Problem L: Well Spoken

Algorithmen für Programmierwettbewerbe

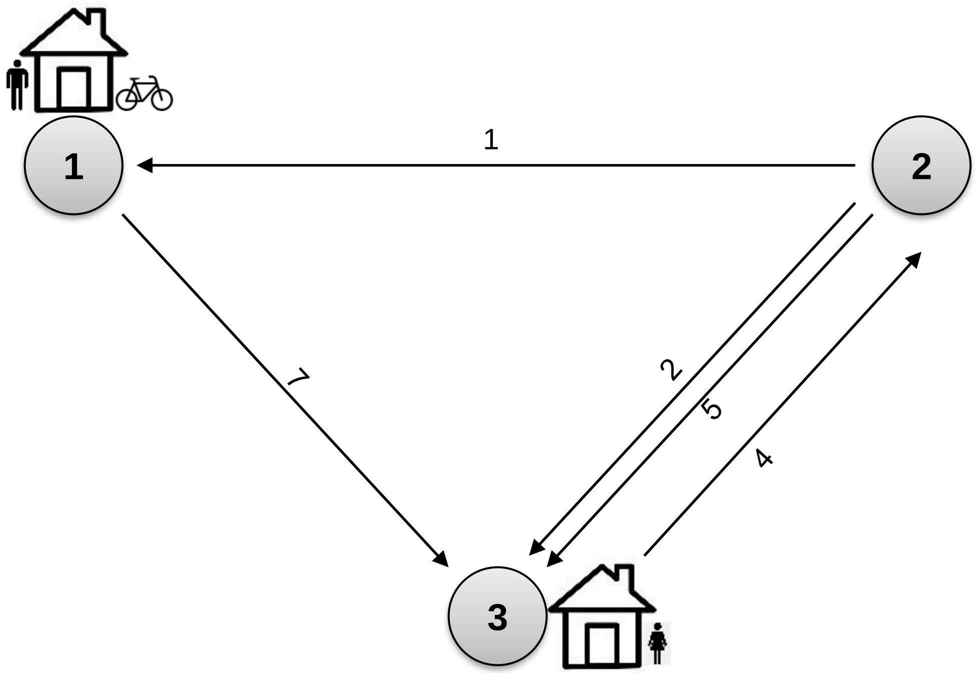
Hendrik Meininger

Johannes Schleicher

Problem

Given:

- -Timespan [A:B]
- -Road network
- -Intersections as nodes
- -Streets as edges



Problem:

Find the minimum maximum waiting time, given that Janet be ready between [A:B]

Problem

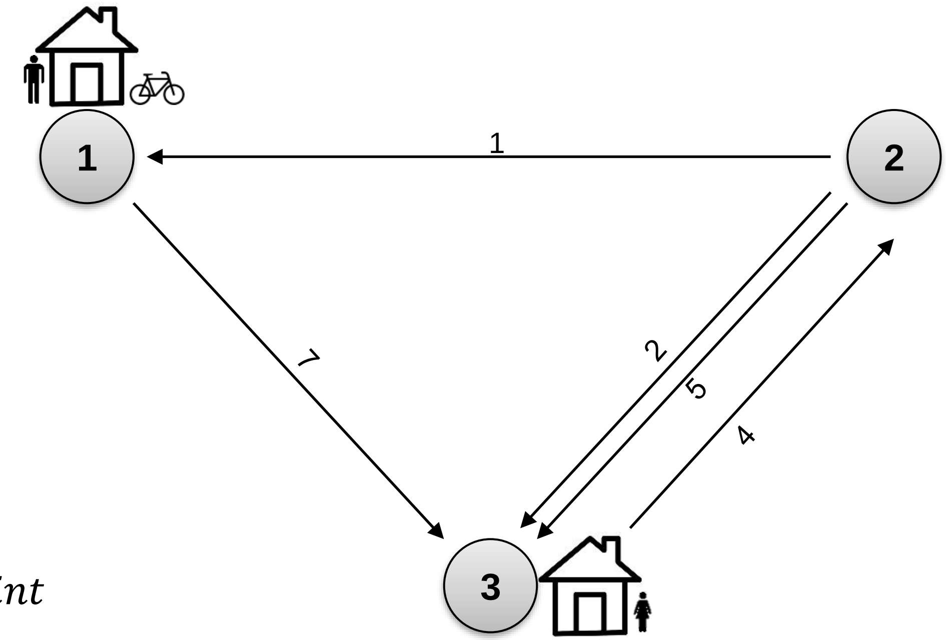
- Input:

```
10 20
3 5
1 3 7
2 1 1
2 3 2
2 3 5
3 2 4
```

[A:B]
|V| |E|

E
$e(start, end, cost)$

[10:20]



- Output:

```
6
```

Output is always an int

Problem:

Find the minimum maximum waiting time, given that Janet be ready between [A:B]

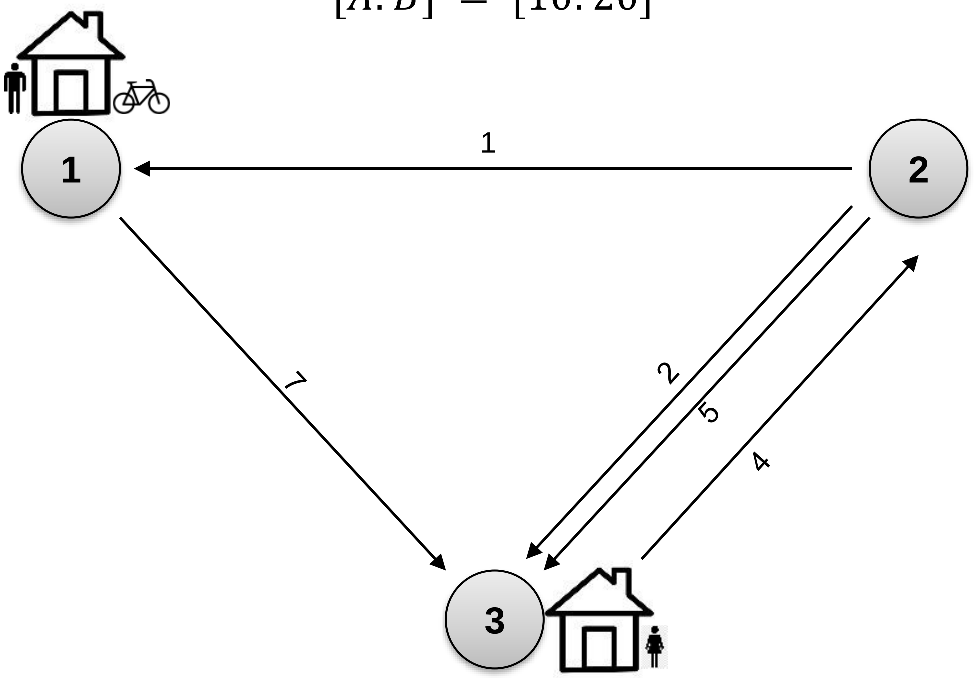
Solution:

1. Compute the distance from Richard to all vertices and from all vertices to Janet using two runs of Dijkstra
2. Binary search on the maximum waiting time boundaries
3. Check if given delay δ is possible

Approach - Example

$$0 \leq delay \leq 7 \rightarrow \delta = \left\lfloor \frac{7-0}{2} \right\rfloor + 0 = 3$$

$$[A{:}\,B] \; = \; [10{:}\,20]$$



Check if delay δ is possible:

1. Mark vertices u as “good“ if $distFromHome(u) + distToJanet(u) \leq A + \delta$ and $\mathrm{distToJanet(u)} \leq \delta$

   If node u does not satisfy this condition, then there won't be a route through u that satisfies the delay delta, given that the signal will come at time A or when arriving at u

Approach - Example

1. Step:
$0 \leq delay \leq 7 \rightarrow \delta = 3$

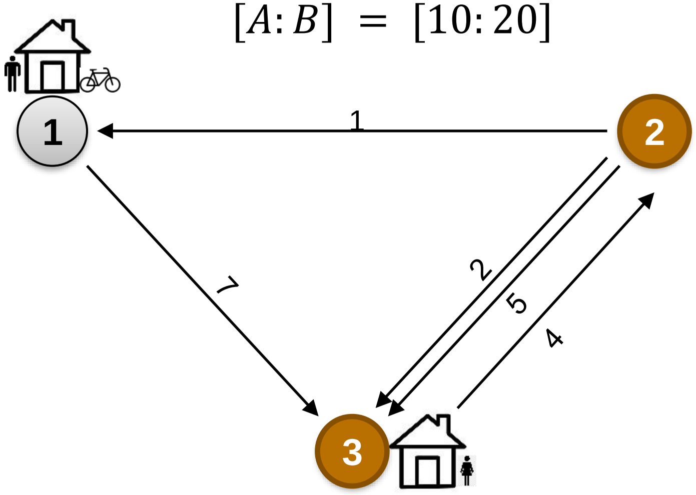
Mark vertices u as “good“ if:
$distFromHome[u] + distToJanet[u] \leq A + \delta \; and \; distToJanet[u] \leq \delta$

u_1: $0 + 7 \leq 10 + 3 \; and \; 7 \leq 3$

u_2: $11 + 2 \leq 10 + 3 \; and \; 2 \leq 3$

u_3: $7 + 0 \leq 10 + 3 \; and \; 0 \leq 3$

$[A\!:B] = [10\!:20]$



Approach

Check if delay δ is possible:

1. Mark vertices u as “good“ if $distFromHome(u) + distToJanet(u) \leq A + \delta$ and $\mathrm{distToJanet(u)} \leq \delta$
   If node u does not satisfy this condition, then there won't be a route through u that satisfies the delay delta, given that the signal will come at time A or when arriving at u

2. Propagate: if u is good and $u \xrightarrow{l} v$ with $l + distToJanet(v) \leq \delta$, then mark v and edge l as good too
   An edge is good, should Richard still be able to meet the delay, if Janet calls while Richard is currently riding that edge

Approach - Example

1. Step:
$0 \leq delay \leq 7 \rightarrow \delta = 3$

$e(start, end, cost)$

Propagate: if u is good and $u \xrightarrow{l} v$ with $l + distToJanet[v] \leq \delta$,
then mark v and edge l as good too

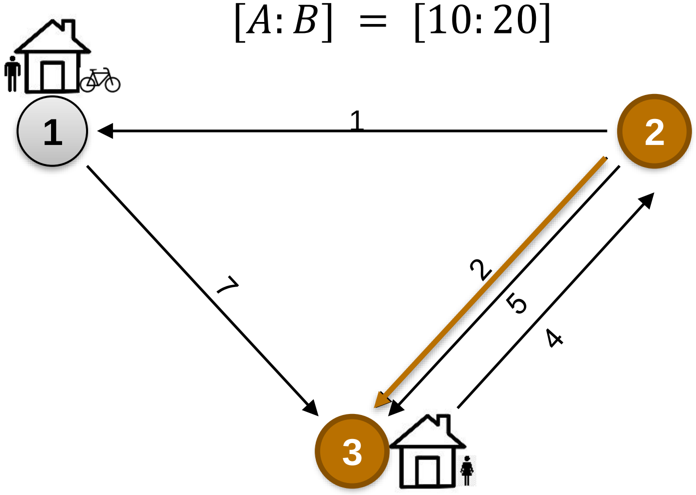
$u_2,\ e(2,1,1)$: $1 + 7 \leq 3$

$u_2,\ e(2,3,2)$: $2 + 0 \leq 3$

$u_2,\ e(2,3,5)$: $5 + 0 \leq 3$

$u_3,\ e(3,2,4)$: $4 + 2 \leq 3$

$[A{:}B] = [10{:}20]$



Approach

Problem › **Approach** › Runtime

Check if delay δ is possible:

1. Mark vertices u as “good“ if $distFromHome(u) + distToJanet(u) \leq A + \delta$ and $\mathrm{distToJanet(u)} \leq \delta$
   If node u does not satisfy this condition, then there won't be a route through u that satisfies the delay delta, given that the signal will come at time A or when arriving at u

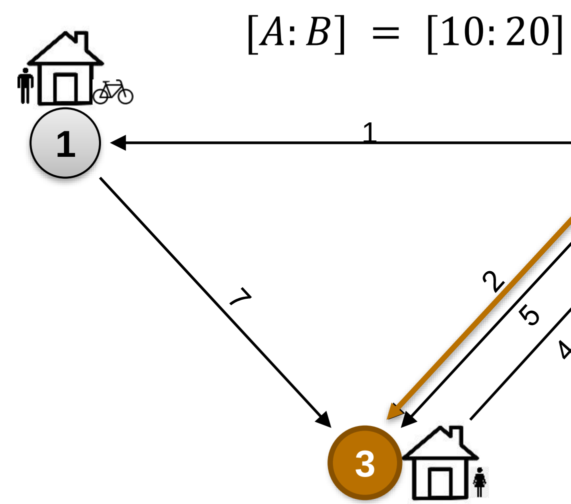
2. Propagate: if u is good and $u \xrightarrow{l} v$ with $l + distToJanet(v) \leq \delta$, then mark v and edge l as good too
   An edge is good, should Richard still be able to meet the delay, if Janet calls while Richard is currently riding that edge

3. If subgraph of good edges has cycle → delay δ is possible
   We can stay in the cycle until Janet calls and arrive at her place at most delta after she has called

Approach - Example

1. Step:
$0 \leq delay \leq 7 \rightarrow \delta = 3$

If subgraph of good edges has cycle → delay δ is possible
$hasCycle() \ = \ false$



Check if delay δ is possible:

1. Mark vertices u as "good" if $distFromHome(u) + distToJanet(u) \leq A + \delta$ and $\mathrm{distToJanet(u)} \leq \delta$
   If node u does not satisfy this condition, then there won't be a route through u that satisfies the delay delta, given that the signal will come at time A or when arriving at u

2. Propagate: if u is good and $u \xrightarrow{l} v$ with $l + distToJanet(v) \leq \delta$, then mark v and edge l as good too
   An edge is good, should Richard still be able to meet the delay, if Janet calls while Richard is currently riding that edge

3. If subgraph of good edges has cycle → delay δ is possible
   We can stay in the cycle until Janet calls and arrive at her place at most delta after she has called

4. Otherwise, the subgraph of good nodes and edges is acyclic. Compute longest time Richard can stay in the subgraph. If this is $\geq B$ then delay δ is possible

Approach – longestPath()

Auxiliary variables:

$indeg[u] := \; indegree\ of\ \ good\ nodes\ in\ subgraph$

corresponds to the number of good incoming edges

$goodNodes \coloneqq u.isGood\ and\ indeg[u] = 0$

goodNodes contains at the beginning all good nodes with indegree 0, so that one can calculate the longest Path correctly afterwards

$latest[u] = A + delay\ \ - distToJanet[u]$

for each good node u, the latest time of arrival at the node, so that the delay can still be met

Approach - Example

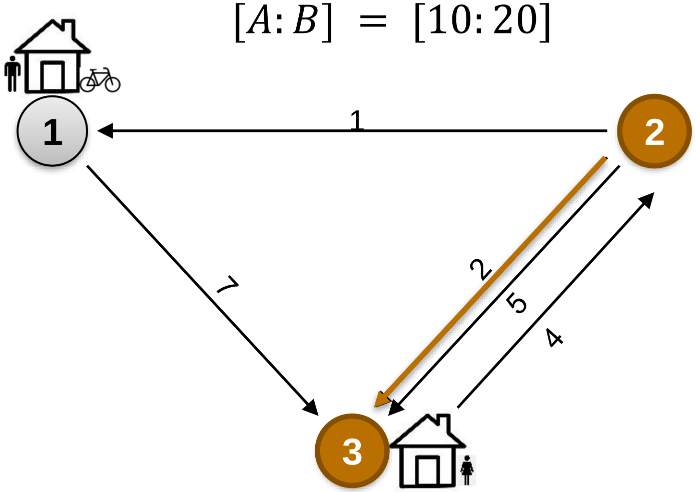
1. Step:
$0 \leq delay \leq 7 \rightarrow \delta = 3$

Compute longest time Richard can stay in the subgraph
If this is $\geq B$ then delay δ is possible
$latest[u] = A + delay \; - distToJanet[u]$

$latest[2] = 10 + 3 - 2 = 11$

$latest[3] = 10 + 3 - 0 = 13$

$[A{:}B] \; = \; [10{:}20]$



Approach – longestPath()

Problem › **Approach** › Runtime

Pseudocode:

$longestPath(\delta)$

$while(!\, goodNodes.isEmpty)$

$u = goodNodes.remove(0)$

$if\ \ latest[u] + distToJanet[u] \geq B$

$return\ true$

$for\ Edge\ e\ in\ goodEdges[u]$

$indeg[e.dest] \mathrel{-}= 1$

$if\ indeg[e.dest] == 0$

$goodNodes.add(e.dest)$

$latest[e.dest] = \max(latest[e.dest],\ latest[u] + e.weight)$

$return\ false$

Approach - Example

1. Step:
$0 \leq delay \leq 7 \rightarrow \delta = 3$

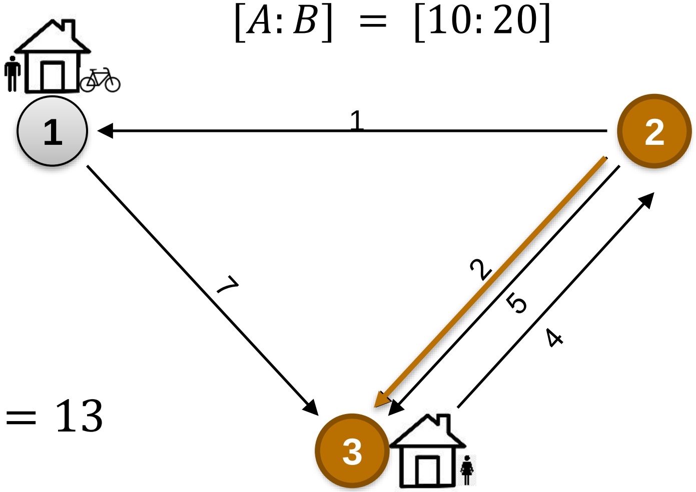
Compute longest time Richard can stay in the subgraph
If this is $\geq B$ then delay δ is possible
$latest[u] = A + delay - distToJanet[u]$

$latest[2] = 11$
$latest[3] = 13$

Start with good nodes with $indeg[u] == 0$
Check if: $latest[u] + distToJanet[u] \geq B$

$latest[2] + distToJanet[2] = 11 + 2 \geq 20$
$latest[3] = \max\big(latest[3], latest[2] + e(2,3,2)\big) = 13$
$latest[3] + distToJanet[3] = 13 + 0 \geq 20$

$[A:B] = [10:20]$



Approach - Example

2. Step:
$4 \leq delay \leq 7 \rightarrow \delta = 5$

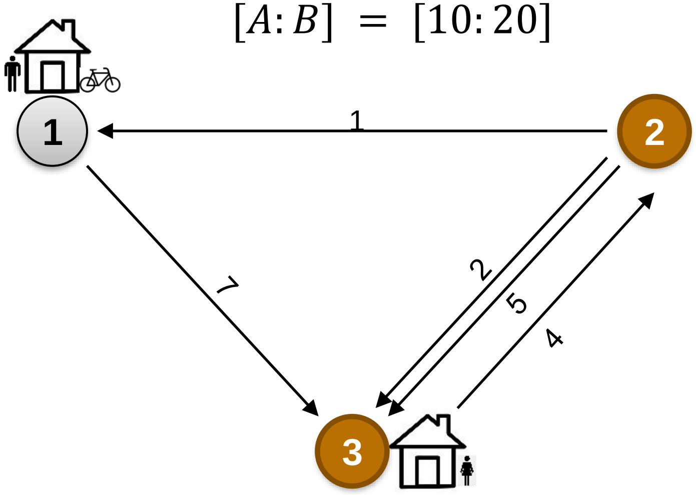
Mark vertices u as "good" if:
$distFromHome[u] + distToJanet[u] \leq A + \delta$ and $distToJanet[u] \leq \delta$

u_1: $0 + 7 \leq 10 + 5$ and $7 \leq 5$

u_2: $11 + 2 \leq 10 + 5$ and $7 \leq 5$

u_3: $7 + 0 \leq 10 + 5$ and $7 \leq 5$

$[A:B] = [10:20]$

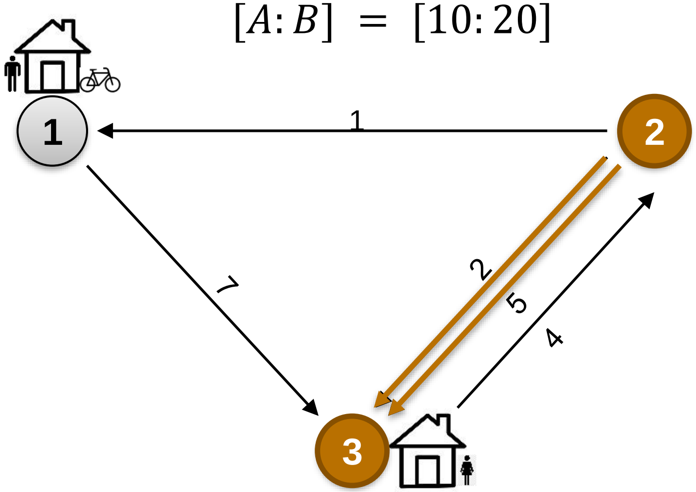


2. Step:
$4 \leq delay \leq 7 \rightarrow \delta = 5$

Propagate: if u is good and $u \xrightarrow{l} v$ with $l + distToJanet[v] \leq \delta$, then mark v and edge l as good too

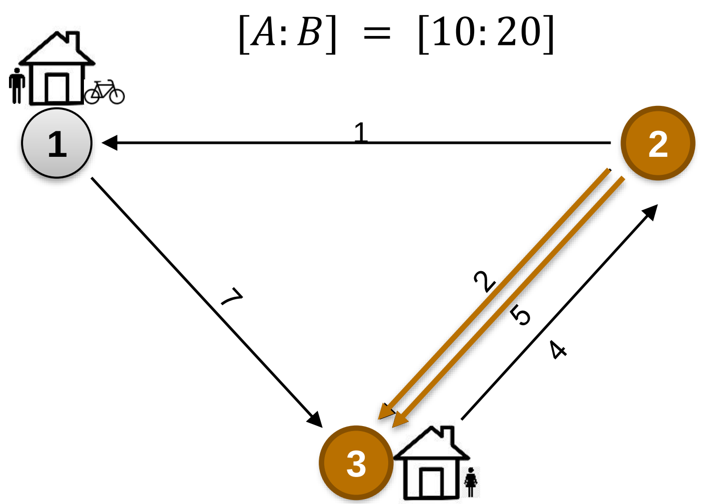
$u_2,\ e(2,1,1)$:	$1 + 7 \leq 5$
$u_2,\ e(2,3,2)$:	$2 + 0 \leq 5$
$u_2,\ e(2,3,5)$:	$5 + 0 \leq 5$
$u_3,\ e(3,2,4)$:	$4 + 2 \leq 5$

$[A{:}B] = [10{:}20]$



2. Step:
$4 \leq delay \leq 7 \rightarrow \delta = 5$

If subgraph of good edges has cycle → delay δ is possible
$hasCycle() = false$



2. Step:
$4 \leq delay \leq 7 \rightarrow \delta = 5$

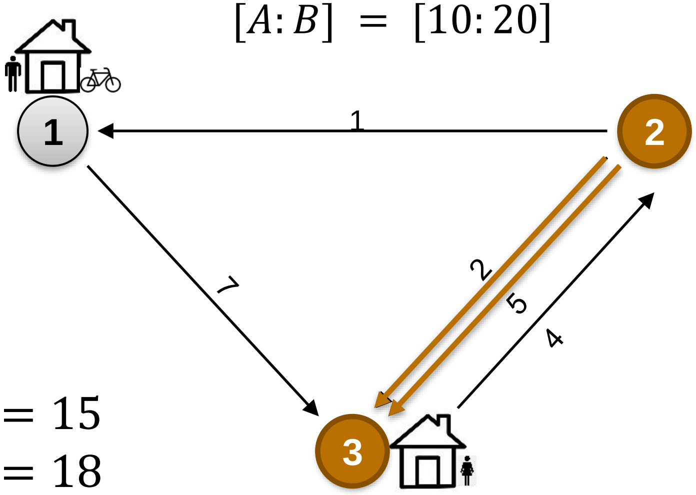
Compute longest time Richard can stay in the subgraph
If this is $\geq B$ then delay δ is possible
$latest[u] = A + delay \ - distToJanet[u]$

$latest[2] = 10 + 5 - 2 = 13$
$latest[3] = 10 + 5 - 0 = 15$

Start with good nodes with $indeg[u] \ == \ 0$
Check if: $latest[u] + distToJanet[u] \geq B$

$latest[2] + distToJanet[2] = 11 + 2 \geq 20$
$latest[3] = \max(latest[3], latest[2] + e(2,3,2)) = 15$
$latest[3] = \max(latest[3], latest[2] + e(2,3,5)) = 18$
$latest[3] + distToJanet[3] = 18 + 0 \geq 20$

$[A:B] \ = \ [10:20]$



Approach - Example

3. Step:
$6 \leq delay \leq 7 \rightarrow \delta = 6$

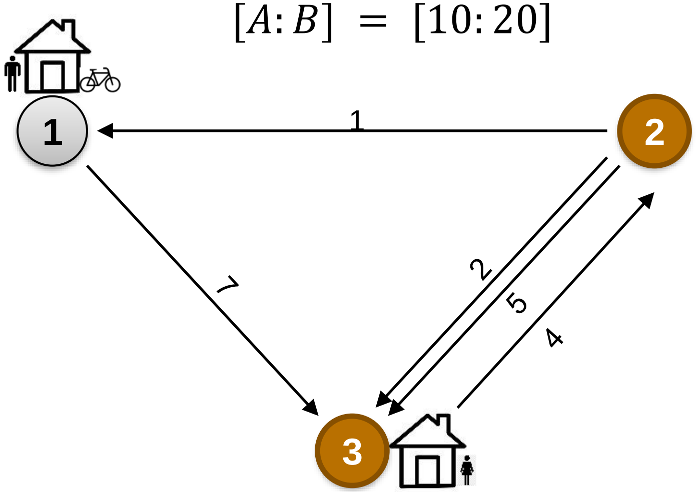
Mark vertices u as “good“ if:
$distFromHome[u] + distToJanet[u] \leq A + \delta \text{ and } distToJanet[u] \leq \delta$

u_1: $0 + 7 \leq 10 + 6 \text{ and } 7 \leq 6$

u_2: $11 + 2 \leq 10 + 6 \text{ and } 7 \leq 6$

u_3: $7 + 0 \leq 10 + 6 \text{ and } 7 \leq 6$

$[A{:}\,B] = [10{:}\,20]$



Approach - Example

3. Step:
$6 \leq delay \leq 7 \rightarrow \delta = 6$

Propagate: if u is good and $u \xrightarrow{l} v$ with $l + distToJanet[v] \leq \delta$, then mark v and edge l as good too

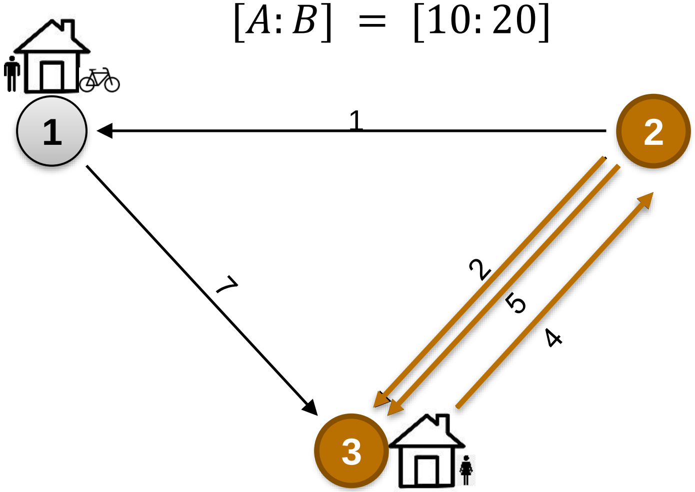
$u_2,\ e(2,1,1)$: $1 + 7 \leq 6$

$u_2,\ e(2,3,2)$: $2 + 0 \leq 6$

$u_2,\ e(2,3,5)$: $5 + 0 \leq 6$

$u_3,\ e(3,2,4)$: $4 + 2 \leq 6$

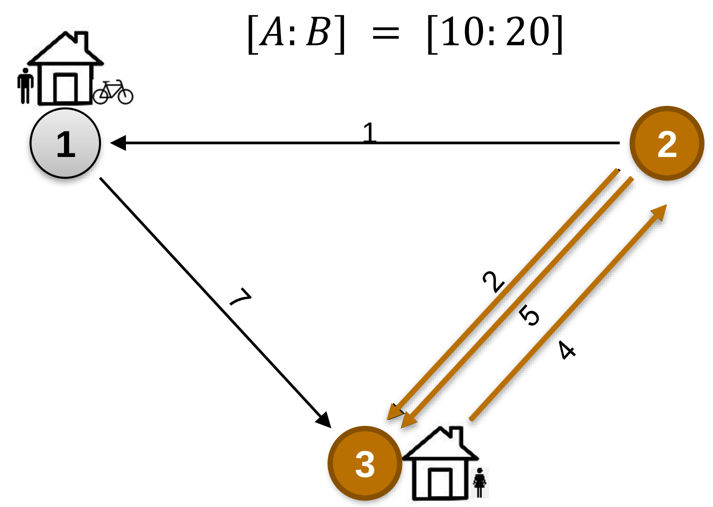
$[A: B] = [10: 20]$



Approach - Example

3. Step:
$6 \leq delay \leq 7 \rightarrow \delta = 6$

If subgraph of good edges has cycle → delay δ is possible
$hasCycle() = true$



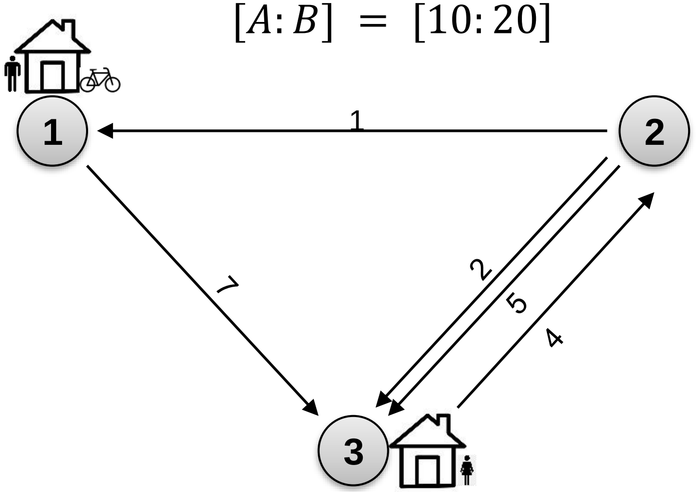
Approach - Example

4. Step:

$6 \leq delay \leq 6 \rightarrow left == right$

$\rightarrow Output := 6$

$[A{:}B] = [10{:}20]$



Runtime

Runtime:

WellSpoken()

Dijkstra
Dijkstra

BinarySearch

checkDelay(δ)

checkDelay: $O(?)$

Dijkstra, Dijkstra: $O(E + V \log V)$

BinarySearch: $O(\log w) * O(?)$
with $w = distToJanet(s)$

$checkDelay(\delta)$

1. Mark vertices u as “good“ if $distFromHome(u) + distToJanet(u) \leq A + \delta$ and $\mathrm{distToJanet(u)} \leq \delta$
2. Propagate: if u is good and $u \xrightarrow{l} v$ with $l + distToJanet(v) \leq \delta$, then mark v and edge l as good too
3. If subgraph of good edges has cycle → delay δ is possible
4. Otherwise the subgraph of good nodes and edges is acyclic. Compute longest time Richard can stay in the subgraph. If this is $\geq B$ then delay δ is possible

$checkDelay(\delta)$

1. Mark vertices u as "good" if $distFromHome(u) + distToJanet(u) \leq A + \delta$ and $\mathrm{distToJanet(u)} \leq \delta$ — $O(V)$
2. Propagate: if u is good and $u \xrightarrow{l} v$ with $l + distToJanet(v) \leq \delta$, then mark v and edge l as good too
3. If subgraph of good edges has cycle → delay δ is possible
4. Otherwise the subgraph of good nodes and edges is acyclic. Compute longest time Richard can stay in the subgraph. If this is $\geq B$ then delay δ is possible

Runtime

$checkDelay(\delta)$	
1. Mark vertices u as "good" if $distFromHome(u) + distToJanet(u) \leq A + \delta$ and $\mathrm{distToJanet(u)} \leq \delta$	$O(V)$
2. Propagate: if u is good and $u \xrightarrow{l} v$ with $l + distToJanet(v) \leq \delta$, then mark v and edge l as good too	$O(E)$
3. If subgraph of good edges has cycle → delay δ is possible	
4. Otherwise the subgraph of good nodes and edges is acyclic. Compute longest time Richard can stay in the subgraph. If this is $\geq B$ then delay δ is possible	

Runtime

$checkDelay(\delta)$		
1.	Mark vertices u as "good" if $distFromHome(u) + distToJanet(u) \leq A + \delta$ and $\mathrm{distToJanet(u)} \leq \delta$	$O(V)$
2.	Propagate: if u is good and $u \xrightarrow{l} v$ with $l + distToJanet(v) \leq \delta$, then mark v and edge l as good too	$O(E)$
3.	If subgraph of good edges has cycle → delay δ is possible	$O(V + E)$
4.	Otherwise the subgraph of good nodes and edges is acyclic. Compute longest time Richard can stay in the subgraph. If this is $\geq B$ then delay δ is possible	

Runtime

$checkDelay(\delta)$	
1. Mark vertices u as "good" if $distFromHome(u) + distToJanet(u) \leq A + \delta$ and $\mathrm{distToJanet(u)} \leq \delta$	$O(V)$
2. Propagate: if u is good and $u \xrightarrow{l} v$ with $l + distToJanet(v) \leq \delta$, then mark v and edge l as good too	$O(E)$
3. If subgraph of good edges has cycle → delay δ is possible	$O(V + E)$
4. Otherwise the subgraph of good nodes and edges is acyclic. Compute longest time Richard can stay in the subgraph. If this is $\geq B$ then delay δ is possible	$O(?)$

Runtime – longestPath()

Problem › Approach › **Runtime**

Pseudocode:

```
longestPath(δ)
    while(!goodNodes.isEmpty)
        u = goodNodes.remove(0)
        if latest[u] + distToJanet[u] ≥ B
            return true
        for Edge e in goodEdges[u]
            indeg[e.dest] −= 1
            if indeg[e.dest] == 0
                goodNodes.add(e.dest)
            latest[e.dest] = max(latest[e.dest], latest[u] + e.weight)
    return false
```

Runtime – longestPath()

Problem › Approach › **Runtime**

Pseudocode:

$longestPath(\delta)$

```
while(! goodNodes.isEmpty)                                              O(V)
    u = goodNodes.remove(0)
    if latest[u] + distToJanet[u] ≥ B
        return true
    for Edge e in goodEdges[u]
        indeg[e.dest] −= 1
        if indeg[e.dest] == 0
            goodNodes.add(e.dest)
        latest[e.dest] = max(latest[e.dest], latest[u] + e.weight)
return false
```

Runtime – longestPath()

Pseudocode:

$longestPath(\delta)$

$while(!\,goodNodes.isEmpty)$ $\quad O(V)$

$\quad u = goodNodes.remove(0)$

$\quad if\ \ latest[u] + distToJanet[u] \geq B$

$\quad\quad return\ true$

$\quad for\ Edge\ e\ in\ goodEdges[u]$ $\quad O(E)$

$\quad\quad indeg[e.dest] -= 1$

$\quad\quad if\ indeg[e.dest] == 0$

$\quad\quad\quad goodNodes.add(e.dest)$

$\quad\quad latest[e.dest] = \max(latest[e.dest],\ latest[u] + e.weight)$

$return\ false$

$\rightarrow O(V + E)$

Runtime

$checkDelay(\delta)$		
1.	Mark vertices u as "good" if $distFromHome(u) + distToJanet(u) \leq A + \delta$ and $\mathrm{distToJanet(u)} \leq \delta$	$O(V)$
2.	Propagate: if u is good and $u \xrightarrow{l} v$ with $l + distToJanet(v) \leq \delta$, then mark v and edge l as good too	$O(E)$
3.	If subgraph of good edges has cycle → delay δ is possible	$O(V+E)$
4.	Otherwise the subgraph of good nodes and edges is acyclic. Compute longest time Richard can stay in the subgraph. If this is $\geq B$ then delay δ is possible	$O(V+E)$

$checkDelay(\delta)$

1. Mark vertices u as "good" if $distFromHome(u) + distToJanet(u) \leq A + \delta$ and $\mathrm{distToJanet(u)} \leq \delta$ — $O(V)$
2. Propagate: if u is good and $u \xrightarrow{l} v$ with $l + distToJanet(v) \leq \delta$, then mark v and edge l as good too — $O(E)$
3. If subgraph of good edges has cycle → delay δ is possible — $O(V + E)$
4. Otherwise the subgraph of good nodes and edges is acyclic. Compute longest time Richard can stay in the subgraph. If this is $\geq B$ then delay δ is possible — $O(V + E)$

$\rightarrow O(V + E)$

Runtime

Runtime:

WellSpoken()

Dijkstra
Dijkstra

BinarySearch

checkDelay(δ)

$O(V + E)$

$O(E + V \log V)$

$O(\log w) * O(V + E)$

with $w = distToJanet(s)$

$$Runtime\colon\ O((V + E) * \log(w))$$