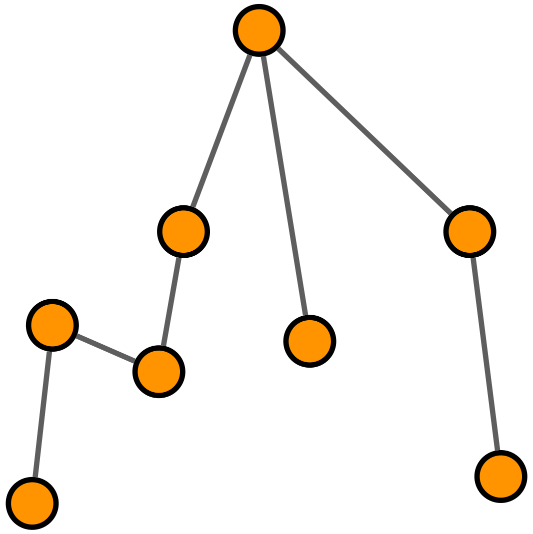
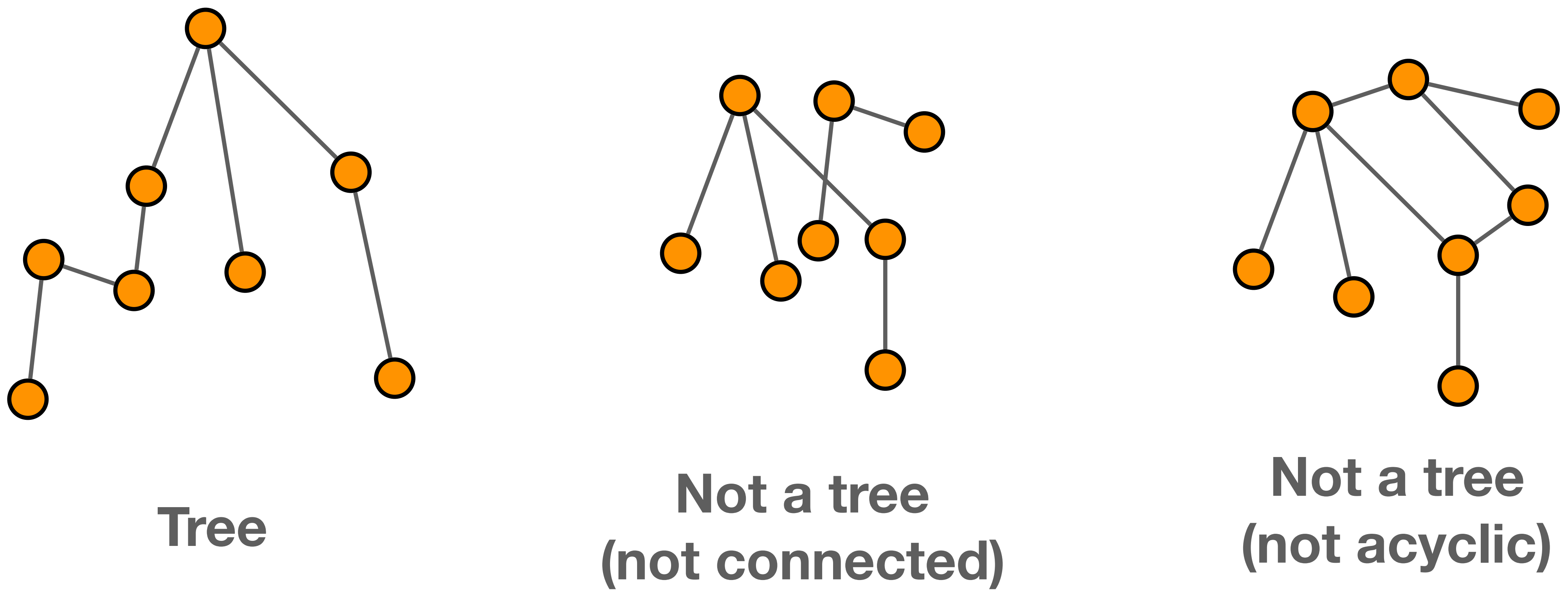
Trees



Trees

- A *tree* is a connected graph without cycles (acyclic graph)



Uniqueness of Paths

- In a tree there is a unique path between any two nodes.
  - Existence: path exists since trees are connected by definition
  - Uniqueness: suppose there are two different paths between two distinct nodes u,v:

$$u=a_0 - a_1 - a_2 - \ldots\ldots - a_t - a_{t+1}=v \quad \text{Path 1}$$

$$u=b_0 - b_1 - b_2 - \ldots\ldots - b_k - b_{k+1}=v \quad \text{Path 2}$$

  - Let s:=min{ $i \mid a_i \neq b_i$}. Note that $s > 0$.
  - Let r := min { $i \mid i > s$ and there exists j with $a_i = b_j$}. Let m be such that $a_r=b_m$. Note that $r \leq t+1$.
  - Then

➡ $a_{s-1} - a_s - \ldots - a_{r-1} - a_r=b_m - b_{m-1} - \ldots - b_s - b_{s-1}=a_{s-1}$

  - is a cycle, a contradiction.

Number of Edges

- Let $G=(V,E)$ be a tree. Then $|E| = |V|-1$.
  - Proof 1: Euler's formula.
    - ➡ The graph is planar (why?).
    - ➡ Number of faces is 1.
    - ➡ $1 - |E|+|V| = 2$, so $|E| = |V|-1$.

Number of Edges

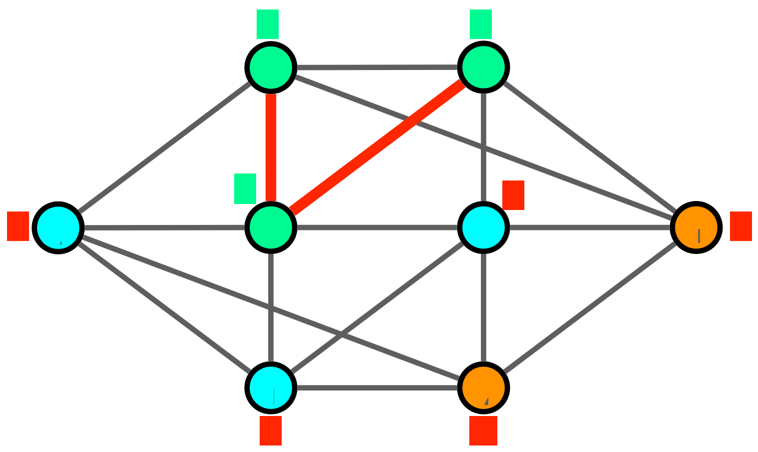
- Let $G=(V,E)$ be a tree. Then $|E| = |V|-1$.
  - Proof 2: Strong induction on $|V|$.
    - ➡ Start: $|V|=1$, then $|E|=0$ (since self-loops don't exist).
    - ➡ Step: Let $|V|=n+1$, $n\geq 1$.Since graph is connected, there is at least one edge $\{u,v\}$ in this graph.
    - ➡ Delete that edge.
    - ➡ G_u graph consisting of all vertices reachable from u after deletion of edge
    - ➡ G_v graph consisting of all vertices reachable from v after deletion of edge
    - ➡ G_u and G_v have disjoint node sets (since otherwise there are two paths from u to v in the original graph).
    - ➡ G_u and G_v are trees (no cycles since original graph does not have cycles, and connected by definition).
    - ➡ Let m be number of vertices in G_u, hence $n+1-m$ is number of vertices in G_v.
    - ➡ By induction hypothesis: number of edges in G_u is $m-1$, and number of edges in G_v is $(n+1-m)-1=n-m$.
    - ➡ Total number of edges in original graph is therefore $m-1+n-m+1=n$. QED

Edge we deleted

Minimality

- Let $G=(V,E)$ be a graph with $|E| < |V|-1$. Then G is not connected.
  - Suppose that G is connected
  - Remove all cycles by removing edges if needed, without disconnecting the graph.
  - New graph $G = (V,E')$ is acyclic, and $|E'| \leq |E| < |V|-1$.
  - It is also connected, since original graph is connected.
  - Hence it is a tree.
  - But then $|E'|=|V|-1$, a contradiction to the result on the previous page.

Dijkstra's Algorithm



Weighted Graphs

- A weighted connected graph (with positive weights) is a graph $G=(V,E)$ together with a weight function $w\colon E \rightarrow \mathbf{R}_{>0}$.
- Problem: given a node in the graph, find shortest paths from that node to all the other nodes.

Example 1: Routing Problem

- Nodes given by routers in a network
- An edge between two nodes if there is a direct connection
- Weight: “round-trip-time”
- Shortest path algorithm determines for every router the shortest (smallest round-trip time) path to all other routers

Example 2: Navigation System

- Nodes: all possible destinations in a country
- Edge: if there is a road connecting one destination to another
- Weight: Distance (can be geographic or temporal)
- A navigation system can find for the current location shortest paths to all other locations

Example 3: Air Travel

- Nodes: cities with an airport
- Edge between two nodes: if there is a direct flight from one city to another
- Weight: length of the flight
- The shortest path algorithm determines from a given city a sequence of flights to any other city with the smallest flight time

Dijkstra's Algorithm

- Fix initial node u_0.
- Determines for all nodes v in the graph
  - The quantity $L(v)$
    - ➡ At the end of the algorithm this will be the length of shortest path from u_0.
  - A node called from(v) which is the predecessor of v in the shortest path from u_0 to v.
- Maintains a set S which at each iteration contains the nodes for which the shortest path has already been determined.
- At the beginning of the algorithm
  - $S = \emptyset$
  - $L(u_0) = 0$, and $L(v) = \infty$ for all $v \neq u_0$
  - from(u_0) = u_0, from(v) = nil for $v \neq u_0$

Dijkstra's Algorithm

- As long as $S \neq V$
  - Select $u \notin S$ with $L(u)$ minimal.
  - Replace S by $S \cup \{u\}$.
  - For all $v \notin S$:
    - ➡ Set $c(v) := L(u) + w(u,v)$
    - ➡ If $c(v) < L(v)$ then replace $L(v)$ by $c(v)$
      - ◉ and replace from(v) by u.

Dijkstra's Algorithm Example

pp. 684

$S = \emptyset$

$L(u_0) = 0$ ⟵ **minimum**

$L(A) = \infty$

$L(B) = \infty$

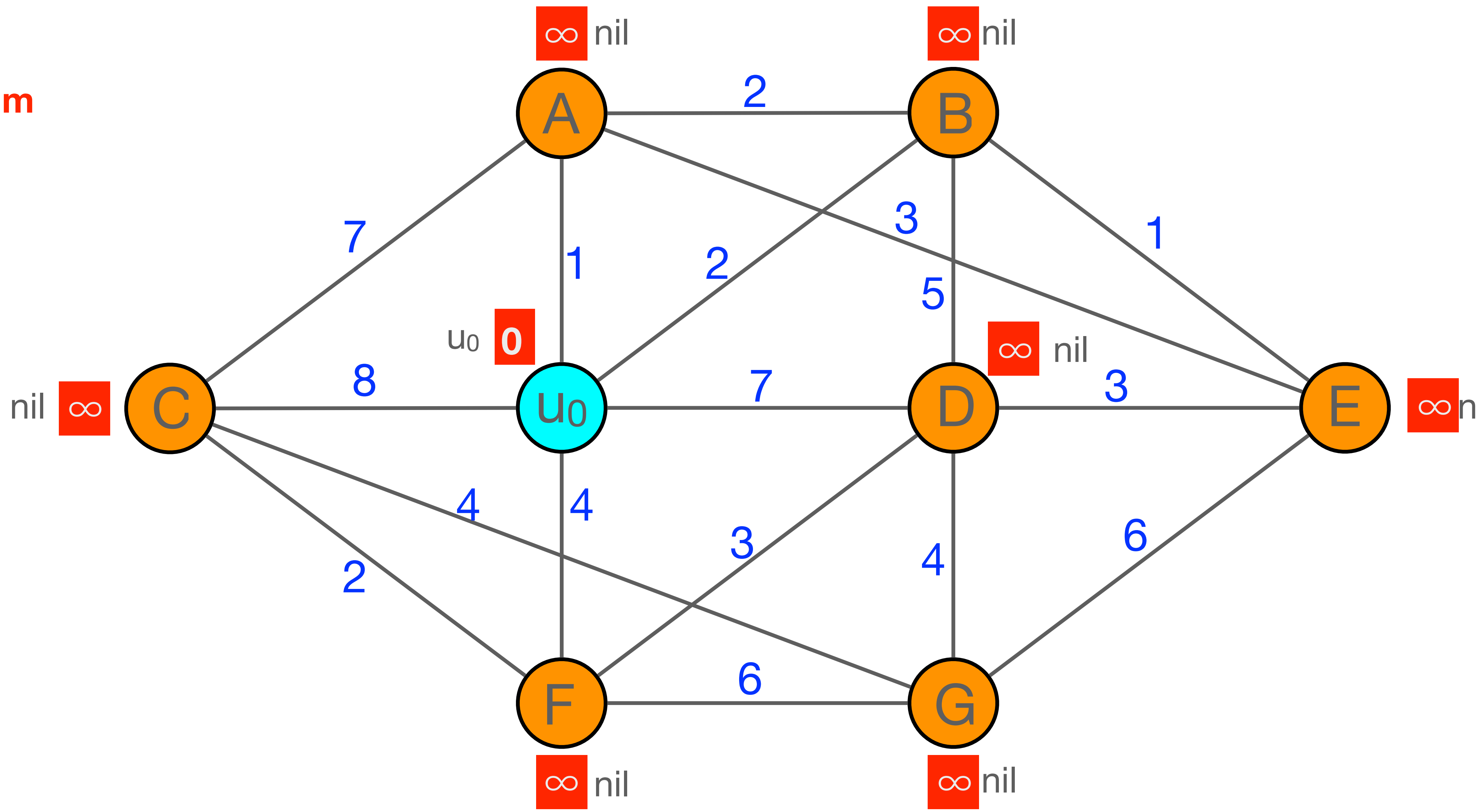
$L(C) = \infty$

$L(D) = \infty$

$L(E) = \infty$

$L(F) = \infty$

$L(G) = \infty$



final distance L

Current distance

Vertex that we know how to reach best

Vertex that has already been visited

Vertex that has not been visited yet

Dijkstra's Algorithm Example

pp. 684

$S = \{u_0\}$

$L(u_0) = 0$

$L(A) = 1$ ⟵ **minimum**

$L(B) = 2$

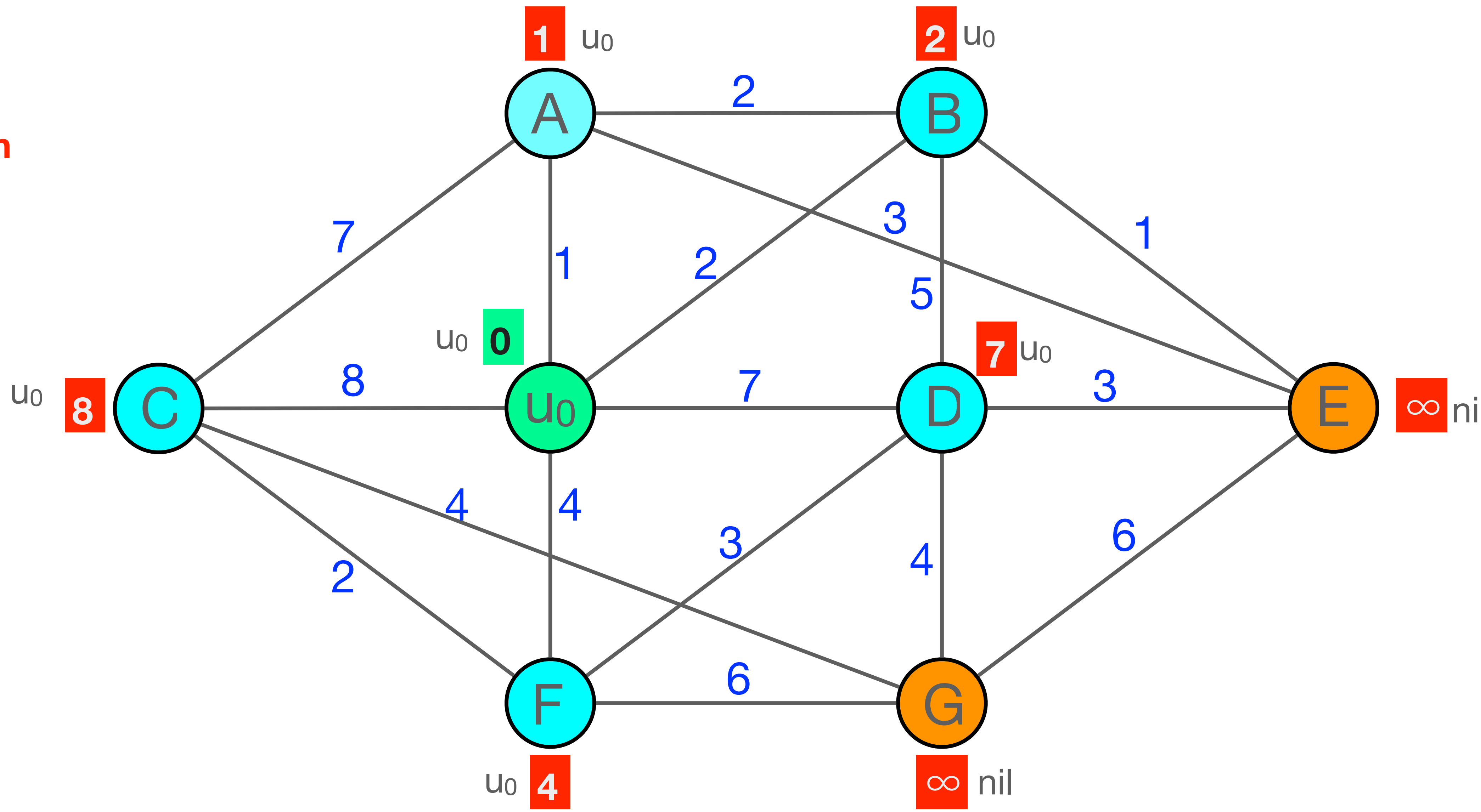
$L(C) = 8$

$L(D) = 7$

$L(E) = \infty$

$L(F) = 4$

$L(G) = \infty$



final distance L

Current distance

Vertex that we know how to reach best

Vertex that has already been visited

Vertex that has not been visited yet

Dijkstra's Algorithm Example

pp. 684

$S = \{u_0, A\}$

$L(u_0) = 0$

$L(A) = 1$

$L(B) = 2$ ⟵ minimum

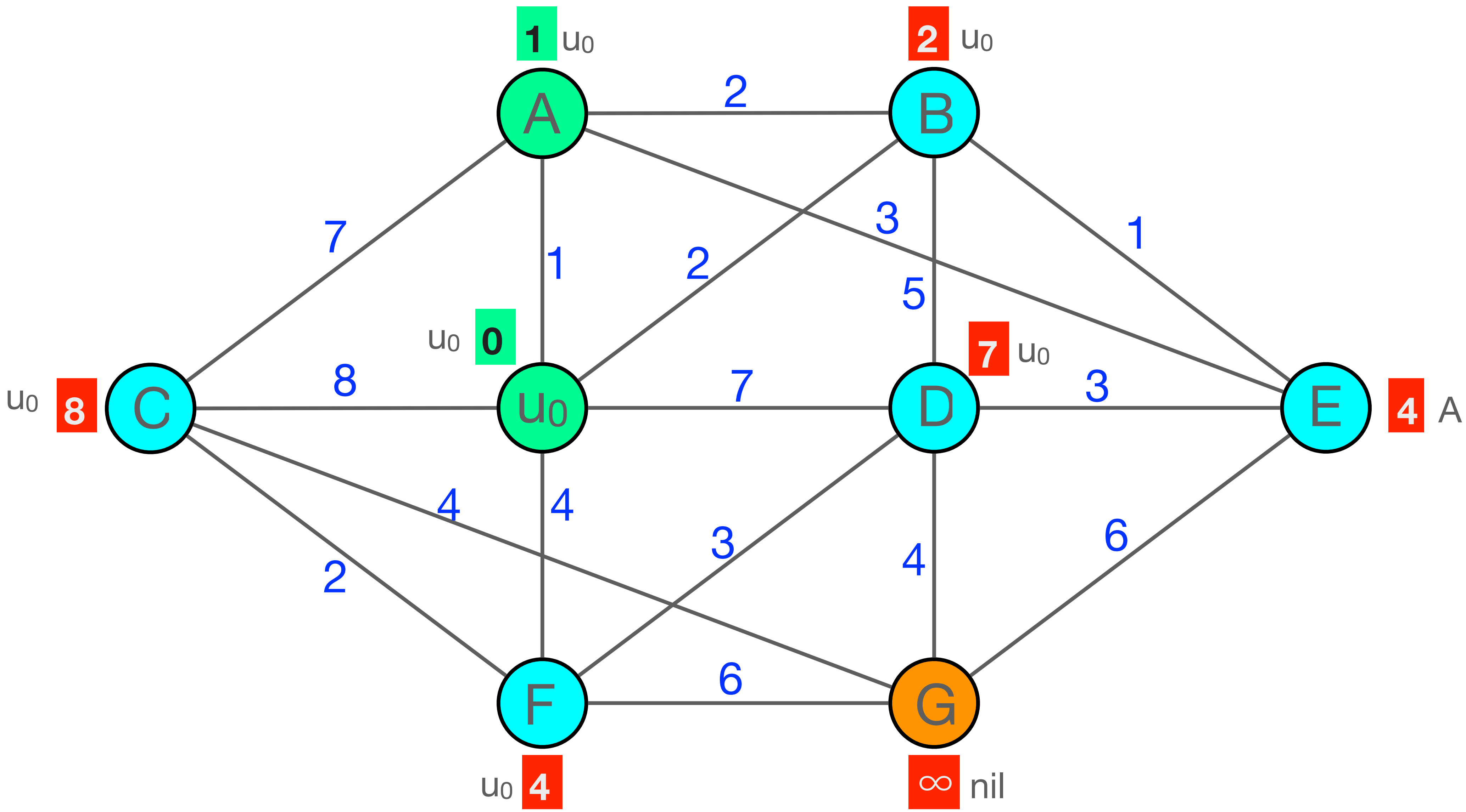
$L(C) = 8$

$L(D) = 7$

$L(E) = 4$

$L(F) = 4$

$L(G) = \infty$



final distance *L*

Current distance

Vertex that we know how to reach best

Vertex that has already been visited

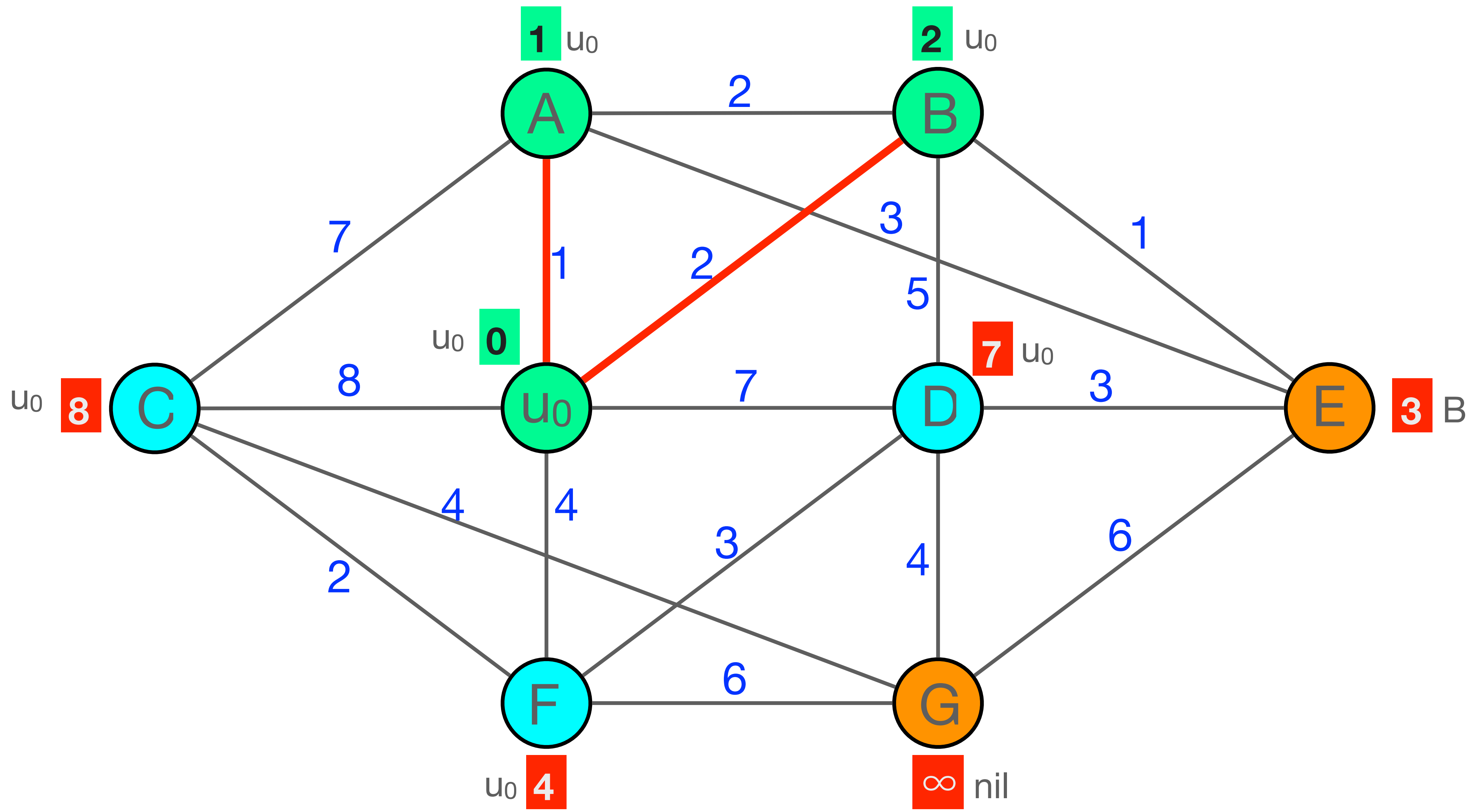
Vertex that has not been visited yet

Dijkstra's Algorithm Example

pp. 684

$S = \{u_0, A, B\}$

$L(u_0) = 0$
$L(A) = 1$
$L(B) = 2$
$L(C) = 8$
$L(D) = 7$
$L(E) = 3$ ⟵ minimum
$L(F) = 4$
$L(G) = \infty$



final distance L

Current distance

Vertex that we know how to reach best

Vertex that has already been visited

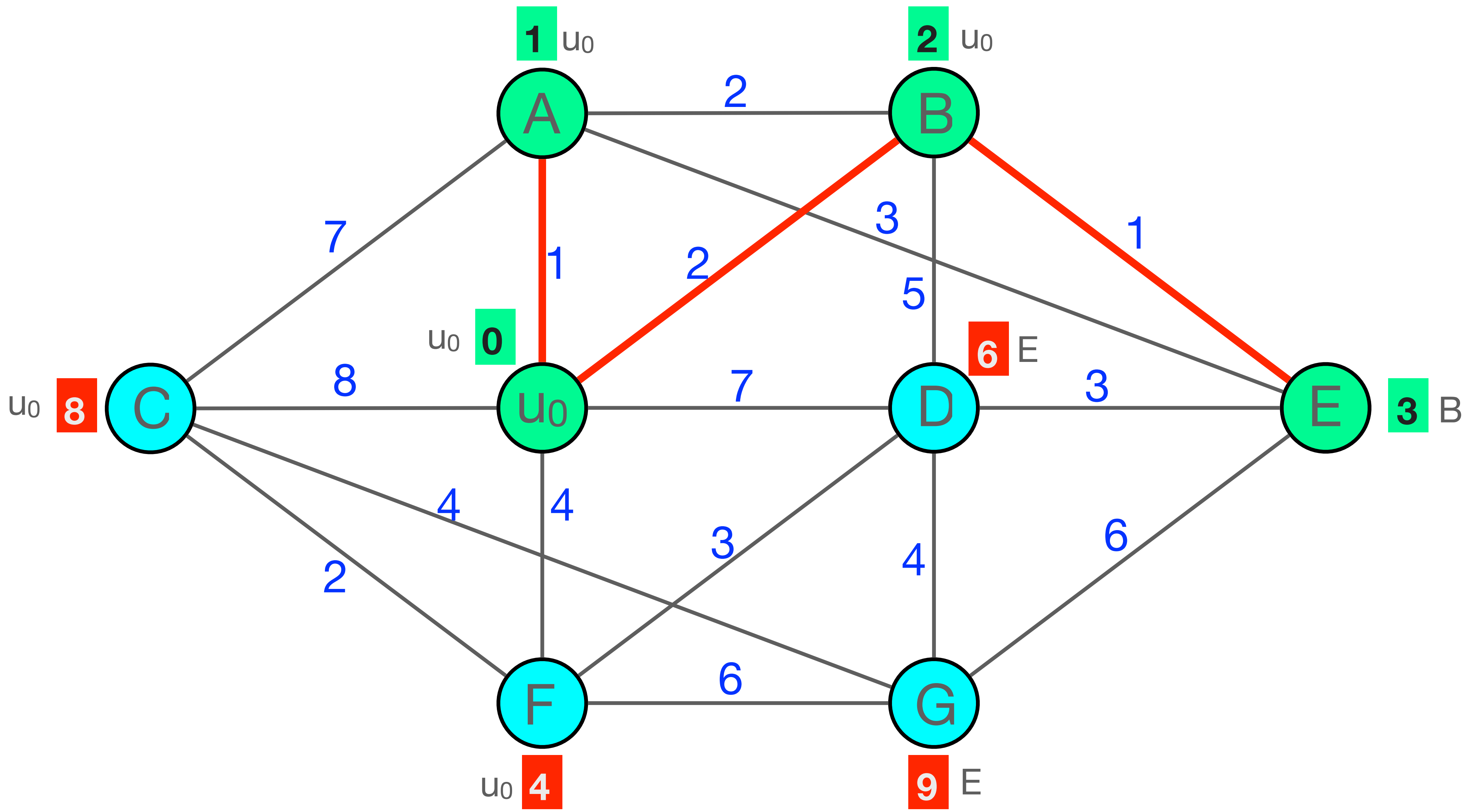
Vertex that has not been visited yet

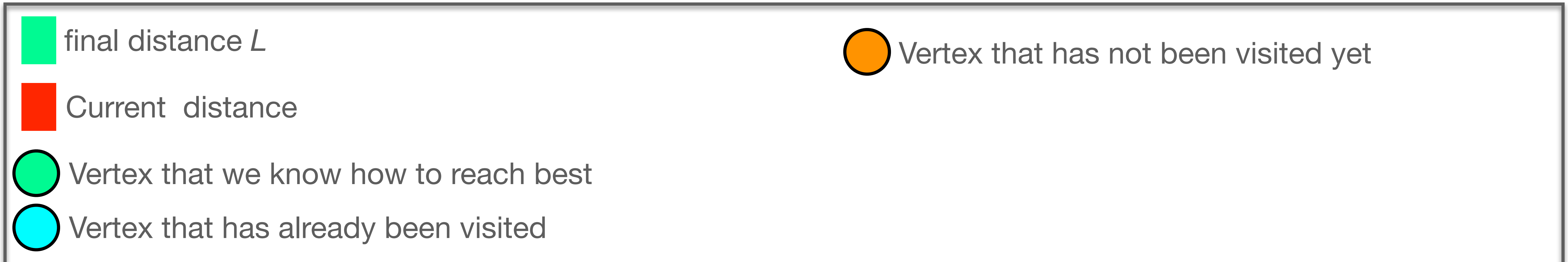
Dijkstra's Algorithm Example

pp. 684

$S = \{u_0, A, B, E\}$

$L(u_0) = 0$
$L(A) = 1$
$L(B) = 2$
$L(C) = 8$
$L(D) = 6$
$L(E) = 3$
$L(F) = 4$ ⟵ **minimum**
$L(G) = 9$



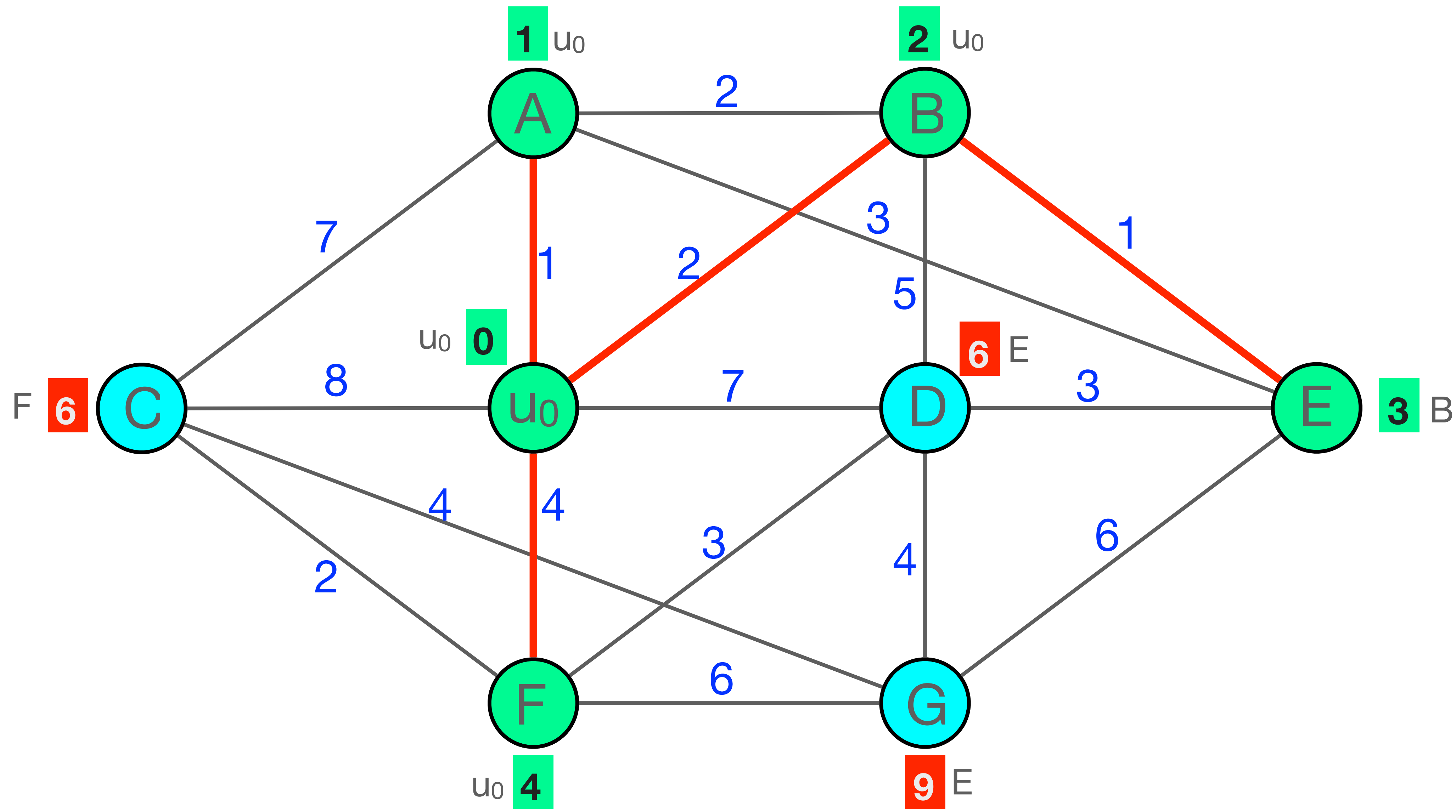


Dijkstra's Algorithm Example

pp. 684

$S = \{u_0, A, B, E, F\}$

$L(u_0) = 0$
$L(A) = 1$
$L(B) = 2$
$L(C) = 6$
$L(D) = 6$ ⟵ minimum
$L(E) = 3$
$L(F) = 4$
$L(G) = 9$



final distance L
Current distance
Vertex that we know how to reach best
Vertex that has already been visited
Vertex that has not been visited yet

Dijkstra's Algorithm Example

pp. 684

S = {u_0, A, B, E, F, D}

$L(u_0) = 0$

$L(A) = 1$

$L(B) = 2$

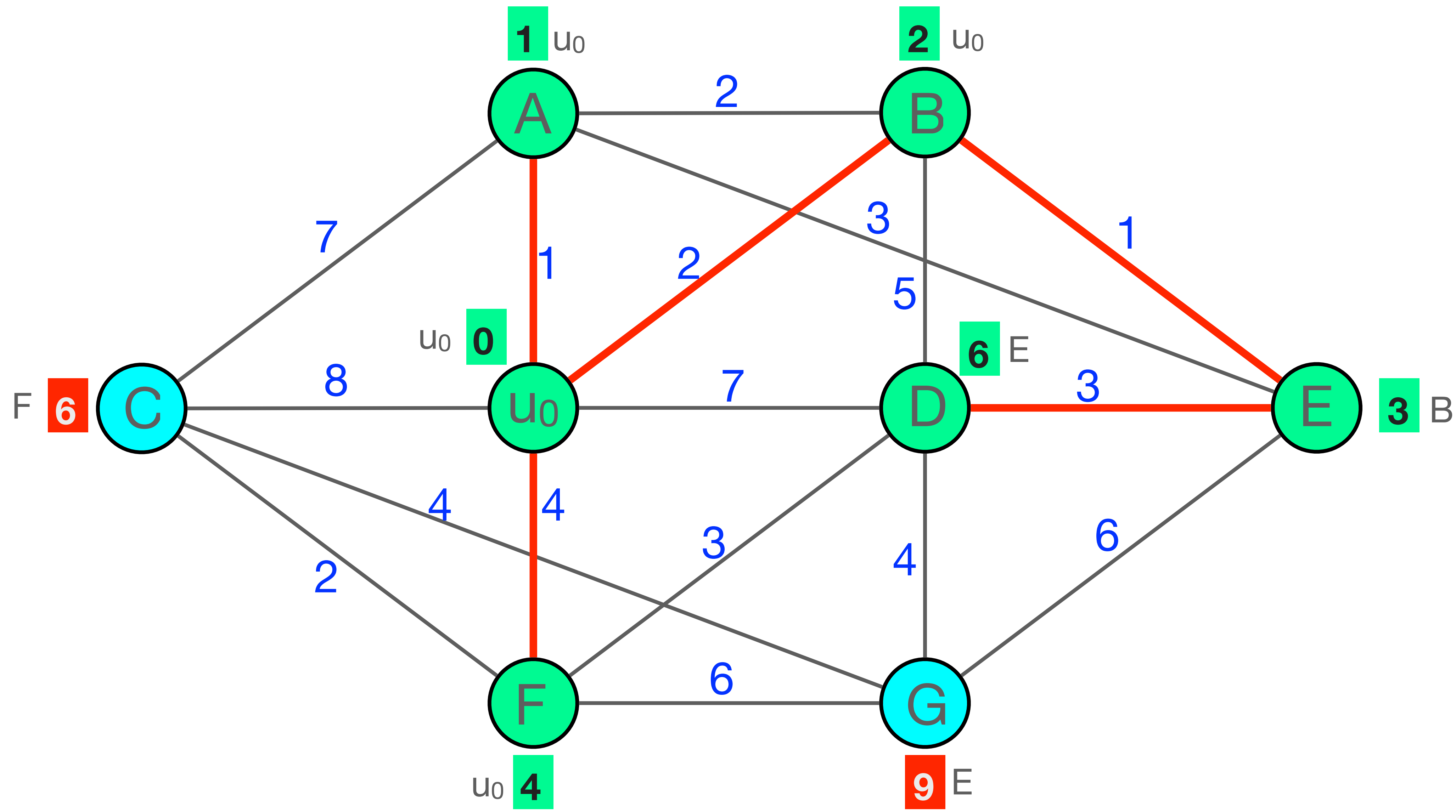
$L(C) = 6$ ⟵ **minimum**

$L(D) = 6$

$L(E) = 3$

$L(F) = 4$

$L(G) = 9$



- final distance L
- Current distance
- Vertex that we know how to reach best
- Vertex that has already been visited
- Vertex that has not been visited yet

Dijkstra's Algorithm Example

pp. 684

$S = \{u_0, A, B, E, F, D, C\}$

$L(u_0) = 0$

$L(A) = 1$

$L(B) = 2$

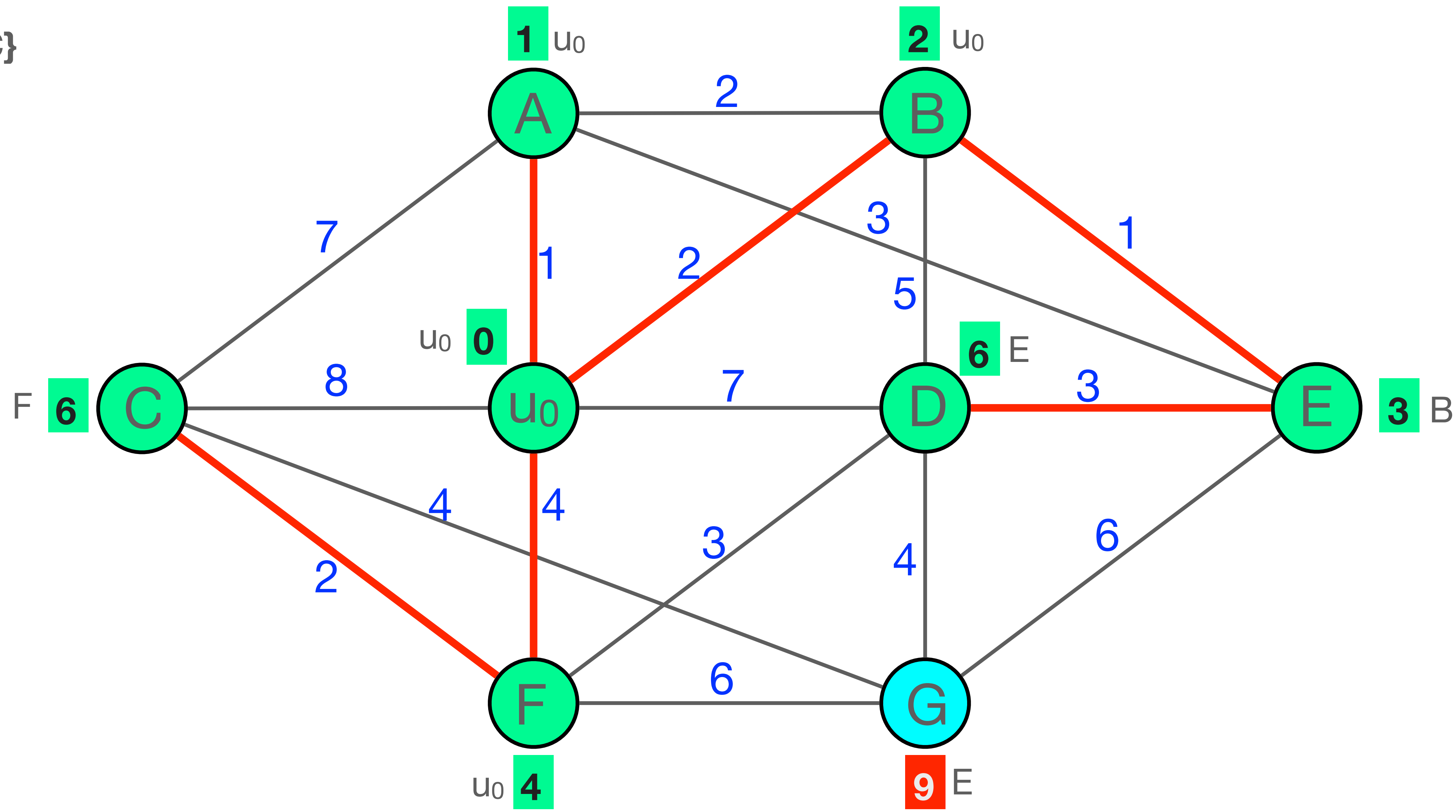
$L(C) = 6$

$L(D) = 6$

$L(E) = 3$

$L(F) = 4$

$L(G) = 9$ ⟵ **minimum**



final distance L

Current distance

Vertex that has not been visited yet

Vertex that we know how to reach best

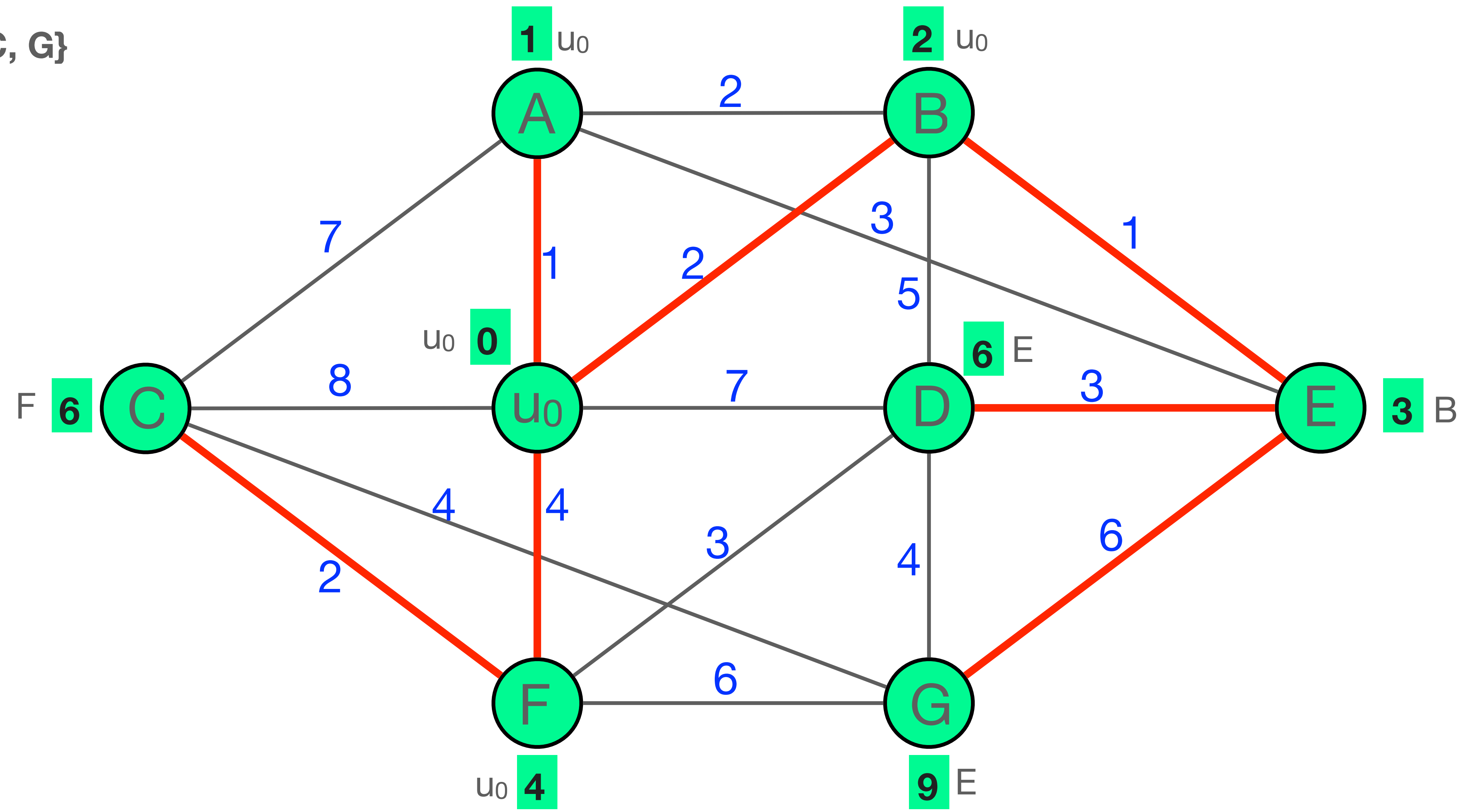
Vertex that has already been visited

Dijkstra’s Algorithm Example

pp. 684

$S = \{u_0, A, B, E, F, D, C, G\}$

$L(u_0) = 0$
$L(A) = 1$
$L(B) = 2$
$L(C) = 6$
$L(D) = 6$
$L(E) = 3$
$L(F) = 4$
$L(G) = 9$



final distance L

Current distance

Vertex that we know how to reach best

Vertex that has already been visited

Vertex that has not been visited yet

Proof of Correctness of Dijkstra's Algorithm

pp. 684-686

- We use induction on the $|S|$ to prove the following two facts:

  (1) For all $v \in S$: $L(v)$ is length of shortest path from u_0 to v.

  (2) For all $v \notin S$: $L(v)$ is length of shortest path from u_0 to v in which all nodes (except for v) are in S.

- Note that (1) is enough to show correctness:
  - At the end of the algorithm, $S=V$, and hence the L-values are the lengths of shortest paths

Induction Start

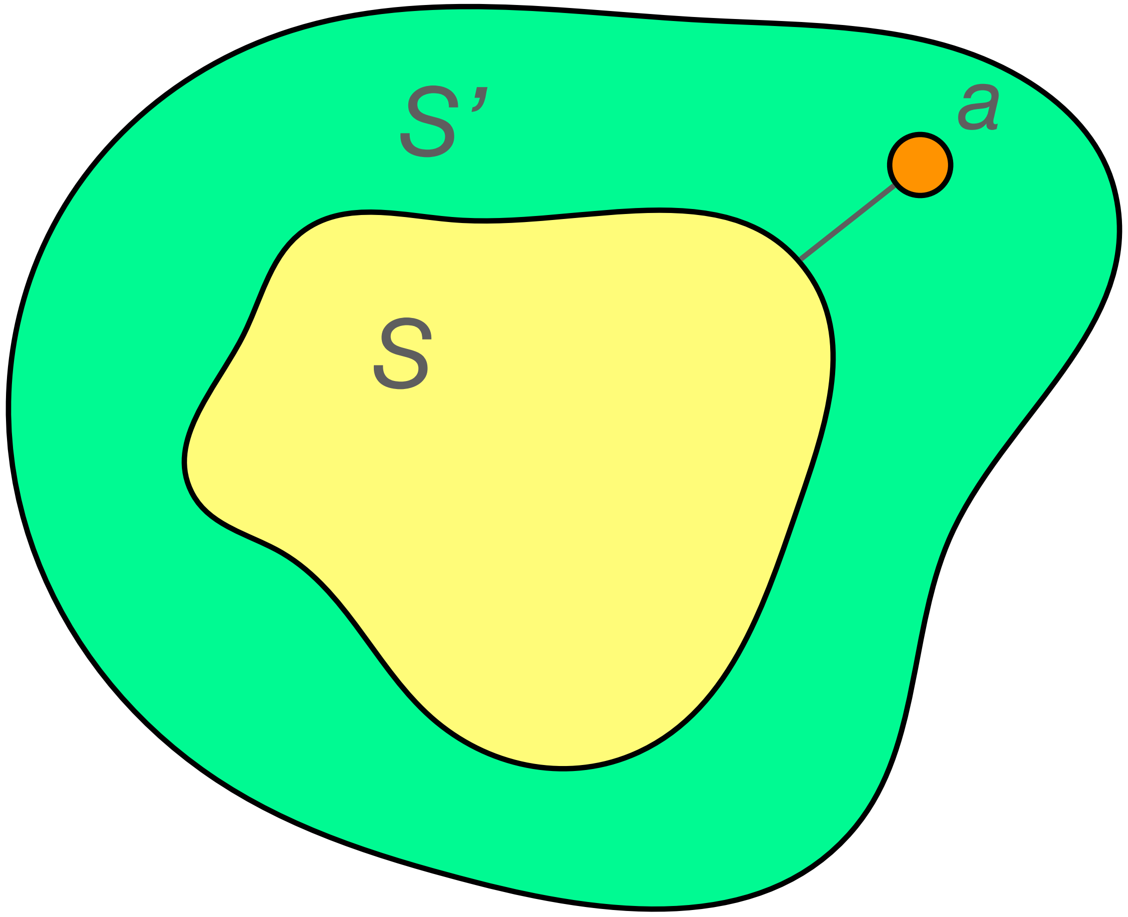
pp. 684-686

- Start: $S = \emptyset$
  - (1) is true, since there are no vertices in S.
  - (2) The only vertex for which $L(v)$ is not infinity is u_0 and for this vertex the L-value is correctly set to 0.

Induction Step

pp. 684-686

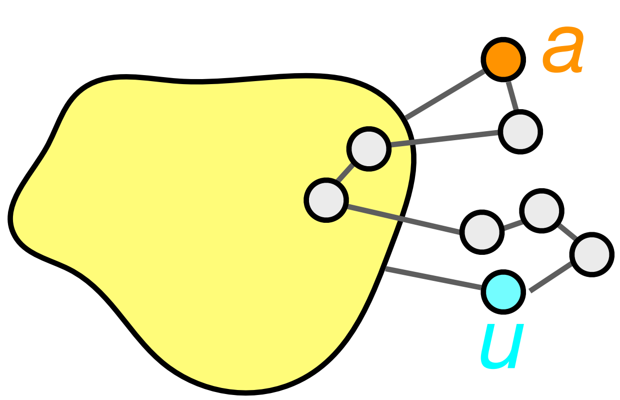
- Let $|S|=k$.
- At the $(k+1)$st iteration, a node a is chosen for which $L(a)$ is minimal, and it is added to the set S.
- The new set is $S' = S$ union $\{a\}$.
- We need to prove (1) and (2) for S'.



Proof of (1) for *S'*

pp. 684-686

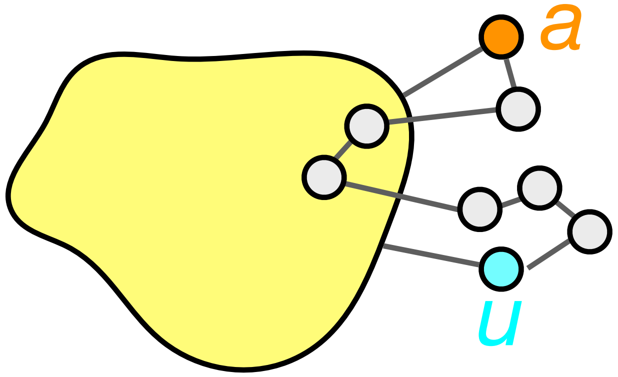
- Need to show: for all $u \in S'$: $L(u)$ is length of shortest path from u_0 to u.
- If $u \neq a$, then this is true by induction hypothesis
- Suppose that $u=a$, i.e., we need to show that $L(a)$ is length of shortest path from u_0 to a.
- If not, then shortest path has some length $c < L(a)$.
- This path will not be entirely in *S*
  - By induction hypothesis, $L(a)$ is length of shortest path to *a* that is entirely in *S* (Condition (2)).
- Therefore, there is node *u* on this shortest path that is outside of *S*, and $u \neq a$.



Proof of (1) for *S'*

pp. 684-686

- Note that $L(u) \geq L(a)$
  - Because *a* was chosen to have smallest *L*-value
- Let u_0-u_1-…-u-v_1-…-v_t-a be a shortest path (of total length *c*) from u_0 to *a*.
- By induction hypothesis $L(u)$ is length of shortest path to *u* with vertices in *S*, hence length of shortest path, i.e., *c*, equals $L(u)+w(u,v_1)+\ldots+w(v_t,a)$.
- Because the edge weights are positive, $c > L(u)$.
- By hypothesis, $c < L(a)$, but this is a contradiction to $L(a) \leq L(u)$.
- So $L(a)$ is length of shortest path
- Proof of (1) for *S'* is complete.



Proof of (2) for *S'*

pp. 684-686

- We need to show: For all $v \notin S'$ $L(v)$ is length of shortest path from u_0 to v in which all nodes (except for v) are in S'.
- We distinguish two cases.
  - Case 1: shortest path to v does not pass through a.
    - ➡ In this case the assertion is true by induction hypothesis.
  - Case 2: shortest path to v passes through a.
    - ➡ Value $L(v)$ is defined as min(old $L(v)$, $L(a)+w(\{a,v\})$).
    - ➡ This is $L(a) + w(\{a,v\})$, since path passes through a.
    - ➡ Length of path cannot be smaller than this value, since $L(a)$ is length of shortest path to a.
      - ◉ Shorter path would mean that $L(a)$ is not length of shortest path.
  - This finishes proof of (2) for *S'*. *QED*

Running Time

- Dijkstra's algorithm uses $O(|V|^2)$ operations.
  - At every iteration of the loop one node is added to the set S, so in total $|V|$ iterations.
  - At iteration k, the L-value of at most $|V|-k$ other nodes is updated.
  - Total number of updates is therefore at most
    - ➡ $(|V|-1)+ (|V|-21)+ (|V|-13)+\ldots.. +1 = O(|V|^2)$