Mathematical Tools for Engineering and Management

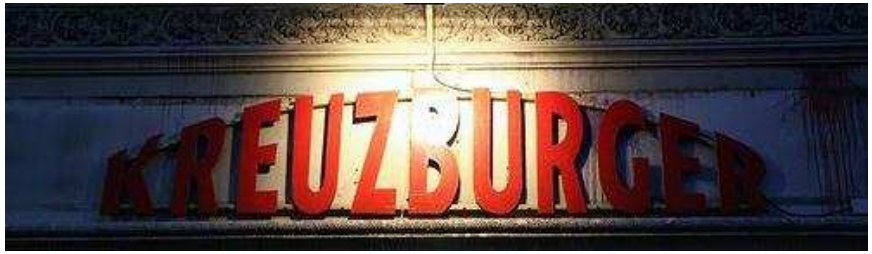
Lecture 10

04 Jan 2012

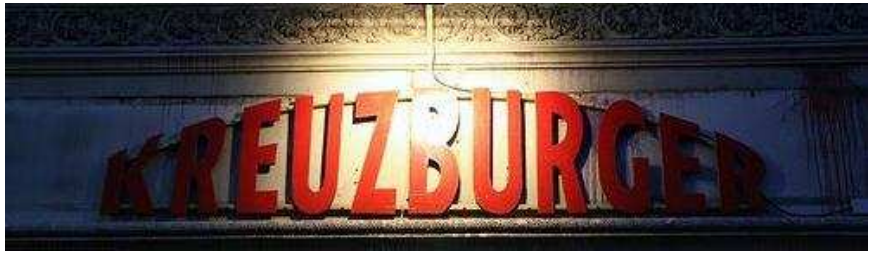
- ▷ Models, Data and Algorithms
- ▷ Linear Optimization
- ▷ Mathematical Background: Polyhedra, Simplex-Algorithm
- ▷ Sensitivity Analysis; (Mixed) Integer Programming
- ▷ MIP Modelling
- ▷ MIP Modelling: More Examples; Branch & Bound
- ▷ Cutting Planes; Combinatorial Optimization: Examples, Graphs, Algorithms
- ▷ TSP-Heuristics
- ▷ Network Flows
- ▷ Shortest Path Problem
- ▷ Complexity Theory
- ▷ Nonlinear Optimization, Scheduling
- ▷ Lot Sizing, Multicriteria Optimization
- ▷ Oral exam

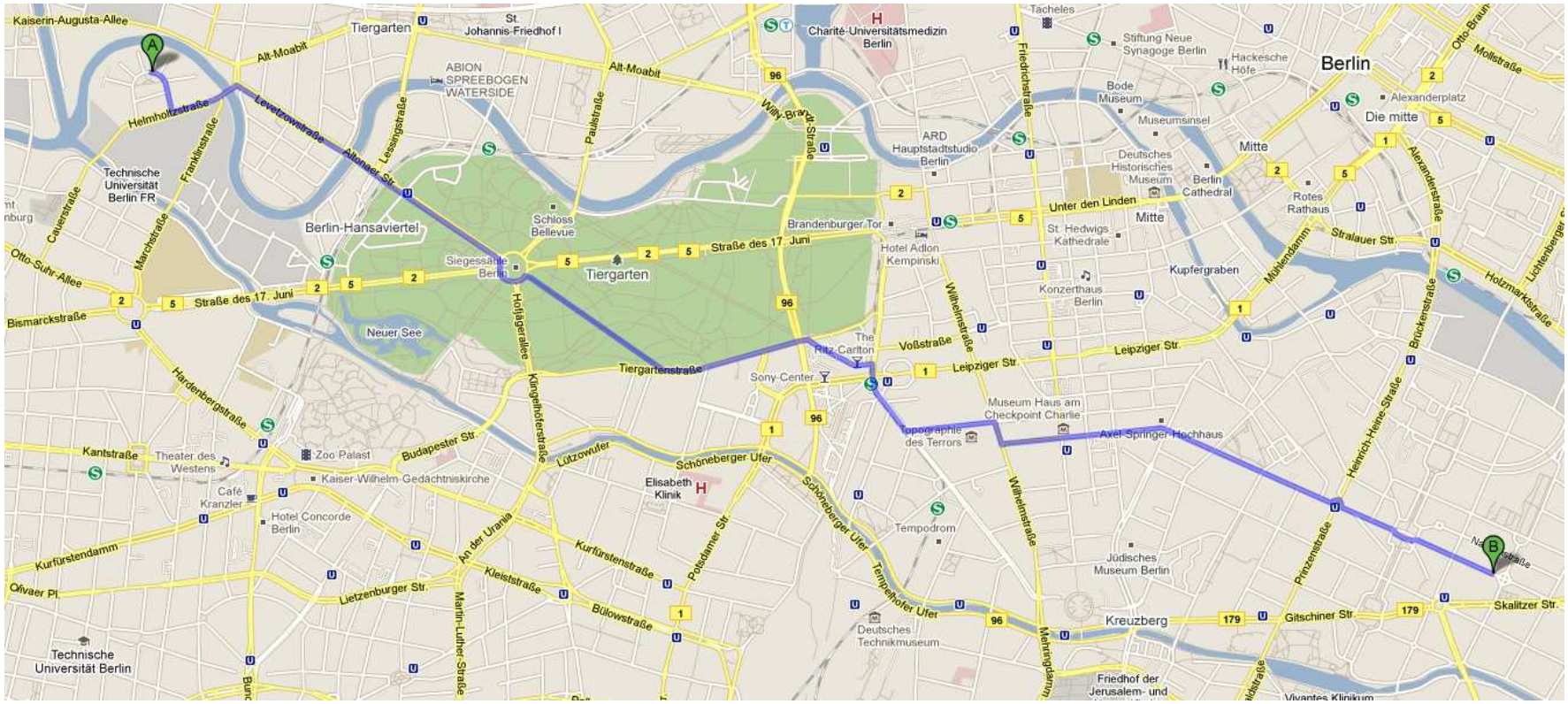
▷ Problem: find a shortest path from PTZ to Kreuzburger

⇝



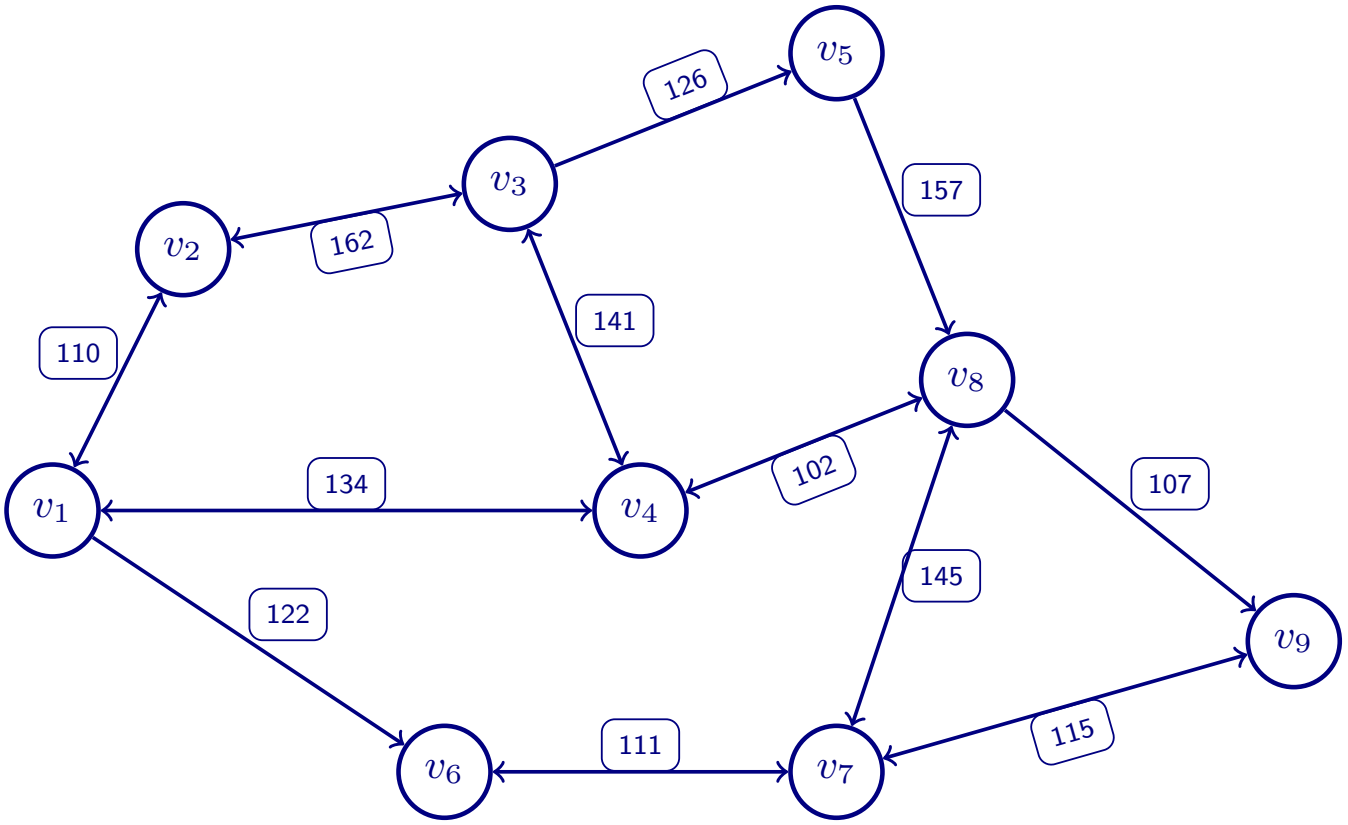
▷ Problem: find a shortest path from PTZ to Kreuzburger



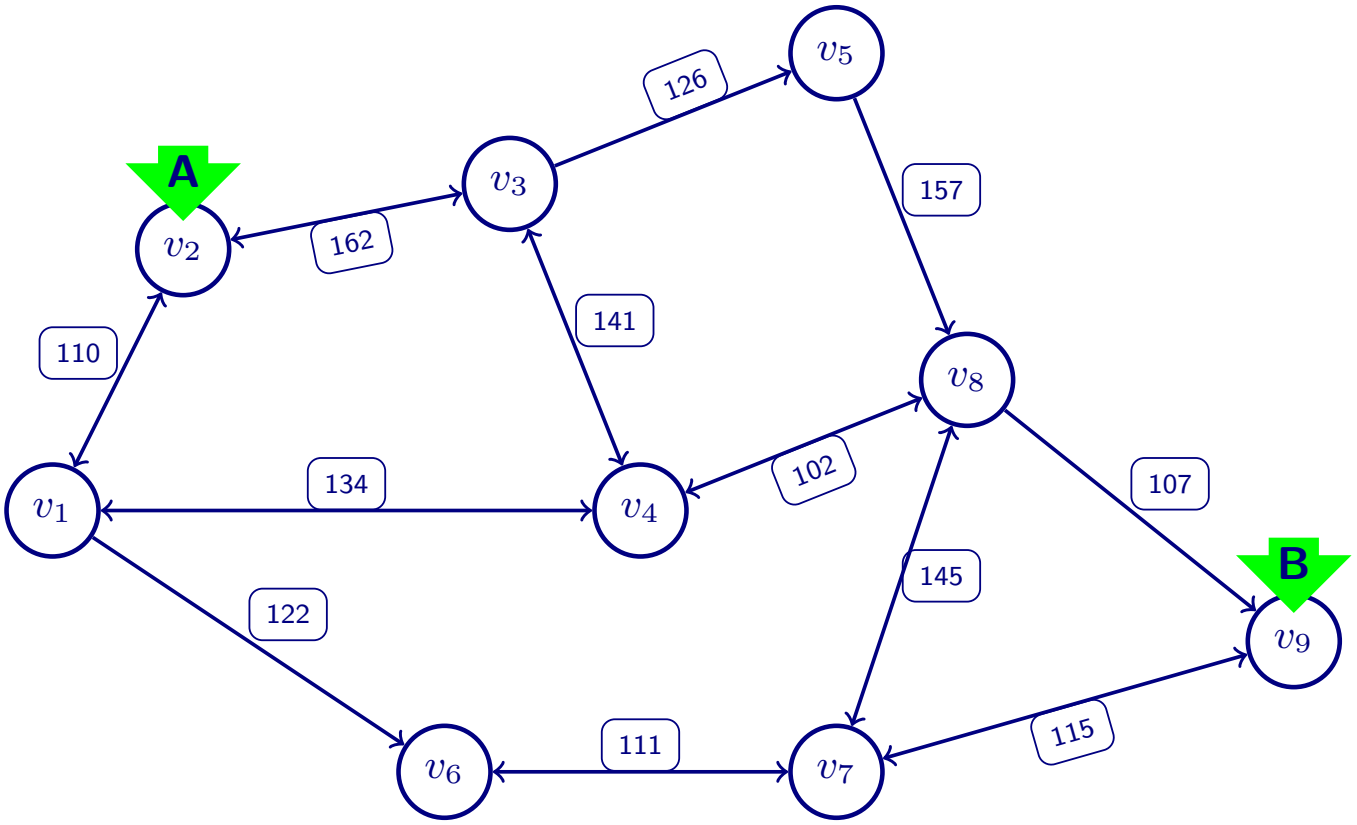


- Given a network – i.e. a directed graph – with a length for each arc, a start node A and a destination B...

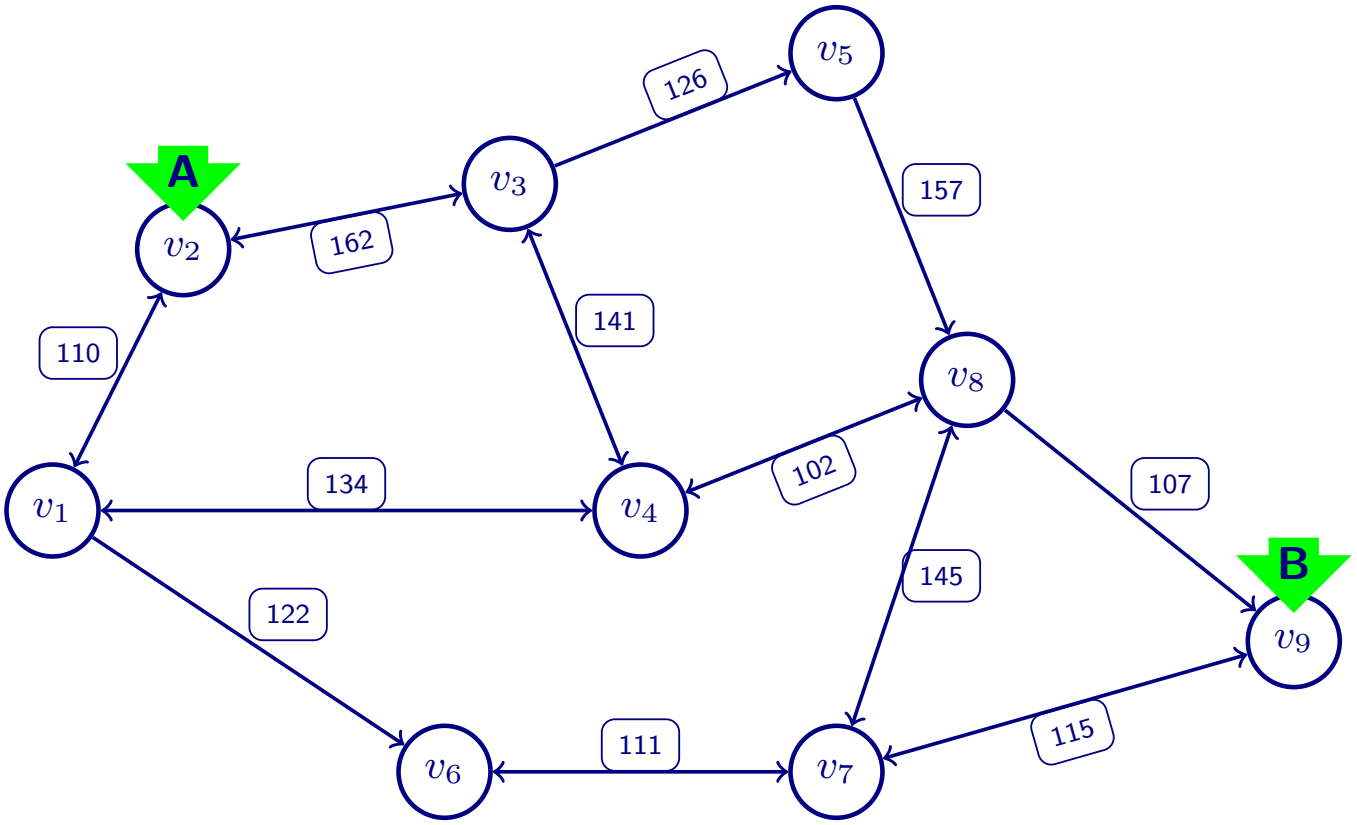
▷ Given a network – i.e. a directed graph – with a length for each arc, a start node A and a destination B...



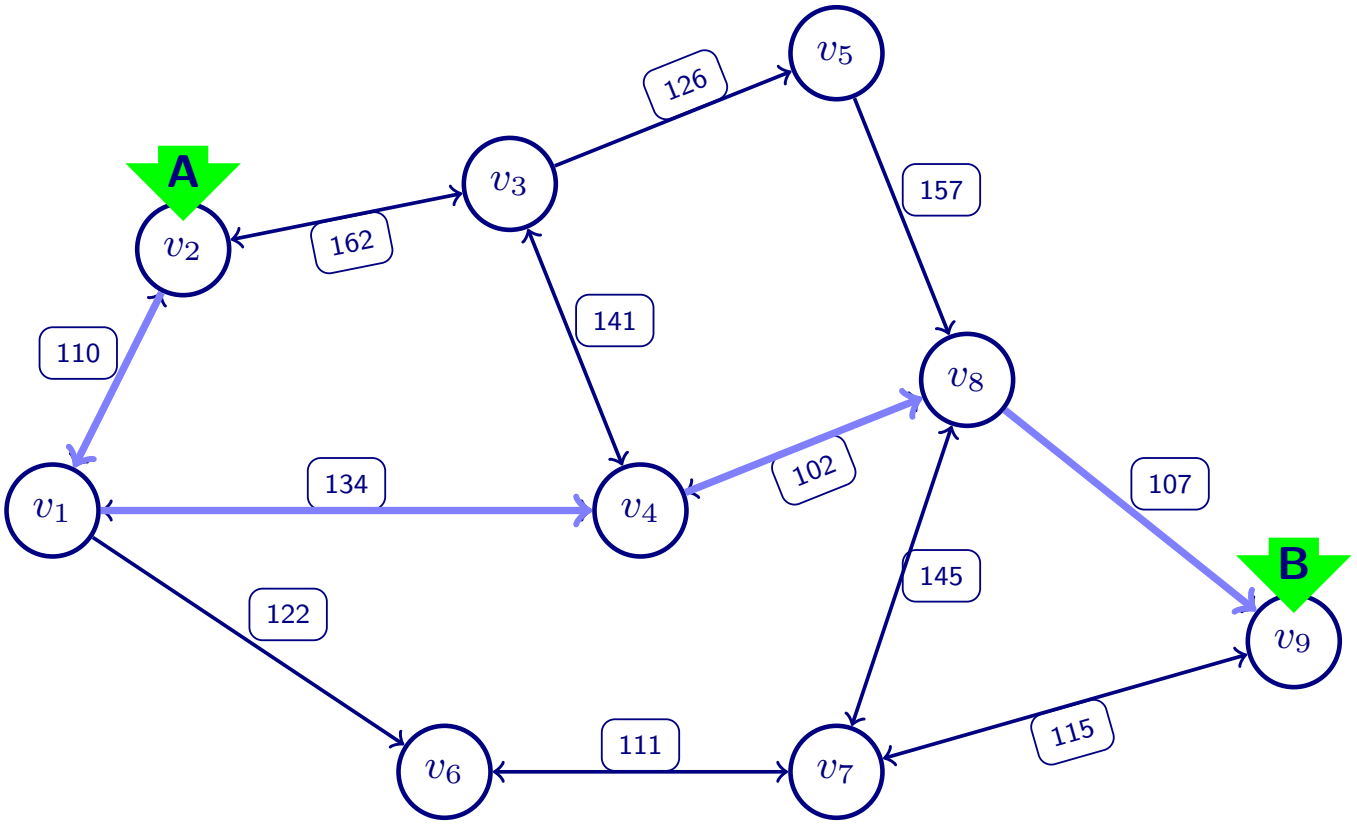
▷ Given a network – i.e. a directed graph – with a length for each arc, a start node A and a destination B...



- Given a network – i.e. a directed graph – with a length for each arc, a start node A and a destination B...
- ...compute a shortest path through the network from A to B

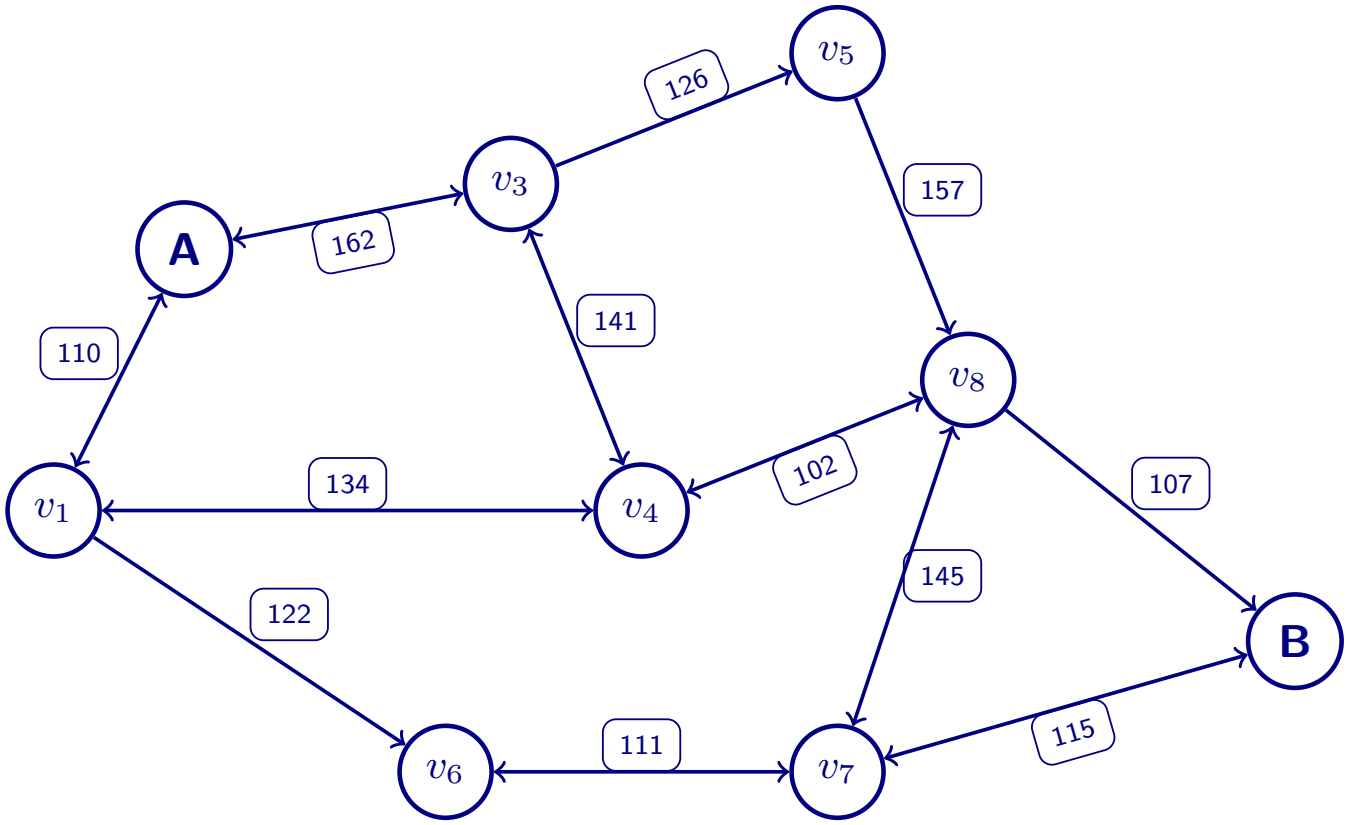


- Given a network – i.e. a directed graph – with a length for each arc, a start node A and a destination B...
- ...compute a shortest path through the network from A to B

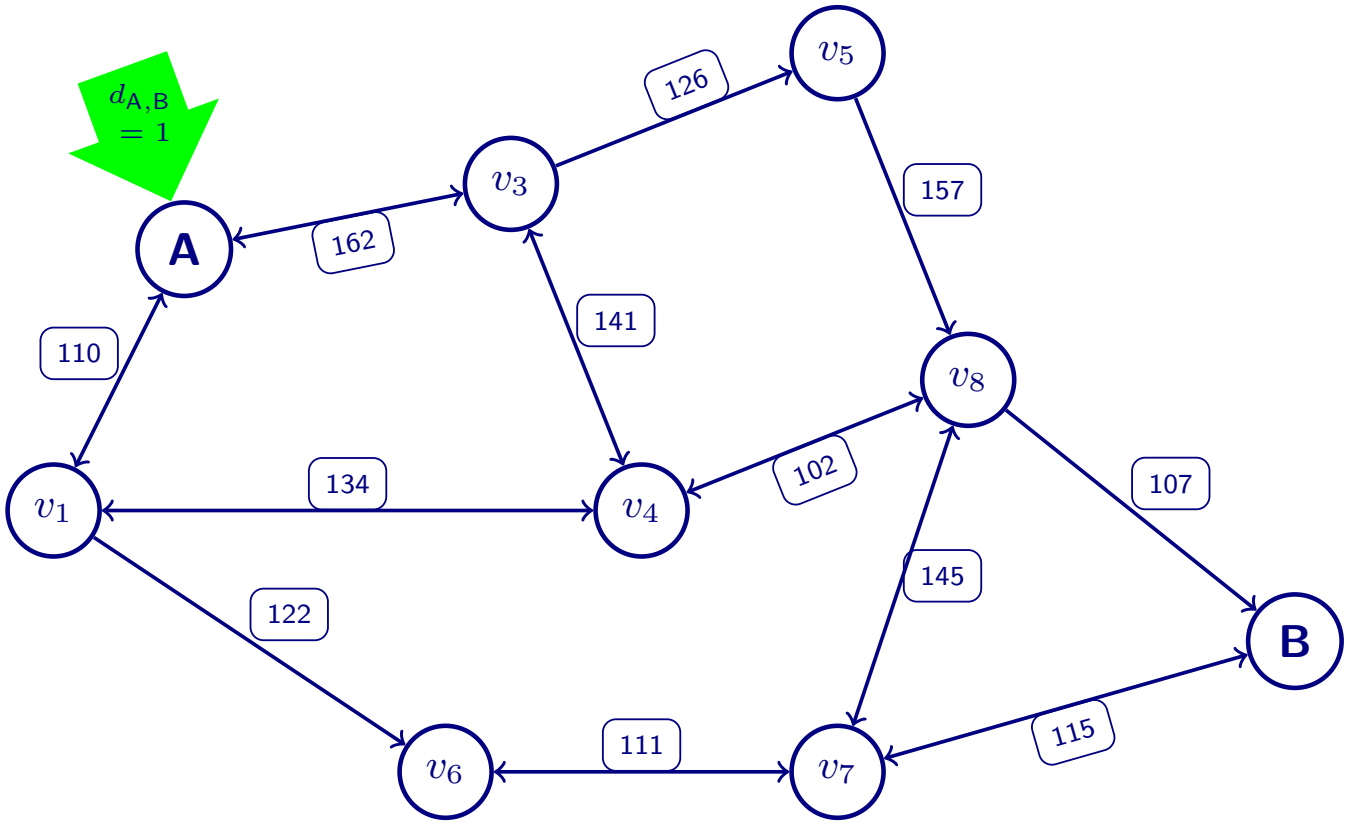


▷ Shortest Path Problem can be formulated as a network flow problem:

▷ Shortest Path Problem can be formulated as a network flow problem:

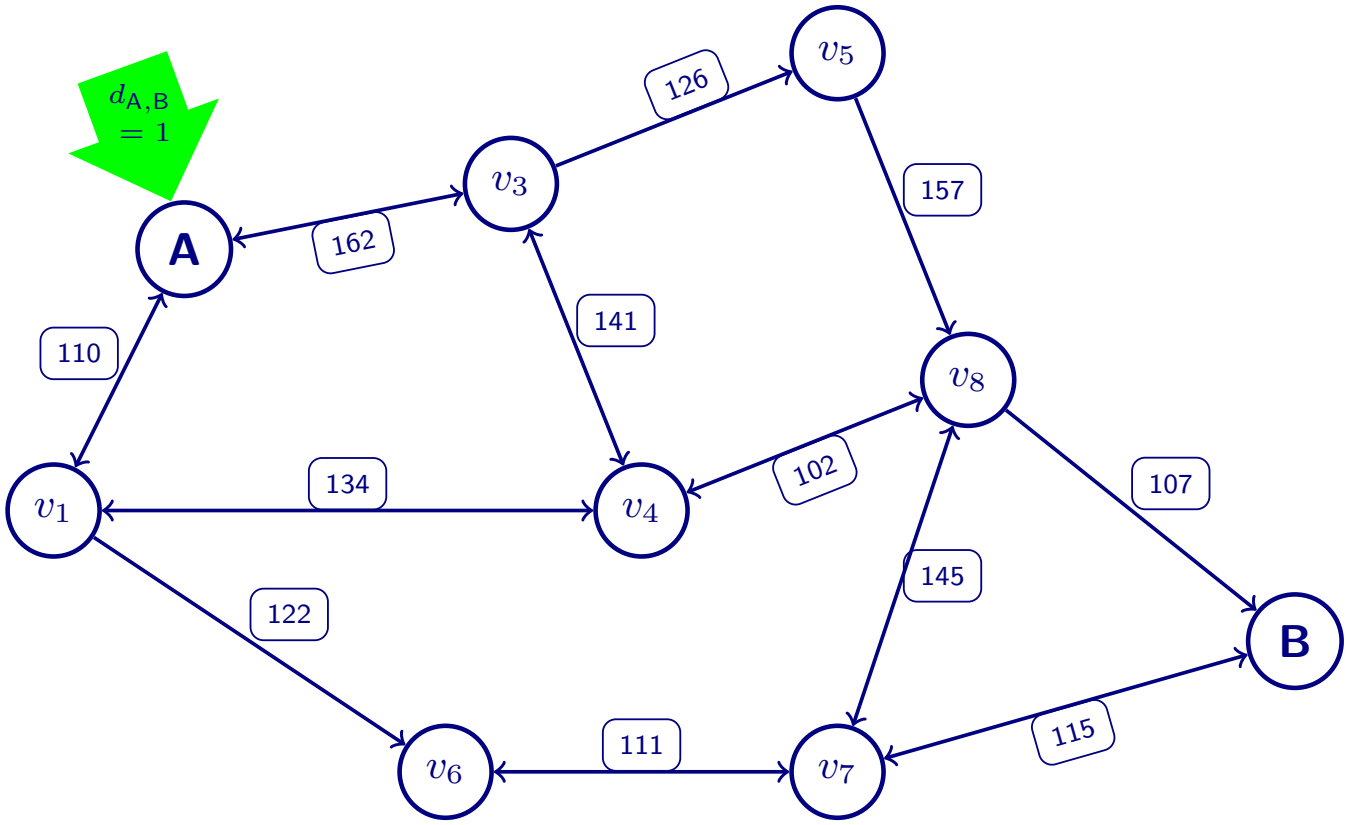


▷ Shortest Path Problem can be formulated as a network flow problem:



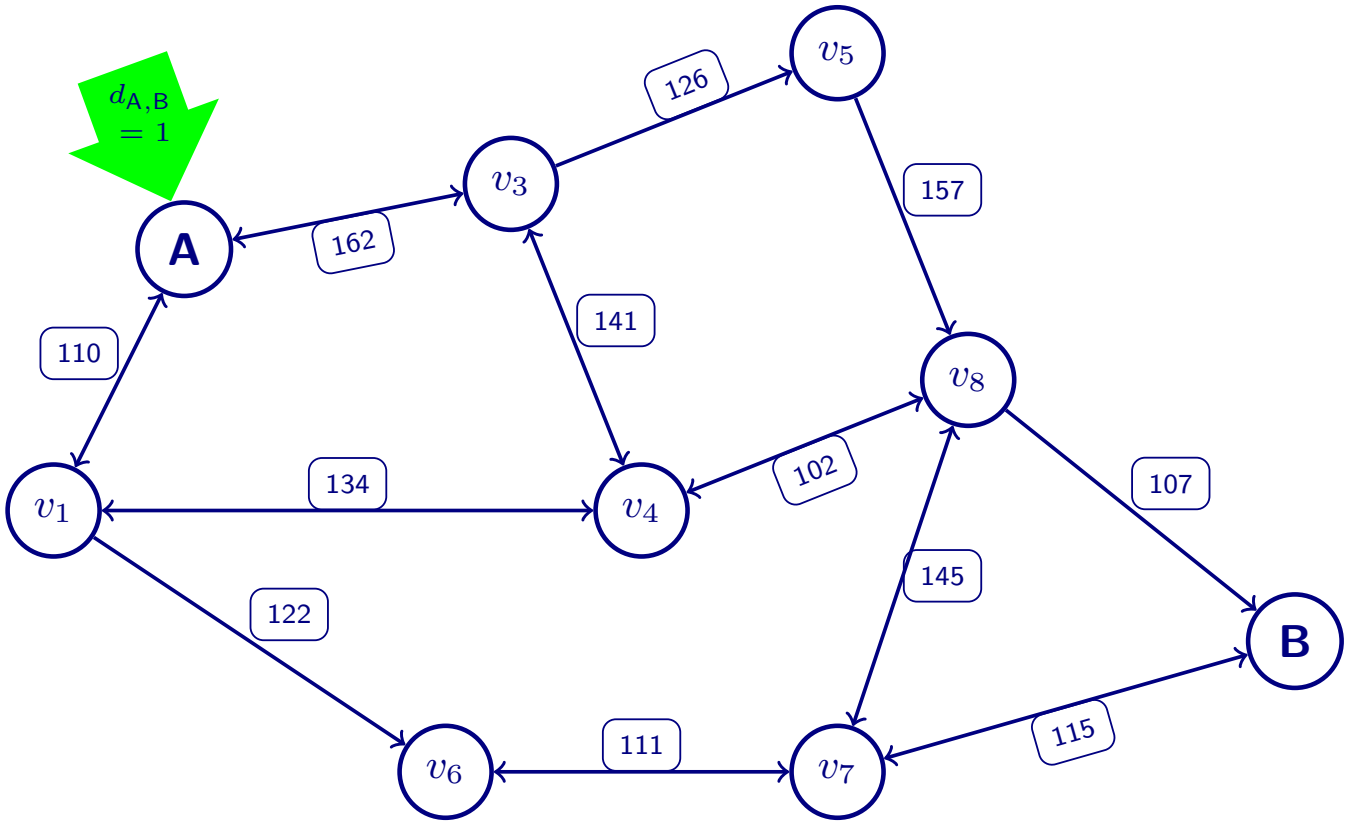
▷ Shortest Path Problem can be formulated as a network flow problem:

- Only one demand: $d_{\mathsf{A,B}} = 1$, all other demands $d_{u,v} = 0$



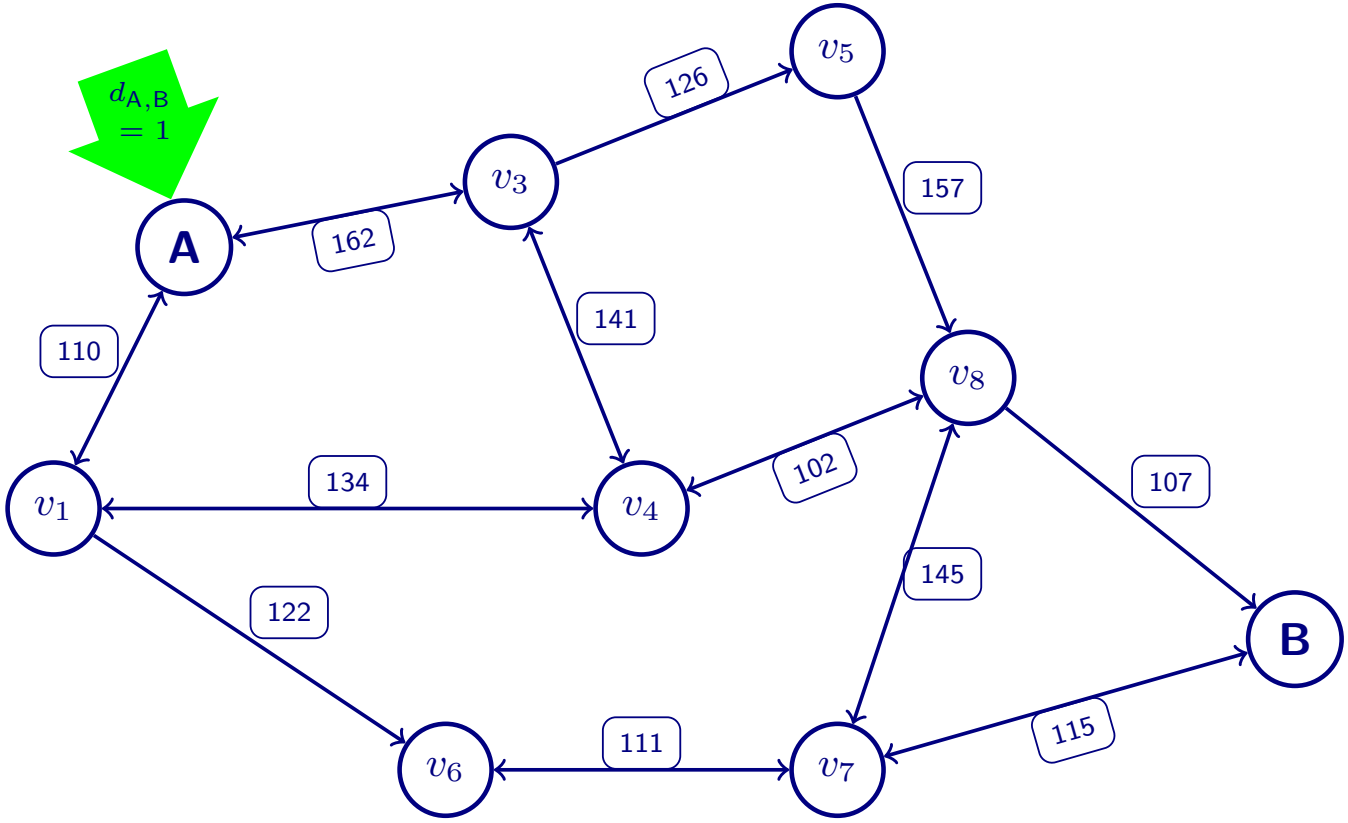
▷ Shortest Path Problem can be formulated as a network flow problem:

- Only one demand: $d_{\mathsf{A,B}} = 1$, all other demands $d_{u,v} = 0$
- Unsplittable flow



▷ Shortest Path Problem can be formulated as a network flow problem:

- Only one demand: $d_{\mathsf{A,B}} = 1$, all other demands $d_{u,v} = 0$
- Unsplittable flow
- No capacities

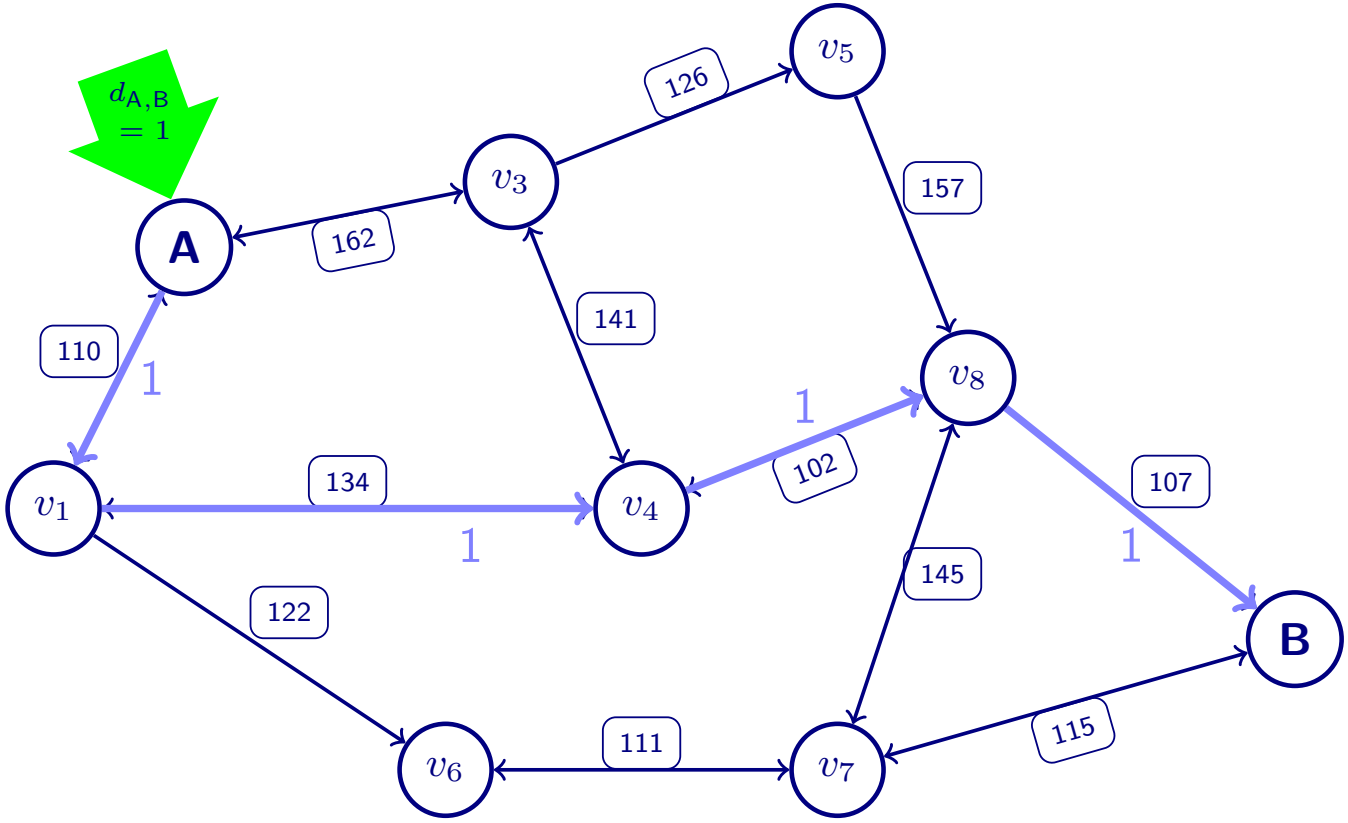


▷ Shortest Path Problem can be formulated as a network flow problem:

- Only one demand: $d_{\mathsf{A,B}} = 1$, all other demands $d_{u,v} = 0$
- Unsplittable flow
- No capacities

➡ Optimal solution:

$y_{(\mathsf{A},v_1)} = y_{(v_1,v_4)} = y_{(v_4,v_8)} = y_{(v_8,\mathsf{B})} = 1$,
all other variables 0

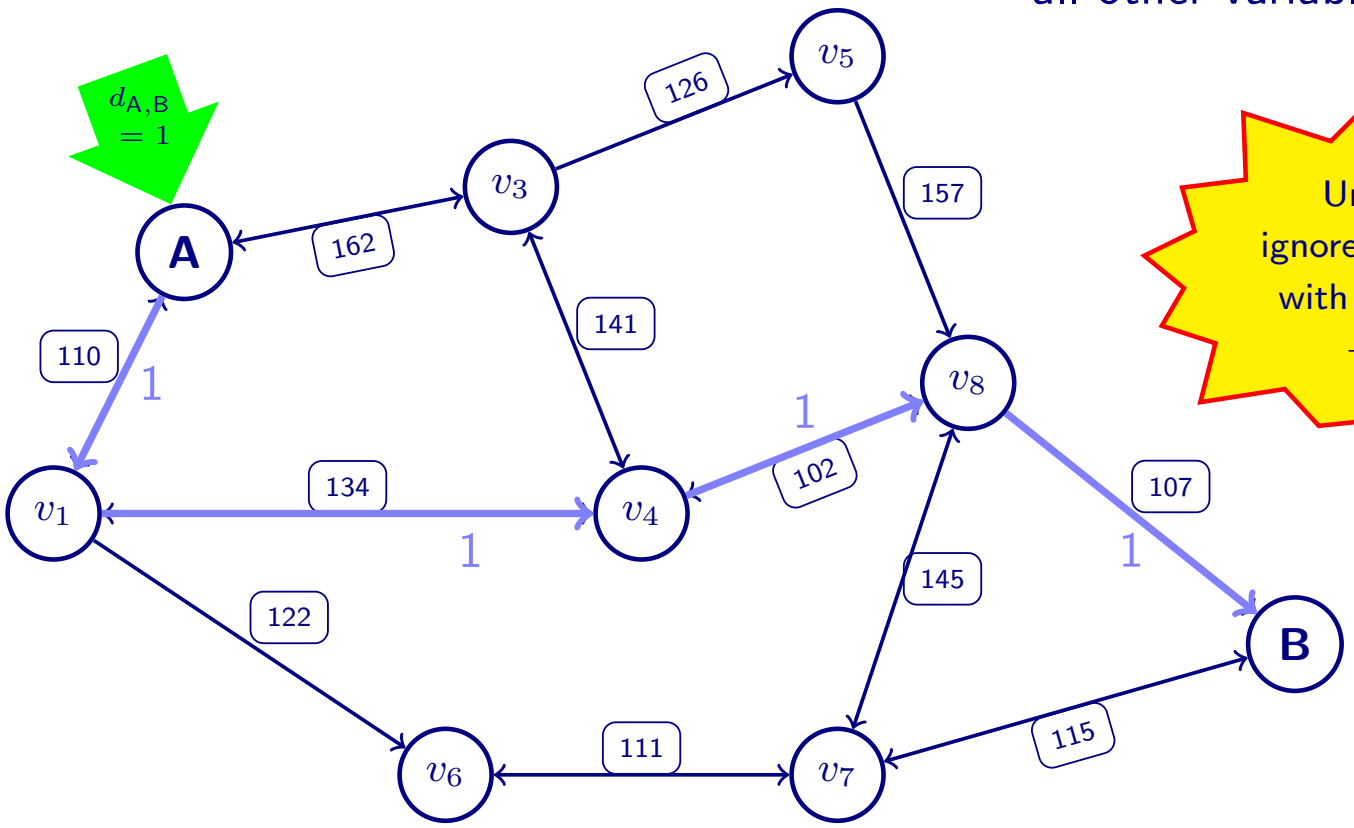


▷ Shortest Path Problem can be formulated as a network flow problem:

- Only one demand: $d_{\mathsf{A,B}} = 1$, all other demands $d_{u,v} = 0$
- Unsplittable flow
- No capacities

➡ Optimal solution:

$y_{(\mathsf{A},v_1)} = y_{(v_1,v_4)} = y_{(v_4,v_8)} = y_{(v_8,\mathsf{B})} = 1$,
all other variables 0



Unsplittability can be ignored if the model is solved with the simplex algorithm → linear program!

▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Edsger Wybe Dijkstra
(1930–2002)

▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

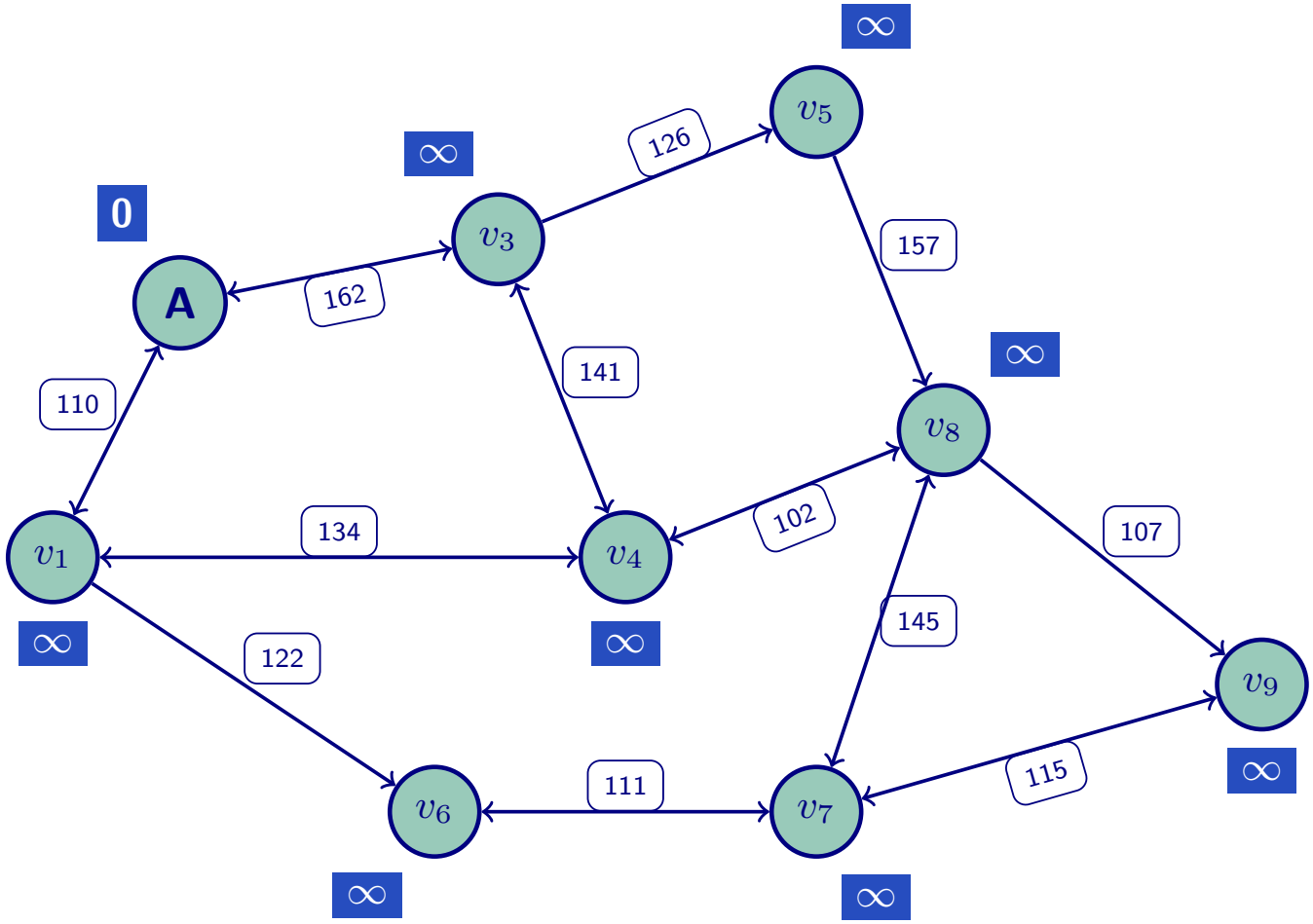
Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

R remaining nodes

$d(v)$ min distance to A



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

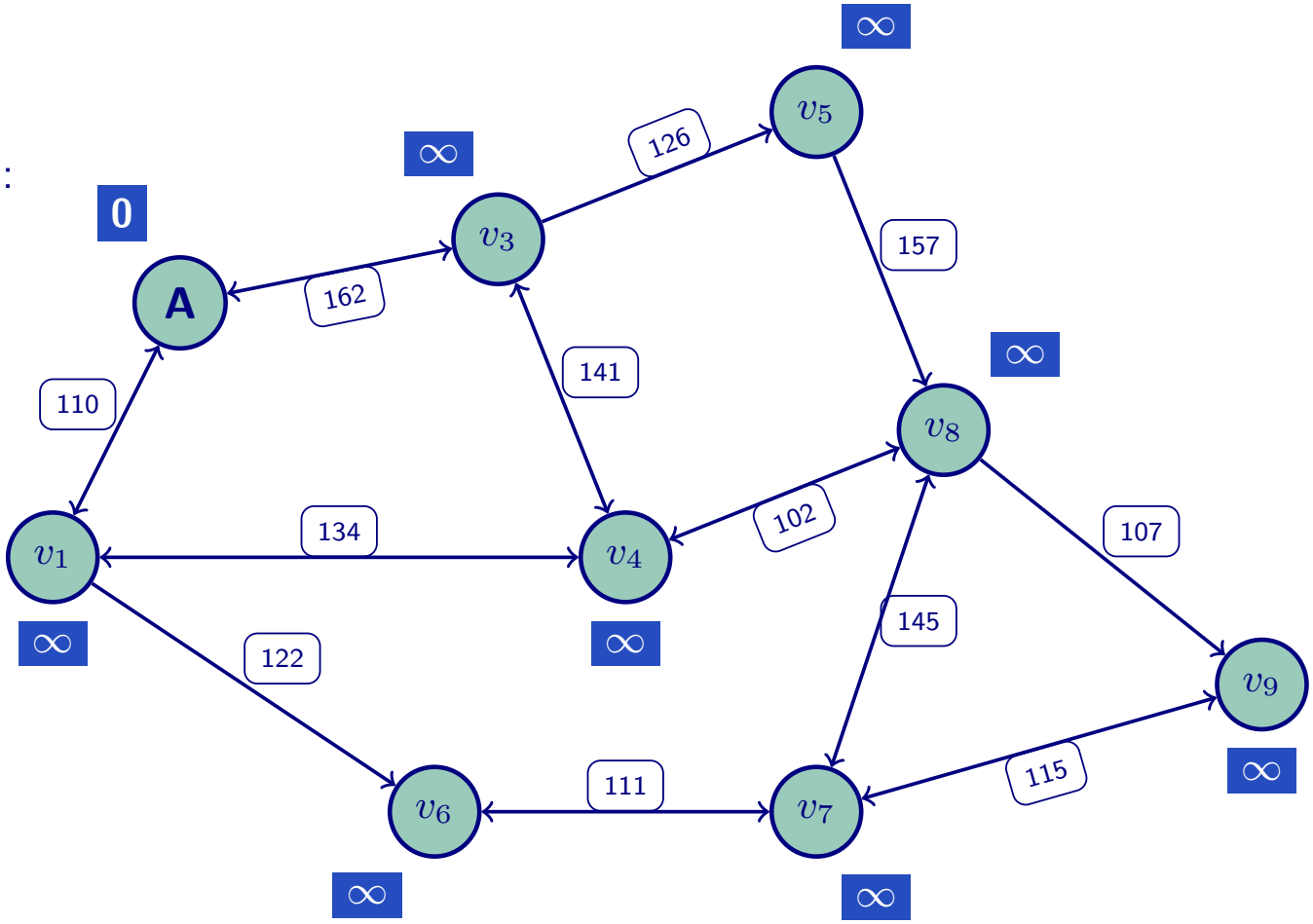
set $d(w) := d(v) + \ell(v, w)$

set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

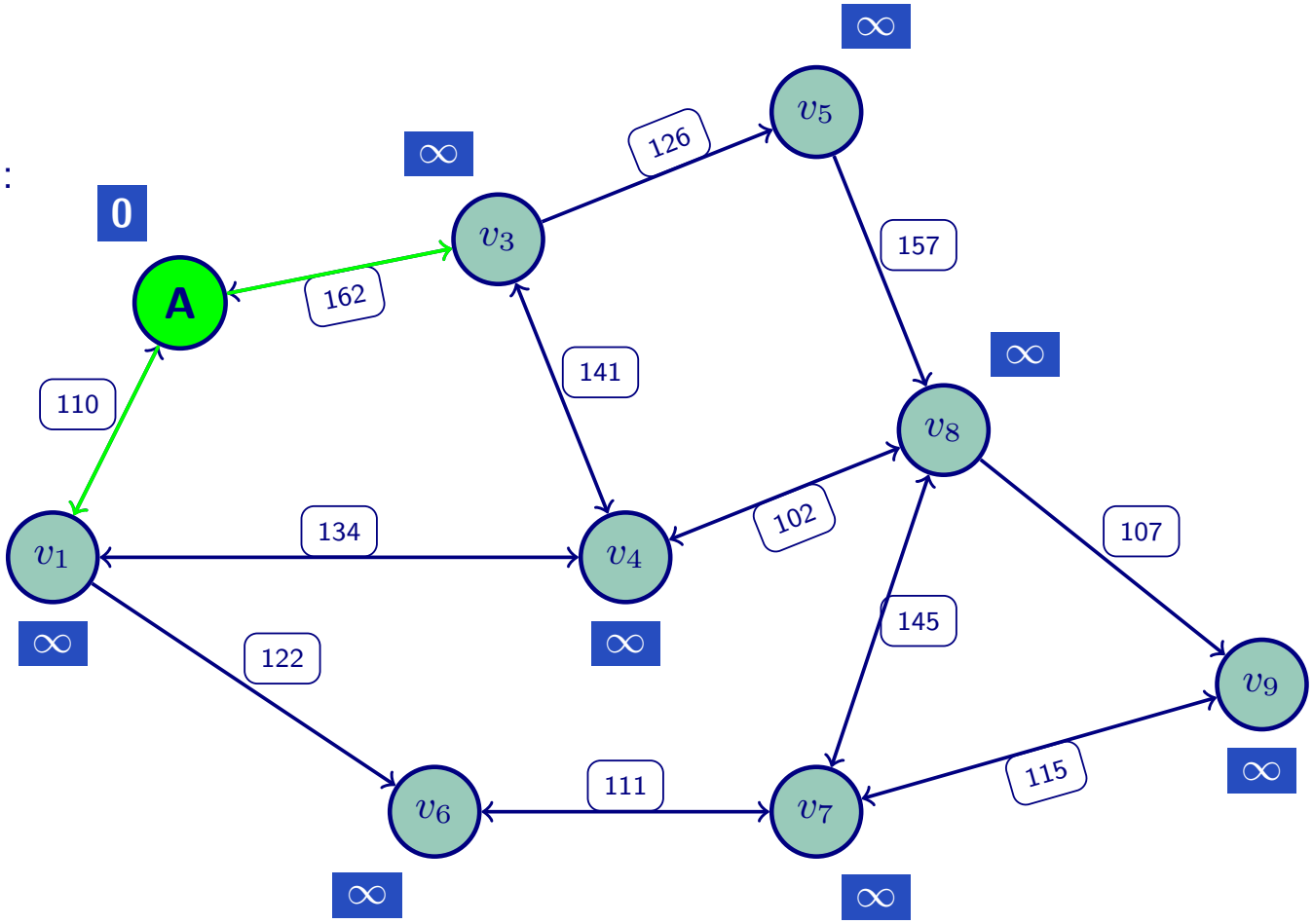
set $d(w) := d(v) + \ell(v, w)$

set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

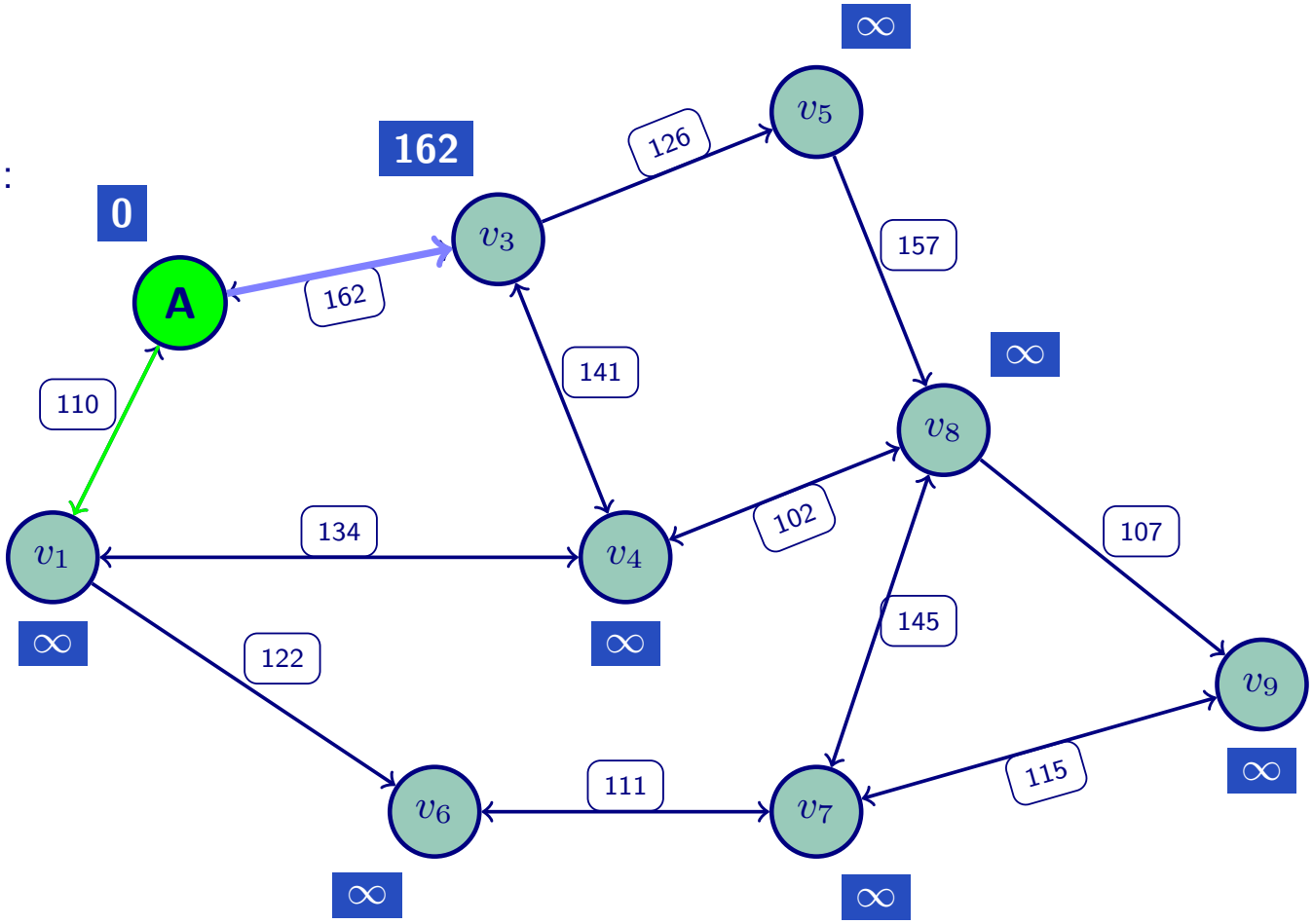
set $d(w) := d(v) + \ell(v, w)$

set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

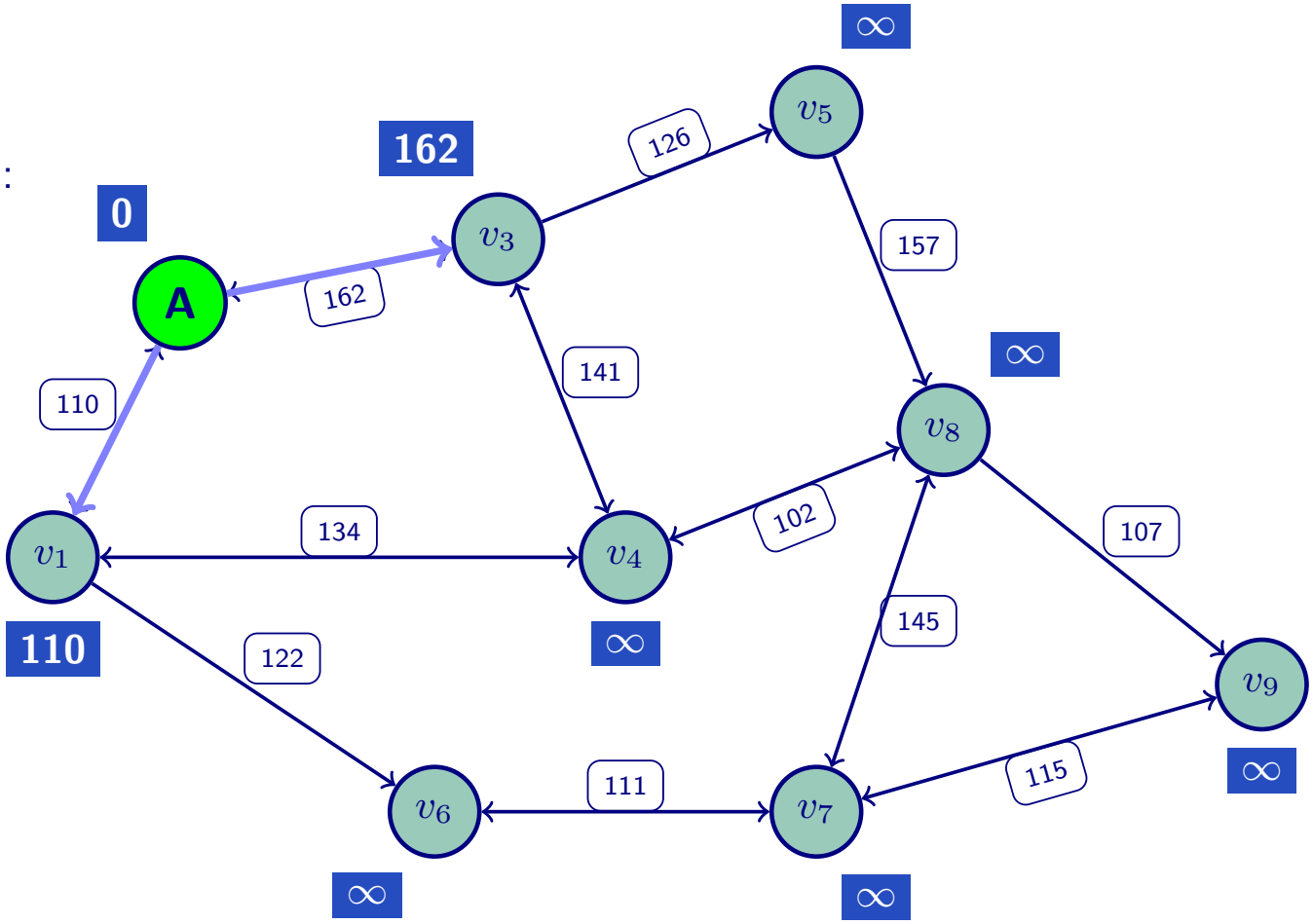
set $d(w) := d(v) + \ell(v, w)$

set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

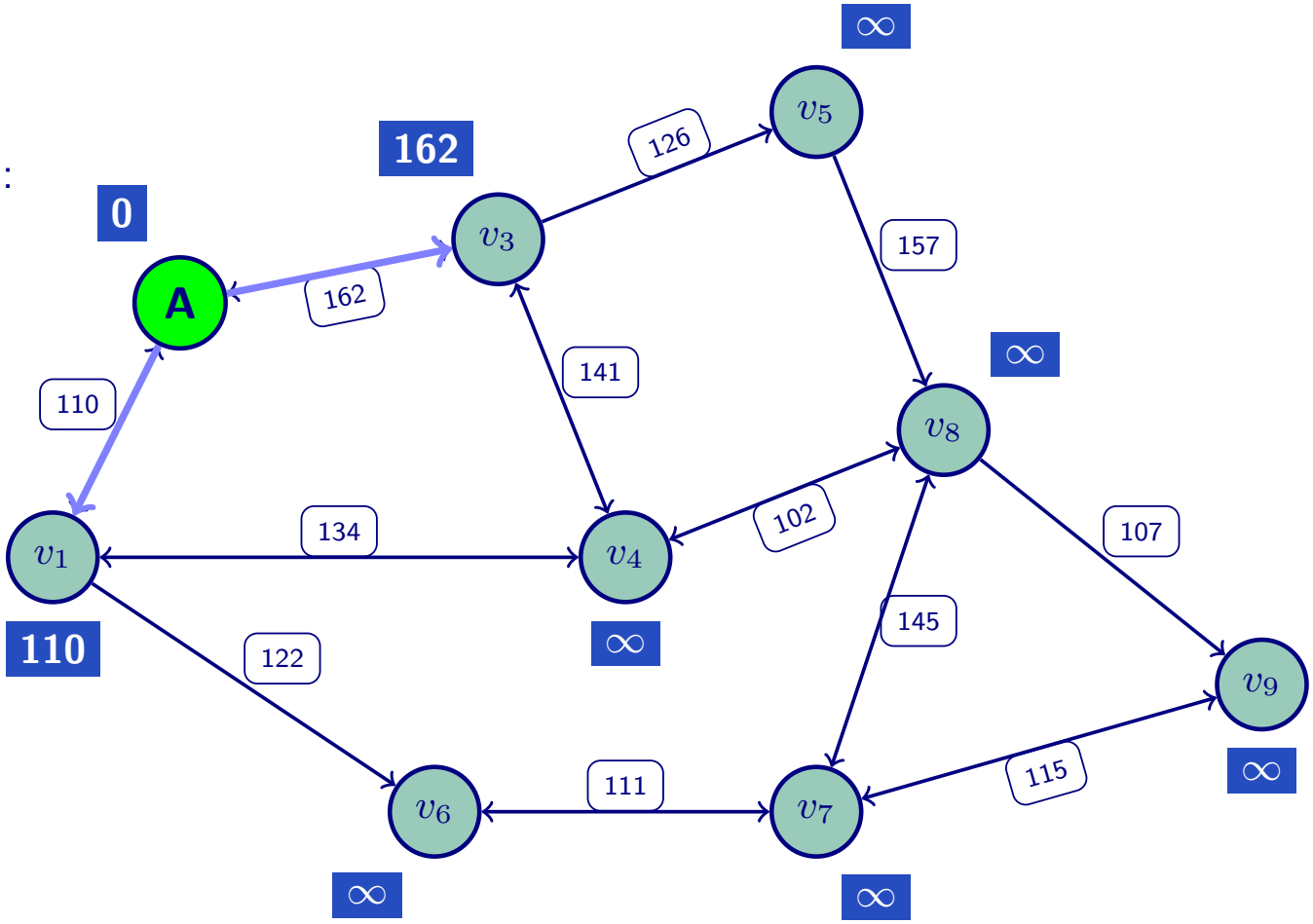
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

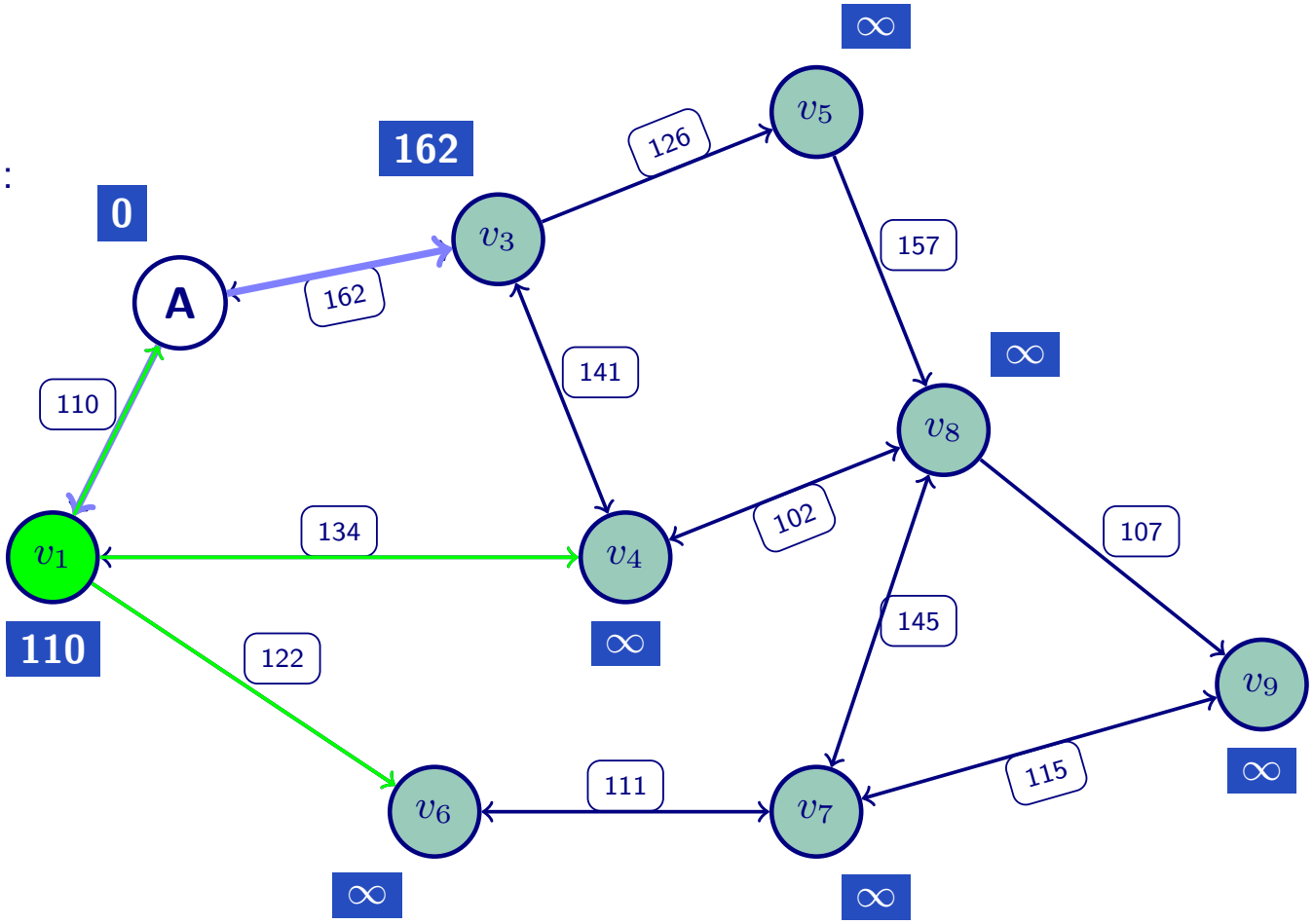
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

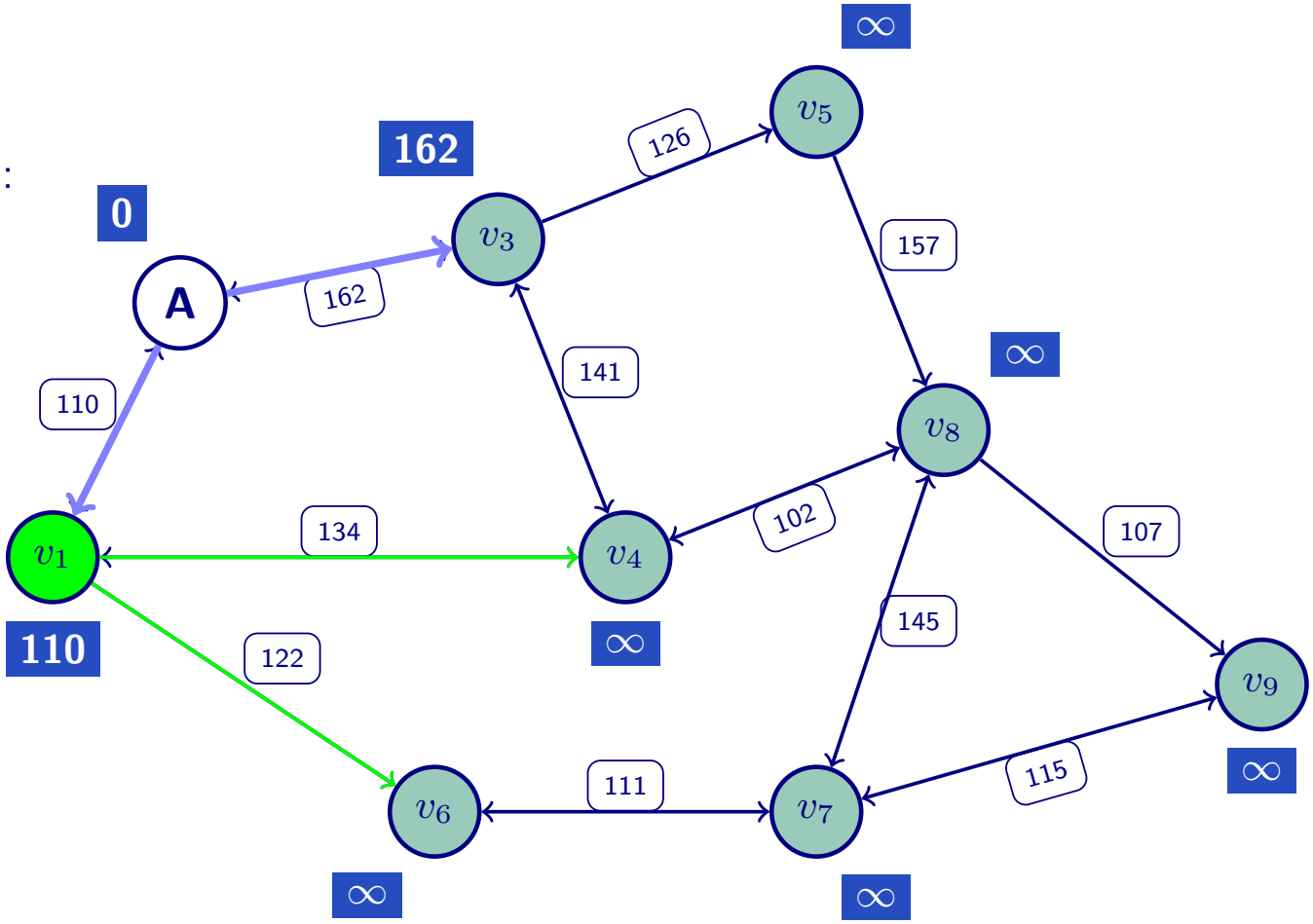
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

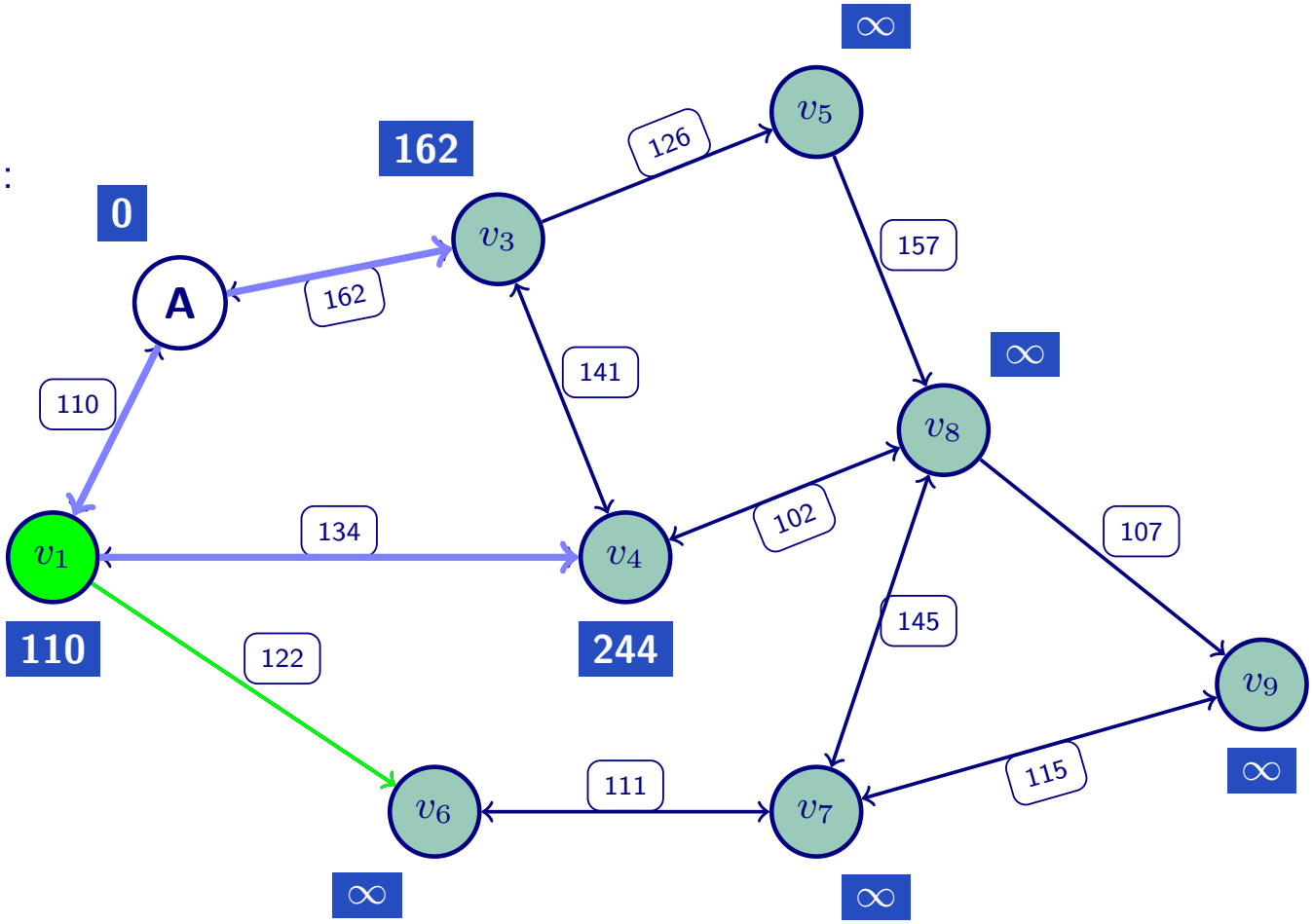
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

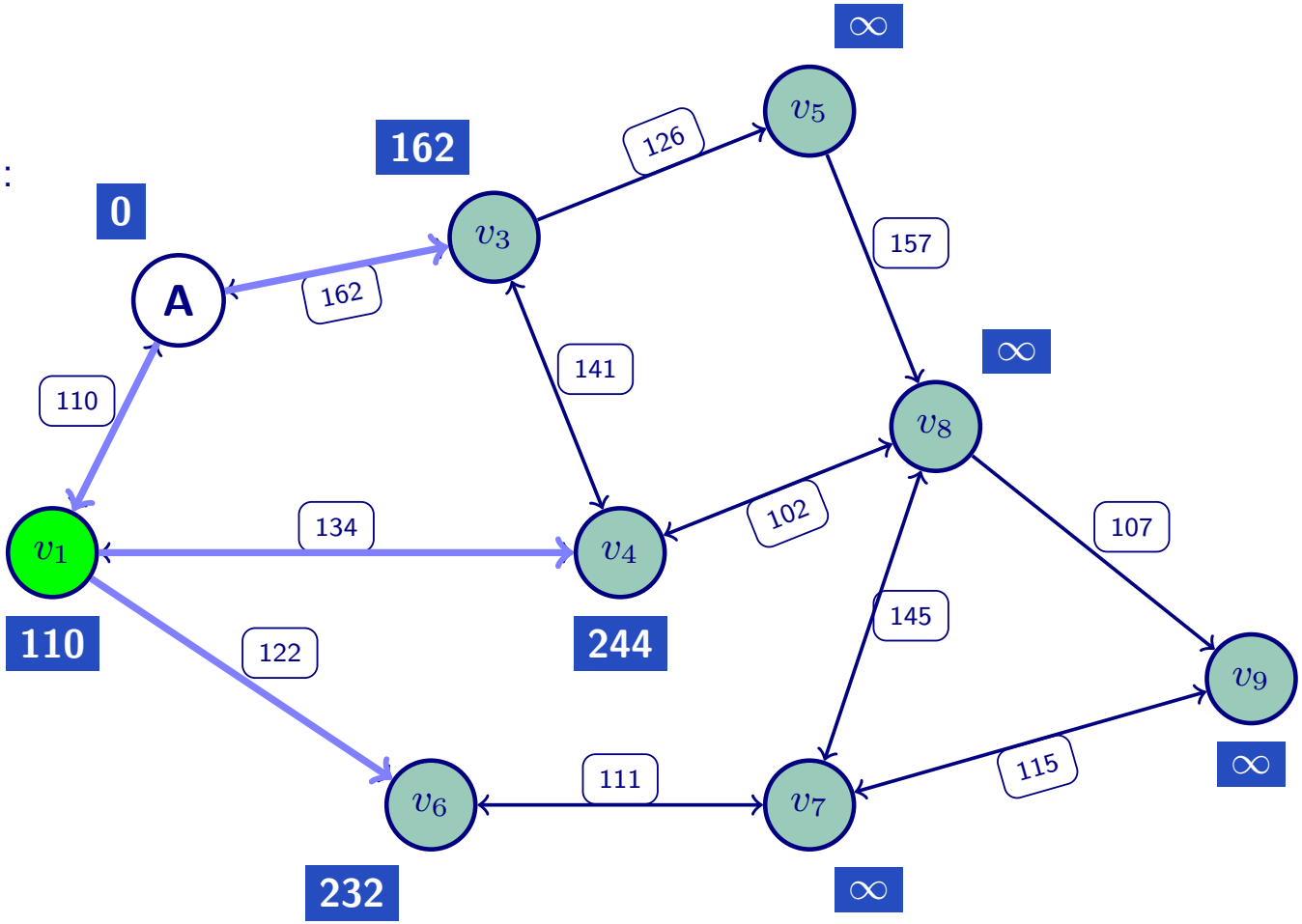
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

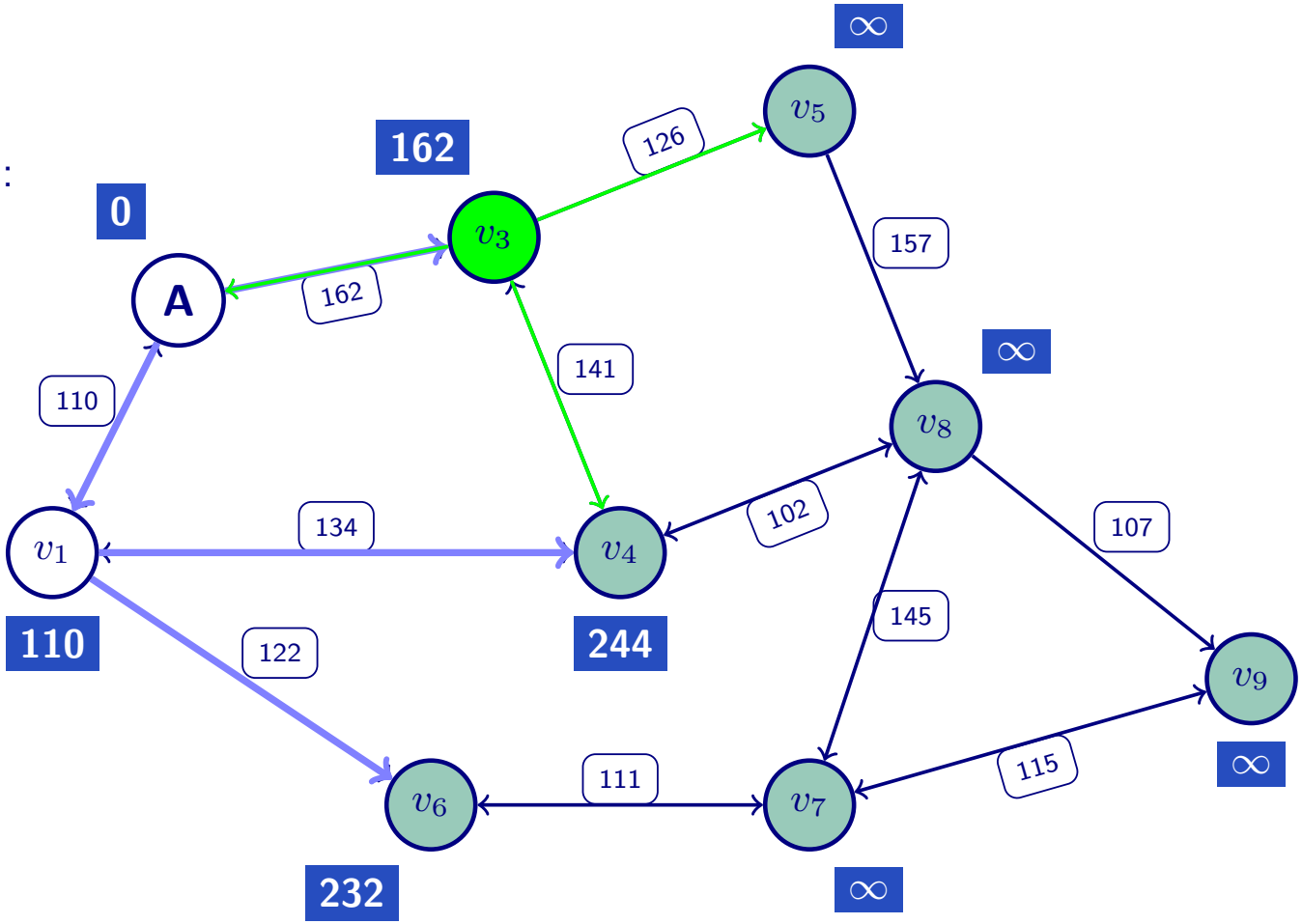
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

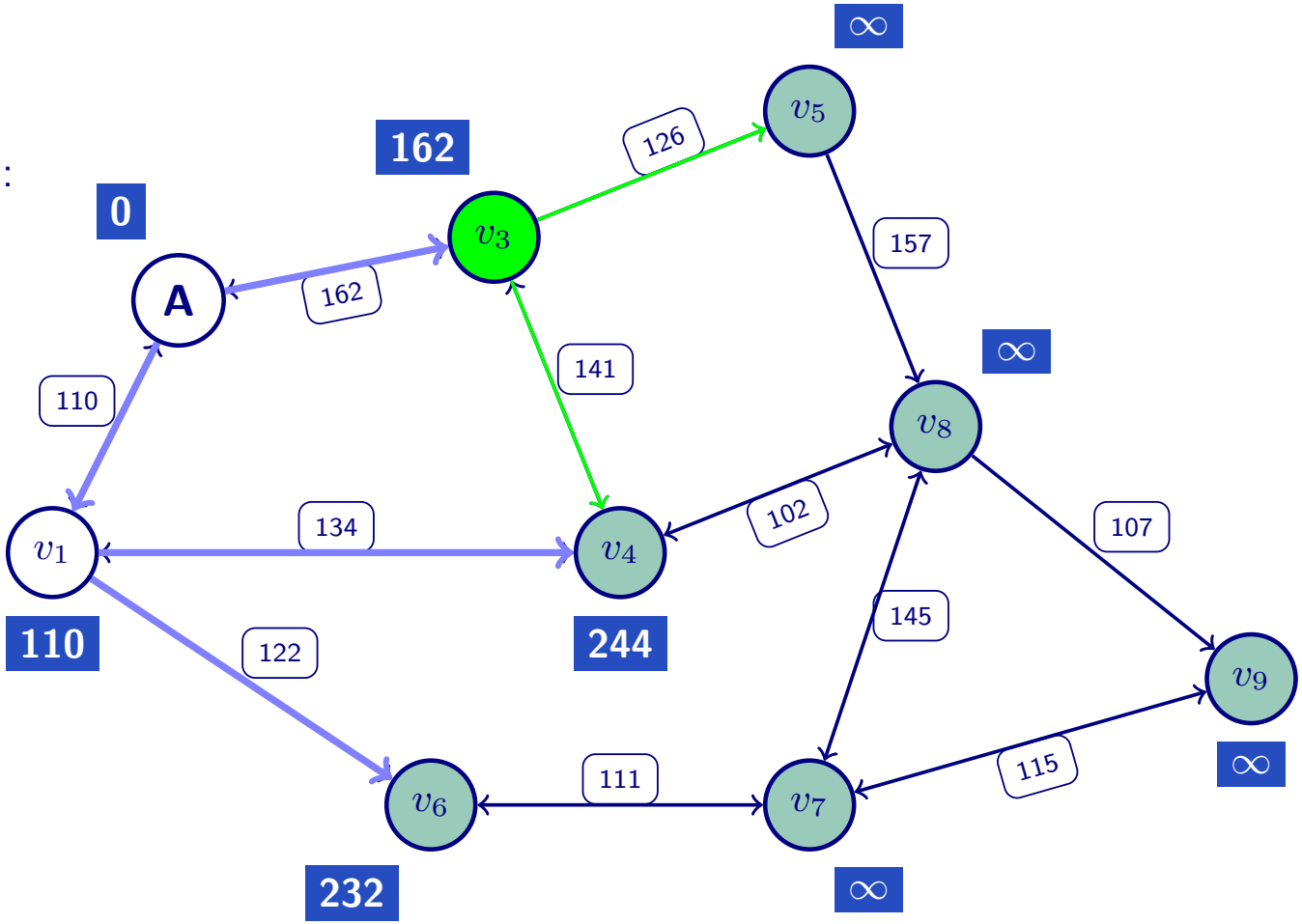
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

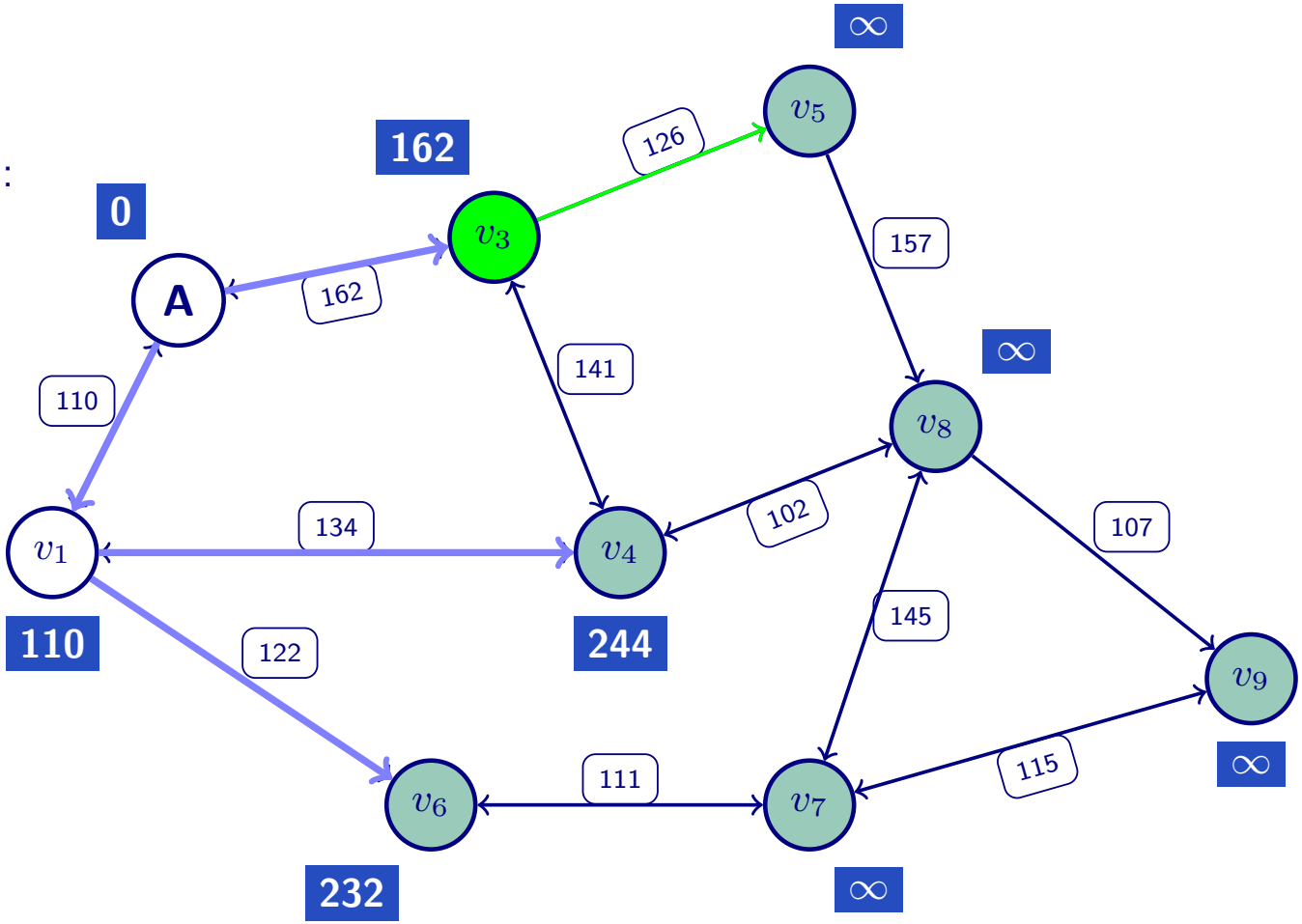
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

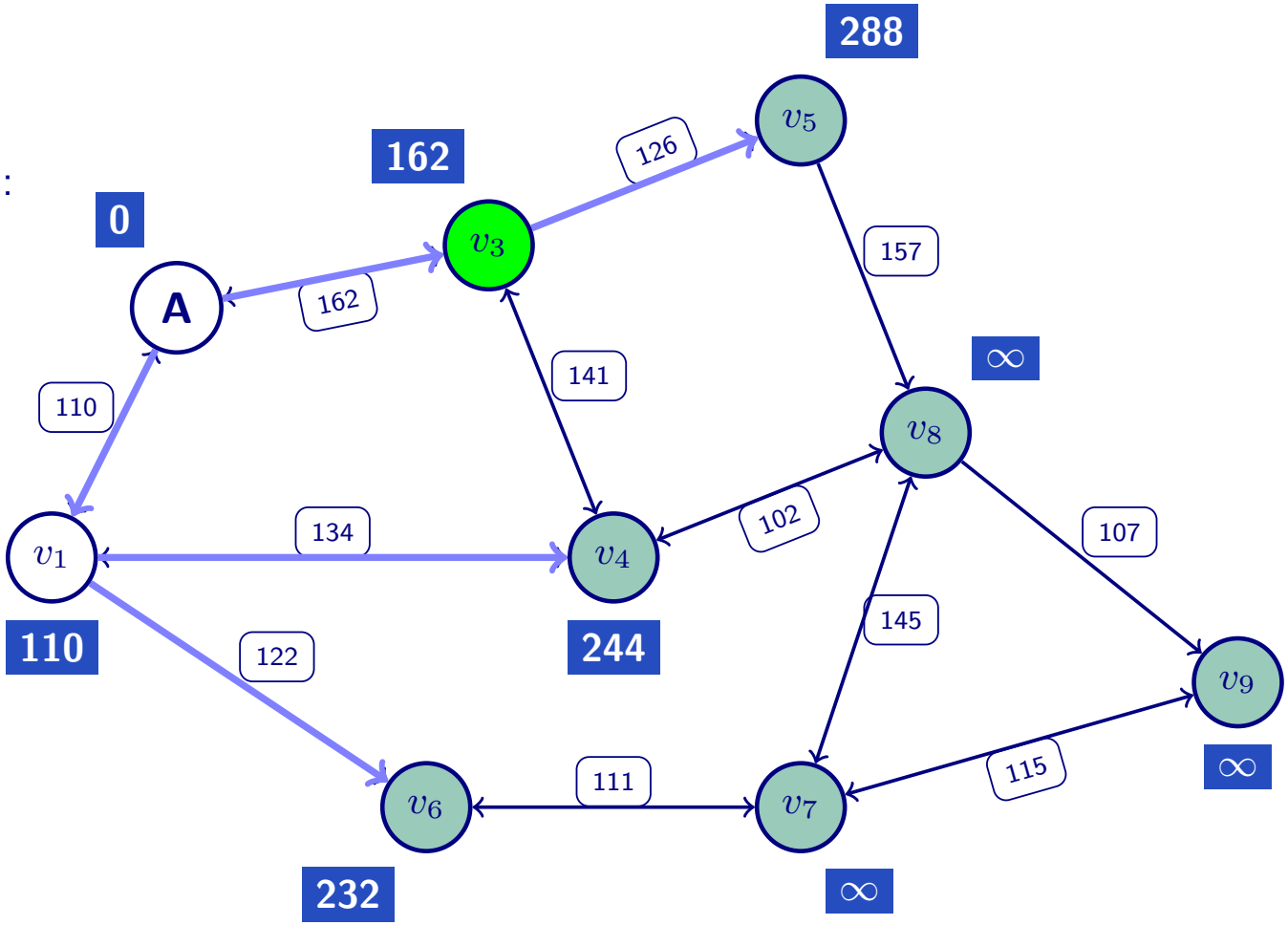
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

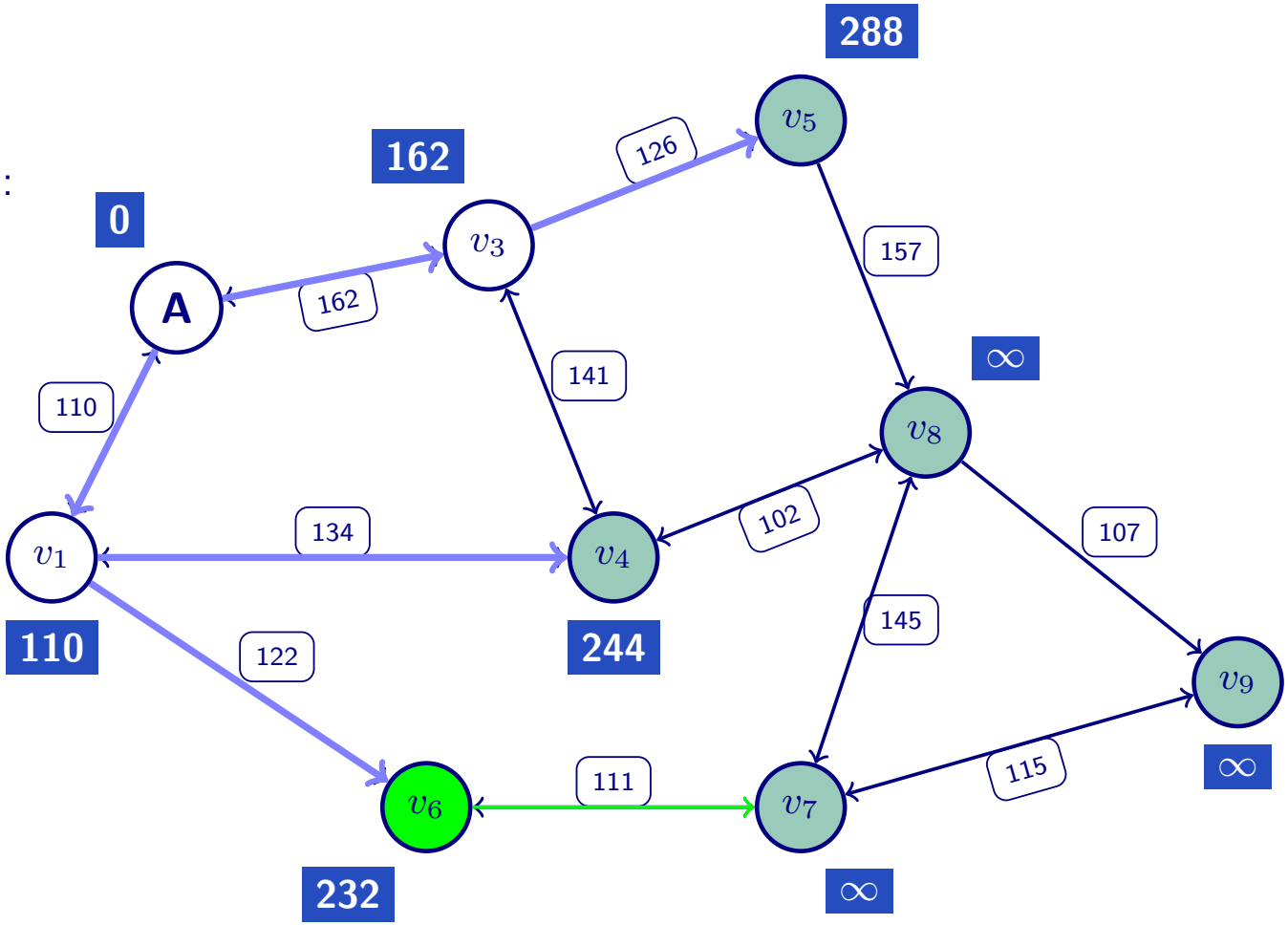
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

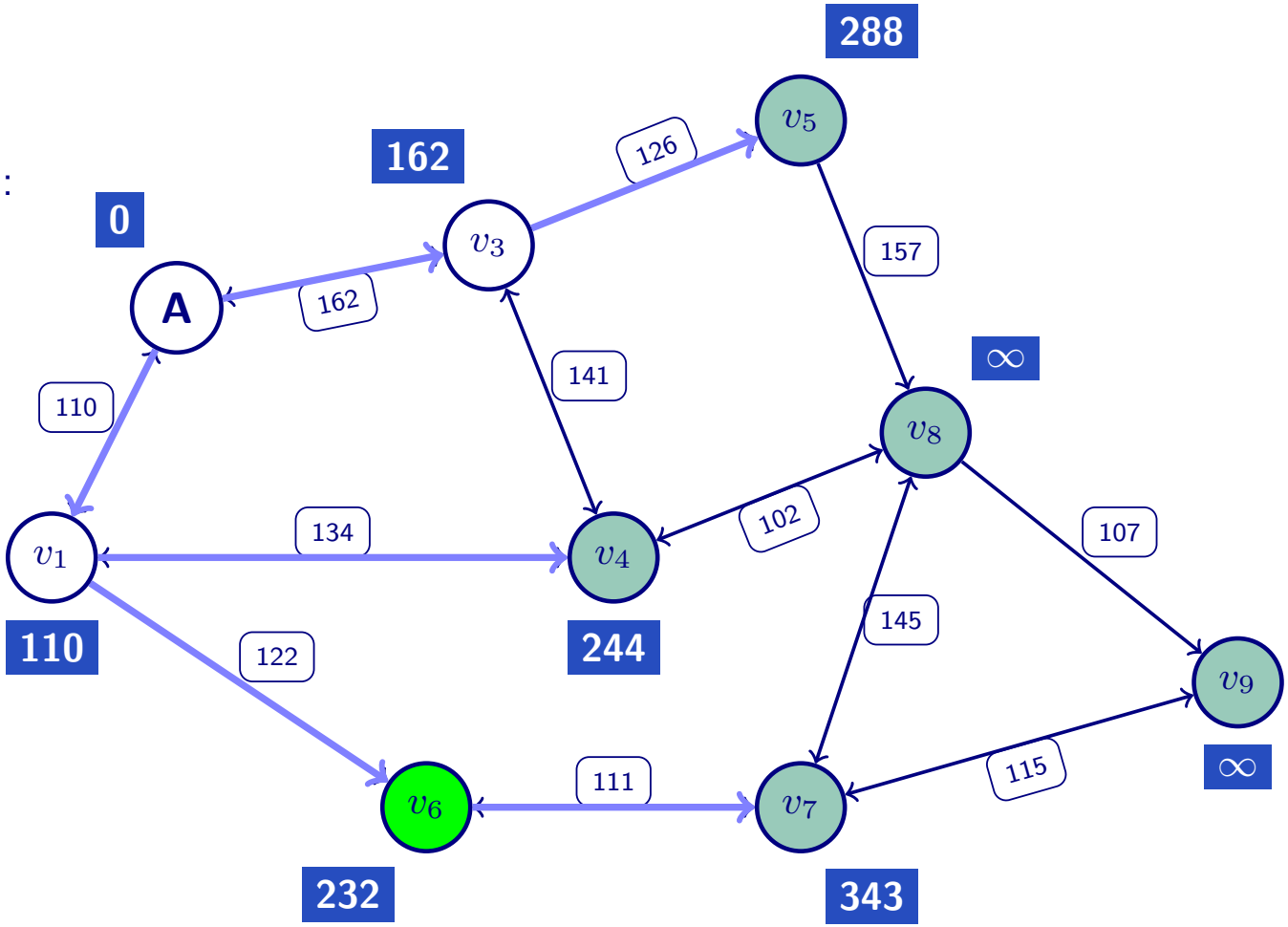
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

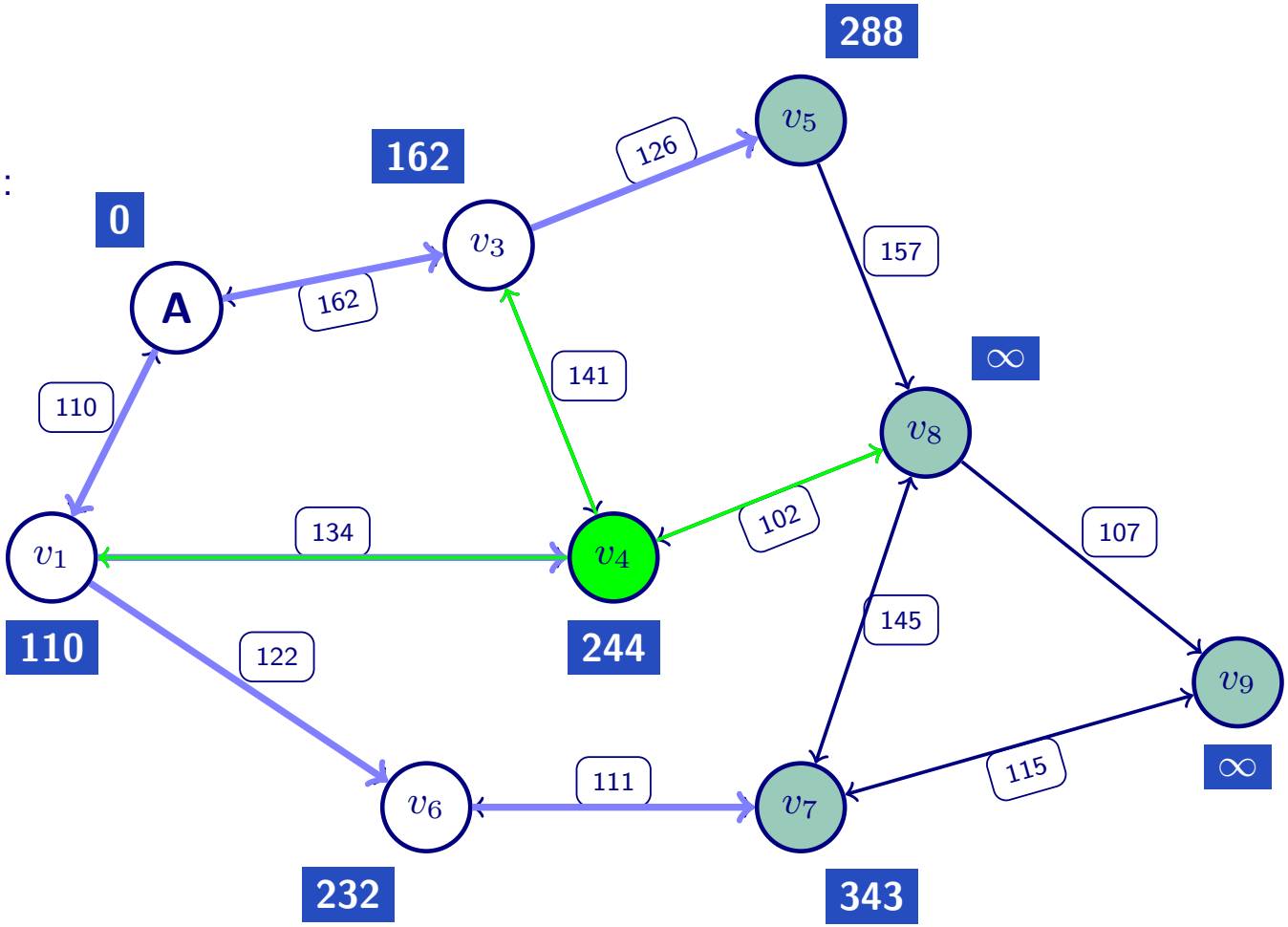
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

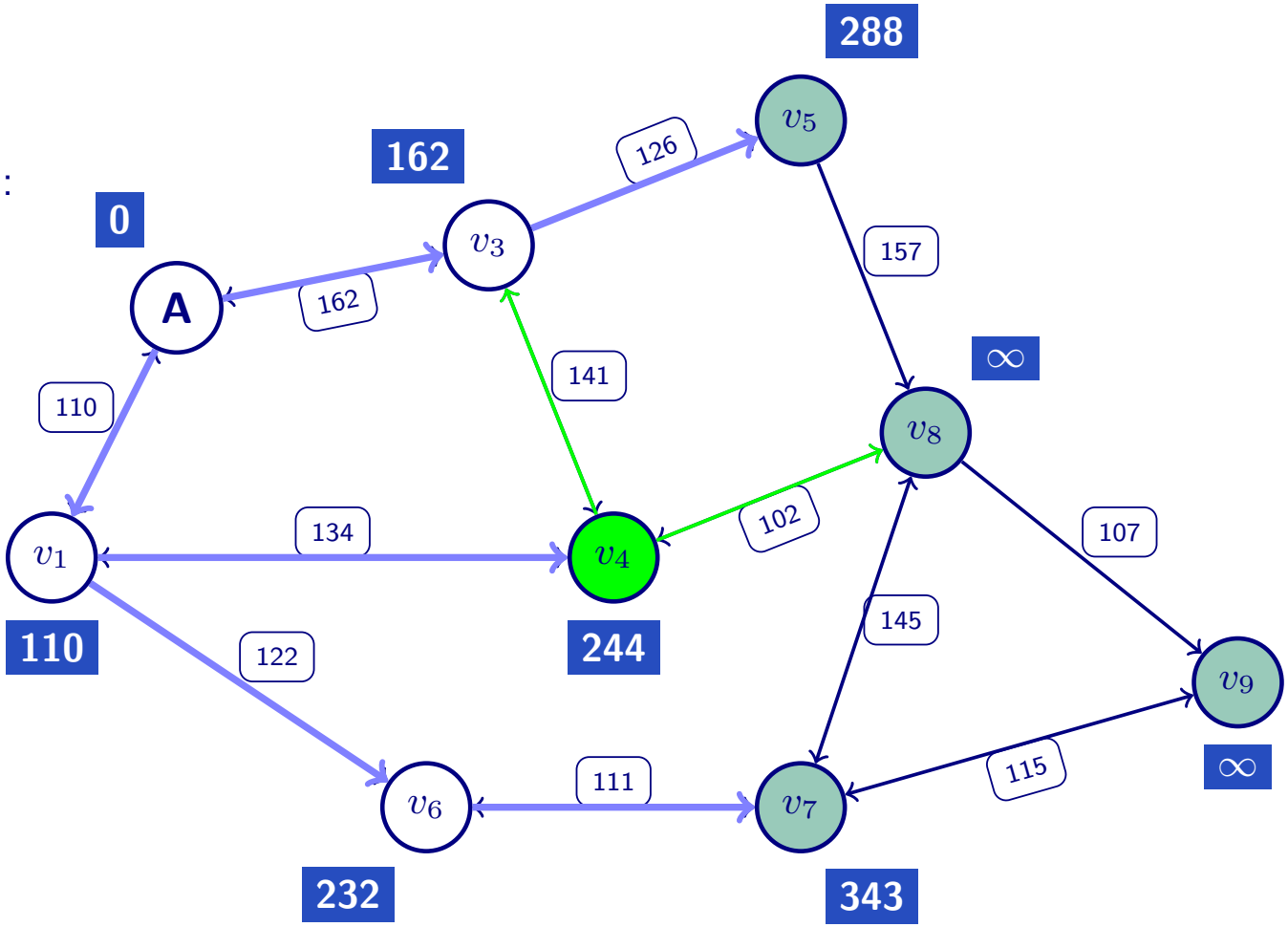
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

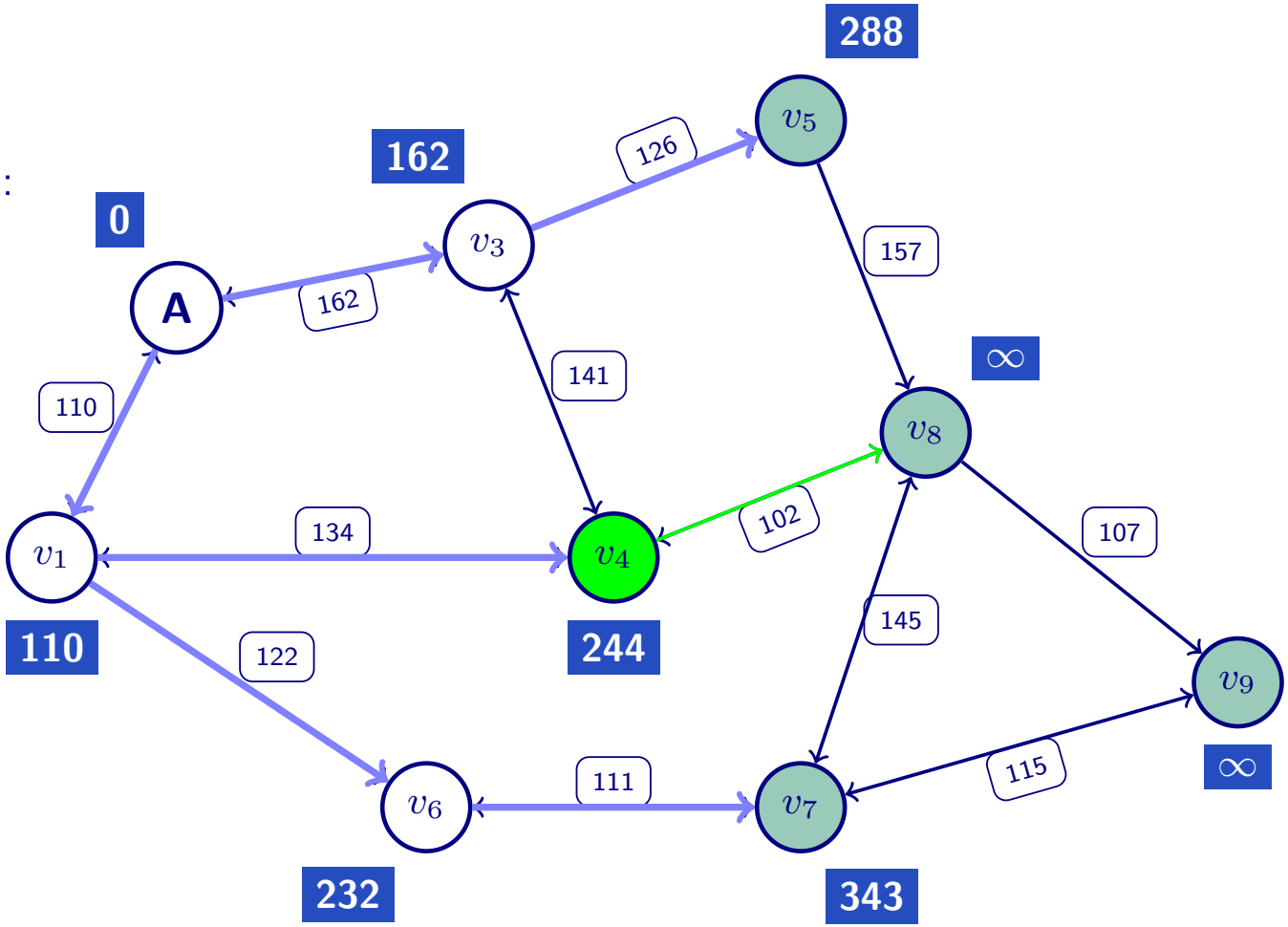
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

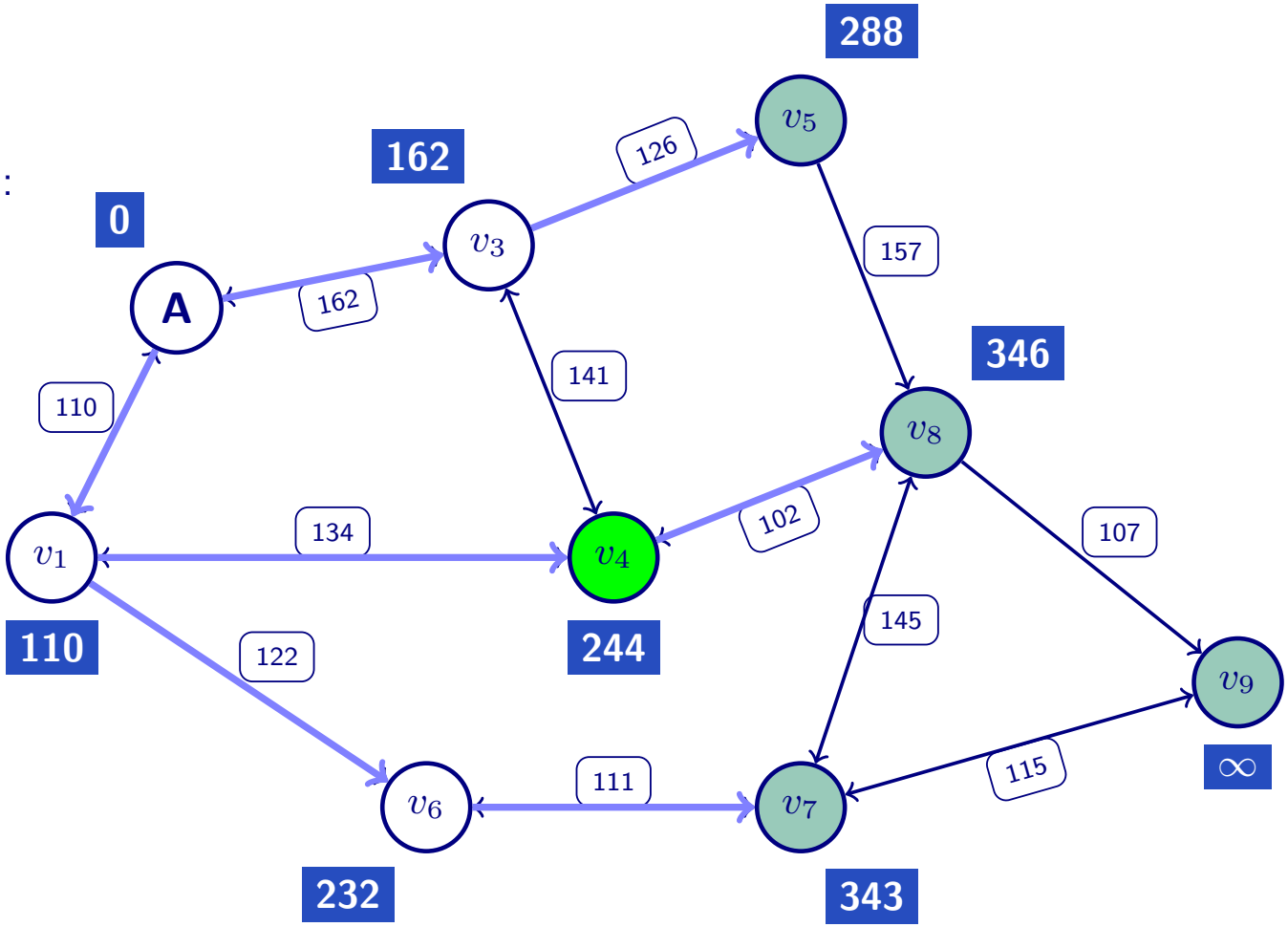
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

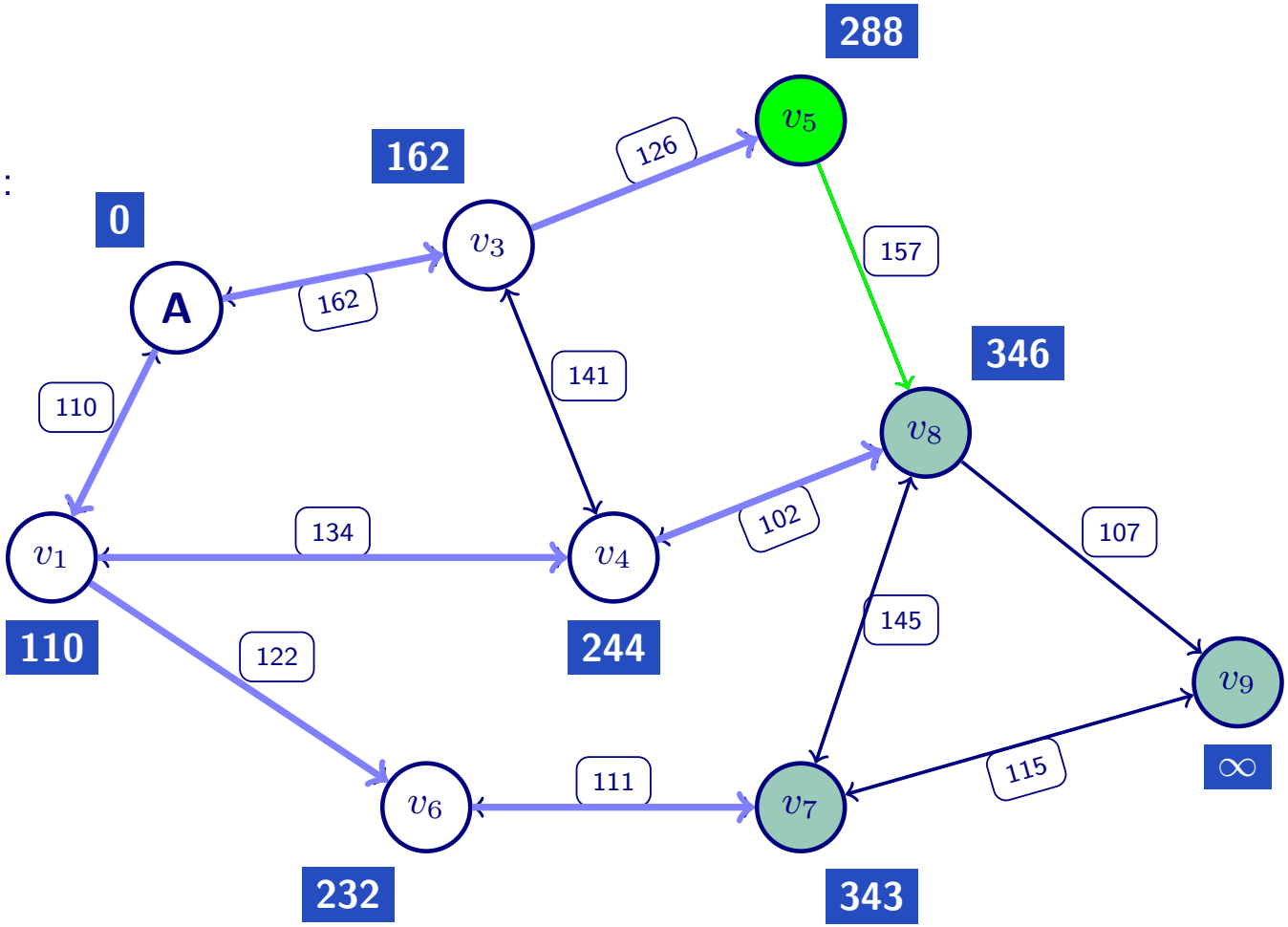
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

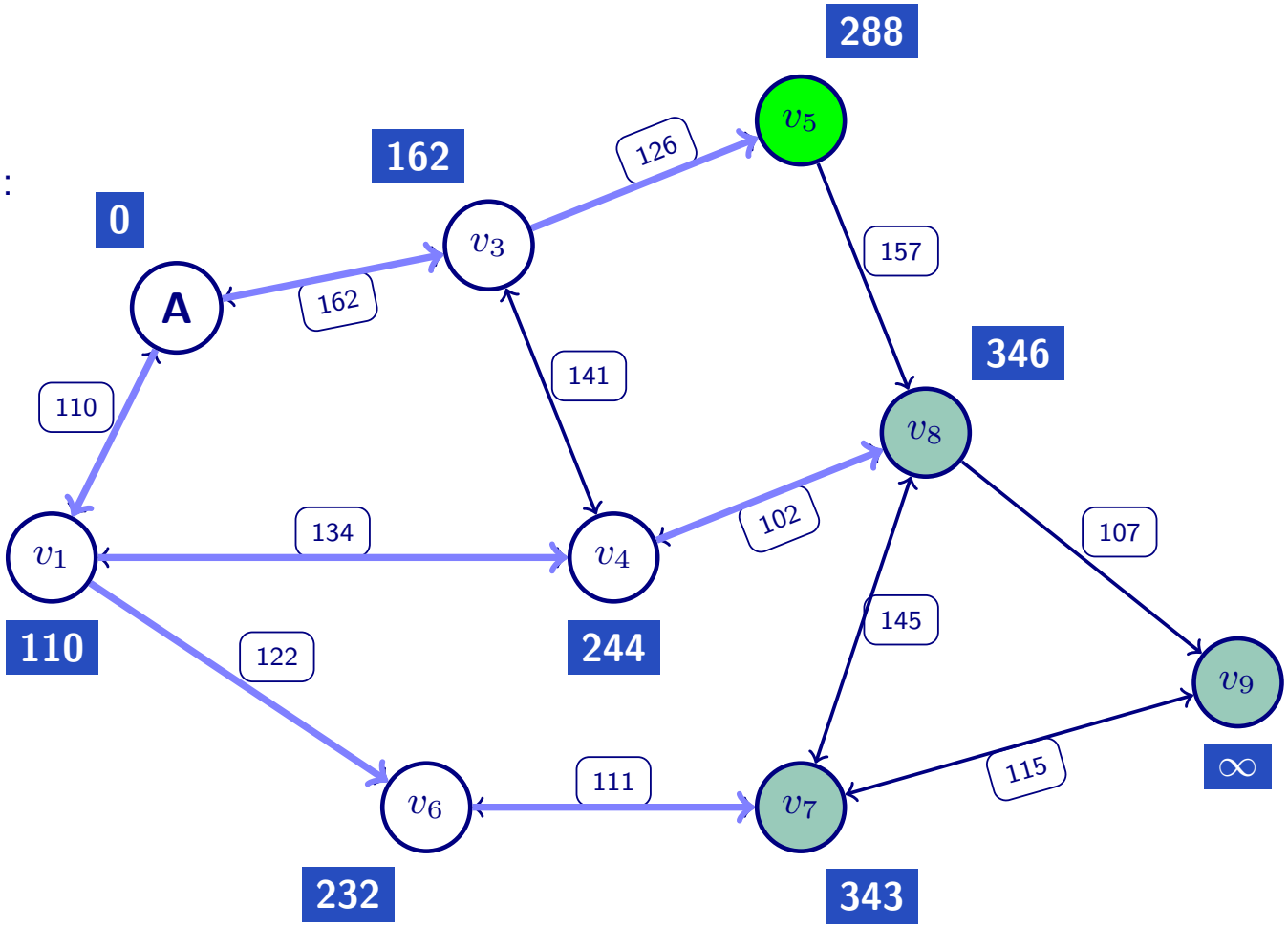
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

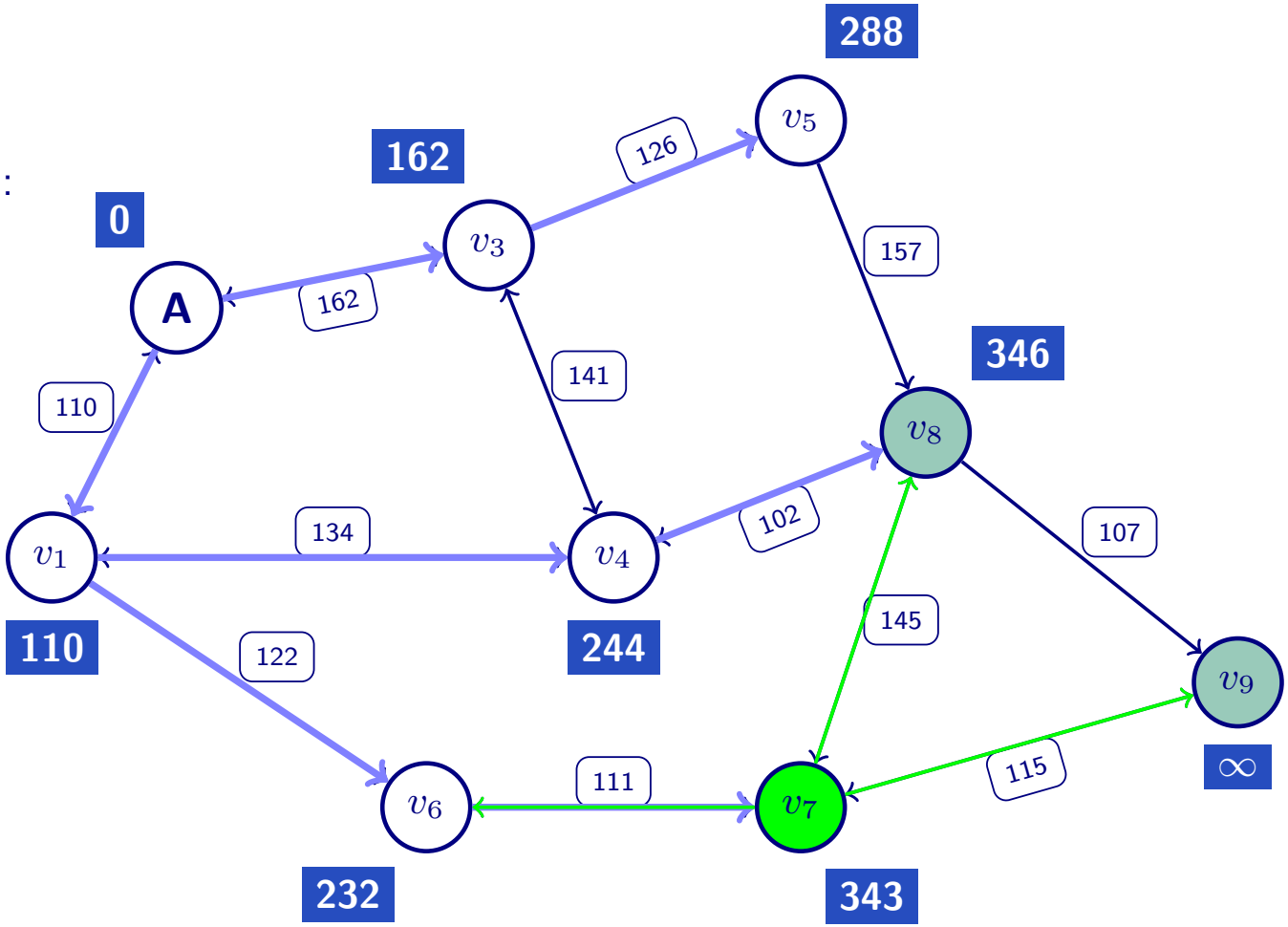
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

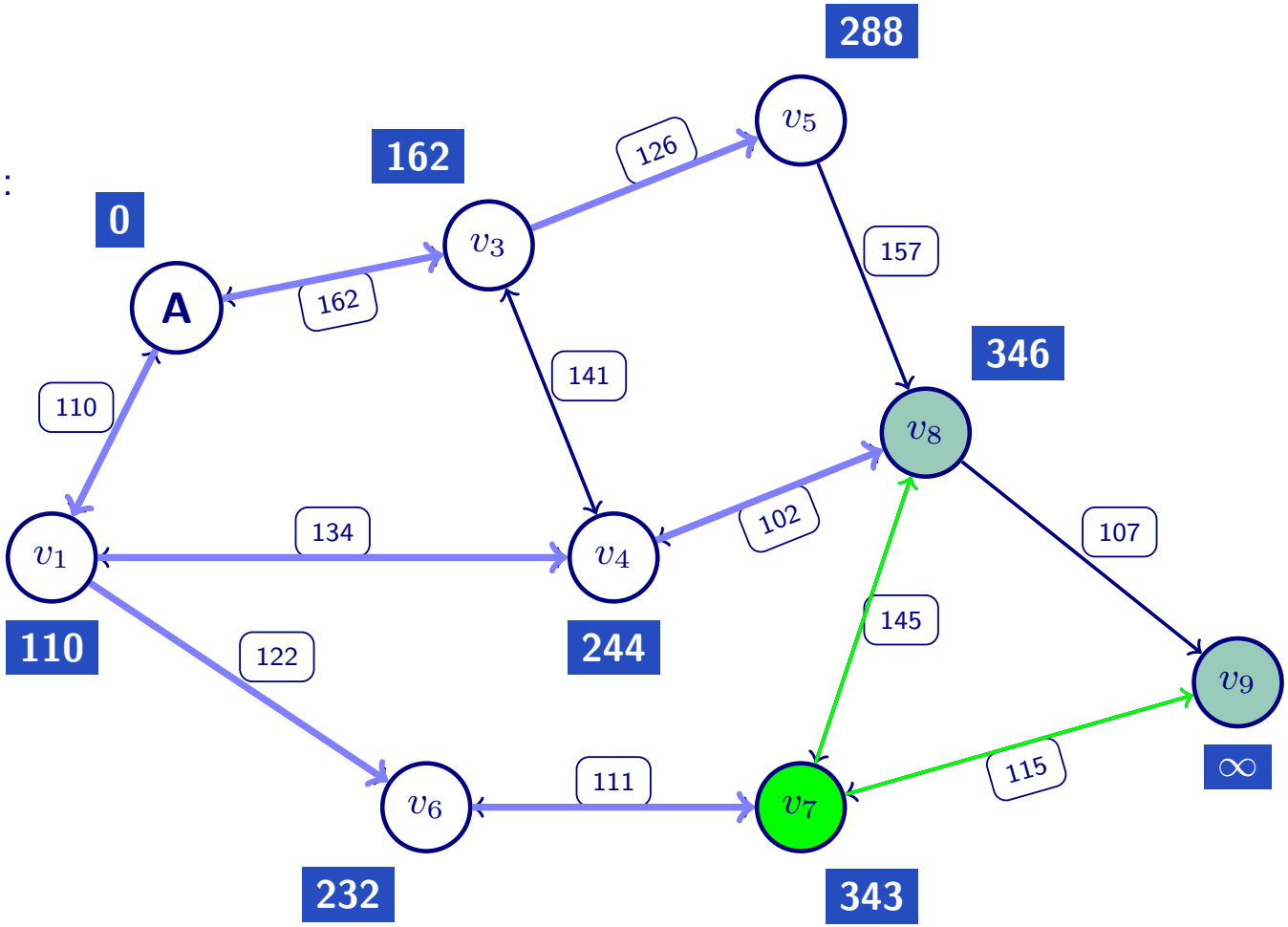
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

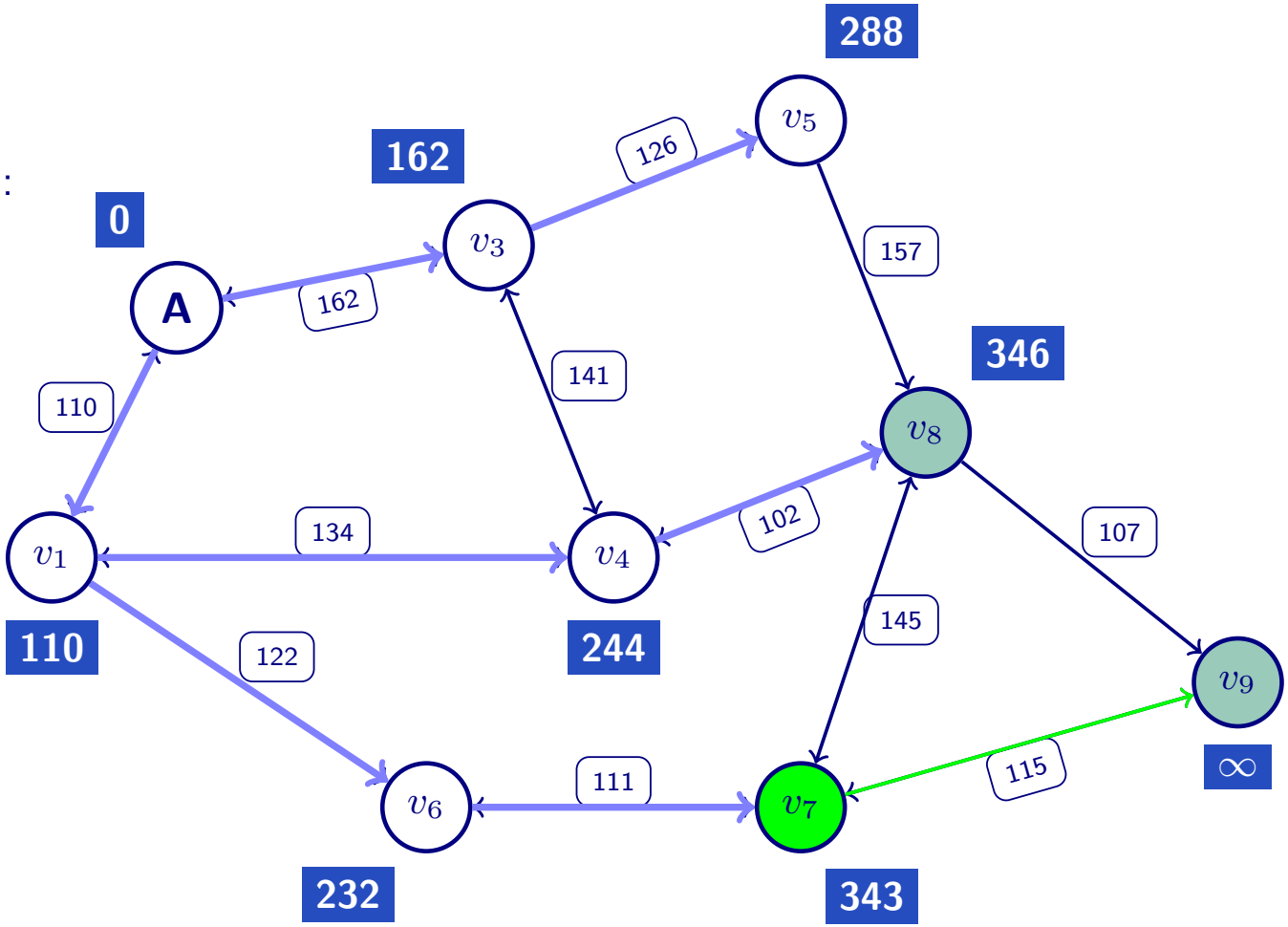
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

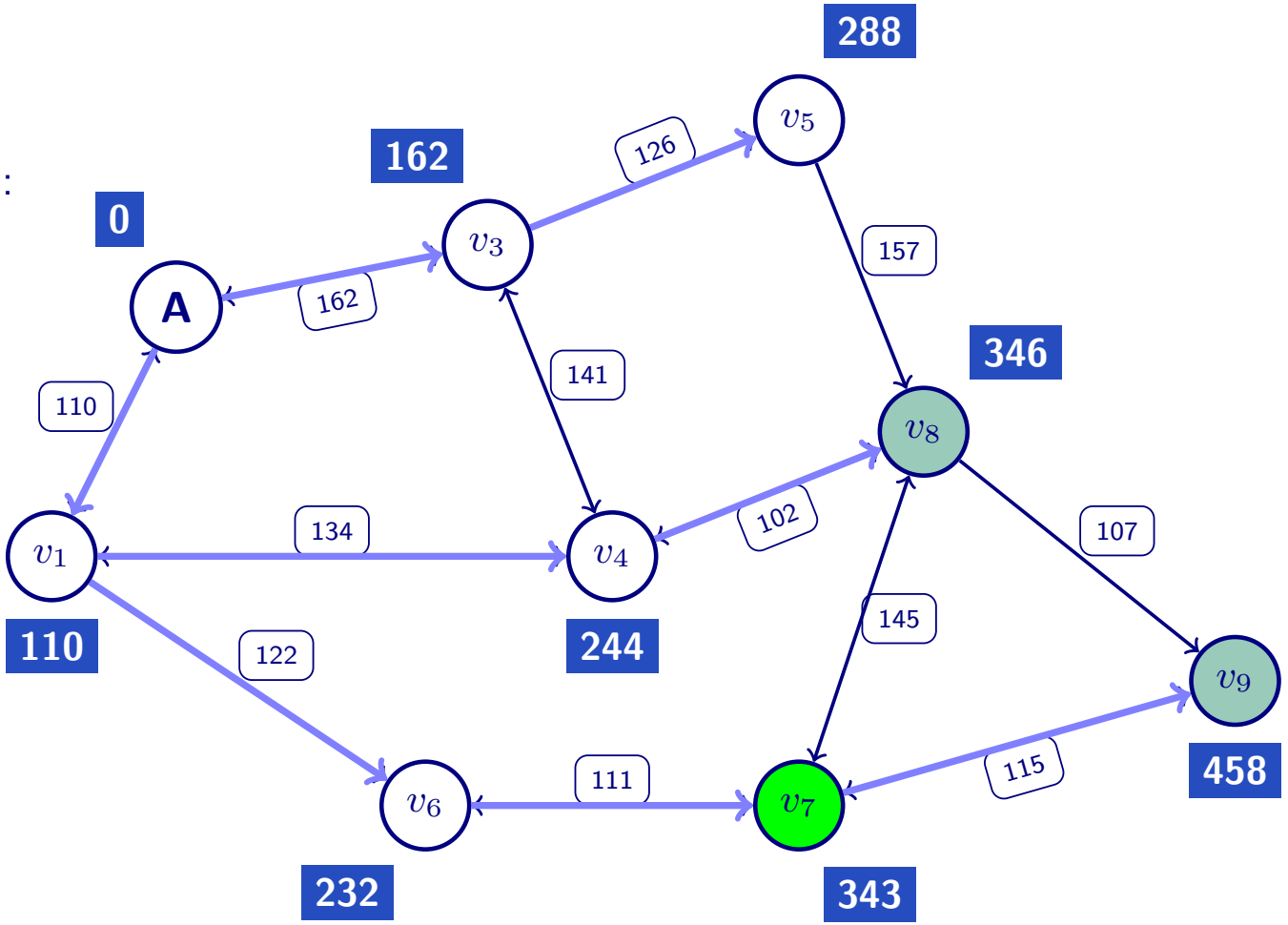
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

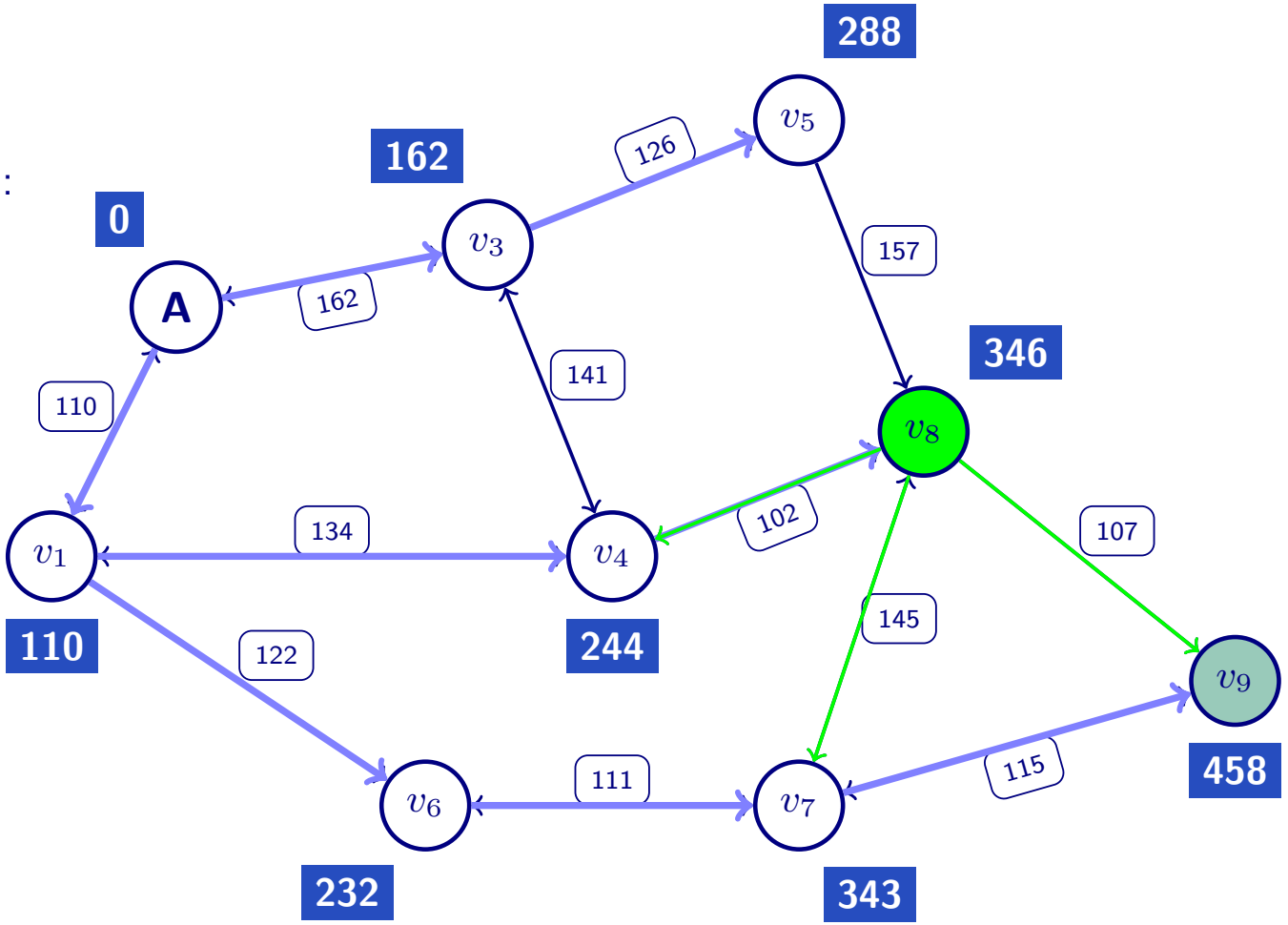
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

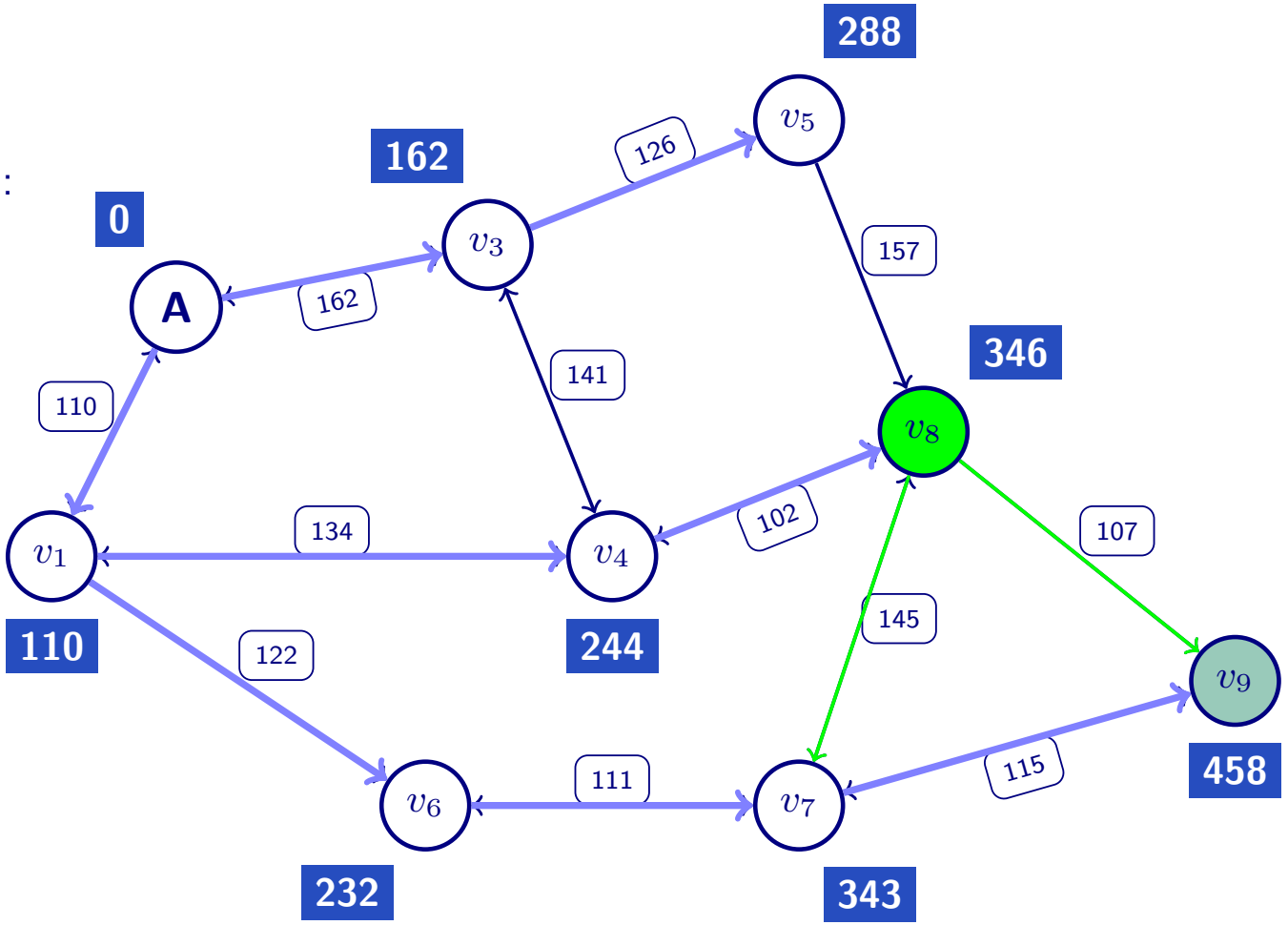
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

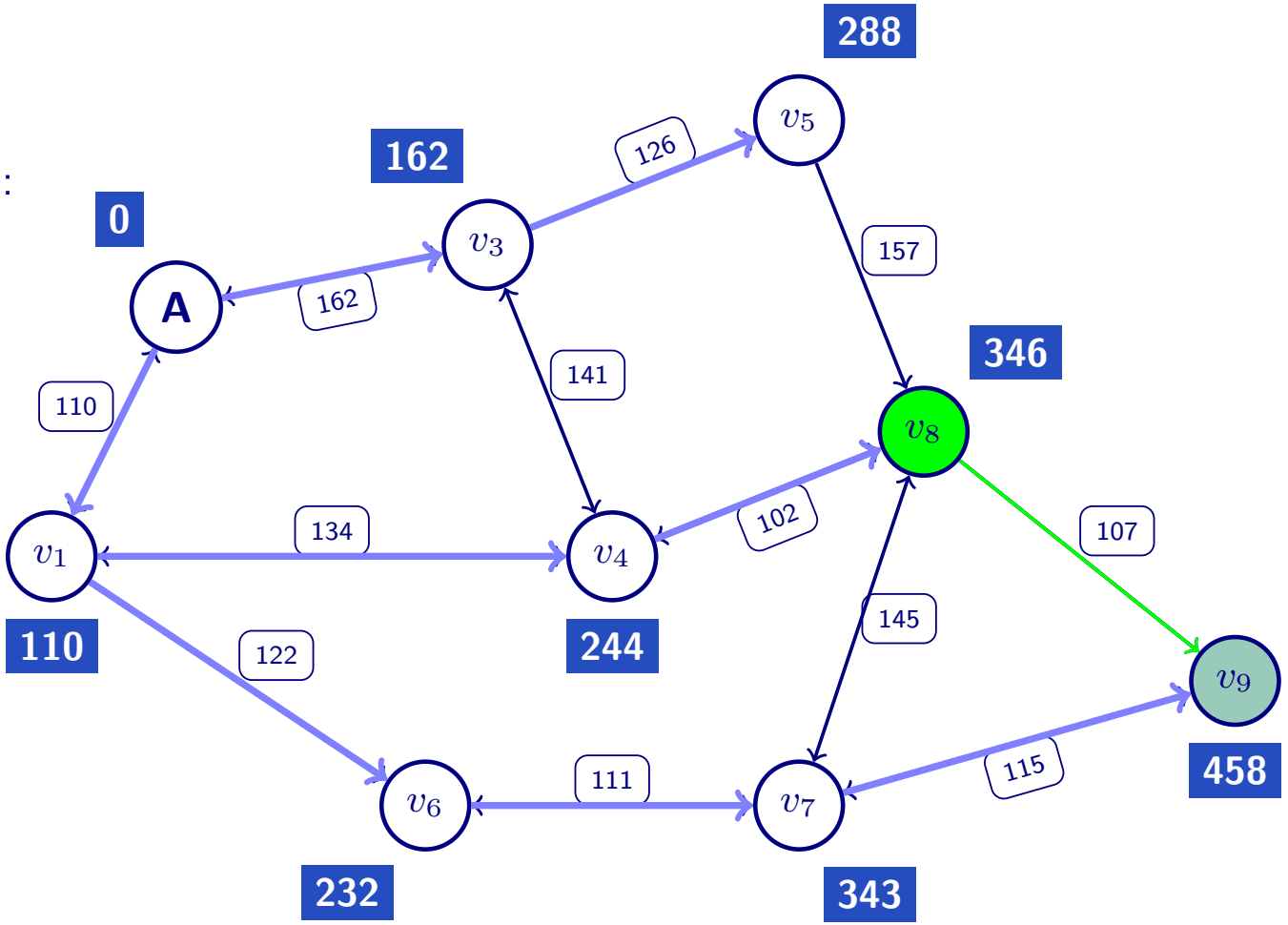
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

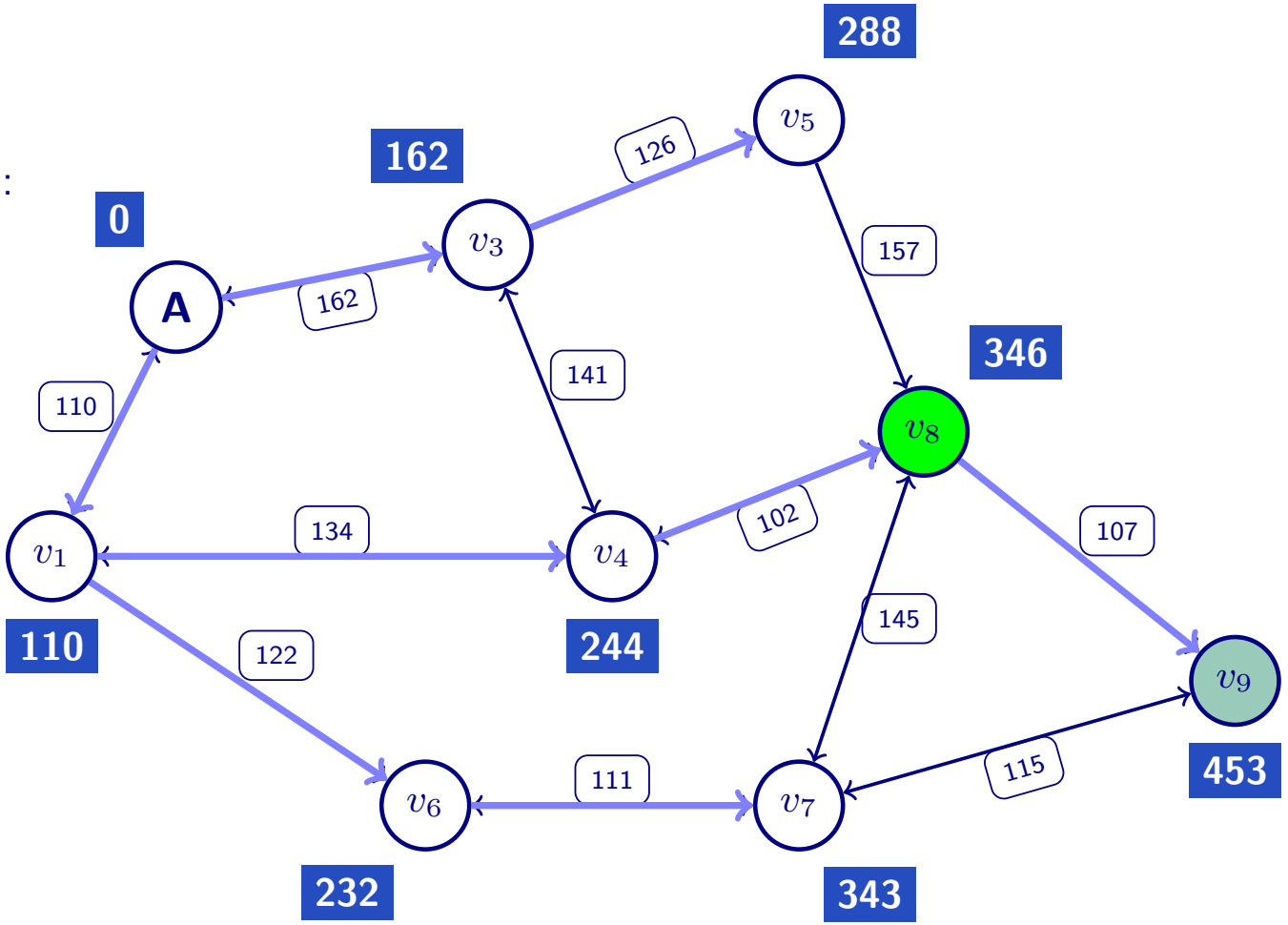
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

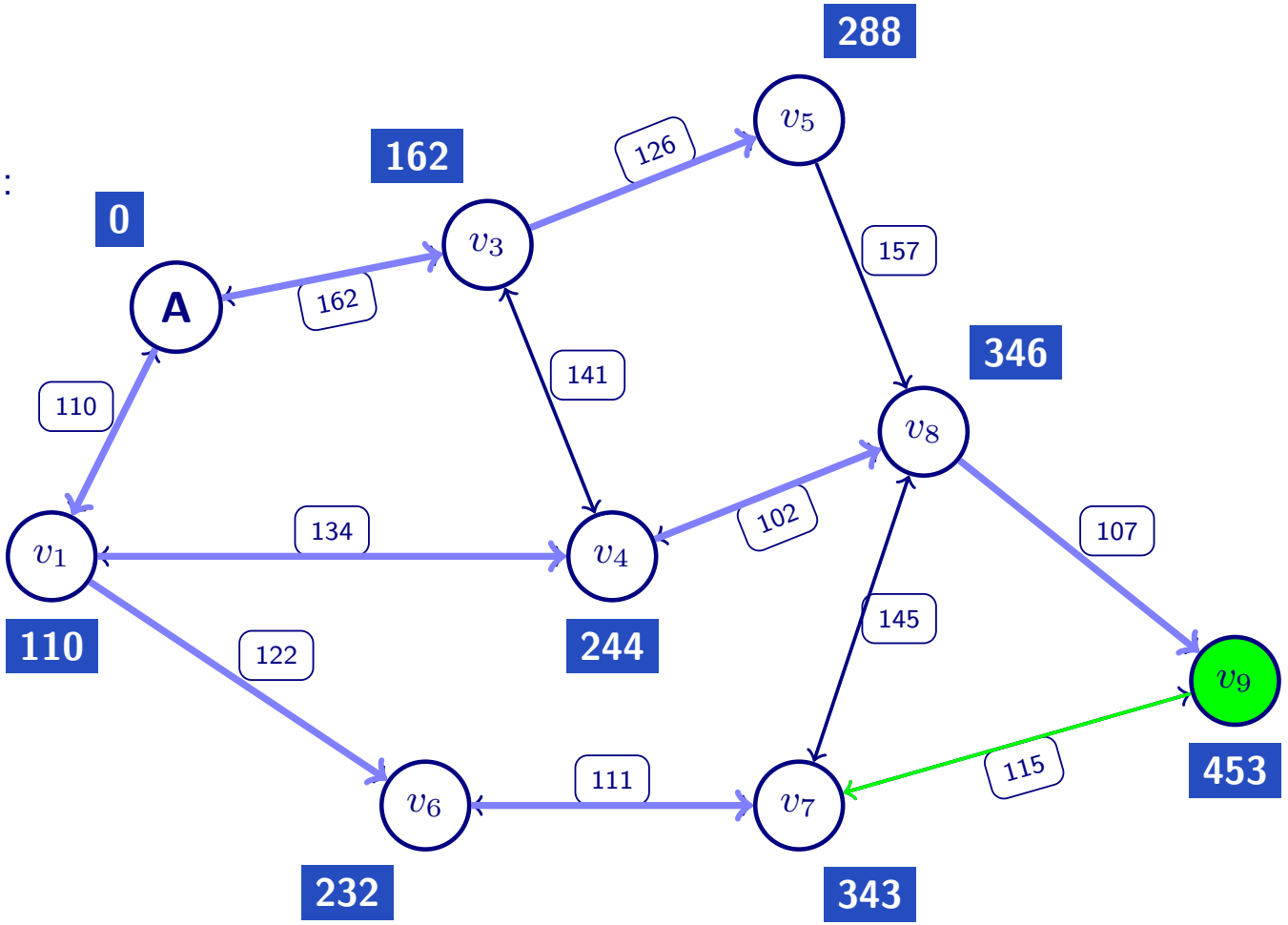
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree



▷ Dijkstra's algorithm computes a shortest path tree from start node A to all other nodes:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

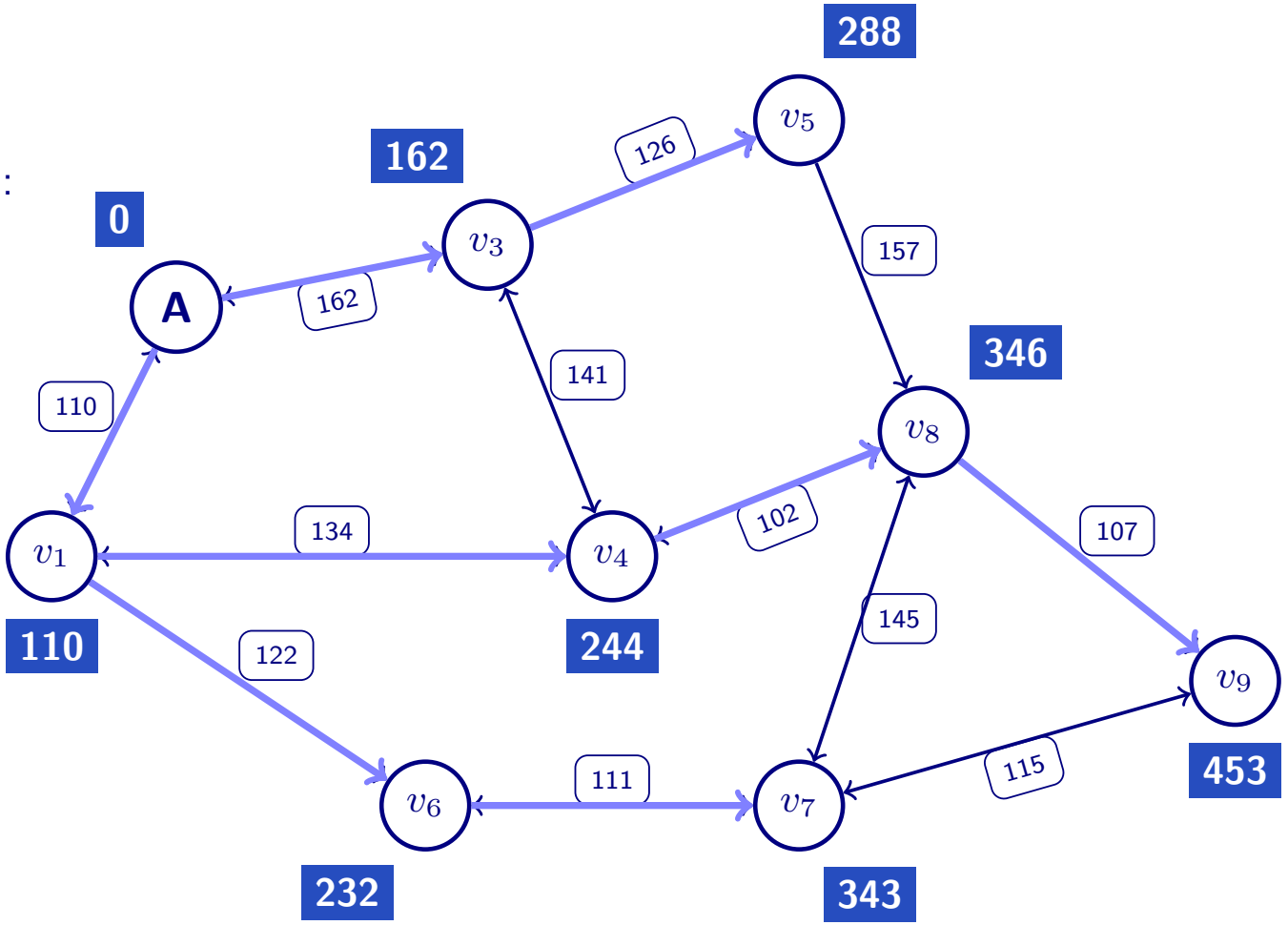
set $\mathrm{pred}(w) := v$

...remove v from R

R remaining nodes

$d(v)$ min distance to A

predecessor tree

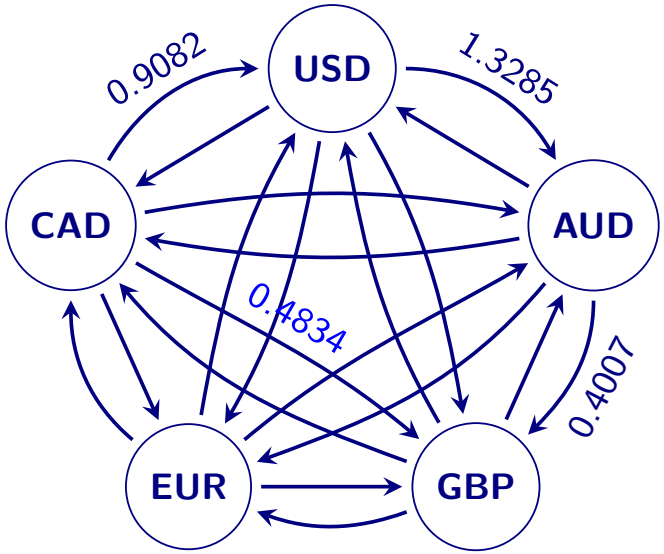


▷ Exchange 100 Mio Can-$ into £

	USD	GBP	CAD	EUR	AUD
USD	1	1.8786	0.9082	1.2953	0.7527
GBP	0.5323	1	0.4834	0.6895	0.4007
CAD	1.1010	2.0685	1	1.4262	0.8288
EUR	0.7720	1.4503	0.7011	1	0.5811
AUD	1.3285	2.4956	1.2065	1.7208	1

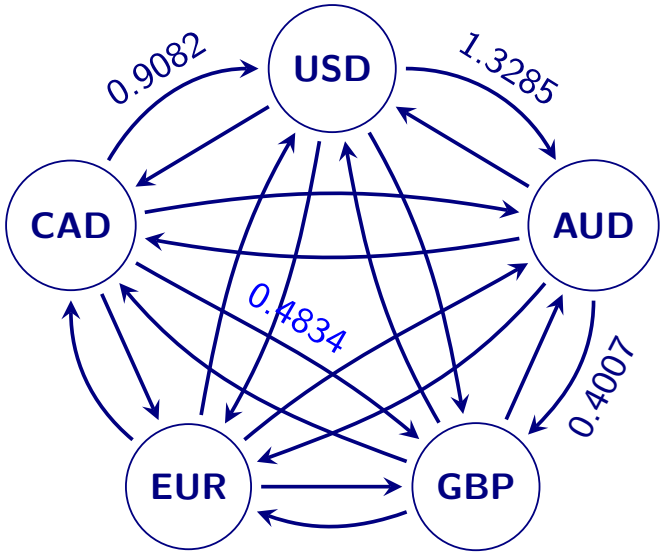
▷ Exchange 100 Mio Can-$ into £

	USD	GBP	CAD	EUR	AUD
USD	1	1.8786	0.9082	1.2953	0.7527
GBP	0.5323	1	0.4834	0.6895	0.4007
CAD	1.1010	2.0685	1	1.4262	0.8288
EUR	0.7720	1.4503	0.7011	1	0.5811
AUD	1.3285	2.4956	1.2065	1.7208	1



▷ Exchange 100 Mio Can-$ into £

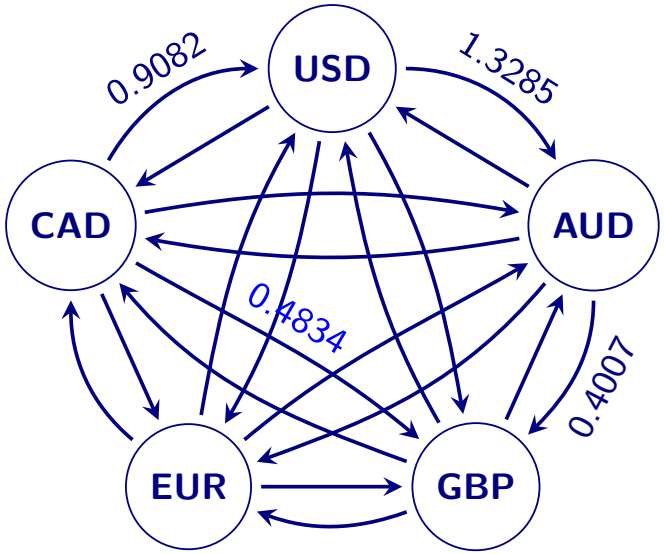
	USD	GBP	CAD	EUR	AUD
USD	1	1.8786	0.9082	1.2953	0.7527
GBP	0.5323	1	0.4834	0.6895	0.4007
CAD	1.1010	2.0685	1	1.4262	0.8288
EUR	0.7720	1.4503	0.7011	1	0.5811
AUD	1.3285	2.4956	1.2065	1.7208	1



▷ Exchanging **CAD** → **USD** → **AUD** → **GBP**:

▷ Exchange 100 Mio Can-$ into £

	USD	GBP	CAD	EUR	AUD
USD	1	1.8786	0.9082	1.2953	0.7527
GBP	0.5323	1	0.4834	0.6895	0.4007
CAD	1.1010	2.0685	1	1.4262	0.8288
EUR	0.7720	1.4503	0.7011	1	0.5811
AUD	1.3285	2.4956	1.2065	1.7208	1

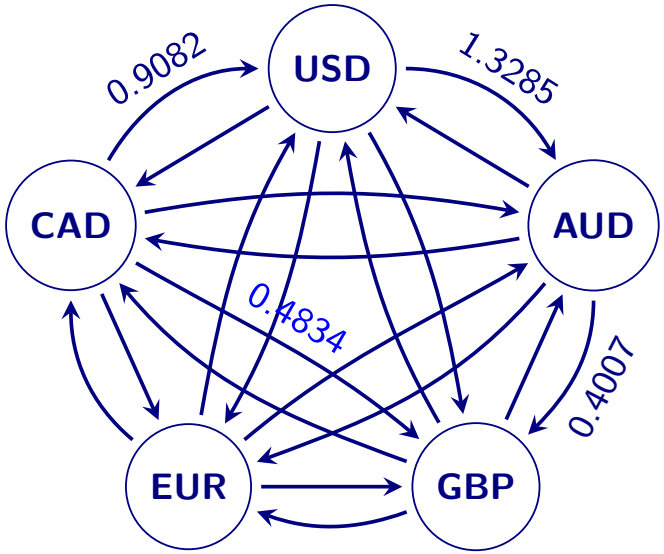


▷ Exchanging **CAD** → **USD** → **AUD** → **GBP**:

➡ Rate: $0.9082 \cdot 1.3285 \cdot 0.4007 = 0.483462 > 0.4834$

▷ Exchange 100 Mio Can-$ into £

	USD	GBP	CAD	EUR	AUD
USD	1	1.8786	0.9082	1.2953	0.7527
GBP	0.5323	1	0.4834	0.6895	0.4007
CAD	1.1010	2.0685	1	1.4262	0.8288
EUR	0.7720	1.4503	0.7011	1	0.5811
AUD	1.3285	2.4956	1.2065	1.7208	1



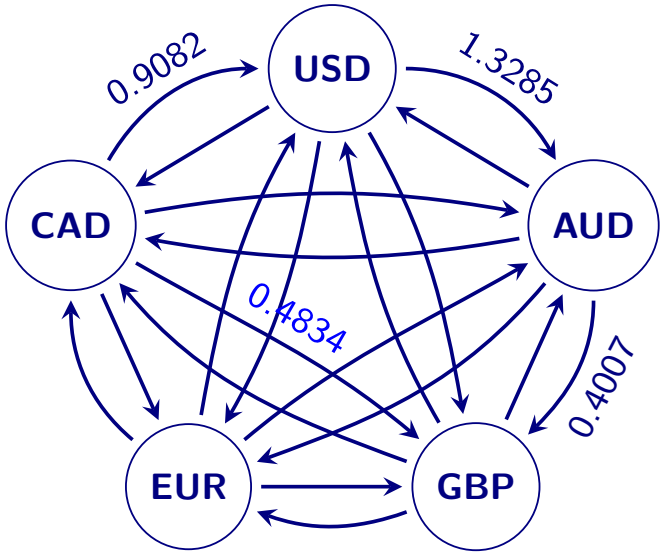
▷ Exchanging **CAD** → **USD** → **AUD** → **GBP**:

➡ Rate: $0.9082 \cdot 1.3285 \cdot 0.4007 = 0.483462 > 0.4834$

➡ What is the best possible exchange rate?

▷ Exchange 100 Mio Can-$ into £

	USD	GBP	CAD	EUR	AUD
USD	1	1.8786	0.9082	1.2953	0.7527
GBP	0.5323	1	0.4834	0.6895	0.4007
CAD	1.1010	2.0685	1	1.4262	0.8288
EUR	0.7720	1.4503	0.7011	1	0.5811
AUD	1.3285	2.4956	1.2065	1.7208	1



▷ Exchanging **CAD** → **USD** → **AUD** → **GBP**:

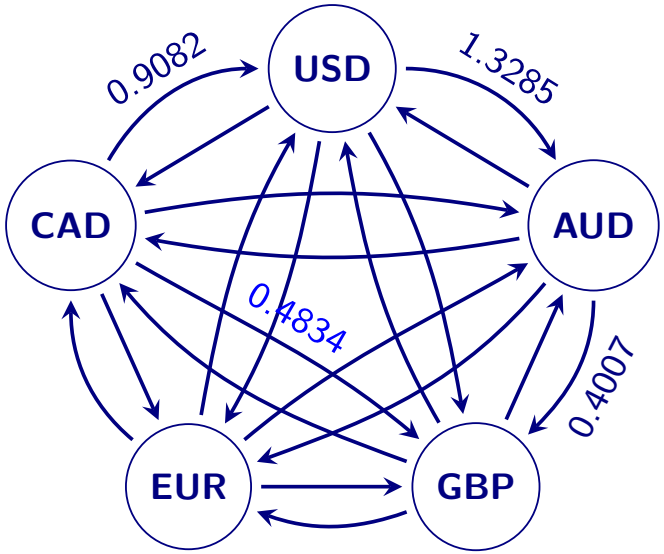
➡ Rate: $0.9082 \cdot 1.3285 \cdot 0.4007 = 0.483462 > 0.4834$

➡ What is the best possible exchange rate?

➡ Find a path from **CAD** to **GBP** with highest possible product!

▷ Exchange 100 Mio Can-$ into £

	USD	GBP	CAD	EUR	AUD
USD	1	1.8786	0.9082	1.2953	0.7527
GBP	0.5323	1	0.4834	0.6895	0.4007
CAD	1.1010	2.0685	1	1.4262	0.8288
EUR	0.7720	1.4503	0.7011	1	0.5811
AUD	1.3285	2.4956	1.2065	1.7208	1



▷ Exchanging **CAD** → **USD** → **AUD** → **GBP**:

➡ Rate: $0.9082 \cdot 1.3285 \cdot 0.4007 = 0.483462 > 0.4834$

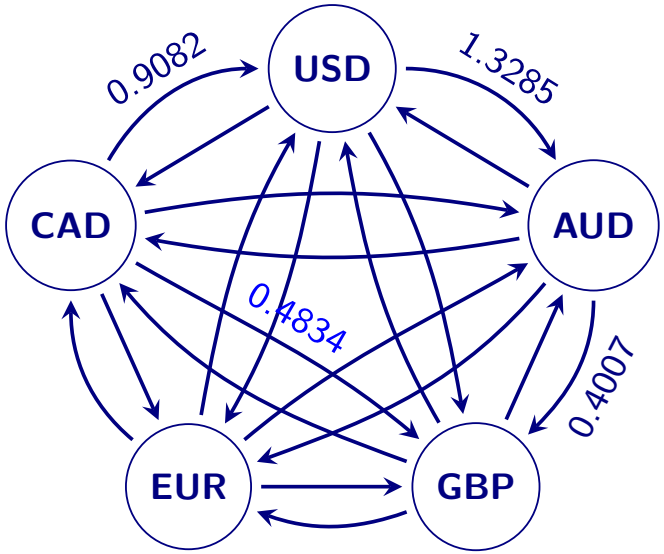
➡ What is the best possible exchange rate?

➡ Find a path from **CAD** to **GBP** with highest possible product!

▷ Problem: subsequent steps result in multiplying (rates) instead of adding (lengths)

▷ Exchange 100 Mio Can-$ into £

	USD	GBP	CAD	EUR	AUD
USD	1	1.8786	0.9082	1.2953	0.7527
GBP	0.5323	1	0.4834	0.6895	0.4007
CAD	1.1010	2.0685	1	1.4262	0.8288
EUR	0.7720	1.4503	0.7011	1	0.5811
AUD	1.3285	2.4956	1.2065	1.7208	1



▷ Exchanging **CAD** → **USD** → **AUD** → **GBP**:

➡ Rate: $0.9082 \cdot 1.3285 \cdot 0.4007 = 0.483462 > 0.4834$

➡ What is the best possible exchange rate?

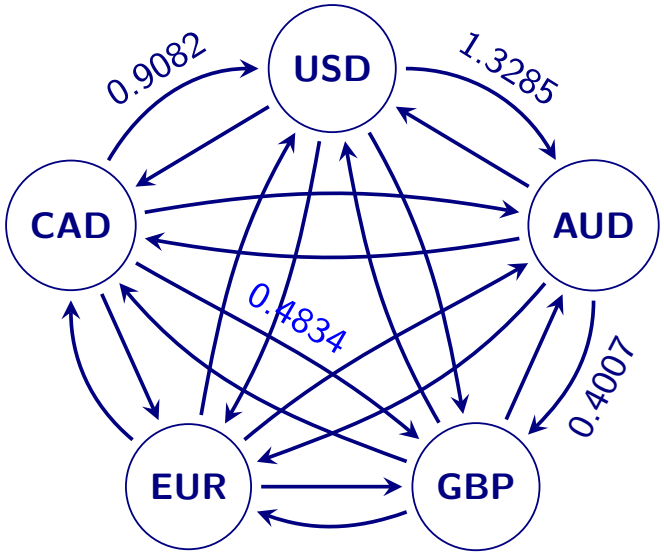
➡ Find a path from **CAD** to **GBP** with highest possible product!

▷ Problem: subsequent steps result in multiplying (rates) instead of adding (lengths)

➡ Use logarithm: $\log(a \cdot b) = \log a + \log b$ and $\log a < \log b \iff a < b$

▷ Exchange 100 Mio Can-$ into £

	USD	GBP	CAD	EUR	AUD
USD	1	1.8786	0.9082	1.2953	0.7527
GBP	0.5323	1	0.4834	0.6895	0.4007
CAD	1.1010	2.0685	1	1.4262	0.8288
EUR	0.7720	1.4503	0.7011	1	0.5811
AUD	1.3285	2.4956	1.2065	1.7208	1



▷ Exchanging **CAD** → **USD** → **AUD** → **GBP**:

➡ Rate: $0.9082 \cdot 1.3285 \cdot 0.4007 = 0.483462 > 0.4834$

➡ What is the best possible exchange rate?

➡ Find a path from **CAD** to **GBP** with highest possible product!

▷ Problem: subsequent steps result in multiplying (rates) instead of adding (lengths)

➡ Use logarithm: $\log(a \cdot b) = \log a + \log b$ and $\log a < \log b \iff a < b$

▷ Plus: search for a longest path (instead of a shortest)

- Count the number of “basic” steps

- Count the number of “basic” steps with respect to the “size” of the input

- Count the number of “basic” steps with respect to the “size” of the input
- basic step: something that takes a fixed constant time

- Count the number of “basic” steps with respect to the “size” of the input
- basic step : something that takes a fixed constant time
- input size : number of elements in the input

- Count the number of “basic” steps with respect to the “size” of the input
- basic step : something that takes a fixed constant time
- input size : number of elements in the input
- Example: Determine the biggest number in a given list

- ▷ Count the number of "basic" steps with respect to the "size" of the input
- ▷ **basic step**: something that takes a fixed constant time
- ▷ **input size**: number of elements in the input
- ▷ Example: Determine the biggest number in a given list
  - Input: List $x_1, \ldots, x_n$ of numbers
  - Algorithm: Set the current biggest number $x_{\max} := -\infty$

    For all i from 1 to n...

    ...if $x_i > x_{\max}$ then set $x_{\max} := x_i$

    ➡ $x_{\max}$ is the biggest number in the input

- ▷ Count the number of "basic" steps with respect to the "size" of the input
- ▷ basic step : something that takes a fixed constant time
- ▷ input size : number of elements in the input
- ▷ Example: Determine the biggest number in a given list
  - Input: List $x_1, \ldots, x_n$ of numbers
  - Algorithm: Set the current biggest number $x_{\max} := -\infty$

    For all i from 1 to n...

    ...if $x_i > x_{\max}$ then set $x_{\max} := x_i$

    ➡ $x_{\max}$ is the biggest number in the input
  - ➡ Input size: n numbers

- ▷ Count the number of "basic" steps with respect to the "size" of the input
- ▷ basic step : something that takes a fixed constant time
- ▷ input size : number of elements in the input
- ▷ Example: Determine the biggest number in a given list
  - Input: List $x_1, \ldots, x_n$ of numbers
  - Algorithm: Set the current biggest number $x_{\max} := -\infty$

    For all i from 1 to n...

    ...if $x_i > x_{\max}$ then set $x_{\max} := x_i$

    ➡ $x_{\max}$ is the biggest number in the input
  - ➡ Input size: n numbers
  - ➡ Runtime:

- ▷ Count the number of "basic" steps with respect to the "size" of the input
- ▷ basic step : something that takes a fixed constant time
- ▷ input size : number of elements in the input
- ▷ Example: Determine the biggest number in a given list
  - Input: List $x_1, \ldots, x_n$ of numbers
  - Algorithm: Set the current biggest number $x_{\max} := -\infty$ ← constant time

    For all i from 1 to n...

    ...if $x_i > x_{\max}$ then set $x_{\max} := x_i$

    ➡ $x_{\max}$ is the biggest number in the input
  - ➡ Input size: n numbers
  - ➡ Runtime:

- ▷ Count the number of "basic" steps with respect to the "size" of the input
- ▷ basic step : something that takes a fixed constant time
- ▷ input size : number of elements in the input
- ▷ Example: Determine the biggest number in a given list
  - Input: List $x_1, \ldots, x_n$ of numbers
  - Algorithm: Set the current biggest number $x_{\max} := -\infty$ ← constant time

    For all i from 1 to n...

    ...if $x_i > x_{\max}$ then set $x_{\max} := x_i$ ← constant time

    ➡ $x_{\max}$ is the biggest number in the input
  - ➡ Input size: n numbers
  - ➡ Runtime:

- ▷ Count the number of "basic" steps with respect to the "size" of the input
- ▷ basic step : something that takes a fixed constant time
- ▷ input size : number of elements in the input
- ▷ Example: Determine the biggest number in a given list
  - Input: List $x_1, \ldots, x_n$ of numbers
  - Algorithm: Set the current biggest number $x_{\max} := -\infty$ — constant time

    For all i from 1 to n... — repeated n times

    ...if $x_i > x_{\max}$ then set $x_{\max} := x_i$ — constant time

    ➡ $x_{\max}$ is the biggest number in the input
  - ➡ Input size: n numbers
  - ➡ Runtime:

- ▷ Count the number of "basic" steps with respect to the "size" of the input
- ▷ basic step : something that takes a fixed constant time
- ▷ input size : number of elements in the input
- ▷ Example: Determine the biggest number in a given list
  - Input: List $x_1, \ldots, x_n$ of numbers
  - Algorithm: Set the current biggest number $x_{\max} := -\infty$ (constant time)

    For all i from 1 to n... (repeated n times)

    ...if $x_i > x_{\max}$ then set $x_{\max} := x_i$ (constant time)

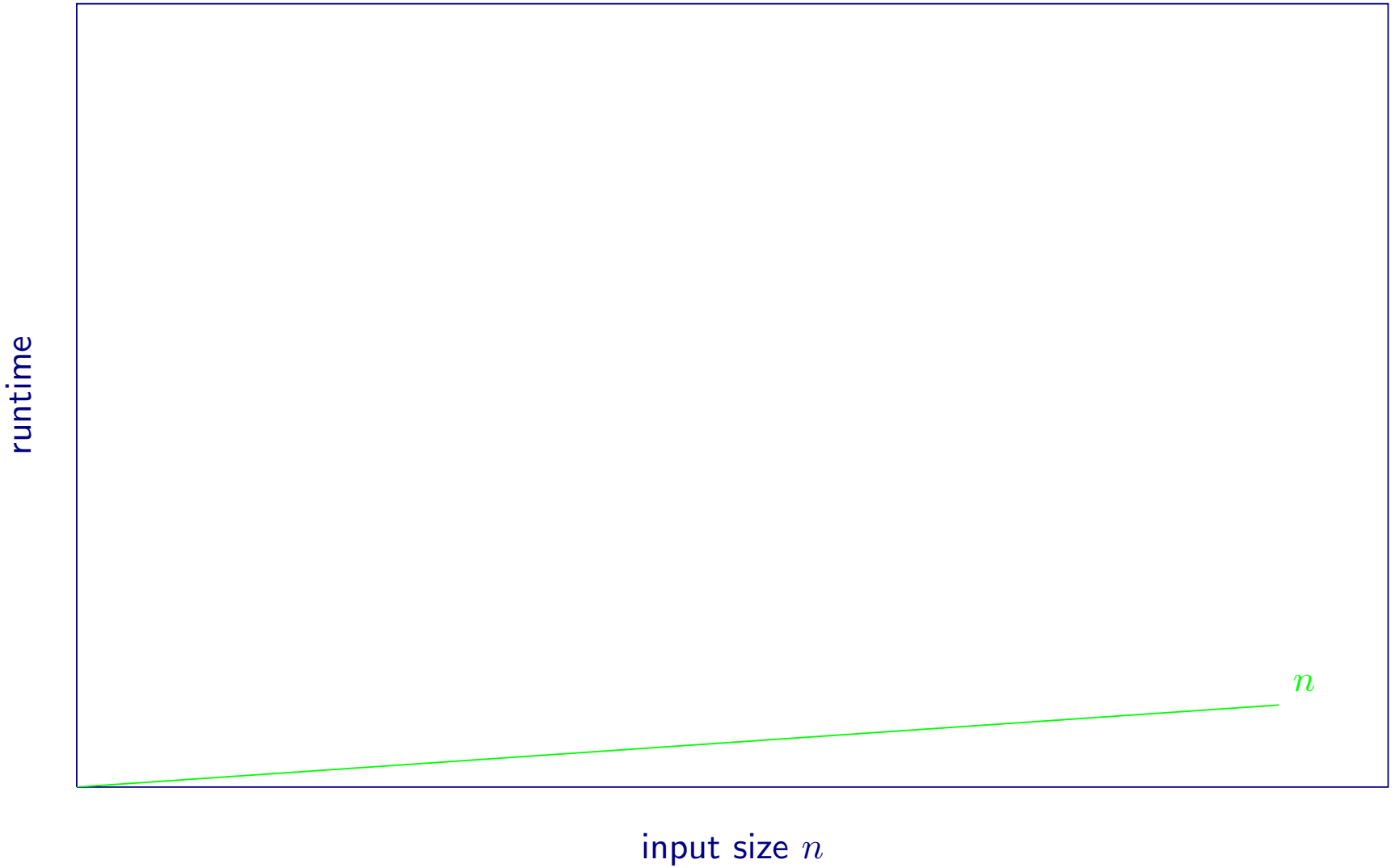
    ➡ $x_{\max}$ is the biggest number in the input
  - ➡ Input size: n numbers
  - ➡ Runtime: $n+1$ "basic steps"

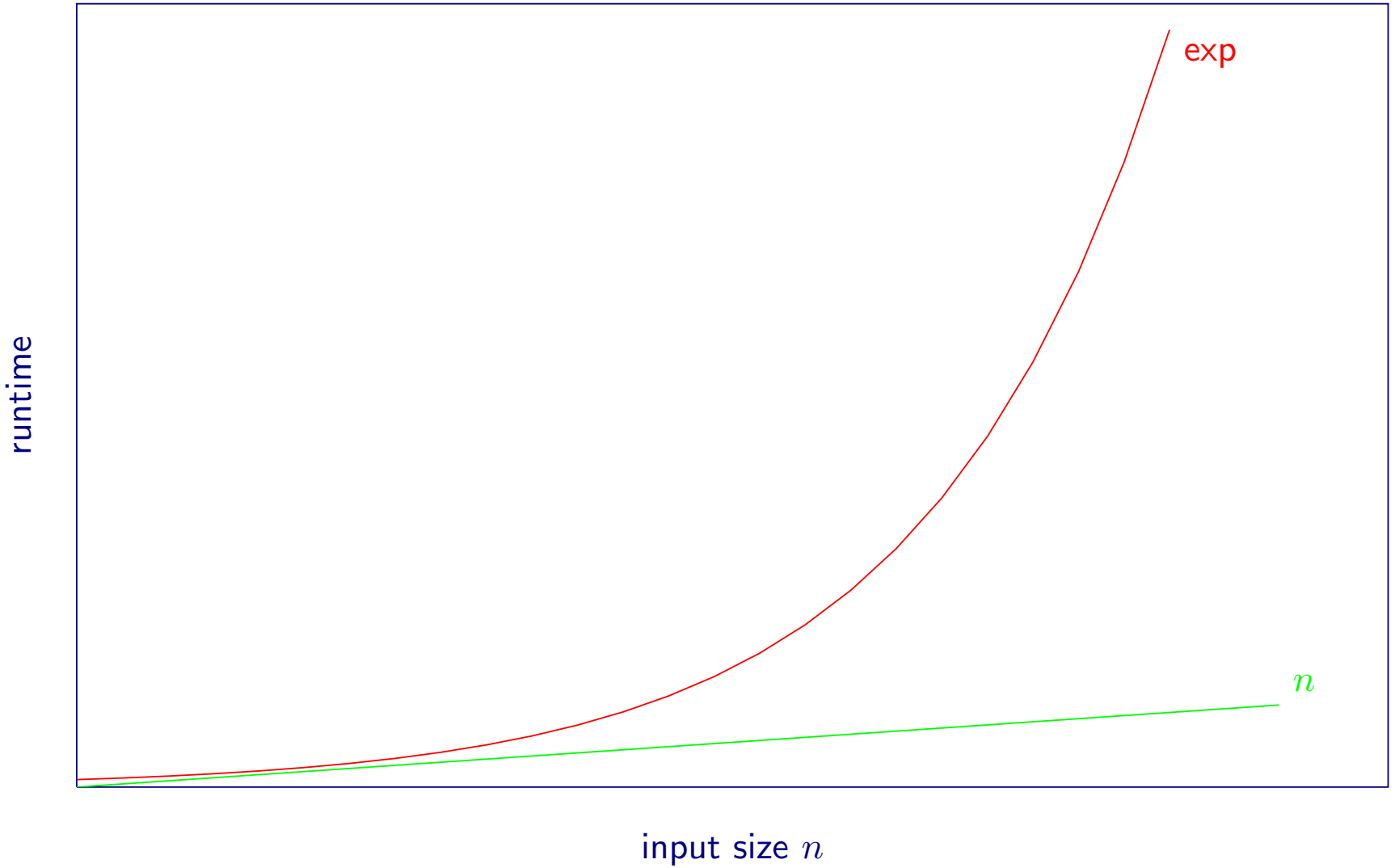
- ▷ Count the number of "basic" steps with respect to the "size" of the input
- ▷ basic step : something that takes a fixed constant time
- ▷ input size : number of elements in the input
- ▷ Example: Determine the biggest number in a given list
  - Input: List $x_1, \ldots, x_n$ of numbers
  - Algorithm: Set the current biggest number $x_{\max} := -\infty$ (constant time)

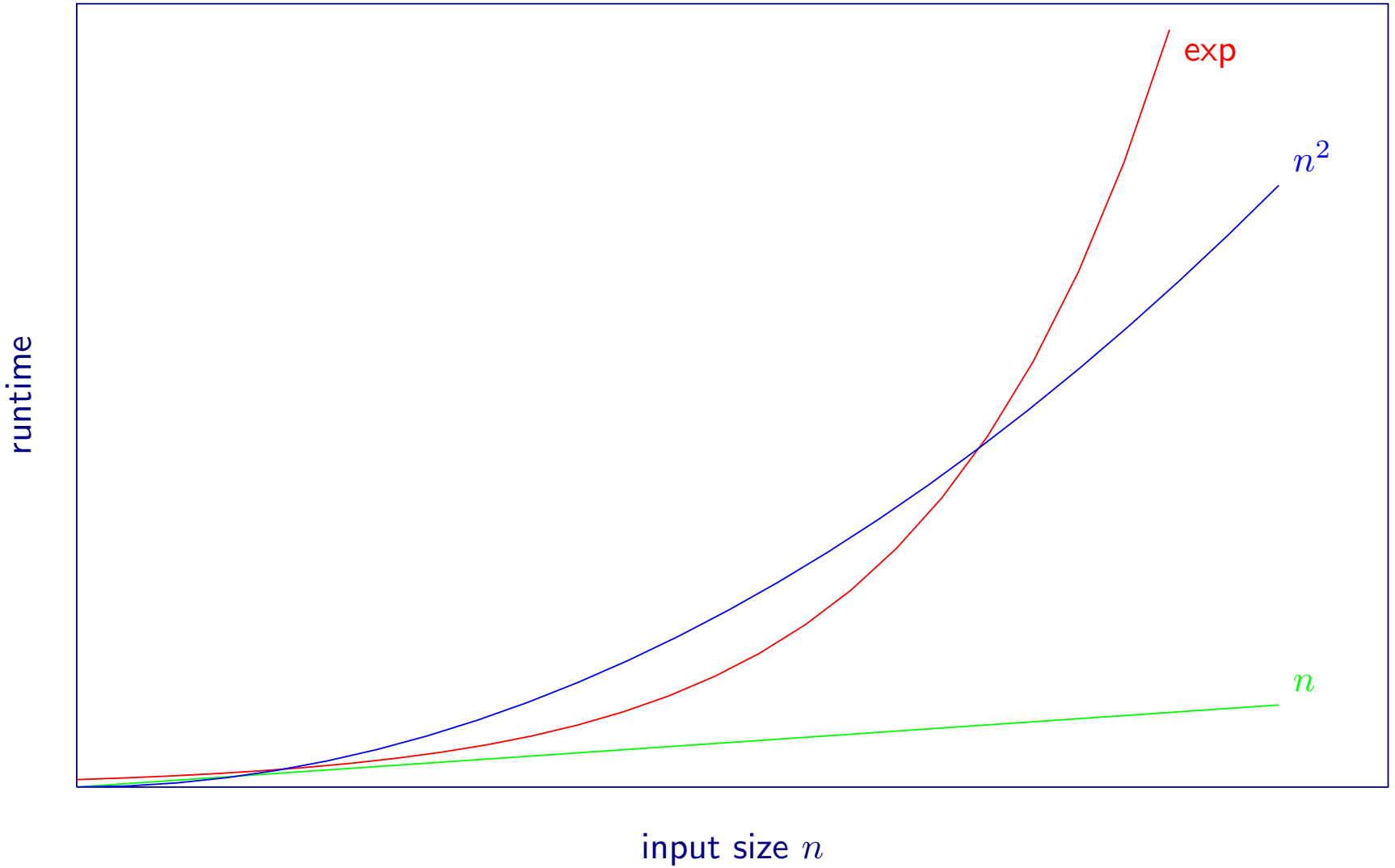
    For all i from 1 to n... (repeated n times)

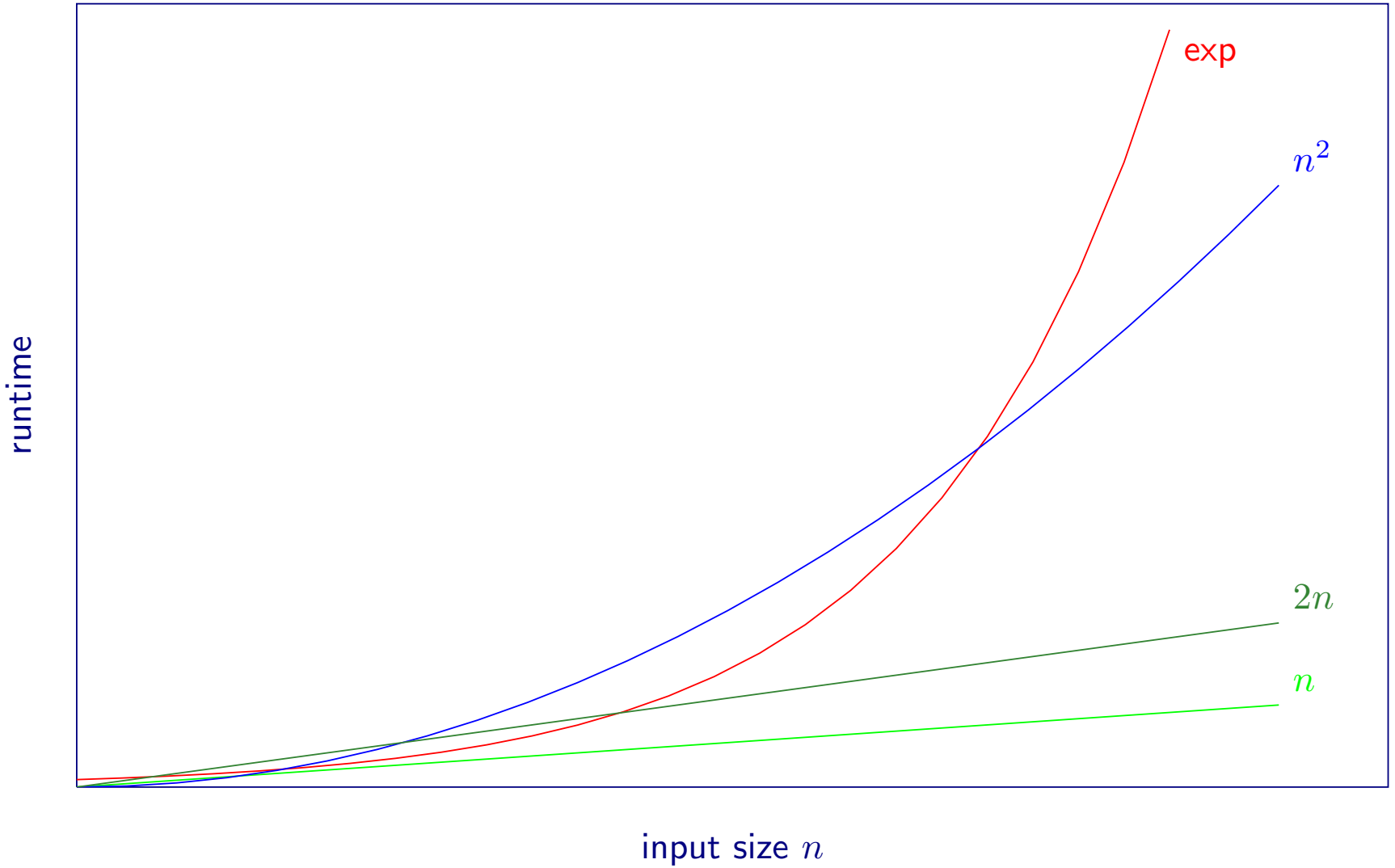
    ...if $x_i > x_{\max}$ then set $x_{\max} := x_i$ (constant time)

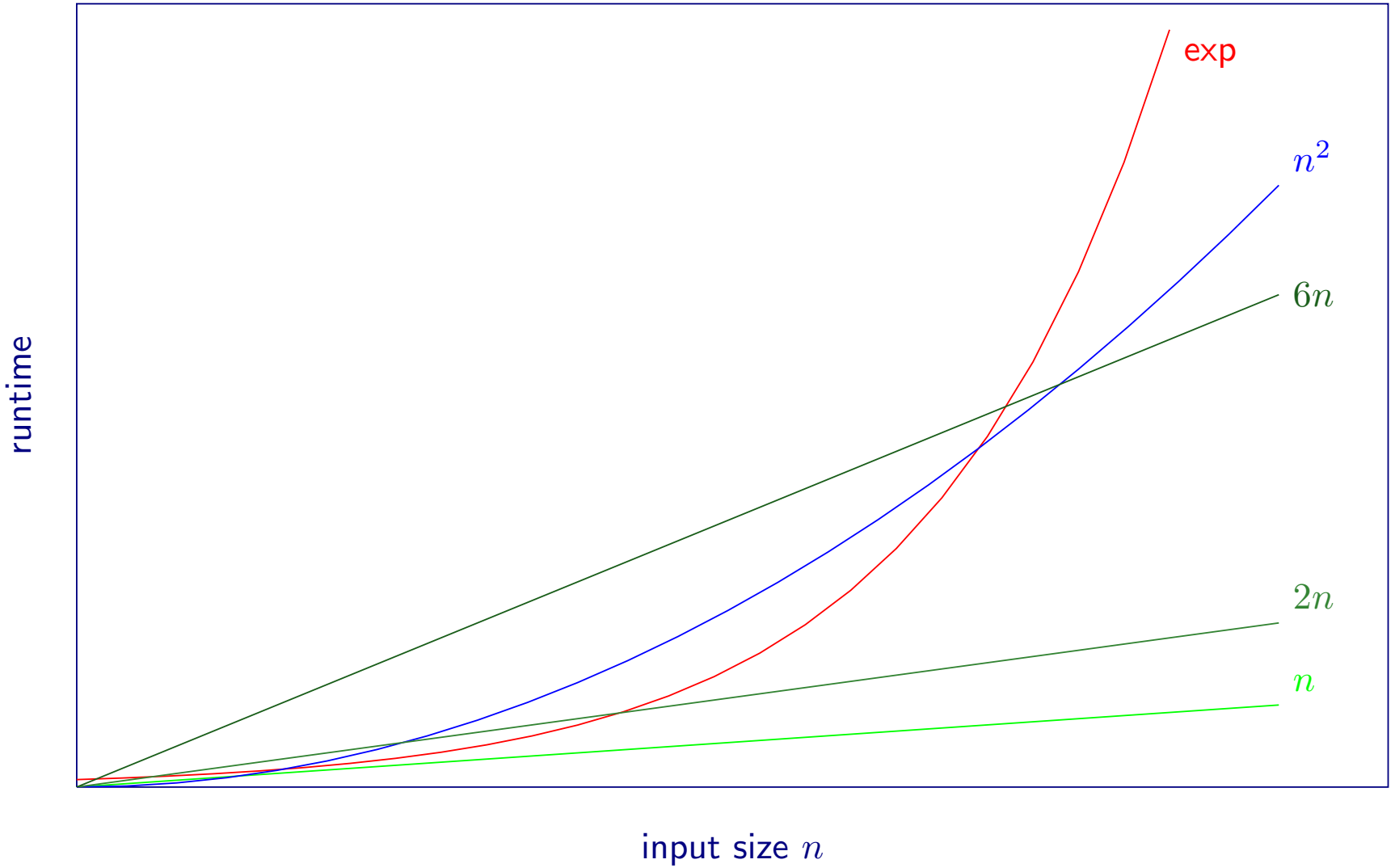
    ➡ $x_{\max}$ is the biggest number in the input
  - ➡ Input size: n numbers
  - ➡ Runtime: $n+1$ "basic steps" (order of n, linear runtime)

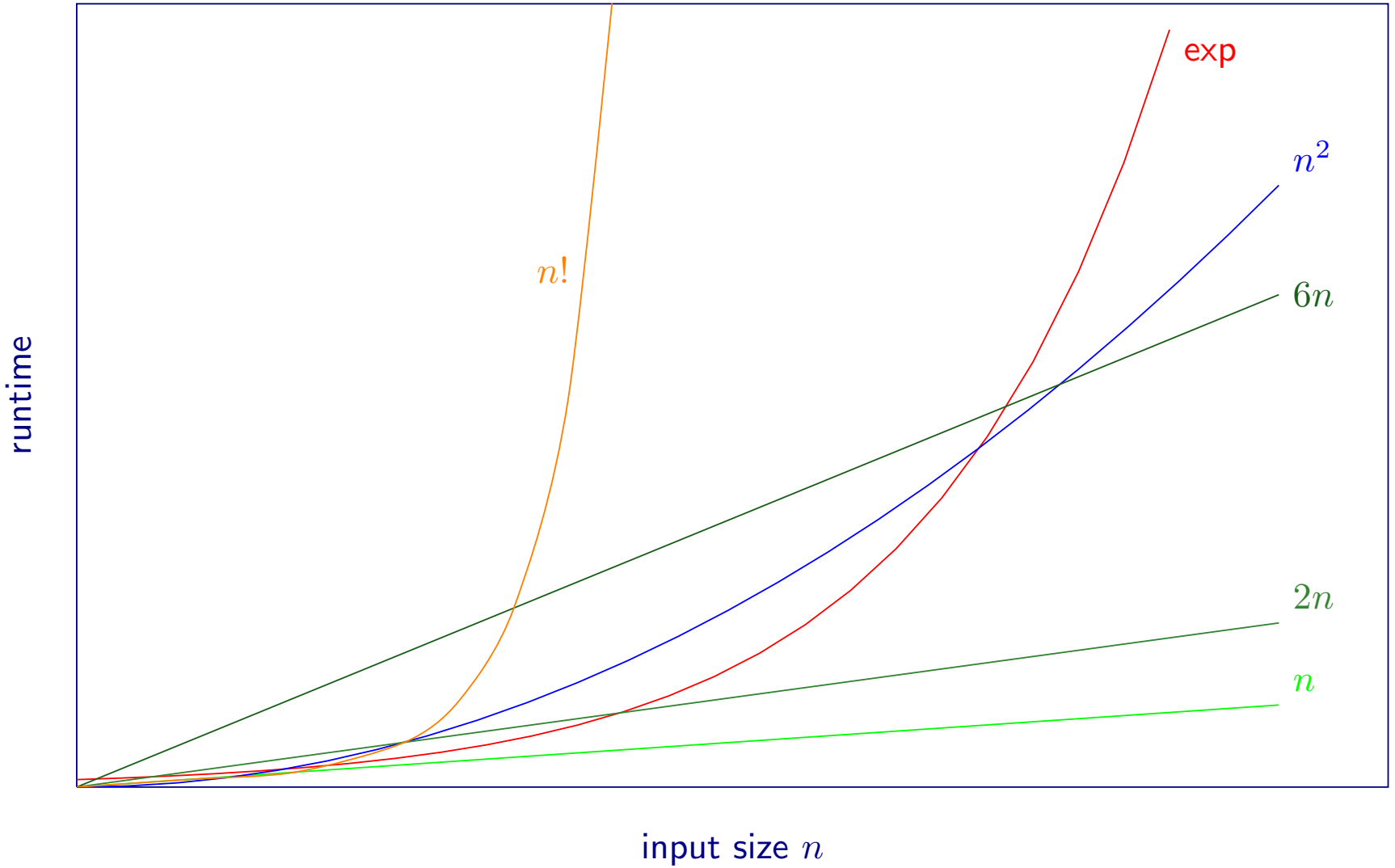

runtime
n
input size n

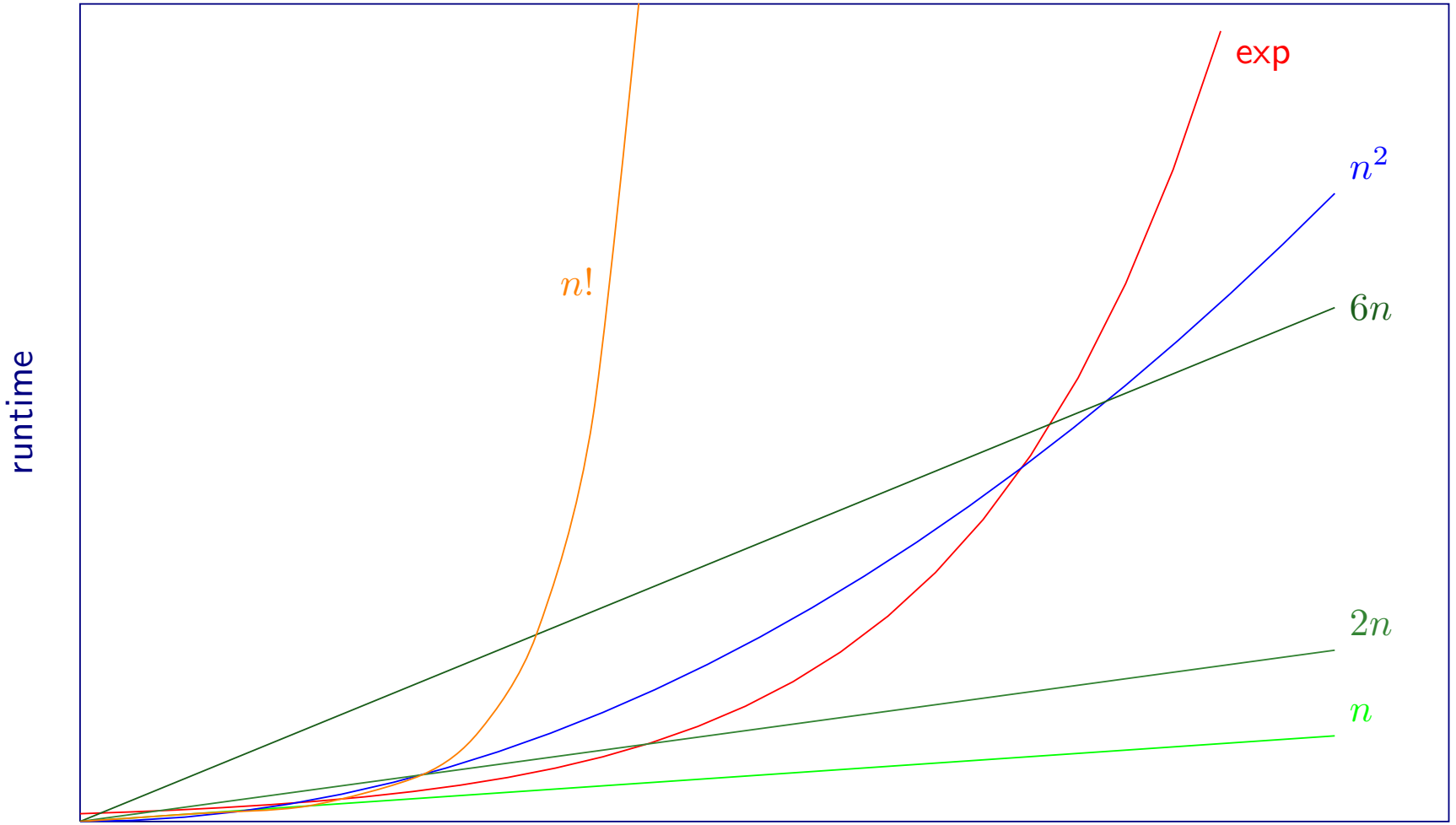

exp
n
runtime
input size n


exp
n^2
n
runtime
input size n


exp
n^2
$2n$
n
runtime
input size n


exp
n^2
$6n$
$2n$
n
runtime
input size n


exp
n^2
$n!$
$6n$
$2n$
n
runtime
input size n



linear — polynomial — exponential

▷ Input: directed graph with n nodes V

▷ Algorithm:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

set $\mathrm{pred}(w) := v$

...remove v from R

▷ Input: directed graph with n nodes V ➡ Input size: n

▷ Algorithm:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

set $\mathrm{pred}(w) := v$

...remove v from R

▷ Input: directed graph with n nodes V ➡ Input size: n

▷ Algorithm:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

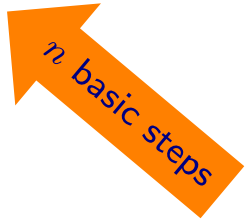
...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

set $\mathrm{pred}(w) := v$

...remove v from R



▷ Input: directed graph with n nodes V ➡ Input size: n

▷ Algorithm:

Set remaining nodes $R := V$, and $d(\text{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \text{A}$, $\text{pred}(\text{A}) := \text{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

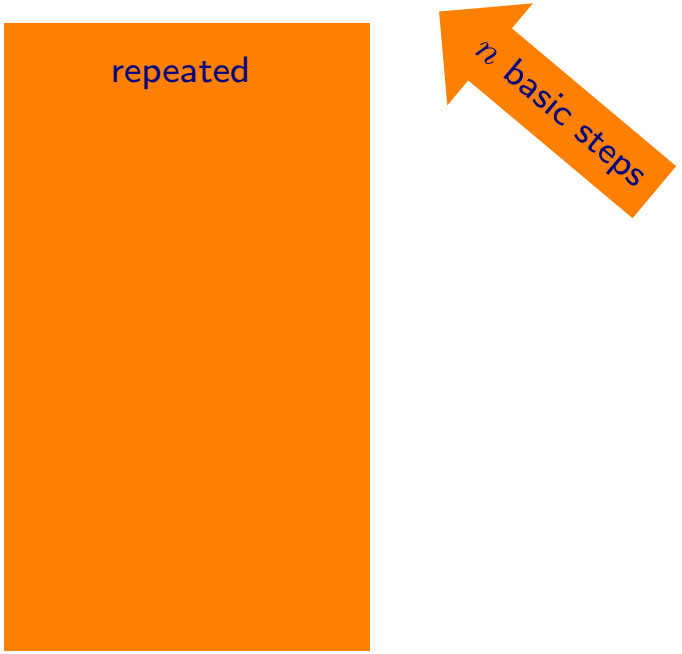
...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

set $\text{pred}(w) := v$

...remove v from R



▷ Input: directed graph with n nodes V ➡ Input size: n

▷ Algorithm:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

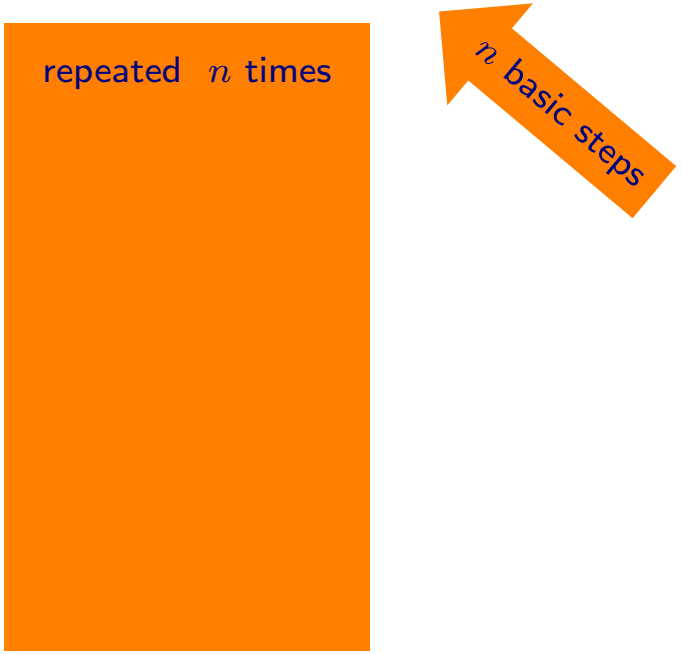
...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

set $\mathrm{pred}(w) := v$

...remove v from R



▷ Input: directed graph with n nodes V ➡ Input size: n

▷ Algorithm:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

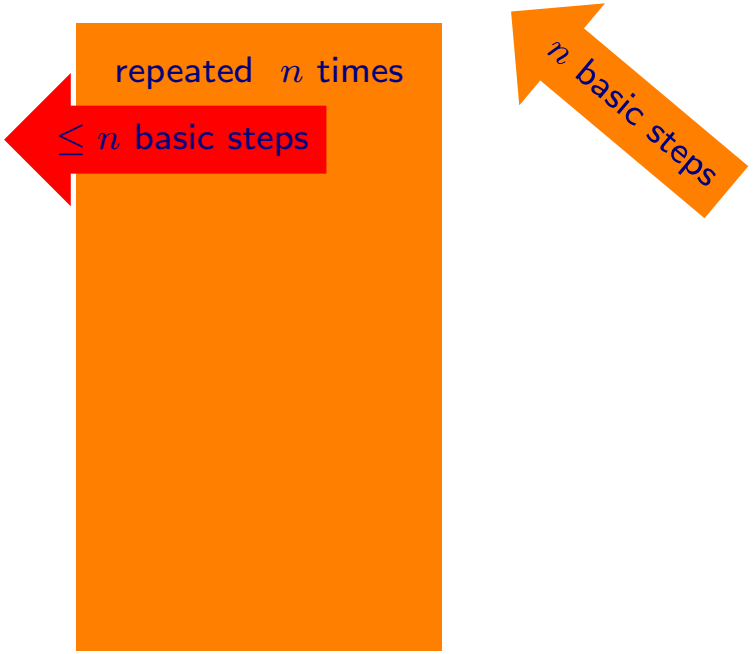
...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

set $\mathrm{pred}(w) := v$

...remove v from R



▷ Input: directed graph with n nodes V ➡ Input size: n

▷ Algorithm:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

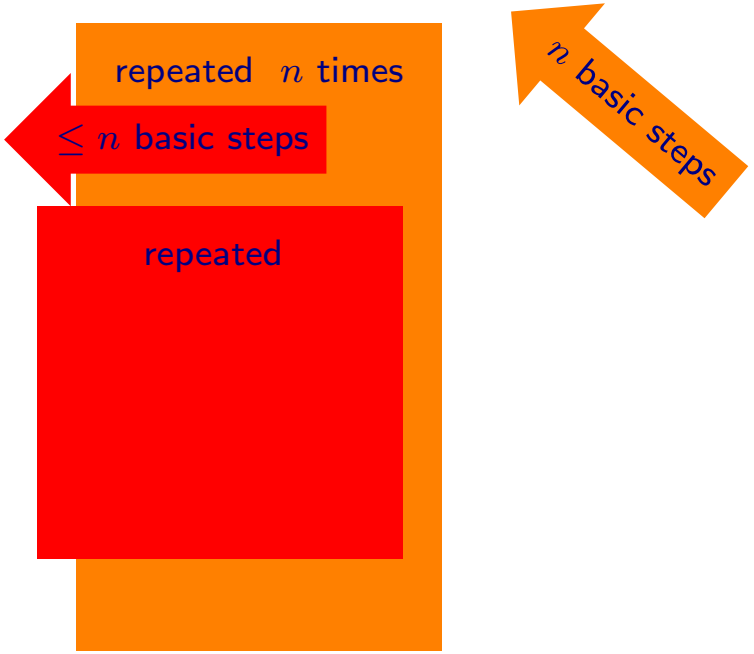
...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

set $\mathrm{pred}(w) := v$

...remove v from R



▷ Input: directed graph with n nodes V ➡ Input size: n

▷ Algorithm:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

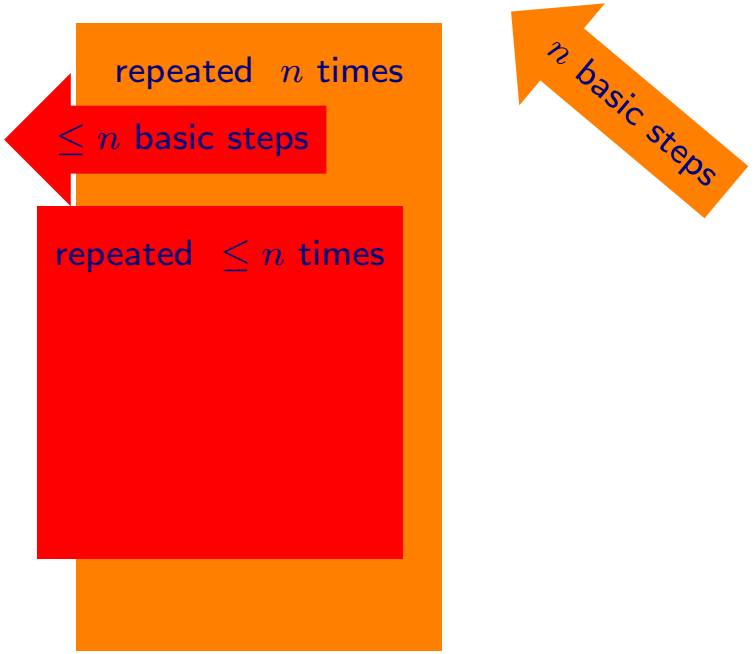
...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

set $\mathrm{pred}(w) := v$

...remove v from R



▷ Input: directed graph with n nodes V ➡ Input size: n

▷ Algorithm:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

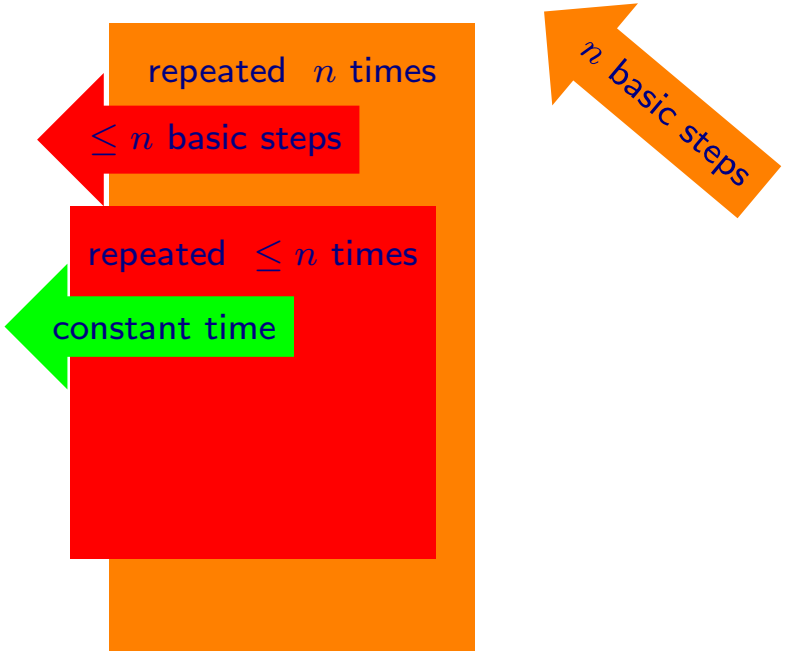
...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

set $\mathrm{pred}(w) := v$

...remove v from R



▷ Input: directed graph with n nodes V ➡ Input size: n

▷ Algorithm:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

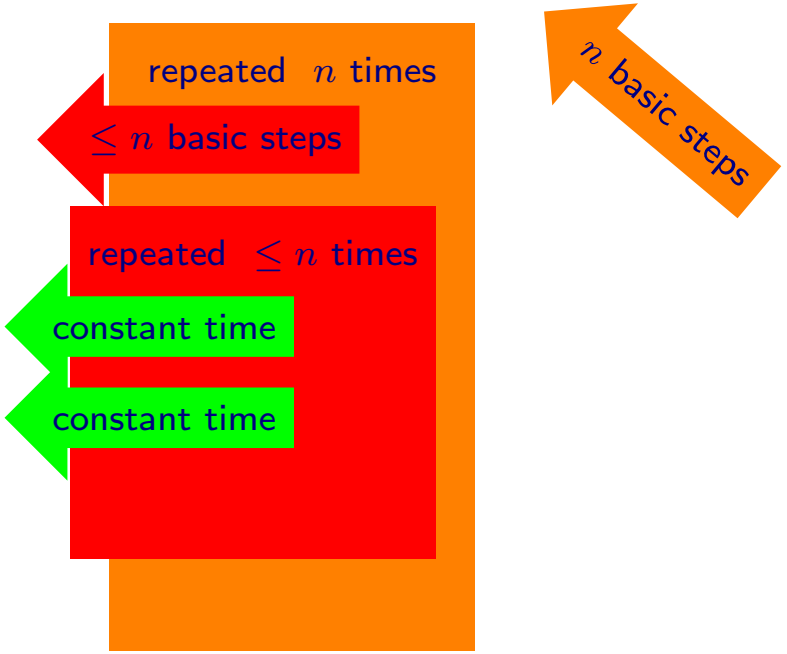
...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

set $\mathrm{pred}(w) := v$

...remove v from R



▷ Input: directed graph with n nodes V ➡ Input size: n

▷ Algorithm:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

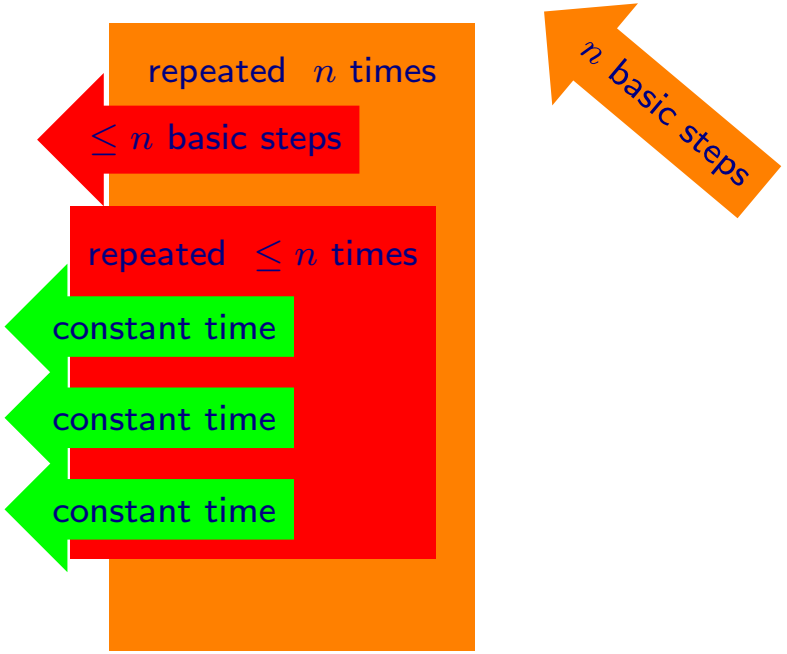
...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

set $\mathrm{pred}(w) := v$

...remove v from R



▷ Input: directed graph with n nodes V ➡ Input size: n

▷ Algorithm:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

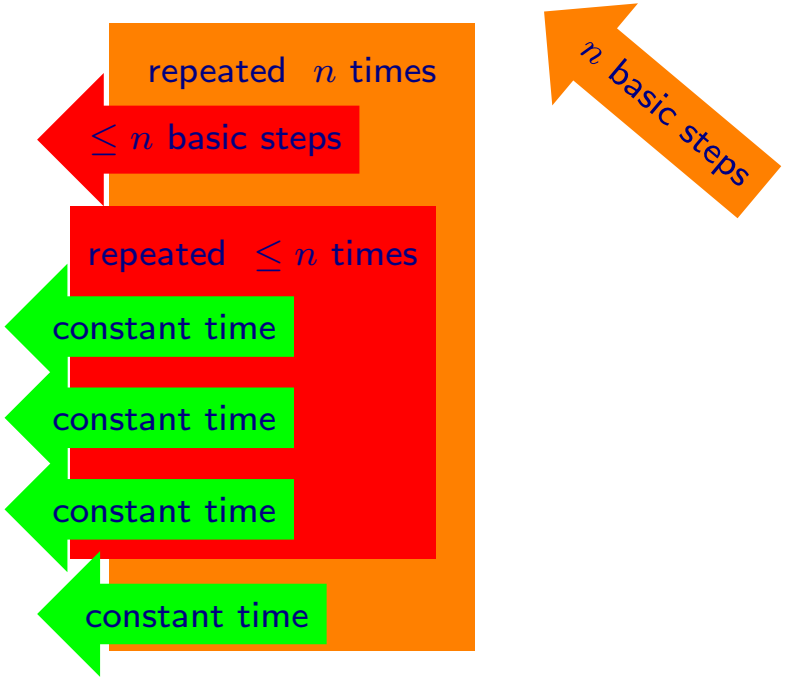
...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

set $\mathrm{pred}(w) := v$

...remove v from R



▷ Input: directed graph with n nodes V ➡ Input size: n

▷ Algorithm:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

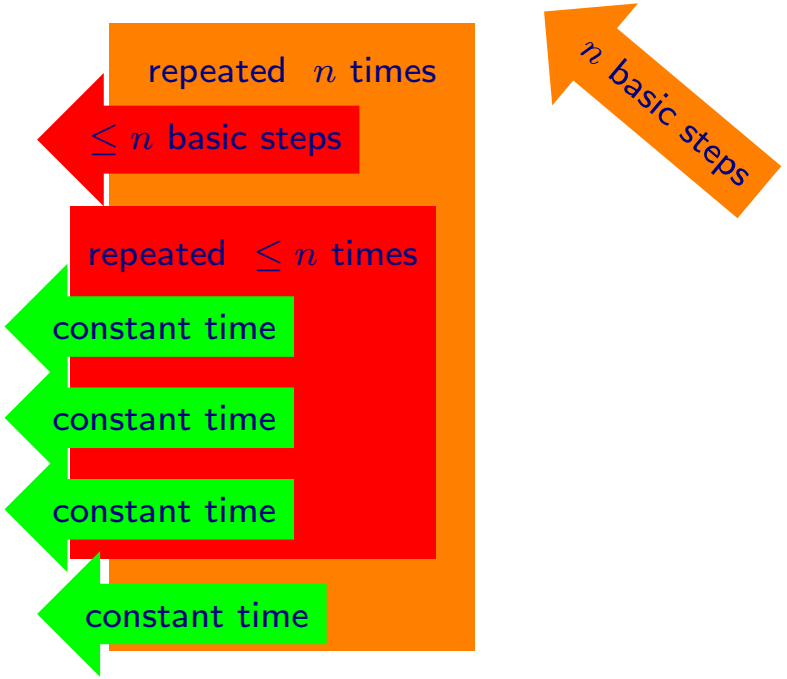
...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

set $\mathrm{pred}(w) := v$

...remove v from R



➡ Altogether: at most $n \cdot (n + 3n + 1) = 4n^2 + n$ "basic steps"

▷ Input: directed graph with n nodes V ➡ Input size: n

▷ Algorithm:

Set remaining nodes $R := V$, and $d(\mathrm{A}) := 0$, $d(v) := \infty$ for all nodes $v \neq \mathrm{A}$, $\mathrm{pred}(\mathrm{A}) := \mathrm{A}$

While R is not empty...

...let $v \in R$ with minimal $d(v)$

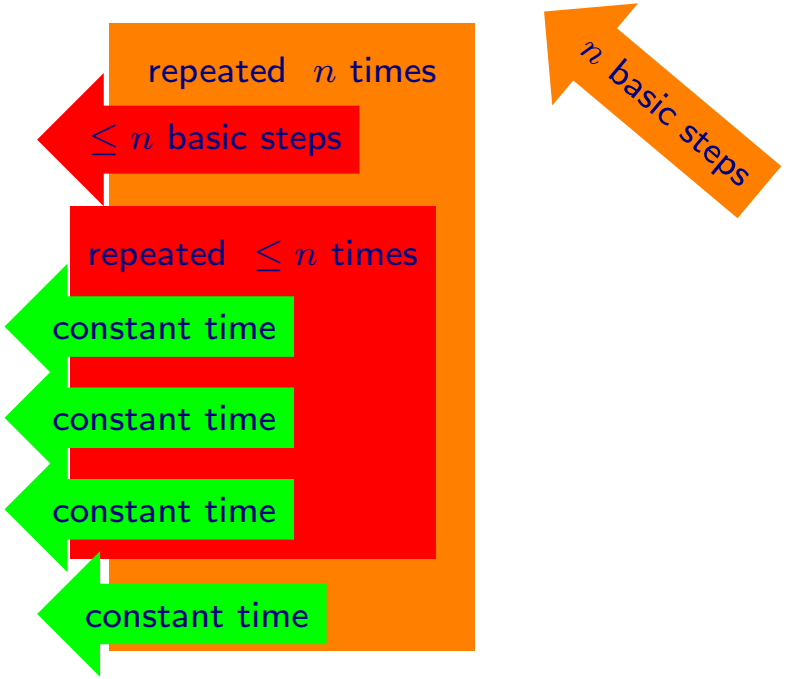
...for all arcs (v, w):

if $d(w) > d(v) + \ell(v, w)$ then:

set $d(w) := d(v) + \ell(v, w)$

set $\mathrm{pred}(w) := v$

...remove v from R



➡ Altogether: at most $n \cdot (n + 3n + 1) = 4n^2 + n$ "basic steps"

➡ Polynomial runtime ($\mathcal{O}(n^2)$)

- ▷ Models, Data and Algorithms
- ▷ Linear Optimization
- ▷ Mathematical Background: Polyhedra, Simplex-Algorithm
- ▷ Sensitivity Analysis; (Mixed) Integer Programming
- ▷ MIP Modelling
- ▷ MIP Modelling: More Examples; Branch & Bound
- ▷ Cutting Planes; Combinatorial Optimization: Examples, Graphs, Algorithms
- ▷ TSP-Heuristics
- ▷ Network Flows
- ▷ Shortest Path Problem
- ▷ Complexity Theory
- ▷ Nonlinear Optimization, Scheduling
- ▷ Lot Sizing, Multicriteria Optimization
- ▷ Oral exam