

Dynamic Graph Component Tracking

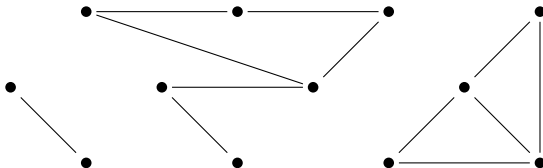
Seminar Talk

Angelika Schwarz

<http://comsys.rwth-aachen.de>

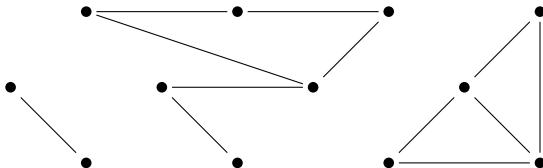
Aachen, February 7, 2014

Graph Connectivity



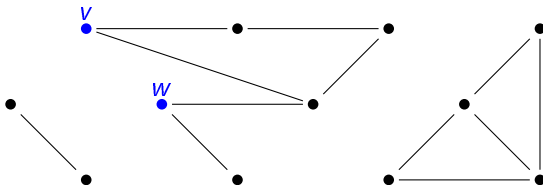
Connectivity Queries

$\text{connected}(v, w)$



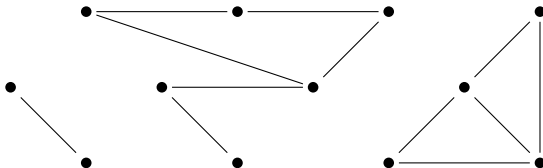
Connectivity Queries

connected(v, w)



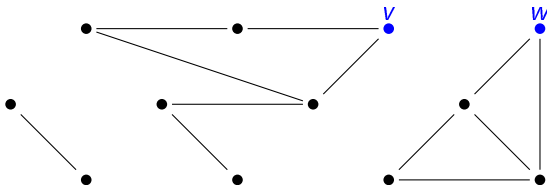
Updates: Insertions

$\text{insert}(e = \{v, w\})$



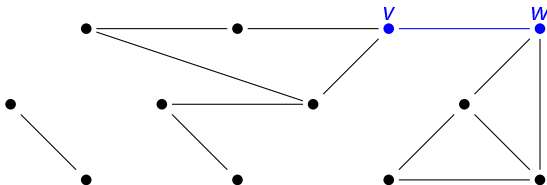
Updates: Insertions

$\text{insert}(e = \{v, w\})$



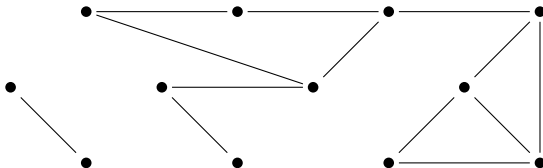
Updates: Insertions

$\text{insert}(e = \{v, w\})$



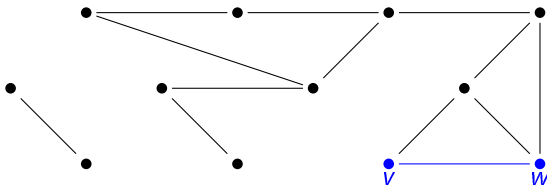
Updates: Deletions

$\text{delete}(e = \{v, w\})$



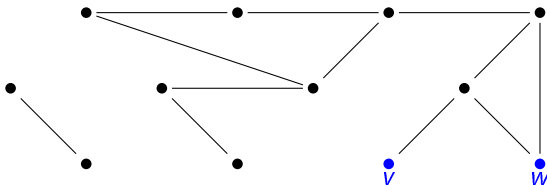
Updates: Deletions

delete($e = \{v, w\}$)



Updates: Deletions

delete($e = \{v, w\}$)



Dynamic Connectivity

The graph is subject to a sequence of

- $\text{connected}(v, w)$
- $\text{insert}(e = \{v, w\})$
- $\text{delete}(e = \{v, w\})$

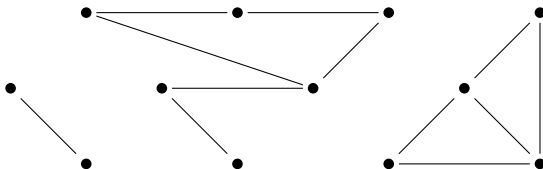
Dynamic Connectivity

The graph is subject to a sequence of

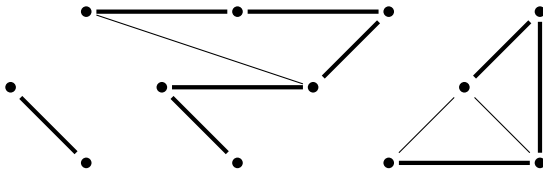
- $\text{connected}(v, w)$
- $\text{insert}(e = \{v, w\})$
- $\text{delete}(e = \{v, w\})$

Maintain a spanning forest of the graph

Graph Connectivity via Spanning Forests

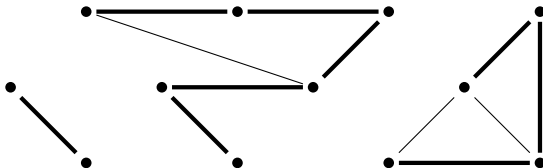


Graph Connectivity via Spanning Forests



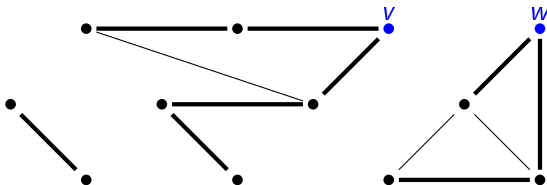
Updates: Insertions

$\text{insert}(e = \{v, w\})$



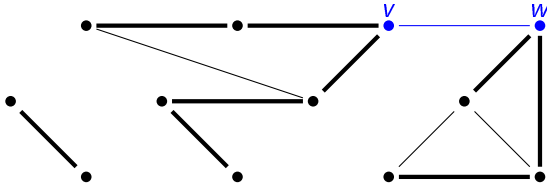
Updates: Insertions

$\text{insert}(e = \{v, w\})$



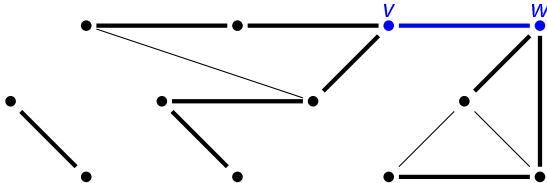
Updates: Insertions

$\text{insert}(e = \{v, w\})$



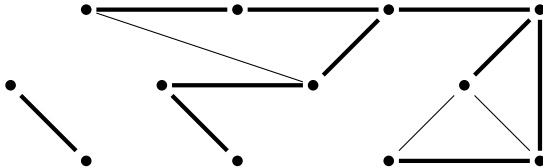
Updates: Insertions

$\text{insert}(e = \{v, w\})$



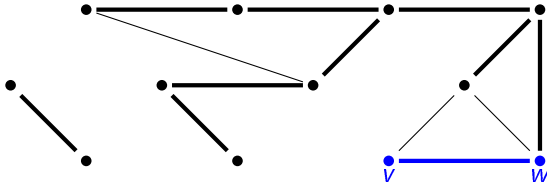
Updates: Deletions

$\text{delete}(e = \{v, w\})$



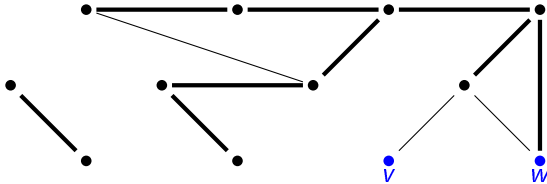
Updates: Deletions

delete($e = \{v, w\}$)



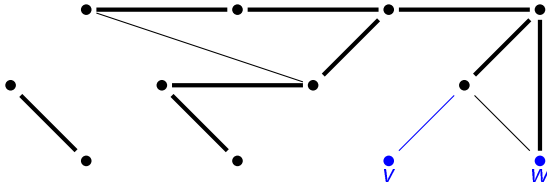
Updates: Deletions

delete($e = \{v, w\}$)



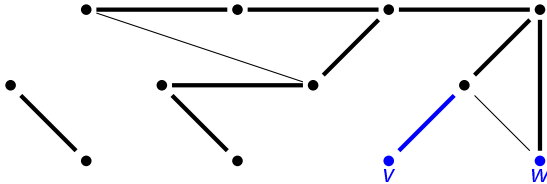
Updates: Deletions

delete($e = \{v, w\}$)



Updates: Deletions

delete($e = \{v, w\}$)



Agenda

Achieved so far:

- Connectivity queries
- Insertions
- Non-tree edge deletions

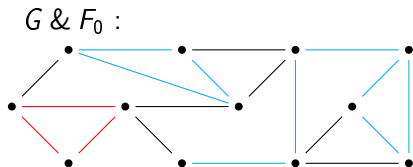
Problem: Deletion of a tree edge!

Goals:

- After tree edge deletion: decide if components are really disconnected or find a replacement edge
- polylogarithmic amortised time bound

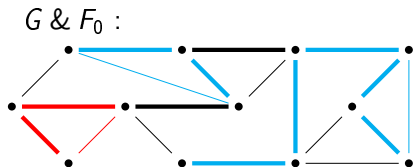
Hierarchical Decomposition

- *level* of an edge:
 $0, \dots, \lfloor \log_2 n \rfloor$
- level increase: 'pay' for unsuccessful testing



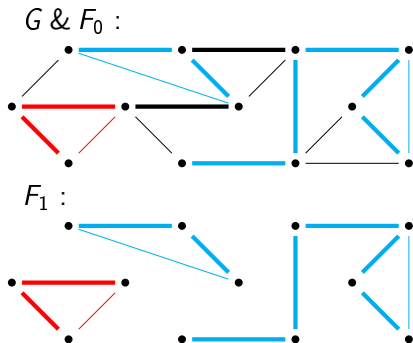
Hierarchical Decomposition

- *level* of an edge:
 $0, \dots, \lfloor \log_2 n \rfloor$
- level increase: 'pay' for
unsuccessful testing



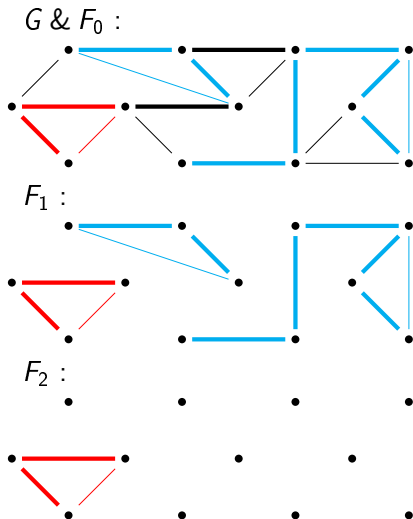
Hierarchical Decomposition

- *level* of an edge:
 $0, \dots, \lfloor \log_2 n \rfloor$
- level increase: 'pay' for
unsuccessful testing



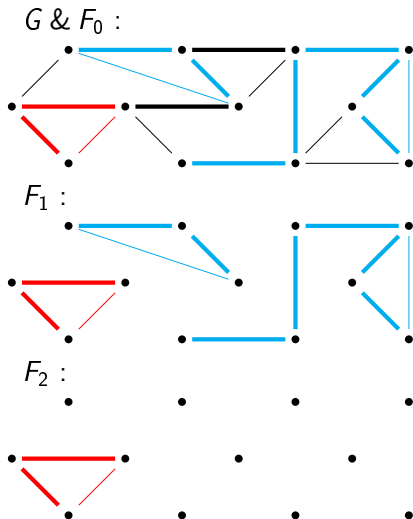
Hierarchical Decomposition

- *level* of an edge:
 $0, \dots, \lfloor \log_2 n \rfloor$
- level increase: 'pay' for
unsuccessful testing



Hierarchical Decomposition

- *level* of an edge:
 $0, \dots, \lfloor \log_2 n \rfloor$
- level increase: 'pay' for
unsuccessful testing
- $F_0 \supseteq F_1 \supseteq \dots \supseteq F_{\lfloor \log_2 n \rfloor}$

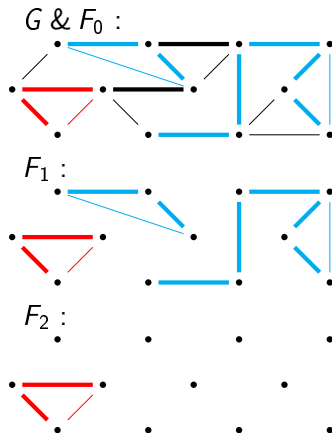


Invariants

- 1 The number of vertices in every connected component in F_i is at most $\lfloor n/2^i \rfloor$.

Invariants

- 1 The number of vertices in every connected component in F_i is at most $\lfloor n/2^i \rfloor$.

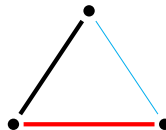
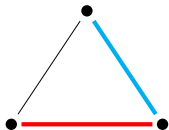


Invariants

- 1 The number of vertices in every connected component in F_i is at most $\lfloor n/2^i \rfloor$.
- 2 F is a maximum spanning forest of G .

Invariants

- 1 The number of vertices in every connected component in F_i is at most $\lfloor n/2^i \rfloor$.
- 2 F is a maximum spanning forest of G .



Zoomed in: Invariants

- ① The number of vertices in every connected component in F_i is at most $\lfloor n/2^i \rfloor$.
 - level of an edge: $0, \dots, \lfloor \log_2 n \rfloor$
- ② F is a maximum spanning forest of G .
 - If a level- l tree edge is deleted, start searching for a replacement edge at level l .

Goal: Design operations in such a way that the invariants are maintained throughout the algorithm

Zoomed in: Invariants

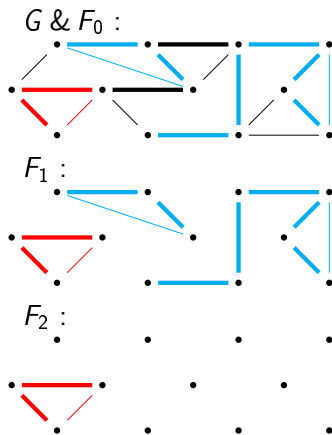
- ① The number of vertices in every connected component in F_i is at most $\lfloor n/2^i \rfloor$.
 - level of an edge: $0, \dots, \lfloor \log_2 n \rfloor$
- ② F is a maximum spanning forest of G .
 - If a level- l tree edge is deleted, start searching for a replacement edge at level l .

Goal: Design operations in such a way that the invariants are maintained throughout the algorithm

- queries: consult F_0
- insertions: new edges are level-0 edges
- deletions: ...

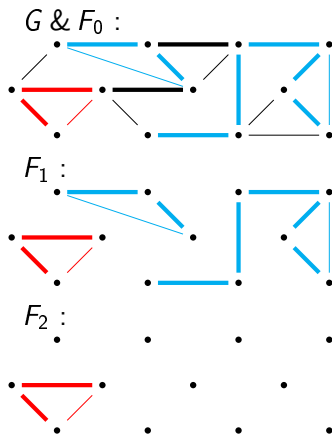
Deletions

```
delete( $e = \{v, w\}$ )  
  if( $e$  is a non-tree edge)  
    delete  $e$ ;  
  else
```



Deletions

```
delete( $e = \{v, w\}$ )  
  if( $e$  is a non-tree edge)  
    delete  $e$ ;  
  else  
     $l := \text{level}(e)$ ;  
    delete  $e$  from  $F_l, \dots, F_0$ ;
```

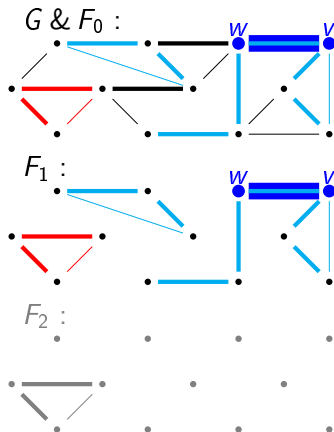


Deletions

```

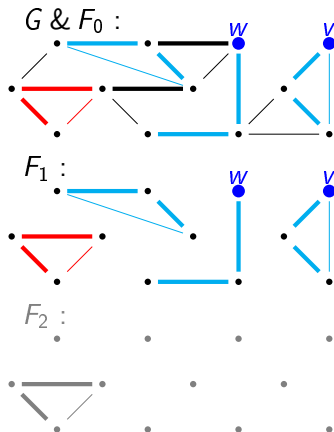
delete( $e = \{v, w\}$ )
  if( $e$  is a non-tree edge)
    delete  $e$ ;
  else
     $l := \text{level}(e)$ ;
    delete  $e$  from  $F_l, \dots, F_0$ ;

```



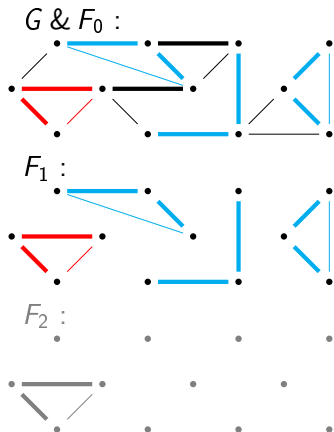
Deletions

```
delete( $e = \{v, w\}$ )  
  if( $e$  is a non-tree edge)  
    delete  $e$ ;  
  else  
     $l := \text{level}(e)$ ;  
    delete  $e$  from  $F_l, \dots, F_0$ ;
```



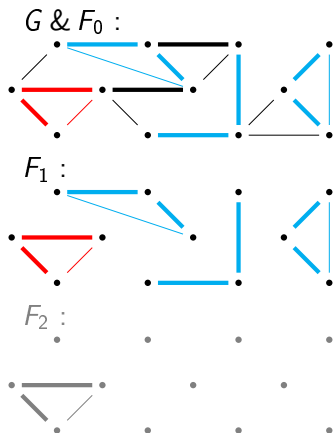
Deletions

```
delete( $e = \{v, w\}$ )  
  if( $e$  is a non-tree edge)  
    delete  $e$ ;  
  else  
     $l := \text{level}(e)$ ;  
    delete  $e$  from  $F_l, \dots, F_0$ ;  
    for  $i = l, \dots, 0$ 
```



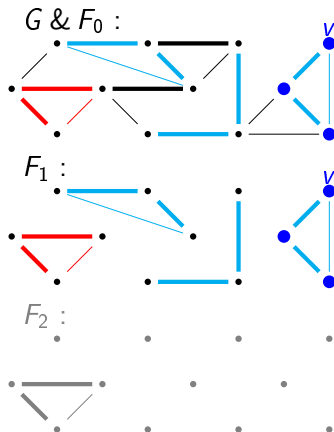
Deletions

```
delete( $e = \{v, w\}$ )  
  if( $e$  is a non-tree edge)  
    delete  $e$ ;  
  else  
     $l := \text{level}(e)$ ;  
    delete  $e$  from  $F_l, \dots, F_0$ ;  
    for  $i = l, \dots, 0$   
      push  $T_v$ 's edges to level  $i+1$ ;
```



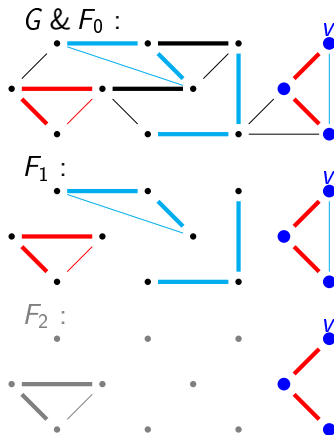
Deletions

```
delete( $e = \{v, w\}$ )  
  if( $e$  is a non-tree edge)  
    delete  $e$ ;  
  else  
     $l := \text{level}(e)$ ;  
    delete  $e$  from  $F_l, \dots, F_0$ ;  
    for  $i = l, \dots, 0$   
      push  $T_v$ 's edges to level  $i+1$ ;
```



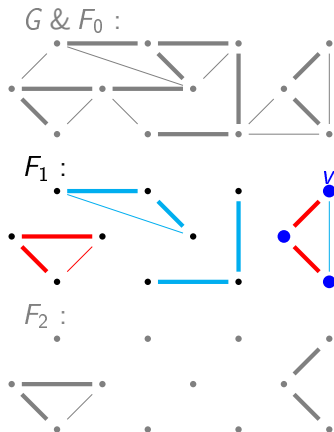
Deletions

```
delete( $e = \{v, w\}$ )  
  if( $e$  is a non-tree edge)  
    delete  $e$ ;  
  else  
     $l := \text{level}(e)$ ;  
    delete  $e$  from  $F_l, \dots, F_0$ ;  
    for  $i = l, \dots, 0$   
      push  $T_v$ 's edges to level  $i+1$ ;
```



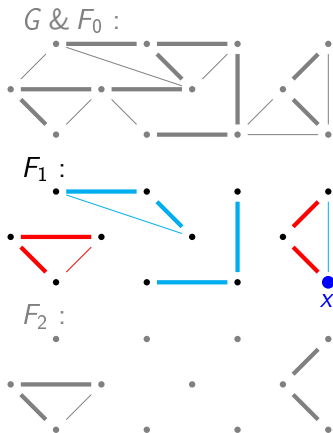
Deletions

```
delete( $e = \{v, w\}$ )  
  if( $e$  is a non-tree edge)  
    delete  $e$ ;  
  else  
     $l := \text{level}(e)$ ;  
    delete  $e$  from  $F_l, \dots, F_0$ ;  
    for  $i = l, \dots, 0$   
      push  $T_v$ 's edges to level  $i+1$ ;  
      for each level- $i$  edge  $\{x, y\}$   
        with  $x \in T_v$ 
```



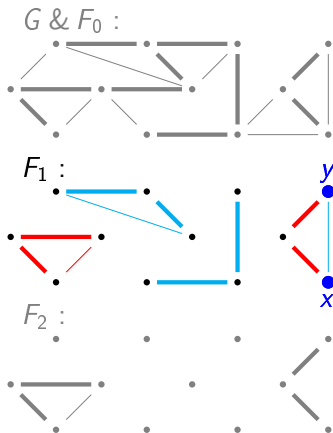
Deletions

```
delete( $e = \{v, w\}$ )  
  if( $e$  is a non-tree edge)  
    delete  $e$ ;  
  else  
     $l := \text{level}(e)$ ;  
    delete  $e$  from  $F_l, \dots, F_0$ ;  
    for  $i = l, \dots, 0$   
      push  $T_v$ 's edges to level  $i+1$ ;  
      for each level- $i$  edge  $\{x, y\}$   
        with  $x \in T_v$ 
```



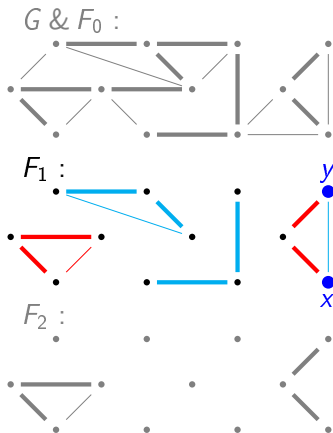
Deletions

```
delete( $e = \{v, w\}$ )  
  if( $e$  is a non-tree edge)  
    delete  $e$ ;  
  else  
     $l := \text{level}(e)$ ;  
    delete  $e$  from  $F_l, \dots, F_0$ ;  
    for  $i = l, \dots, 0$   
      push  $T_v$ 's edges to level  $i+1$ ;  
      for each level- $i$  edge  $\{x, y\}$   
        with  $x \in T_v$ 
```



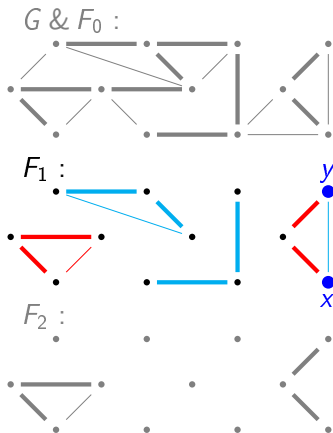
Deletions

```
delete( $e = \{v, w\}$ )  
  if( $e$  is a non-tree edge)  
    delete  $e$ ;  
  else  
     $l := \text{level}(e)$ ;  
    delete  $e$  from  $F_l, \dots, F_0$ ;  
    for  $i = l, \dots, 0$   
      push  $T_v$ 's edges to level  $i+1$ ;  
      for each level- $i$  edge  $\{x, y\}$   
        with  $x \in T_v$   
        if( $y \in T_v$ )  
          else
```



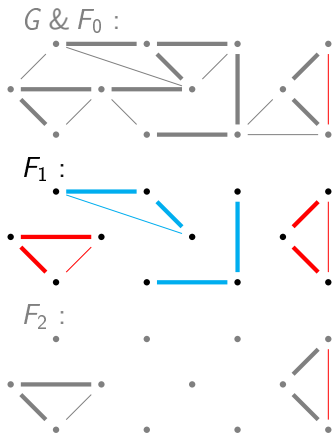
Deletions

```
delete( $e = \{v, w\}$ )  
  if( $e$  is a non-tree edge)  
    delete  $e$ ;  
  else  
     $l := \text{level}(e)$ ;  
    delete  $e$  from  $F_l, \dots, F_0$ ;  
    for  $i = l, \dots, 0$   
      push  $T_v$ 's edges to level  $i+1$ ;  
      for each level- $i$  edge  $\{x, y\}$   
        with  $x \in T_v$   
        if( $y \in T_v$ )  
           $\text{level}(\{x, y\}) := i+1$ ;  
        else
```



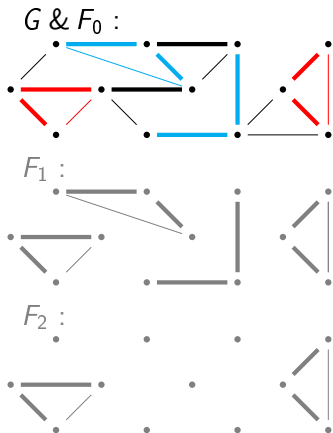
Deletions

```
delete( $e = \{v, w\}$ )  
  if( $e$  is a non-tree edge)  
    delete  $e$ ;  
  else  
     $l := \text{level}(e)$ ;  
    delete  $e$  from  $F_l, \dots, F_0$ ;  
    for  $i = l, \dots, 0$   
      push  $T_v$ 's edges to level  $i+1$ ;  
      for each level- $i$  edge  $\{x, y\}$   
        with  $x \in T_v$   
        if( $y \in T_v$ )  
           $\text{level}(\{x, y\}) := i+1$ ;  
        else
```



Deletions

```
delete( $e = \{v, w\}$ )  
  if( $e$  is a non-tree edge)  
    delete  $e$ ;  
  else  
     $l := \text{level}(e)$ ;  
    delete  $e$  from  $F_l, \dots, F_0$ ;  
    for  $i = l, \dots, 0$   
      push  $T_v$ 's edges to level  $i+1$ ;  
      for each level- $i$  edge  $\{x, y\}$   
        with  $x \in T_v$   
        if( $y \in T_v$ )  
           $\text{level}(\{x, y\}) := i+1$ ;  
        else
```



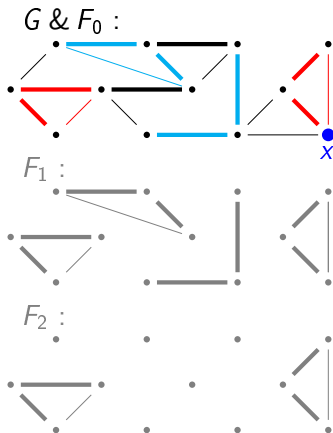
Deletions

```

delete( $e = \{v, w\}$ )
  if( $e$  is a non-tree edge)
    delete  $e$ ;
  else
     $l := \text{level}(e)$ ;
    delete  $e$  from  $F_l, \dots, F_0$ ;
    for  $i = l, \dots, 0$ 
      push  $T_v$ 's edges to level  $i+1$ ;
      for each level- $i$  edge  $\{x, y\}$ 
        with  $x \in T_v$ 

        if( $y \in T_v$ )
           $\text{level}(\{x, y\}) := i+1$ ;
        else

```



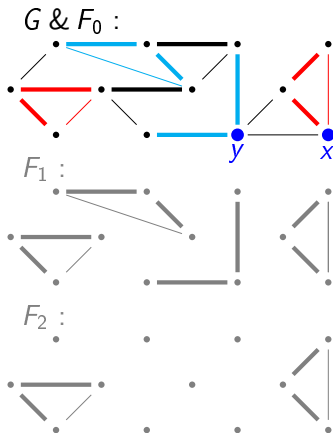
Deletions

```

delete( $e = \{v, w\}$ )
  if( $e$  is a non-tree edge)
    delete  $e$ ;
  else
     $l := \text{level}(e)$ ;
    delete  $e$  from  $F_l, \dots, F_0$ ;
    for  $i = l, \dots, 0$ 
      push  $T_v$ 's edges to level  $i+1$ ;
      for each level- $i$  edge  $\{x, y\}$ 
        with  $x \in T_v$ 

        if( $y \in T_v$ )
           $\text{level}(\{x, y\}) := i+1$ ;
        else

```

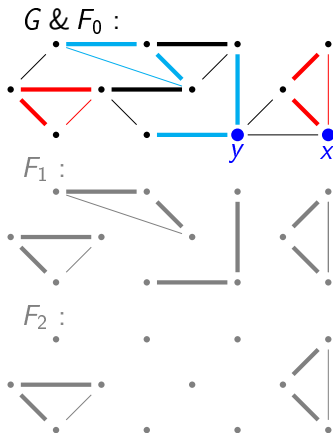


Deletions

```

delete( $e = \{v, w\}$ )
  if( $e$  is a non-tree edge)
    delete  $e$ ;
  else
     $l := \text{level}(e)$ ;
    delete  $e$  from  $F_l, \dots, F_0$ ;
    for  $i = l, \dots, 0$ 
      push  $T_v$ 's edges to level  $i+1$ ;
      for each level- $i$  edge  $\{x, y\}$ 
        with  $x \in T_v$ 

        if( $y \in T_v$ )
           $\text{level}(\{x, y\}) := i+1$ ;
        else
          add  $\{x, y\}$  to  $F_i, \dots, F_0$ ;
          stop;
  
```

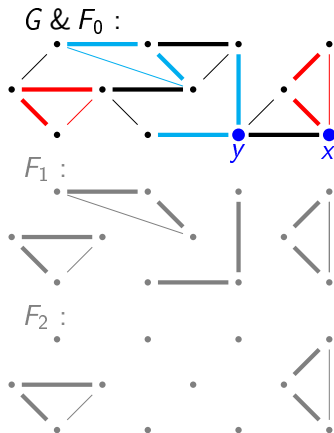


Deletions

```

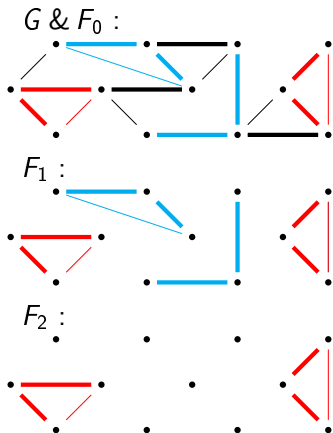
delete( $e = \{v, w\}$ )
  if( $e$  is a non-tree edge)
    delete  $e$ ;
  else
     $l := \text{level}(e)$ ;
    delete  $e$  from  $F_l, \dots, F_0$ ;
    for  $i = l, \dots, 0$ 
      push  $T_v$ 's edges to level  $i+1$ ;
      for each level- $i$  edge  $\{x, y\}$ 
        with  $x \in T_v$ 

        if( $y \in T_v$ )
           $\text{level}(\{x, y\}) := i+1$ ;
        else
          add  $\{x, y\}$  to  $F_i, \dots, F_0$ ;
          stop;
  
```



Deletions

```
delete( $e = \{v, w\}$ )  
  if( $e$  is a non-tree edge)  
    delete  $e$ ;  
  else  
     $l := \text{level}(e)$ ;  
    delete  $e$  from  $F_l, \dots, F_0$ ;  
    for  $i = l, \dots, 0$   
      push  $T_v$ 's edges to level  $i+1$ ;  
      for each level- $i$  edge  $\{x, y\}$   
        with  $x \in T_v$   
        if( $y \in T_v$ )  
           $\text{level}(\{x, y\}) := i+1$ ;  
        else  
          add  $\{x, y\}$  to  $F_i, \dots, F_0$ ;  
        stop;
```



Summary

Dynamic graph component tracking with

- Connectivity queries in sublogarithmic time
- Insertions in polylogarithmic time
- Deletions in polylogarithmic amortised time

Key ideas

- Reduction of the search space through hierarchical decomposition
- Search strategy for replacement edges due to *maximum* spanning forests