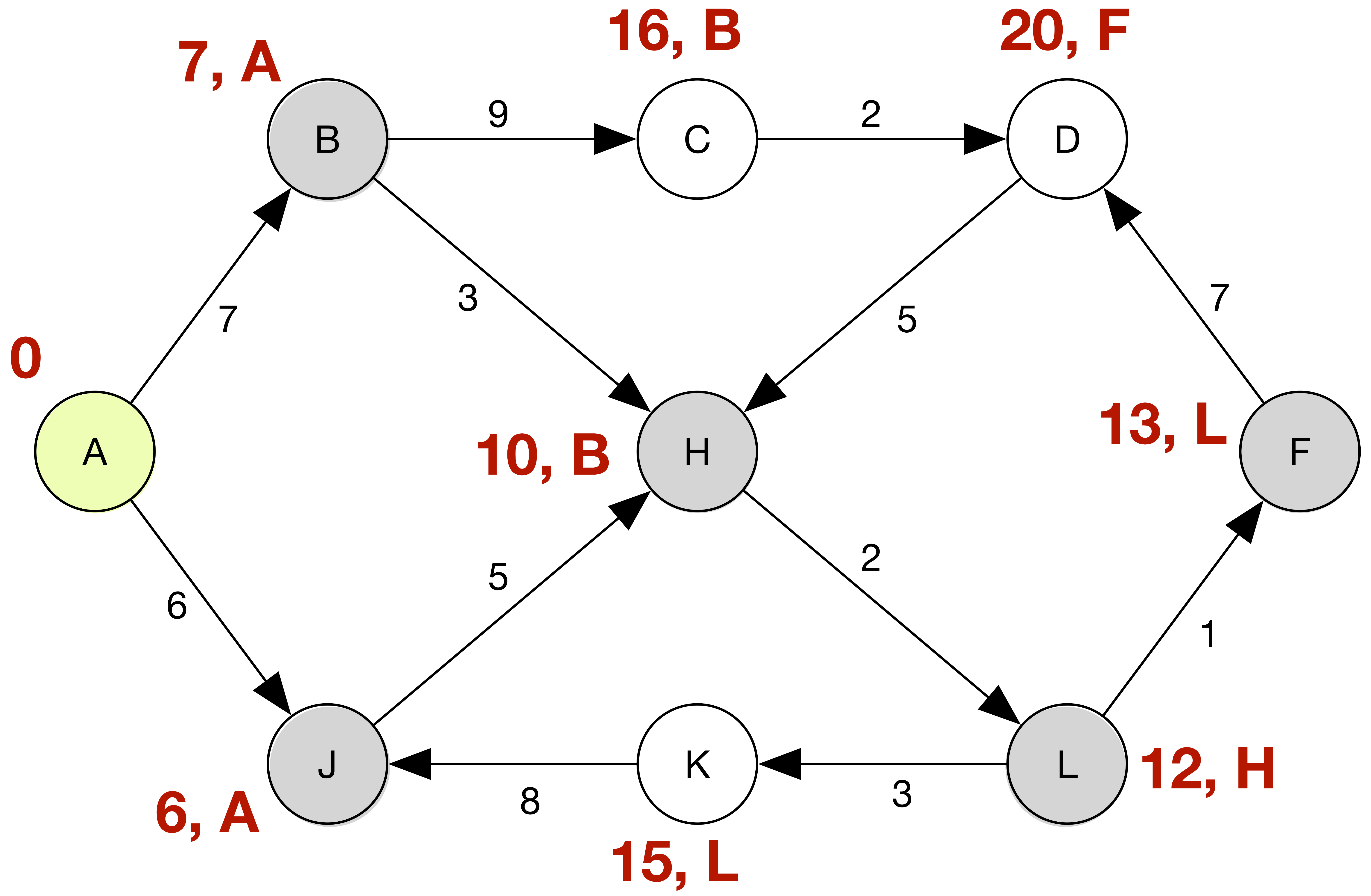


Dijkstra

Choose next node from which to explore

Unvisited set

U = {C, D, K}

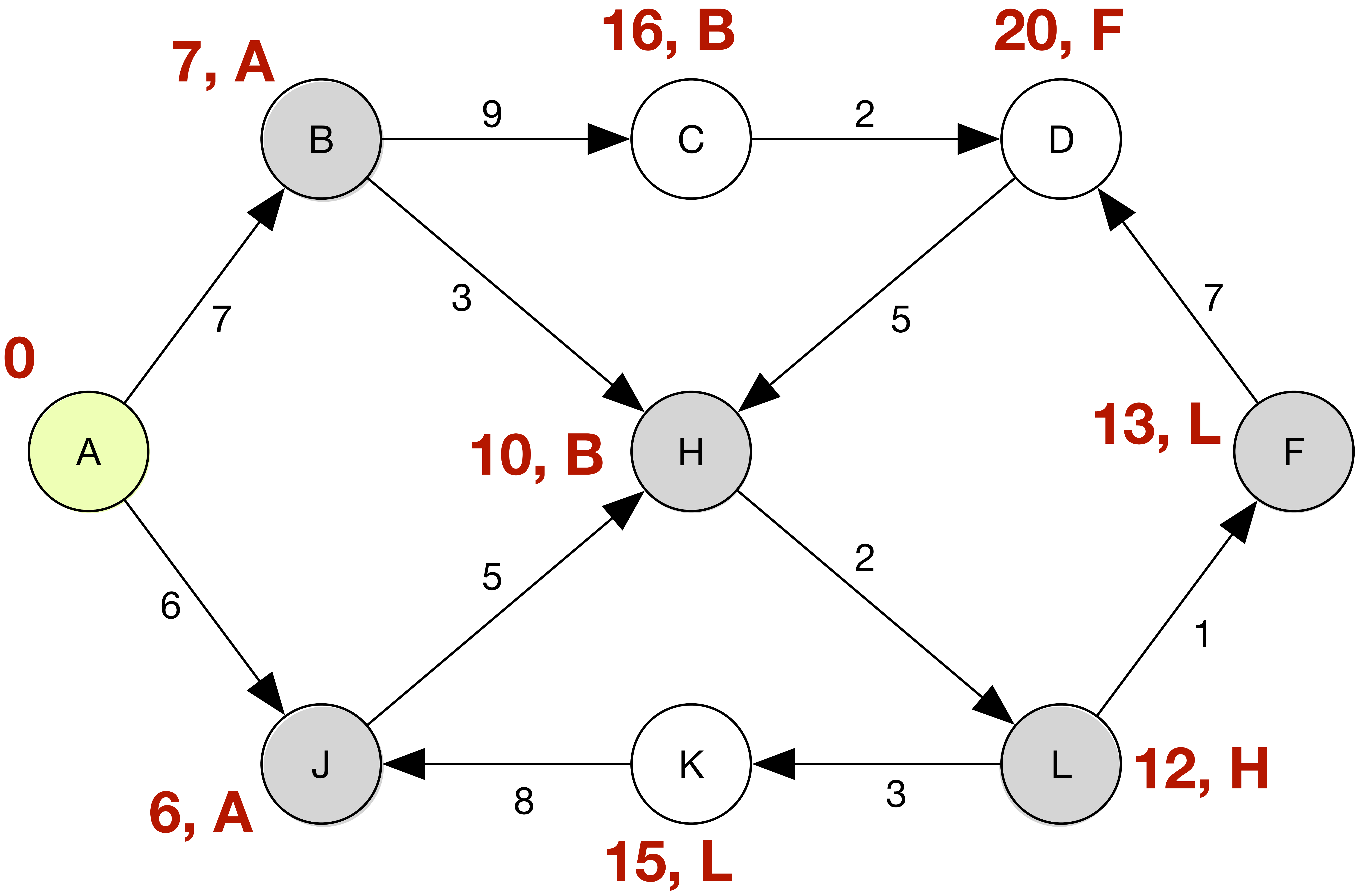


Dijkstra

Unvisited set

U = {C, D, K}

Explore from K and calculate distances

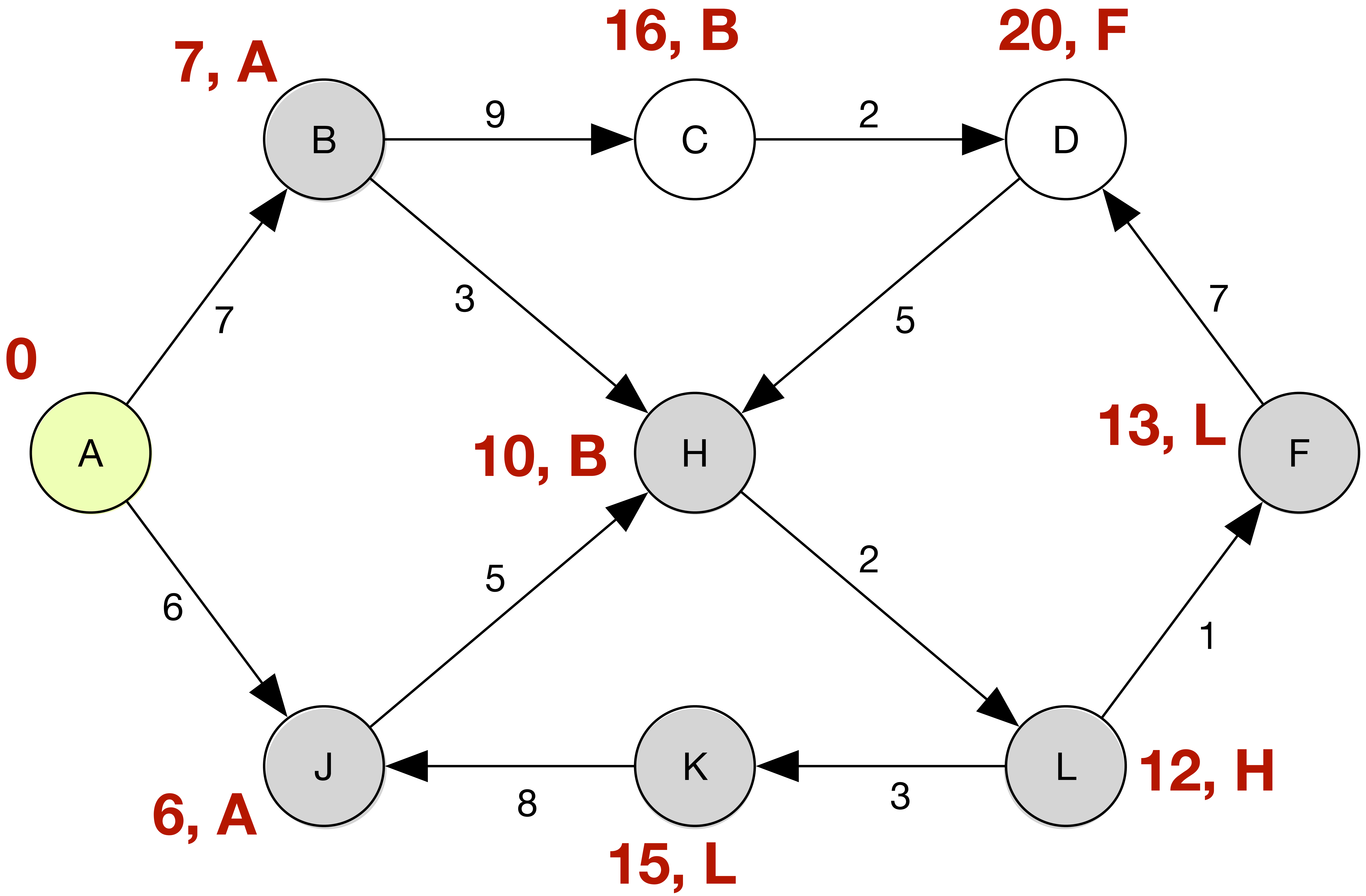


Dijkstra

Mark K as visited (remove from set U)

Unvisited set

U = {C, D}

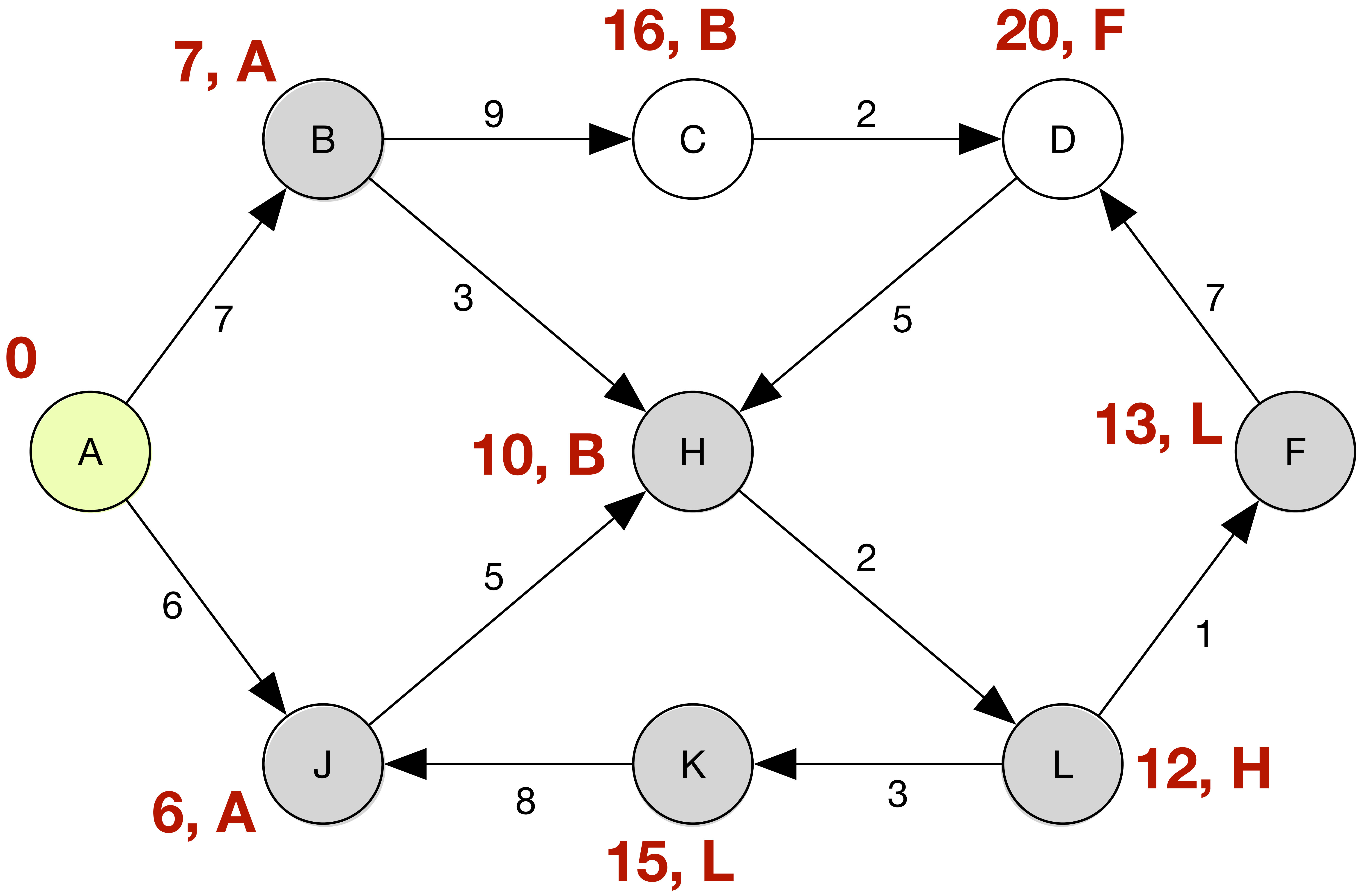


Dijkstra

Choose next node from which to explore

Unvisited set

U = {C, D}

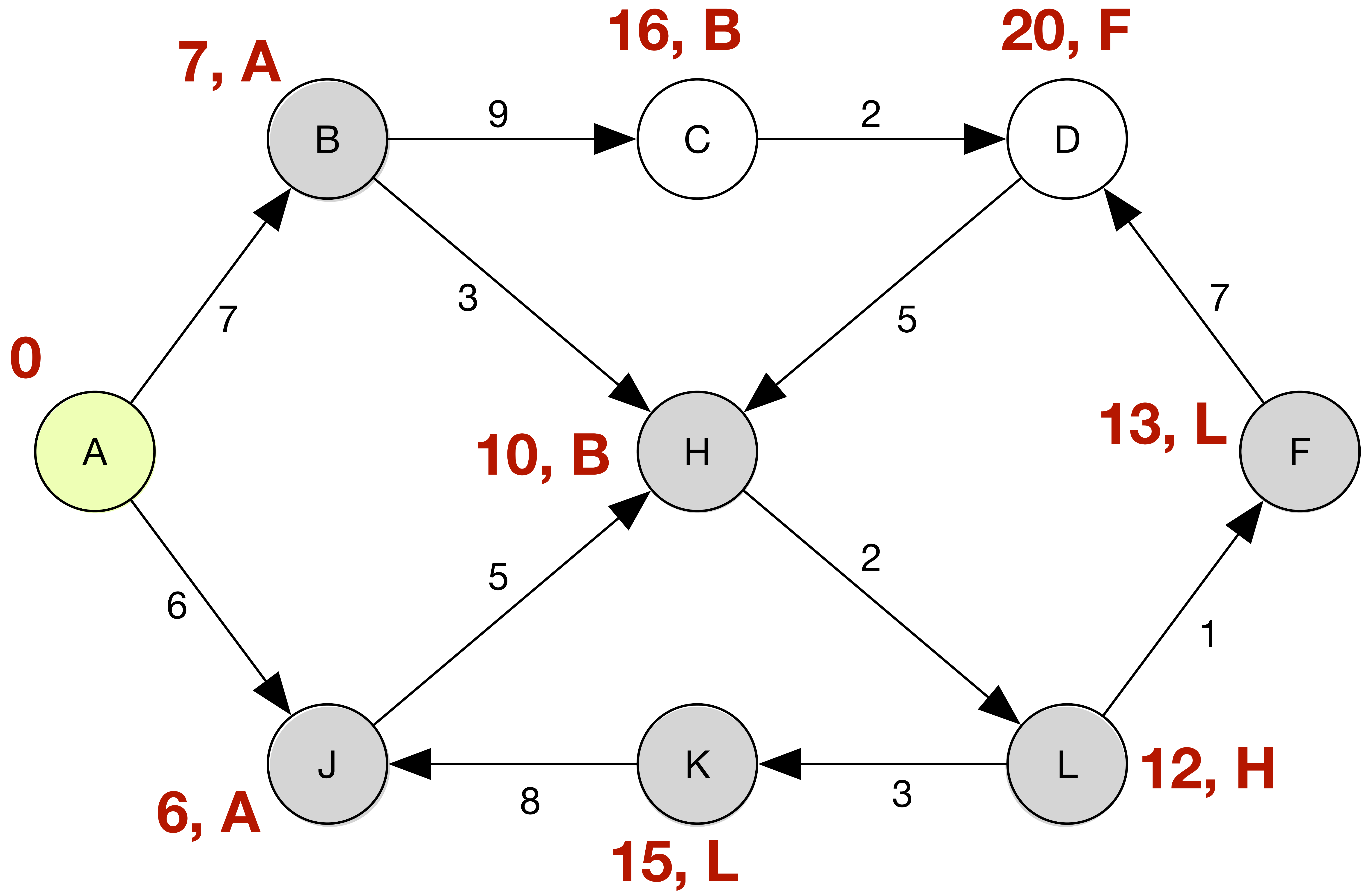


Dijkstra

Unvisited set

U = {C, D}

Explore from C and calculate distances

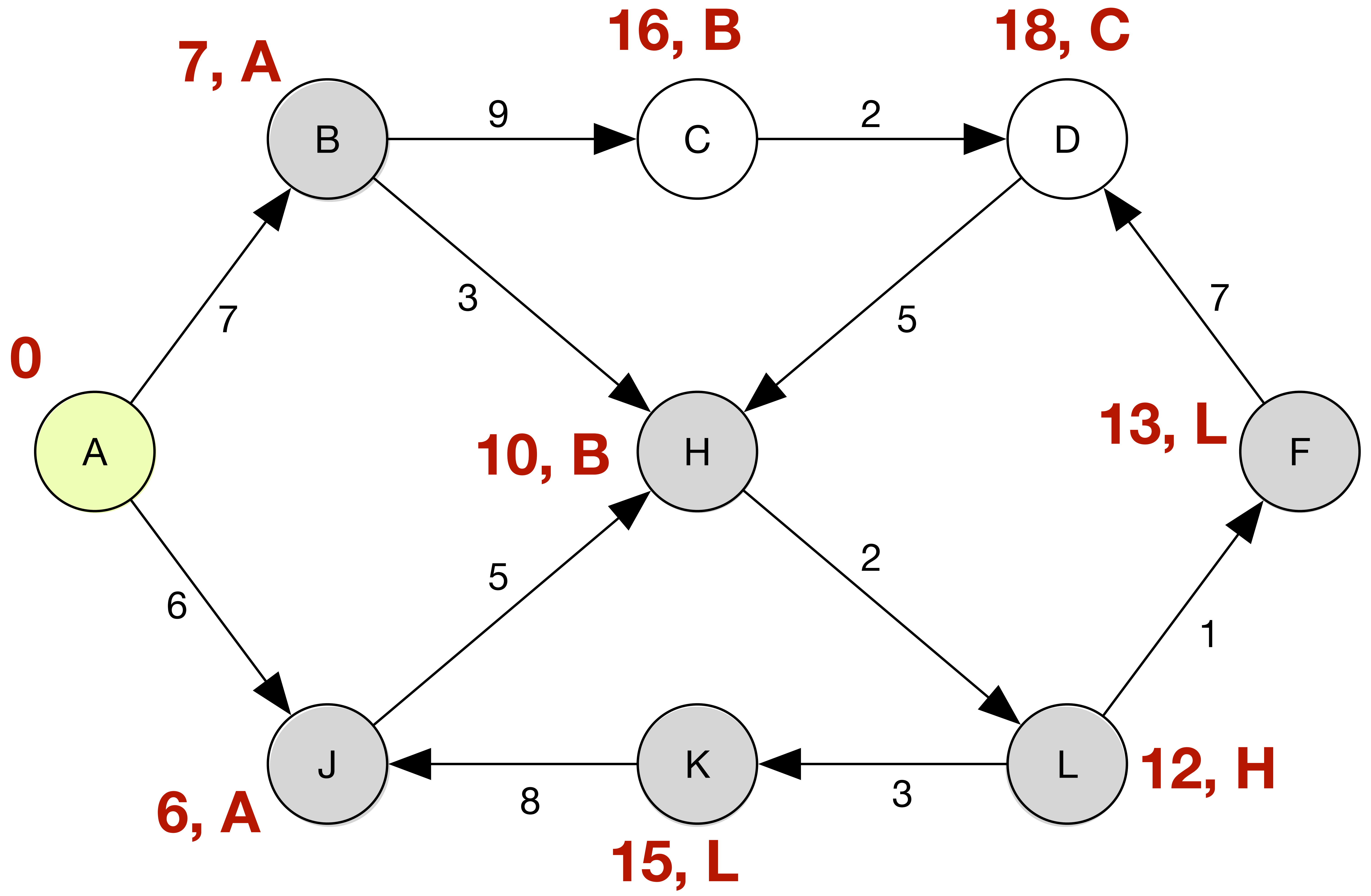


Dijkstra

Unvisited set

U = {C, D}

Explore from C and calculate distances

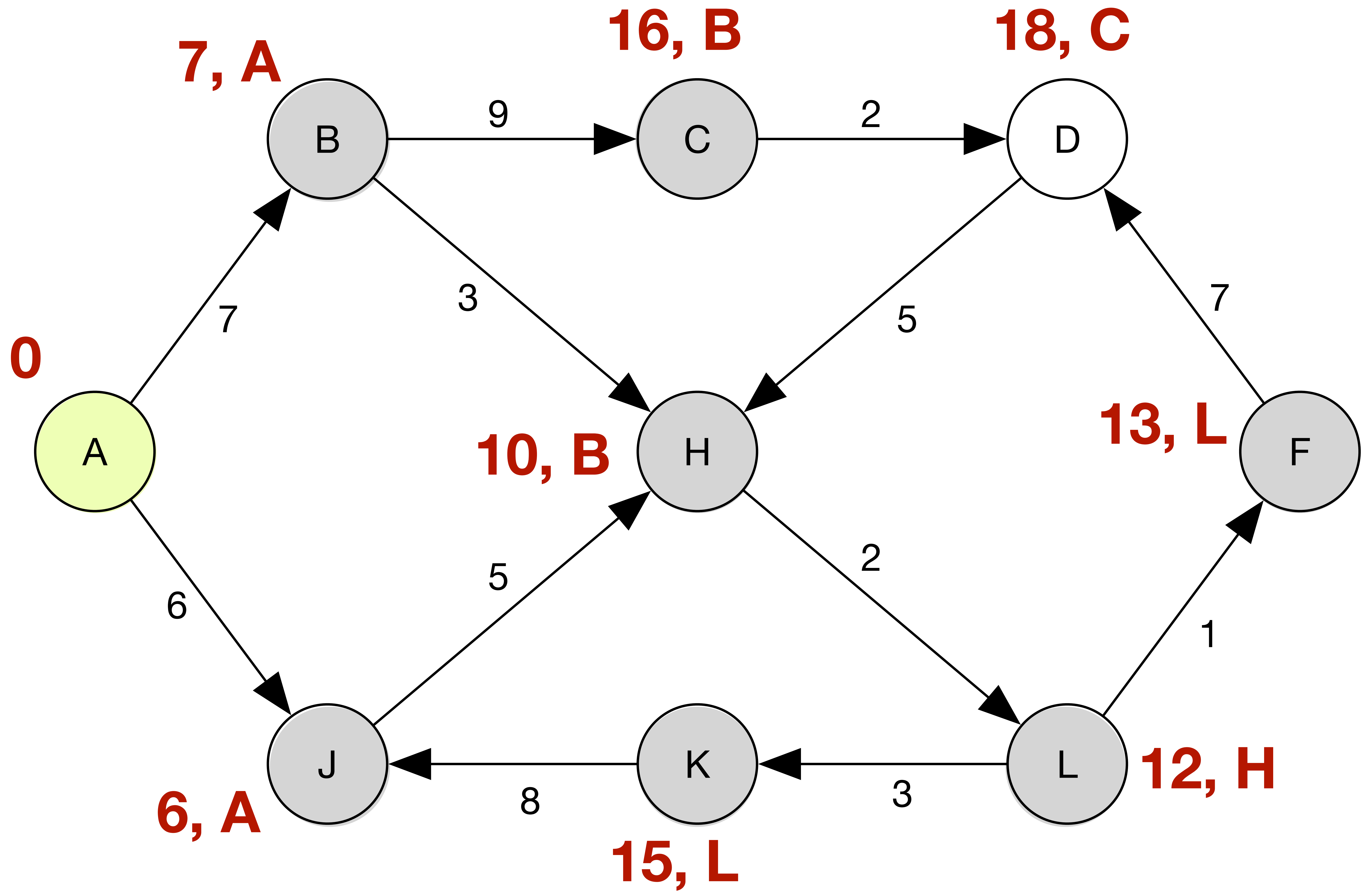


Dijkstra

Mark C as visited (remove from set U)

Unvisited set

U = {D}

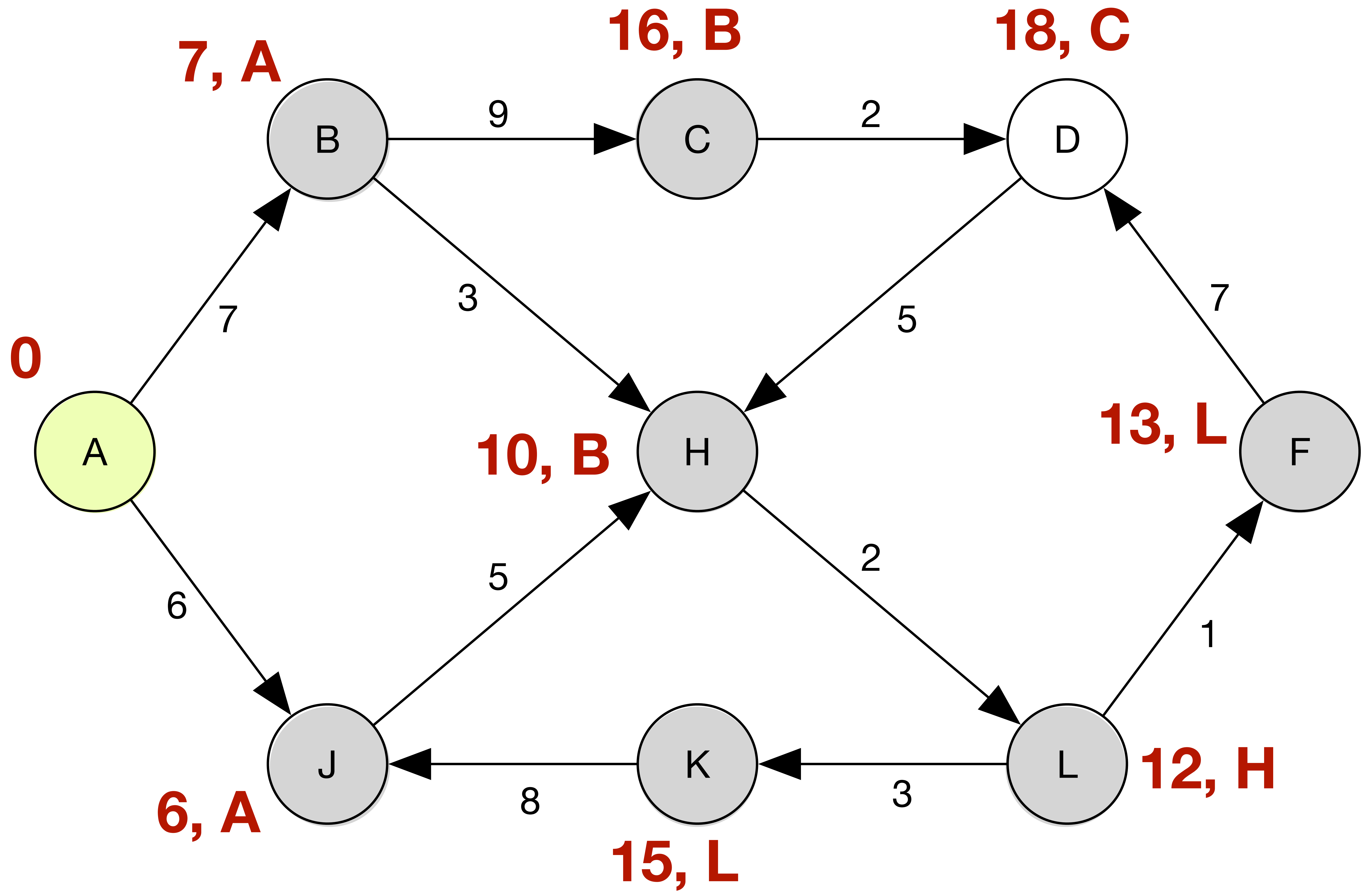


Dijkstra

Explore from D and calculate distances.

Unvisited set

U = {D}

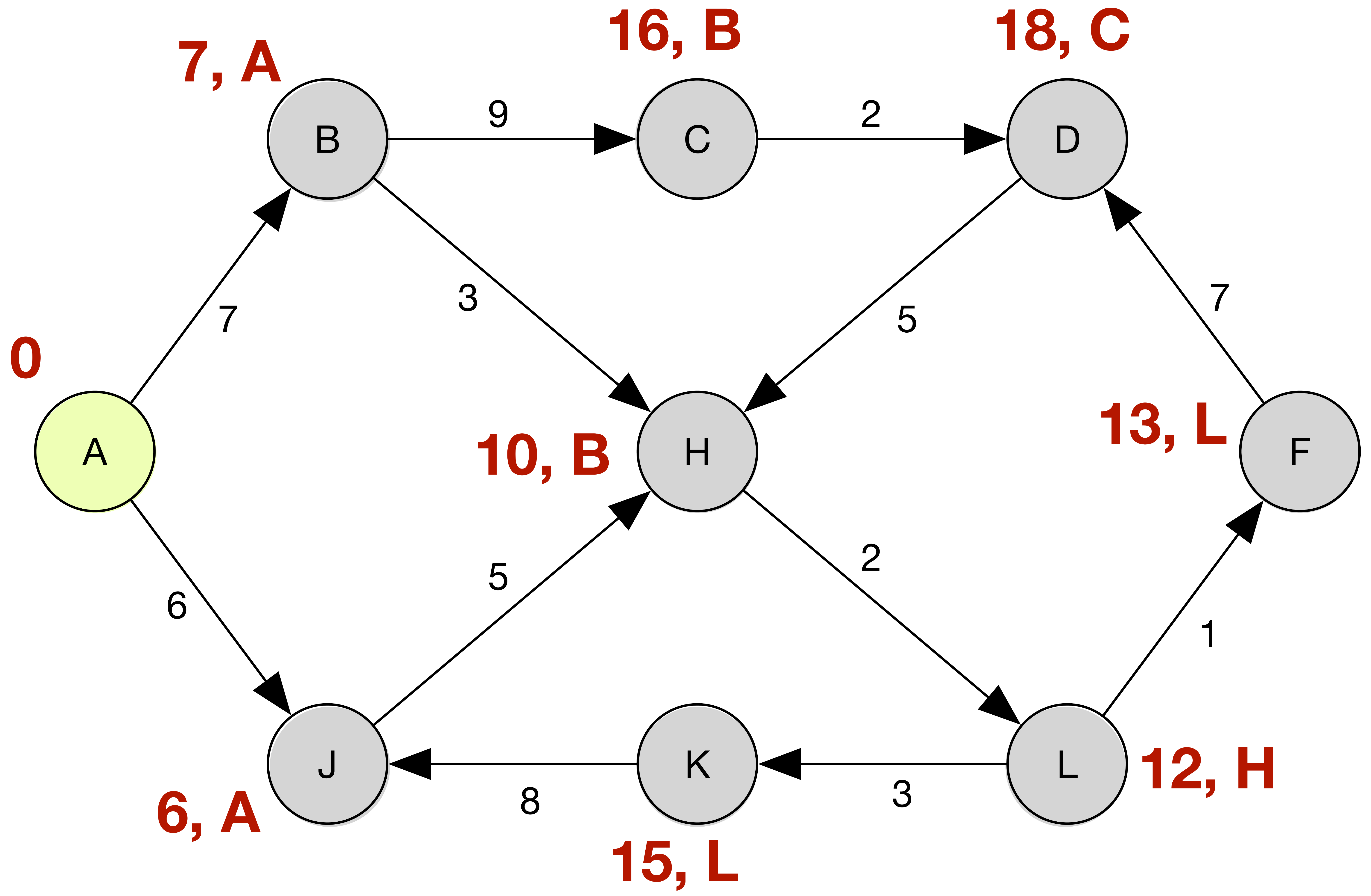


Dijkstra

Unvisited set

U = {}

Mark D as visited (remove from set U)

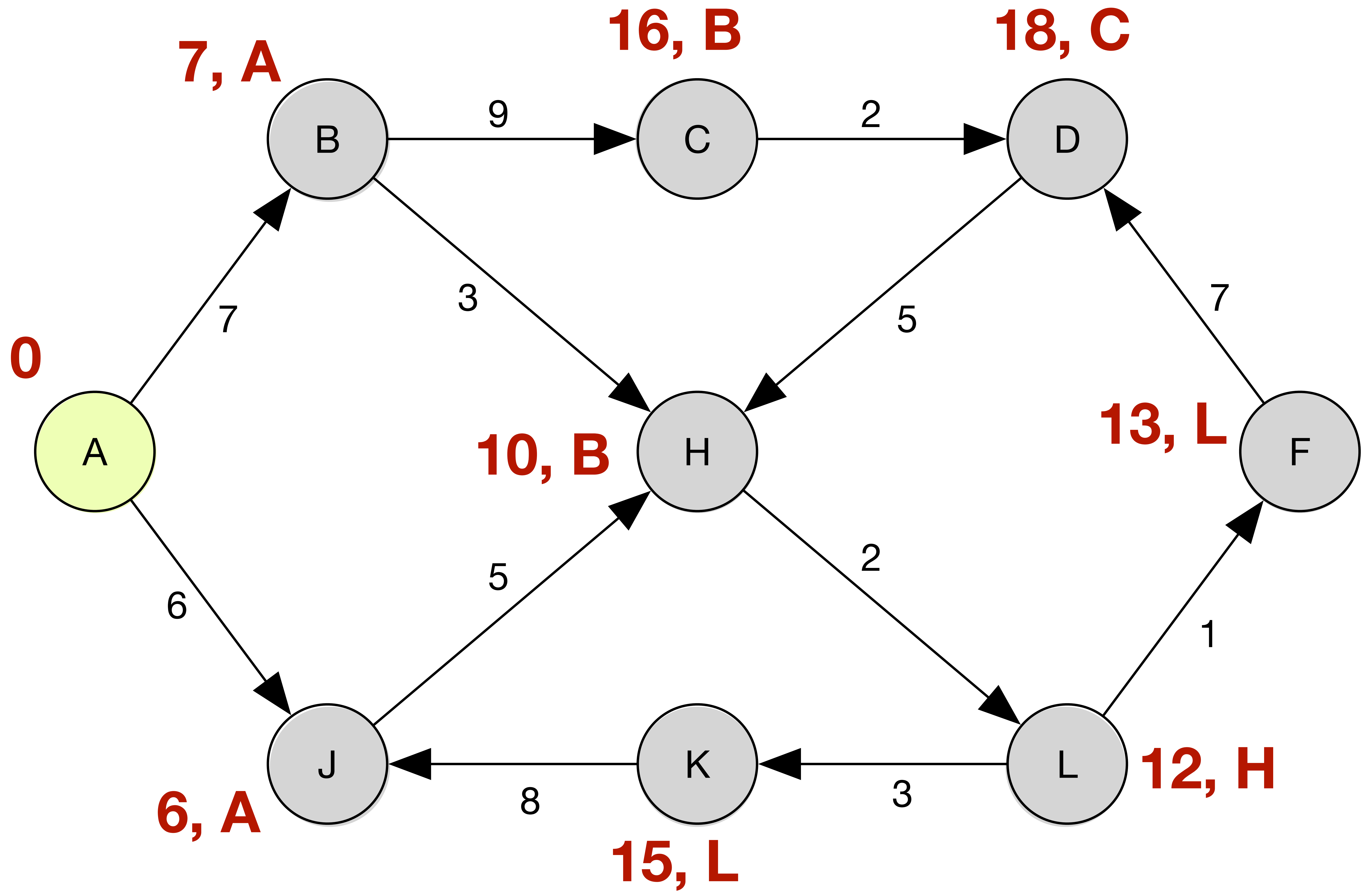


Dijkstra

Done!

Unvisited set

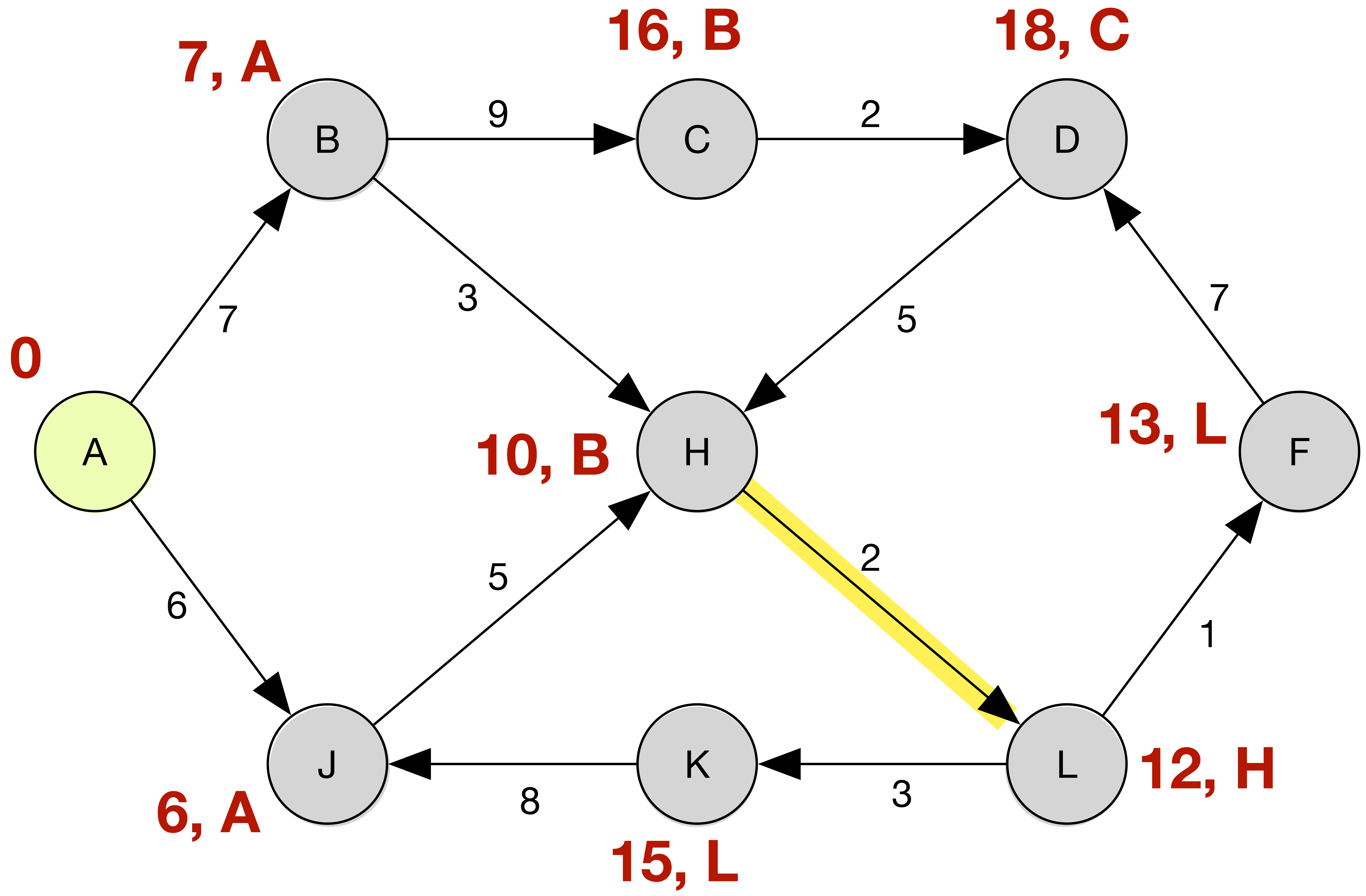
U = {}



Dijkstra

Unvisited set

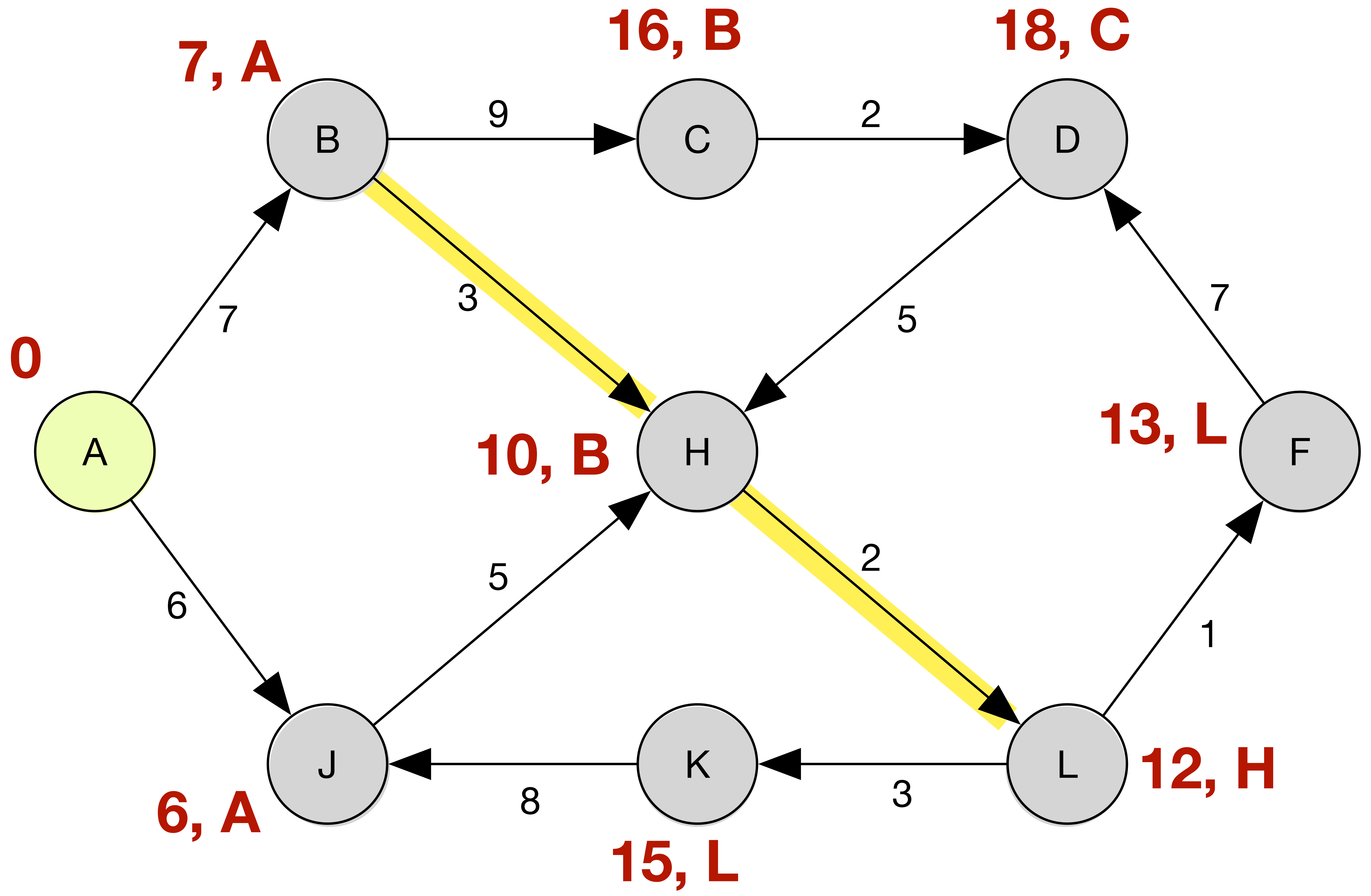
U = {}



Dijkstra

Unvisited set

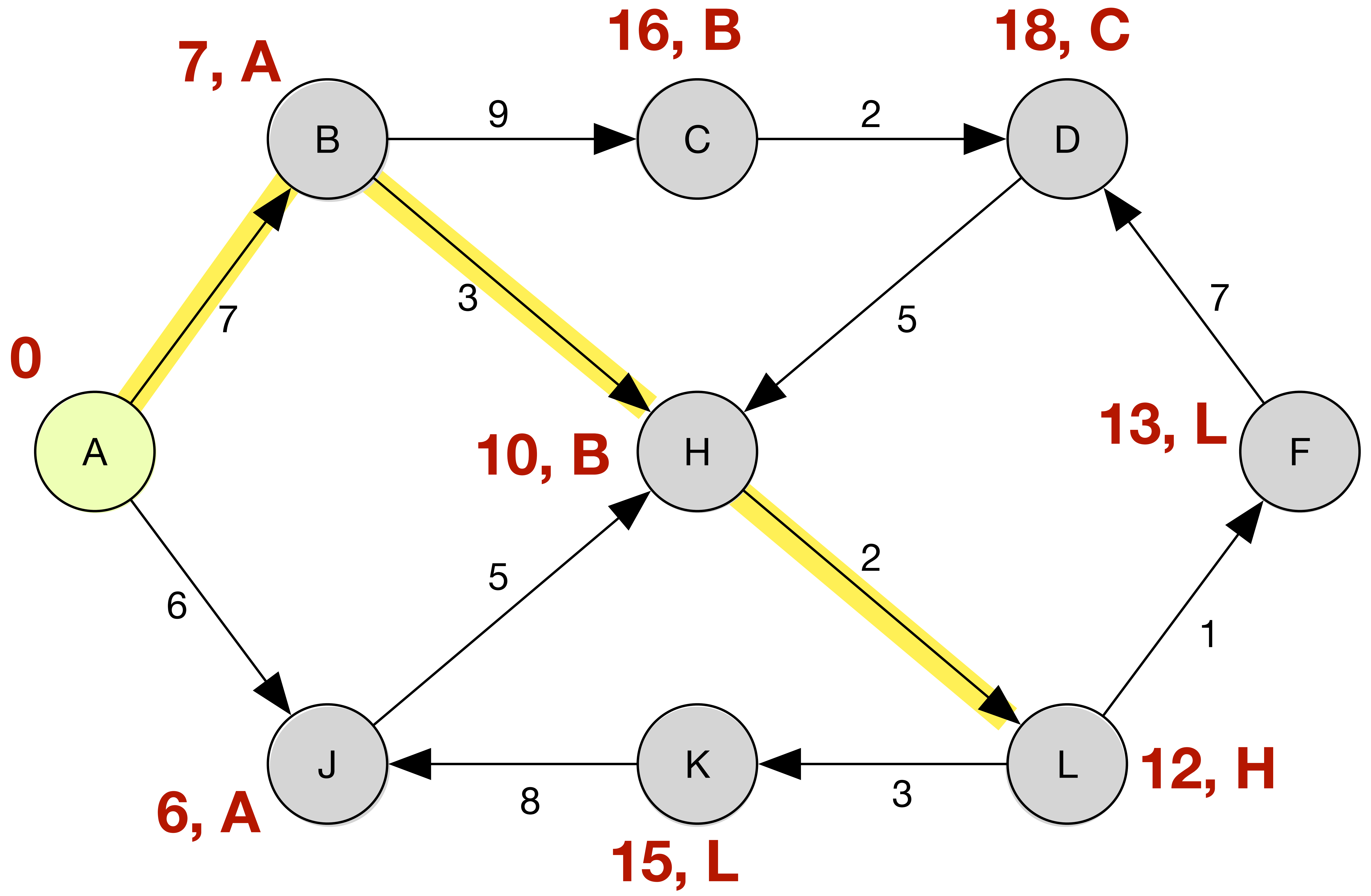
U = {}



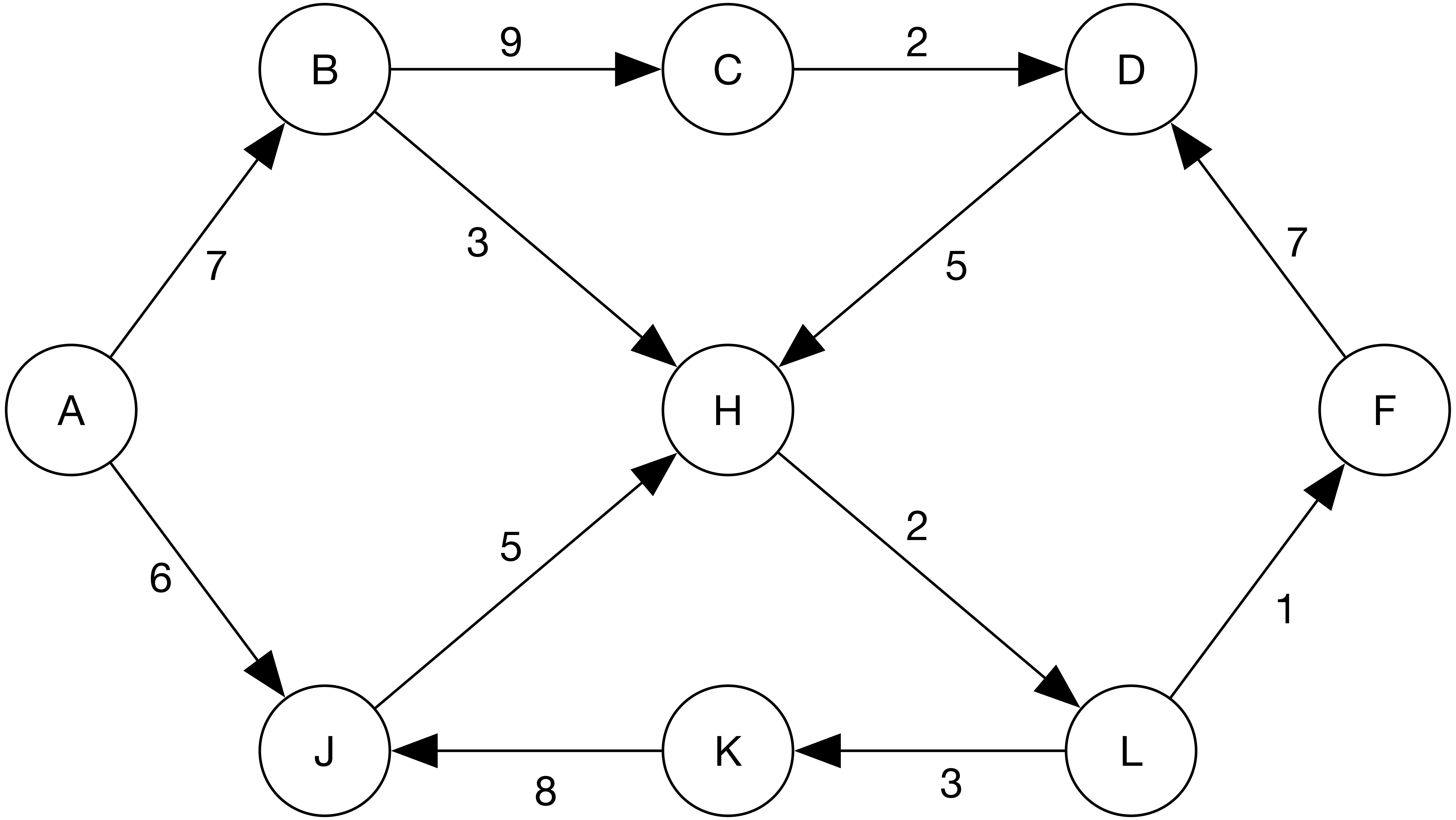
Dijkstra

Unvisited set

U = {}



Dijkstra



Dijkstra's algorithm

What did you notice about how we chose the nodes to visit?

Dijkstra's algorithm

What did you notice about how we chose the nodes to visit?

We always chose the node with the smallest distance.

Dijkstra's algorithm

What did you notice about how we chose the nodes to visit?

We always chose the node with the smallest distance.

Can you think of a data structure that's handy for always extracting the smallest value?

Dijkstra's algorithm

What did you notice about how we chose the nodes to visit?

We always chose the node with the smallest distance.

Can you think of a data structure that's handy for always extracting the smallest value?

A minimum priority queue!

Dijkstra's algorithm: pseudocode

```
function dijkstra(G, S)  // G is the graph; S is the starting node
    for each node V in G
        arrived_from[V] = null
        if V = S
             distance[V] = 0
        else
            distance[V] = infinite
        add V to priority queue Q

    while Q is not empty
        V = get min from Q
        for each unvisited neighbor N of V
            distance = distance[V] + distance to N
            if distance < distance[N]  // We've found a shorter distance
                distance[N] = distance
                arrived_from[N] = V
```

Dijkstra's algorithm

Dijkstra's algorithm works for any directed graph so long as all weights are non-negative

Dijkstra's algorithm

Dijkstra's algorithm works for any directed graph so long as all weights are non-negative

Worst case complexity (using min priority queue)

$$\mathcal{O}((|V| + |E|) \log|V|)$$

Dijkstra's algorithm

Dijkstra's algorithm works for any directed graph so long as all weights are non-negative

Worst case complexity (using min priority queue)

$$\mathcal{O}((|V| + |E|) \log|V|)$$

Greedy algorithm. Always chooses the best (shortest) distance found so far.

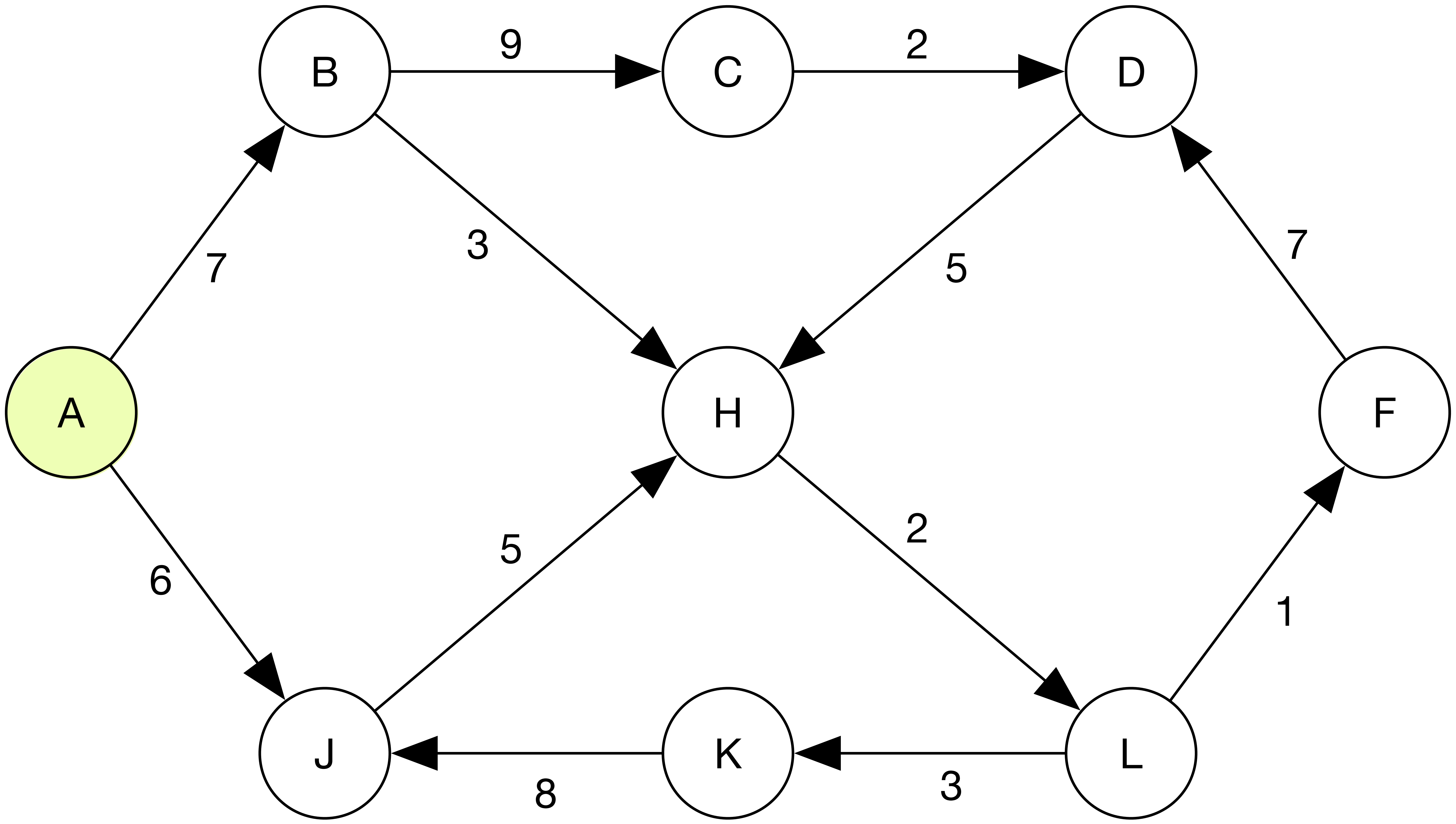
Dijkstra's algorithm

Greedy approach. We always use the best (shortest) distance we've calculated so far, and we never go back -- once a node is marked as visited we never revisit it.

Dijkstra

Unvisited set

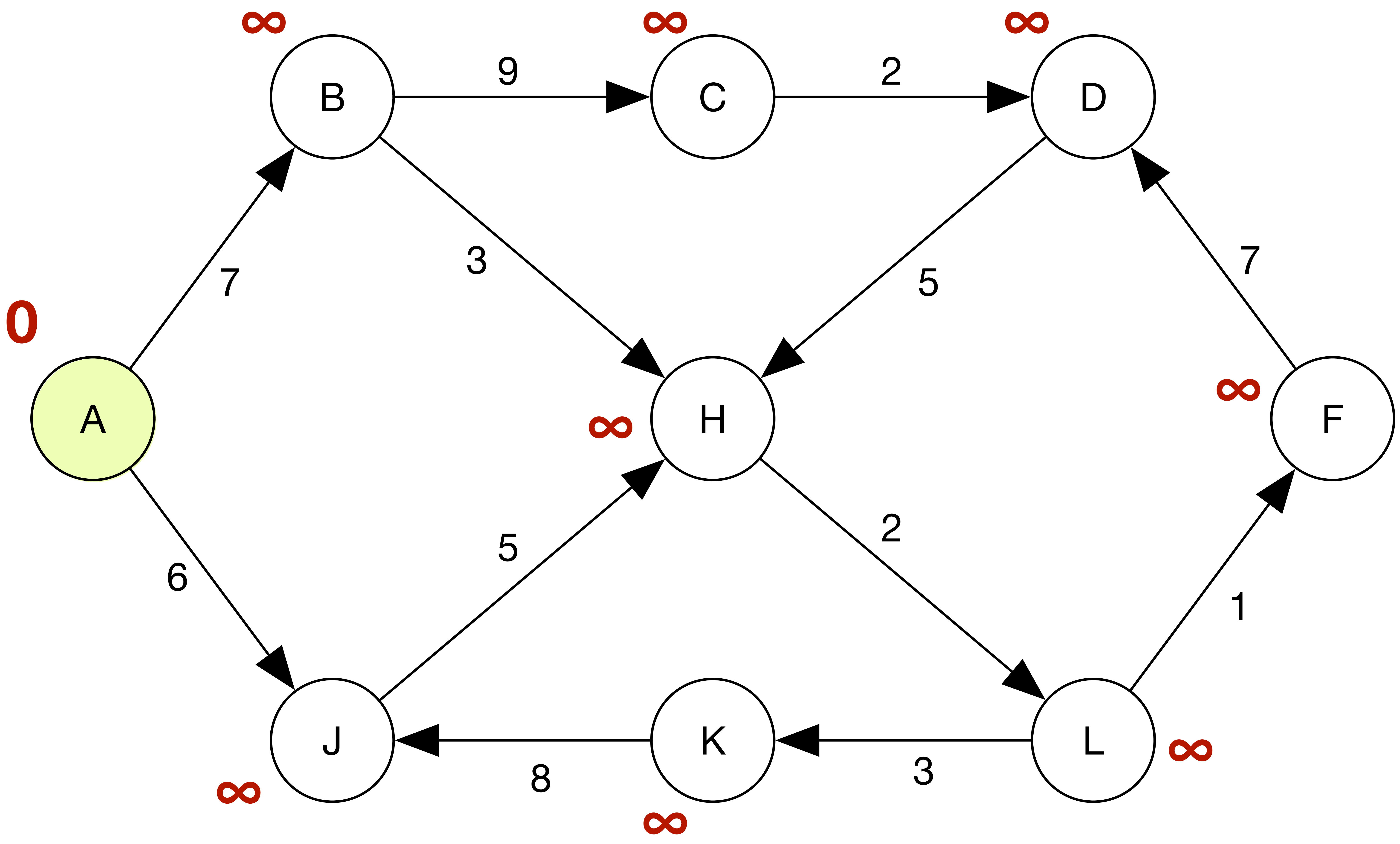
U = {A, B, C, D,
F, H, J, K, L}



Dijkstra

Unvisited set

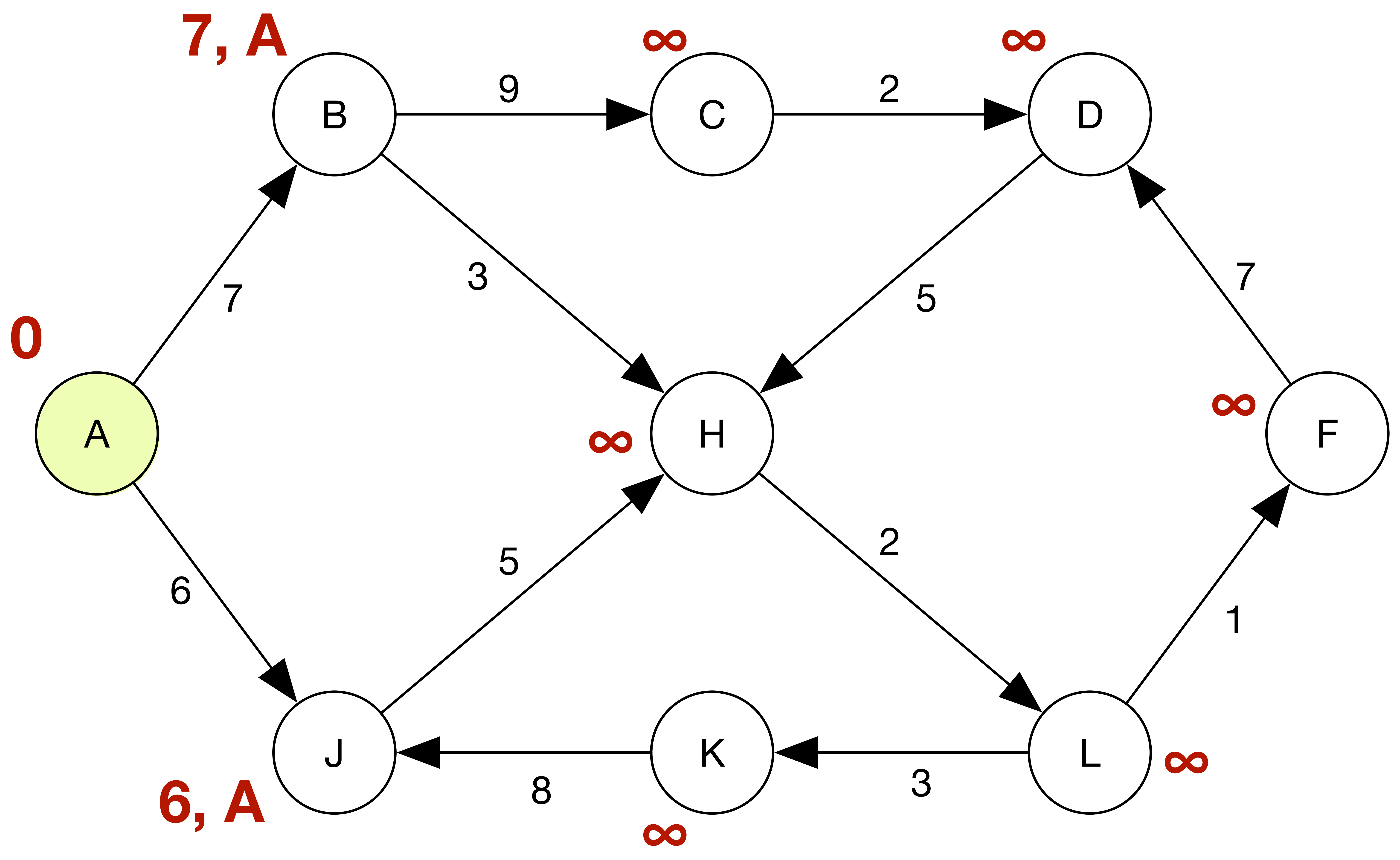
U = {A, B, C, D,
F, H, J, K, L}



Dijkstra

Unvisited set

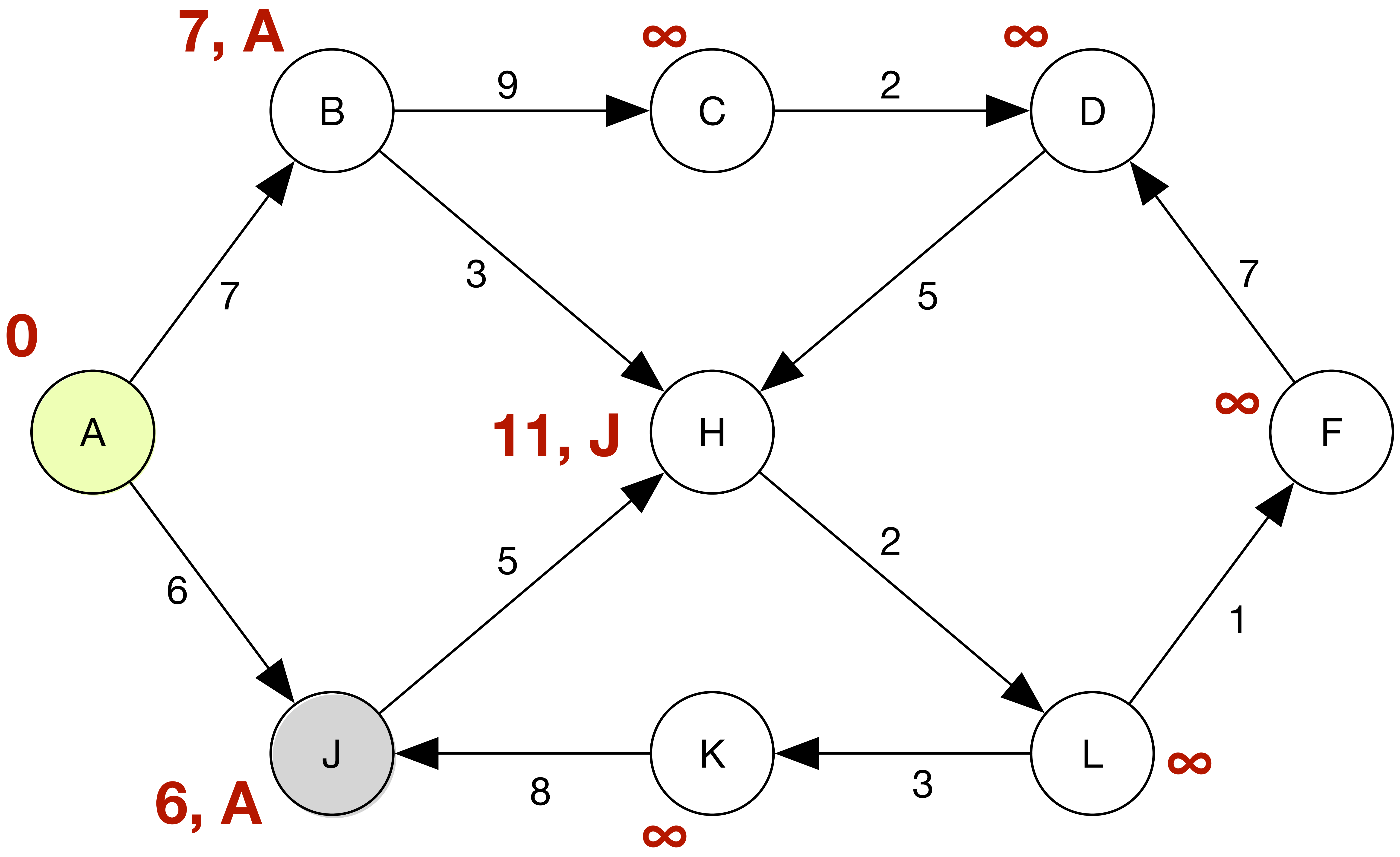
U = {A, B, C, D,
F, H, J, K, L}



Dijkstra

Unvisited set

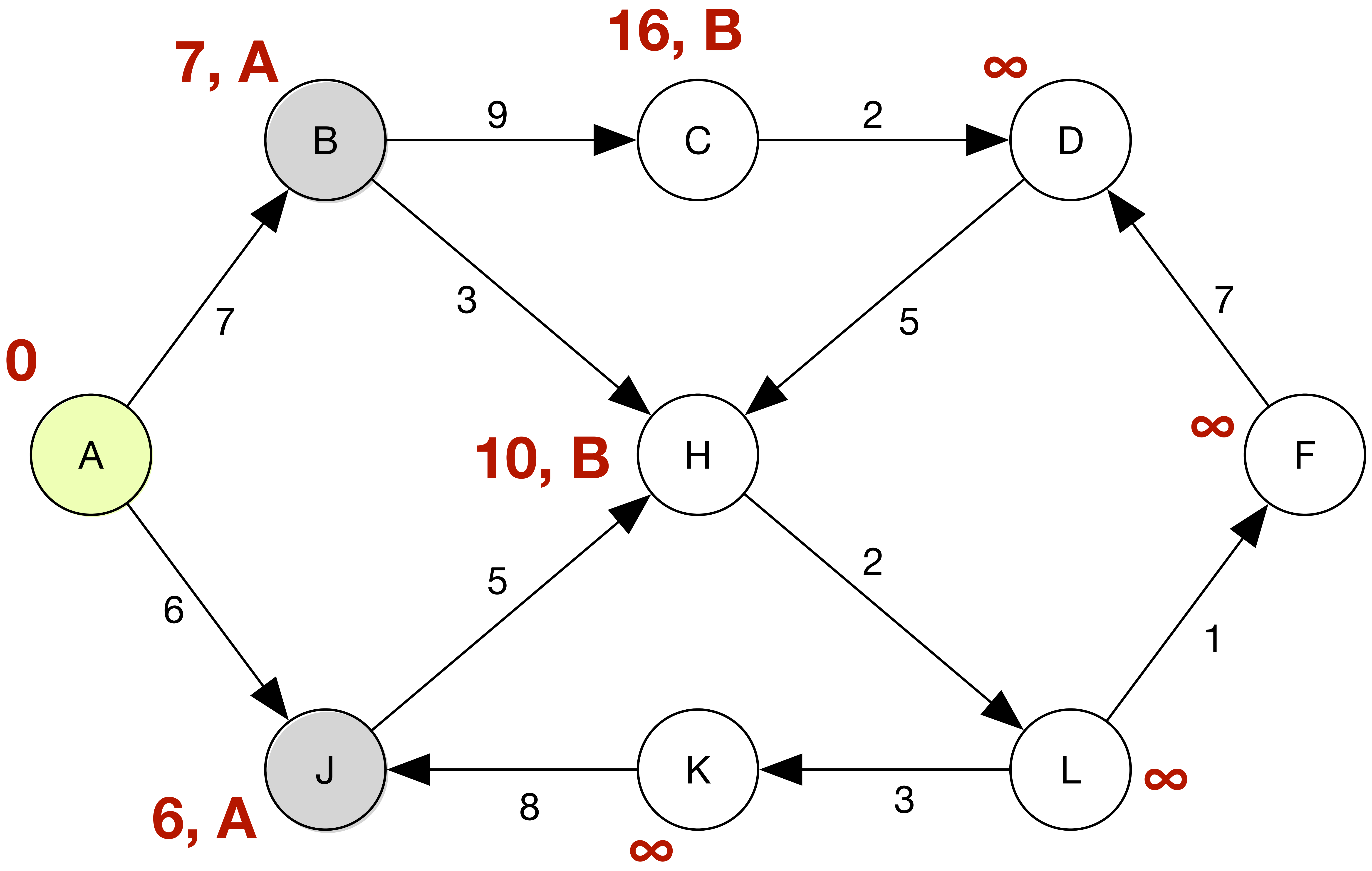
U = {B, C, D,
F, H, K, L}



Dijkstra

Unvisited set

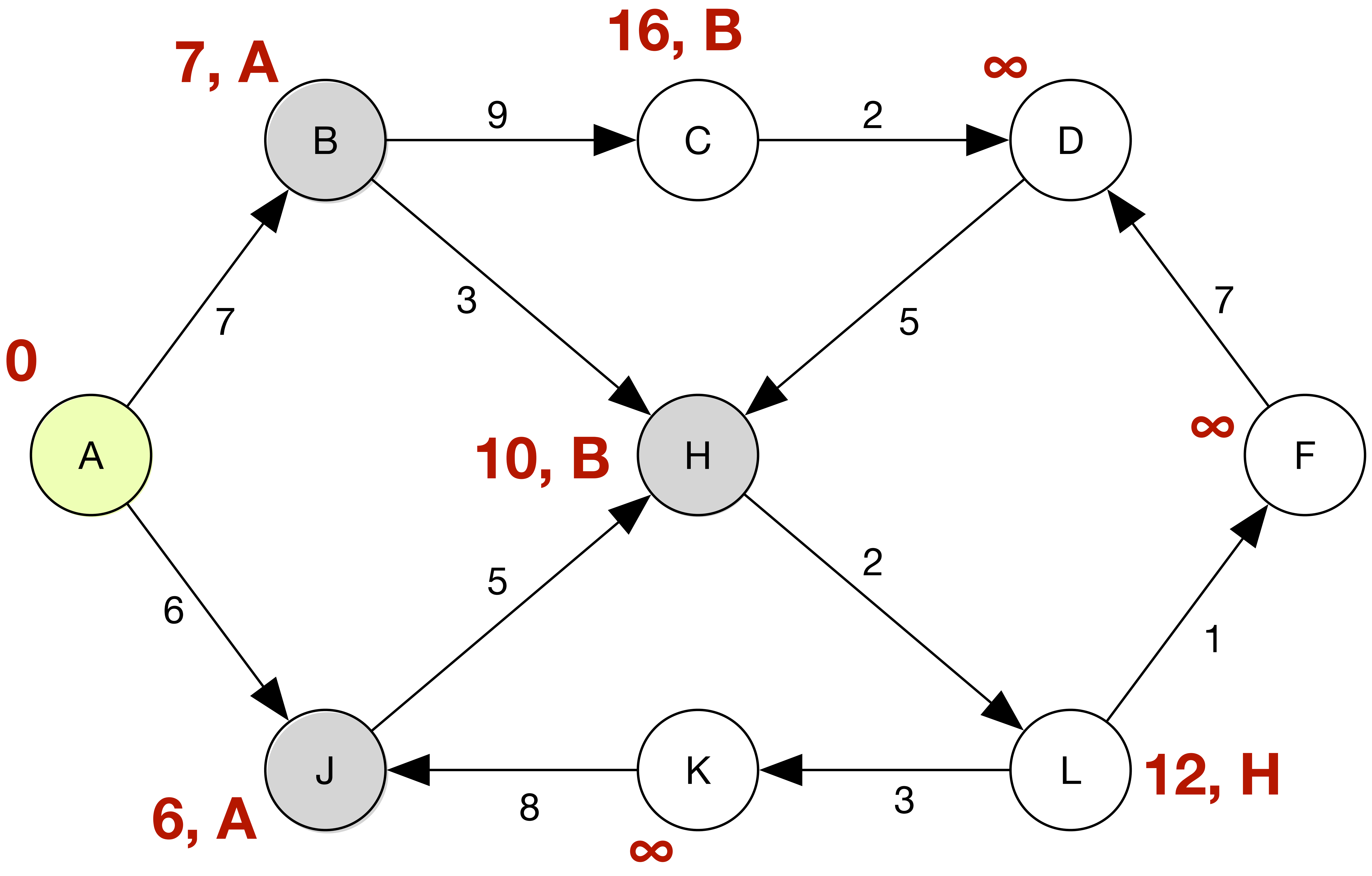
U = {C, D,
F, H, K, L}



Dijkstra

Unvisited set

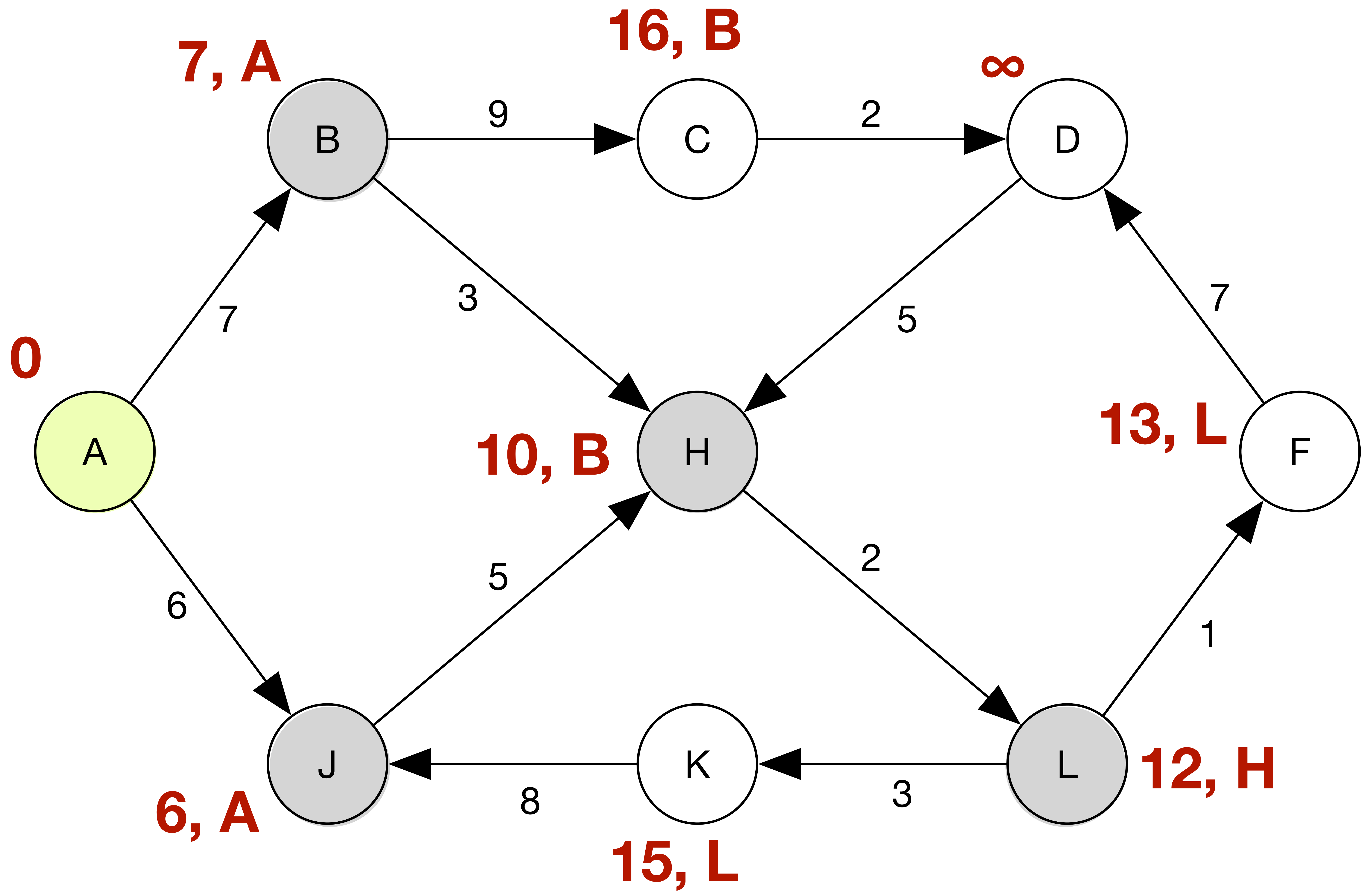
U = {C, D,
F, K, L}



Dijkstra

Unvisited set

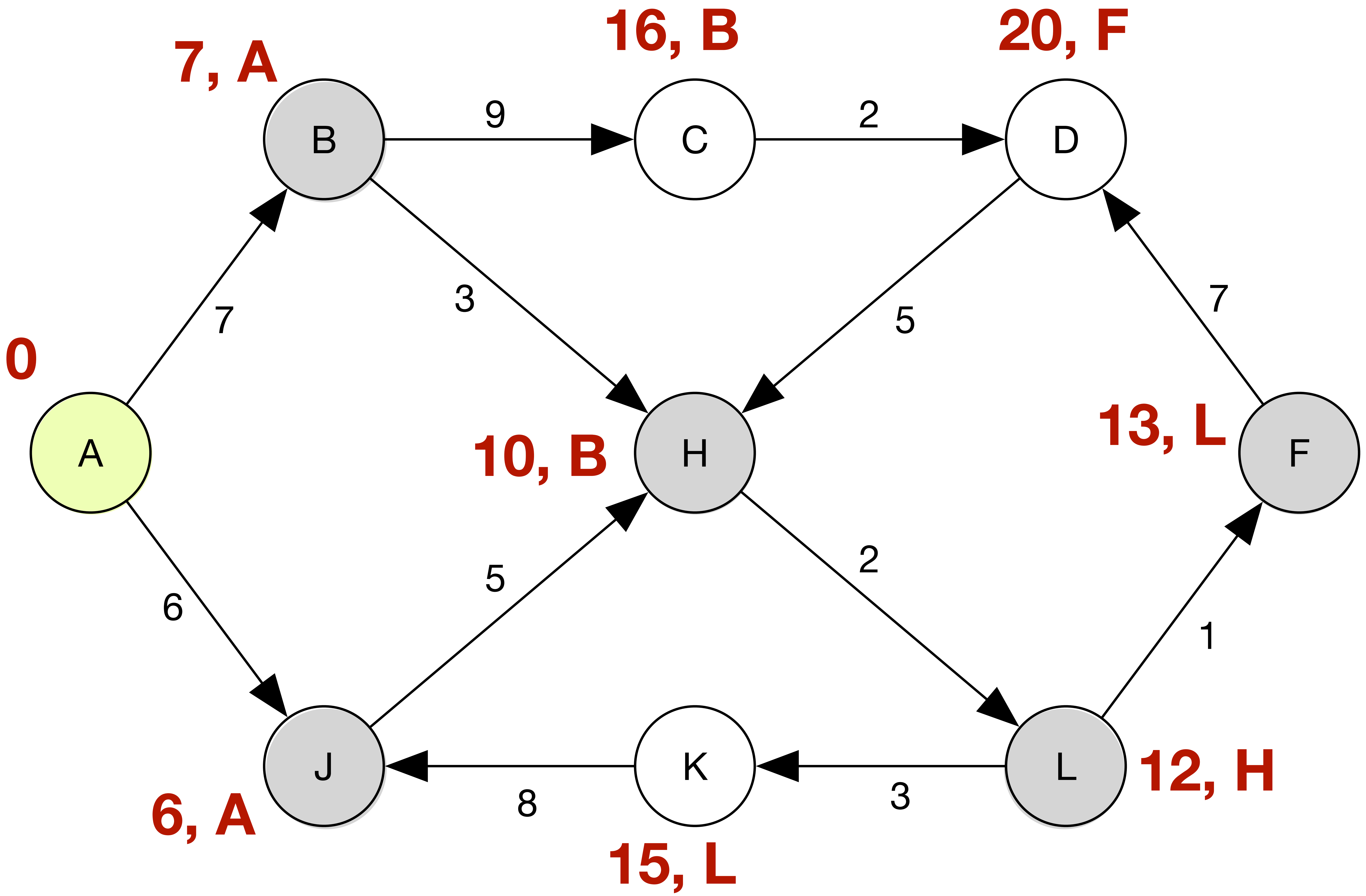
U = {C, D,
F, K}



Dijkstra

Unvisited set

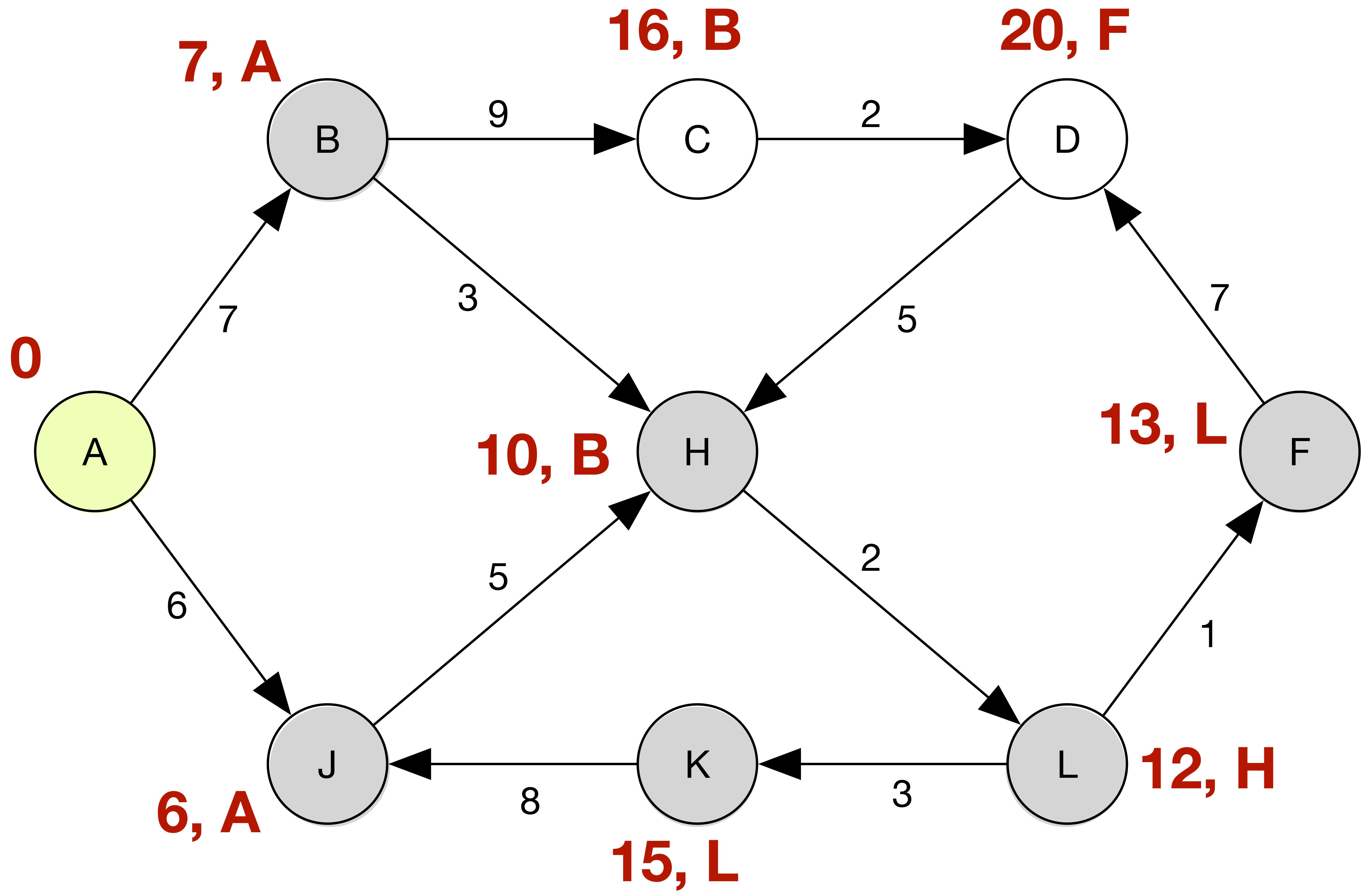
U = {C, D, K}



Dijkstra

Unvisited set

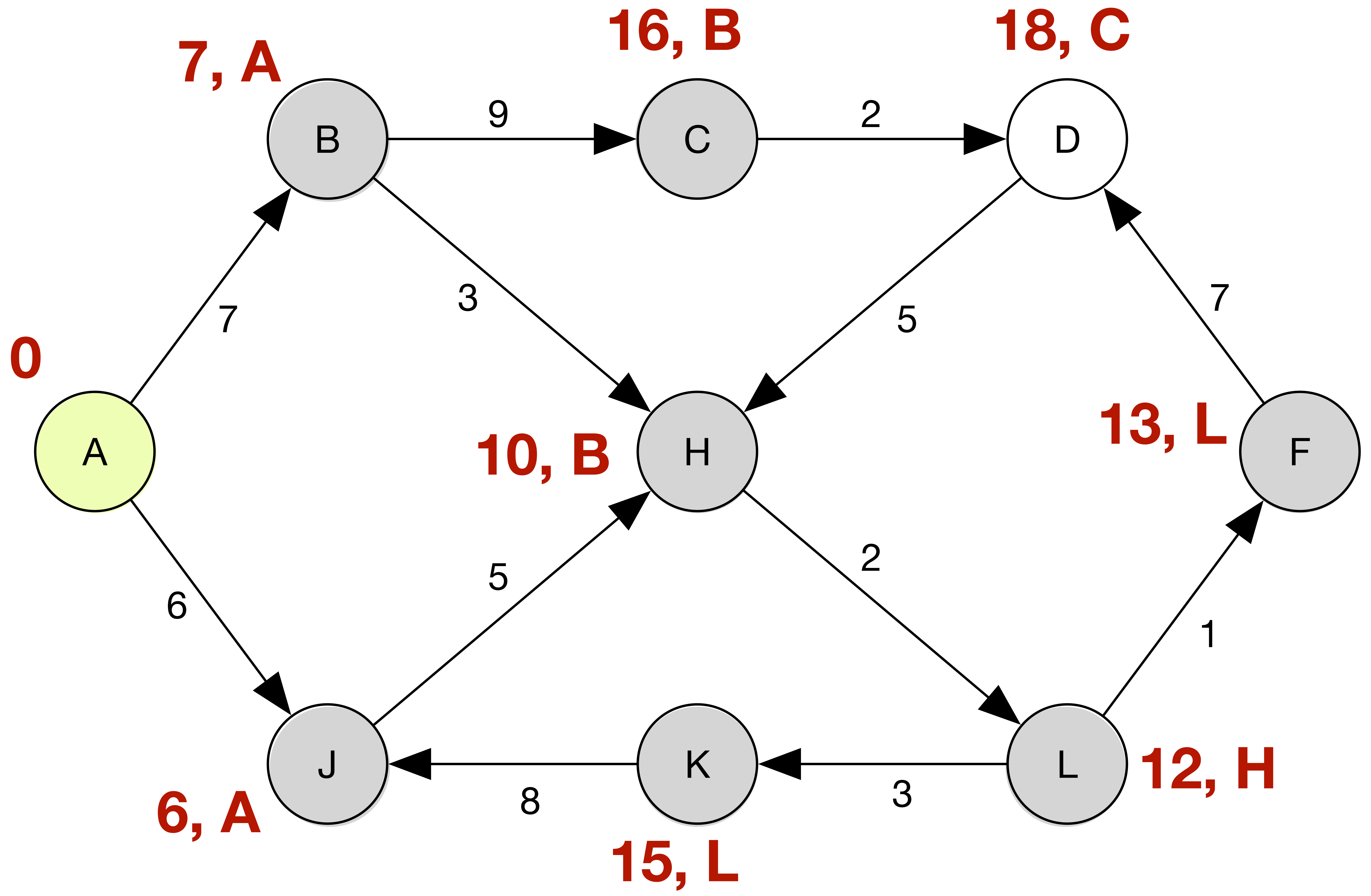
U = {C, D}



Dijkstra

Unvisited set

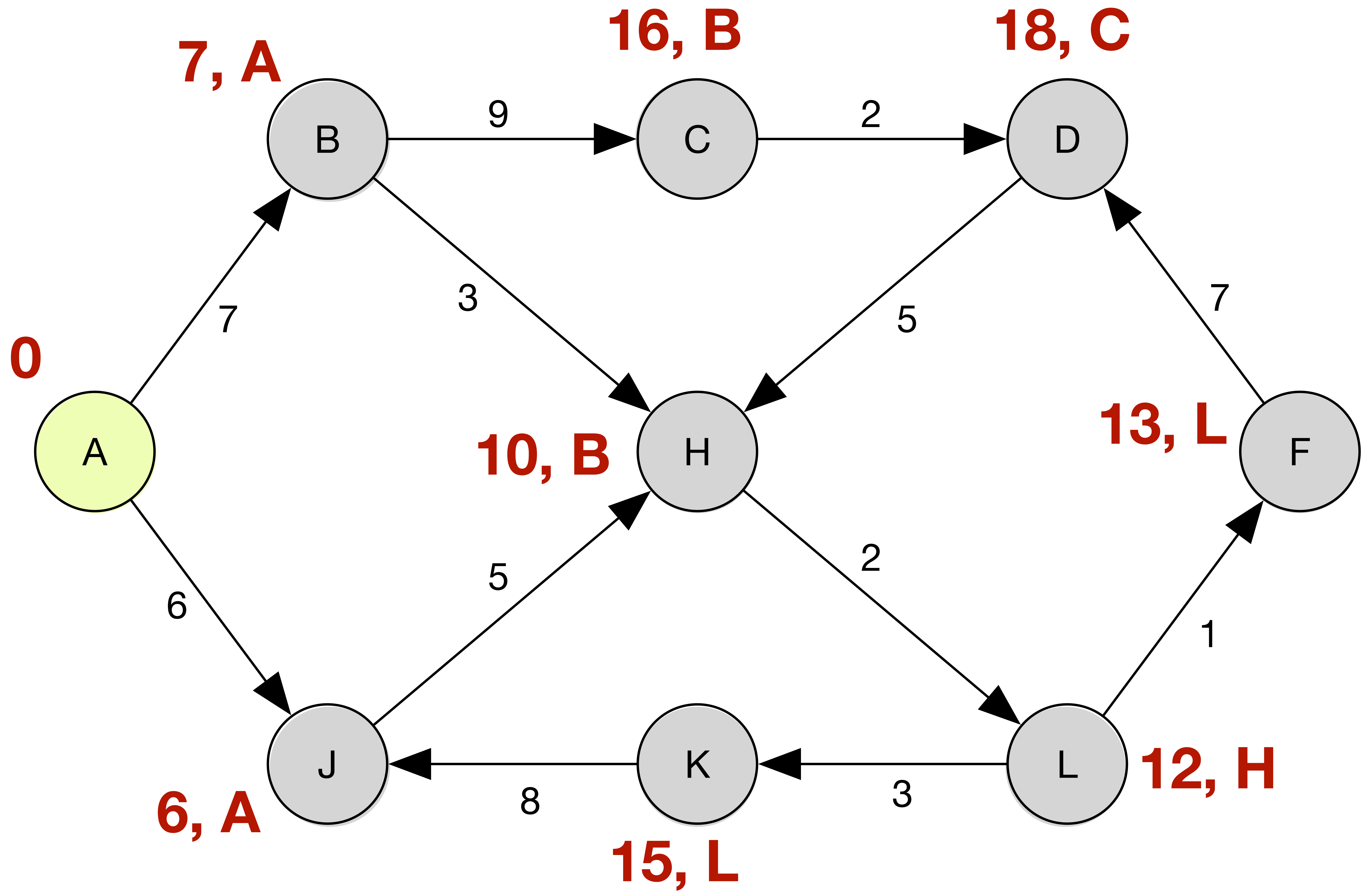
U = {D}



Dijkstra

Unvisited set

U = {}

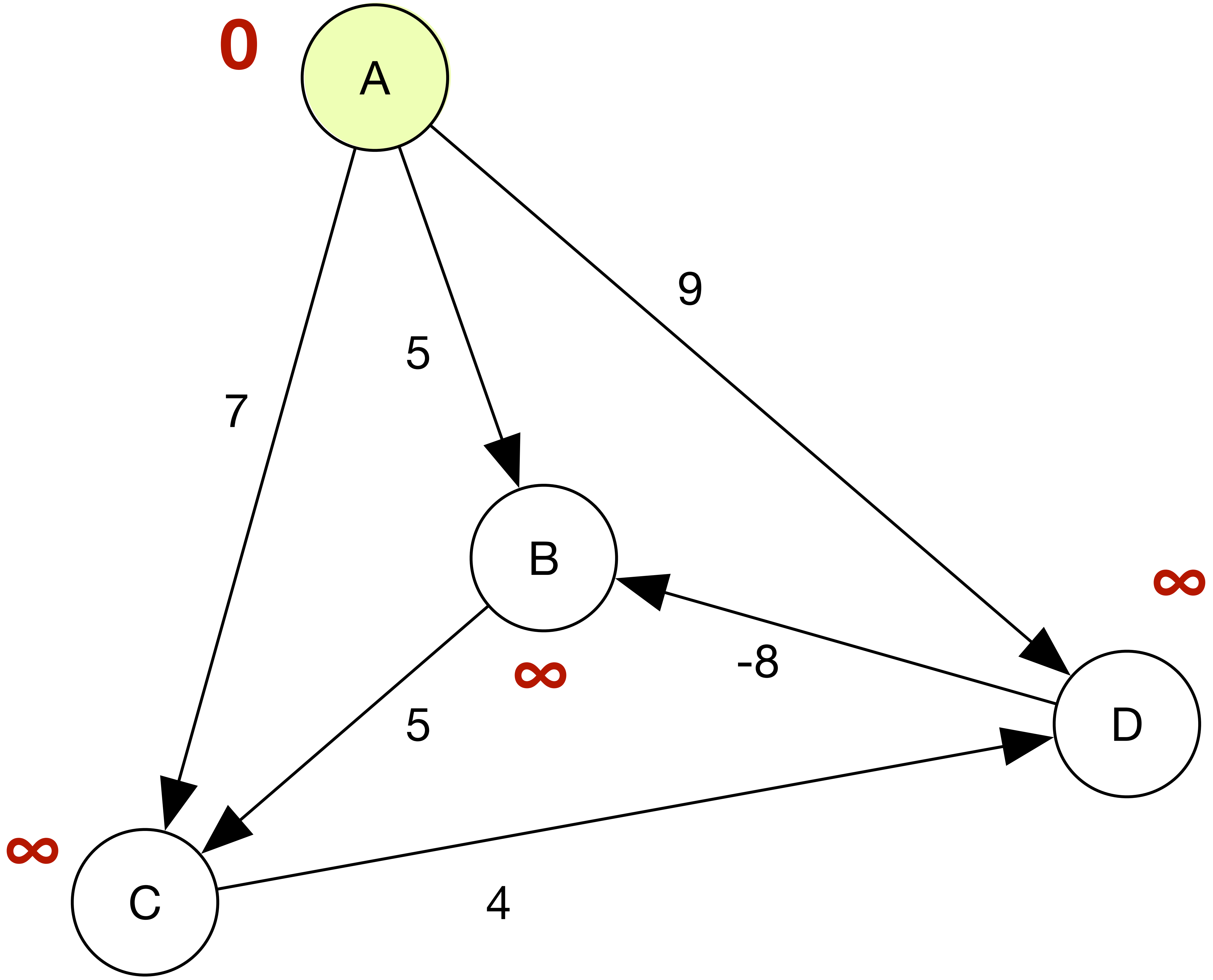


Dijkstra's algorithm

Why doesn't Dijkstra's algorithm work with negative weights?

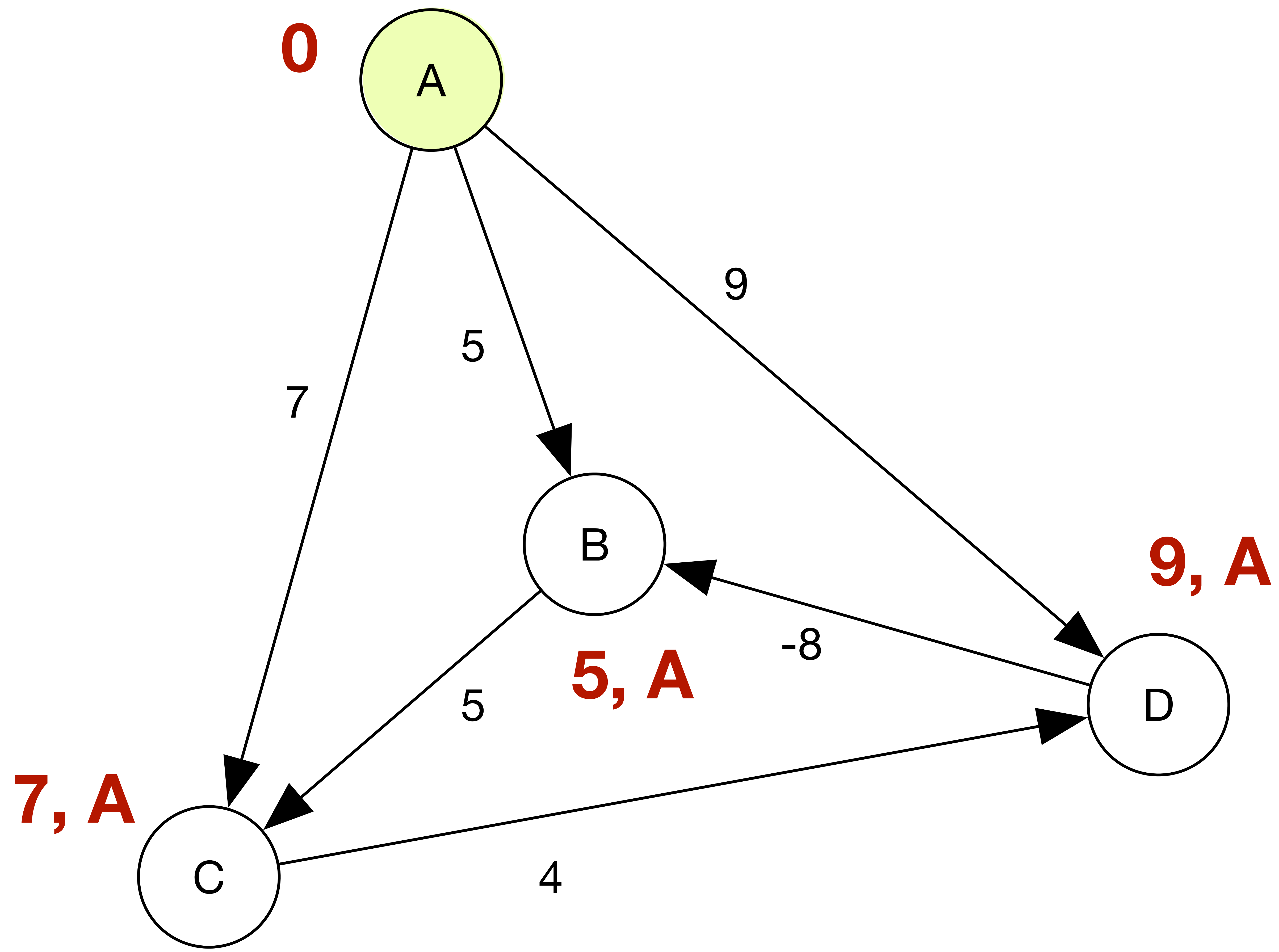
Dijkstra's algorithm

U = {A, B, C, D}



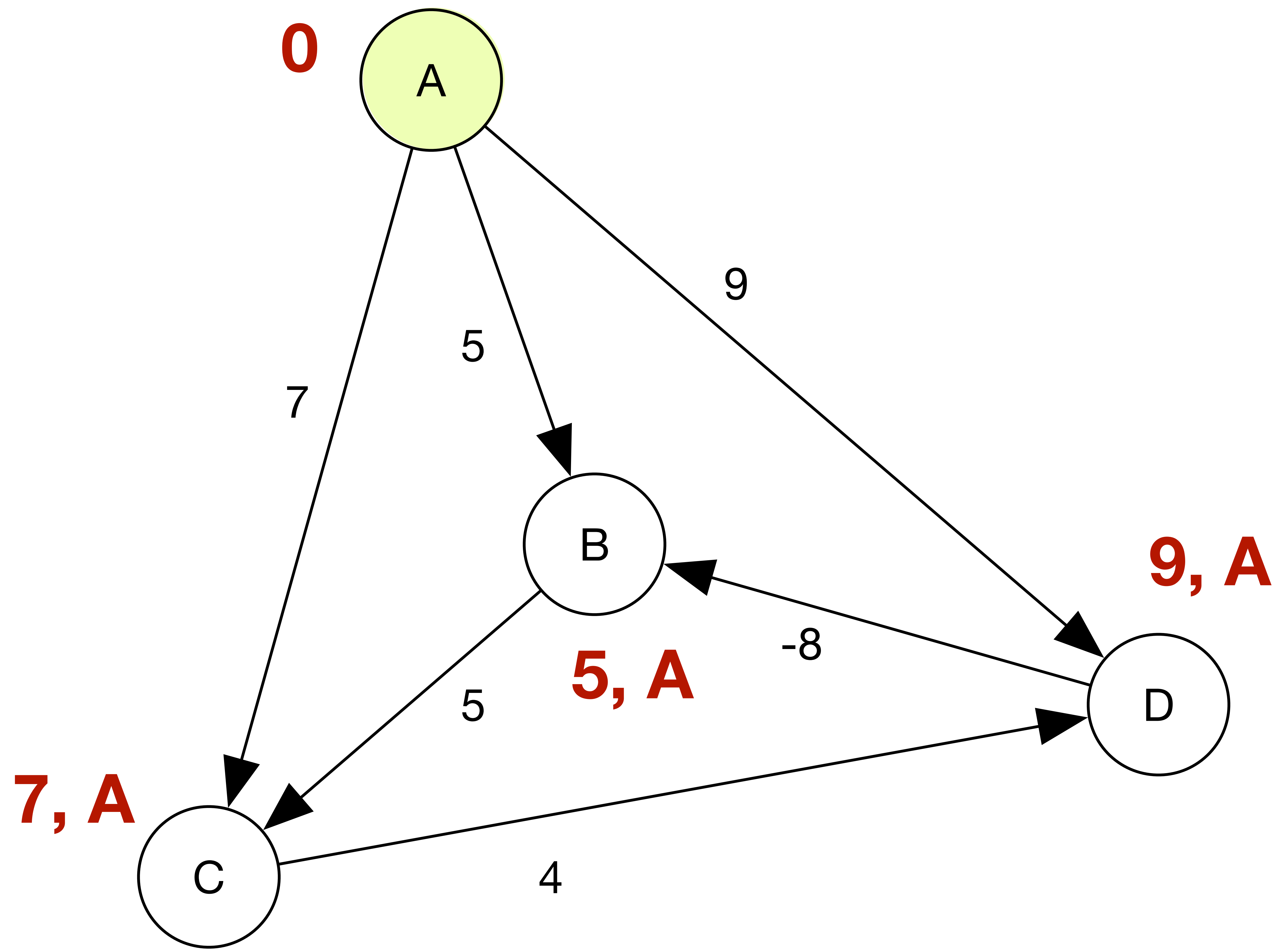
Dijkstra's algorithm

U = {B, C, D}



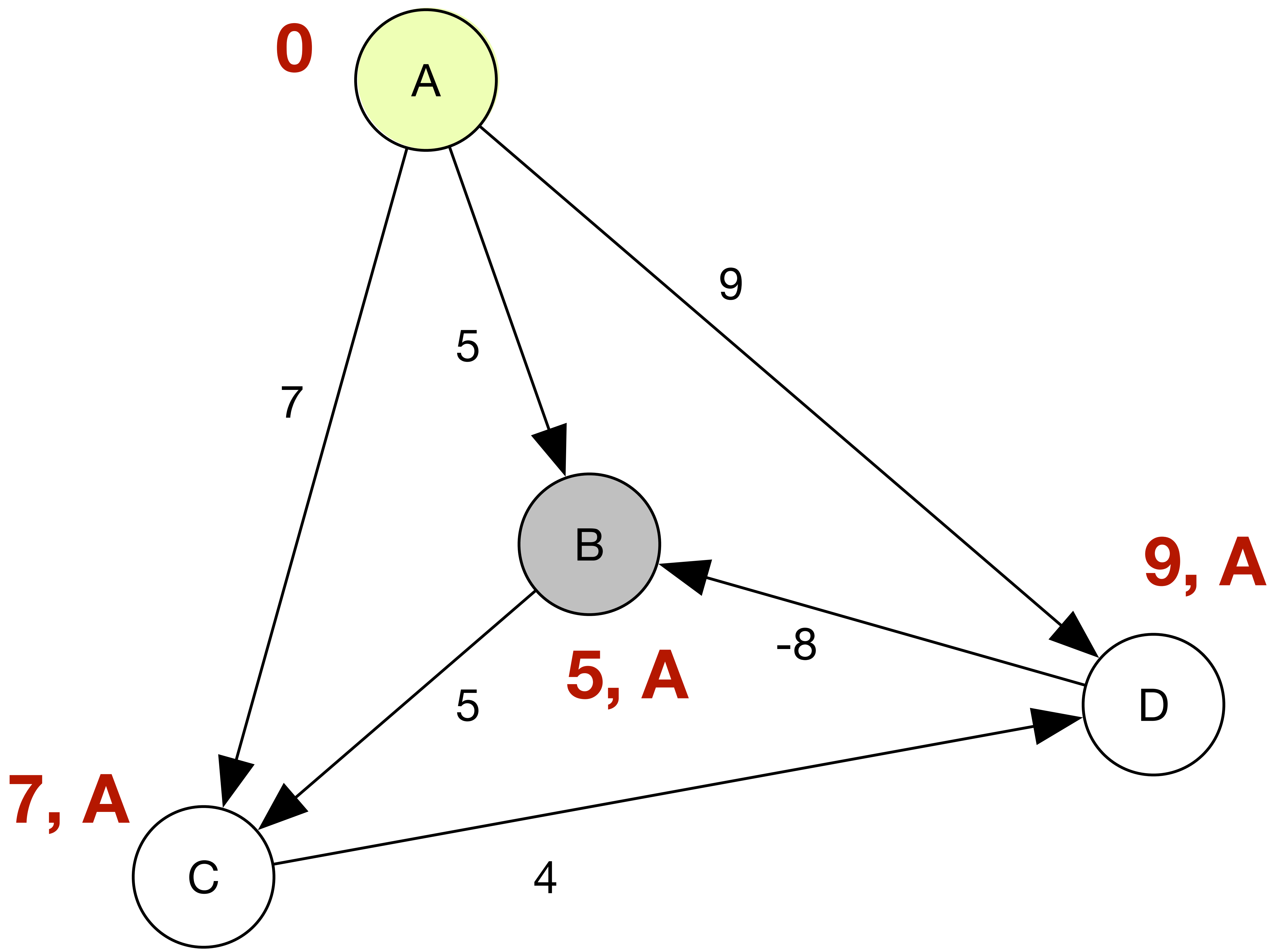
Dijkstra's algorithm

U = {B, C, D}



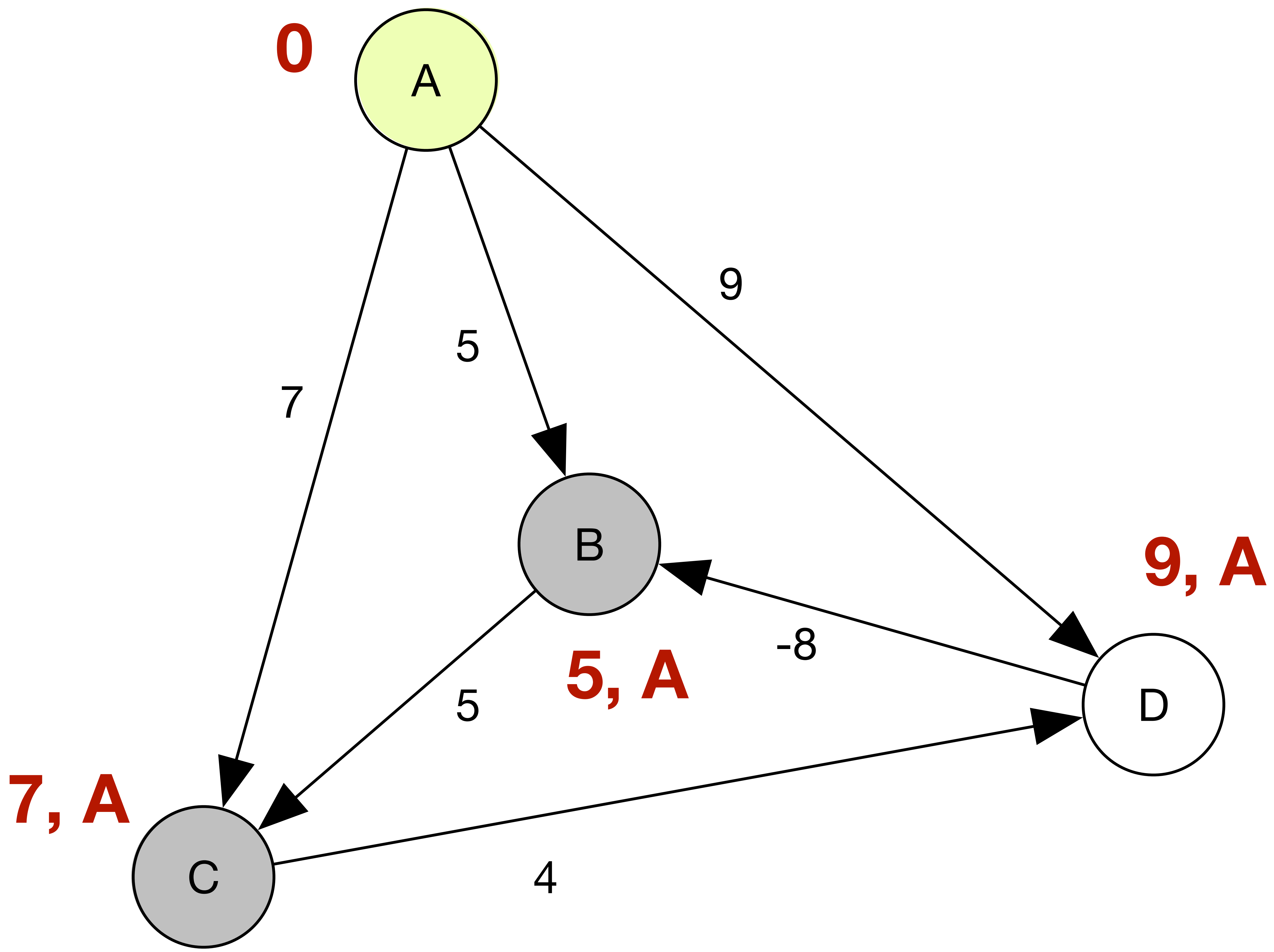
Dijkstra's algorithm

U = {C, D}



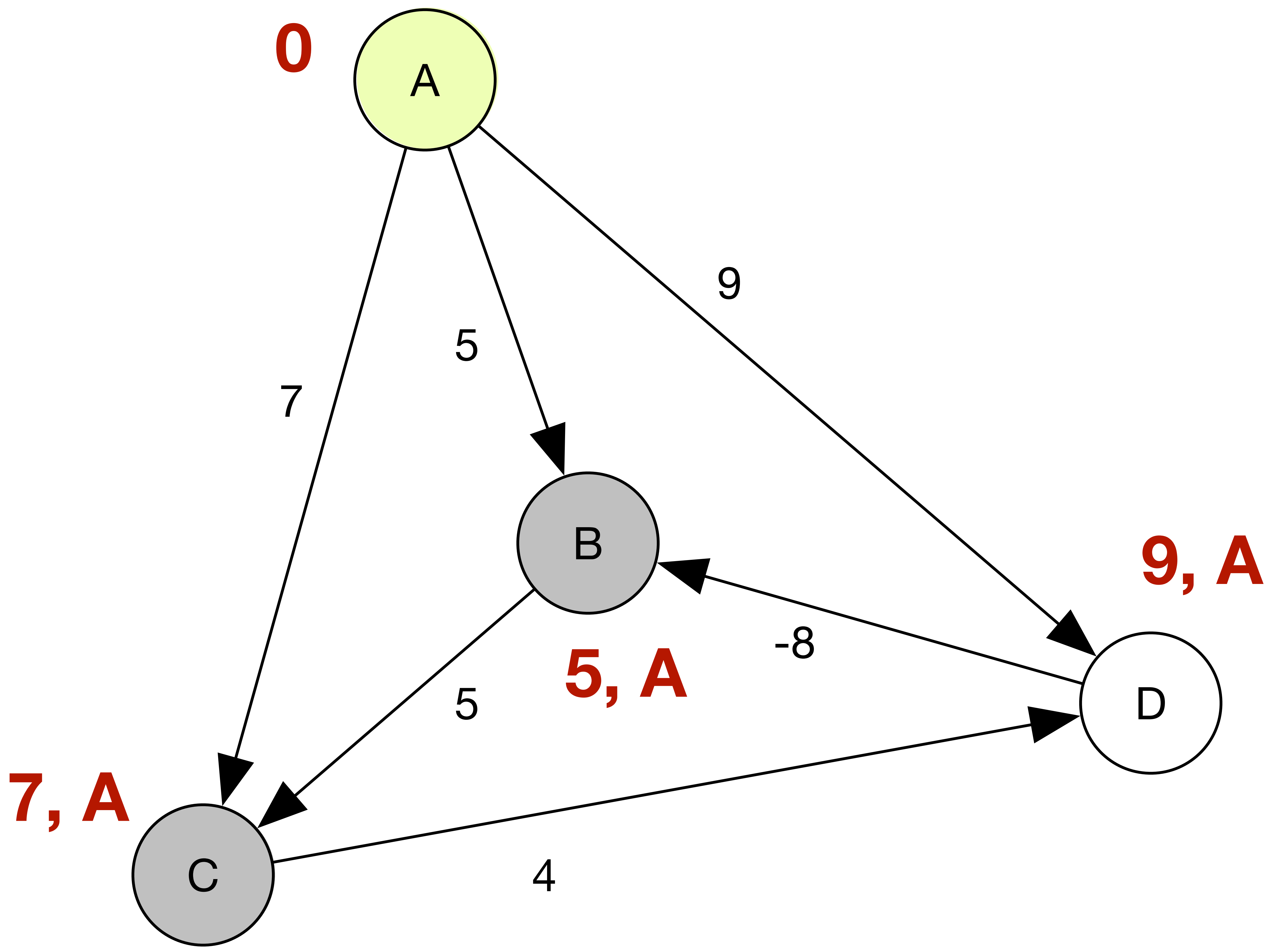
Dijkstra's algorithm

U = {D}



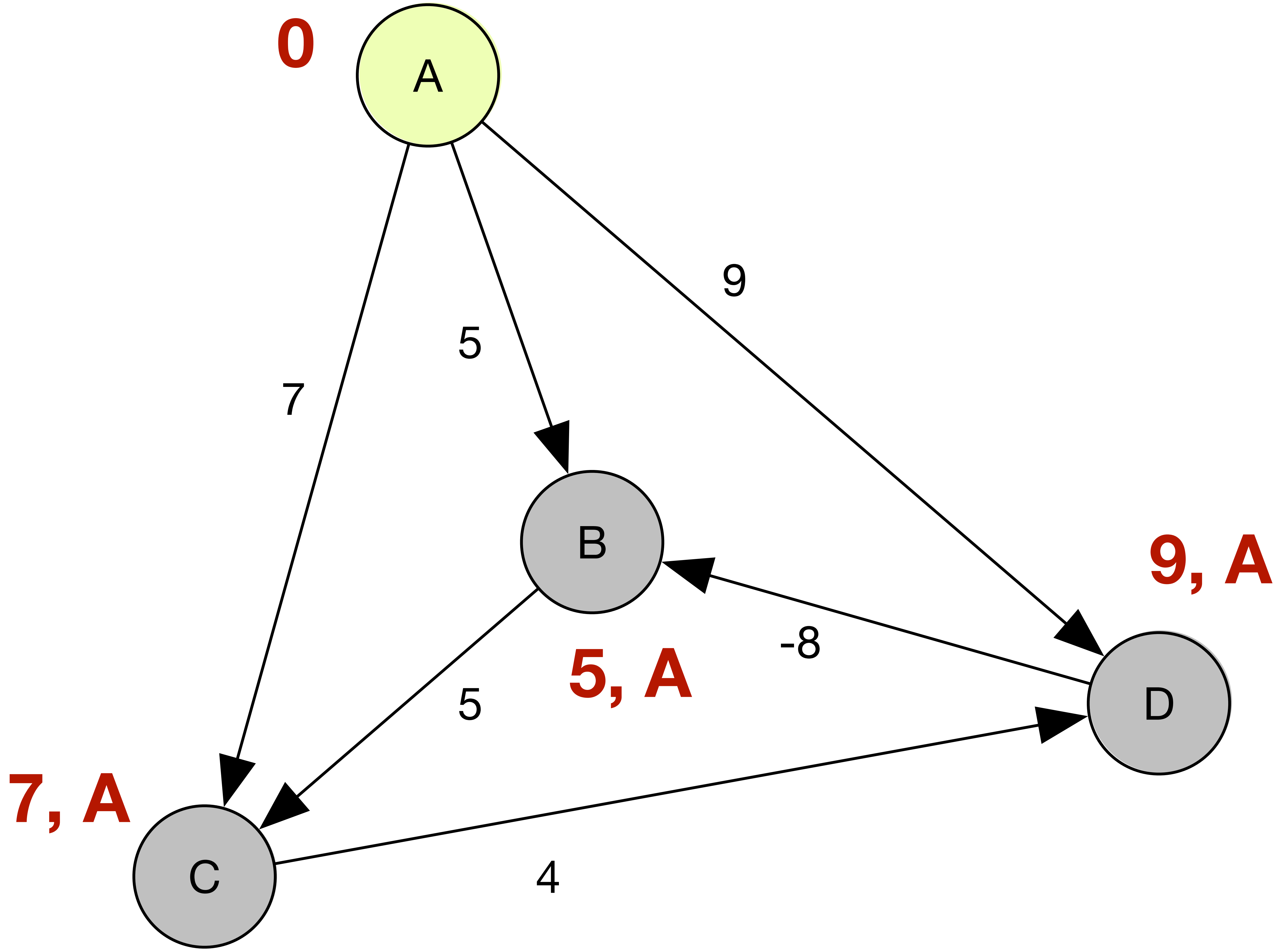
Dijkstra's algorithm

U = {}



Dijkstra's algorithm

U = {}

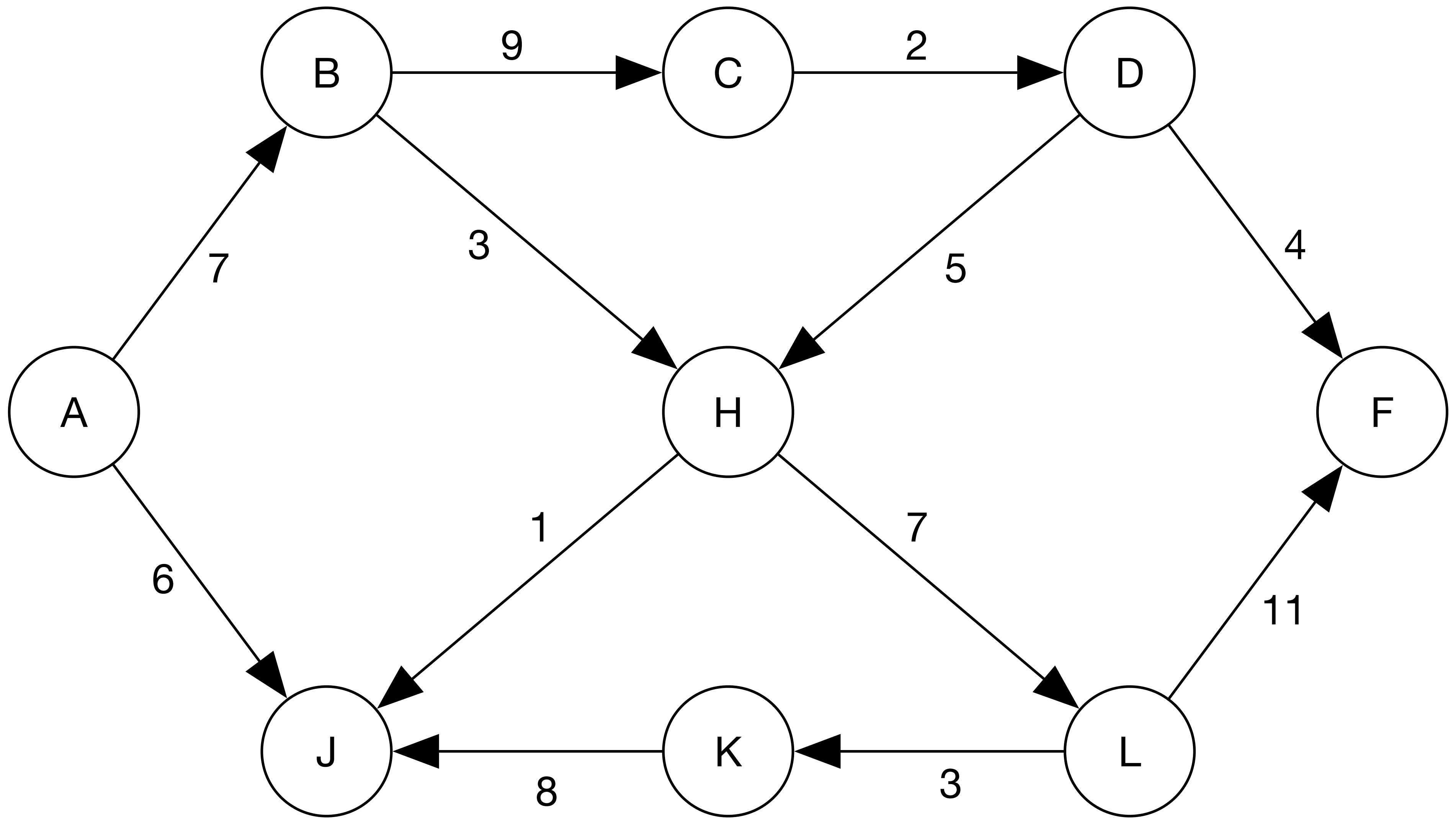


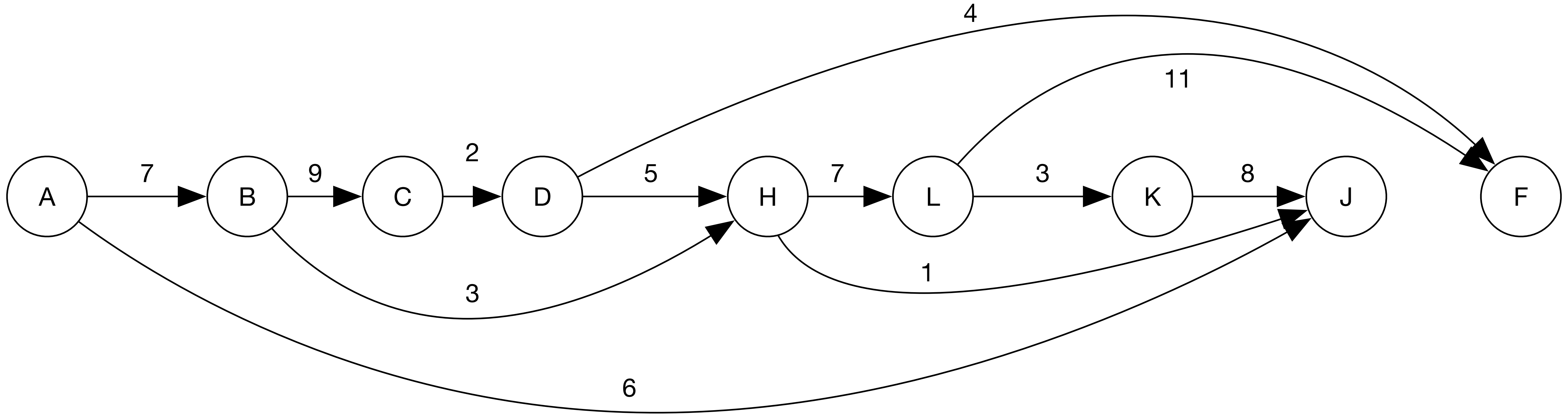
Dijkstra's algorithm: Can we do better?

Worst case complexity (using min priority queue)

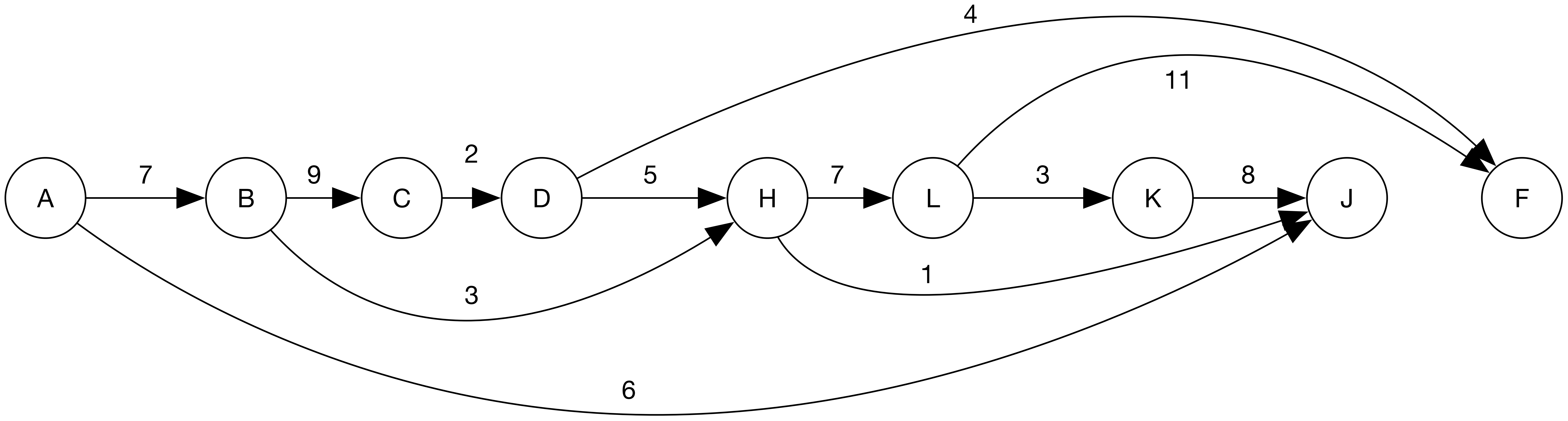
$$\mathcal{O}((|V| + |E|) \log|V|)$$

Dijkstra's algorithm: Can we do better?

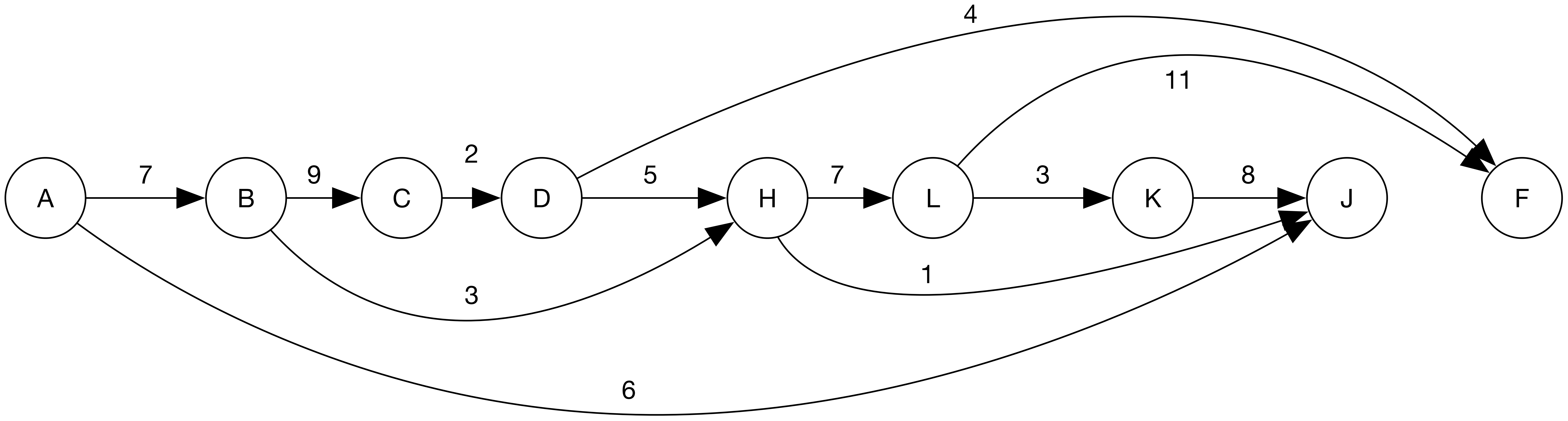




Dijkstra's algorithm: Can we do better?



Dijkstra's algorithm: Can we do better?



Dijkstra's algorithm

Worst case complexity directed graph with cycles (using min priority queue)

$$\mathcal{O}((|V| + |E|)\ \log|V|)$$

Worst case complexity directed acyclic graph (using topological sort)

$$\mathcal{O}(|V| + |E|)$$

Dijkstra's algorithm

Greedy approach. Grab the best answer so far; never backtrack.

Dynamic programming. Save partial solutions along the way and reconstruct complete solutions from the partial solutions.

Dijkstra's algorithm

Greedy approach. Grab the best answer so far; never backtrack.

Dynamic programming. Save partial solutions along the way and reconstruct complete solutions from the partial solutions.

What to do about negative weights?

Dijkstra's algorithm

Greedy approach. Grab the best answer so far; never backtrack.

Dynamic programming. Save partial solutions along the way and reconstruct complete solutions from the partial solutions.

What to do about negative weights? Bellman-Ford algorithm.