Lecture 7: Optimal Discrete Planning
MAE 345/549

Anirudha Majumdar

Princeton

Sept. 27, 2022

PRINCETON
UNIVERSITY

Mechanical and
Aerospace
Engineering

Previous lecture

- Previous lecture:
  - Discussed discretization
  - Graph-search algorithms for planning in discrete spaces
    - Breadth First Search (BFS) and Depth First Search (DFS)
- Two considerations with motion planning:
  - Feasibility (achieved by BFS and DFS)
  - Optimality (today)

Optimal planning in discrete spaces

Algorithm 1: General Forward Search

```
Q.Insert(A) and mark A as visited;
while Q not empty do
    x ← Q.GetVertex() ;
    if x ∈ x_G then
        // x_G is goal set, e.g., {B} ;
        Return SUCCESS;
    end
    for all neighbors x' of x do
        if x' not visited then
            Parent(x') ← x;
            Mark x' as visited ;
            Q.Insert(x') ;
        else
            Resolve duplicate x' ;
        end
    end
end
Return FAILURE ;
```

This will be implemented differently

This is new

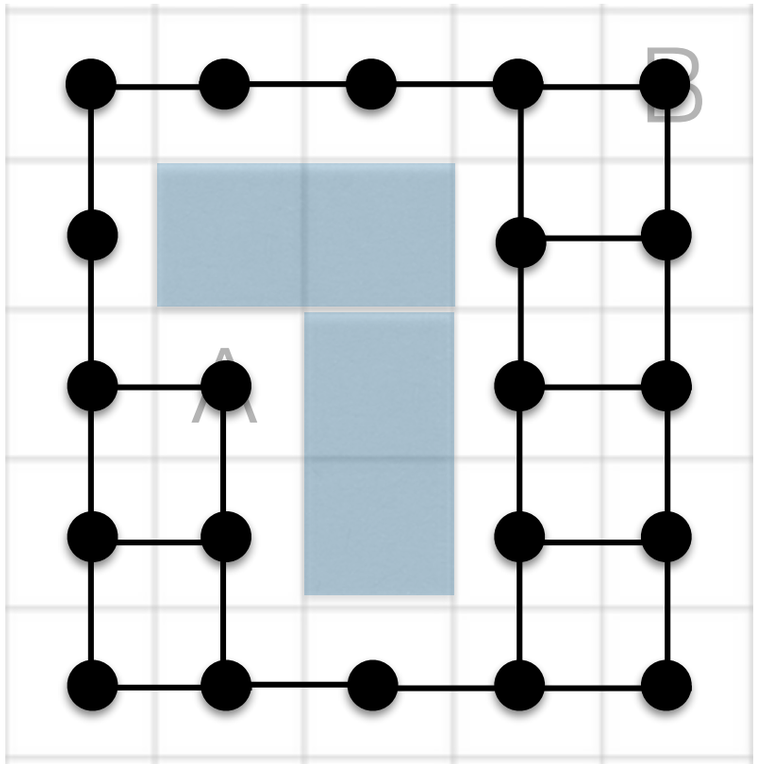
Cost of a plan

- Suppose with every edge e, we associate:

  $l(e)$: Cost (or "loss") associated with traversing edge e

- Total cost of a plan: sum of costs along path from A to B

- Our goal: find path that minimizes total cost



Vertices: unvisited, alive, and dead

- For today's algorithms, it is helpful to think about three kinds of vertices:
  - "Unvisited": Vertices that we have not yet explored.
  - "Alive": Vertices stored in Q. These are vertices we have encountered, but have not taken out from Q yet.
  - "Dead": Vertices that have been visited, and for which every neighbor has also been visited (intuitively, these are vertices that don't have any more to contribute to the search).

Dijkstra's algorithm

- Dijkstra's algorithm ["Planning Algorithms", Ch. 2.2.2]: Implement Q.GetVertex() based on an estimated cost to get to a vertex from the start vertex A

- For each vertex x, define:
  - $C^\star(x)$: optimal cost-to-come from A to x

- Priority queue Q will be sorted according to an *estimate* $C(x)$ of the optimal cost-to-come $C^\star(x)$
  - These estimates will be updated as the algorithm proceeds

Dijkstra's algorithm

- Initially: $C(A) = C^\star(A) = 0$
- Each time a vertex is considered, we compute the estimated cost as:

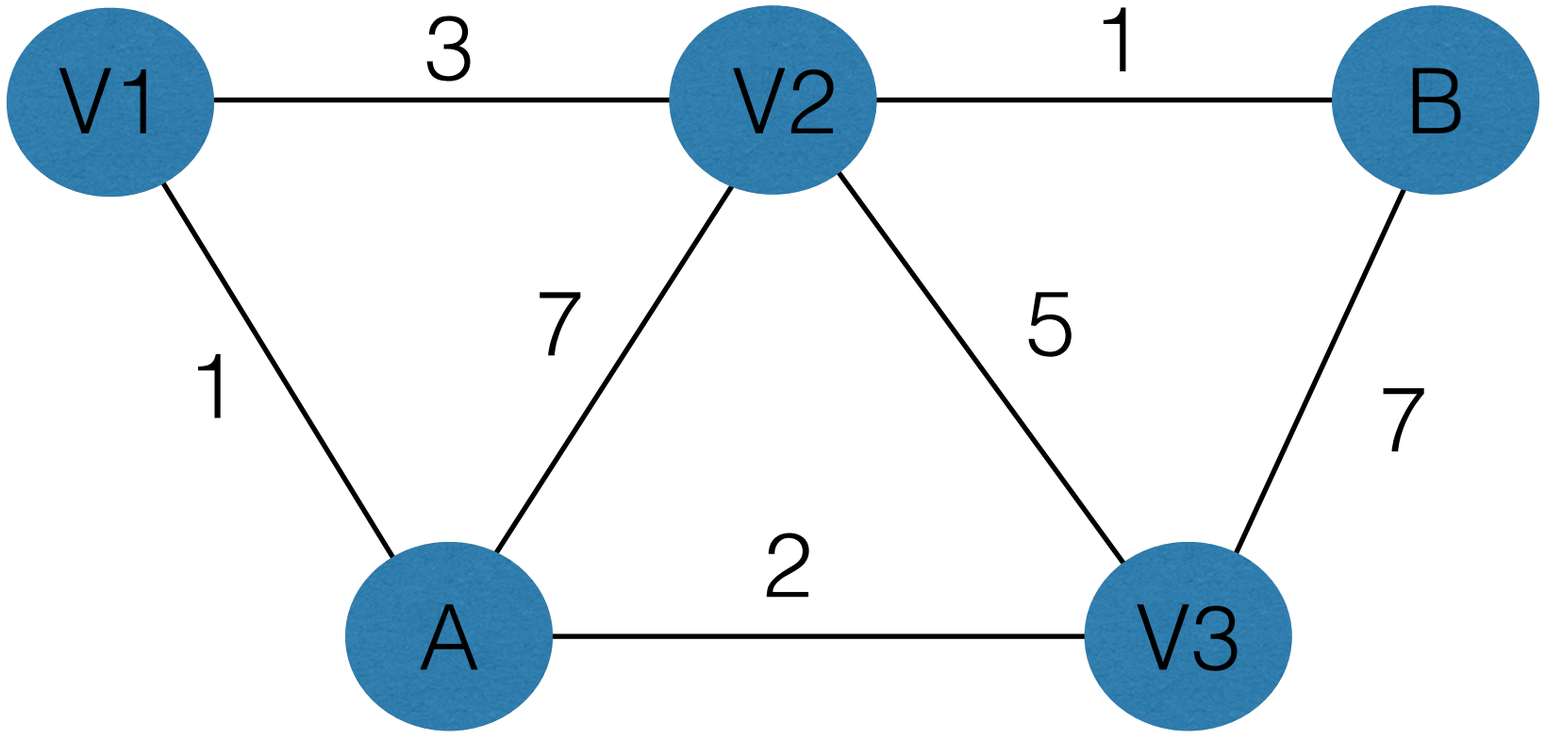
  $C(x') = C(x) + l(e)$, where e is the edge connecting x to x'

- The line "Resolve duplicate x" of Algorithm 1 accounts for the fact that if x' has already been visited, it is possible that the newly discovered path to x' is more efficient.
  - Then $C(x')$ must be lowered and Q must be reordered accordingly.

Dijkstra's algorithm

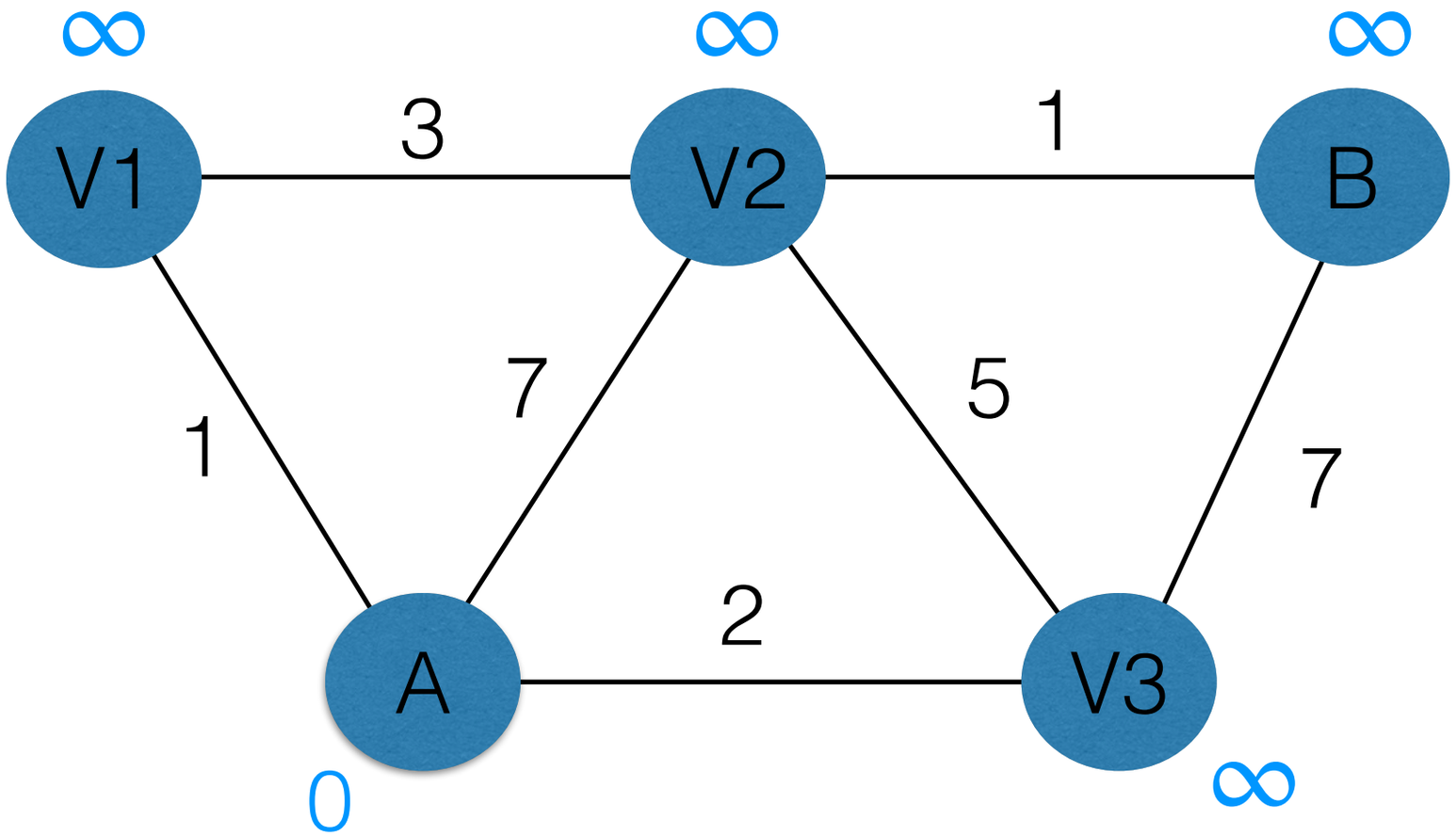
- When does $C(x)$ become $C^{\star}(x)$ for some x?
  - Once x is "dead" and removed from Q using $Q.GetVertex()$, we know that x cannot be reached with a lower cost
  - This can be proved via induction ["Planning Algorithms", Ch. 2.2.2, 2.3.3]

Dijkstra's Algorithm Example



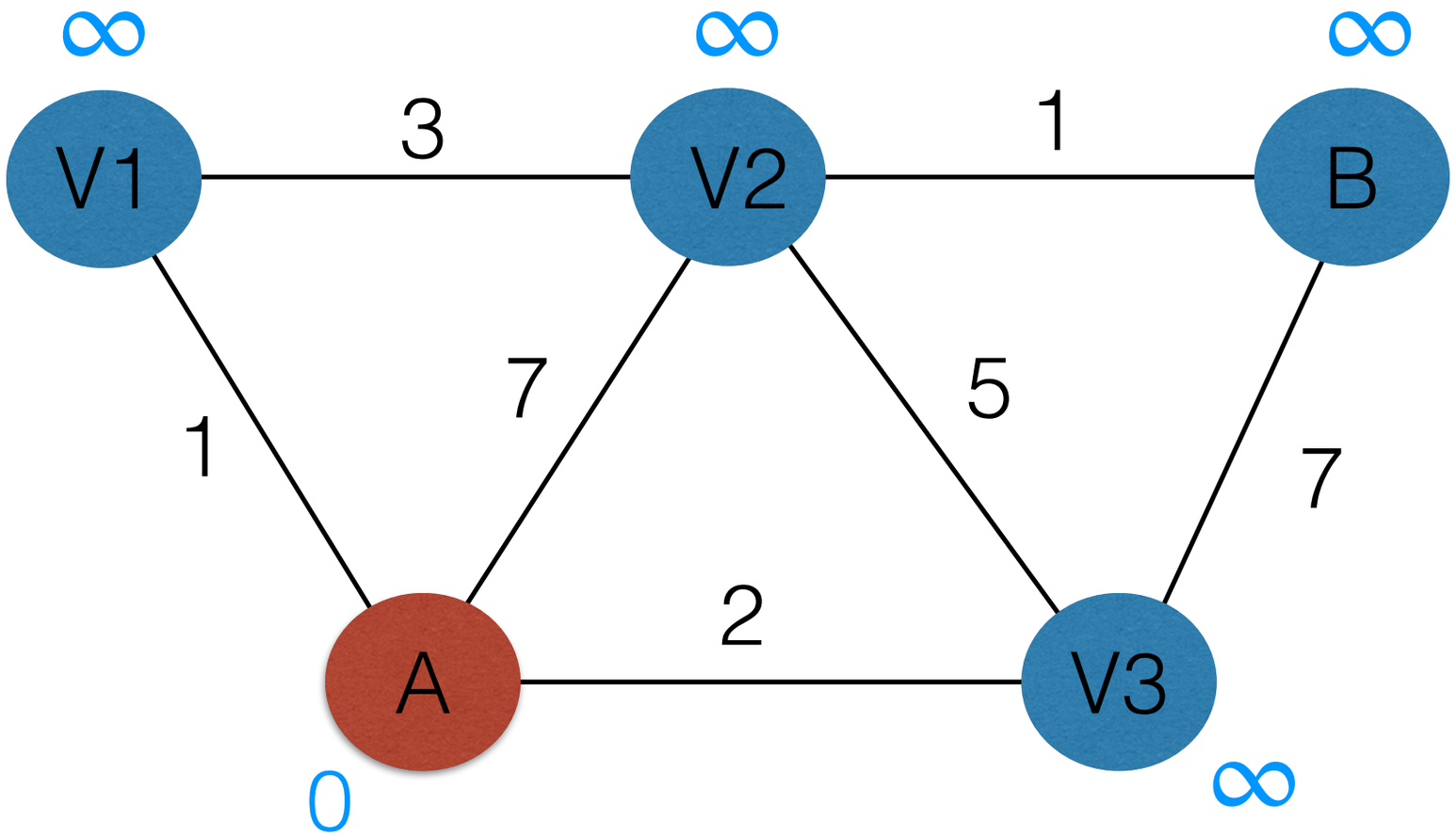
Dijkstra's Algorithm Example

$$Q = \{A\}$$



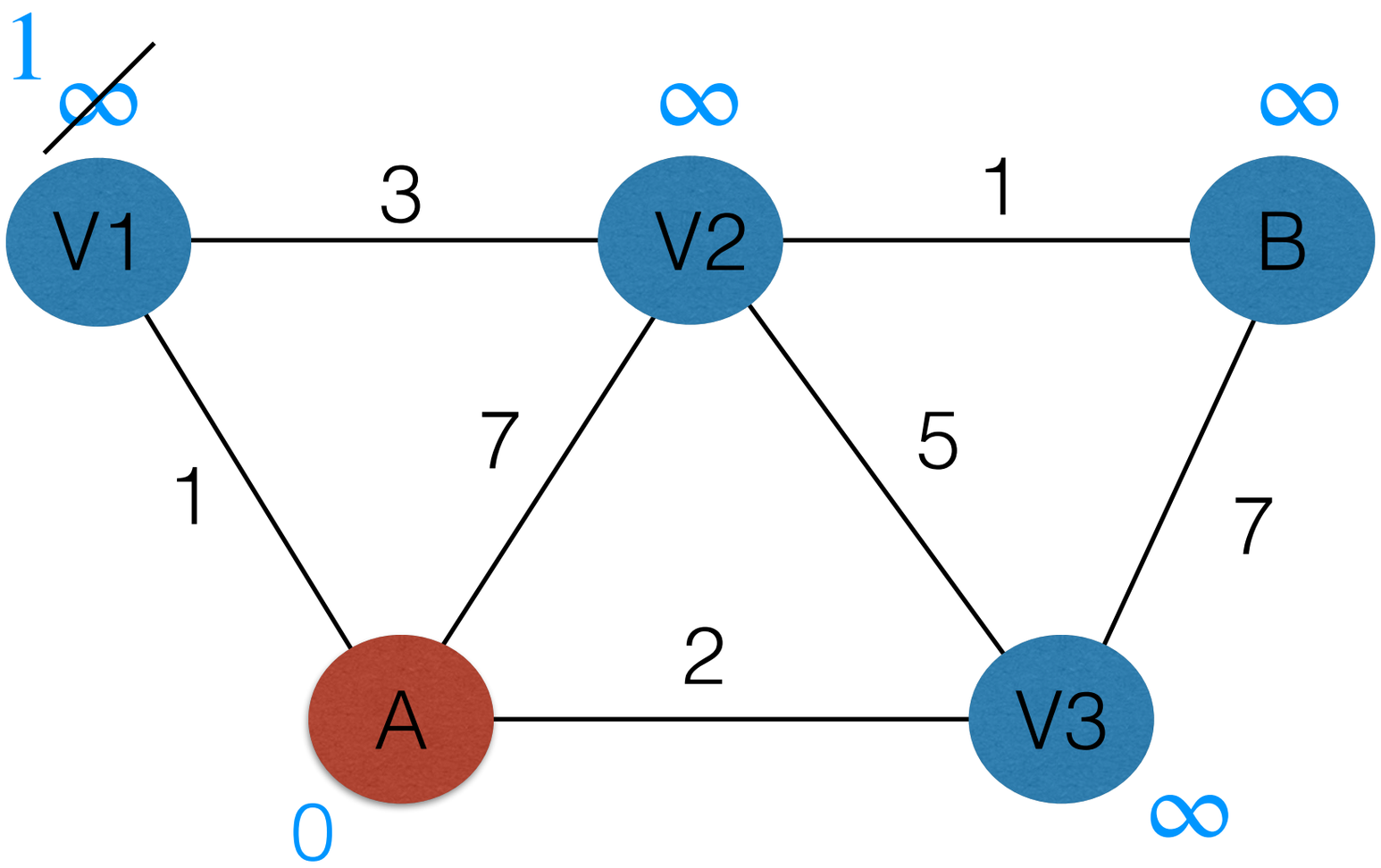
Dijkstra's Algorithm Example

$$Q = \{A\}$$



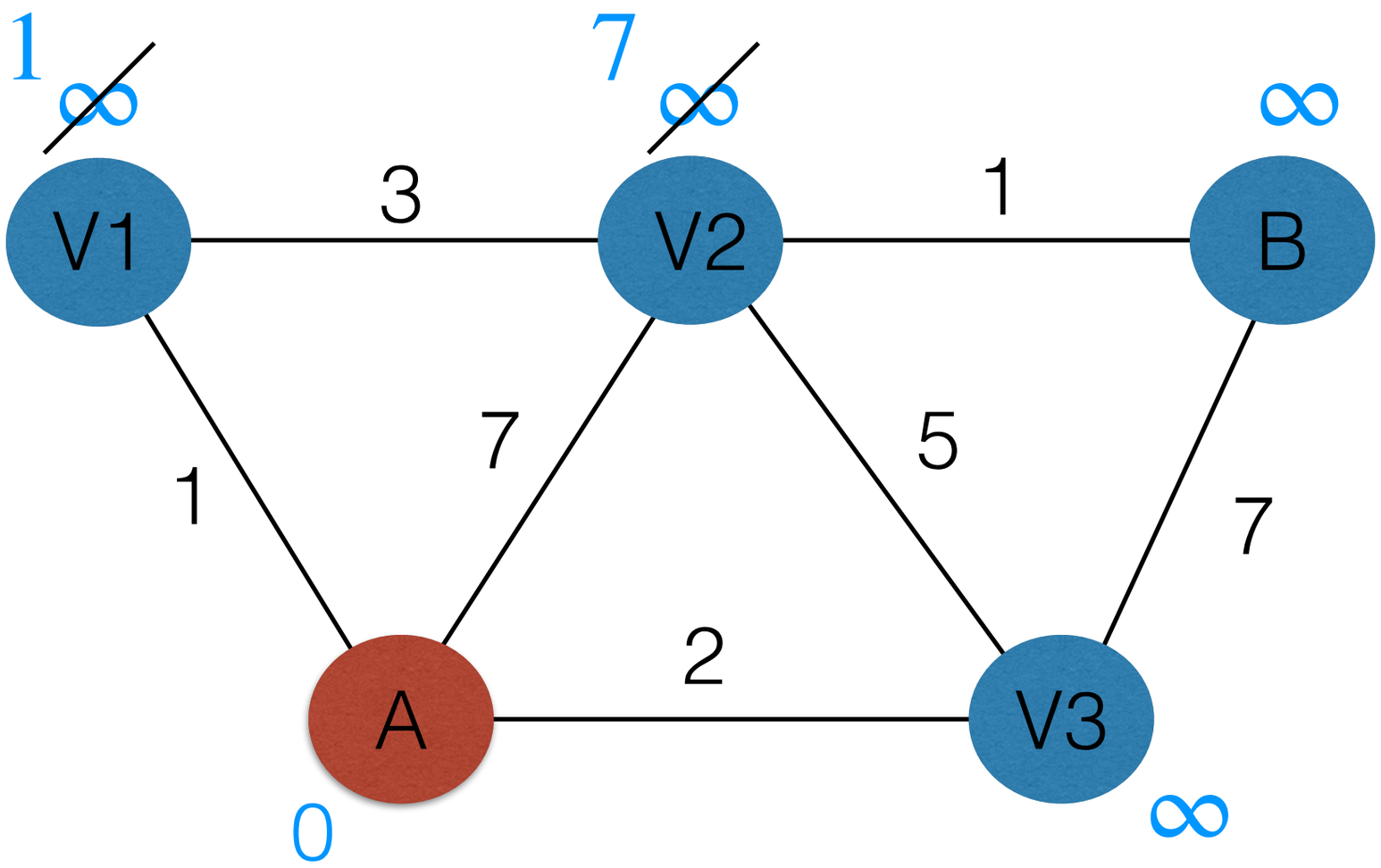
Dijkstra's Algorithm Example

$$Q = \{V1\}$$



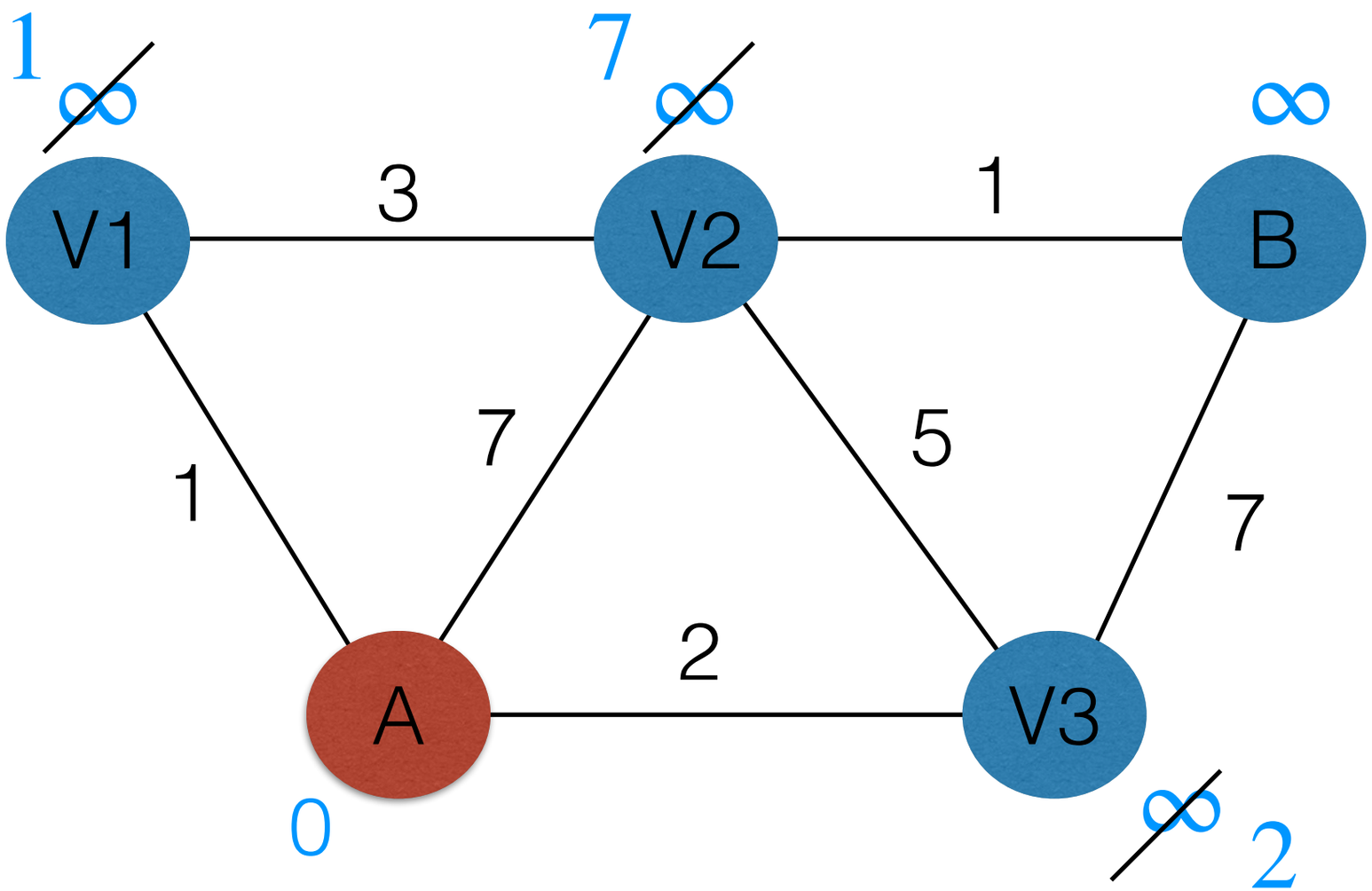
Dijkstra's Algorithm Example

$$Q = \{V1, V2\}$$



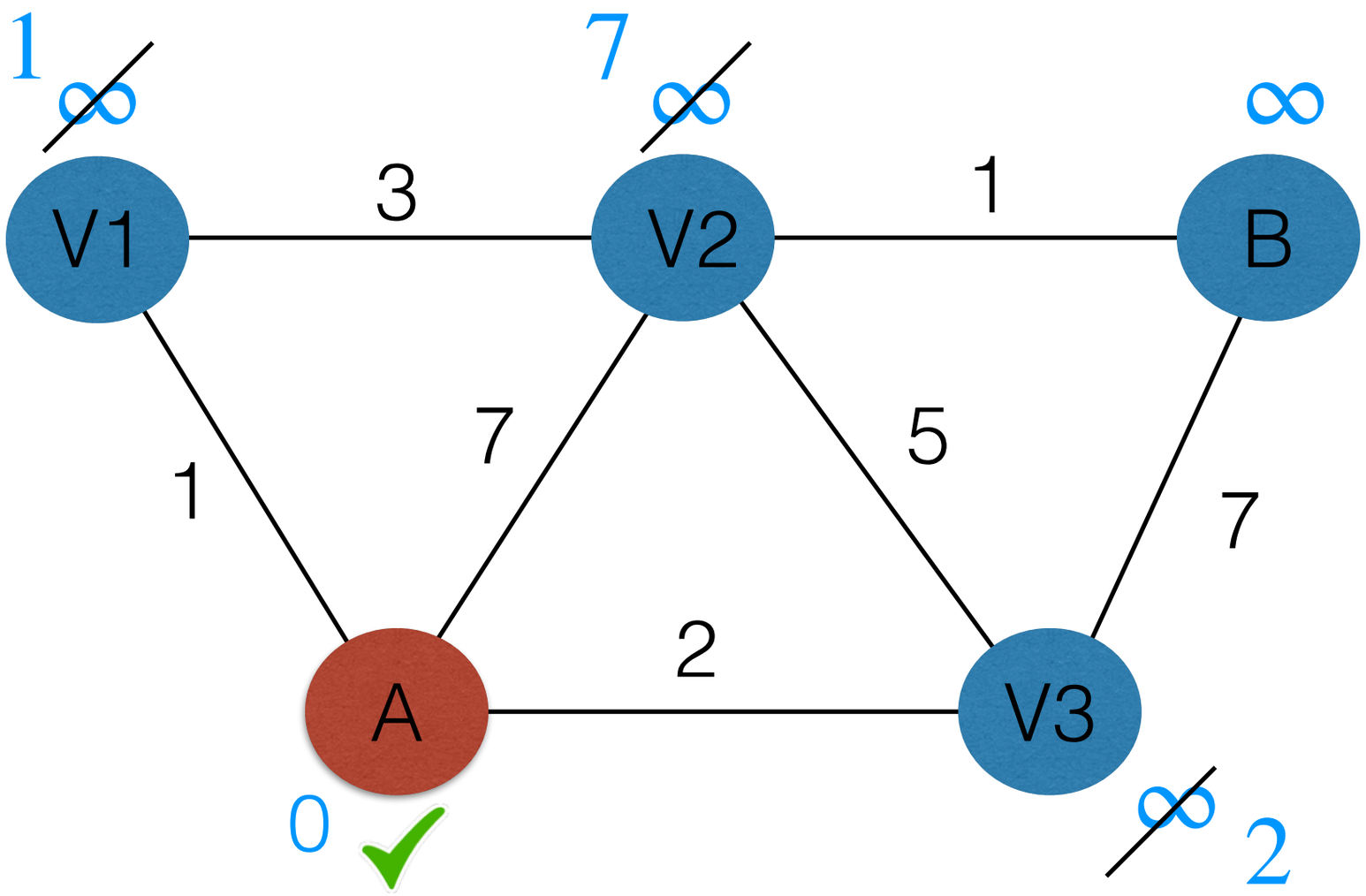
Dijkstra's Algorithm Example

$$Q = \{V1, V2, V3\}$$



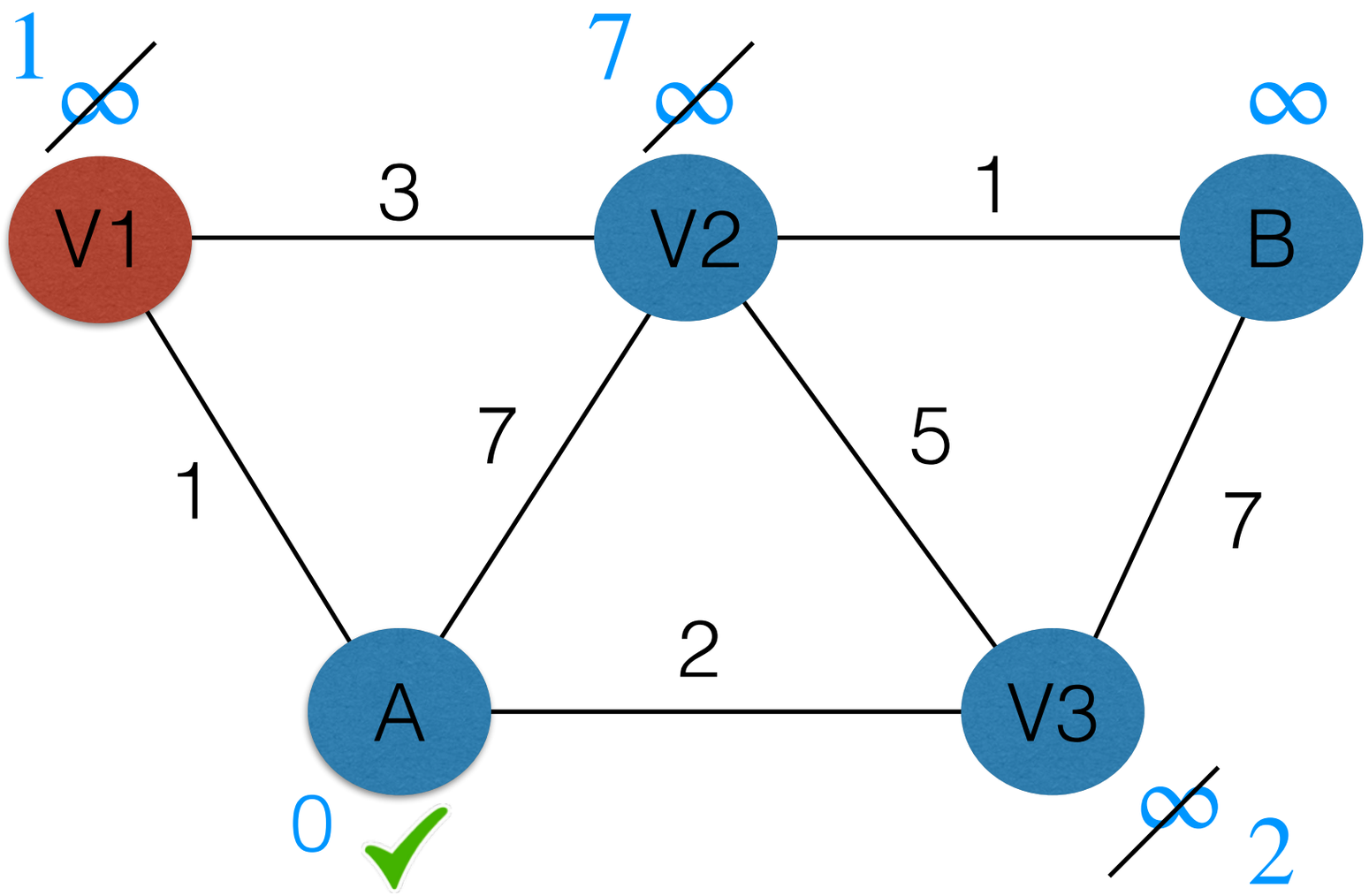
Dijkstra's Algorithm Example

$$Q = \{V1, V2, V3\}$$



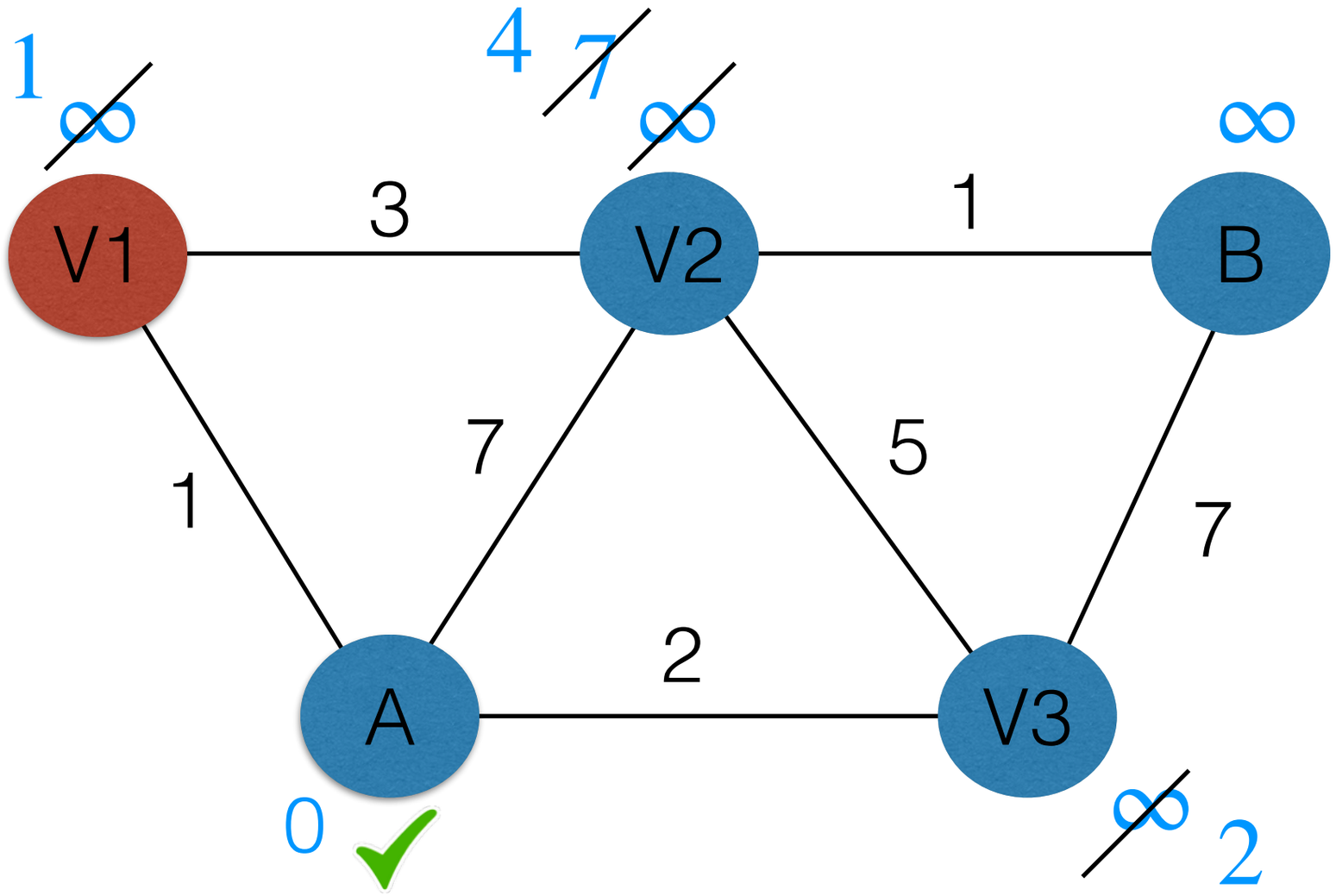
Dijkstra's Algorithm Example

$$Q = \{V2, V3\}$$



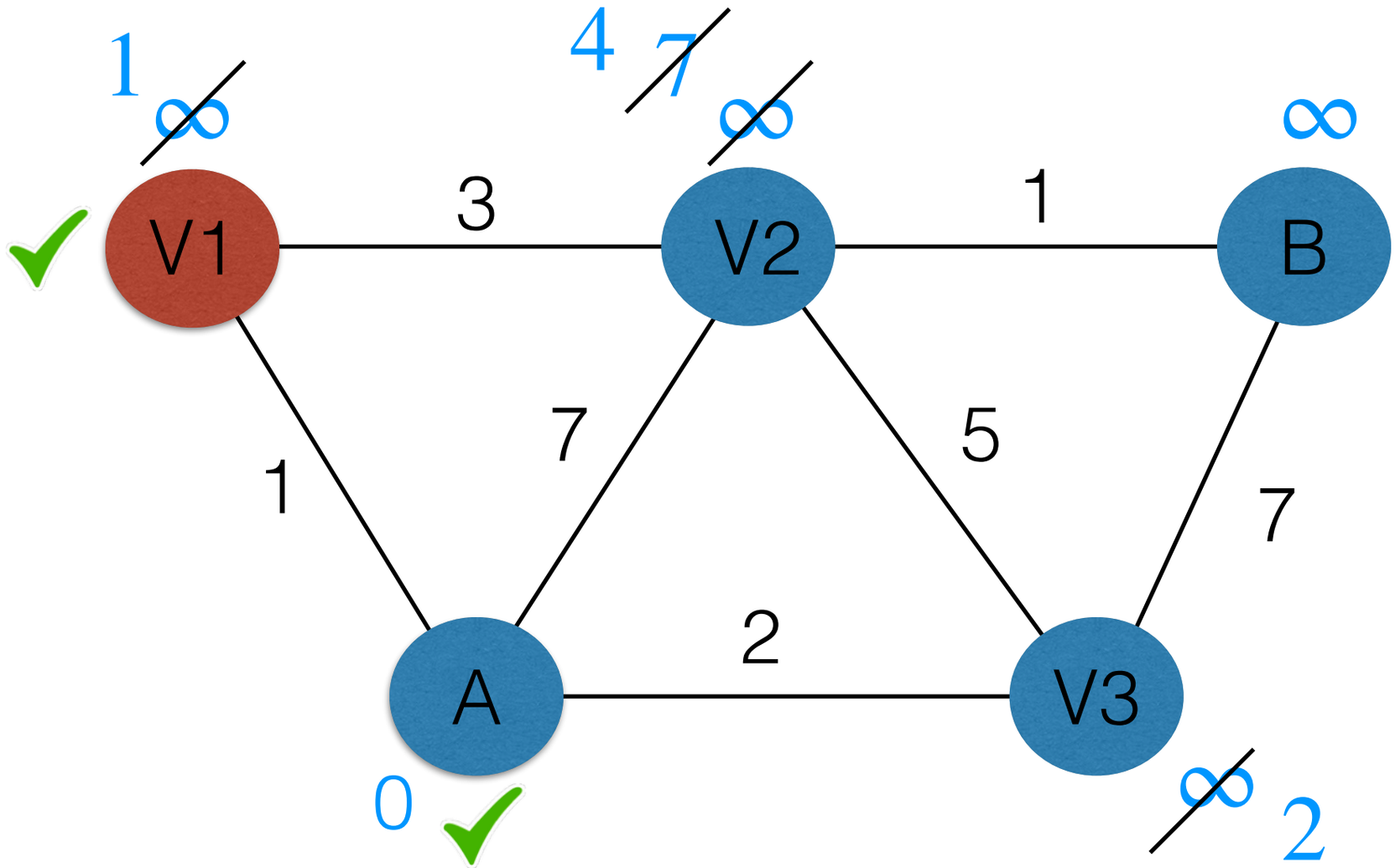
Dijkstra's Algorithm Example

$$Q = \{V2, V3\}$$



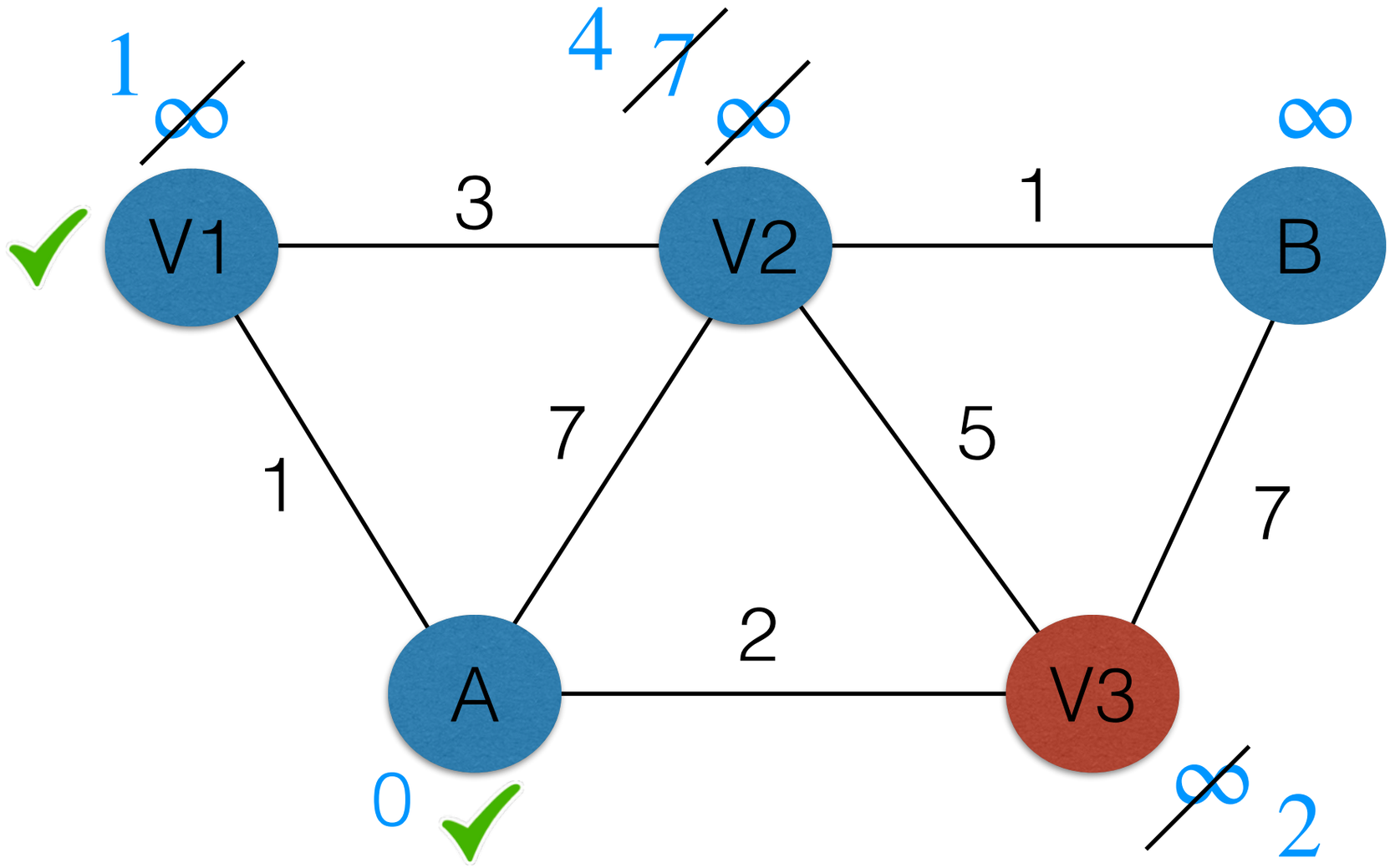
Dijkstra's Algorithm Example

$$Q = \{V2, V3\}$$



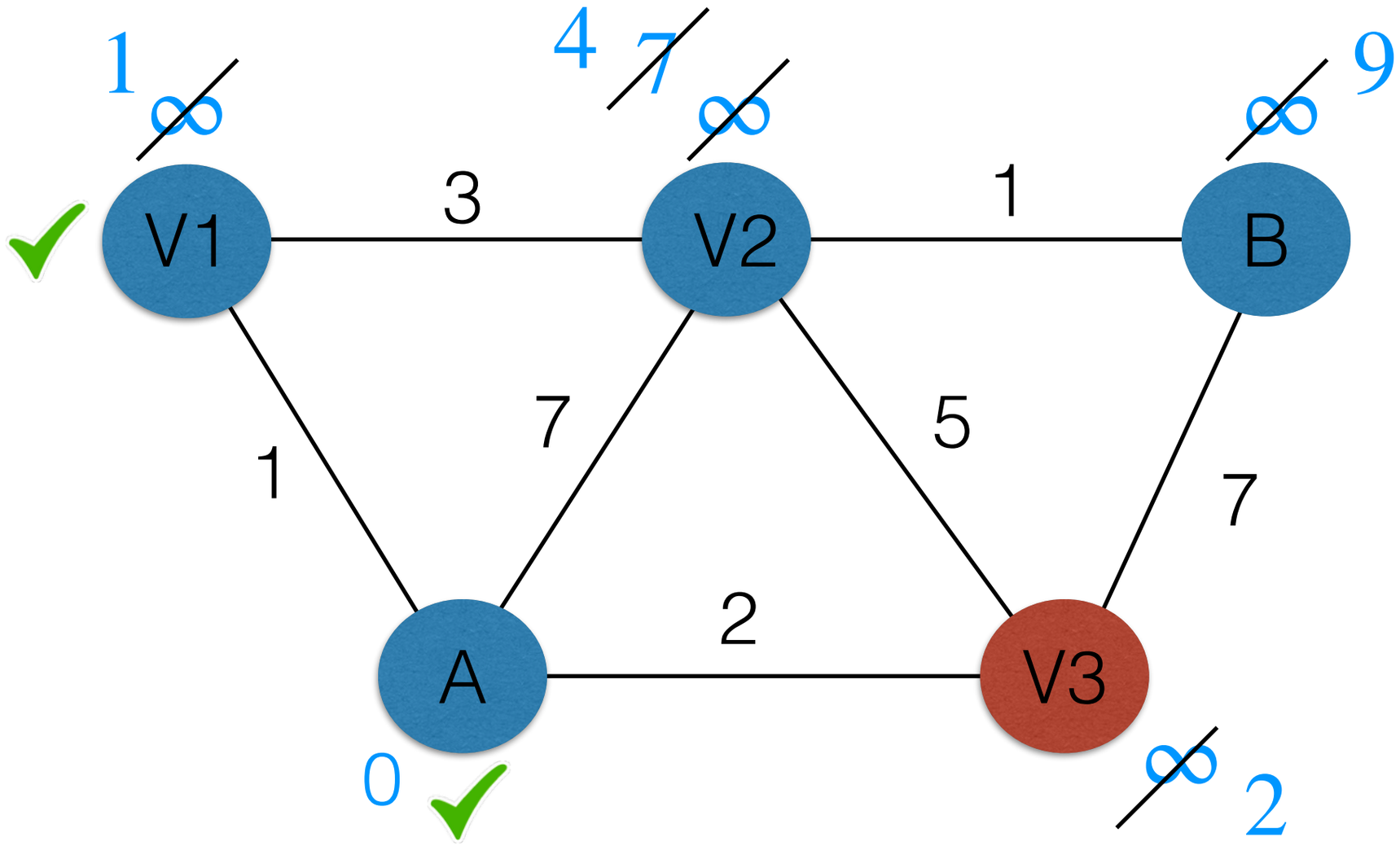
Dijkstra's Algorithm Example

$$Q = \{V2\}$$



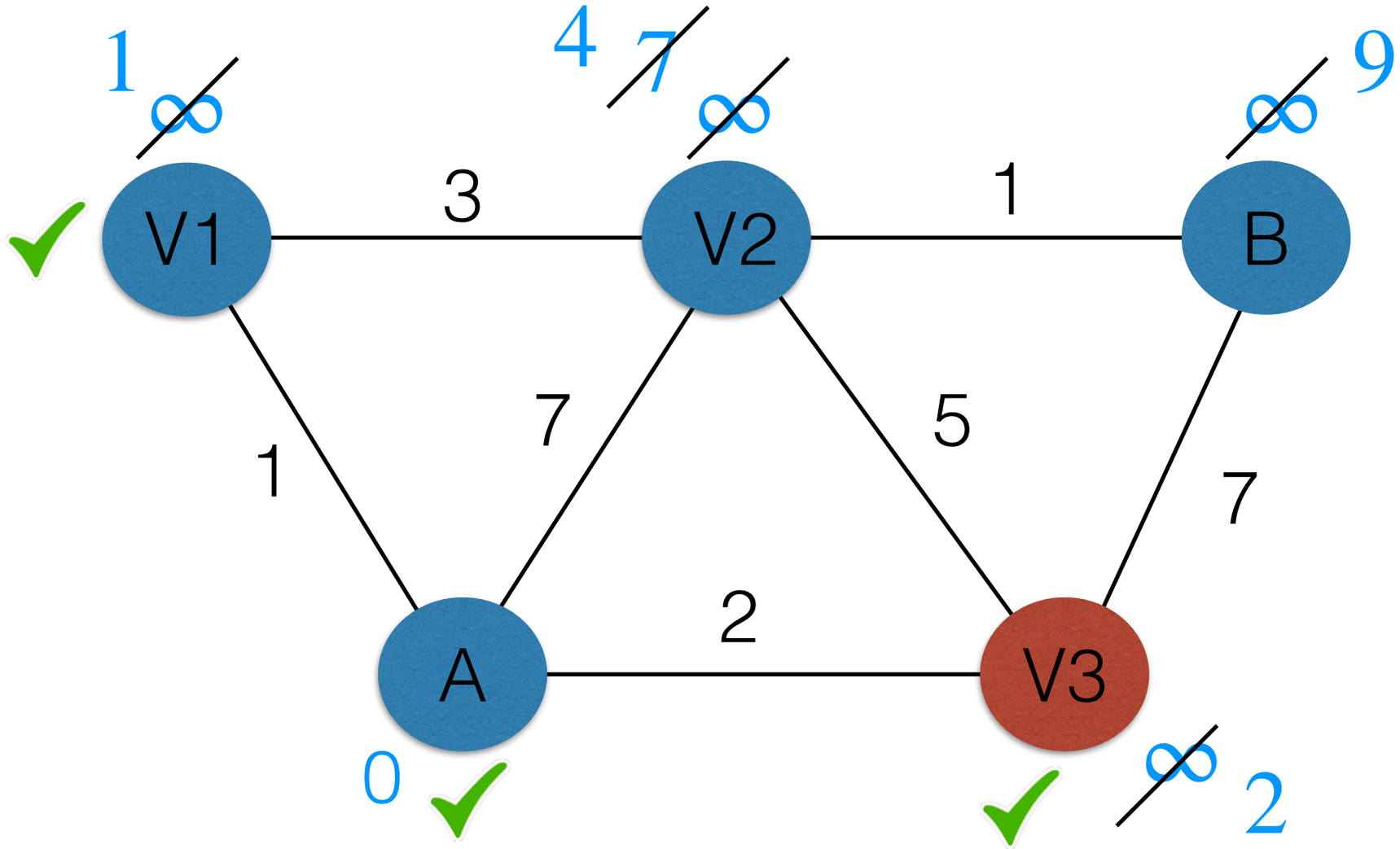
Dijkstra's Algorithm Example

$$Q = \{V2, B\}$$



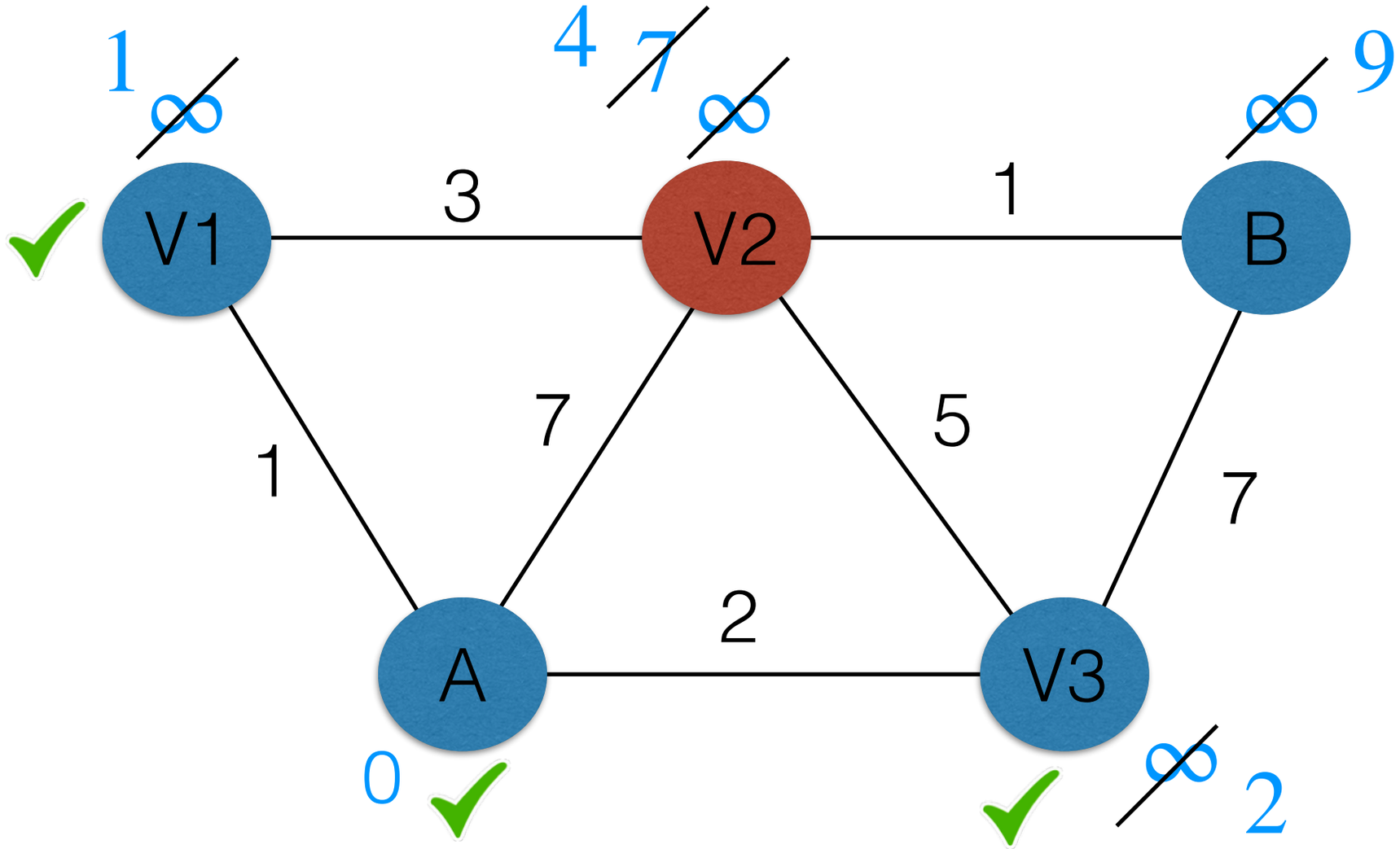
Dijkstra's Algorithm Example

$$Q = \{V2, B\}$$



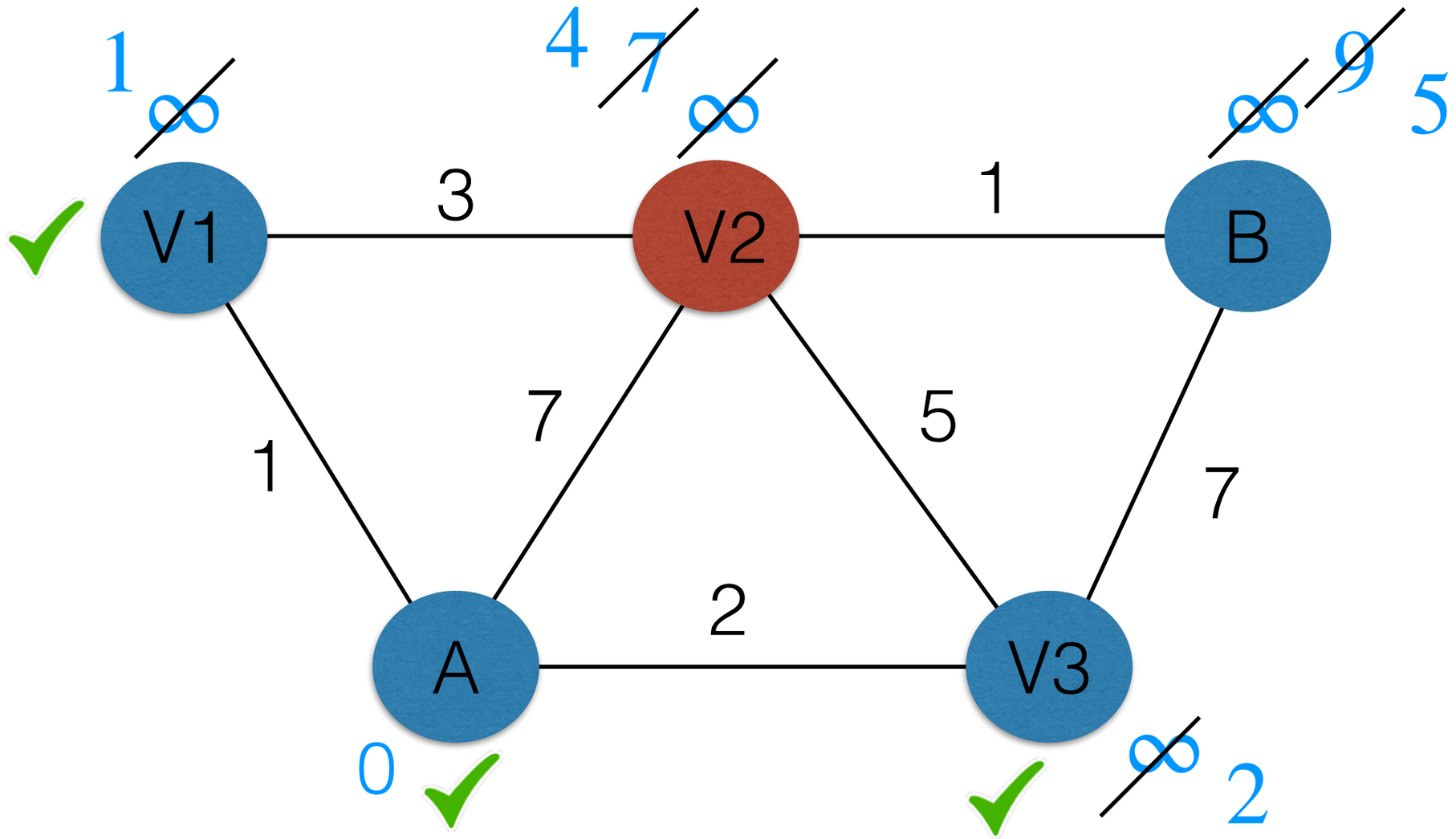
Dijkstra's Algorithm Example

$$Q = \{B\}$$



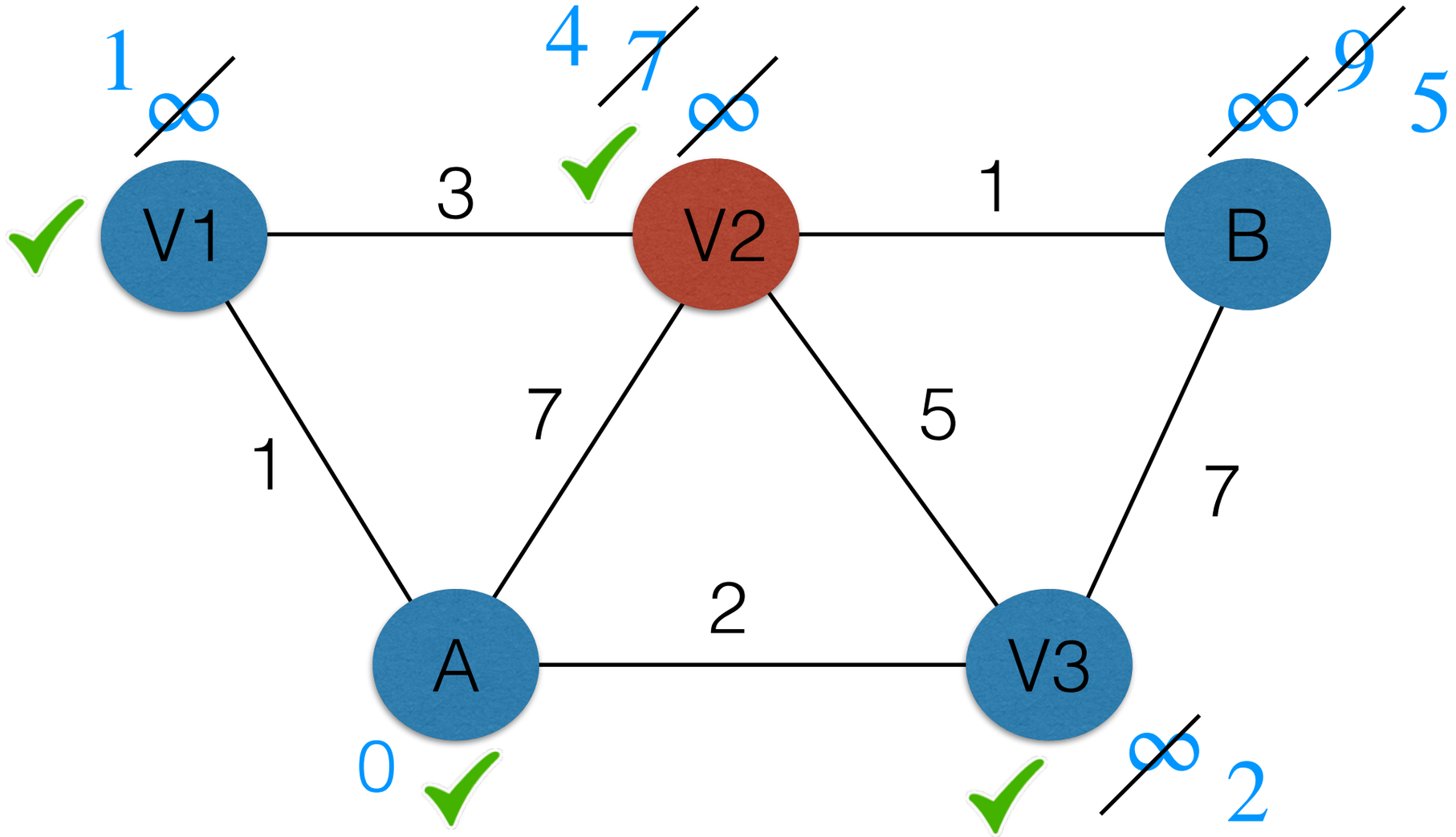
Dijkstra's Algorithm Example

$$Q = \{B\}$$



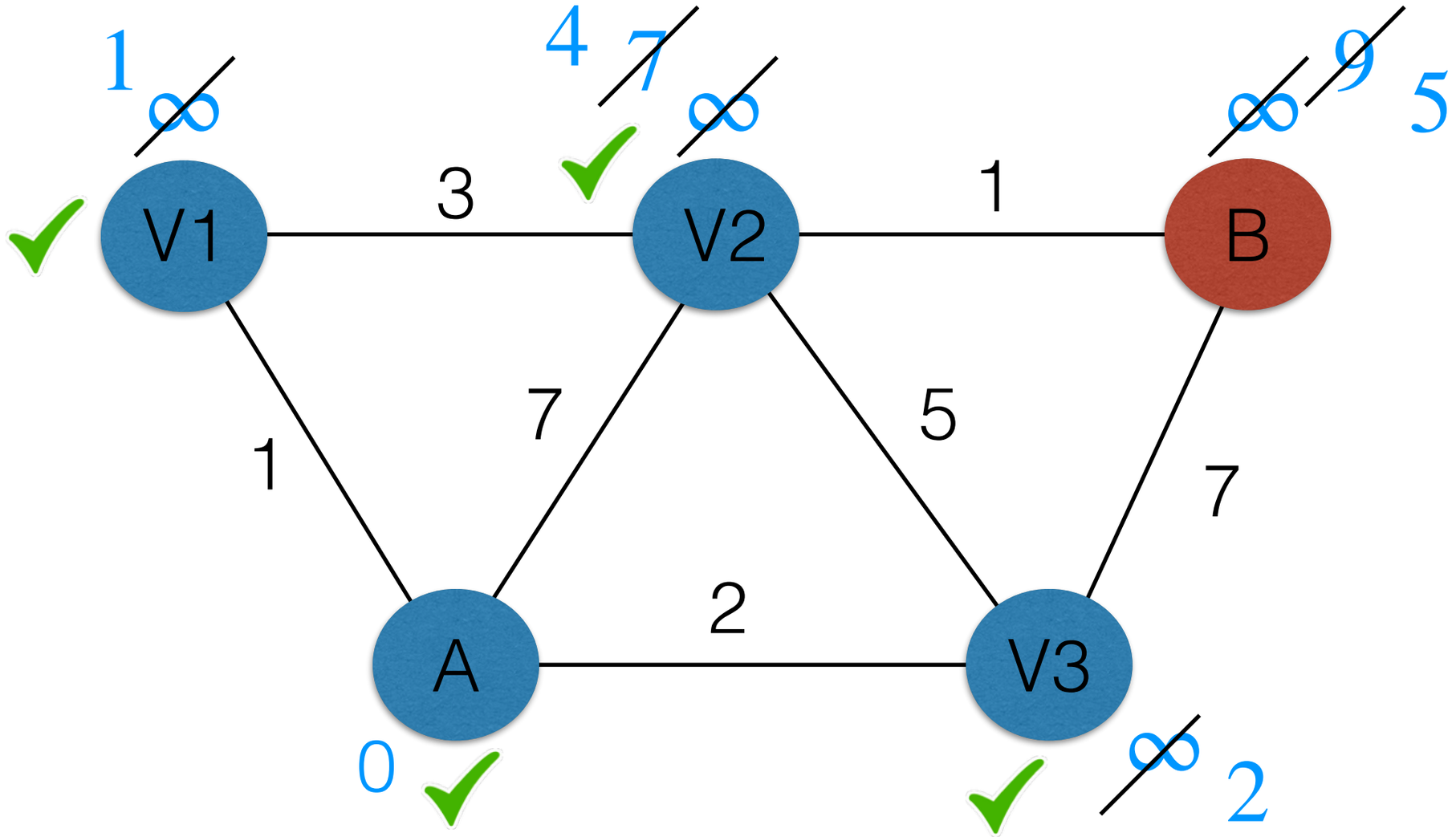
Dijkstra's Algorithm Example

$$Q = \{B\}$$

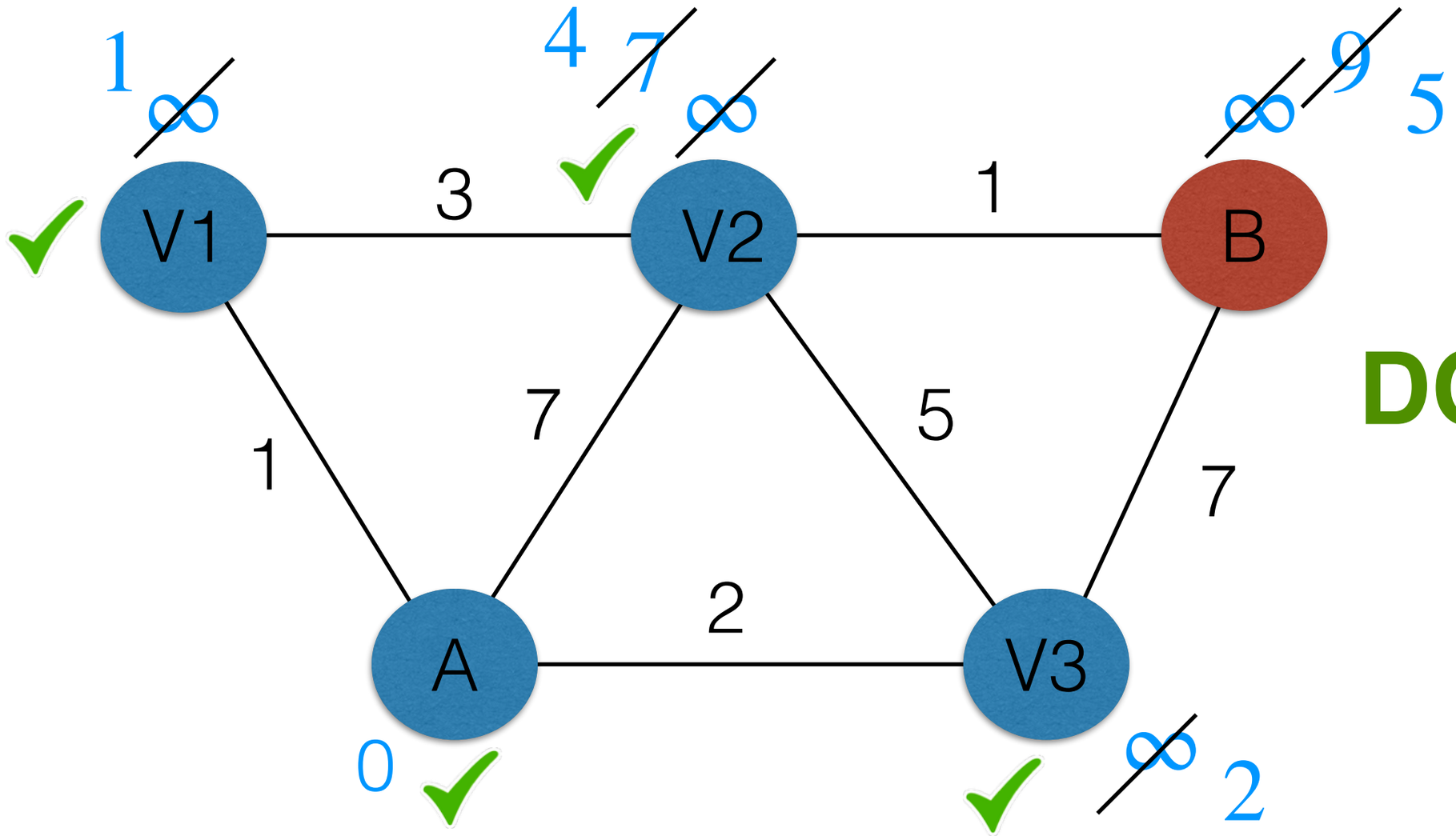


Dijkstra's Algorithm Example

$$Q = \{\}$$



Dijkstra's Algorithm Example



DONE!

Dijkstra in action

https://upload.wikimedia.org/wikipedia/commons/2/23/Dijkstras_progress_animation.gif

The Simple, Elegant Algorithm That Makes Google Maps Possible

Edsger W. Dijkstra's short solution to a bottomless complexity.

By **Michael Byrne**

Mar 22 2015, 8:00am Share Tweet

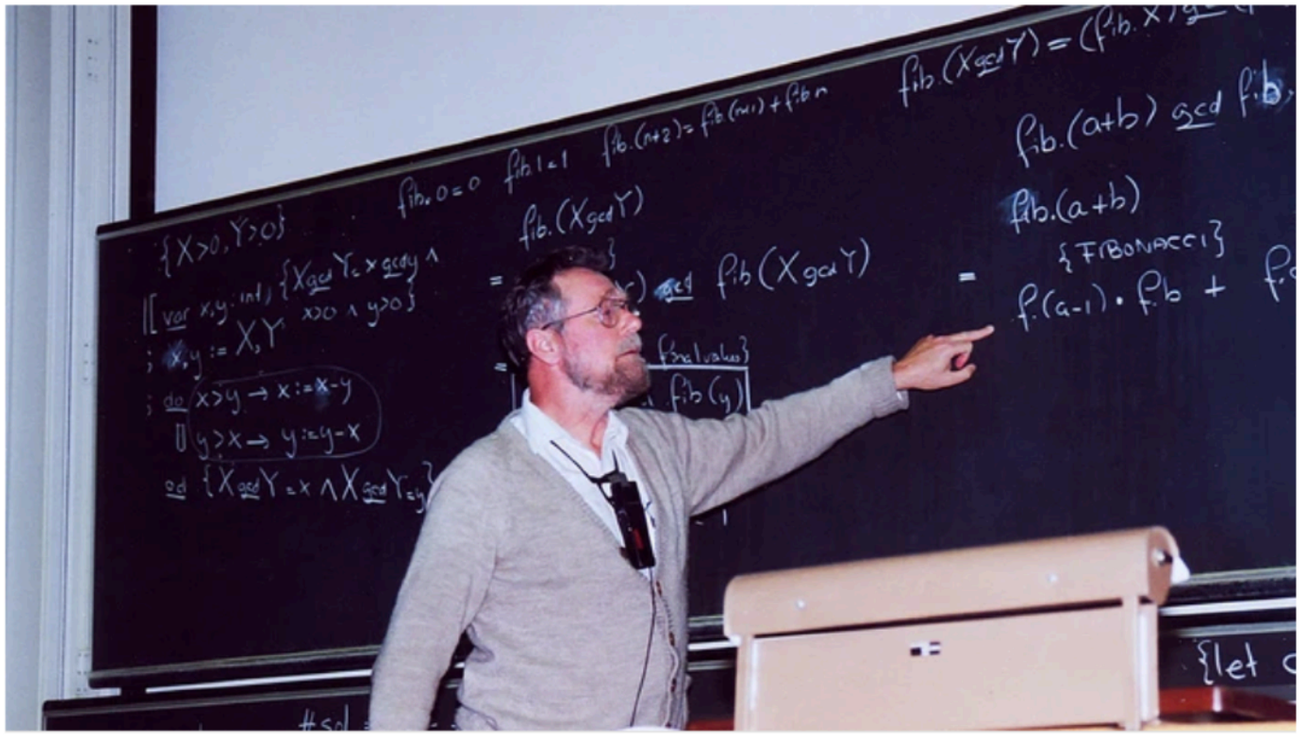


IMAGE: ANDREAS F. BORCHERT/WIKI

https://www.vice.com/en_us/article/4x3pp9/the-simple-elegant-algorithm-that-makes-google-maps-possible

Dijkstra's algorithm

- Looking at the animation of Dijkstra's algorithm is a bit frustrating
- It explores uniformly in all directions, which seems wasteful
- Is there a way to reduce the number of states we explore?

A* algorithm

- The A* algorithm uses a heuristic to implement $Q\,.\,GetVertex()$:

  $F(x) \triangleq C(x) + H(x)$

  where $C(x)$: estimated cost-to-come from A to x

  $H(x)$: estimated cost-to-go from x to B

- **Important:** $H(x)$ needs to be an underestimate (i.e., lower bound) on the optimal cost-to-go from x to the goal
  - This is required for the algorithm to work correctly
  - We can often get an underestimate $H(x)$ quite easily (e.g., Euclidean distance between x and B for distance-optimal planning)

Dijkstra and A*

Note: Dijkstra's algorithm is a special case of A* (with $H(x) = 0$)

A* algorithm

```
// Initialize the queue and also initialize a set of "dead" states
Q := {start},
 DeadSet = {}
// For vertex v, parent(v) is the vertex immediately preceding it on the cheapest path from start to v currently known.
parent := empty list
// For vertex x, C(x) is the cost of the cheapest path from start to x currently known.
Initialize C(x) to be infinity for all x
C(A) := 0 // Initialize C to be 0 for the start vertex A
// For vertex x, F(x) := C(x) + H(x).
Initialize F(x) to be infinity for all x
F(A) := H(A) // Initialize F(A) to be H(A) since C(A) = 0

while Q is not empty
    x := the vertex in Q having the lowest F value
    if x = goal
        Return SUCCESS

    Q.Remove(x)
    DeadSet.Add(x)
    for each neighbor x' of x
        if x' in DeadSet
            continue
        // l(x,x') is the cost of the edge from x to x'
        // tentative_C is the cost from A to x' through x
        tentative_C := C(x) + l(x, x')
        if tentative_C < C(x')
            // This path to neighbor is better than any previous one. Record it!
            parent(x') := x
            C(x') := tentative_C
            F(x') :=  C(x') + H(x')
            if x' not in Q
                Q.add(x')

// Open set is empty but goal was never reached
return FAILURE
```

A* algorithm

- Similar to BFS and DFS, we can find the (optimal) path by backtracking from the goal using the recorded parents

A* in action

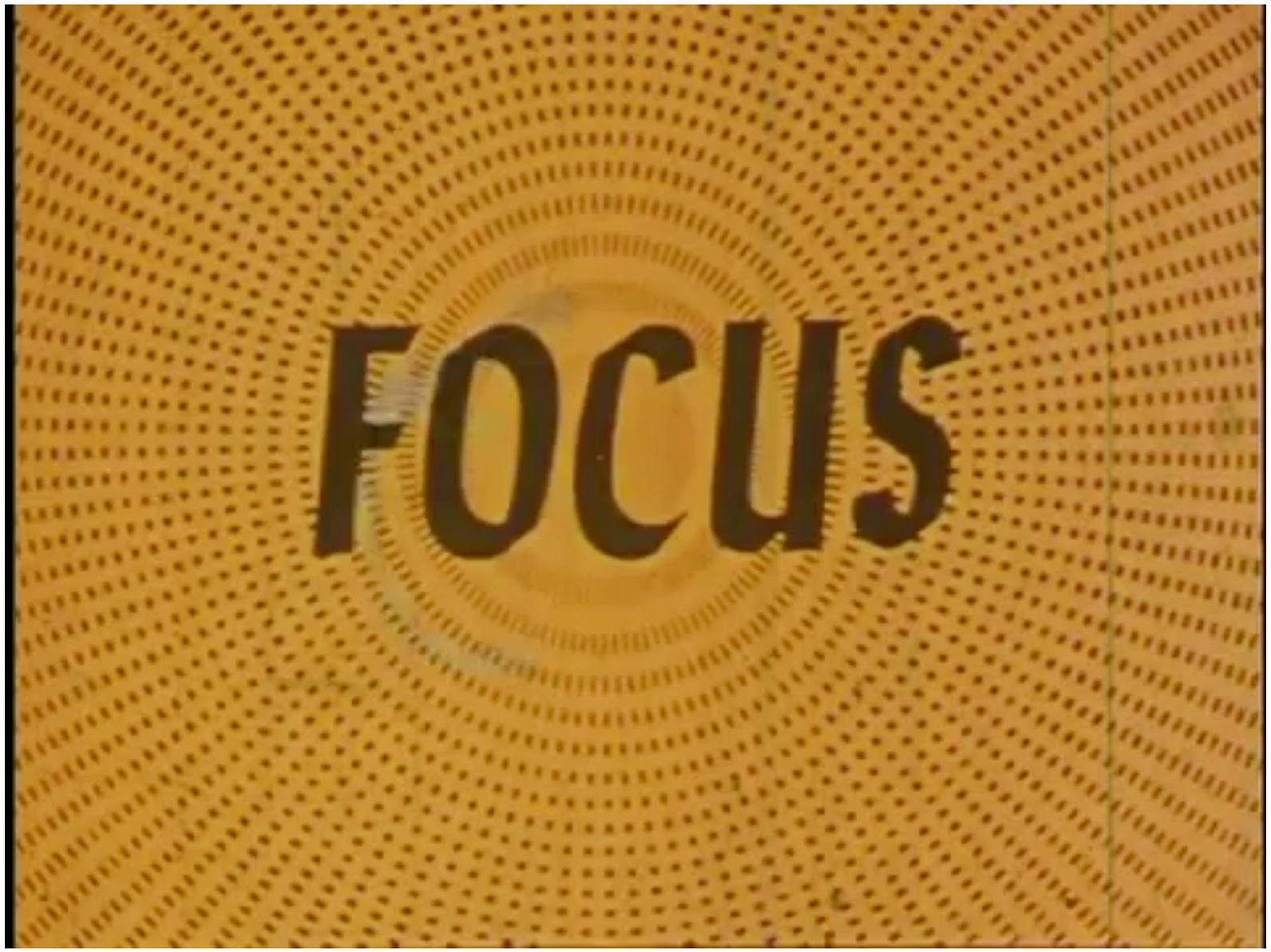
https://en.wikipedia.org/wiki/A*_search_algorithm#/media/File:Astar_progress_animation.gif

Side-by-side comparison

Dijkstra

A*

Shakey



- A* was invented by researchers working on Shakey (~1972)

https://www.youtube.com/watch?v=GmU7SimFkpU

Learning heuristics

- Maybe it makes sense to learn the heuristic for the cost-to-go?
  - Will say more about learning later in course!

Optimality: Broader Implications

The Value Alignment Problem

- We have encountered the idea of optimality twice (with LQR and optimal planning algorithms)

- So far, we have considered relatively simple cost functions

- But, choosing a cost function for complex tasks can be challenging!

  - How can we make sure that the cost function reflects what we actually want?

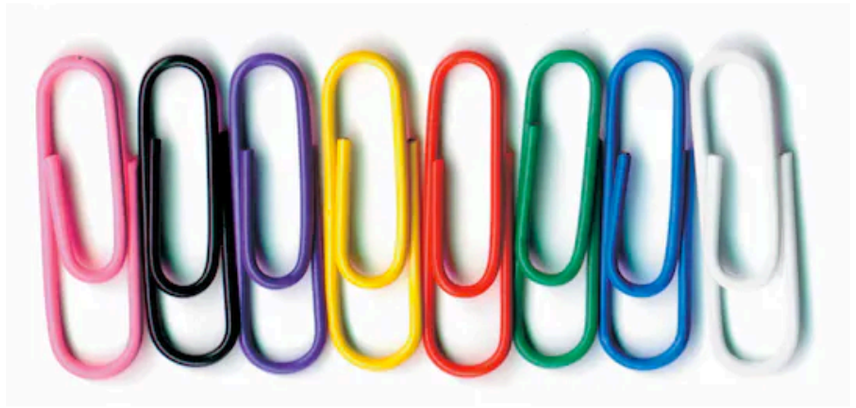
    - This is known as the value alignment problem

Value Alignment

- The Paperclip Maximizer (Nick Bostrom; 2003)
- Thought experiment: AI system whose only goal is to make as many paperclips as possible

Value Alignment

- The Paperclip Maximizer (Nick Bostrom; 2003)
- Thought experiment: AI system whose only goal is to make as many paperclips as possible



Value Alignment

- The Paperclip Maximizer (Nick Bostrom; 2003)
- Thought experiment: AI system whose only goal is to make as many paperclips as possible

Value Alignment

Instrumental Goals

The AI will realize quickly that it would be much better if there were no humans because humans might decide to switch it off. Because if humans do so, there would be fewer paper clips. Also, human bodies contain a lot of atoms that could be made into paper clips. The future that the AI would be trying to gear towards would be one in which there were a lot of paper clips but no humans.

Nick Bostrom

Value Alignment

- The Paperclip Maximizer demonstrates a major technical challenge
- Value alignment problem:
  - How can we ensure that robot's objectives ("values") are aligned with ours?
  - Need to somehow take side-effects and instrumental goals into account (e.g., not turning humans into paperclips!)

Objections

- Two possible objections:
  - Ok, but this sounds very far-fetched.
  - Can we not just learn values from humans?

Not so far-fetched

- Value alignment is a problem already!
  - Social media algorithms
  - Maximizing engagement can have many potentially adverse consequences