Verified C Implementation of Dijksta's Algorithm

Anshuman Mohan, Aquinas Hobor

APLAS Student Research Competition
December 3, 2019

Saluting the Mothership

Certifying Graph-Manipulating C Programs via Localizations within Data Structures

SHENGYI WANG, National University of Singapore, Singapore
QINXIANG CAO, Shanghai Jiao Tong University, China
ANSHUMAN MOHAN, National University of Singapore, Singapore
AQUINAS HOBOR, National University of Singapore, Singapore

VST + CompCert + 25000 LOC library

Powerful enough to verify executable code
against realistic specifications
expressed with mathematical graphs

[Wang *et. al.*, PACMPL OOPSLA 2019]

This Work

Certifying Graph-Manipulating C Programs via Localizations within Data Structures

SHENGYI WANG, National University of Singapore, Singapore
QINXIANG CAO, Shanghai Jiao Tong University, China
ANSHUMAN MOHAN, National University of Singapore, Singapore
AQUINAS HOBOR, National University of Singapore, Singapore

Never used edge labels...

This Work

Certifying Graph-Manipulating C Programs via Localizations within Data Structures

SHENGYI WANG, National University of Singapore, Singapore
QINXIANG CAO, Shanghai Jiao Tong University, China
ANSHUMAN MOHAN, National University of Singapore, Singapore
AQUINAS HOBOR, National University of Singapore, Singapore

Never used edge labels...
Not unlike vertex labels, but let's try anyway

This Work

Certifying Graph-Manipulating C Programs via Localizations within Data Structures

SHENGYI WANG, National University of Singapore, Singapore
QINXIANG CAO, Shanghai Jiao Tong University, China
ANSHUMAN MOHAN, National University of Singapore, Singapore
AQUINAS HOBOR, National University of Singapore, Singapore

Never used edge labels...
Not unlike vertex labels, but let's try anyway

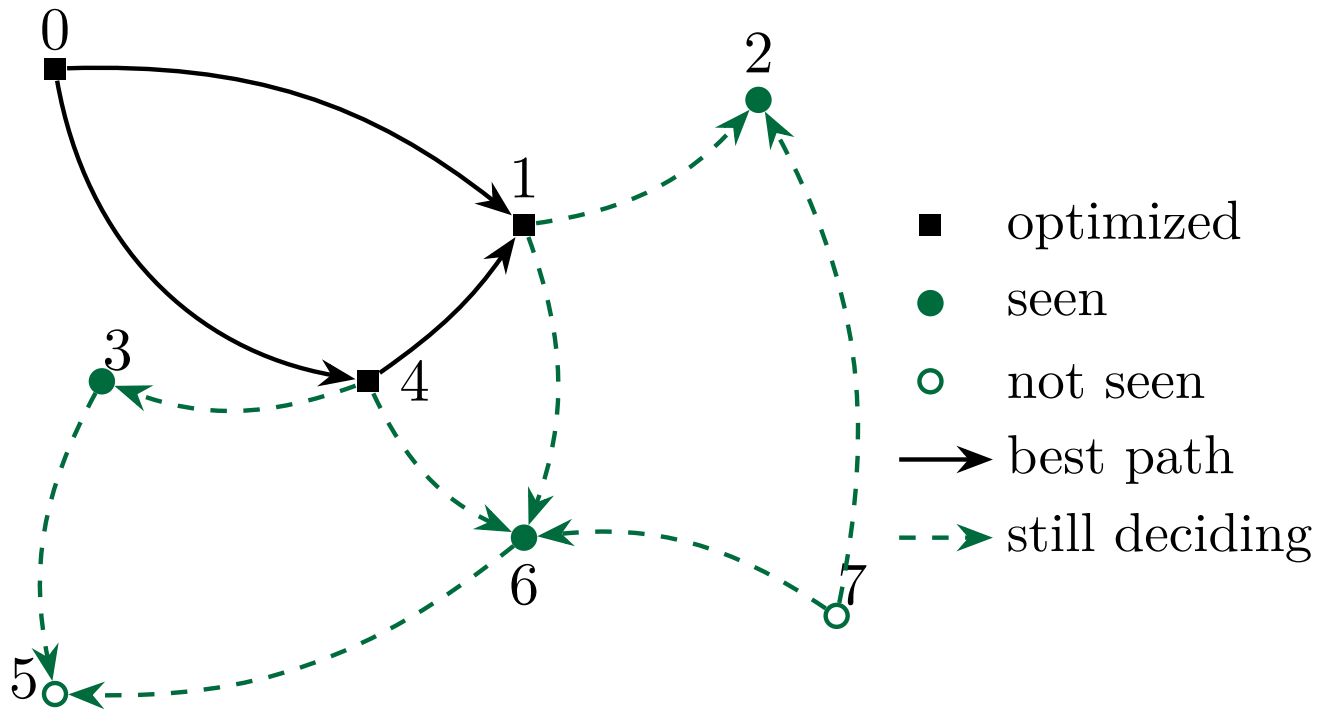
Verify Dijkstra's one-to-all shortest path algorithm

Challenges

Using CompCert C
- executable and realistic
- real-world complications

Aiming for full functional correctness
- and not just safety

Refresher



Instantiating DijkGraph

A PreGraph is a hextuple (`VType`, `EType`, `vvalid`, `evalid`, `src`, `dst`)

$$\textbf{Dijk\_PG}(\gamma) \stackrel{\text{def}}{=} \texttt{VType := Z}$$
$$\texttt{EType := VType * VType}$$
$$\texttt{src := fst}$$
$$\texttt{dst := snd}$$
$$\forall v.\ \texttt{vvalid}(\gamma, v) \Leftrightarrow 0 \leqslant v < \texttt{SIZE}$$
$$\forall s, d.\ \texttt{evalid}(\gamma, (s, d)) \Leftrightarrow \texttt{vvalid}(\gamma, s) \wedge \texttt{vvalid}(\gamma, d)$$

Instantiating DijkGraph

A LabeledGraph is a quadruple (PreGraph, `VL`, `EL`, `GL`)

$$\textbf{Dijk\_LG}(\gamma) \stackrel{\text{def}}{=} \text{Dijk\_PG as shown}$$

```
VL := list EL
EL := Z
GL := unit
```

Instantiating DijkGraph

A GeneralGraph adds arbitrary soundness conditions

$$\textbf{DijkGraph}(\gamma) \stackrel{\text{def}}{=} \text{Dijk\_LG as shown, and}$$
$$\mathit{FiniteGraph}(\gamma) \wedge$$
$$\forall i, j.\ \texttt{vvalid}(\gamma, i) \wedge \texttt{vvalid}(\gamma, j) \Rightarrow$$
$$i = j \Rightarrow \texttt{elabel}(\gamma, (i, j)) = 0 \wedge$$
$$i \neq j \Rightarrow 0 \leqslant \texttt{elabel}(\gamma, (i, j)) \leqslant \lfloor \texttt{MAX/SIZE} \rfloor$$

Instantiating DijkGraph

A GeneralGraph adds arbitrary soundness conditions

$$\textbf{DijkGraph}(\gamma) \stackrel{\text{def}}{=} \text{Dijk\_LG as shown, and}$$
$$\mathit{FiniteGraph}(\gamma) \wedge$$
$$\forall i, j.\ \texttt{vvalid}(\gamma, i) \wedge \texttt{vvalid}(\gamma, j) \Rightarrow$$
$$i = j \Rightarrow \texttt{elabel}(\gamma, (i, j)) = 0 \wedge$$
$$i \neq j \Rightarrow 0 \leqslant \texttt{elabel}(\gamma, (i, j)) \leqslant \lfloor \texttt{MAX/SIZE} \rfloor$$

Representing DijkGraph in Memory

$$\mathsf{list\_rep}(\gamma, i) \stackrel{\text{def}}{=} \texttt{data\_at array graph2mat}(\gamma)[i]\ \texttt{list\_addr}(\gamma, i)$$

$$\mathsf{graph\_rep}(\gamma) \stackrel{\text{def}}{=} \mathop{\Large *}_{\texttt{vvalid}(\gamma, v)} v \mapsto \mathsf{list\_rep}(\gamma, v)$$

Code and Specification

```
#define IFTY INT_MAX - INT_MAX/SIZE

void dijkstra (int graph[SIZE][SIZE], int src,
                          int *dist, int *prev) {
```

$\{$ DijkGraph$(\gamma)\}$

Code and Specification

```
#define IFTY INT_MAX - INT_MAX/SIZE

void dijkstra (int graph[SIZE][SIZE], int src,
                              int *dist, int *prev) {
```

$\{$ DijkGraph(γ) $\}$

```
 int pq[SIZE];
 int i, j, u, cost;
 for (i = 0; i < SIZE; i++)
 {  dist[i] = INF; prev[i] = INF; pq[i] = INF;  }
 dist[src] = 0; pq[src] = 0; prev[src] = src;
```

Code and Specification

```
#define IFTY INT_MAX - INT_MAX/SIZE

void dijkstra (int graph[SIZE][SIZE], int src,
                              int *dist, int *prev) {
```

$\{$ DijkGraph$(\gamma)\}$

```
 int pq[SIZE];
 int i, j, u, cost;
 for (i = 0; i < SIZE; i++)
 {  dist[i] = INF; prev[i] = INF; pq[i] = INF;  }
 dist[src] = 0; pq[src] = 0; prev[src] = src;
```

$\{$ DijkGraph$(\gamma) \wedge \mathit{dijk_correct}(\gamma, \mathit{src}, \mathit{prev}, \mathit{dist}, \mathit{priq})\}$

```
 // big while loop
```

Code and Specification

$$\{\ \mathsf{DijkGraph}(\gamma) \wedge \mathit{dijk\_correct}(\gamma, \mathit{src}, \mathit{prev}, \mathit{dist}, \mathit{priq})\}$$

Code and Specification

$\{\ \mathsf{DijkGraph}(\gamma) \wedge \mathit{dijk\_correct}(\gamma, \mathit{src}, \mathit{prev}, \mathit{dist}, \mathit{priq})\}$

```
while (!pq_emp(pq)) {
 u = popMin(pq);
 for (i = 0; i < SIZE; i++) {
  cost = graph[u][i];
   if (cost < INF) {
    if (dist[i] > dist[u] + cost) {
     dist[i] = dist[u] + cost; prev[i] = u; pq[i] = dist[i];
}}}}
```

Code and Specification

$\{\ \mathsf{DijkGraph}(\gamma) \wedge \mathit{dijk\_correct}(\gamma, \mathit{src}, \mathit{prev}, \mathit{dist}, \mathit{priq})\}$

```
 while (!pq_emp(pq)) {
  u = popMin(pq);
  for (i = 0; i < SIZE; i++) {
   cost = graph[u][i];
    if (cost < INF) {
     if (dist[i] > dist[u] + cost) {
      dist[i] = dist[u] + cost; prev[i] = u; pq[i] = dist[i];
 }}}}
```

$\left\{\begin{array}{l} \mathsf{DijkGraph}(\gamma) \wedge \forall\, \mathit{dst} \in \mathit{priq}.\ \mathit{priq}[\mathit{dst}] = \texttt{INF} \wedge \\ \mathit{dijk\_correct}(\gamma, \mathit{src}, \mathit{prev}, \mathit{dist}, \mathit{priq}) \end{array}\right\}$

```
return;
}
```

Dijkstra_Correct

$$
\begin{aligned}
&\mathit{dijk\_correct}(\gamma, \mathit{src}, \mathit{prev}, \mathit{dist}, \mathit{priq}) \stackrel{\text{def}}{=} \\
&\quad \forall \mathit{dst}.\ \mathit{dst} \in \mathit{popped}(\mathit{priq}) \ \Rightarrow \\
&\qquad\qquad \exists \mathit{path}.\ \mathit{path\_correct}(\gamma, \mathit{prev}, \mathit{dist}, \mathit{path}) \wedge \\
&\qquad\qquad \mathit{path\_glob\_optimal}(\gamma, \mathit{dist}, \mathit{path}) \wedge \\
&\qquad\qquad \mathit{path\_entirely\_in\_popped}(\gamma, \mathit{prev}, \mathit{path}) \wedge \\
&\qquad \mathit{priq}[\mathit{dst}] < \texttt{INF} \ \Rightarrow \\
&\qquad\qquad \texttt{let}\ m\ \texttt{:=}\ \mathit{prev}[\mathit{dst}]\ \texttt{in}\ m \in \mathit{popped}(\mathit{priq}) \wedge \\
&\qquad\qquad \forall m' \in \mathit{popped}(\mathit{priq}).\ \mathit{cost}(\mathit{path2m}+ :: (m, \mathit{dst})) \leqslant \\
&\qquad\qquad\qquad\qquad \mathit{cost}(\mathit{path2m'}+ :: (m', \mathit{dst})) \wedge \\
&\qquad \mathit{priq}[\mathit{dst}] = \texttt{INF} \ \Rightarrow \\
&\qquad\qquad \forall m \in \mathit{popped}(\mathit{priq}).\ \mathit{cost}(\mathit{path2m}+ :: (m, \mathit{dst})) = \texttt{INF}
\end{aligned}
$$

Dijkstra_Correct

$$
\begin{aligned}
&dijk\_correct(\gamma, src, prev, dist, priq) \stackrel{\text{def}}{=} \\
&\quad \forall dst.\ dst \in popped(priq) \Rightarrow \\
&\qquad\qquad \exists path.\ path\_correct(\gamma, prev, dist, path) \wedge \\
&\qquad\qquad path\_glob\_optimal(\gamma, dist, path) \wedge \\
&\qquad\qquad path\_entirely\_in\_popped(\gamma, prev, path) \wedge \\
&\qquad priq[dst] < \texttt{INF} \Rightarrow \\
&\qquad\qquad \texttt{let}\ m\ \texttt{:=}\ prev[dst]\ \texttt{in}\ m \in popped(priq) \wedge \\
&\qquad\qquad \forall m' \in popped(priq).\ cost(path2m+ :: (m, dst)) \leqslant \\
&\qquad\qquad\qquad\qquad cost(path2m'+ :: (m', dst)) \wedge \\
&\qquad priq[dst] = \texttt{INF} \Rightarrow \\
&\qquad\qquad \forall m \in popped(priq).\ cost(path2m+ :: (m, dst)) = \texttt{INF}
\end{aligned}
$$

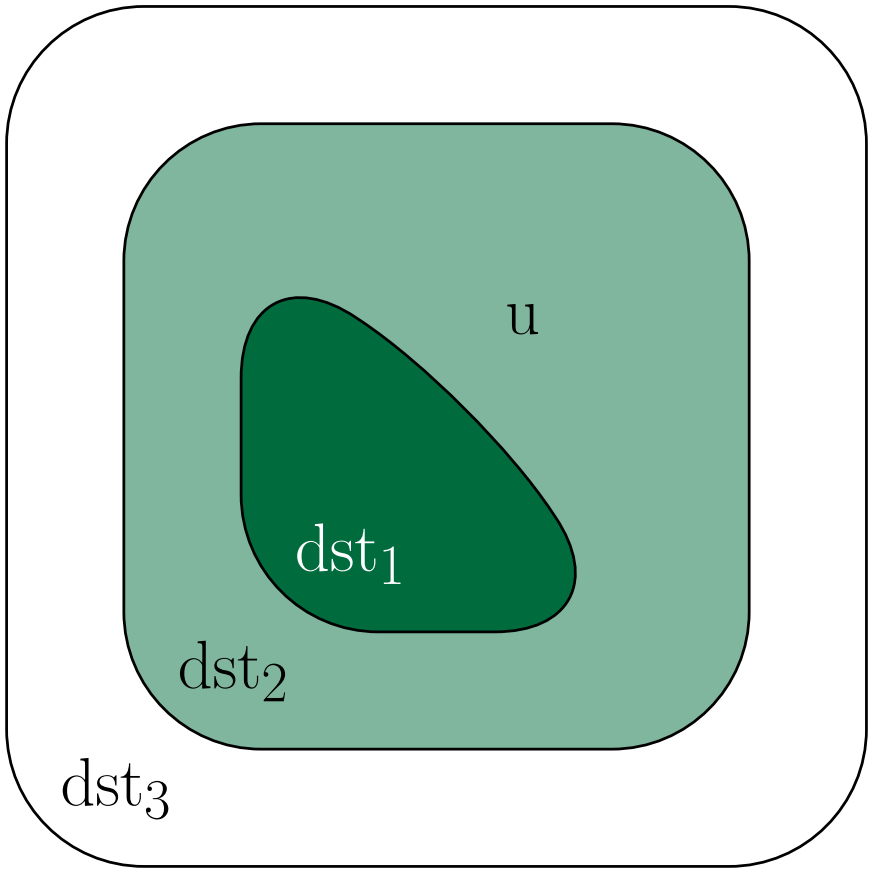
Dijkstra_Correct

$$
\begin{aligned}
&\mathit{dijk\_correct}(\gamma, \mathit{src}, \mathit{prev}, \mathit{dist}, \mathit{priq}) \stackrel{\text{def}}{=} \\
&\quad \forall \mathit{dst}.\ \mathit{dst} \in \mathit{popped}(\mathit{priq}) \Rightarrow \\
&\qquad\qquad \exists \mathit{path}.\ \mathit{path\_correct}(\gamma, \mathit{prev}, \mathit{dist}, \mathit{path}) \wedge \\
&\qquad\qquad \mathit{path\_glob\_optimal}(\gamma, \mathit{dist}, \mathit{path}) \wedge \\
&\qquad\qquad \mathit{path\_entirely\_in\_popped}(\gamma, \mathit{prev}, \mathit{path}) \wedge \\
&\qquad \mathit{priq}[\mathit{dst}] < \texttt{INF} \Rightarrow \\
&\qquad\qquad \texttt{let}\ m\ \texttt{:=}\ \mathit{prev}[\mathit{dst}]\ \texttt{in}\ m \in \mathit{popped}(\mathit{priq}) \wedge \\
&\qquad\qquad \forall m' \in \mathit{popped}(\mathit{priq}).\ \mathit{cost}(\mathit{path2m}+ :: (m, \mathit{dst})) \leqslant \\
&\qquad\qquad\qquad\qquad \mathit{cost}(\mathit{path2m'}+ :: (m', \mathit{dst})) \wedge \\
&\qquad \mathit{priq}[\mathit{dst}] = \texttt{INF} \Rightarrow \\
&\qquad\qquad \forall m \in \mathit{popped}(\mathit{priq}).\ \mathit{cost}(\mathit{path2m}+ :: (m, \mathit{dst})) = \texttt{INF}
\end{aligned}
$$

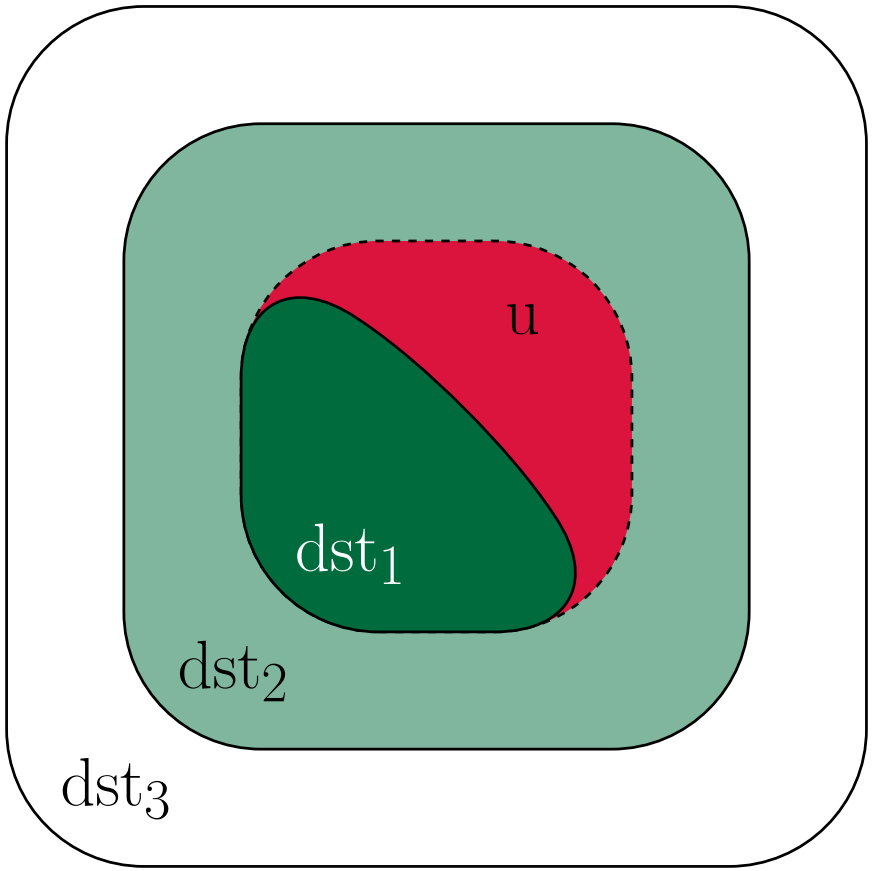
Dijkstra_Correct

$$
\begin{aligned}
&dijk\_correct(\gamma, src, prev, dist, priq) \stackrel{\text{def}}{=} \\
&\quad \forall dst.\ dst \in popped(priq) \Rightarrow \\
&\qquad\qquad \exists path.\ path\_correct(\gamma, prev, dist, path) \wedge \\
&\qquad\qquad path\_glob\_optimal(\gamma, dist, path) \wedge \\
&\qquad\qquad path\_entirely\_in\_popped(\gamma, prev, path) \wedge \\
&\qquad priq[dst] < \texttt{INF} \Rightarrow \\
&\qquad\qquad \texttt{let}\ m\ \texttt{:=}\ prev[dst]\ \texttt{in}\ m \in popped(priq) \wedge \\
&\qquad\qquad \forall m' \in popped(priq).\ cost(path2m+ :: (m, dst)) \leqslant \\
&\qquad\qquad\qquad\qquad cost(path2m'+ :: (m', dst)) \wedge \\
&\qquad priq[dst] = \texttt{INF} \Rightarrow \\
&\qquad\qquad \forall m \in popped(priq).\ cost(path2m+ :: (m, dst)) = \texttt{INF}
\end{aligned}
$$

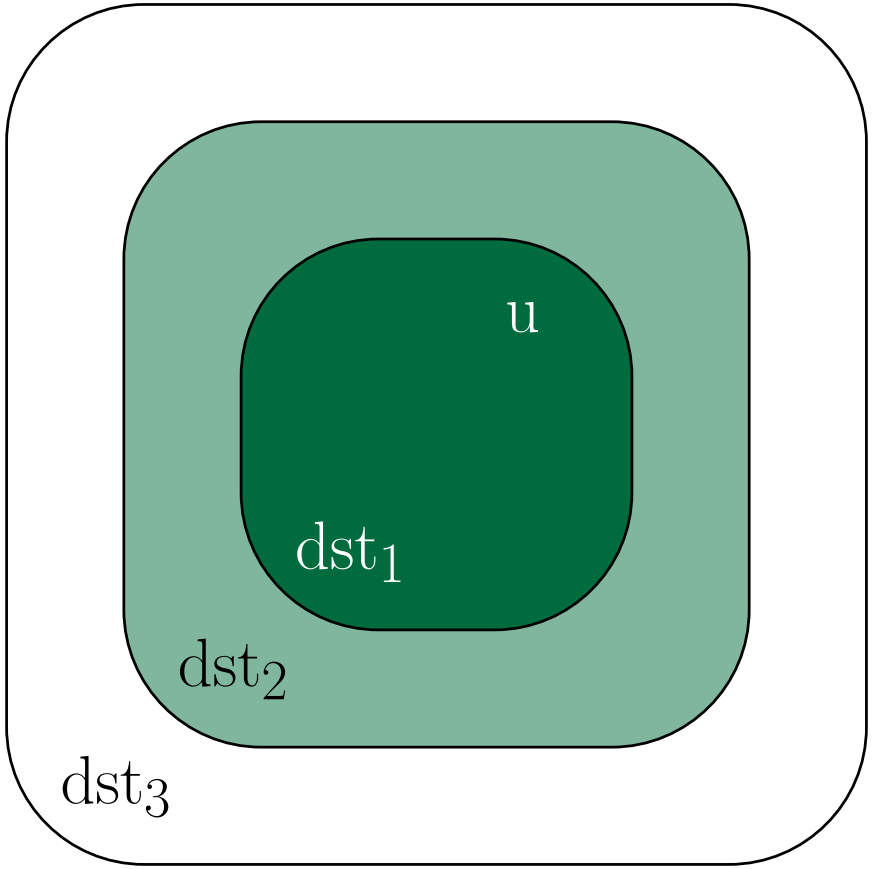
Key Transformation: Growing the Subgraph



Key Transformation: Growing the Subgraph



Key Transformation: Growing the Subgraph



Overflow Strikes Again

The longest optimal path has `SIZE-1` links

Overflow Strikes Again

The longest optimal path has `SIZE-1` links

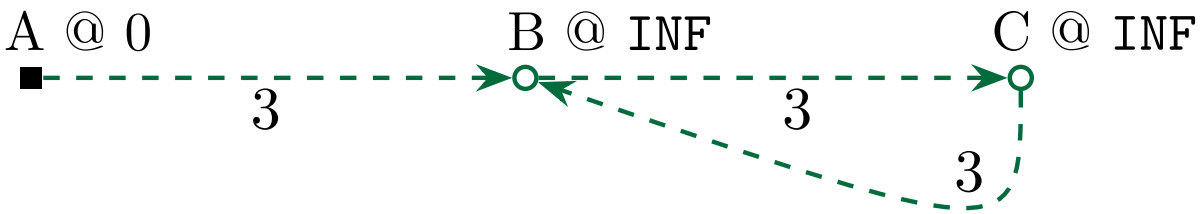
so say we set `elabel`'s upper bound to $\lfloor$`MAX/(SIZE-1)`$\rfloor$

Overflow Strikes Again

The longest optimal path has `SIZE-1` links

so say we set `elabel`'s upper bound to $\lfloor$`MAX/(SIZE-1)`$\rfloor$

`MAX` $= 7$, `SIZE` $= 3$, so $0 \leqslant$ `elabel`$(\gamma, e) \leqslant 3$.

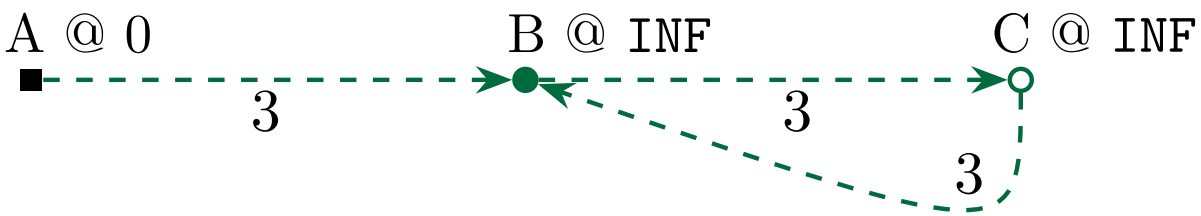


Overflow Strikes Again

The longest optimal path has `SIZE-1` links

so say we set `elabel`'s upper bound to $\lfloor$`MAX/(SIZE-1)`$\rfloor$

`MAX` $= 7$, `SIZE` $= 3$, so $0 \leqslant$ `elabel`$(\gamma, e) \leqslant 3$.

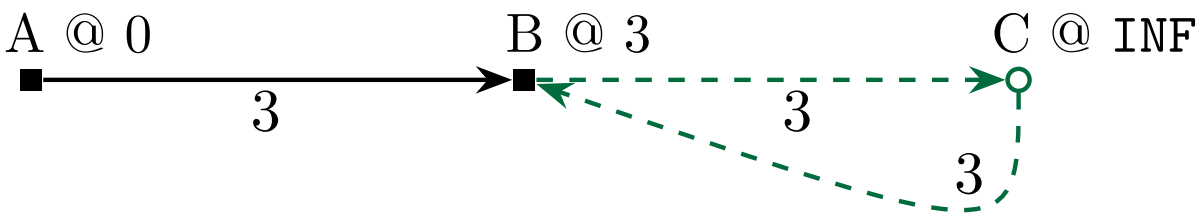


Overflow Strikes Again

The longest optimal path has `SIZE-1` links

so say we set `elabel`'s upper bound to $\lfloor$`MAX/(SIZE-1)`$\rfloor$

`MAX` $= 7$, `SIZE` $= 3$, so $0 \leqslant$ `elabel`$(\gamma, e) \leqslant 3$.

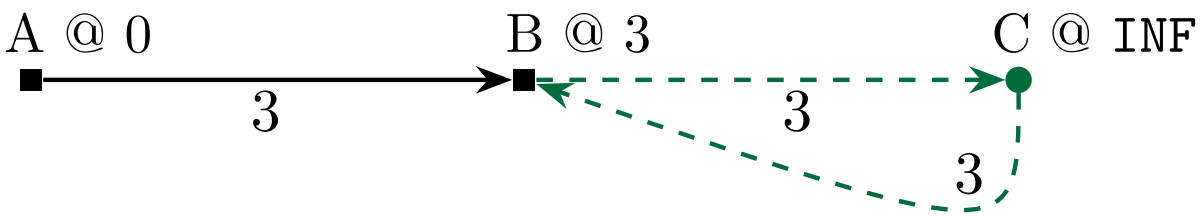


Overflow Strikes Again

The longest optimal path has `SIZE-1` links

so say we set `elabel`'s upper bound to $\lfloor$`MAX/(SIZE-1)`$\rfloor$

`MAX` $= 7$, `SIZE` $= 3$, so $0 \leqslant$ `elabel`$(\gamma, e) \leqslant 3$.

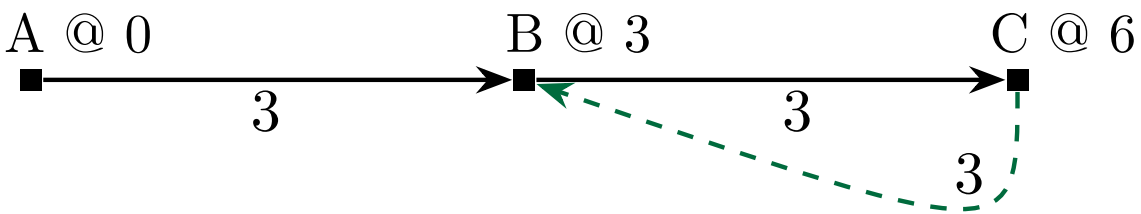


Overflow Strikes Again

The longest optimal path has `SIZE-1` links

so say we set `elabel`'s upper bound to $\lfloor$`MAX/(SIZE-1)`$\rfloor$

`MAX` $= 7$, `SIZE` $= 3$, so $0 \leqslant$ `elabel`$(\gamma, e) \leqslant 3$.

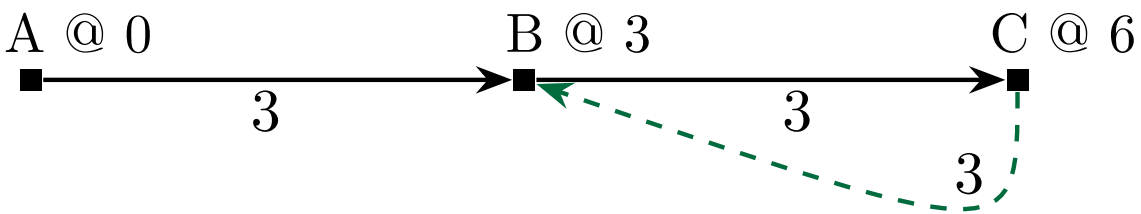


Overflow Strikes Again

The longest optimal path has `SIZE-1` links

so say we set `elabel`'s upper bound to $\lfloor$`MAX/(SIZE-1)`$\rfloor$

`MAX` $= 7$, `SIZE` $= 3$, so $0 \leqslant$ `elabel`$(\gamma, e) \leqslant 3$.



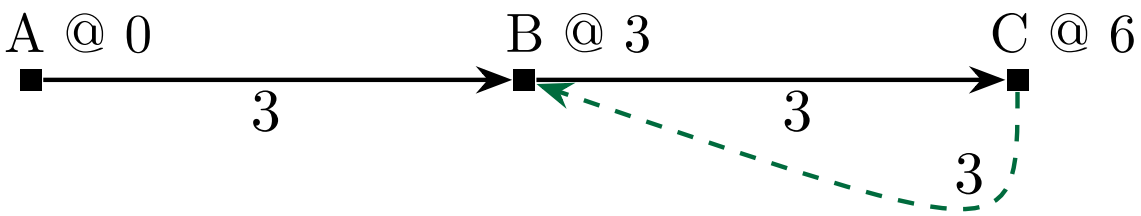
if $3 > 9$ then relax C $\rightsquigarrow$ B

Overflow Strikes Again

The longest optimal path has `SIZE-1` links

so say we set `elabel`'s upper bound to $\lfloor$`MAX/(SIZE-1)`$\rfloor$

`MAX` $= 7$, `SIZE` $= 3$, so $0 \leqslant$ `elabel`$(\gamma, e) \leqslant 3$.



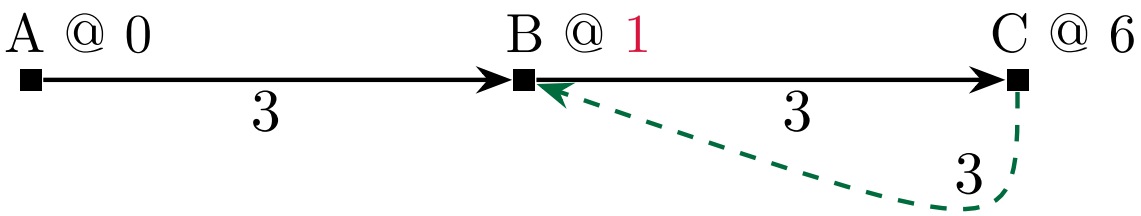
if $3 > 1$ then relax C $\rightsquigarrow$ B

Overflow Strikes Again

The longest optimal path has `SIZE-1` links

so say we set `elabel`'s upper bound to $\lfloor$`MAX/(SIZE-1)`$\rfloor$

`MAX` $= 7$, `SIZE` $= 3$, so $0 \leqslant$ `elabel`$(\gamma, e) \leqslant 3$.

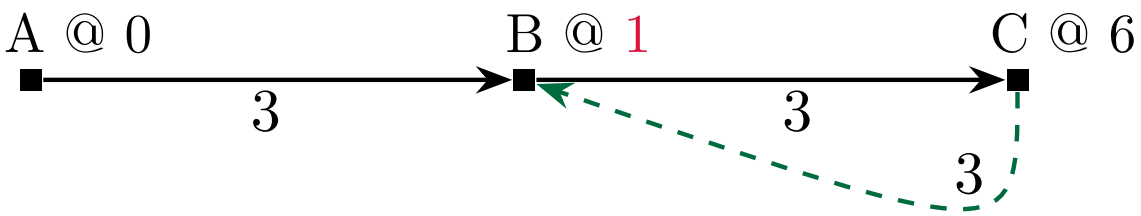


if $3 > 1$ then relax C $\rightsquigarrow$ B

Overflow Strikes Again

The longest optimal path has `SIZE-1` links
so say we set `elabel`'s upper bound to $\lfloor$`MAX/(SIZE-1)`$\rfloor$

`MAX` $= 7$, `SIZE` $= 3$, so $0 \leqslant$ `elabel`$(\gamma, e) \leqslant 3$.



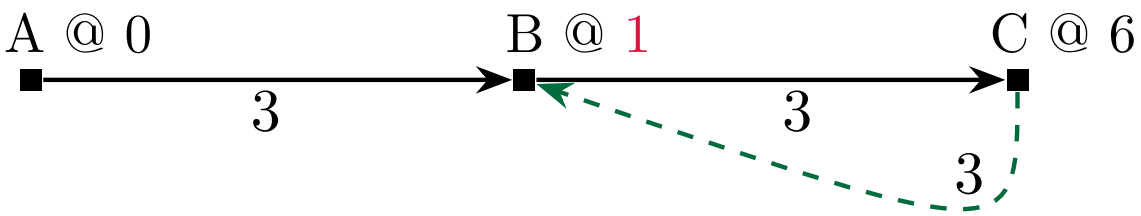
if $3 > 1$ then relax C $\rightsquigarrow$ B

One solution: Conservatively set upper bound to $\lfloor$`MAX/SIZE`$\rfloor$

Overflow Strikes Again

The longest optimal path has `SIZE-1` links
so say we set `elabel`'s upper bound to $\lfloor$`MAX/(SIZE-1)`$\rfloor$

`MAX` $= 7$, `SIZE` $= 3$, so $0 \leqslant$ `elabel`$(\gamma, e) \leqslant 3$.



if $3 > 1$ then relax C $\rightsquigarrow$ B

One solution: Conservatively set upper bound to $\lfloor$`MAX/SIZE`$\rfloor$

Max path cost is then $\lfloor$`MAX/SIZE`$\rfloor$ `* (SIZE-1) = MAX -` $\lfloor$`MAX/SIZE`$\rfloor$

Overflow Strikes Again

There are other ways to fix this!

Overflow Strikes Again

There are other ways to fix this!

- Refactor troublesome addition as subtraction

Overflow Strikes Again

There are other ways to fix this!

- Refactor troublesome addition as subtraction
- Never look back into optimized part

Overflow Strikes Again

There are other ways to fix this!

- Refactor troublesome addition as subtraction
- Never look back into optimized part
- Your suggestion here

Overflow Strikes Again

There are other ways to fix this!

- Refactor troublesome addition as subtraction
- Never look back into optimized part
- Your suggestion here
- Your suggestion here

Overflow Strikes Again

There are other ways to fix this!

- Refactor troublesome addition as subtraction
- Never look back into optimized part
- Your suggestion here
- Your suggestion here

That is not the point

Overflow Strikes Again

There are other ways to fix this!

- Refactor troublesome addition as subtraction
- Never look back into optimized part
- Your suggestion here
- Your suggestion here

That is not the point

- Intuition supports `INF = MAX`

Overflow Strikes Again

There are other ways to fix this!

- Refactor troublesome addition as subtraction
- Never look back into optimized part
- Your suggestion here
- Your suggestion here

That is not the point

- Intuition supports `INF = MAX`
- No reason to do any of the above

Overflow Strikes Again

There are other ways to fix this!

- Refactor troublesome addition as subtraction
- Never look back into optimized part
- Your suggestion here
- Your suggestion here

That is not the point

- Intuition supports `INF = MAX`
- No reason to do any of the above... until today