Lecture 13: Weighted Shortest Paths

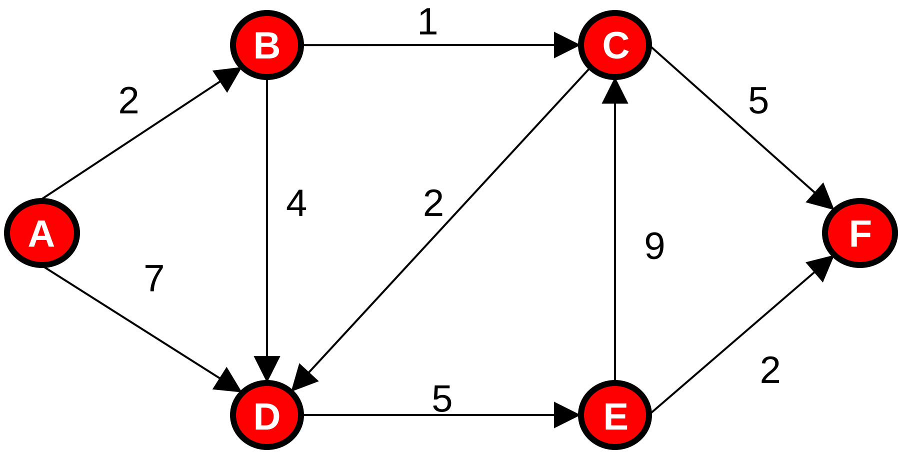
These slides include material originally prepared by Dr. Ron Cytron, Dr. Jeremy Buhler, and Dr. Steve Cole.

Announcements

- Lab 13 released tomorrow – Dijkstra’s algorithm
- Studio 13 Thursday
- Exam 3 May 1st 10 am – 12 pm

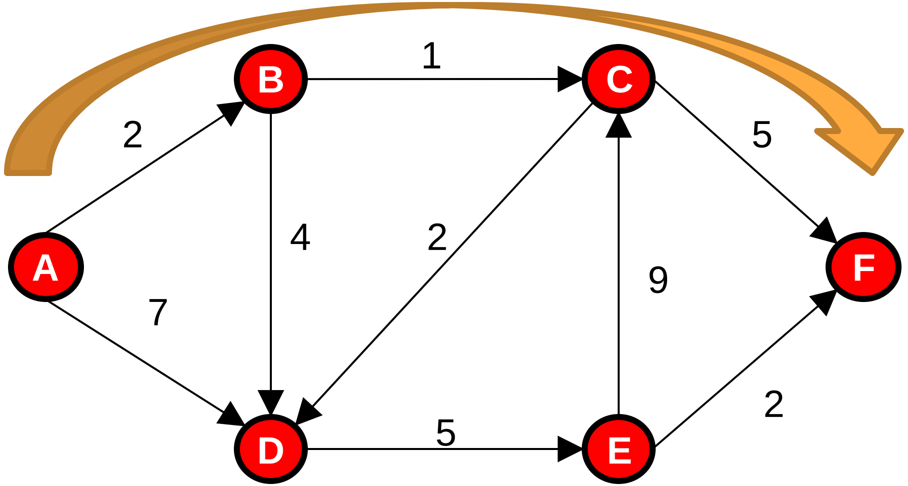
Adding Weights to Graphs

- A weighted graph assigns to each edge e a real-valued **weight** w(e)
- For today, we will assume that **w(e) ≥ 0**.



Path Lengths

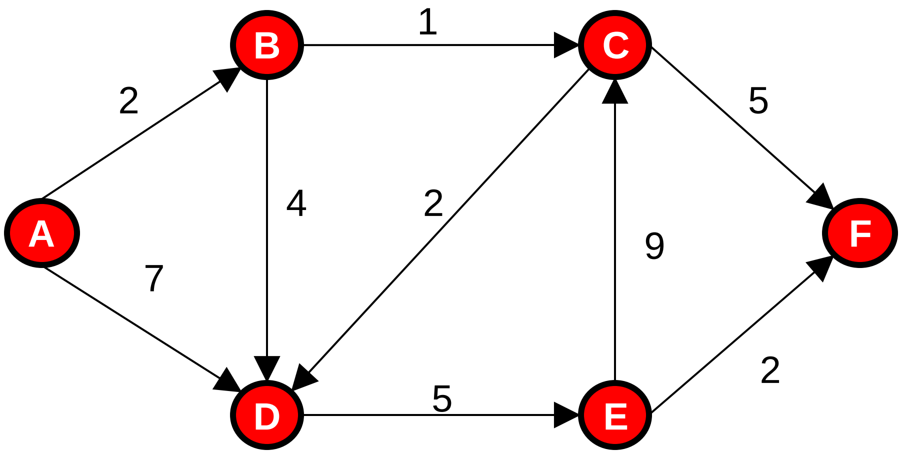
- The **length** (or total weight) of a path is the sum of its edges' weights



Total weight (ABCF)
= 2+1+5=8

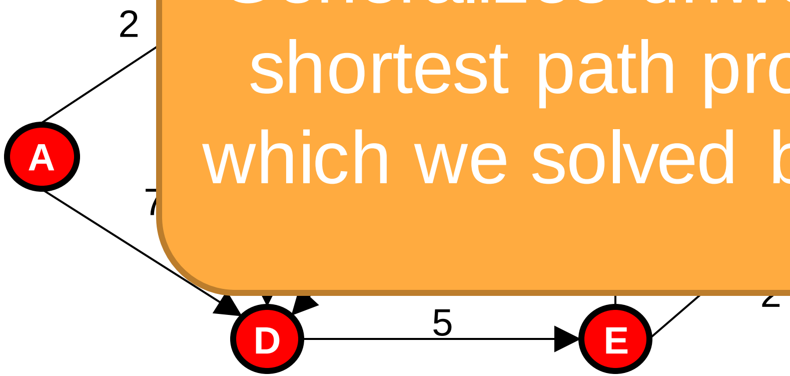
Weighted Shortest Paths

- **Problem**: given a starting vertex v, *find path of* ***least total weight*** from v to each other vertex in the graph.



Weighted Shortest Paths

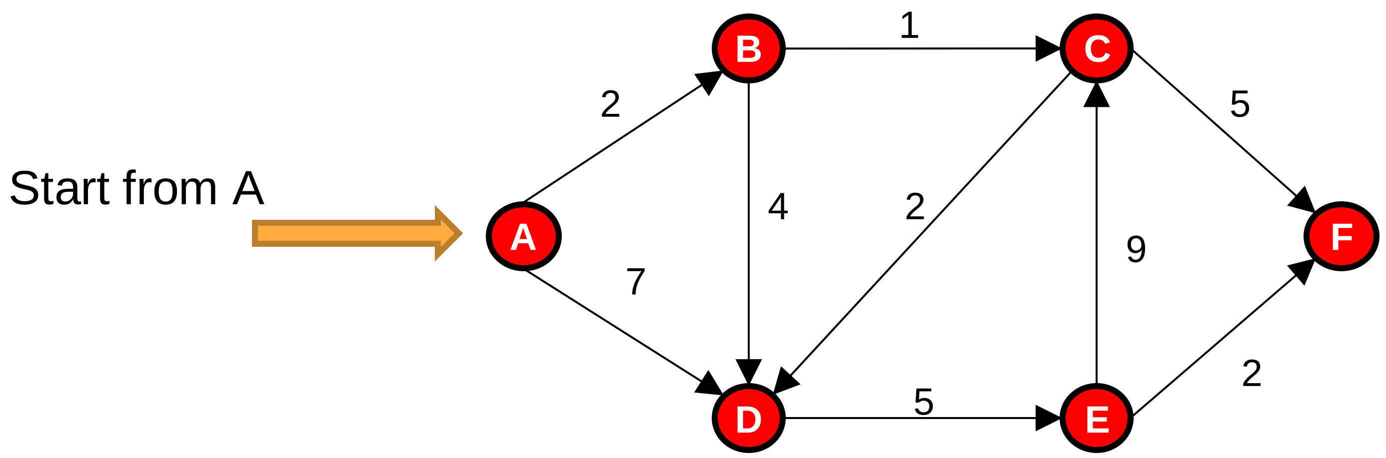
- **Problem**: given a starting vertex v, *find path of* ***least total weight*** from v to each other vertex in the graph.



Generalizes unweighted shortest path problem, which we solved by BFS.

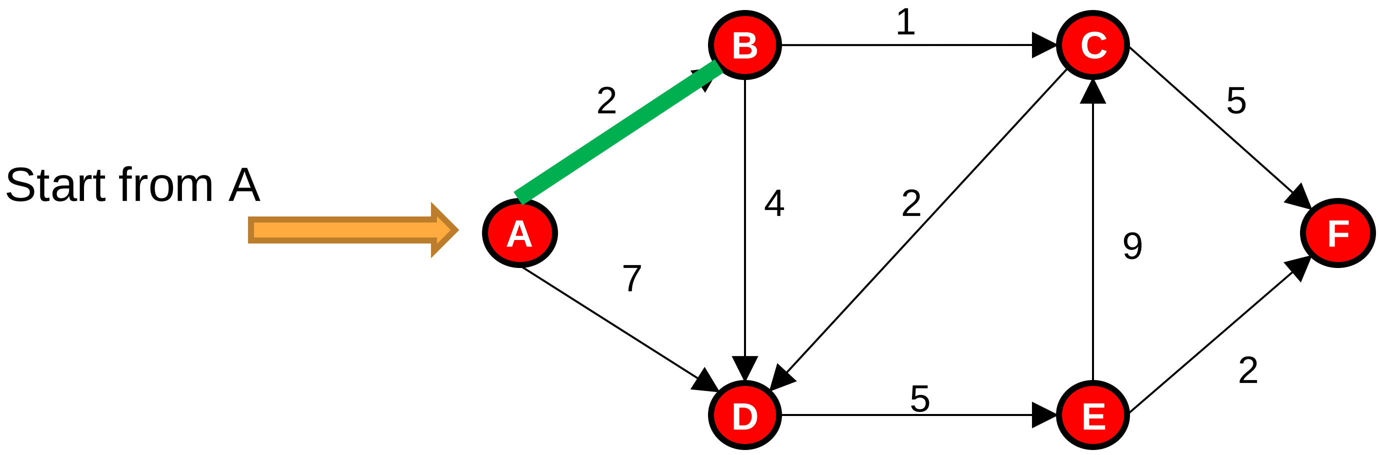
Weighted Shortest Paths

- Given a starting vertex v, find path of least total weight D(v,u) from v to each other vertex u in the graph.



Weighted Shortest Paths

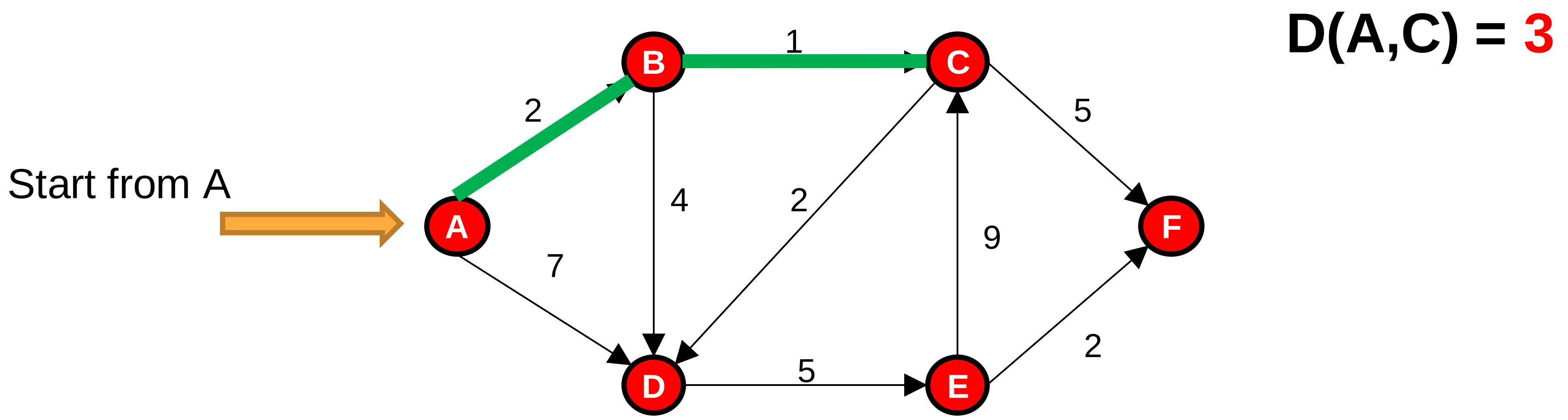
- Given a starting vertex v, find path of least total weight D(v,u) from v to each other vertex u in the graph.



D(A,B) = 2

Weighted Shortest Paths

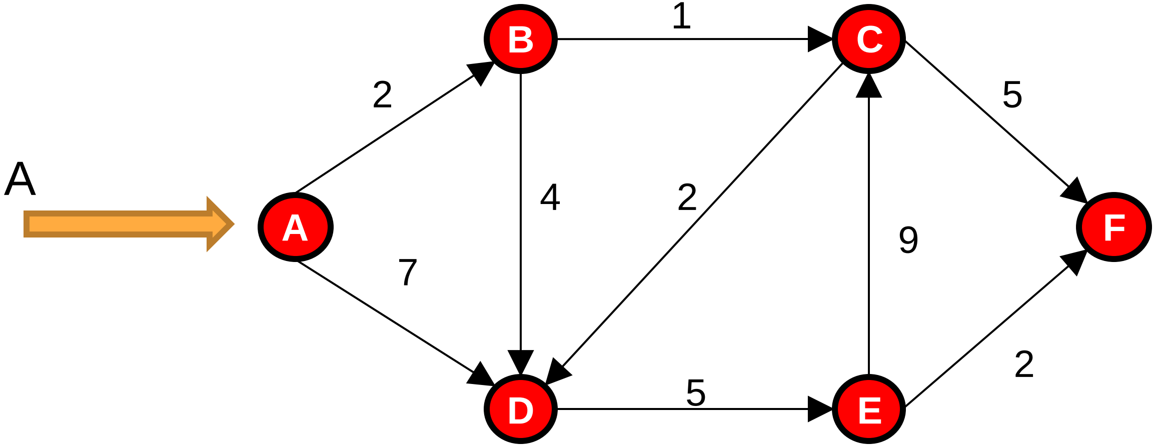
- Given a starting vertex v, find path of least total weight D(v,u) from v to each other vertex u in the graph.



Weighted Shortest Paths

- Given a starting vertex v, find path of least total weight D(v,u) from v to each other vertex u in the graph.

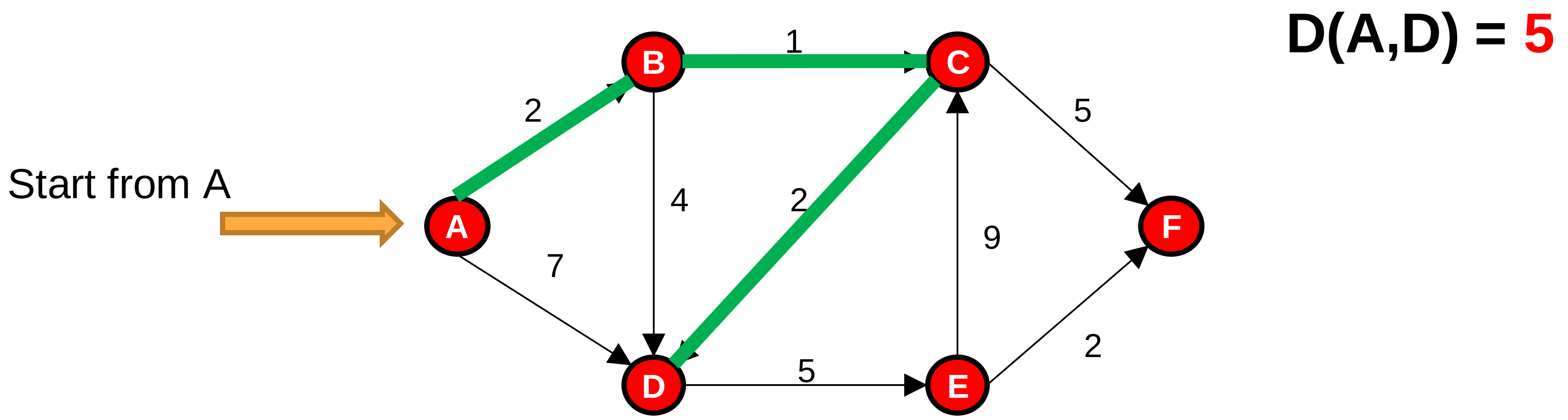
Start from A



D(A,D) = ???

Weighted Shortest Paths

- Given a starting vertex v, find path of least total weight D(v,u) from v to each other vertex u in the graph.

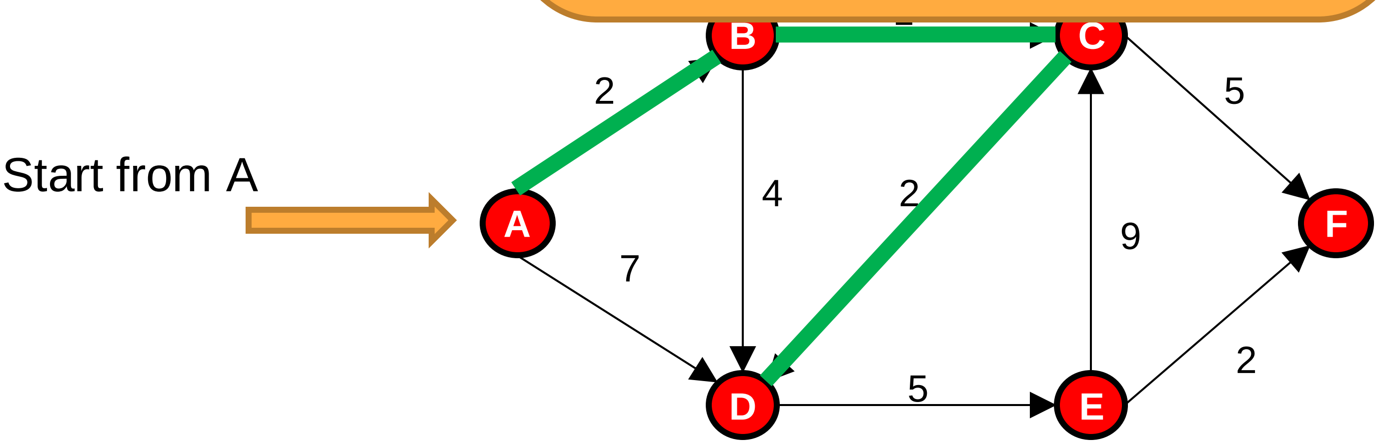


Weighted Sh

- Given a starting ... nt D(v,u) from v to each other ve

Weighted shortest path to a vertex may not be one with fewest edges.

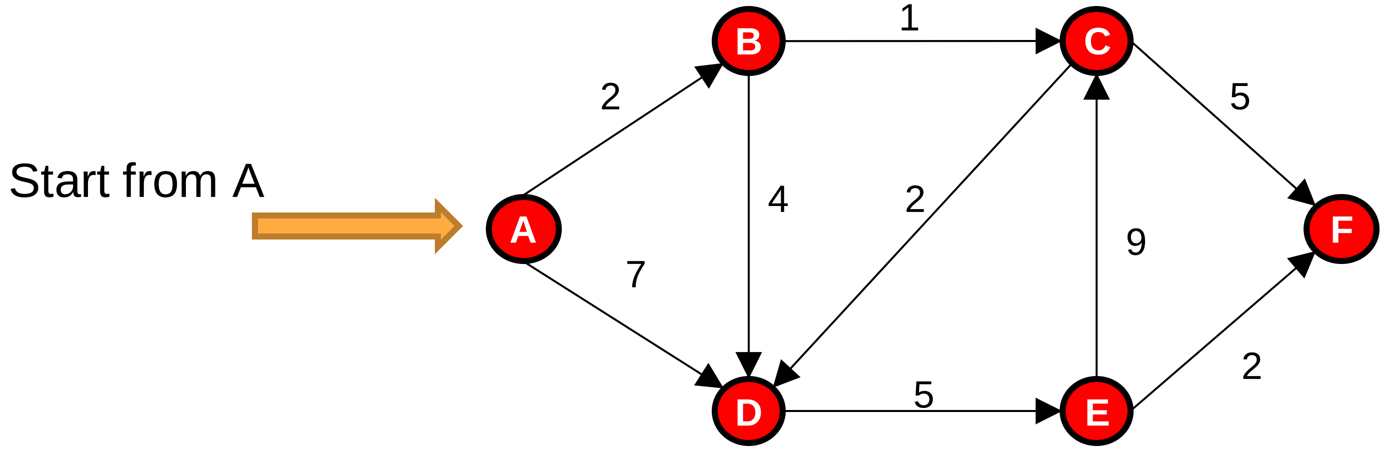
D(A,D) = 5



Weighted Shortest Paths

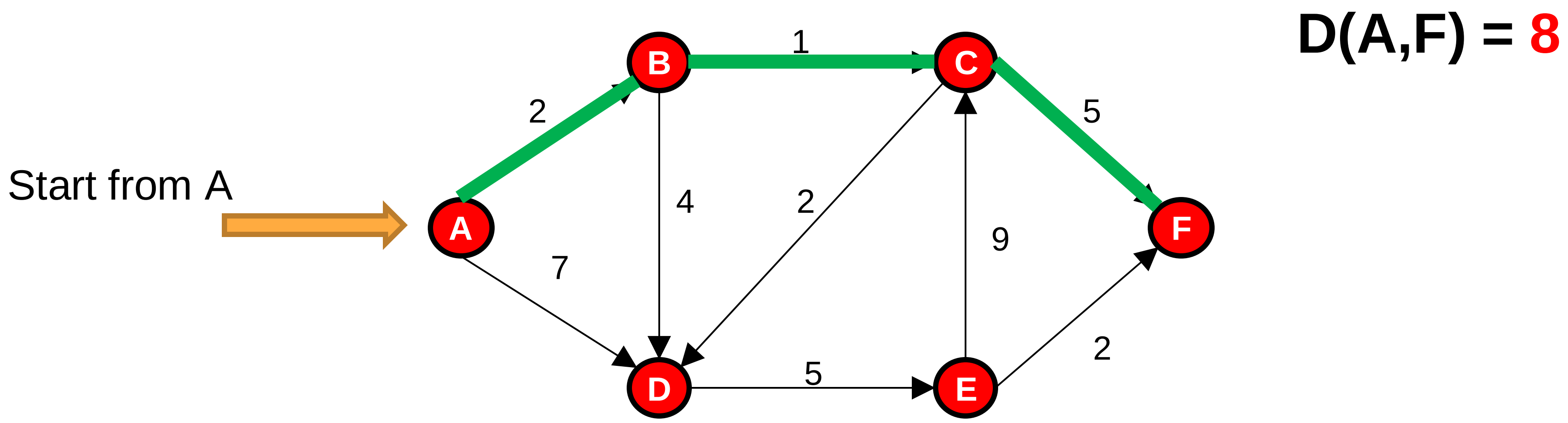
- Given a starting vertex v, find path of least total weight D(v,u) from v to each other vertex u in the graph.

D(A,F) = ???



Weighted Shortest Paths

- Given a starting vertex v, find path of least total weight D(v,u) from v to each other vertex u in the graph.

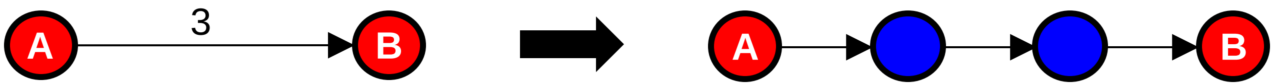


Why Solve Weighted Shortest Paths?

- Road map with distances between cities – shortest route
- Routing network with cost to each hop – cheapest way to send data
- State-space search in AI with action costs (A* search)
- …

How Can We Solve Weighted Shortest Paths?

- Could reduce to unweighted problem:



- Apply to every edge; use BFS on resulting graph
- Only works if weights are integers
- Would be very expensive for graphs with large weights

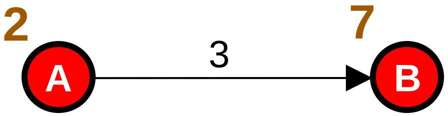
Alternate Strategy – "Relaxation"

- Explore graph while maintaining, for each vertex v, length of shortest path to v seen *so far*. Store this shortest path estimate as **v.dist**.

- Whenever we follow an edge (v,u), check whether

$$v.dist + w(v,u) < u.dist$$

- If so, *we've found a new, shorter path to u via v.*



Old B.dist = 7

- Update v.dist with new estimate and continue

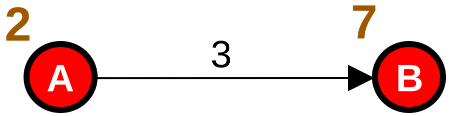
Alternate Strategy – “Relaxation”

- Explore graph while maintaining, for each vertex v, length of shortest path to v seen *so far*. Store this shortest path estimate as **v.dist**.

- Whenever we follow an edge (v,u), check whether

$$v.dist + w(v,u) < u.dist$$

- If so, *we’ve found a new, shorter path to u via v.*



2+3=5 < 7

- Update v.dist with new estimate and continue

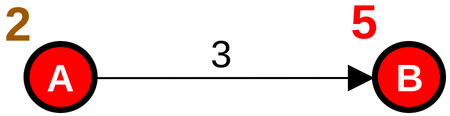
Alternate Strategy – “Relaxation”

- Explore graph while maintaining, for each vertex v, length of shortest path to v seen *so far*. Store this shortest path estimate as **v.dist**.

- Whenever we follow an edge (v,u), check whether

$$v.dist + w(v,u) < u.dist$$

- If so, *we’ve found a new, shorter path to u via v.*



B.dist ← 5

- Update v.dist with new estimate and continue

Who Gets to Relax, When?

- Need to decide which vertices to relax at each step.
- Also, need to know when we are done!
- **Proposal** (*E. Dijkstra* *): at each step, explore edges out of vertex v with *smallest* v.dist, and relax all its adjacent vertices.
- Stop when each vertex has had its outgoing edges explored once.

* Fun fact: font on the title slide is Dijkstra's handwriting!

Dijkstra’s Shortest Path Algorithm

- starting vertex v gets v.dist ← 0; all other u get u.dist ← ∞
- mark all vertices as unfinished

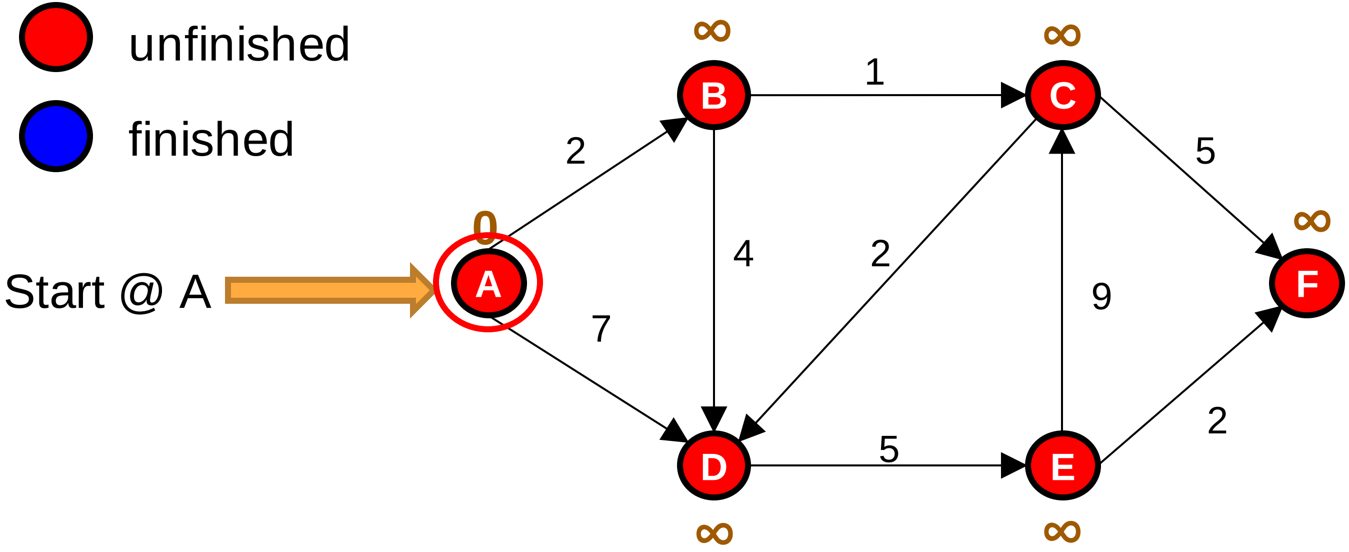
- while (*any vertex unfinished*)
-     v ← unfinished vertex with smallest v.dist
-     for each edge (v,u)
-         if (v.dist + w(u,v) < u.dist)
-             u.dist ← v.dist + w(u,v)   // relax!
-     mark v finished

Example of Dijkstra’s Algorithm

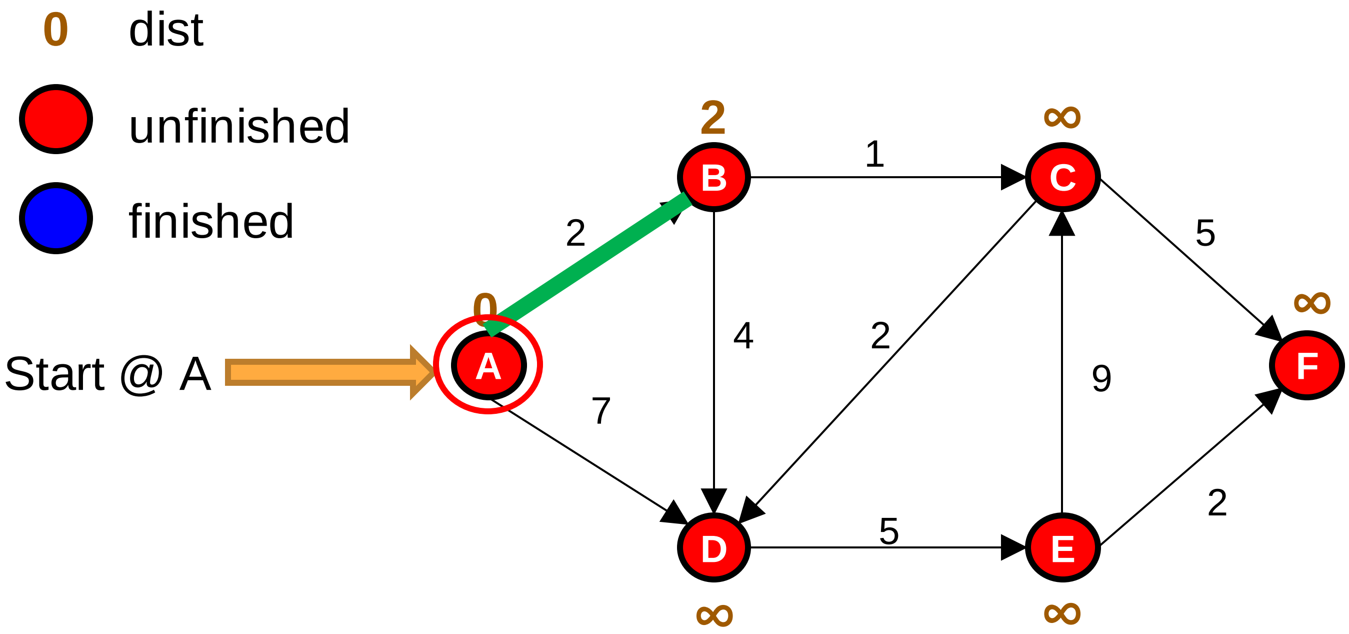
0 dist

unfinished

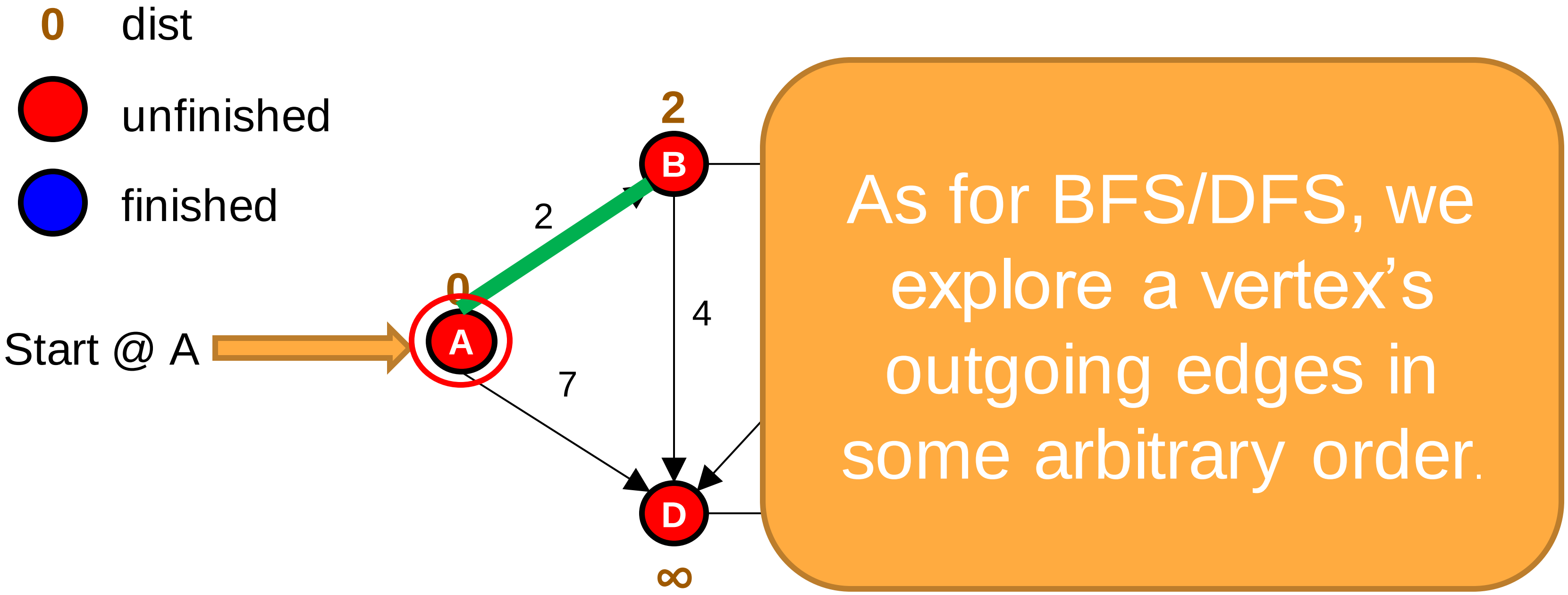
finished



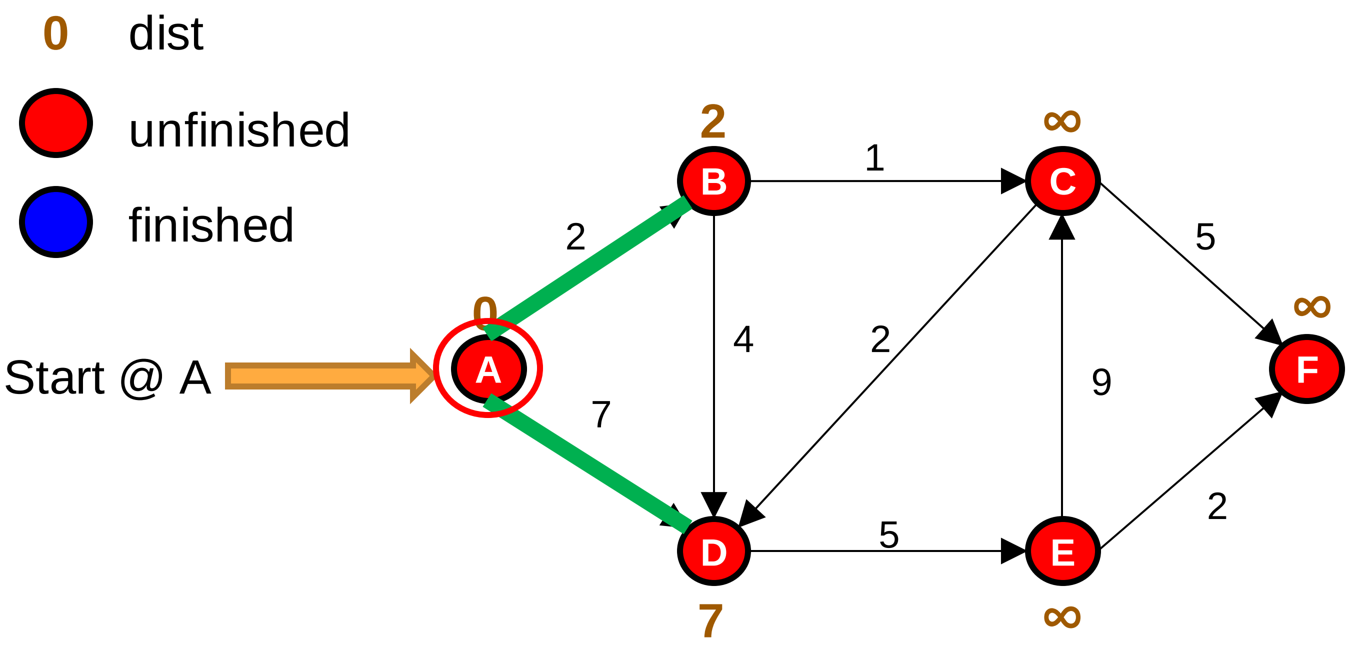
Example of Dijkstra’s Algorithm



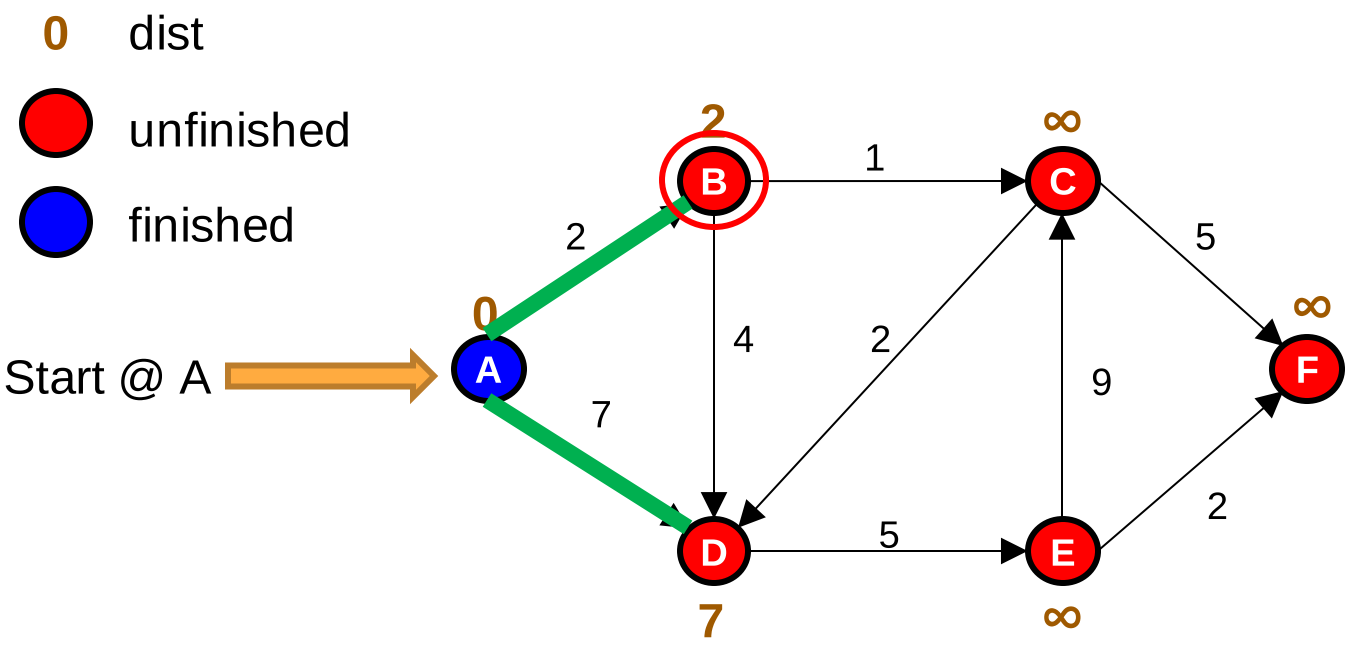
Example of Dijkstra's Algorithm



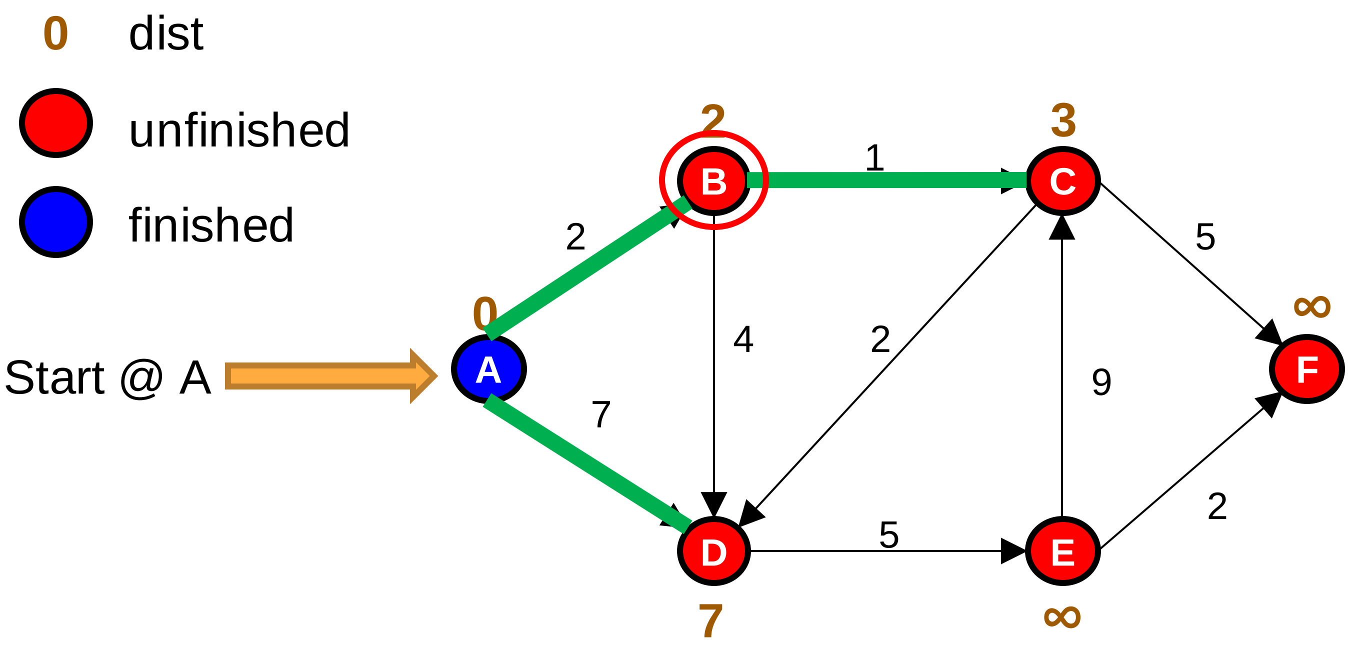
Example of Dijkstra's Algorithm



Example of Dijkstra's Algorithm

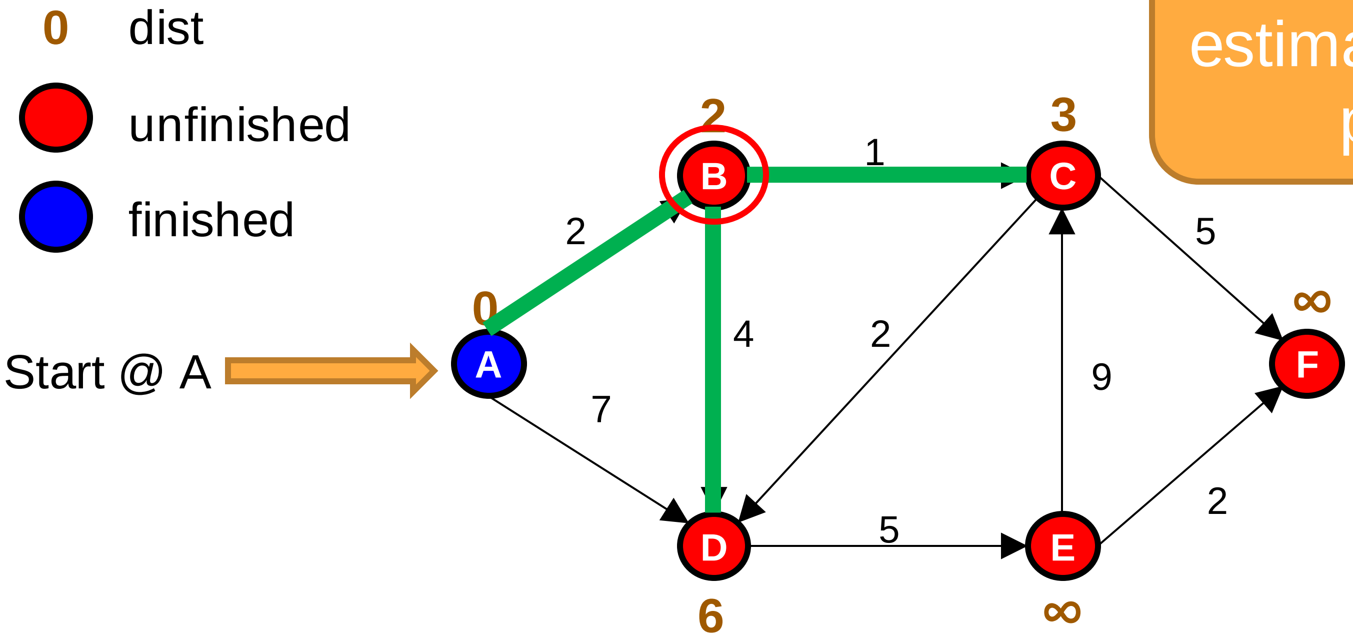


Example of Dijkstra's Algorithm

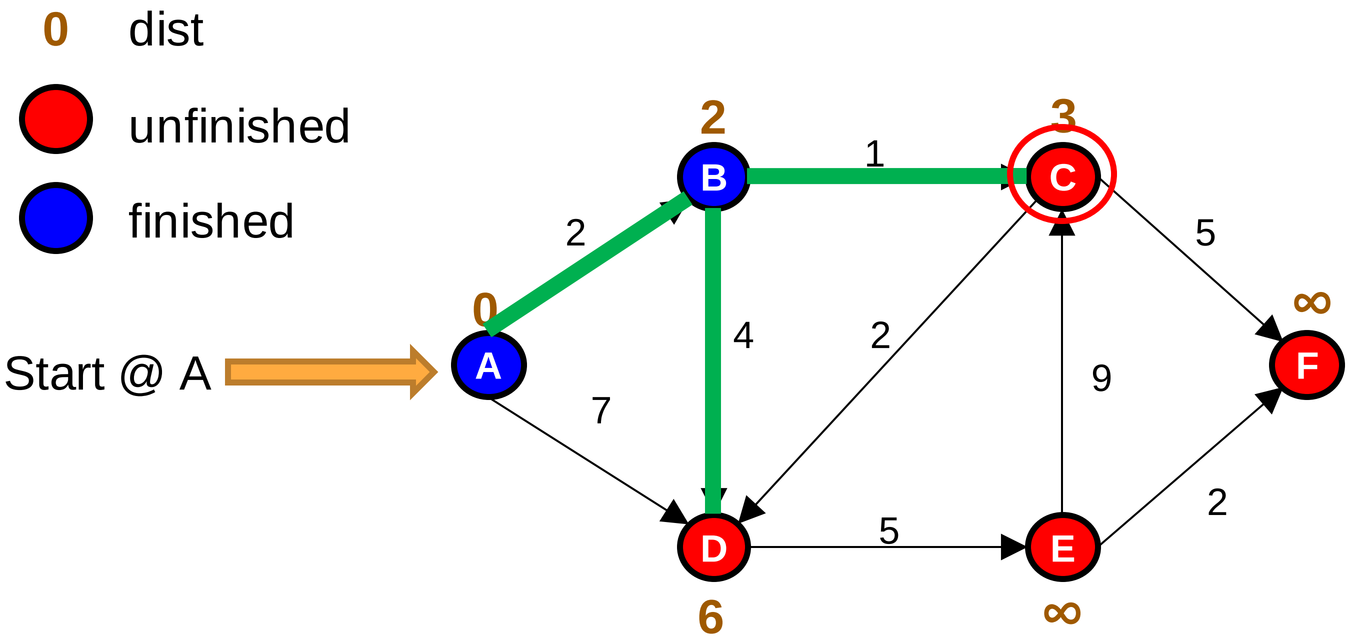


Example of Dijkstra's Algorithm

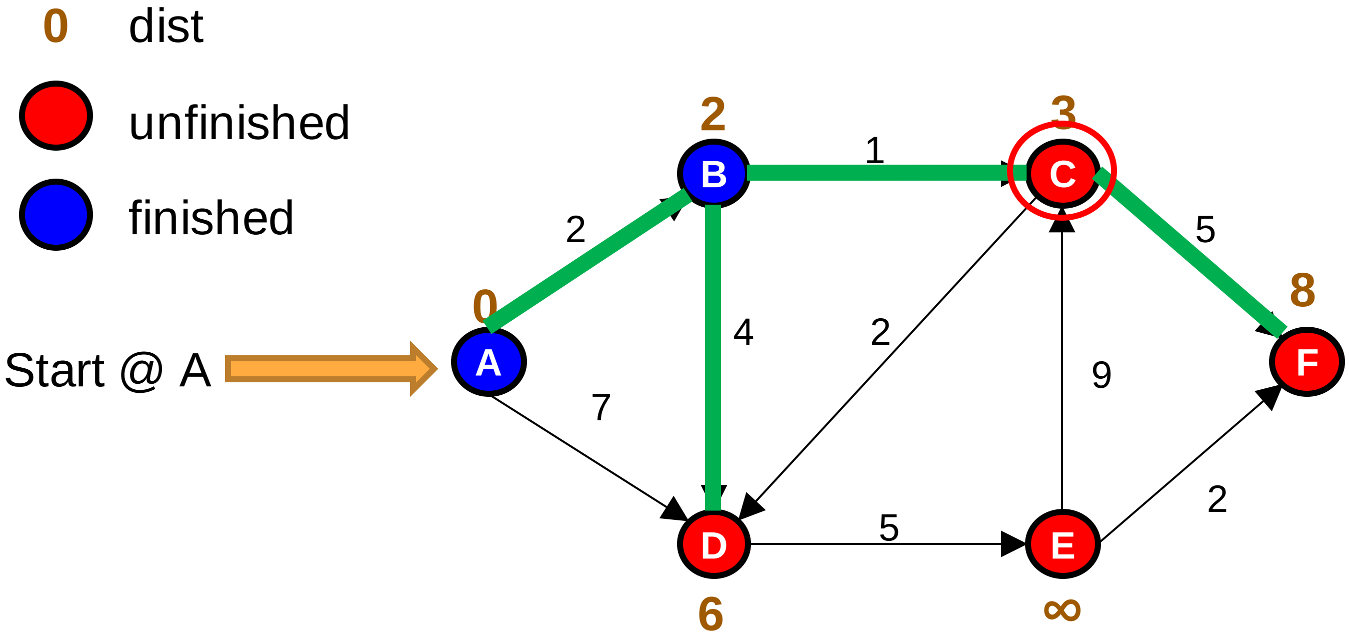
Relaxation can change estimated distance and parent edge!



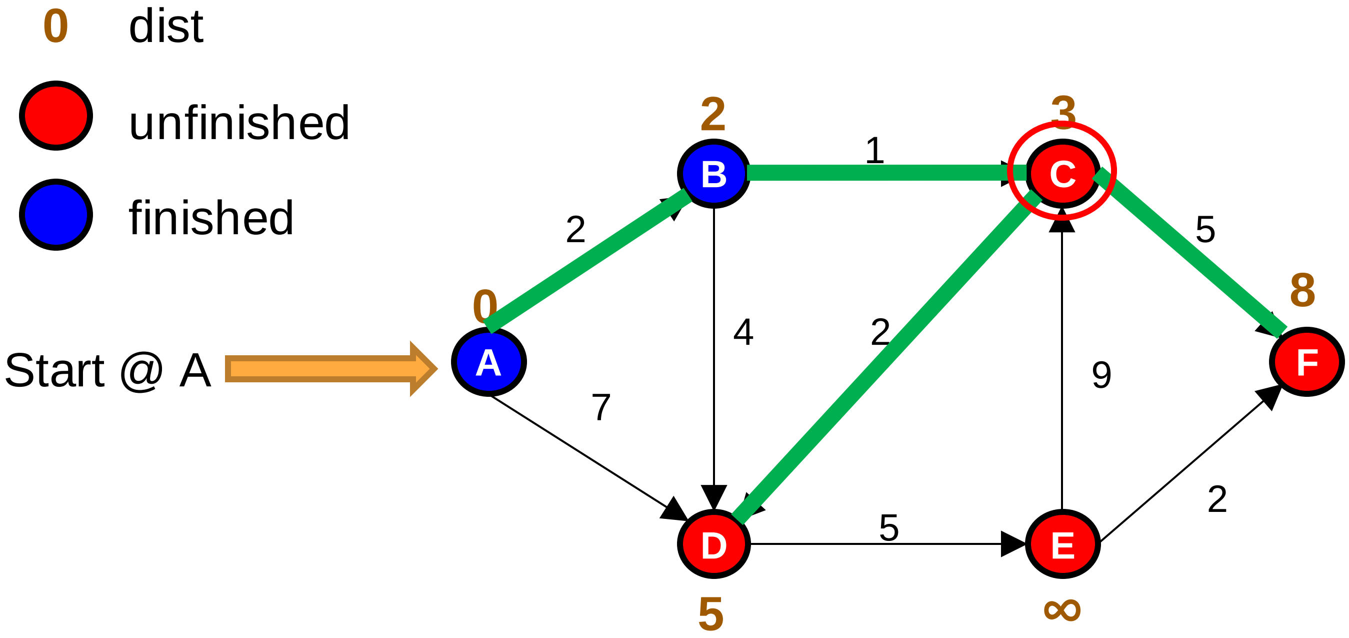
Example of Dijkstra's Algorithm



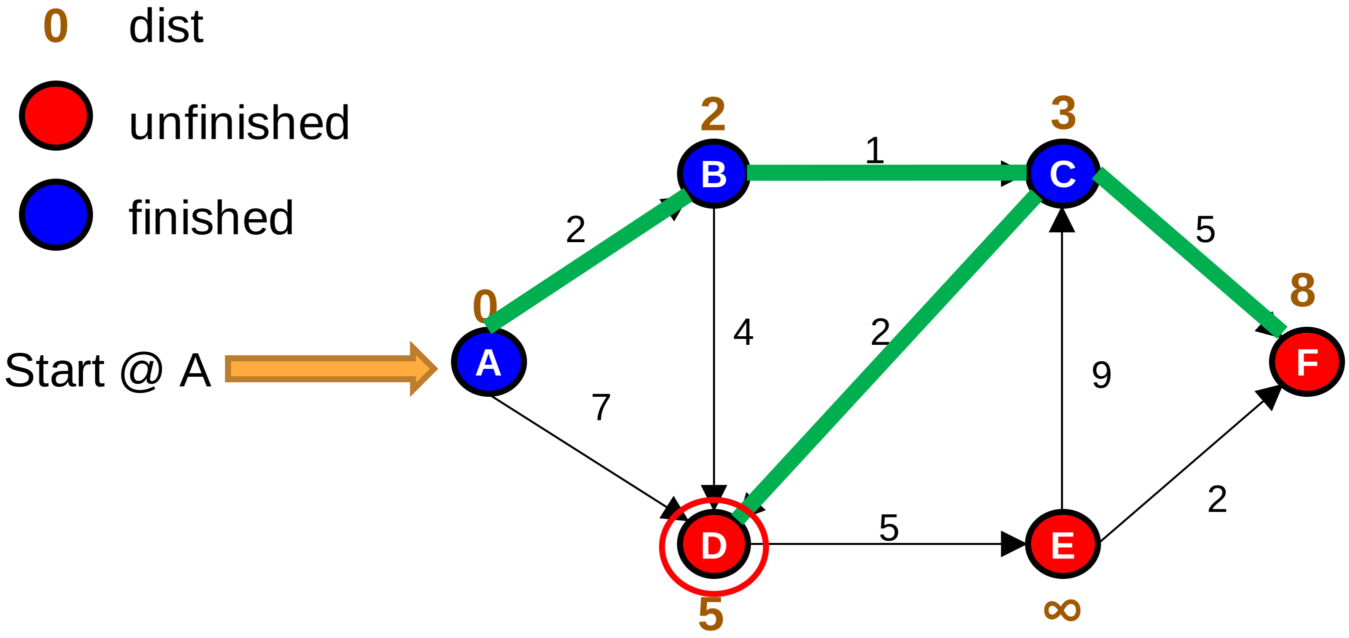
Example of Dijkstra's Algorithm



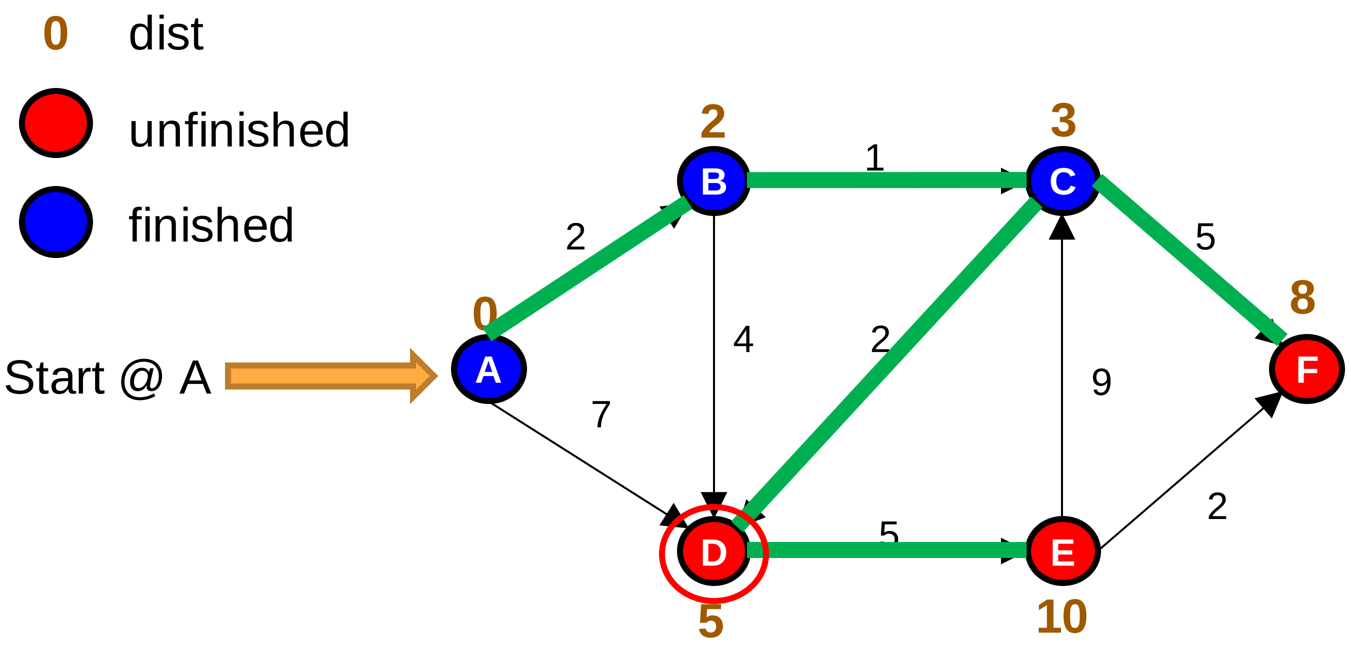
Example of Dijkstra's Algorithm



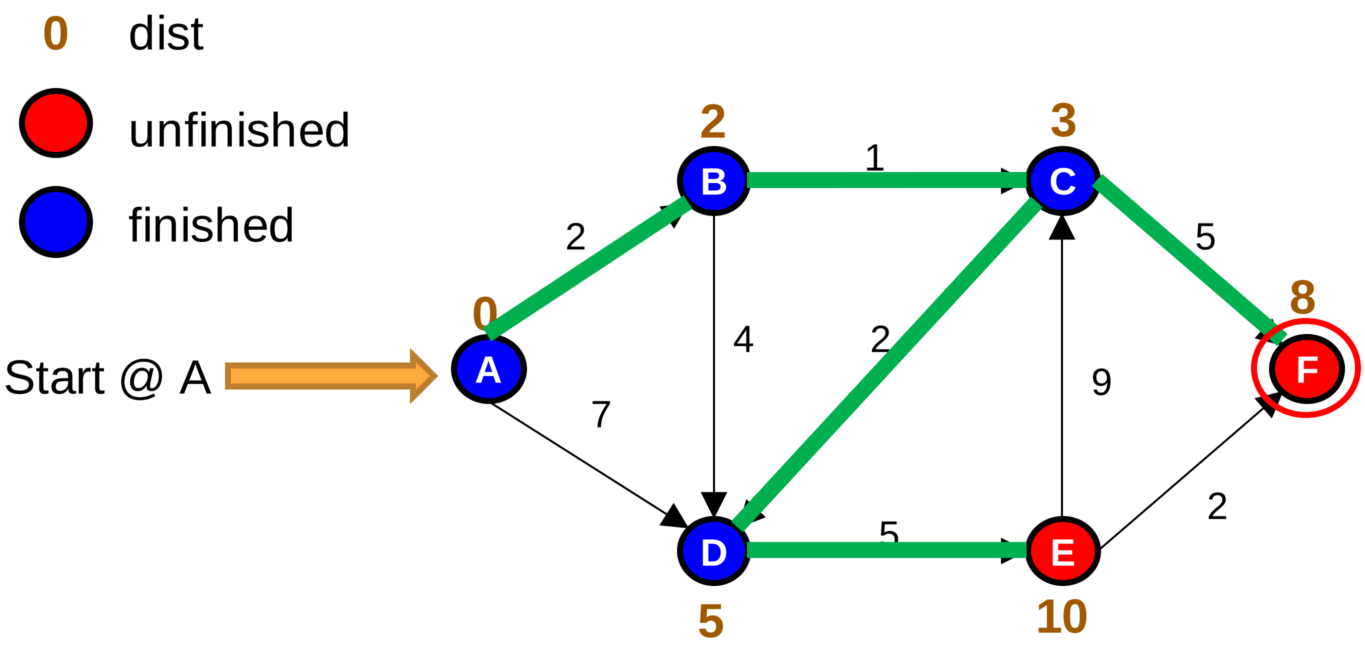
Example of Dijkstra's Algorithm



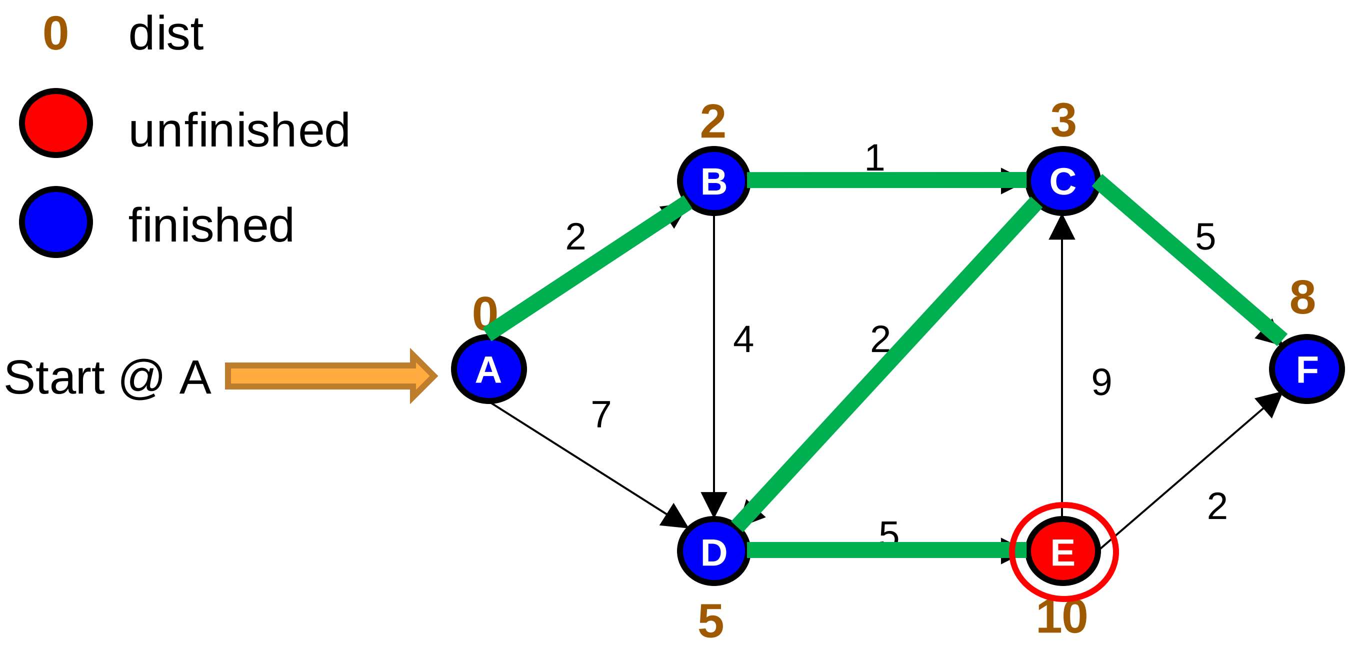
Example of Dijkstra’s Algorithm



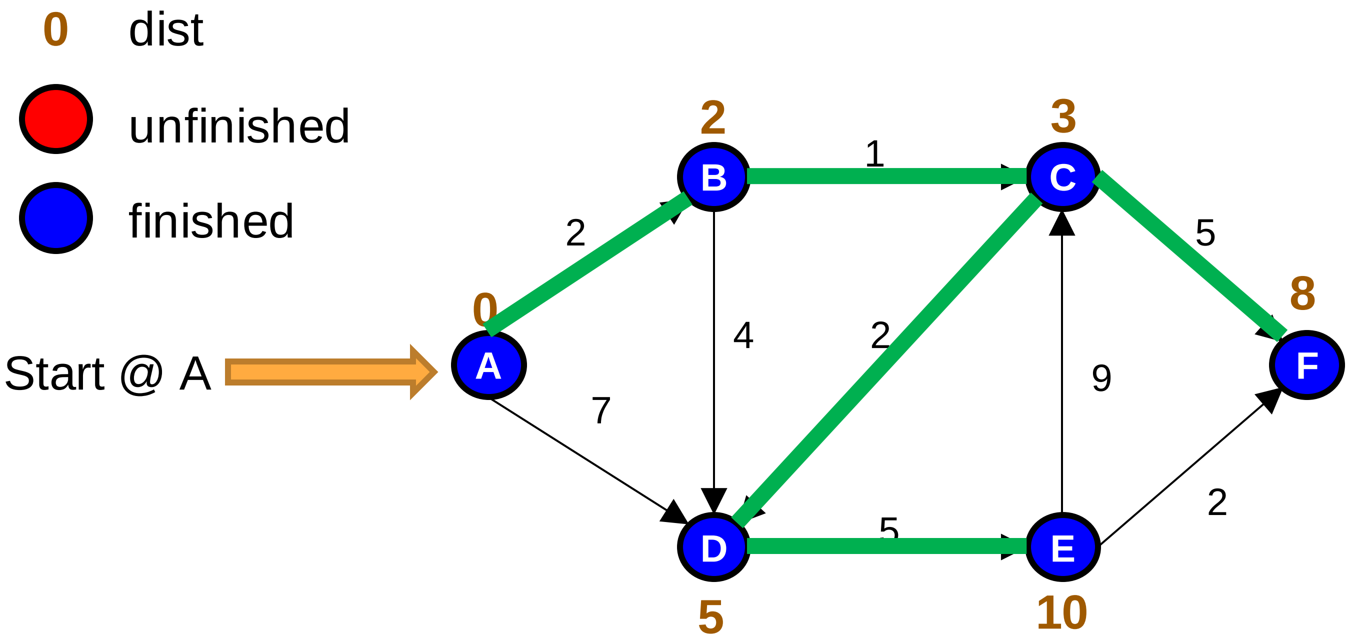
Example of Dijkstra's Algorithm



Example of Dijkstra's Algorithm



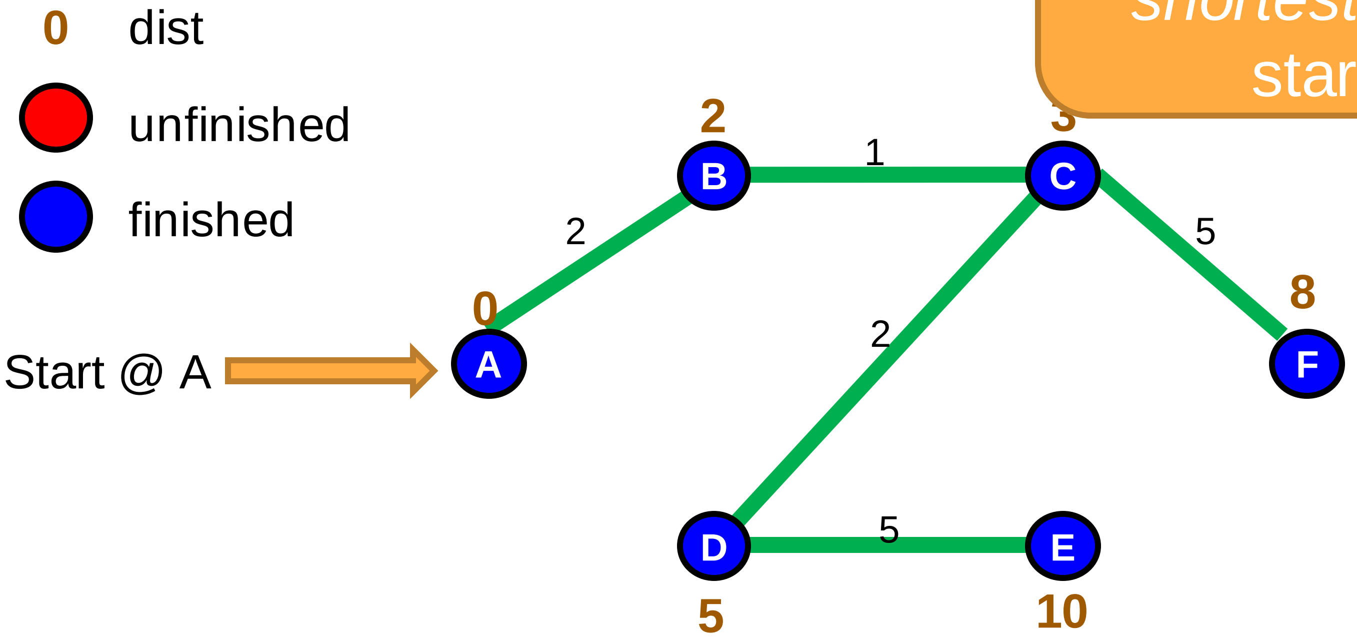
Example of Dijkstra's Algorithm



DONE!

Example of Dijkstra's Algor

As with BFS, parent edges from Dijkstra's algo form a *shortest-path tree* from starting vertex.



Correctness of Dijkstra’s Algorithm

- **Claim**: when we explore the edges out of vertex v, v has its correct shortest-path distance D(start, v) stored in current best estimate v.dist.

- **Pf**: by induction on order of exploration.

- **Bas**: starting vertex is explored first, with its correct shortest-path distance of 0.

Correctness of Dijkstra’s Algorithm

- **Claim**: when we explore the edges out of vertex v, v has its correct shortest-path distance D(start, v) stored in current best estimate v.dist.

- **Ind:** suppose the algorithm is about to choose v for exploration.

- Assume that v.dist > D(start, v) (i.e. v’s distance is wrong).

Correctness of Dijkstra's Algorithm

- **Claim**: when we explore the edges out of vertex v, v has its correct shortest-path distance D(start, v) stored in current best estimate v.dist.

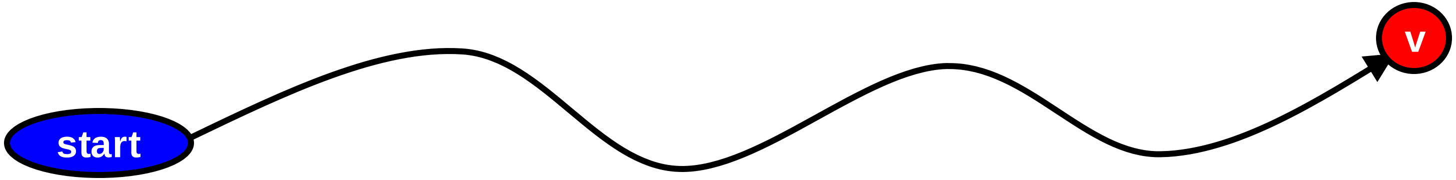
- **Ind:** suppose the algorithm is about to choose v for exploration.

- Assume that v.dist > D(start, v) (i.e. v's distance is wrong).

- *[we will derive a contradiction… hence, v.dist must be = D(start, v)]*

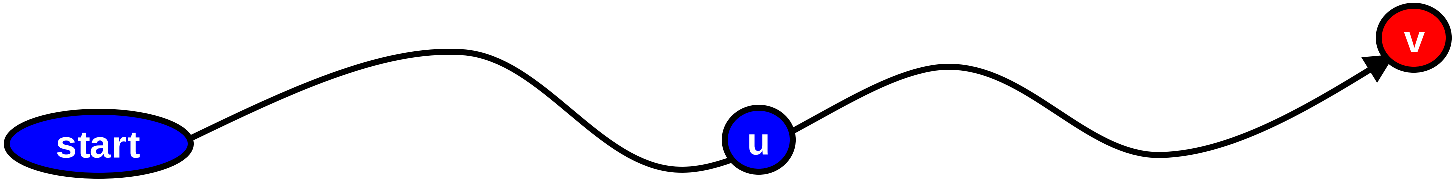
Correctness of Dijkstra's Algorithm

- **Ind:** suppose the algorithm is about to choose v for exploration.
- Assume that v.dist > D(start, v) (i.e. v's distance is wrong).
- Consider a *shortest* path from start to v.



Correctness of Dijkstra's Algorithm

- **Ind:** suppose the algorithm is about to choose v for exploration.
- Assume that v.dist > D(start, v) (i.e. v's distance is wrong).
- Consider a *shortest* path from start to v.



- Let u be last **finished** (i.e., already explored) vertex on this path.

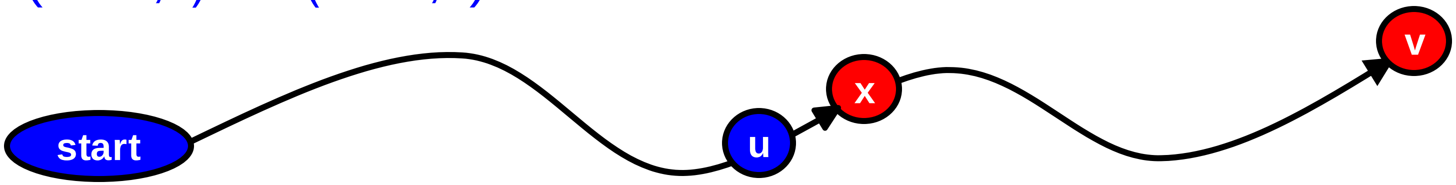
Correctness of Dijkstra's Algorithm

- By IH, u had its correct shortest-path distance when it was explored.
- Moreover, D(start, u) ≤ D(start, v), since *u precedes v* on shortest path to v.
- If edge u → v is on shortest path, then exploring u's outgoing edges assigns v its correct shortest-path distance D(start,v). →←

Correctness of Dijkstra's Algorithm

- If edge u → v is on shortest path, then exploring u's outgoing edges assigns v its correct shortest-path distance D(start,v). →←

- Otherwise, some *other* vertex x lies between u and v on this path, with D(start,x) ≤ D(start,v).



- Since v does *not* have its correct shortest-path distance, **v.dist > x.dist**, and so x would be explored next, not v. →← **QED**

How Do We Track Next Vertex to Explore?

- Maintain collection of unfinished vertices
- At each step, must efficiently **find** vertex v in collection with smallest v.dist and **remove** it
- But vertices' distances may change repeatedly due to relaxation!
- Changes are all in one direction (*decrease*)

What Data Structure Can We Use To Track Distances to Unfinished Vertices?

Use a Priority Queue!

- Maintain priority queue PQ of unfinished vertices, keyed on **dist**
- Initially, every vertex is inserted into PQ w/its starting dist
- At each step, find next vertex to explore by PQ.extractMin()
- Decreasing v.dist is done using v's **`Decreaser`** object
- We assume a map D[ ] from vertices to their Decreasers

Dijkstra's Shortest Path Algorithm w/Prio Queue

- v.dist ← 0; D[v] ← PQ.insert(starting vertex v)
- For all other vertices u
- u.dist ← ∞; D[u] ← PQ.insert(u)
- while (PQ not empty)
- v ← PQ.extractMin()
- for each edge (v,u)
- if (v.dist + w(v, u) < u.dist)
- u.dist ← v.dist + w(v,u)
- D[u].decrease(u)

Dijkstra's Shortest Path Algorithm w/Queue

- v.dist ← 0; D[v] ← PQ.insert(starting vertex v)
- For all other vertices u
- u.dist ← ∞; D[u] ← PQ.insert(u)
- while (PQ not empty)
- v ← PQ.extractMin()
- for each edge (v,u)
- if (v.dist + w(v, u) < u.dist)
- u.dist ← v.dist + w(v,u)
- D[u].decrease(u)

Note that Lab 13 creates pairs (vertex, dist) rather than vertices with an internal distance field.

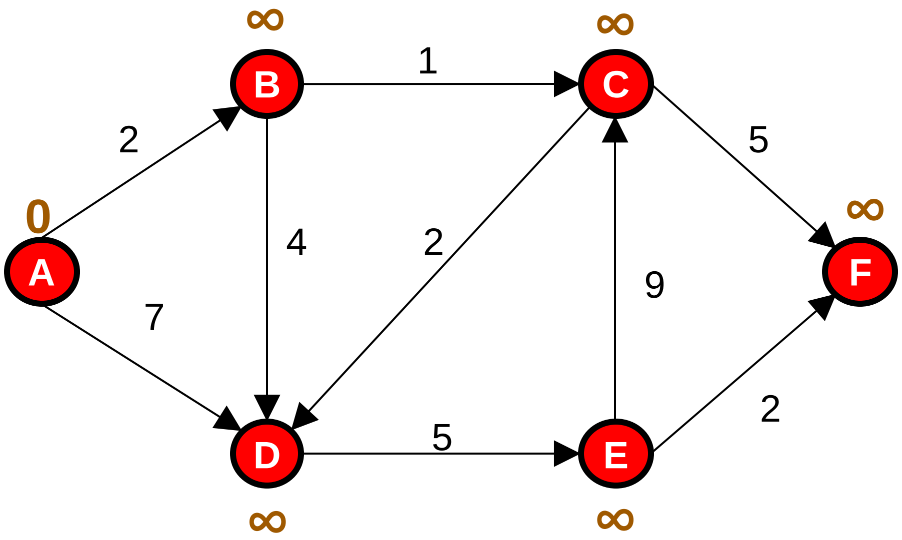
Example of Dijkstra's Algorithm

0 dist

unfinished

finished

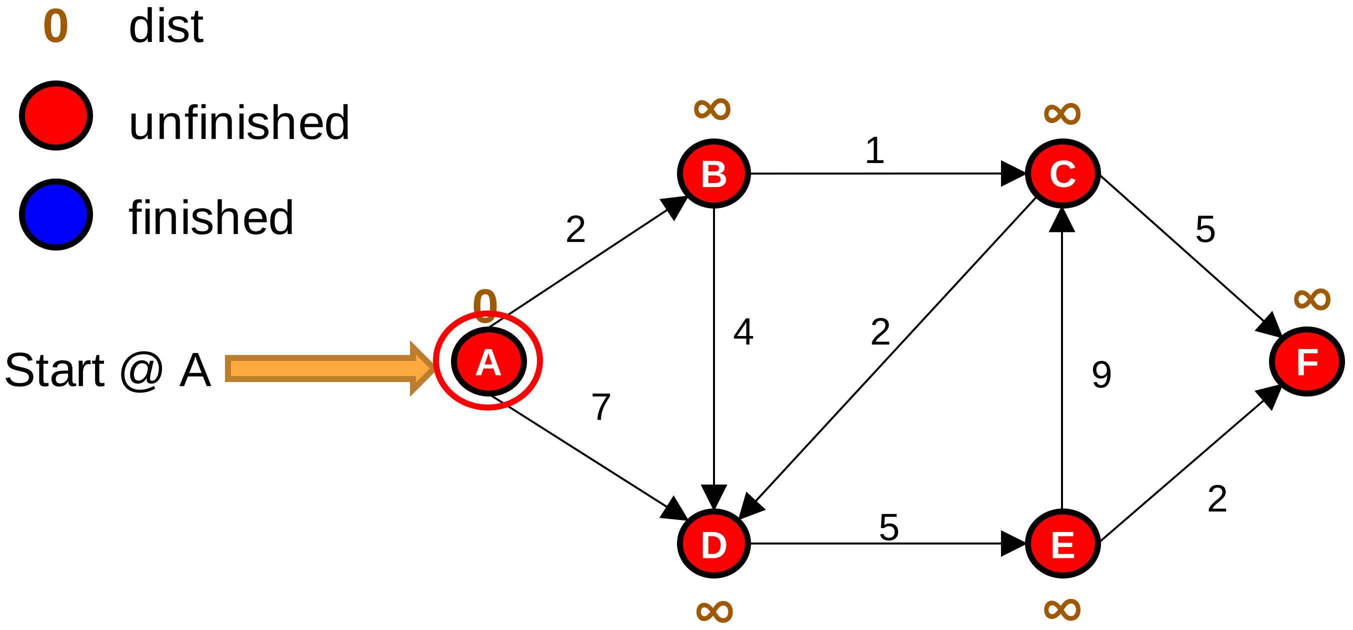
Start @ A



PQ

A	0
B	∞
C	∞
D	∞
E	∞
F	∞

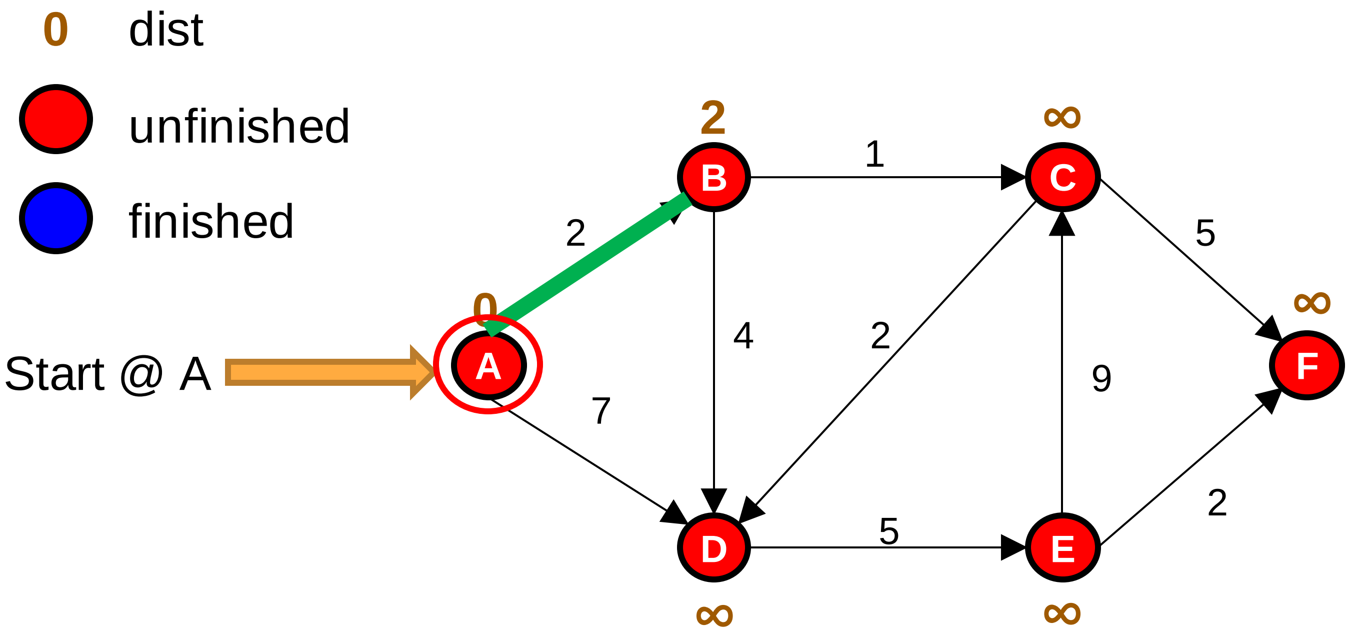
Example of Dijkstra's Algorithm



PQ

A		
	B	∞
	C	∞
	D	∞
	E	∞
	F	∞

Example of Dijkstra's Algorithm



PQ

B	2
C	∞
D	∞
E	∞
F	∞

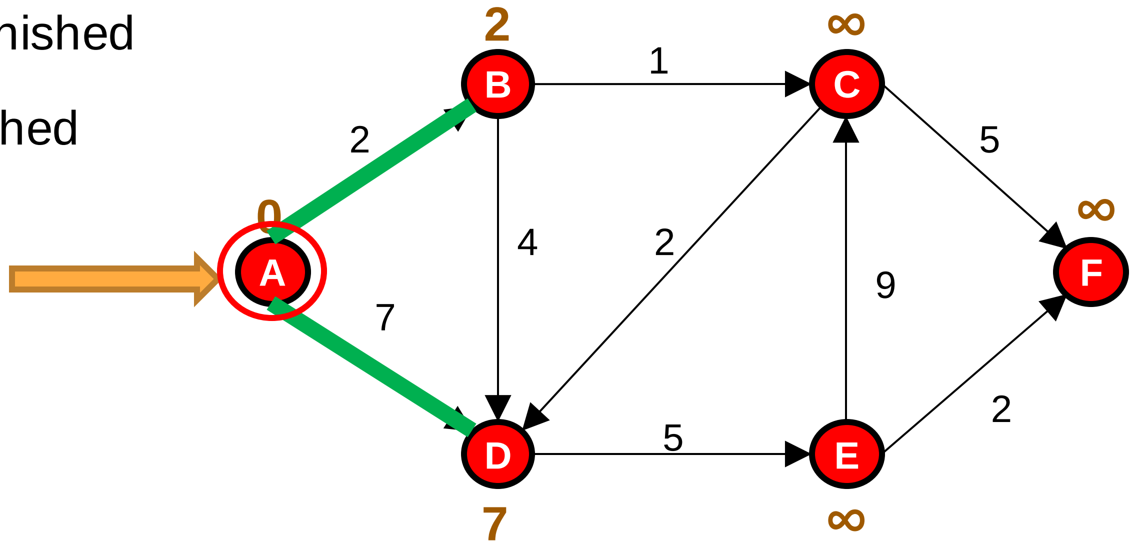
Example of Dijkstra's Algorithm

0 dist

unfinished

finished

Start @ A



PQ

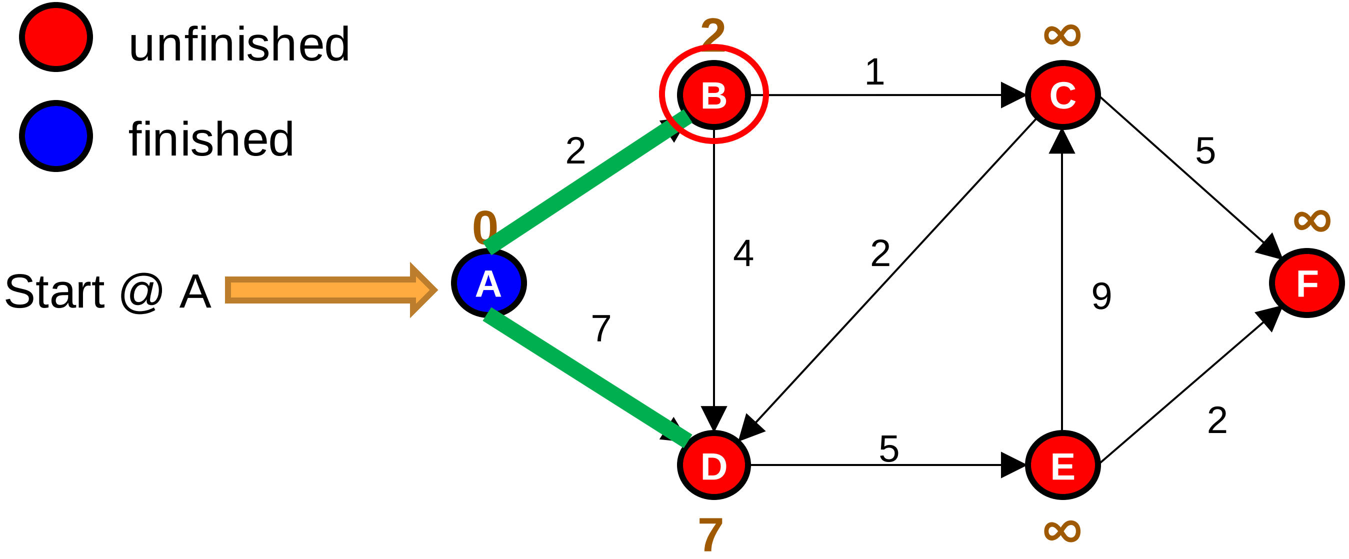
B	2
C	∞
D	7
E	∞
F	∞

Example of Dijkstra's Algorithm

0 dist

unfinished

finished

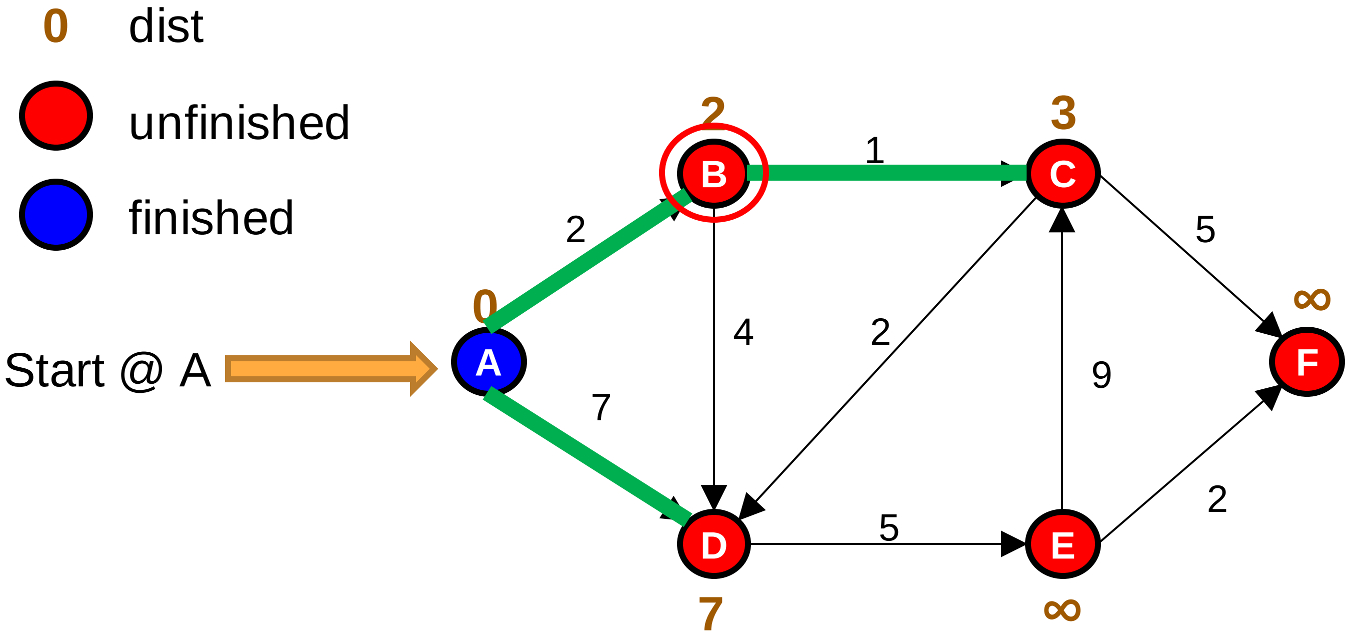


PQ

B

C	∞
D	7
E	∞
F	∞

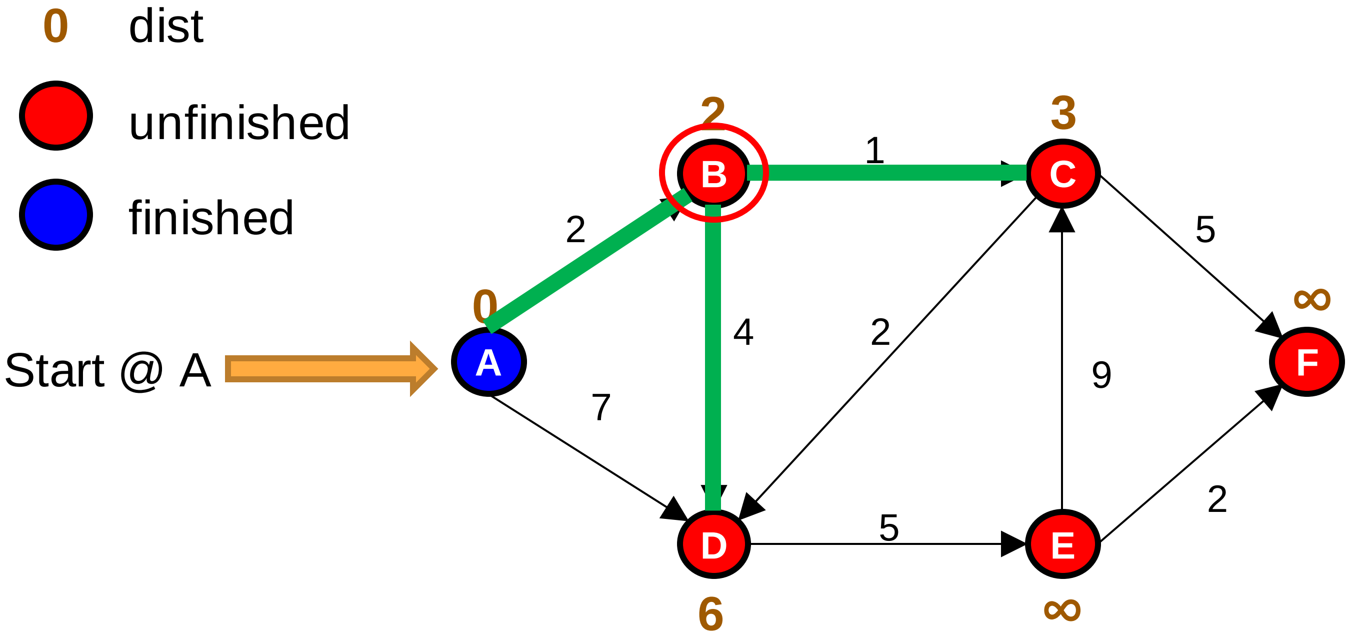
Example of Dijkstra's Algorithm



PQ

C	3
D	7
E	∞
F	∞

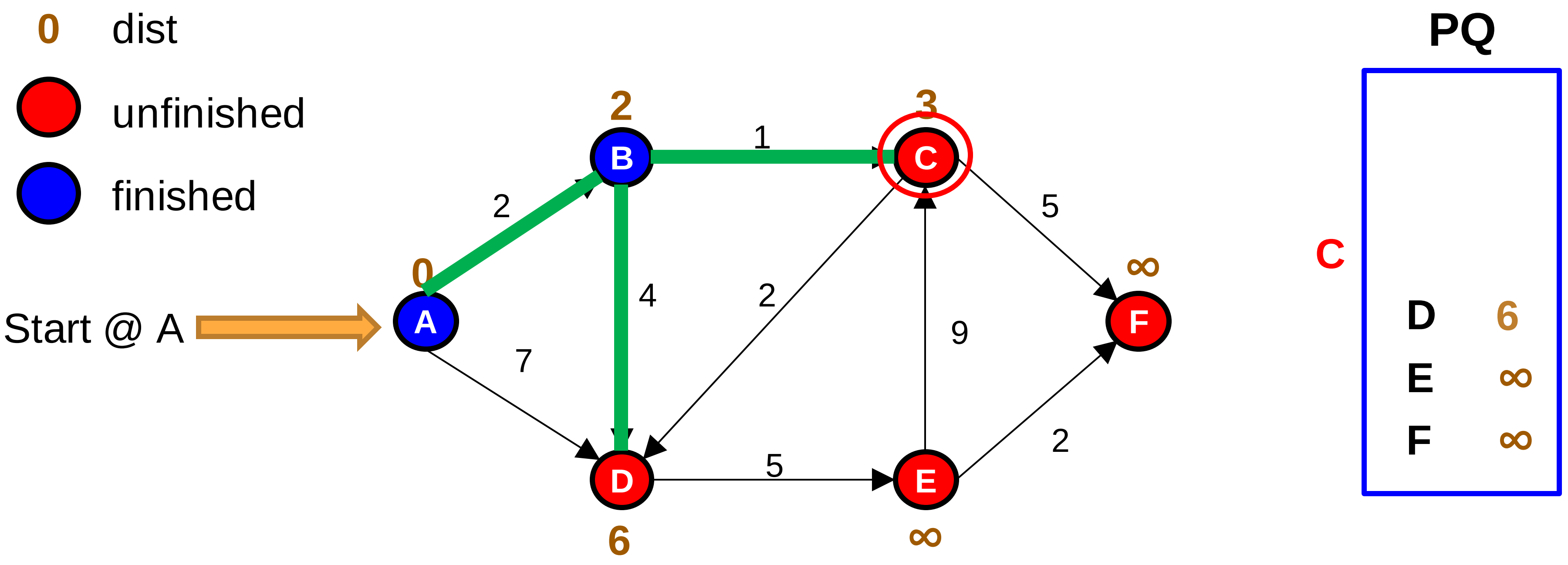
Example of Dijkstra's Algorithm



PQ

C	3
D	6
E	∞
F	∞

Example of Dijkstra's Algorithm

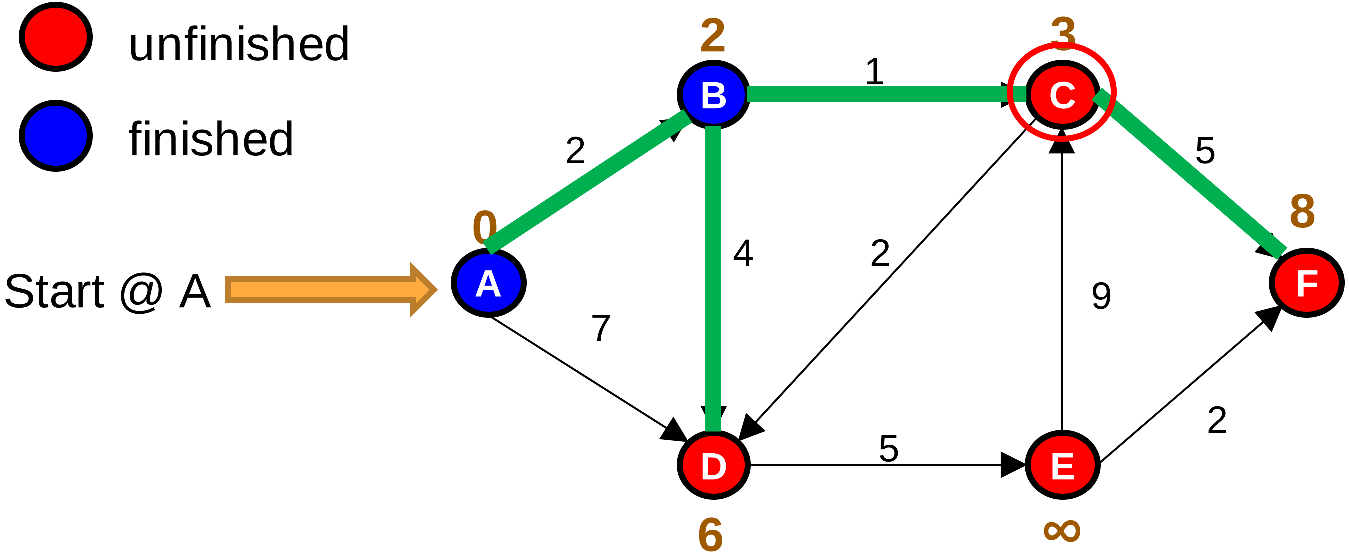


Example of Dijkstra's Algorithm

0 dist

unfinished

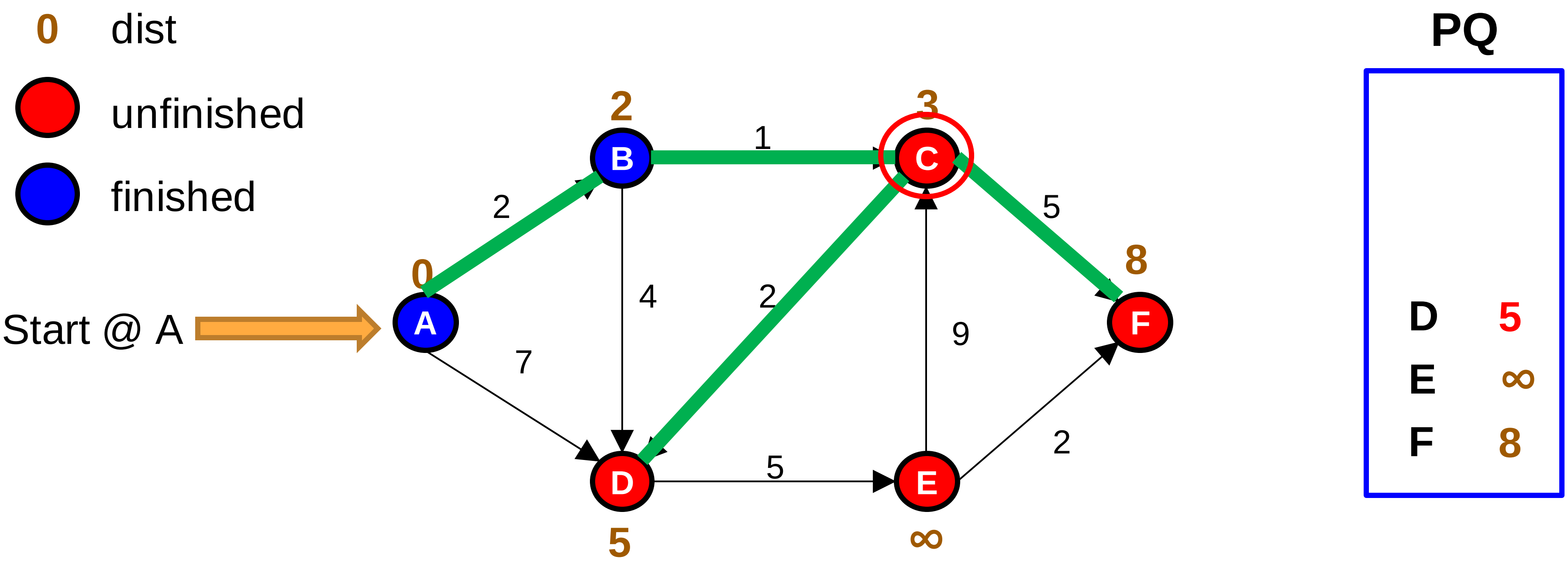
finished



PQ

D	6
E	∞
F	8

Example of Dijkstra’s Algorithm



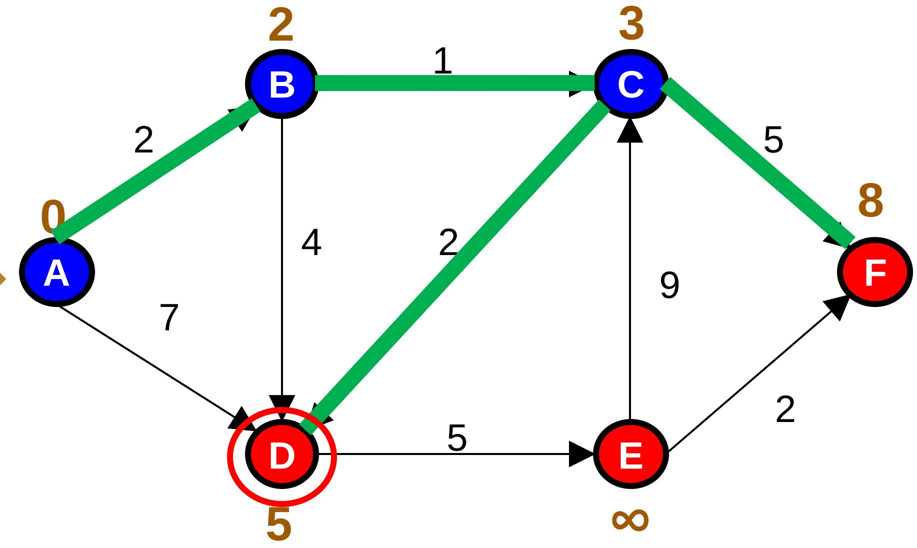
Example of Dijkstra's Algorithm

0 dist

unfinished

finished

Start @ A

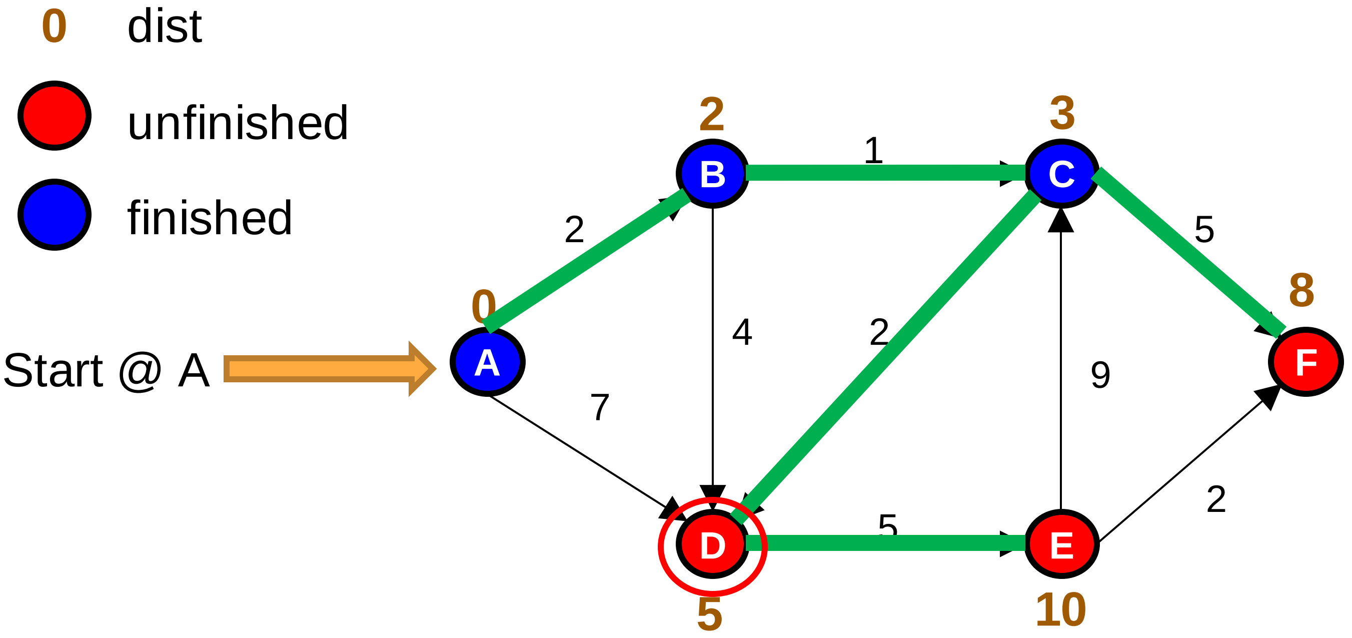


PQ

D

E	∞
F	8

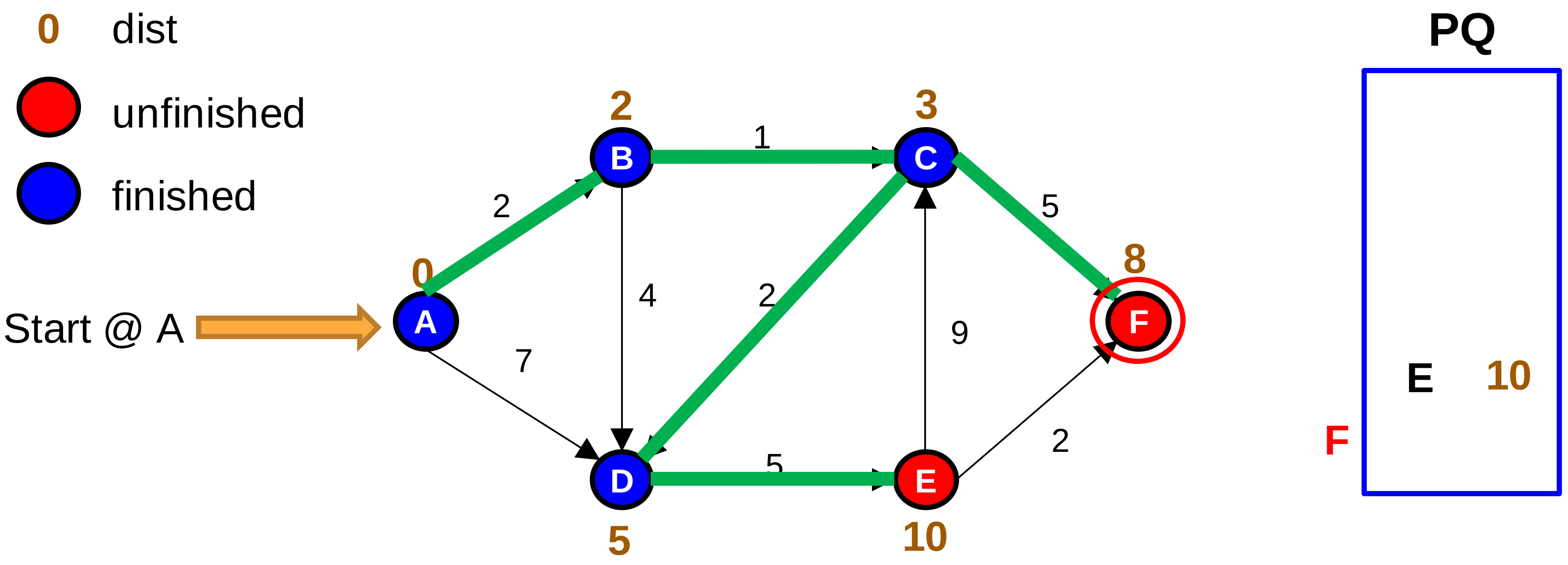
Example of Dijkstra's Algorithm



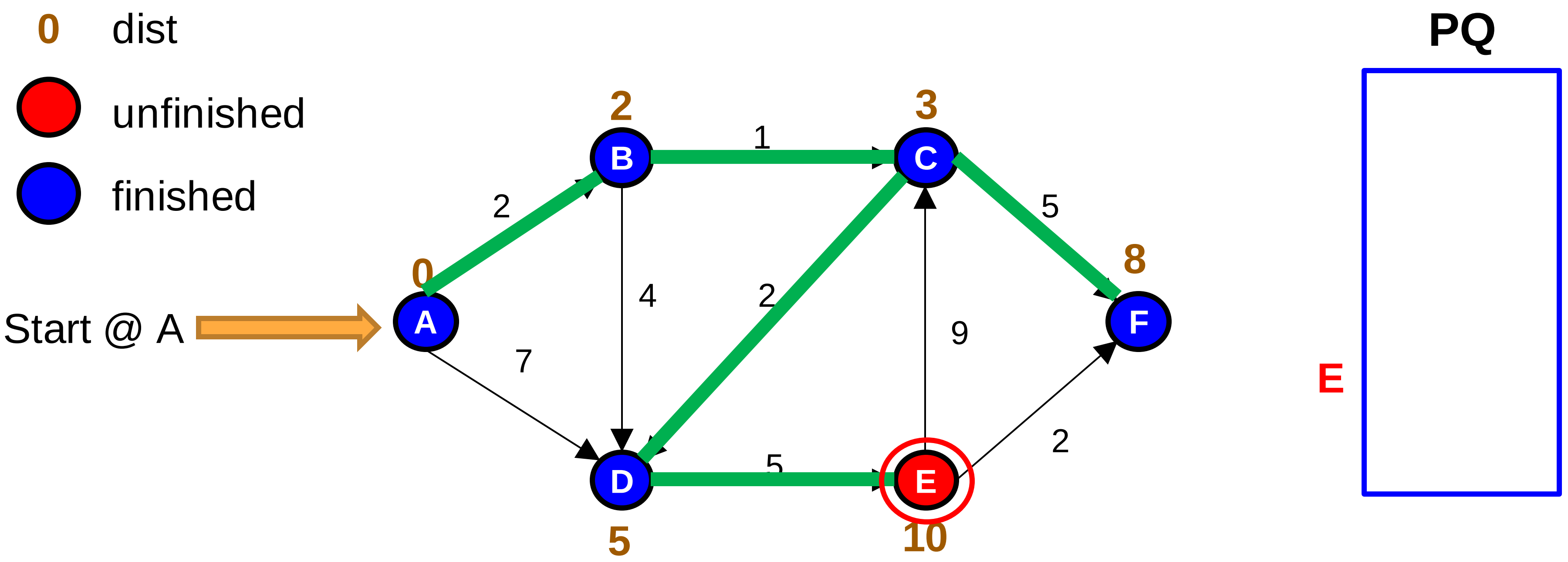
PQ

E	10
F	8

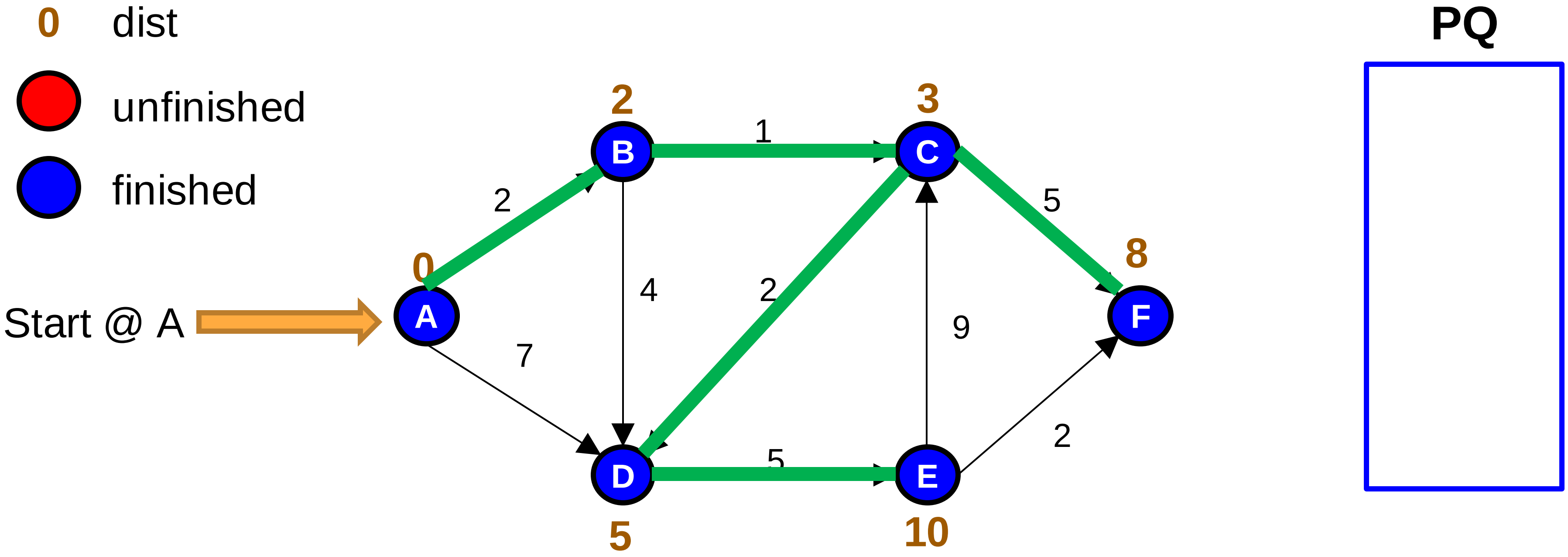
Example of Dijkstra's Algorithm



Example of Dijkstra’s Algorithm



Example of Dijkstra's Algorithm



DONE!

Running Time of Dijkstra's Algorithm

- For each **vertex**, we do one PQ insert() and one PQ extractMin()

- For each **edge**, we do one PQ decrease()

Running Time of Dijkstra's Algorithm

- For each **vertex**, we do one PQ insert() and one PQ extractMin()
- For each **edge**, we do one PQ decrease()
- Hence, total cost is $|V|(T_{insert} + T_{extractMin}) + |E|\, T_{decrease}$
- Times for PQ operations using binary heap are all **???**

Running Time of Dijkstra's Algorithm

- For each **vertex**, we do one PQ insert() and one PQ extractMin()
- For each **edge**, we do one PQ decrease()
- Hence, total cost is $|V|(T_{insert} + T_{extractMin}) + |E|\ T_{decrease}$
- Times for PQ operations using binary heap are all **$\Theta(\log |V|)$**
- **Hence, algorithm runs in time $\Theta((|V| + |E|) \log |V|)$**

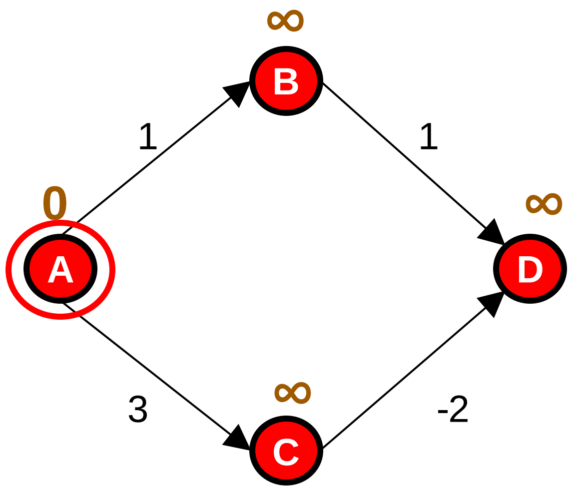
Running Time of Dijkstra's Algorithm

- For each **vert** ... O extractMin()
- For each **ed**
- Hence, total ... se
- Times for PC ... **(log |V|)**
- **Hence, algorithm runs in time Θ((|V| + |E|) log |V|)**

Slightly faster algorithms are possible for dense graphs, using much fancier heaps with O(1) decrease time.

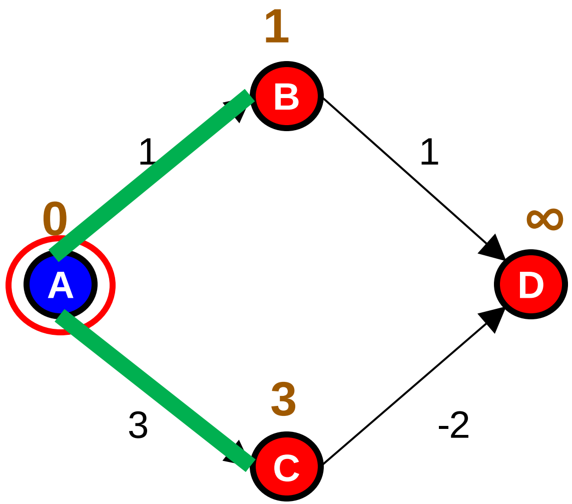
Why Must Edge Weights Be Non-Negative?

- With negative weights, Dijkstra's algorithm does not necessarily find a shortest (smallest total sum of edge weights) path.



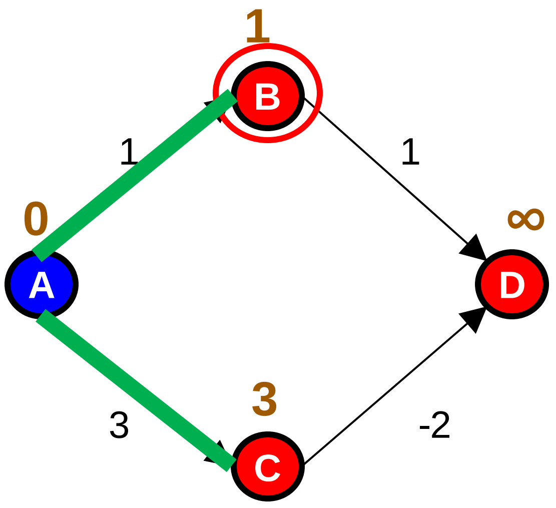
Why Must Edge Weights Be Non-Negative?

- With negative weights, Dijkstra's algorithm does not necessarily find a shortest (smallest total sum of edge weights) path.



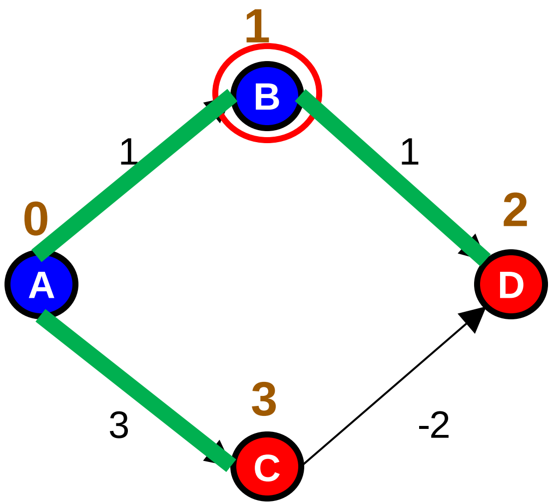
Why Must Edge Weights Be Non-Negative?

- With negative weights, Dijkstra’s algorithm does not necessarily find a shortest (smallest total sum of edge weights) path.



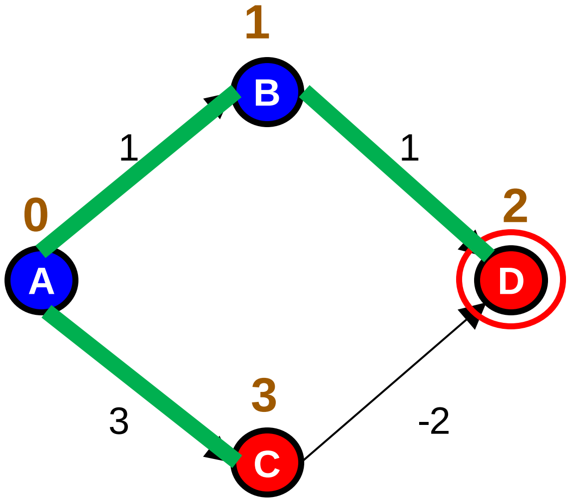
Why Must Edge Weights Be Non-Negative?

- With negative weights, Dijkstra's algorithm does not necessarily find a shortest (smallest total sum of edge weights) path.



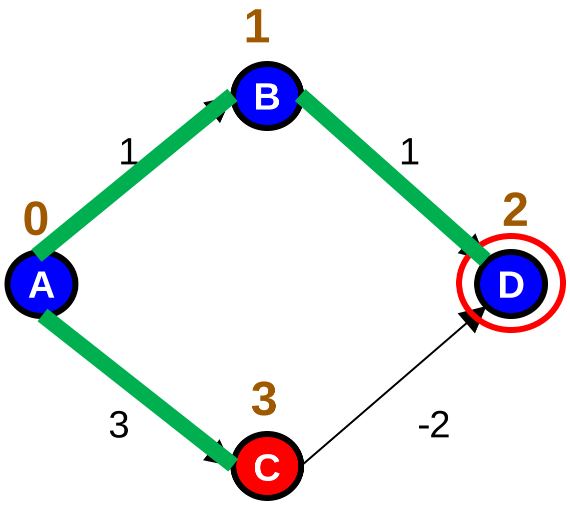
Why Must Edge Weights Be Non-Negative?

- With negative weights, Dijkstra's algorithm does not necessarily find a shortest (smallest total sum of edge weights) path.



Why Must Edge Weights Be Non-Negative?

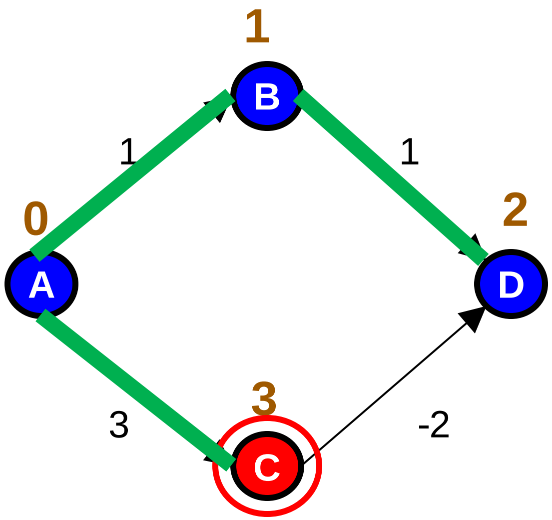
- With negative weights, Dijkstra's algorithm does not necessarily find a shortest (smallest total sum of edge weights) path.



D is *finalized* with distance 2.

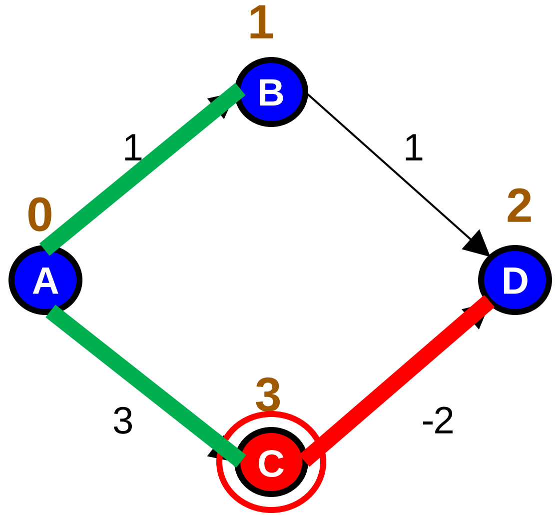
Why Must Edge Weights Be Non-Negative?

- With negative weights, Dijkstra's algorithm does not necessarily find a shortest (smallest total sum of edge weights) path.



Why Must Edge Weights Be Non-Negative?

- With negative weights, Dijkstra's algorithm does not necessarily find a shortest (smallest total sum of edge weights) path.



But *shortest* path is of length 3 + -2 = 1

Alternatives to Dijkstra

- If negative-weight edges are allowed…
- May use *Bellman-Ford* algorithm ($\Theta(|V||E|)$)
- If shortest-path distances are desired between every pair of vertices…
- May use *Floyd-Warshall* algorithm ($\Theta(|V|^3)$)
- (*Other approaches may be better for sparse graphs*)