Papers We Love Madrid

Diff & Blame

Alberto Cortés <alcortesm@gmail.com>
Madrid, 2015-12-02

Presentation

- Alberto Cortés
  - Former University teacher
  - Current Software Developer
- Source{d}
  - Recruitment company
    - “Tech Recruitment Done Right”
    - We analyze public open-source repositories to match awesome developers with awesome job offers

Blame?

Alice

file.txt

1	de
2	es
3	mx
4	fr
5	eu
6	us

Blame?

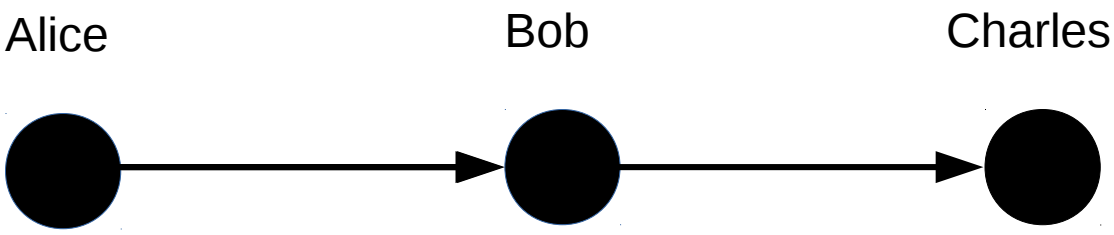
file.txt

1	de
2	es
3	mx
4	fr
5	eu
6	us

file.txt

1	it
2	ho
3	de
4	es
5	mx
6	us

Blame?



file.txt

1	de
2	es
3	mx
4	fr
5	eu
6	us

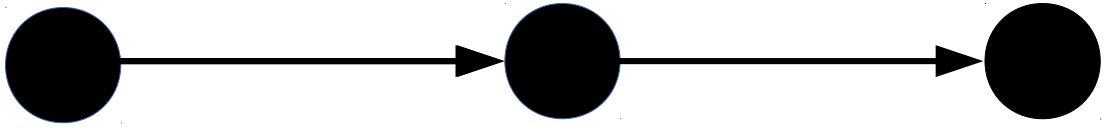
file.txt

1	it
2	ho
3	de
4	es
5	mx
6	us

file.txt

1	it
2	ho
3	de
4	us
5	ch
6	fi
7	ru

Blame?



file.txt

1	de
2	es
3	mx
4	fr
5	eu
6	us

file.txt

1	it
2	ho
3	de
4	es
5	mx
6	us

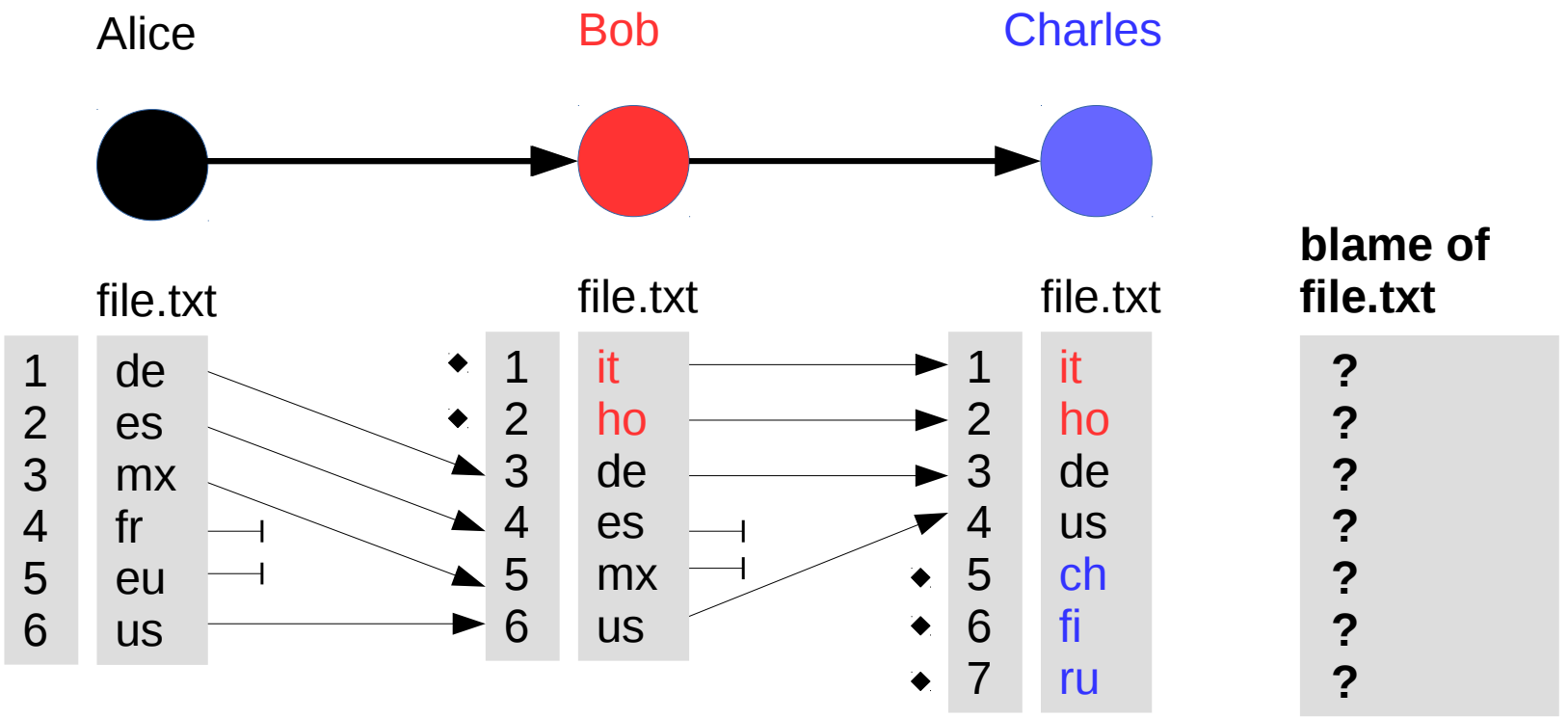
file.txt

1	it
2	ho
3	de
4	us
5	ch
6	fi
7	ru

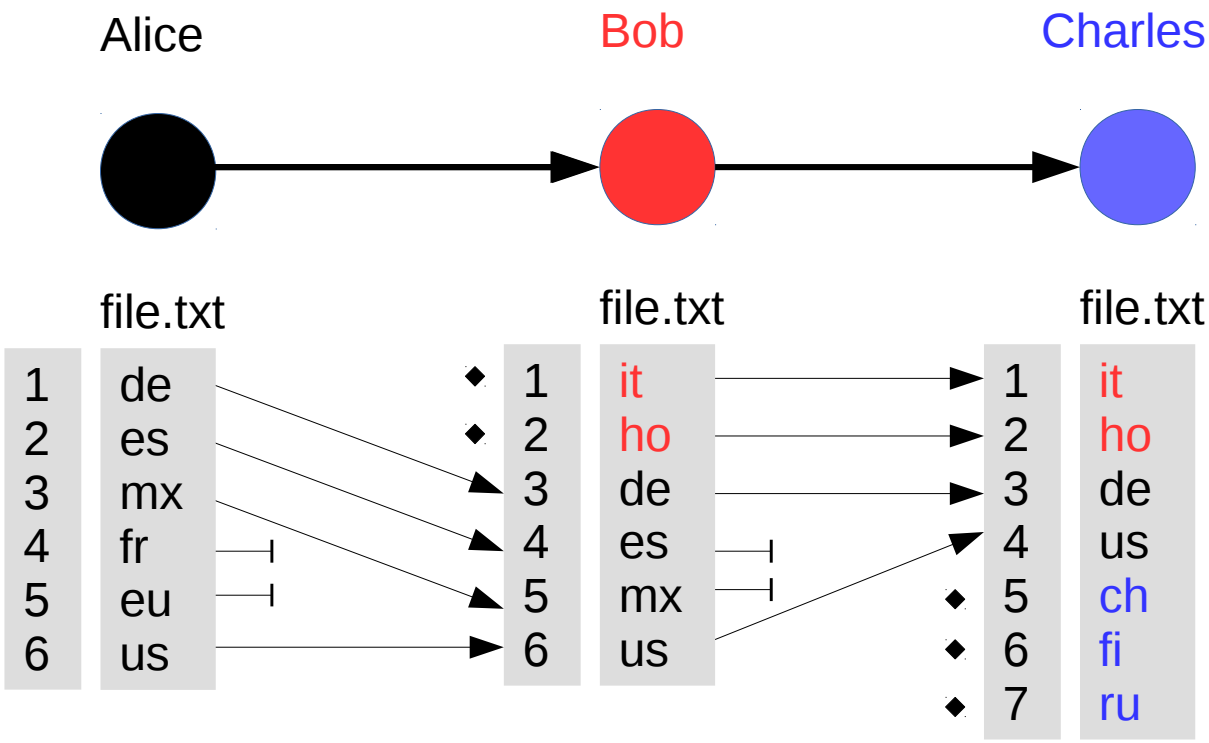
blame of file.txt

?	
?	
?	
?	
?	
?	
?	

Blame?


Alice
Bob
Charles
file.txt
file.txt
file.txt
blame of file.txt
1 2 3 4 5 6
de es mx fr eu us
1 2 3 4 5 6
it ho de es mx us
1 2 3 4 5 6 7
it ho de us ch fi ru
? ? ? ? ? ? ?

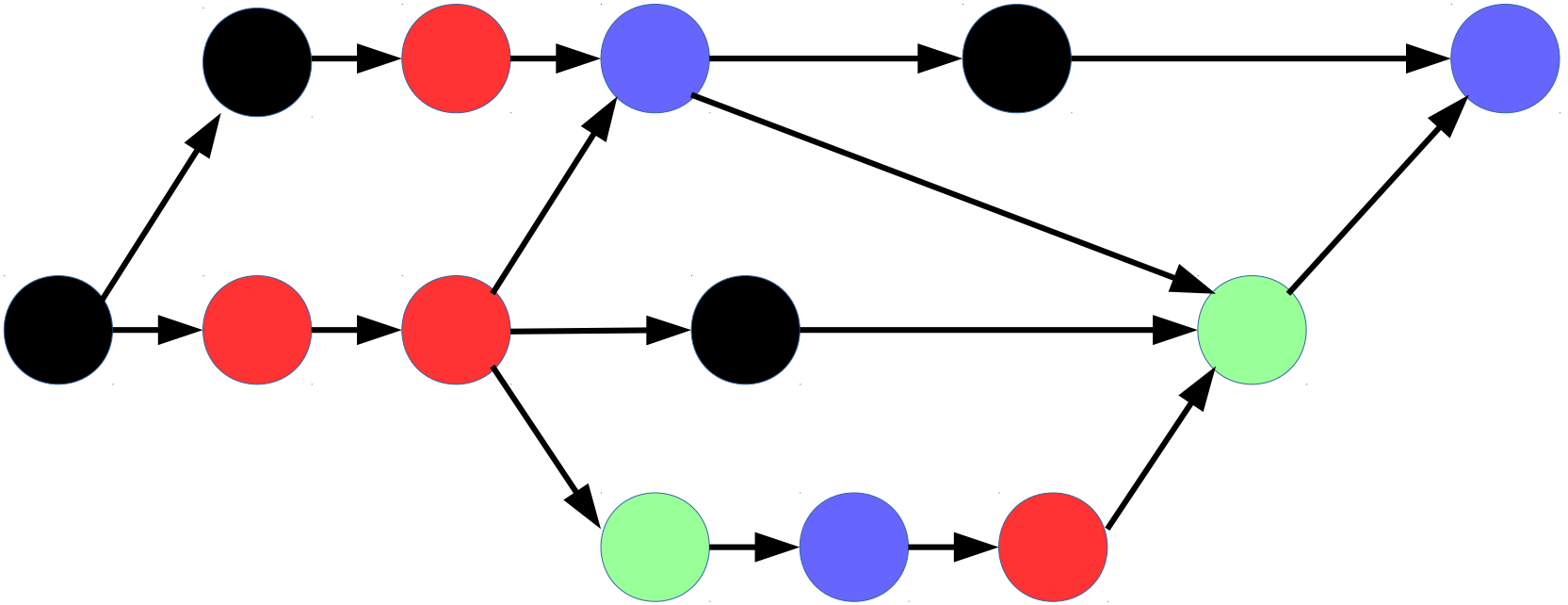
Blame?



blame of file.txt

Bob
Bob
Alice
Alice
Charles
Charles
Charles

Blame?



Goals

-
-
-
- Understand the ***blaming*** algorithm used in modern revision control systems (e.g. Git) (2006)

Goals

-
-

- Understand the ***diff*** algorithm (1976)
- Understand the ***blaming*** algorithm used in modern revision control systems (e.g. Git) (2006)

Goals

-
- Understand the ***Longest common subsequence problem*** (?)
- Understand the ***diff*** algorithm (1976)
- Understand the ***blaming*** algorithm used in modern revision control systems (e.g. Git) (2006)

Goals

- Understand ***Levenshtein distance*** (1965)
- Understand the ***Longest common subsequence problem*** (?)
- Understand the ***diff*** algorithm (1976)
- Understand the ***blaming*** algorithm used in modern revision control systems (e.g. Git) (2006)

Levenshtein distance

Levenshtein distance

- A measure of the difference between two strings

  minimum number of edits needed to transform one string into the other

  - Edits are (one char) insertions, deletions or substitutions

Levenshtein distance

- A measure of the difference between two strings

  minimum number of edits needed to transform one string into the other

  - Edits are (one char) insertions, deletions or substitutions

```
   Insertion

s0 = ”pain”
s1 = ”plain”

(Insert l at 1)
```

Levenshtein distance

- A measure of the difference between two strings

  minimum number of edits needed to transform one string into the other

  - Edits are (one char) insertions, deletions or substitutions

```
   Insertion

s0 = ”pain”
s1 = ”plain”

(Insert l at 1)
```

```
   Deletion

s0 = ”pain”
s2 = ”pan”

  (Delete 2)
```

Levenshtein distance

- A measure of the difference between two strings

  minimum number of edits needed to transform one string into the other

  - Edits are (one char) insertions, deletions or substitutions

```
   Insertion

s0 = ”pain”
s1 = ”plain”

(Insert l at 1)
```

```
   Deletion

s0 = ”pain”
s2 = ”pan”

  (Delete 2)
```

```
 Substitution

s0 = ”pain”
s3 = ”pawn”

 (subs. w at 2)
```

Levenshtein distance

```
s0 = ”Lost”
s1 = ”plot”
```

Levenshtein distance

```
delete all s0, insert all s1:
    # edits: 8
```

```
s0 = "Lost"
s1 = "plot"
```

Levenshtein distance

```
delete all s0, insert all s1:
    # edits: 8
```

```
s0 = "Lost"
s1 = "plot"
```

```
delete until "t", insert "plo":
    # edits: 6
```

Levenshtein distance

```
s0 = "Lost"
s1 = "plot"
```

```
delete all s0, insert all s1:
    # edits: 8

delete until "t", insert "plo":
    # edits: 6

insert p, subs. L, delete s:
    # edits: 3
```

Levenshtein distance

```
s0 = ”Hello World”
s1 = ”hello World”
```

$lev_{s0,s1} = ?$

Levenshtein distance

```
s0 = "Hello World"
s1 = "hello World"
```

$lev_{s0,s1} = 1$ (subs. h at 0)

Levenshtein distance

```
s0 = ”Hello World”
s1 = ”hello World”
```

$lev_{s0,s1} = 1$ (subs. h at 0)

$lev_{s0,s0} = ?$

Levenshtein distance

s0 = ”Hello World”
s1 = ”hello World”

$lev_{s0,s1}$ = 1 (subs. h at 0)

$lev_{s0,s0}$ = 0 (do nothing)

Levenshtein distance

s0 = ”Hello World”
s1 = ”hello World”

s2 = ””
s3 = ”sitting”

$lev_{s0,s1}$ = 1 (subs. H at 0)

$lev_{s0,s0}$ = 0 (do nothing)

$lev_{s2,s3}$ = ?

Levenshtein distance

```
s0 = "Hello World"
s1 = "hello World"


s2 = ""
s3 = "sitting"
```

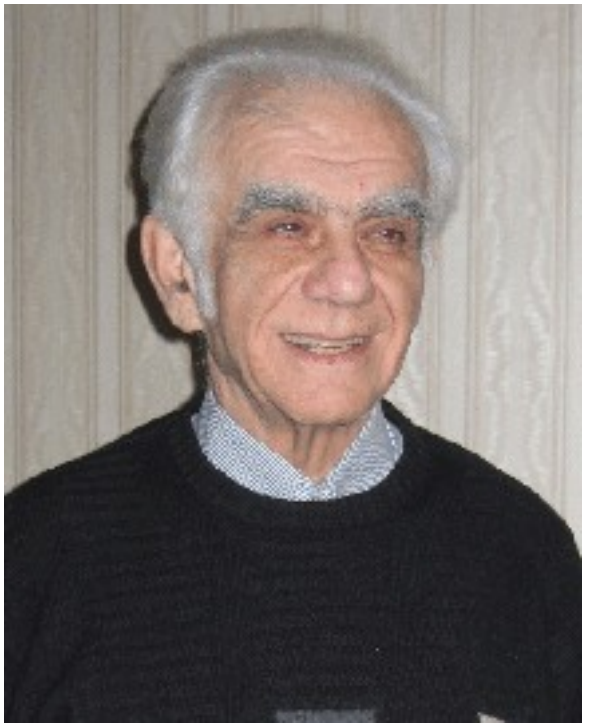
$lev_{s0,s1} = 1$ (subs. H at 0)

$lev_{s0,s0} = 0$ (do nothing)

$lev_{s2,s3} = 7$ (insert all)

Levenshtein distance

- Vladimir Levenshtein (*1935):
  - IEEE Hamming medal 2006
  - “*contributions to the theory of error-correcting codes and information theory, including the Levenshtein distance*”
- Applications:
  - Spell checkers
  - OCR correction
  - Fuzzy string searching
  - DNA sequence alignment
  - String comparison



Влади́мир Ио́сифович Левенште́йн
(photo courtesy: IEEE)

Levenshtein distance

- Instead of calculating `lev` of `a` and `b`
- Let's calculate `lev` of `a` up to char `i` and `b` up to char `j`:

```
a = ”kitten”    len(a) = 6
b = ”sitting”   len(b) = 7
```

$lev_{a,b}(6,7)$ = lev of ”kitten” and ”sitting”
$lev_{a,b}(3,5)$ = lev of ”kitten” and ”sitting”
$lev_{a,b}(0,2)$ = lev of ”kitten” and ”sitting”
$lev_{a,b}(0,0)$ = lev of ”kitten” and ”sitting”

Levenshtein distance

a = ”kitten” i=0
b = ”sitting” j=4

$lev_{a,b}(0,j) = j$

Levenshtein distance

a = ”kitten” i=3
b = ”sitting” j=0

$lev_{a,b}(0,j) = j$
$lev_{a,b}(i,0) = i$

Levenshtein distance

a = ”kitten” i=0
b = ”sitting” j=0

$$lev_{a,b}(0,j) = j$$
$$lev_{a,b}(i,0) = i$$
$$lev_{a,b}(0,0) = 0$$

Levenshtein distance

$$lev_{a,b}(0,j) = j$$
$$lev_{a,b}(i,0) = i$$
$$lev_{a,b}(0,0) = 0$$

$$\text{if } \min(i,j) = 0 \text{ then } lev_{a,b}(i,j) = \max(i, j)$$

Levenshtein distance

$$lev_{a,b}(0,j) = j$$
$$lev_{a,b}(i,0) = i$$
$$lev_{a,b}(0,0) = 0$$

if $\min(i,j) = 0$
then $lev_{a,b}(i,j) = \max(i, j)$

$$lev_{a,b}(i,j) = \qquad)$$

Levenshtein distance

a = ”Lost” i=2
b = ”plot” j=3

$lev_{a,b}(0,j) = j$
$lev_{a,b}(i,0) = i$
$lev_{a,b}(0,0) = 0$

if $\min(i,j) = 0$
then $lev_{a,b}(i,j) = \max(i, j)$

if $a_i == b_j$ (do nothing)

$lev_{a,b}(i,j) =$

Levenshtein distance

a = ”Lost” i=2
b = ”plot” j=3

$lev_{a,b}(0,j) = j$
$lev_{a,b}(i,0) = i$
$lev_{a,b}(0,0) = 0$

if $\min(i,j) = 0$
then $lev_{a,b}(i,j) = \max(i, j)$

$lev_{a,b}(i-1,j-1)$ if $a_i == b_j$ (do nothing)

$lev_{a,b}(i,j) =$

Levenshtein distance

a = ”kitten” i=5
b = ”sitting” j=5

$lev_{a,b}(0,j) = j$
$lev_{a,b}(i,0) = i$
$lev_{a,b}(0,0) = 0$

if min(i,j) = 0
then $lev_{a,b}(i,j) = max(i, j)$

$lev_{a,b}(i,j) =$

$lev_{a,b}(i-1,j-1)$ if $a_i == b_j$ (do nothing)

if $a_i != b_j$ (substitution)

Levenshtein distance

a = ”kitten” i=5
b = ”sitting” j=5

$lev_{a,b}(0,j) = j$
$lev_{a,b}(i,0) = i$
$lev_{a,b}(0,0) = 0$

if min(i,j) = 0
then $lev_{a,b}(i,j) = max(i, j)$

$$lev_{a,b}(i,j) = \begin{cases} lev_{a,b}(i-1,j-1) & \text{if } a_i == b_j \quad \text{(do nothing)} \\ 1 + lev_{a,b}(i-1,j-1) & \text{if } a_i != b_j \quad \text{(substitution)} \end{cases}$$

Levenshtein distance

a = ”kitten” i=6
b = ”sitting” j=7

$lev_{a,b}(0,j) = j$
$lev_{a,b}(i,0) = i$
$lev_{a,b}(0,0) = 0$

if min(i,j) = 0
then $lev_{a,b}(i,j) = max(i, j)$

$$lev_{a,b}(i,j) = \begin{cases} lev_{a,b}(i-1,j-1) & \text{if } a_i == b_j \quad \text{(do nothing)} \\ 1 + lev_{a,b}(i-1,j-1) & \text{if } a_i != b_j \quad \text{(substitution)} \\ 1 + lev_{a,b}(i,j-1) & \text{(insertion)} \end{cases}$$

Levenshtein distance

a = ”sitting” i=7
b = ”kitten” j=6

$lev_{a,b}(0,j) = j$
$lev_{a,b}(i,0) = i$
$lev_{a,b}(0,0) = 0$

if min(i,j) = 0
then $lev_{a,b}(i,j) = max(i, j)$

$$lev_{a,b}(i,j) = \begin{cases} lev_{a,b}(i-1,j-1) & \text{if } a_i == b_j \quad \text{(do nothing)} \\ 1 + lev_{a,b}(i-1,j-1) & \text{if } a_i != b_j \quad \text{(substitution)} \\ 1 + lev_{a,b}(i,j-1) & \text{(insertion)} \\ 1 + lev_{a,b}(i-1,j) & \text{(deletion)} \end{cases}$$

Levenshtein distance

a = ”kitten” len(a) = 6
b = ”sitting” len(b) = 7

$$lev_{a,b}(0,j) = j$$
$$lev_{a,b}(i,0) = i$$
$$lev_{a,b}(0,0) = 0$$

if $\min(i,j) = 0$ then $lev_{a,b}(i,j) = \max(i, j)$

$$lev_{a,b}(i,j) = \begin{cases} lev_{a,b}(i-1,j-1) & \text{if } a_i == b_j \quad \text{(do nothing)} \\ 1 + lev_{a,b}(i-1,j-1) & \text{if } a_i != b_j \quad \text{(substitution)} \\ 1 + lev_{a,b}(i,j-1) & \text{(insertion)} \\ 1 + lev_{a,b}(i-1,j) & \text{(deletion)} \end{cases}$$

Levenshtein distance

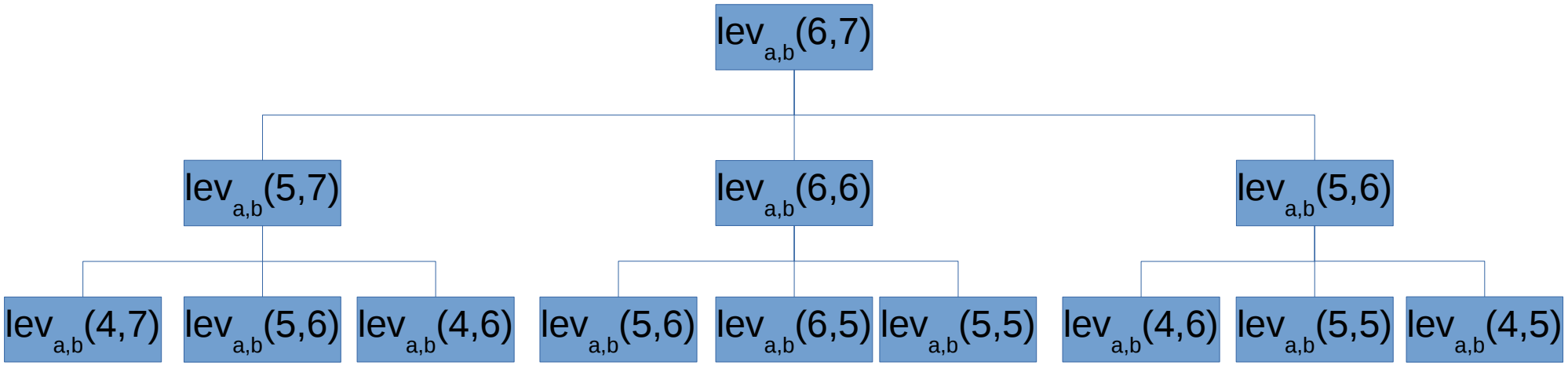
$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

- $1_{(a_i \neq b_j)}$ is 1 if character i from a is different than character j from b, and 0 otherwise.
- Min cases: deletion, insertion, substitution

Levenshtein distance

- Recursive implementation:
  - Straightforward:

```
a = ”kitten”      b = ”sitting”
len(a) = 6        len(b) = 7
```

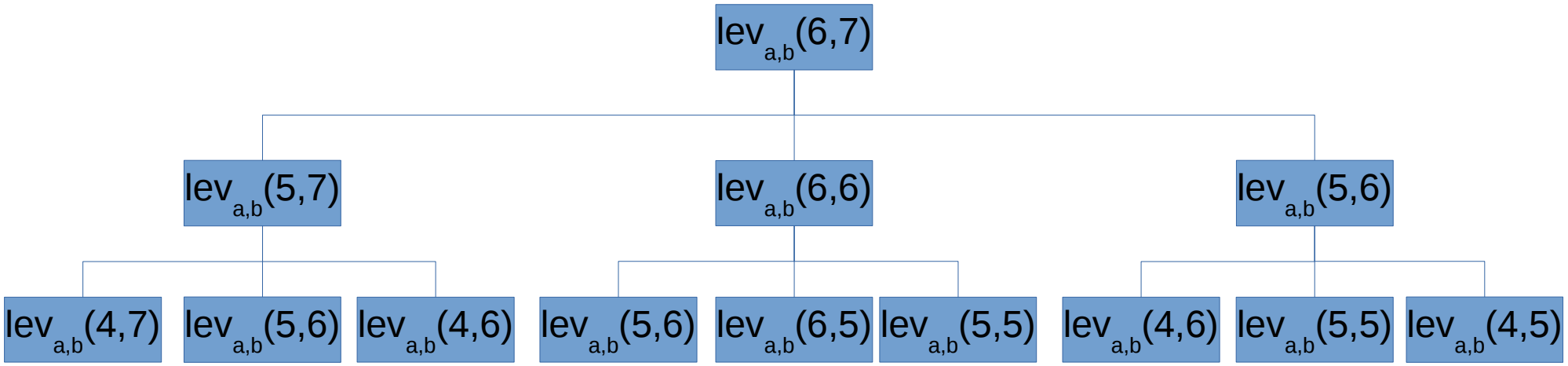


  - About 700 calls ($\sim 3^{\min(\text{len}(a), \text{len}(b))}$)

Levenshtein distance

- Recursive implementation:
  - Straightforward:

```
a = ”kitten”      b = ”sitting”
len(a) = 6        len(b) = 7
```

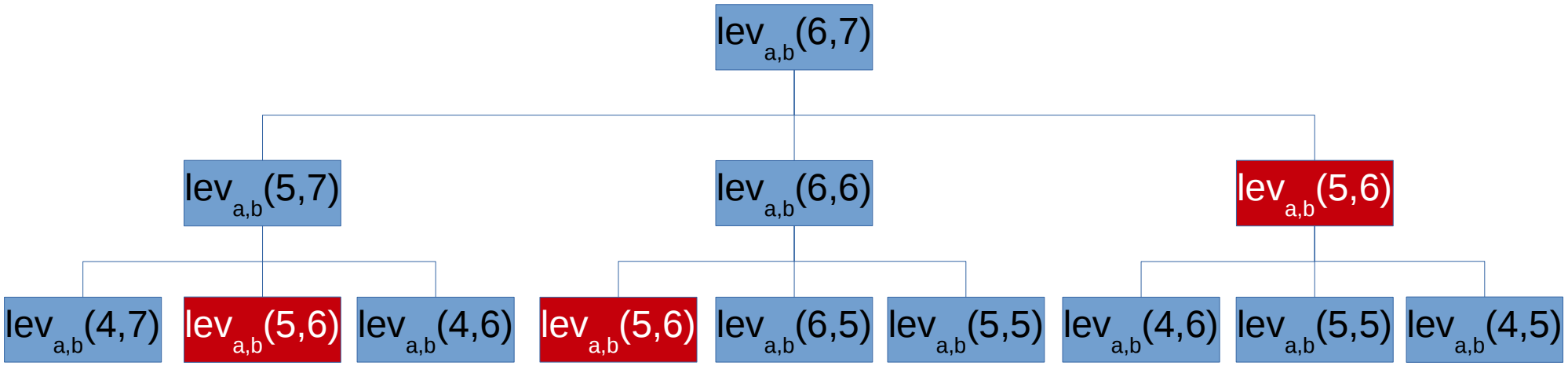


  - About 700 calls ($\sim 3^{\min(\text{len}(a), \text{len}(b))}$)
  - **Don't panic**, we only need 42 calls

Levenshtein distance

- Recursive implementation:
  - Straightforward:

```
a = ”kitten”      b = ”sitting”
len(a) = 6        len(b) = 7
```

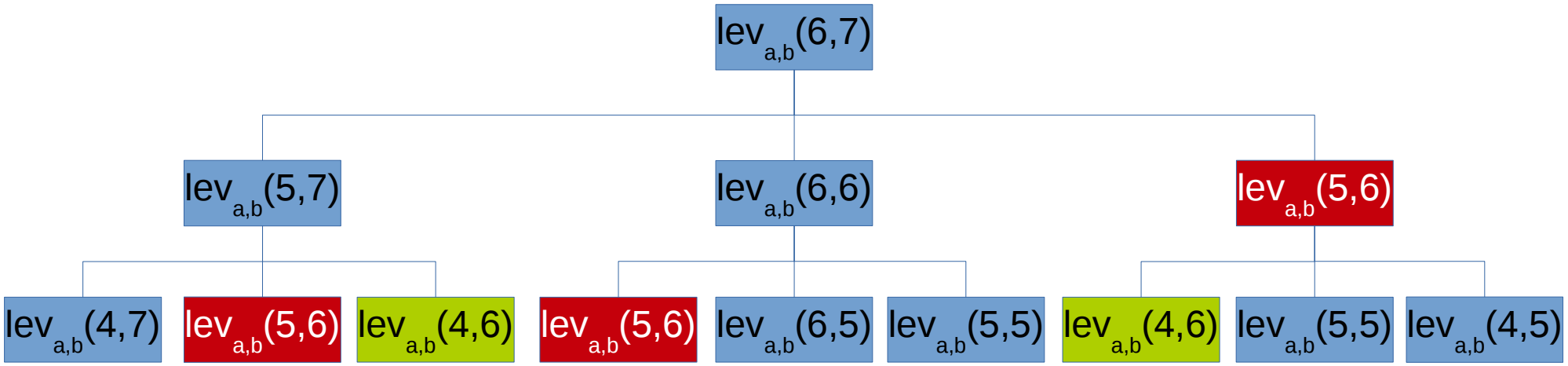


  - About 700 calls ($\sim 3^{\min(\text{len}(a), \text{len}(b))}$)
  - **Don't panic**, we only need 42 calls

Levenshtein distance

- Recursive implementation:
  - Straightforward:

```
a = ”kitten”      b = ”sitting”
len(a) = 6        len(b) = 7
```

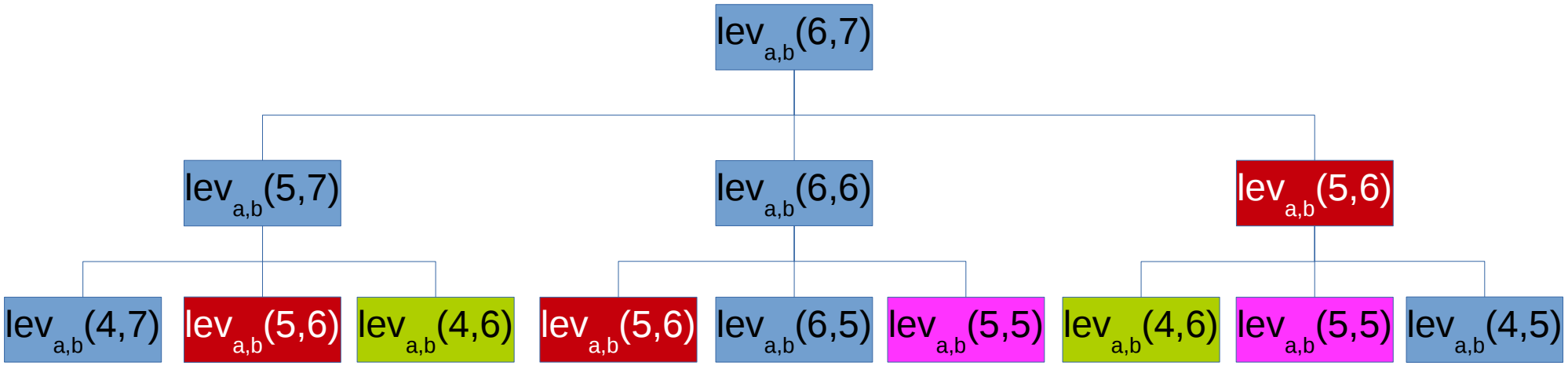


  - About 700 calls ($\sim 3^{\min(\text{len}(a), \text{len}(b))}$)
  - **Don't panic**, we only need 42 calls

Levenshtein distance

- Recursive implementation:
  - Straightforward:

```
a = ”kitten”      b = ”sitting”
len(a) = 6        len(b) = 7
```



  - About 700 calls ($\sim 3^{\min(\text{len}(a), \text{len}(b))}$)
  - **Don't panic**, we only need 42 calls: **O(len(a)len(b))**

Levenshtein distance

- Dynamic programing version:
  - Bottom-up approach
  - Much more fun
  - Example:

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		lev(0,0)	lev(0,1)	lev(0,2)	lev(0,3)	lev(0,4)
1	p	lev(1,0)	lev(1,1)	lev(1,2)	lev(1,3)	lev(1,4)
2	l	lev(2,0)	lev(2,1)	lev(2,2)	lev(2,3)	lev(2,4)
3	o	lev(3,0)	lev(3,1)	lev(3,2)	lev(3,3)	lev(3,4)
4	t	lev(4,0)	lev(4,1)	lev(4,2)	lev(4,3)	lev(4,4)

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0						
1	p					
2	l					
3	o					
4	t					?

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j) + 1 \\ \mathrm{lev}_{a,b}(i,j-1) + 1 \\ \mathrm{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		0				
1	p					
2	l					
3	o					
4	t					?

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j) + 1 \\ \mathrm{lev}_{a,b}(i,j-1) + 1 \\ \mathrm{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		0	1			
1	p					
2	l					
3	o					
4	t					?

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j) + 1 \\ \mathrm{lev}_{a,b}(i,j-1) + 1 \\ \mathrm{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		0	1	2		
1	p					
2	l					
3	o					
4	t					?

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j) + 1 \\ \mathrm{lev}_{a,b}(i,j-1) + 1 \\ \mathrm{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		0	1	2	3	4
1	p					
2	l					
3	o					
4	t					?

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j) + 1 \\ \mathrm{lev}_{a,b}(i,j-1) + 1 \\ \mathrm{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		0	1	2	3	4
1	p	1				
2	l	2				
3	o	3				
4	t	4				?

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j) + 1 \\ \mathrm{lev}_{a,b}(i,j-1) + 1 \\ \mathrm{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		0	1	2	3	4
1	p	1	min(1+1,1 +1, 0+1)			
2	l	2				
3	o	3				
4	t	4				

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		0	1	2	3	4
1	p	1	1			
2	l	2				
3	o	3				
4	t	4				

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		0	1	2	3	4
1	p	1	1	min(2+1,1 +1, 1+1)		
2	l	2				
3	o	3				
4	t	4				

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		0	1	2	3	4
1	p	1	1	2		
2	l	2				
3	o	3				
4	t	4				

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		0	1	2	3	4
1	p	1	1	2	3	
2	l	2				
3	o	3				
4	t	4				

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		0	1	2	3	4
1	p	1	1	2	3	4
2	l	2				
3	o	3				
4	t	4				

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		0	1	2	3	4
1	p	1	1	2	3	4
2	l	2	2			
3	o	3				
4	t	4				

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		0	1	2	3	4
1	p	1	1	2	3	4
2	l	2	2	2		
3	o	3				
4	t	4				

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		0	1	2	3	4
1	p	1	1	2	3	4
2	l	2	2	2	3	
3	o	3				
4	t	4				

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		0	1	2	3	4
1	p	1	1	2	3	4
2	l	2	2	2	3	4
3	o	3				
4	t	4				

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		0	1	2	3	4
1	p	1	1	2	3	4
2	l	2	2	2	3	4
3	o	3	3			
4	t	4				

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		0	1	2	3	4
1	p	1	1	2	3	4
2	l	2	2	2	3	4
3	o	3	3	**2**		
4	t	4				

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		0	1	2	3	4
1	p	1	1	2	3	4
2	l	2	2	2	3	4
3	o	3	3	2	3	
4	t	4				

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		0	1	2	3	4
1	p	1	1	2	3	4
2	l	2	2	2	3	4
3	o	3	3	2	3	4
4	t	4				

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		0	1	2	3	4
1	p	1	1	2	3	4
2	l	2	2	2	3	4
3	o	3	3	2	3	4
4	t	4	4			

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		0	1	2	3	4
1	p	1	1	2	3	4
2	l	2	2	2	3	4
3	o	3	3	2	3	4
4	t	4	4	3		

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		0	1	2	3	4
1	p	1	1	2	3	4
2	l	2	2	2	3	4
3	o	3	3	2	3	4
4	t	4	4	3	3	

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

	j=	0	1	2	3	4
i =			L	o	s	t
0		0	1	2	3	4
1	p	1	1	2	3	4
2	l	2	2	2	3	4
3	o	3	3	2	3	4
4	t	4	4	3	3	3

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Levenshtein distance

- Conclusions:
  - The Levenshtein distance between “Lost” and “plot” is 3
    - This means you can turn one into the other in only 3 edits
    - An edit is a single character insertion, deletion or substitution

What we don't know is what are those edits!

The longest common subsequence problem (LCS problem)

LCS

Find the longest substring common to all strings in a set (often only 2)

```
s0 = "pain"
s1 = "pains"

LCS0,1 = ?
```

LCS

Find the longest substring common to all strings in a set (often only 2)

```
s0 = "pain"
s1 = "pains"

LCS0,1 = "pain"
```

LCS

Find the longest substring common to all strings in a set (often only 2)

```
s0 = "pain"
s1 = "pains"

LCS0,1 = "pain"
```

```
s0 = "pain"
s2 = "plans"

LCS0,2 = ?
```

LCS

Find the longest substring common to all strings in a set (often only 2)

```
s0 = "pain"
s1 = "pains"

LCS0,1 = "pain"
```

```
s2 = "pain"
s3 = "plans"

LCS0,2 = "pan"
```

LCS

Find the longest substring common to all strings in a set (often only 2)

```
s0 = "pain"
s1 = "pains"

LCS0,1 = "pain"
```

```
s2 = "pain"
s3 = "plans"

LCS0,2 = "pan"
```

```
s4 = "AAACCGTGAGTTATTCGTTCTAGAA"
s5 = "CACCCCTAAGGTACCTTTGGTTC"

LCS4,5 = ?
```

thanks to columbia.edu for this example

LCS

Find the longest substring common to all strings in a set (often only 2)

```
s0 = "pain"
s1 = "pains"

LCS0,1 = "pain"
```

```
s2 = "pain"
s3 = "plans"

LCS0,2 = "pan"
```

```
s4 = "AAACCGTGAGTTATTCGTTCTAGAA"
s5 = "CACCCCTAAGGTACCTTTGGTTC"

LCS4,5 = "ACCTAGTACTTTG"
```

thanks to columbia.edu for this example

LCS

- Why is this interesting?
  - It turns out that finding the edits to turn one string into another is really easy if you know their LCS:

pain → plans

LCS

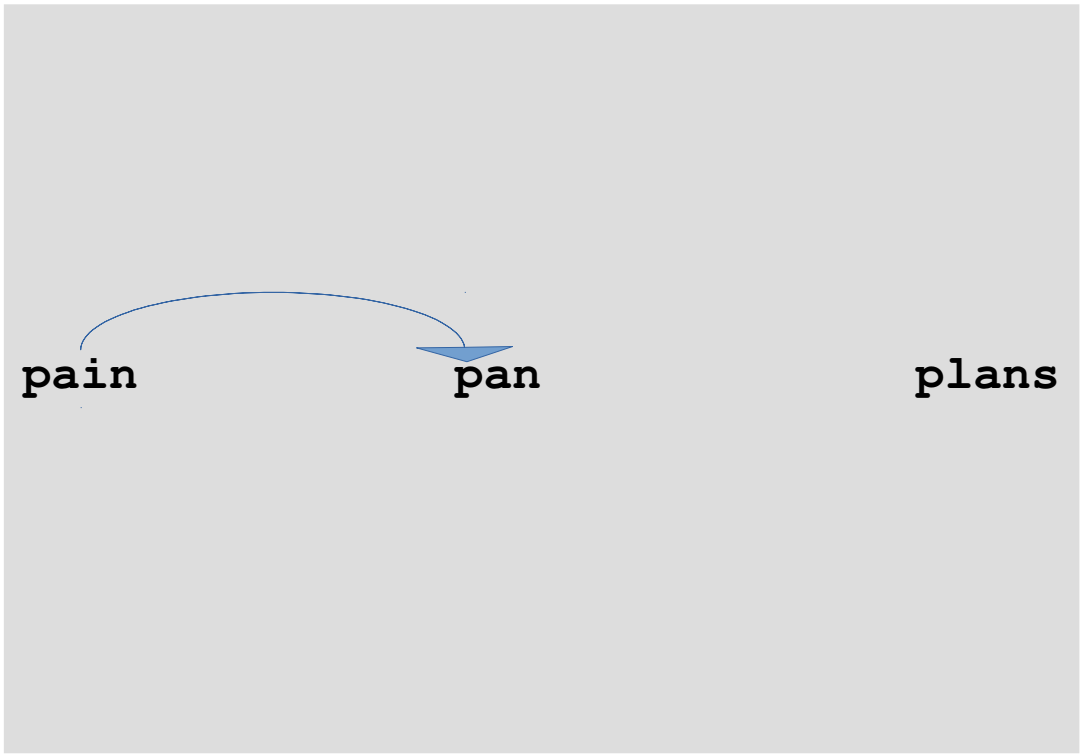
- Why is this interesting?
  - It turns out that finding the edits to turn one string into another is really easy if you know their LCS:

pain **pan** **plans**

pain → plans

LCS

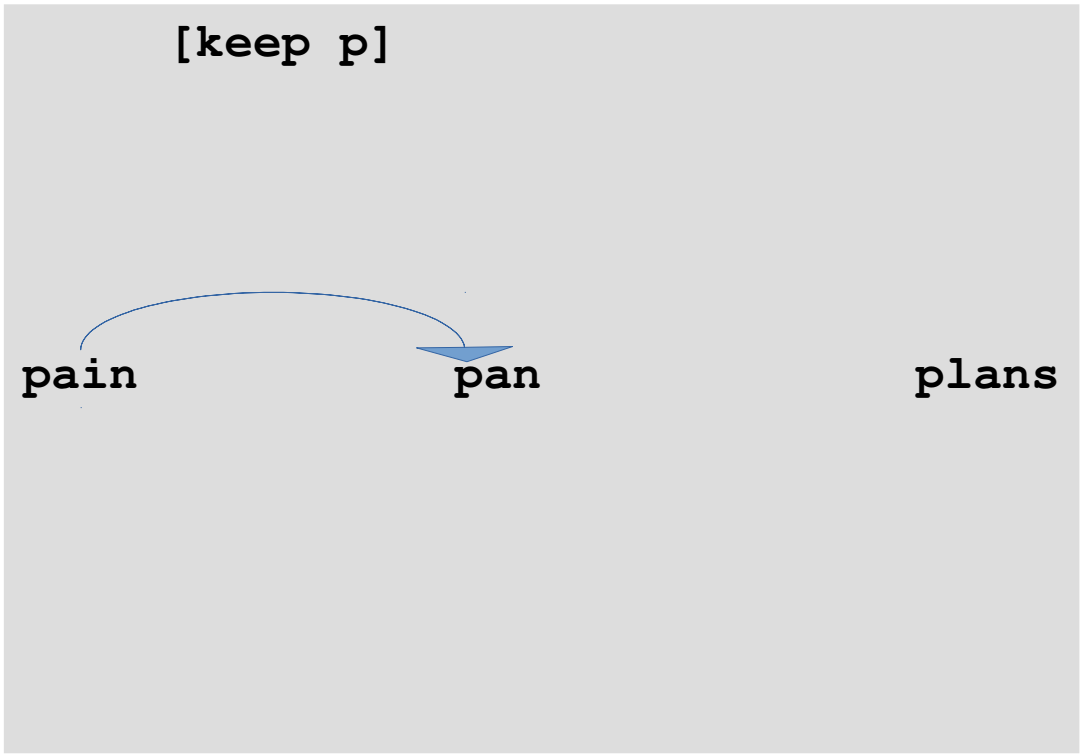
- Why is this interesting?
  - It turns out that finding the edits to turn one string into another is really easy if you know their LCS:




```
pain → plans
```

LCS

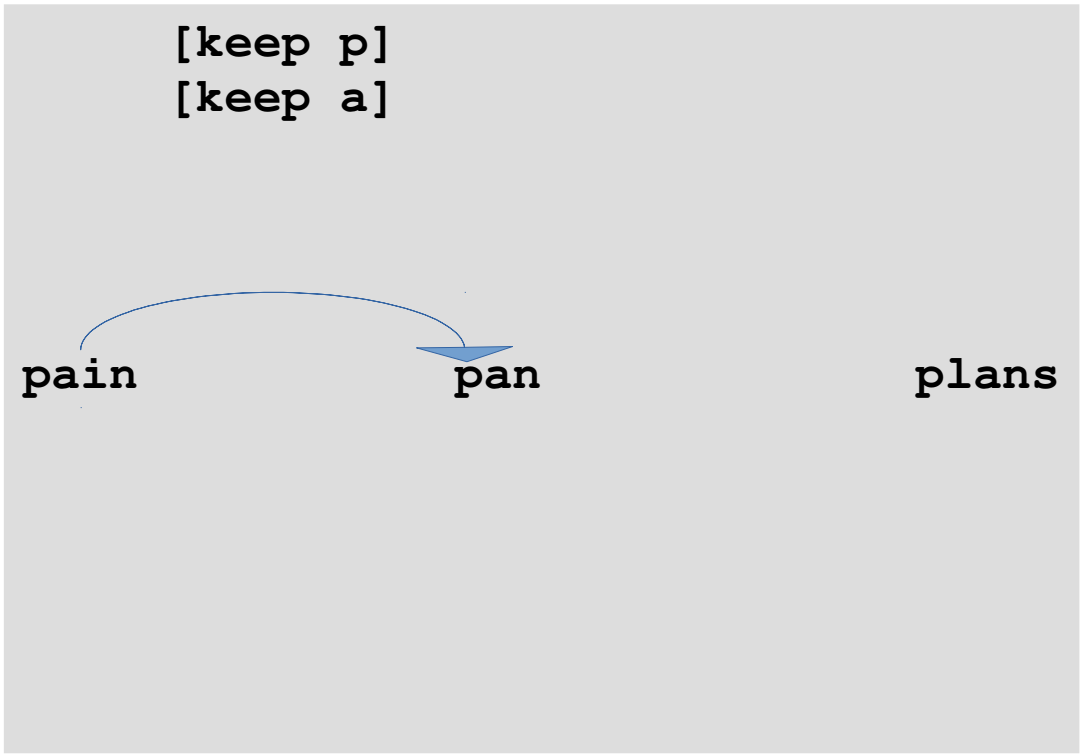
- Why is this interesting?
  - It turns out that finding the edits to turn one string into another is really easy if you know their LCS:




```
pain → plans
```

LCS

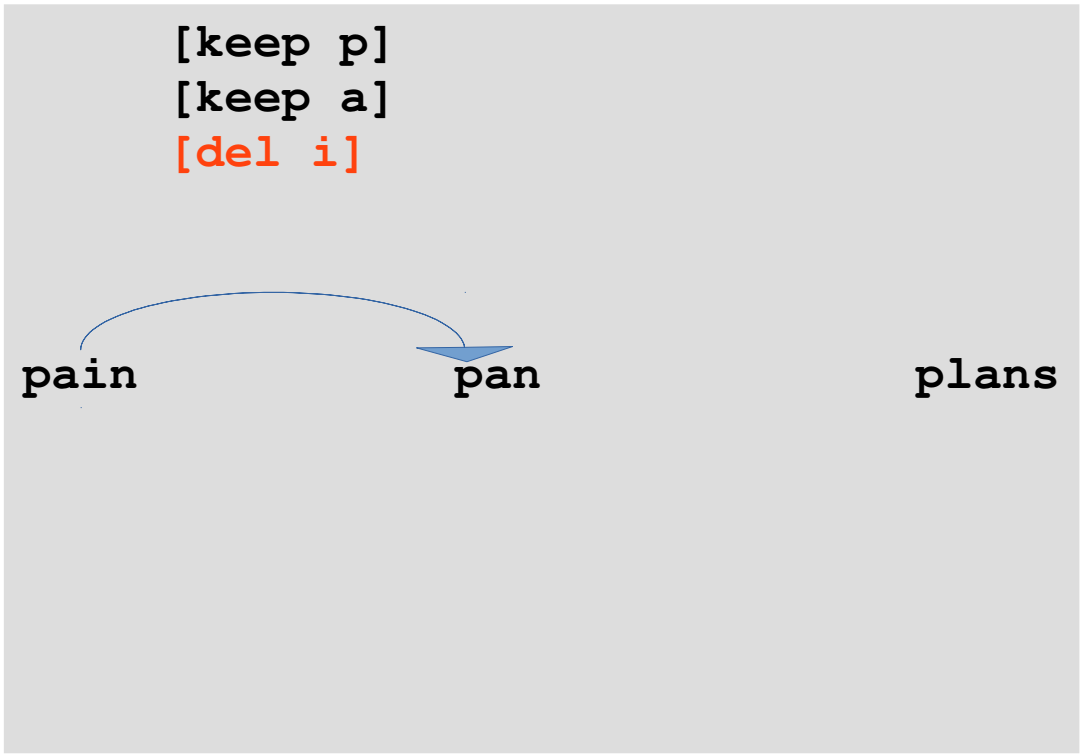
- Why is this interesting?
  - It turns out that finding the edits to turn one string into another is really easy if you know their LCS:




```
pain → plans
```

LCS

- Why is this interesting?
  - It turns out that finding the edits to turn one string into another is really easy if you know their LCS:

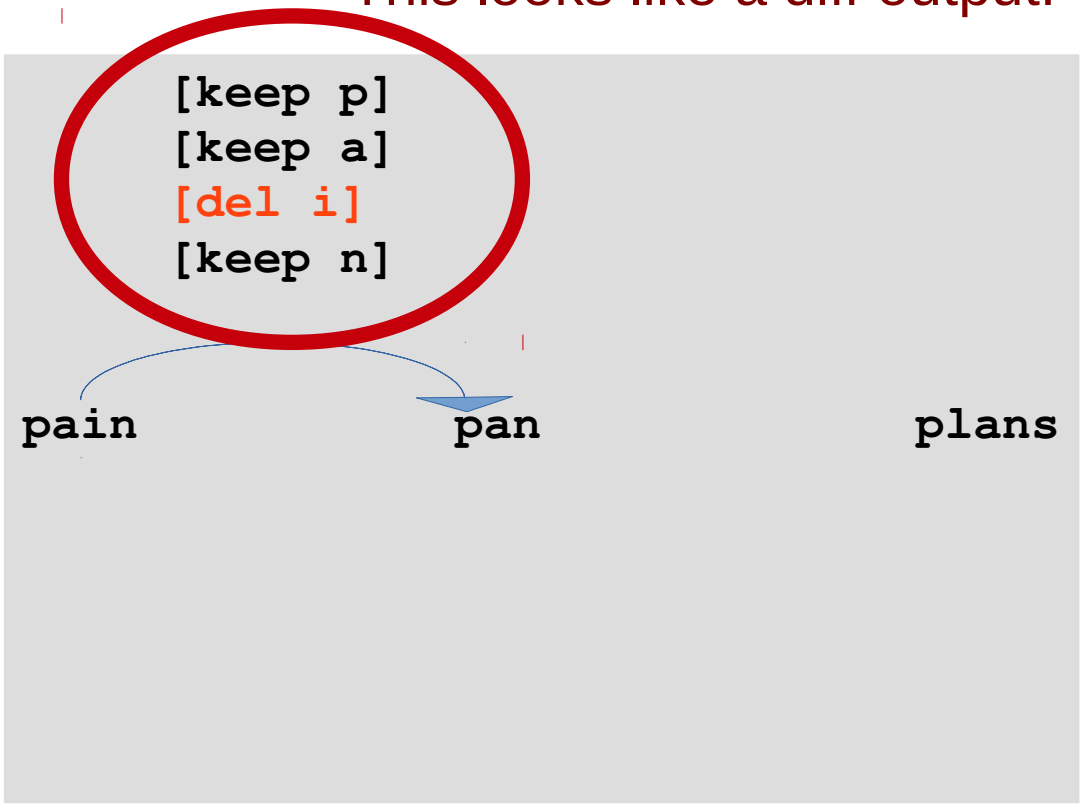


pain → plans

LCS

- Why is this interesting?
  - It turns out that finding the edits to turn one string into another is really easy if you know their LCS:

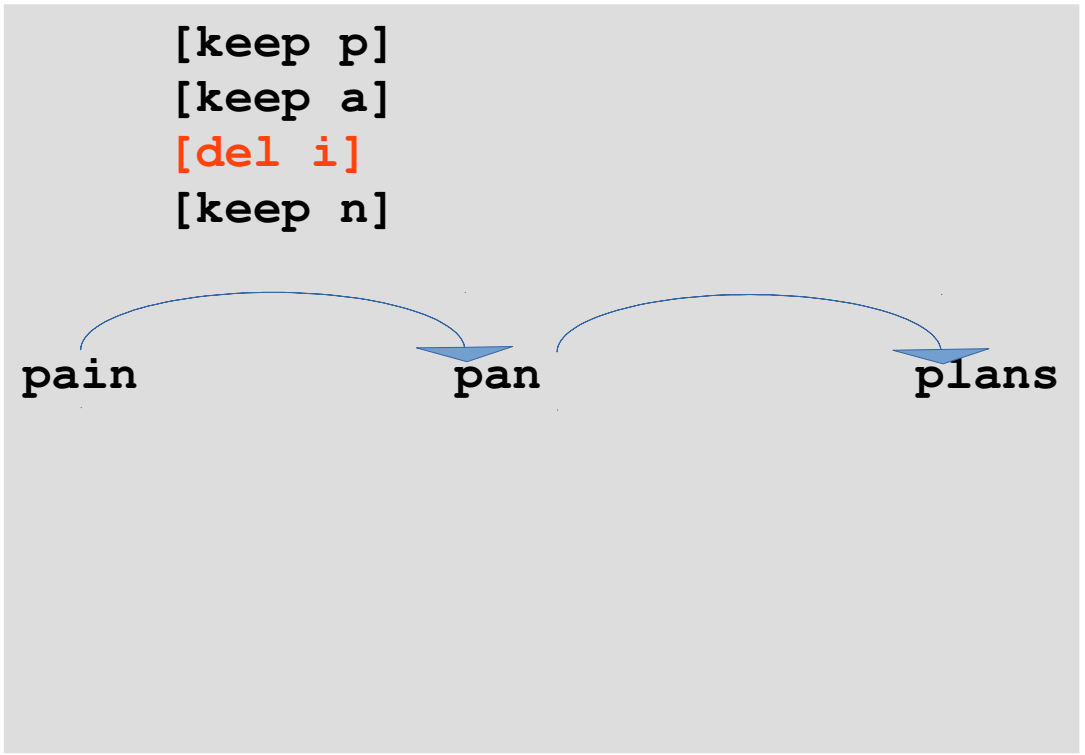
This looks like a diff output!




```
pain → plans
```

LCS

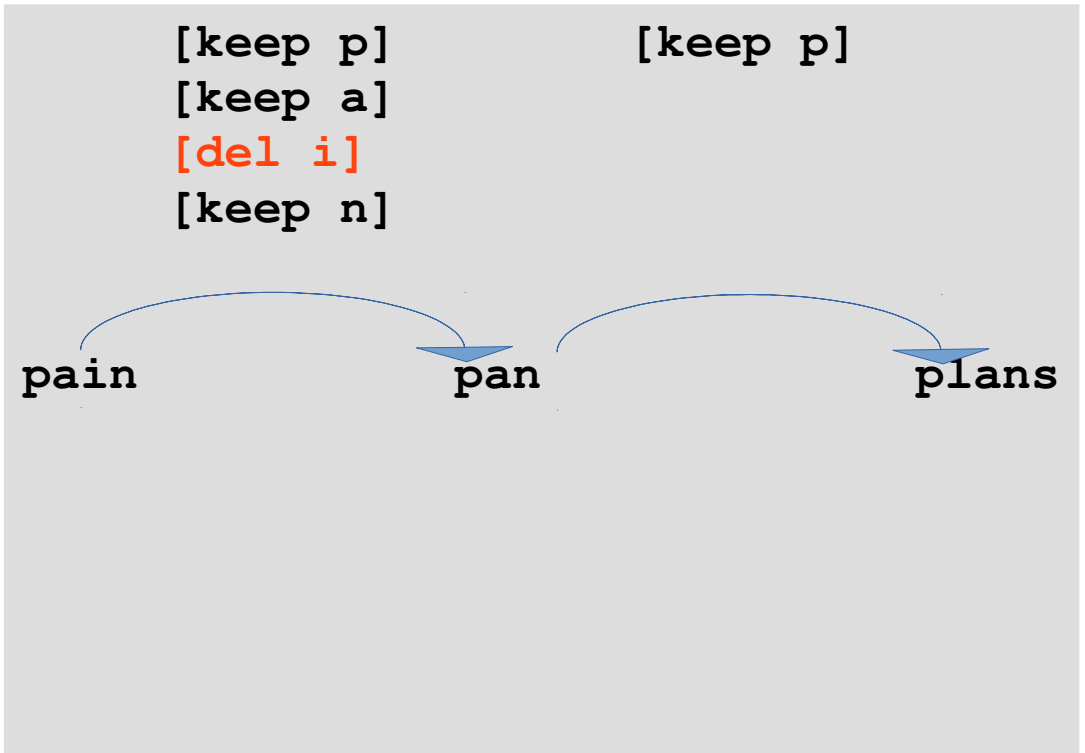
- Why is this interesting?
  - It turns out that finding the edits to turn one string into another is really easy if you know their LCS:




```
pain → plans
```

LCS

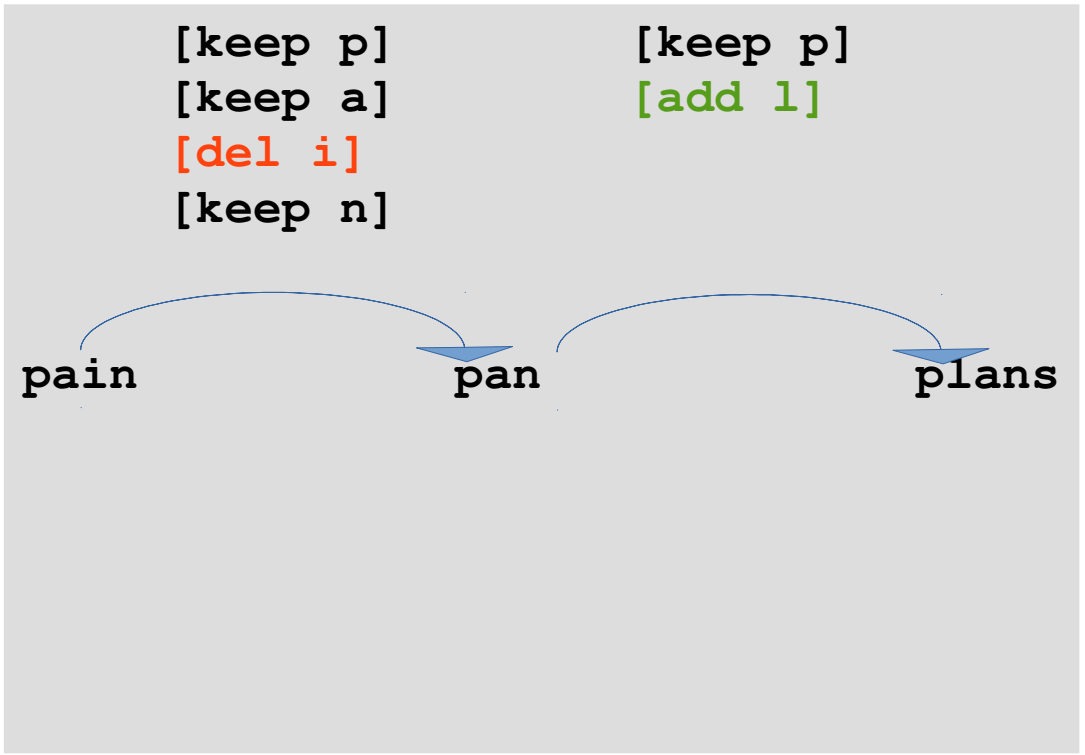
- Why is this interesting?
  - It turns out that finding the edits to turn one string into another is really easy if you know their LCS:



pain → plans

LCS

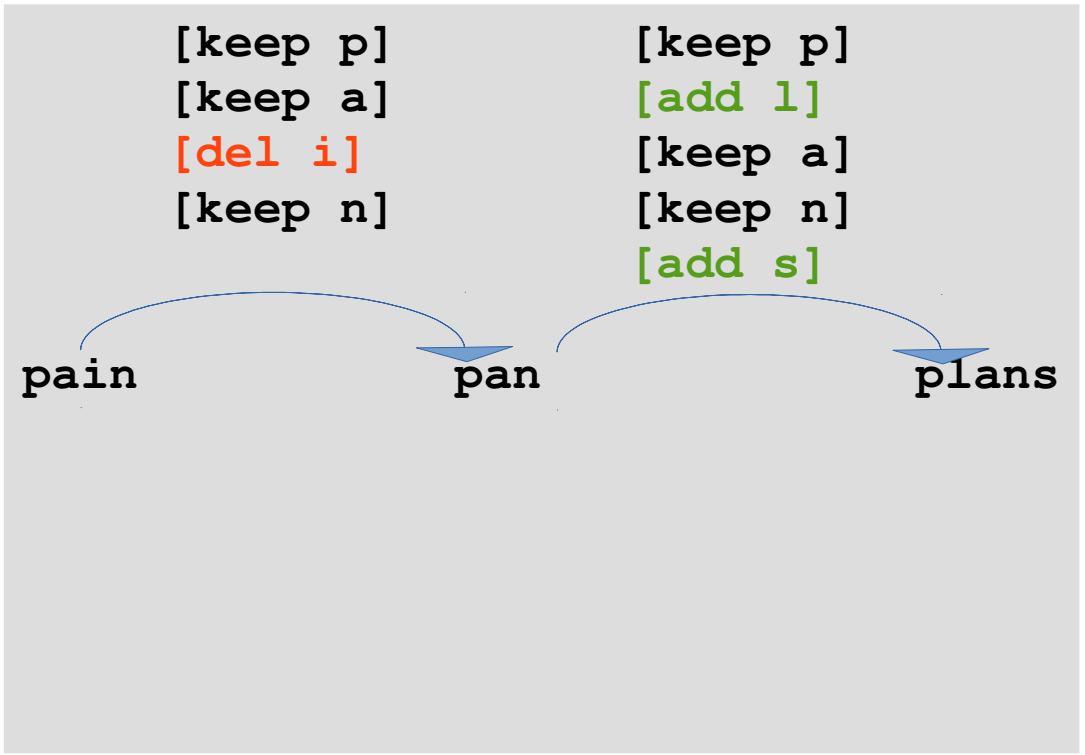
- Why is this interesting?
  - It turns out that finding the edits to turn one string into another is really easy if you know their LCS:




```
pain → plans
```

LCS

- Why is this interesting?
  - It turns out that finding the edits to turn one string into another is really easy if you know their LCS:

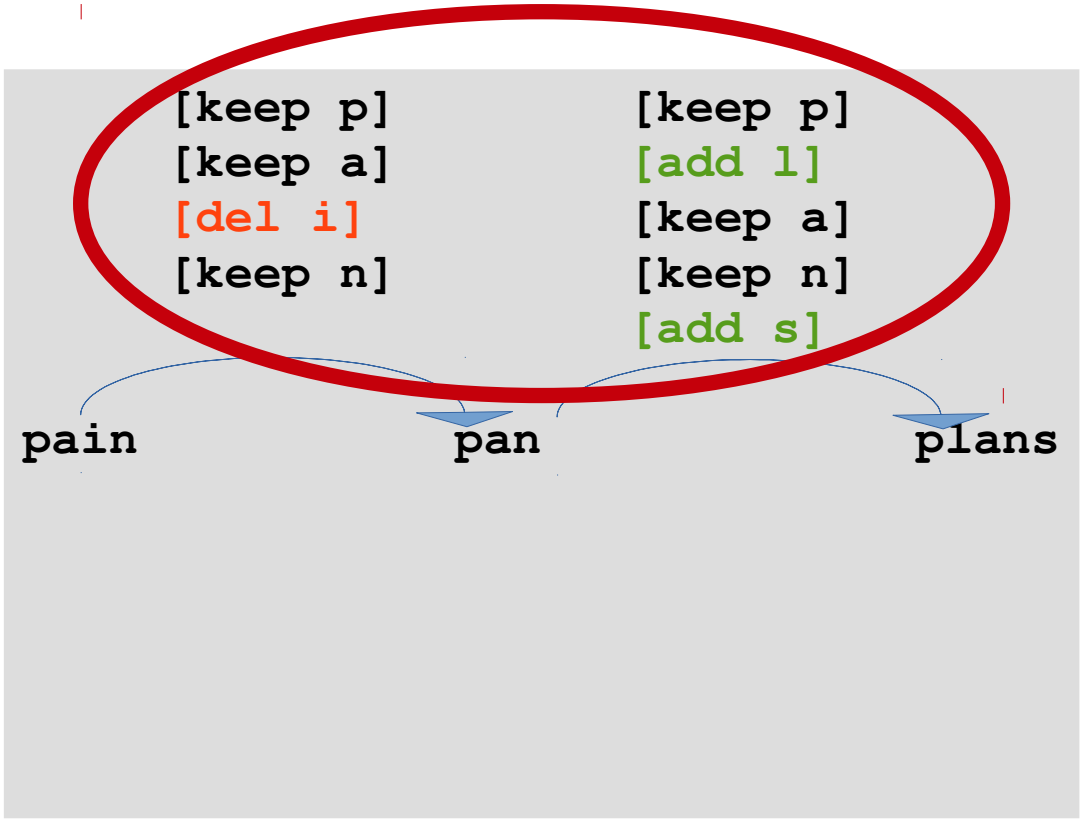



```
pain → plans
```

LCS

- Why is this interesting?
  - It turns out that finding the edits to turn one string into another is really easy if you know their LCS:

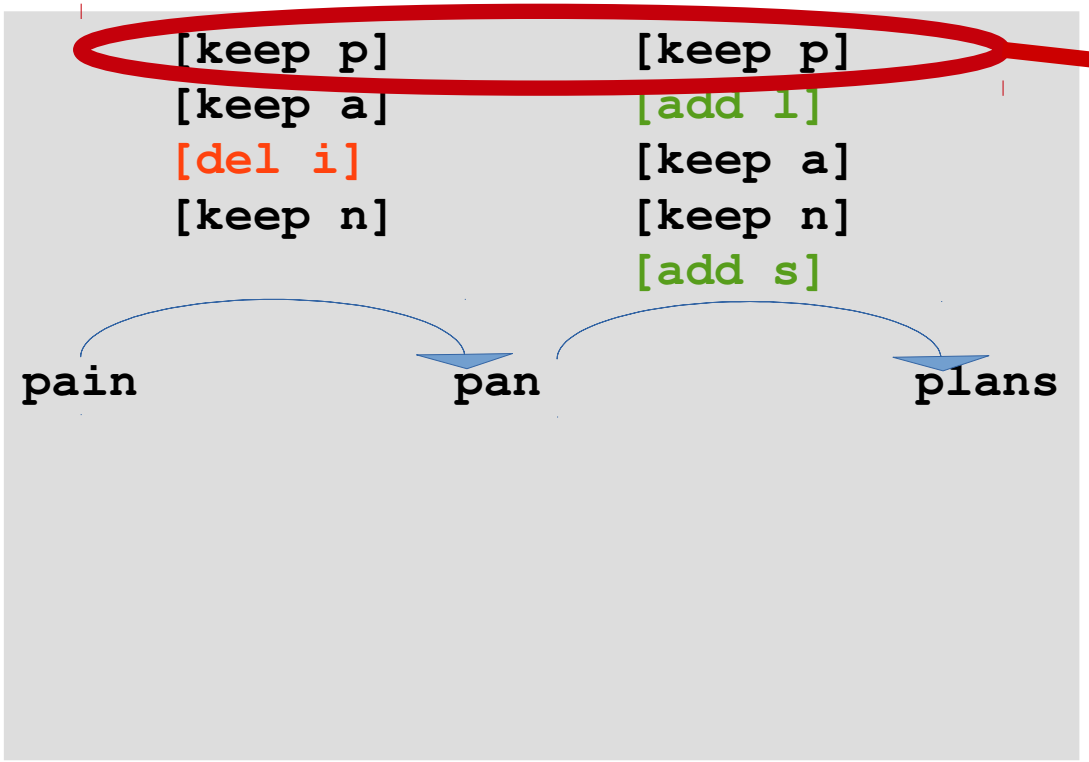
Let's “merge” these two diff “outputs”!

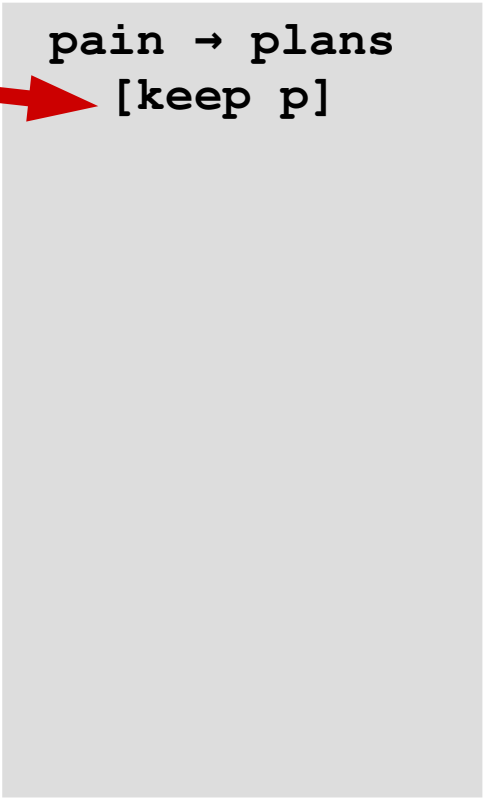



```
pain → plans
```

LCS

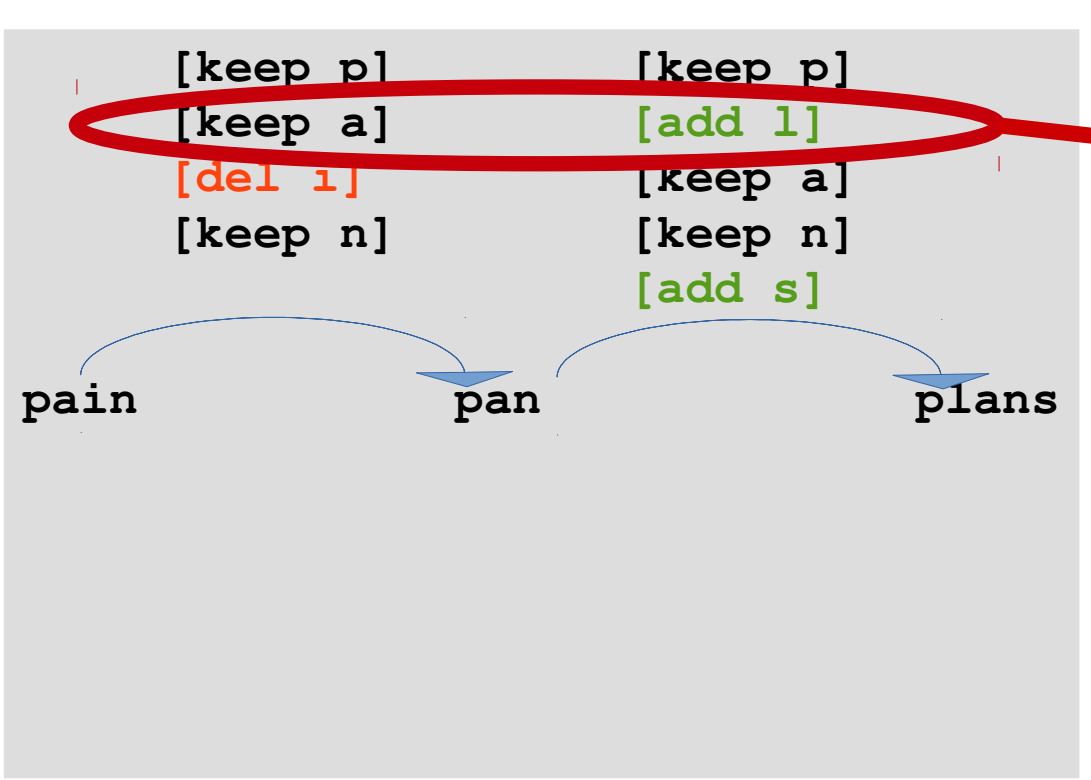
- Why is this interesting?
  - It turns out that finding the edits to turn one string into another is really easy if you know their LCS:





LCS

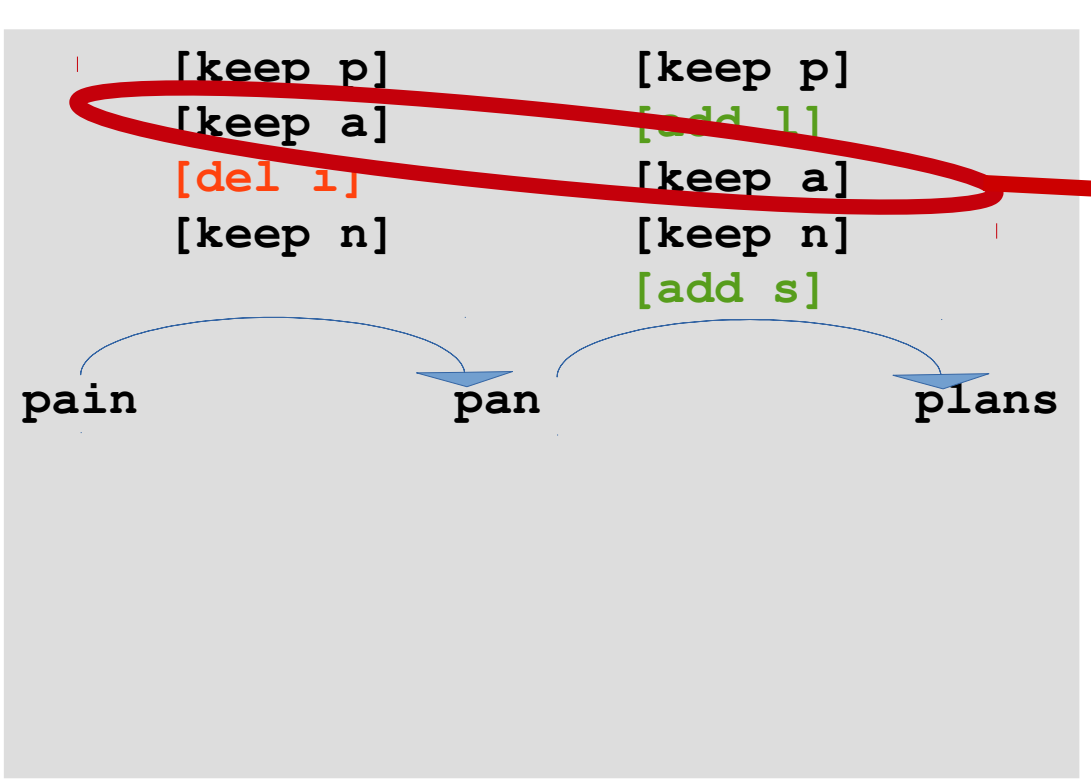
- Why is this interesting?
  - It turns out that finding the edits to turn one string into another is really easy if you know their LCS:




```
pain → plans
  [keep p]
  [add l]
```

LCS

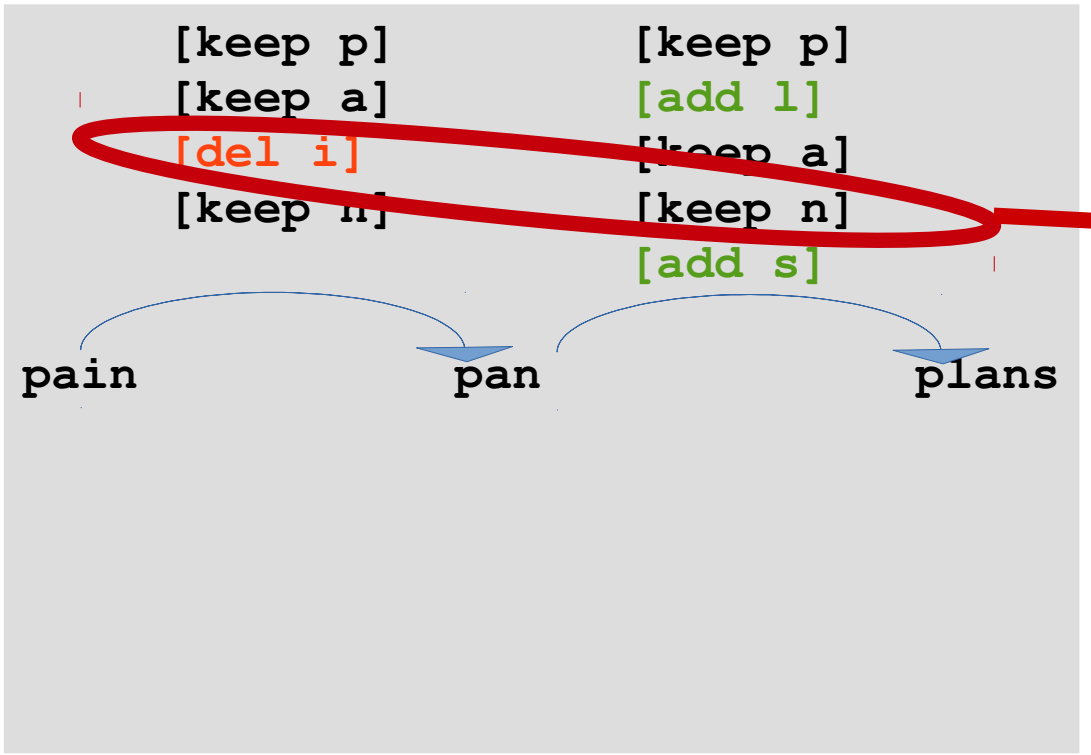
- Why is this interesting?
  - It turns out that finding the edits to turn one string into another is really easy if you know their LCS:




```
pain → plans
   [keep p]
   [add l]
   [keep a]
```

LCS

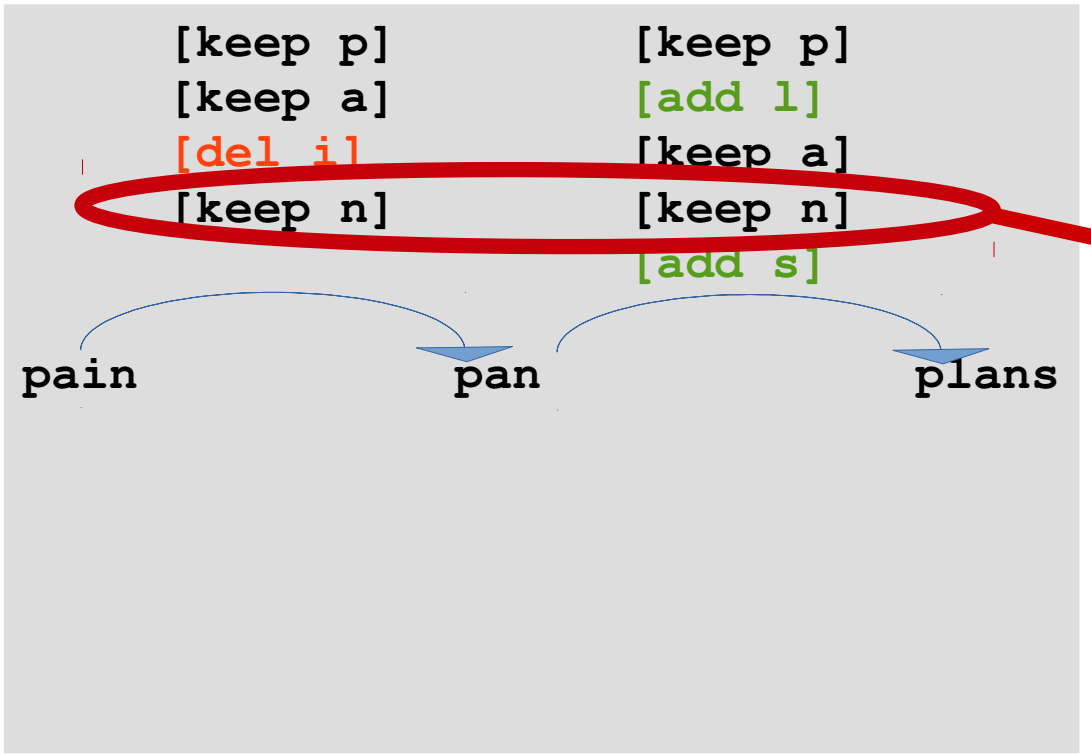
- Why is this interesting?
  - It turns out that finding the edits to turn one string into another is really easy if you know their LCS:




```
pain → plans
   [keep p]
   [add l]
   [keep a]
   [del i]
```

LCS

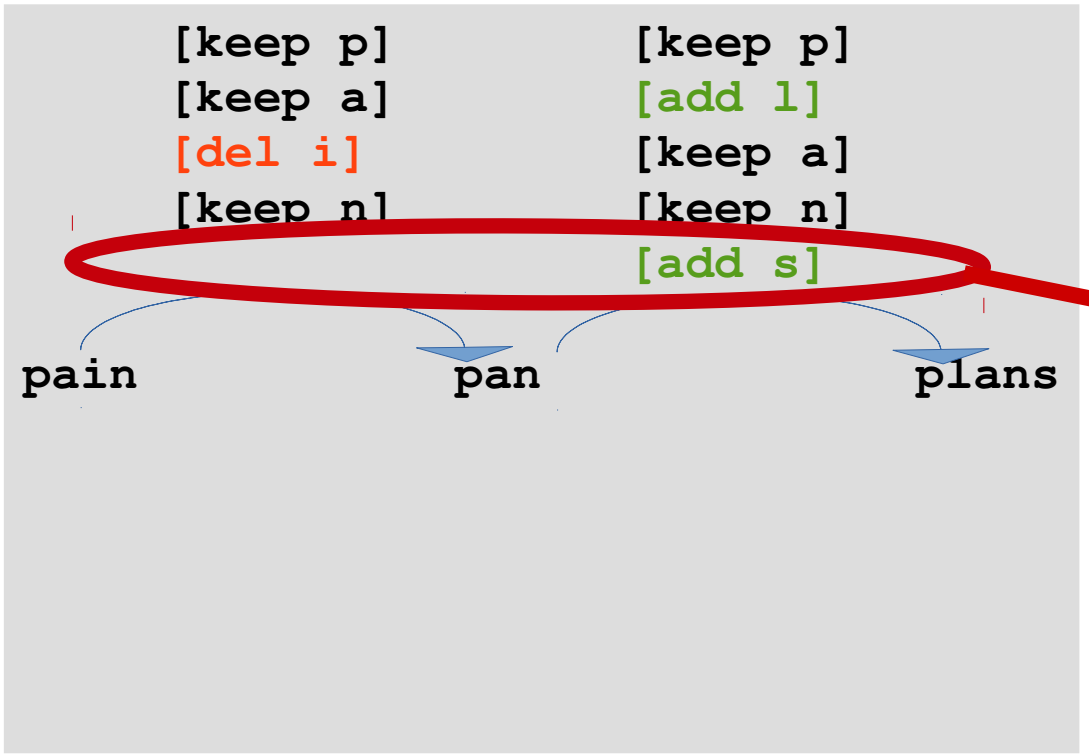
- Why is this interesting?
  - It turns out that finding the edits to turn one string into another is really easy if you know their LCS:




```
pain → plans
   [keep p]
   [add l]
   [keep a]
   [del i]
   [keep n]
```

LCS

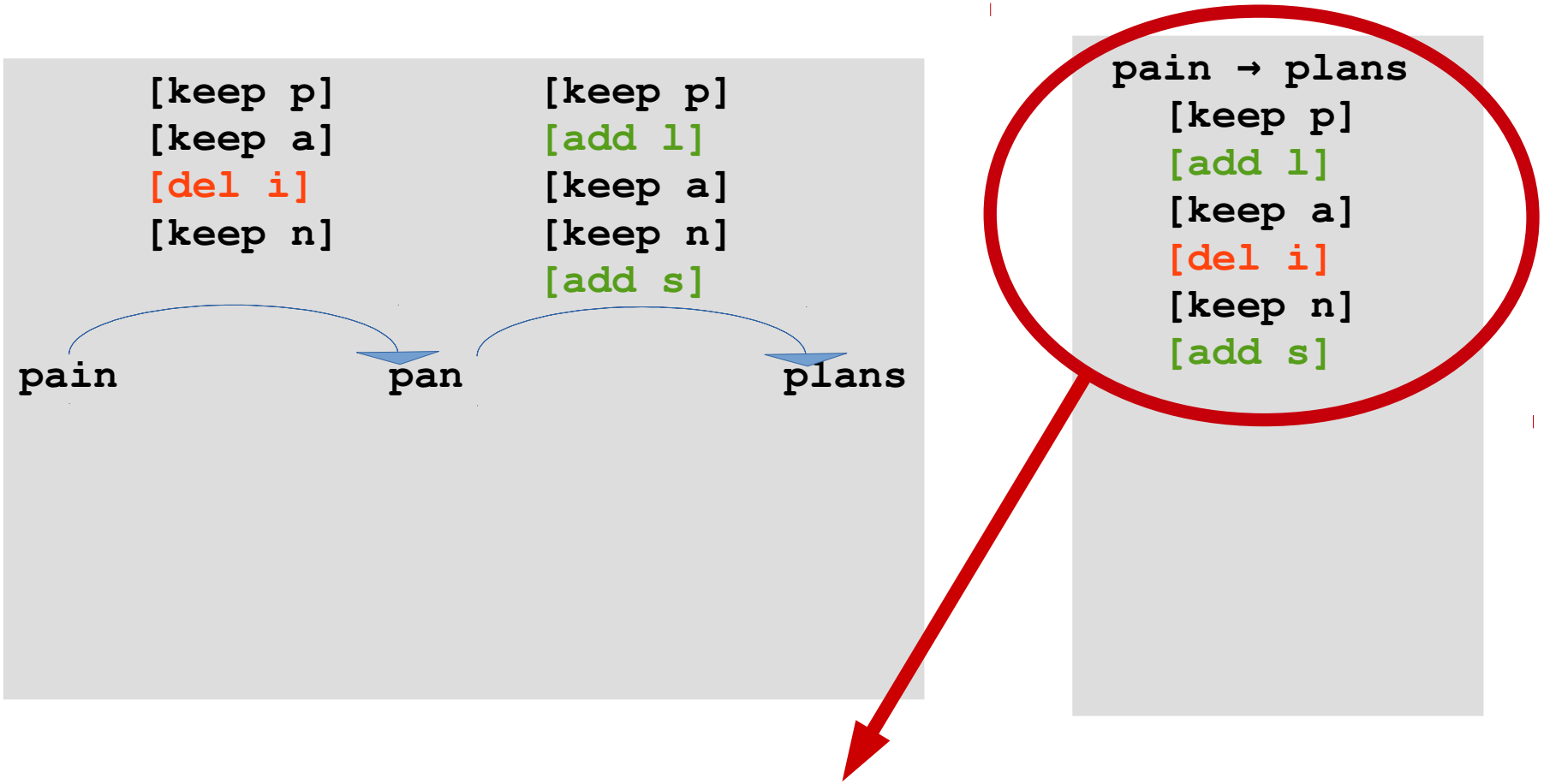
- Why is this interesting?
  - It turns out that finding the edits to turn one string into another is really easy if you know their LCS:




```
pain → plans
  [keep p]
  [add l]
  [keep a]
  [del i]
  [keep n]
  [add s]
```

LCS

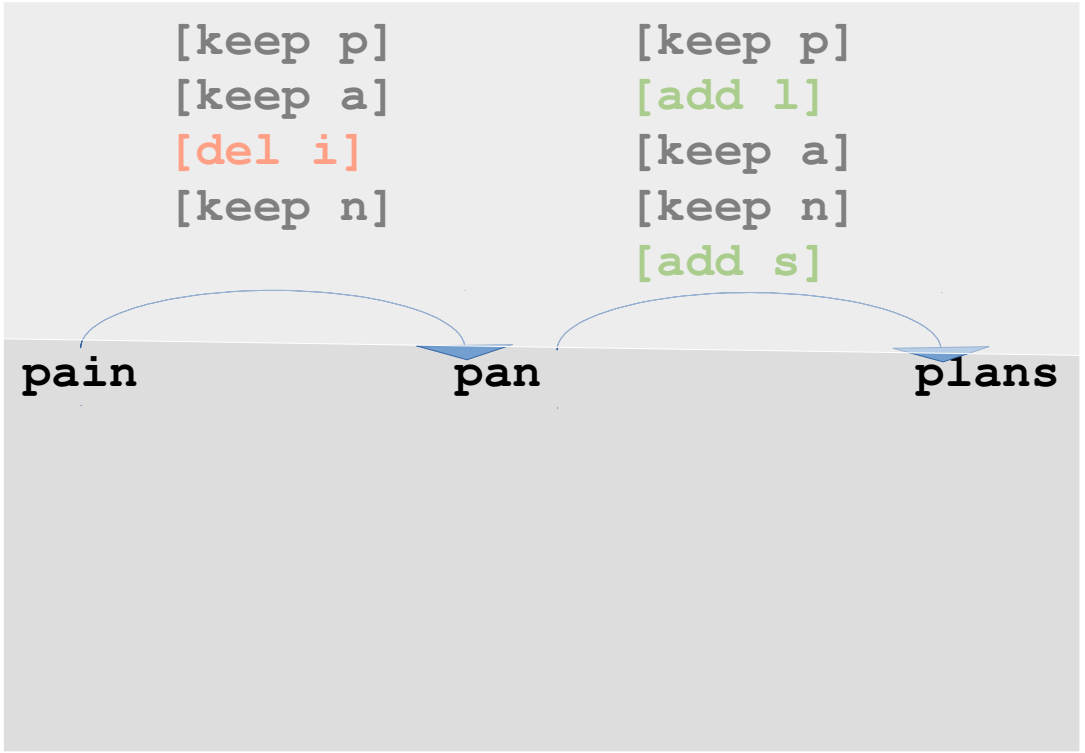
- Why is this interesting?
  - It turns out that finding the edits to turn one string into another is really easy if you know their LCS:



This is the full diff between “pain” and “plans”!

LCS

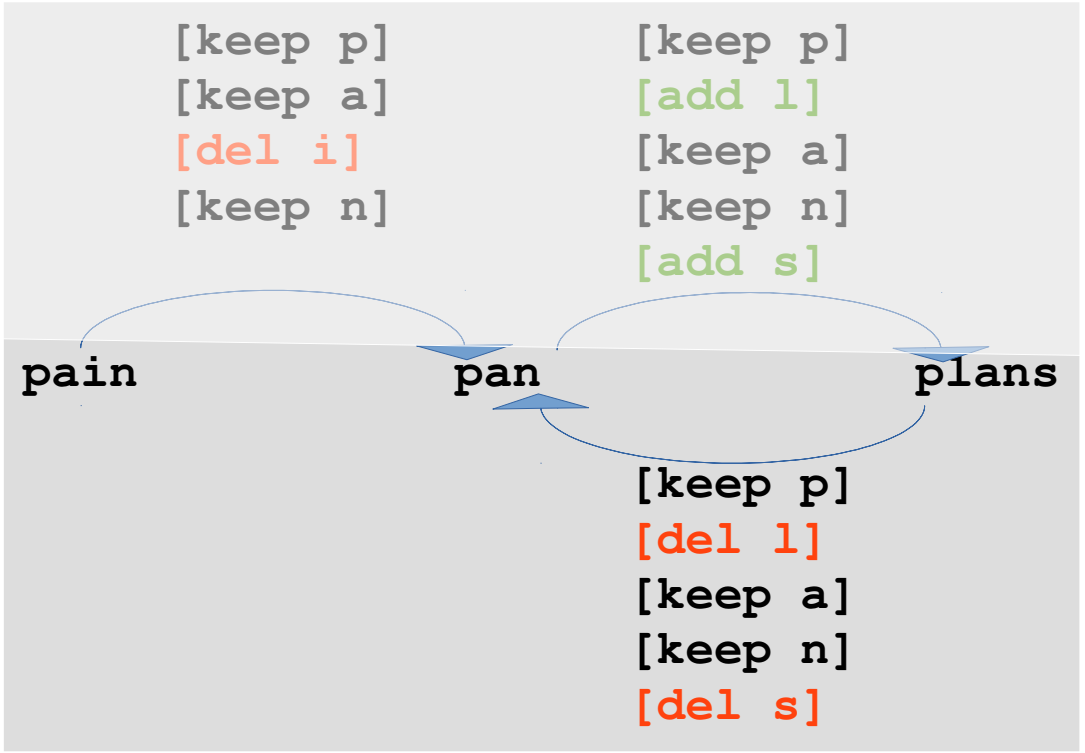
- Why is this interesting?
  - It turns out that finding the edits to turn one string into another is really easy if you know their LCS:




```
pain → plans
   [keep p]
   [add l]
   [keep a]
   [del i]
   [keep n]
   [add s]
plans → pain
```

LCS

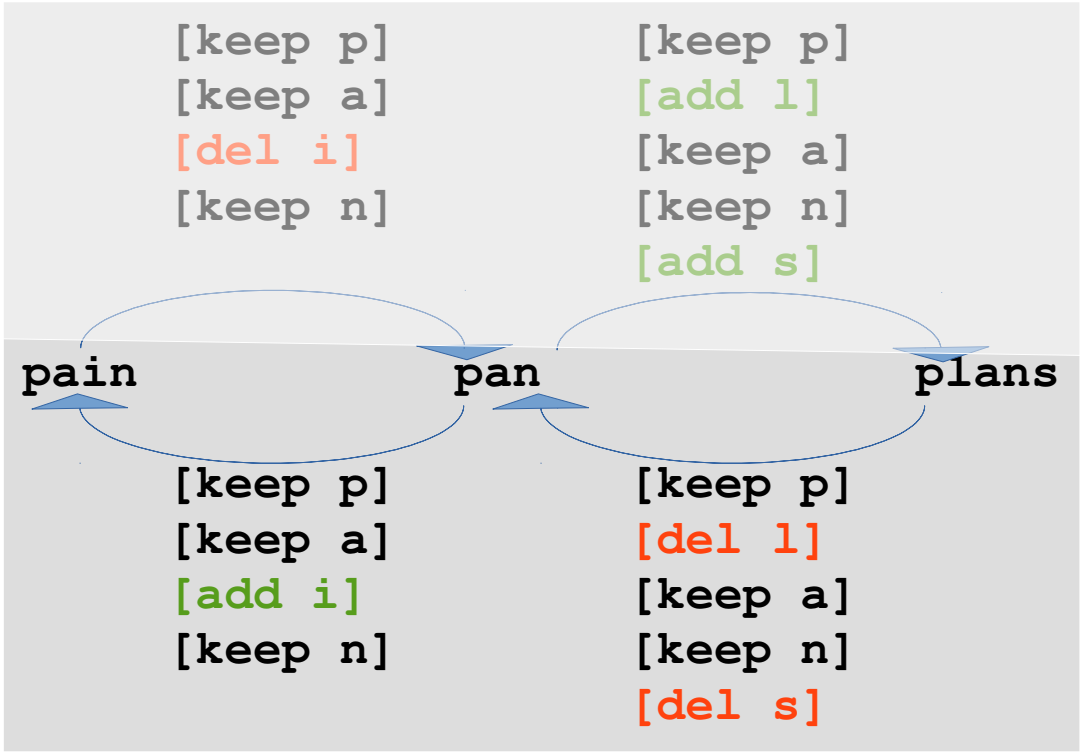
- Why is this interesting?
  - It turns out that finding the edits to turn one string into another is really easy if you know their LCS:




```
pain → plans
   [keep p]
   [add l]
   [keep a]
   [del i]
   [keep n]
   [add s]
plans → pain
```

LCS

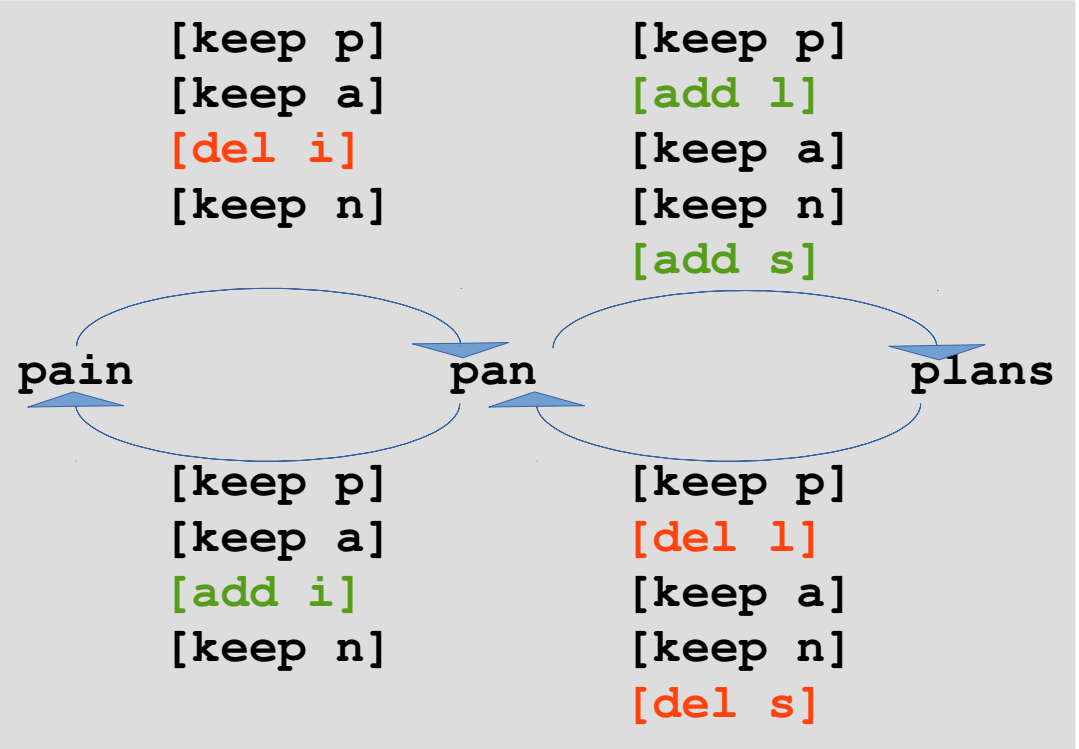
- Why is this interesting?
  - It turns out that finding the edits to turn one string into another is really easy if you know their LCS:




```
pain → plans
   [keep p]
   [add l]
   [keep a]
   [del i]
   [keep n]
   [add s]
plans → pain
```

LCS

- Why is this interesting?
  - It turns out that finding the edits to turn one string into another is really easy if you know their LCS:




```
pain → plans
   [keep p]
   [add l]
   [keep a]
   [del i]
   [keep n]
   [add s]
plans → pain
   [keep p]
   [del l]
   [keep a]
   [add i]
   [keep n]
   [del s]
```

LCS

- Conclusion
  - calculating LCS is equivalent to finding the edits of a diff

Diff

Diff

- Original version from 1970
  - Used heuristics to calculate LCS
- Final version from 1974 (UNIX 5th ed.) by Douglas McIlroy (*1932)
  - Mathematician and engineer
  - diff, sort, spell, join, tr, and UNIX pipes

Douglas McIlroy
(Photo from Wikipedia)

Diff

- The final version of diff is described in:

Hunt, James W.; McIlroy, M. Douglas (June 1976). "*An Algorithm for Differential File Comparison*." Computing Science Technical Report (Bell Laboratories) 41.

- Approx. the same LCS algorithm described here, but with a lot of optimizations:
  - Hirschberg's k-candidates
  - Hashing
  - Presorting into equivalent classes
  - Merging by binary search
  - Dynamic storage allocation

Diff

- But... diff is *line oriented* not characater oriented (as LCS)
  - Correct, but that is not a big deal, we can use hashing, for example:

```
1  package main
2
3  import ”fmt”
4
5  func main() {
6      fmt.Println(”Hello world!”)
7  }
```

```
a = ”                ”
```

Diff

- But... diff is *line oriented* not characater oriented *(*as LCS)
  - Correct, but that is not a big deal, we can use hashing, for example:

```
1  package main
2
3  import ”fmt”
4
5  func main() {
6      fmt.Println(”Hello world!”)
7  }
```

```
a = ”Ж          ”
```

Diff

- But... diff is *line oriented* not characater oriented *(*as LCS)
  - Correct, but that is not a big deal, we can use hashing, for example:

```
1  package main
2
3  import "fmt"
4
5  func main() {
6      fmt.Println("Hello world!")
7  }
```

```
a = "Ж√    "
```

Diff

- But... diff is *line oriented* not characater oriented *(*as LCS)
  - Correct, but that is not a big deal, we can use hashing, for example:

```
1  package main
2
3  import ”fmt”
4
5  func main() {
6      fmt.Println(”Hello world!”)
7  }
```

```
a = ”Ж√€      ”
```

Diff

- But... diff is *line oriented* not characater oriented *(*as LCS)
  - Correct, but that is not a big deal, we can use hashing, for example:

```
1  package main
2
3  import ”fmt”
4
5  func main() {
6      fmt.Println(”Hello world!”)
7  }
```

```
a = ”Ж√€√    ”
```

Diff

- But... diff is *line oriented* not characater oriented *(*as LCS)
  - Correct, but that is not a big deal, we can use hashing, for example:

```
1  package main
2
3  import ”fmt”
4
5  func main() {
6      fmt.Println(”Hello world!”)
7  }
```

```
a = ”Ж√€√A?©”
```

Diff

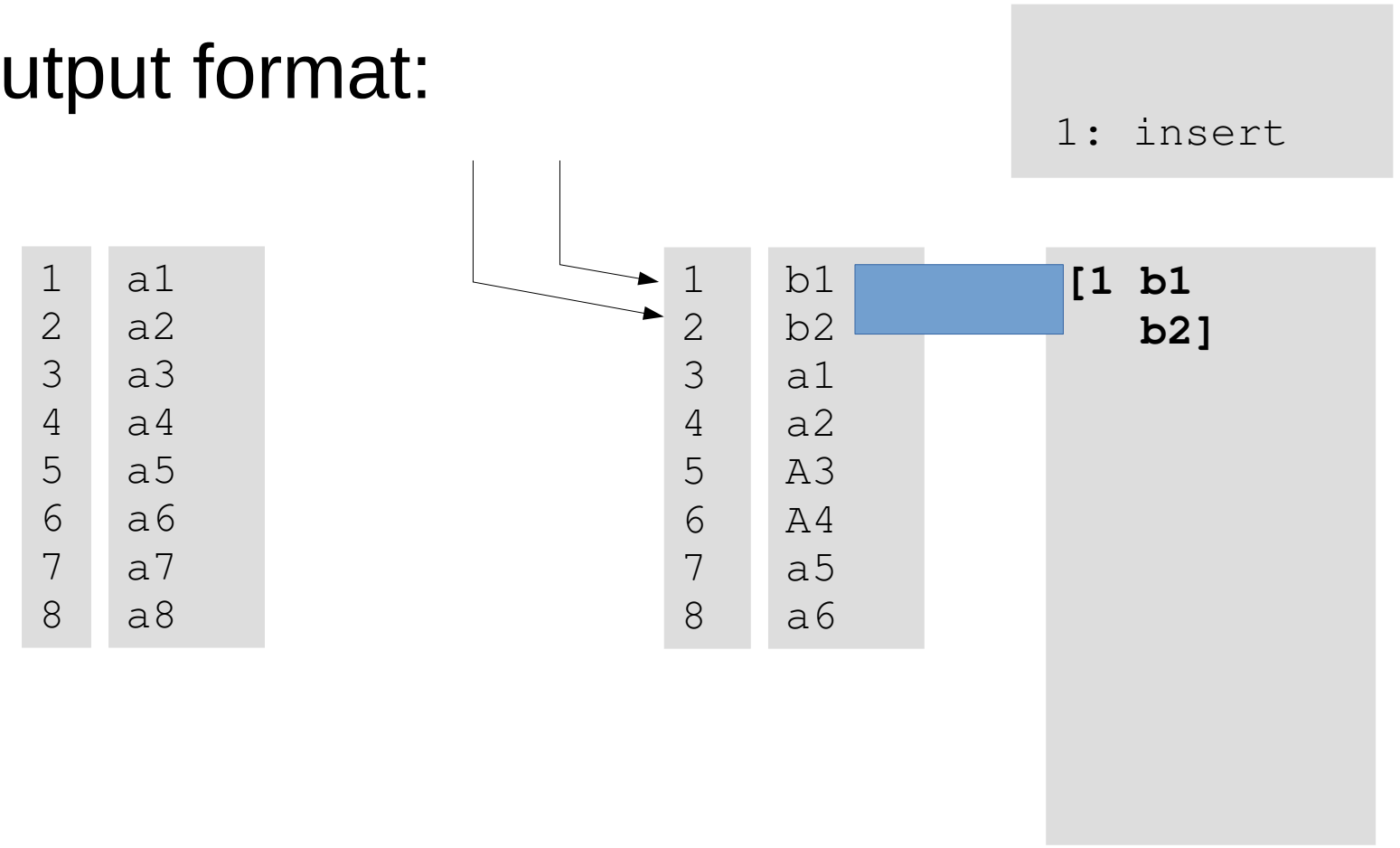
- Output format:

1	a1
2	a2
3	a3
4	a4
5	a5
6	a6
7	a7
8	a8

1	b1
2	b2
3	a1
4	a2
5	A3
6	A4
7	a5
8	a6

Diff

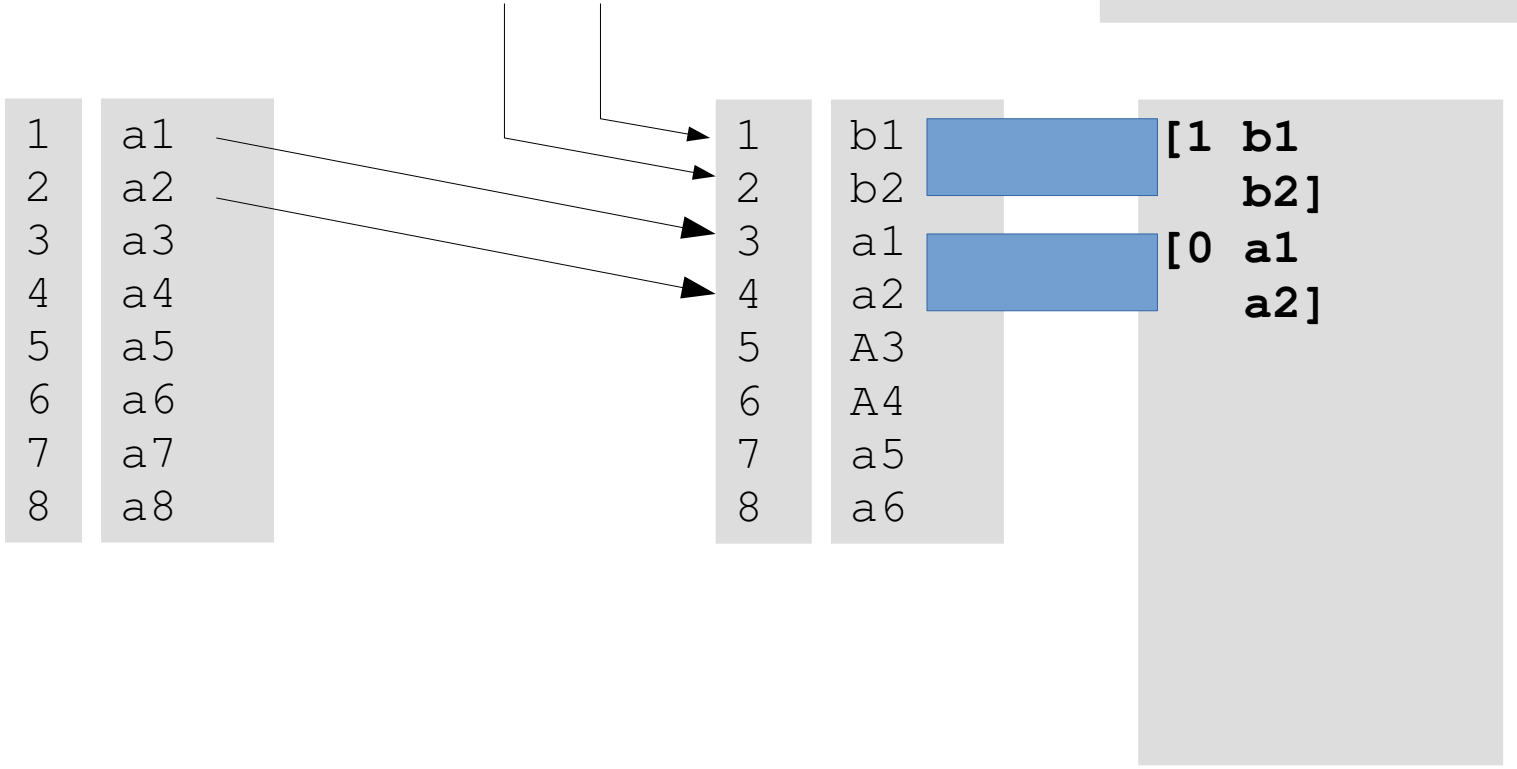
- Output format:



Diff

- Output format:

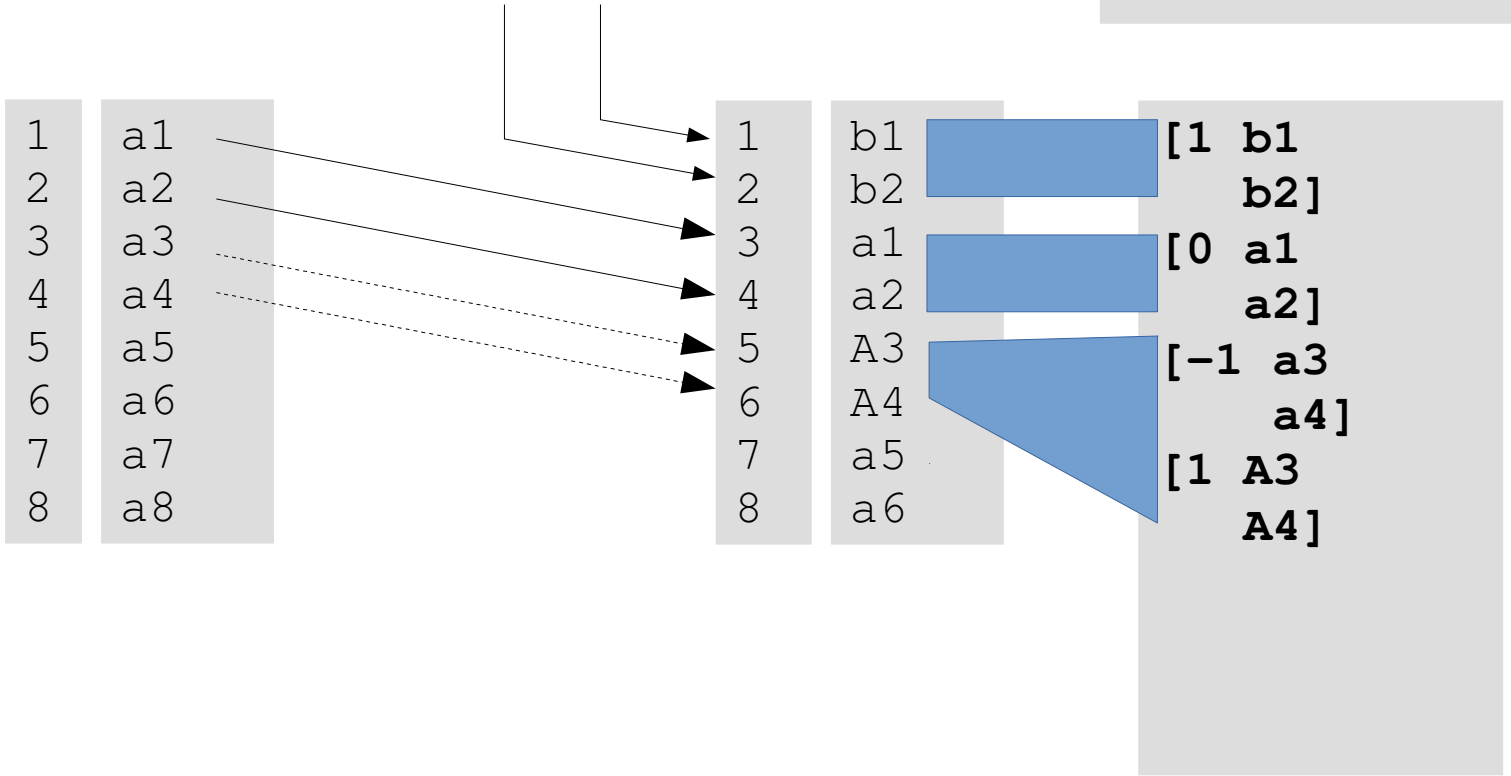
```
0: keep
1: insert
```



Diff

- Output format:

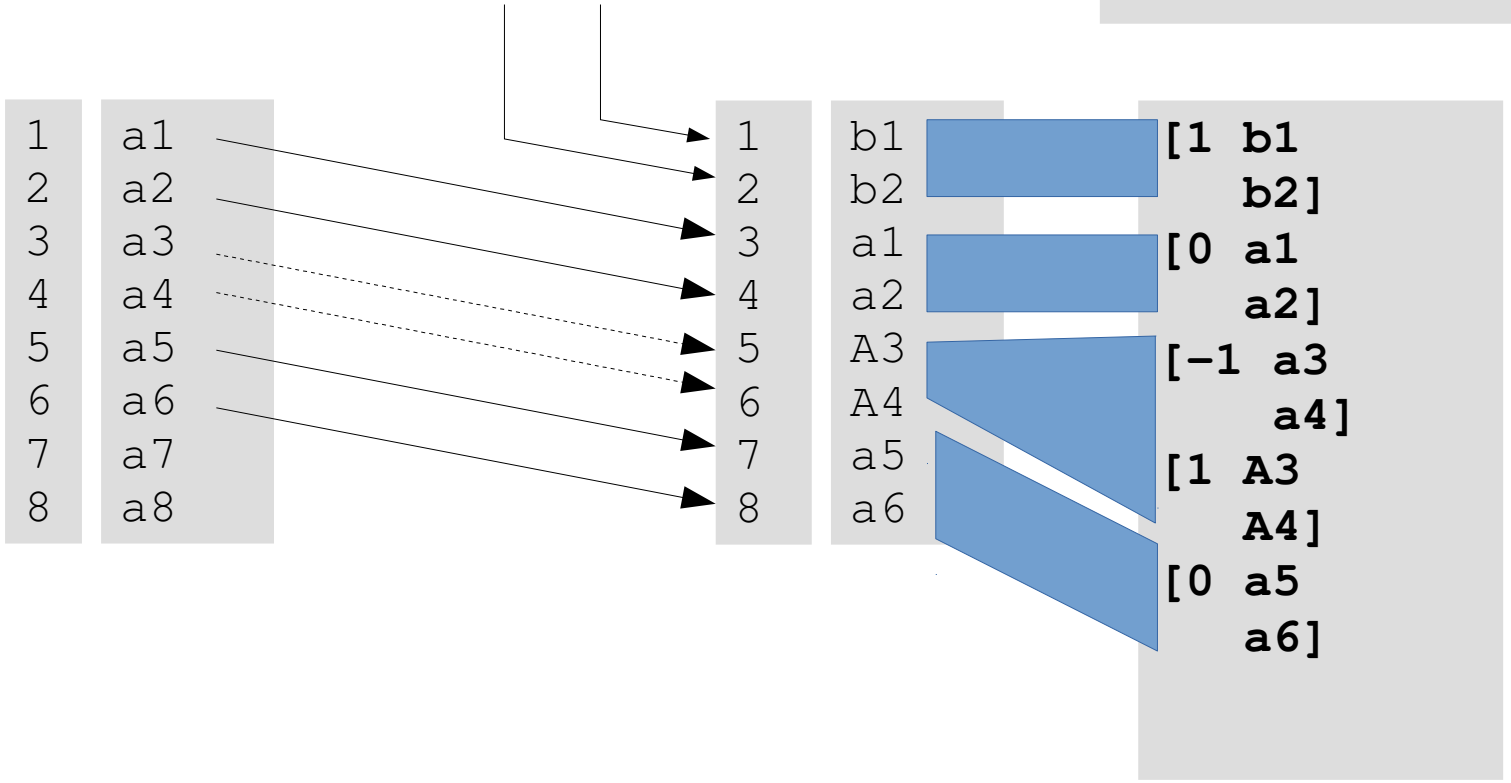
```
-1: delete
 0: keep
 1: insert
```



Diff

- Output format:

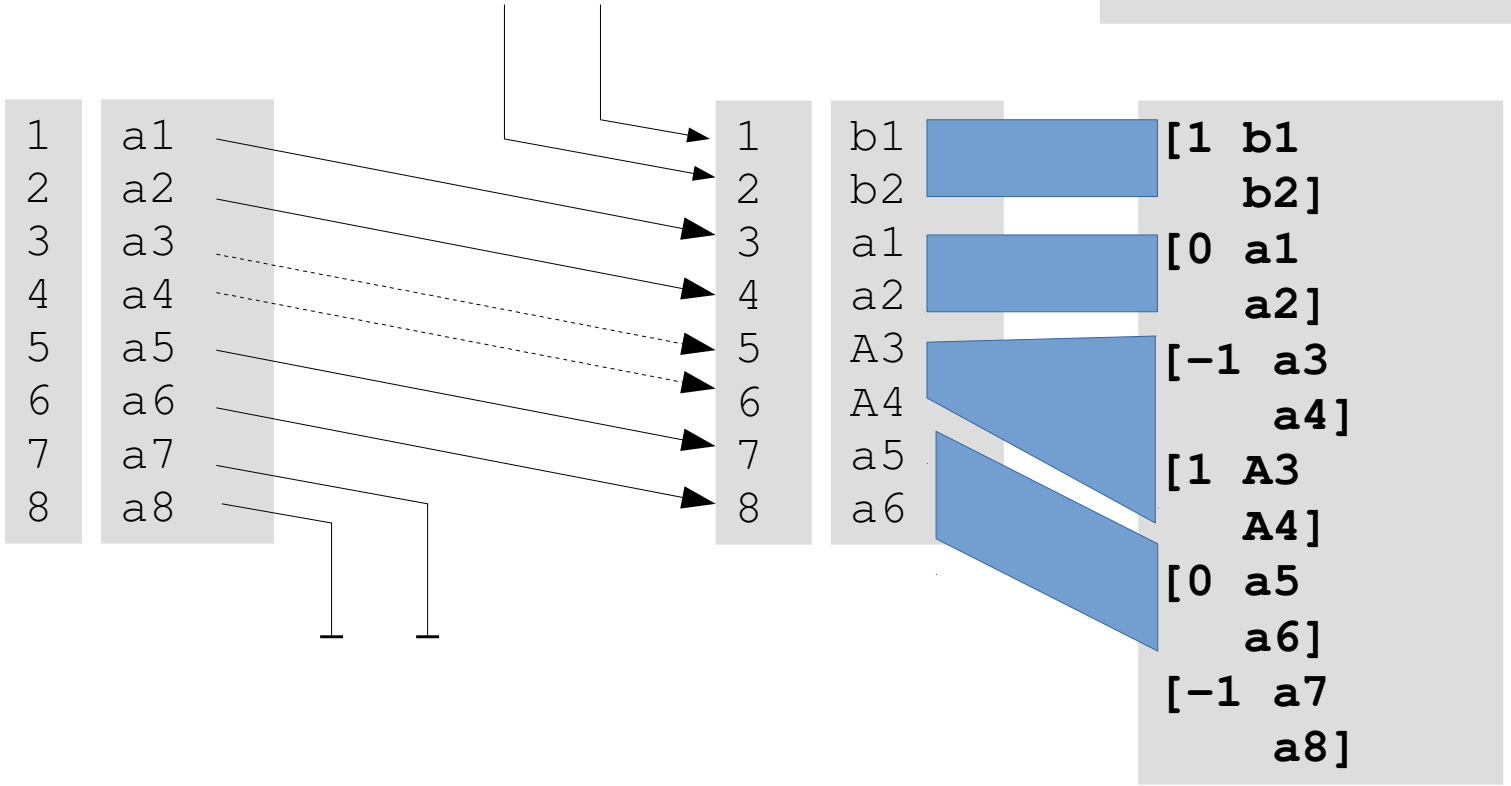
```
-1: delete
 0: keep
 1: insert
```



Diff

- Output format:

```
-1: delete
 0: keep
 1: insert
```



Diff

- Conclusions
  - Diff uses LCS to calculate the minimum set of edits to turn one file into another
  - Standard LCS can be used if you turn lines into 'chars'
  - Diff is highly optimized
  - Output format example:

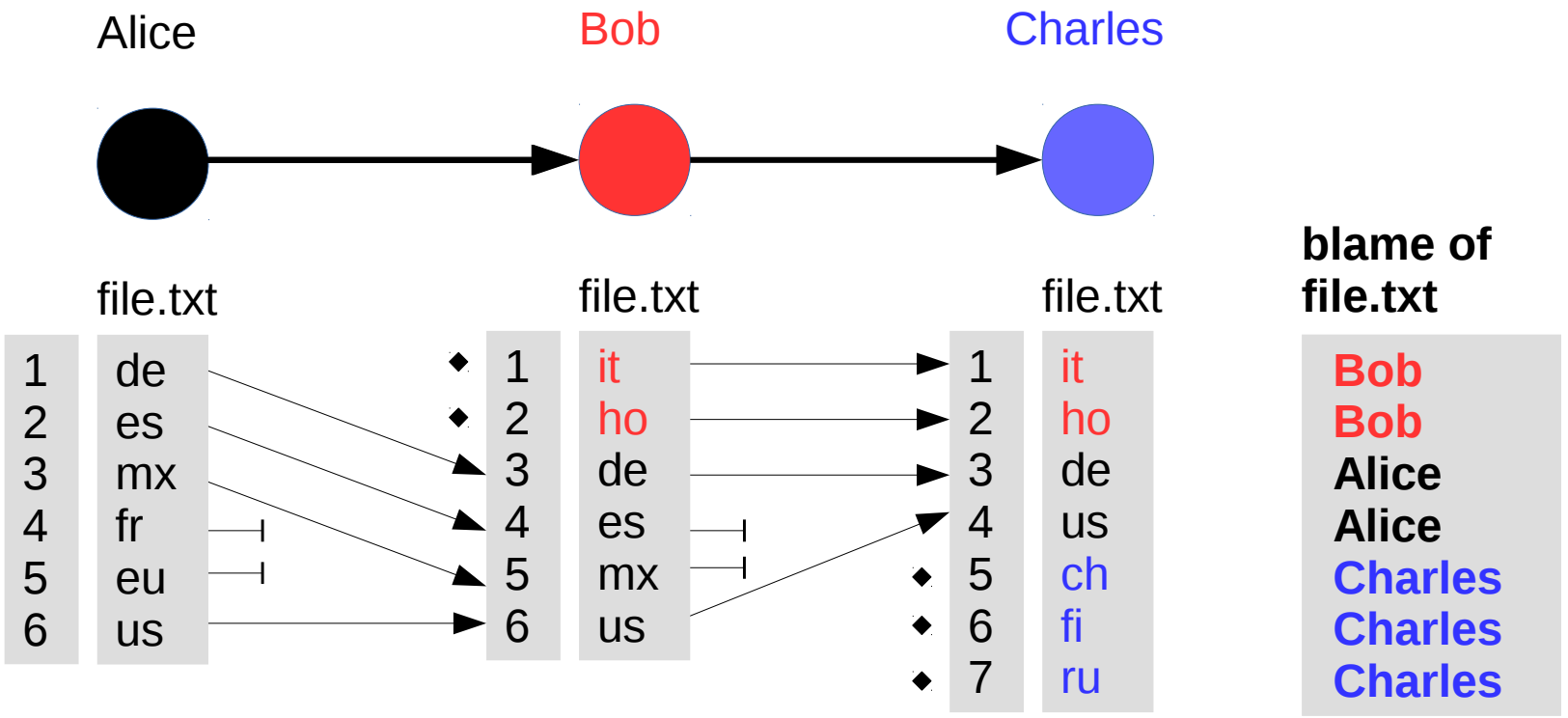
```
[1 line1
   line2]
[0 line3
   line4]
[-1 line5
    line6]
```

```
-1: delete
 0: keep
 1: insert
```

Blame

Blame

To blame a file is to find who wrote each of its lines

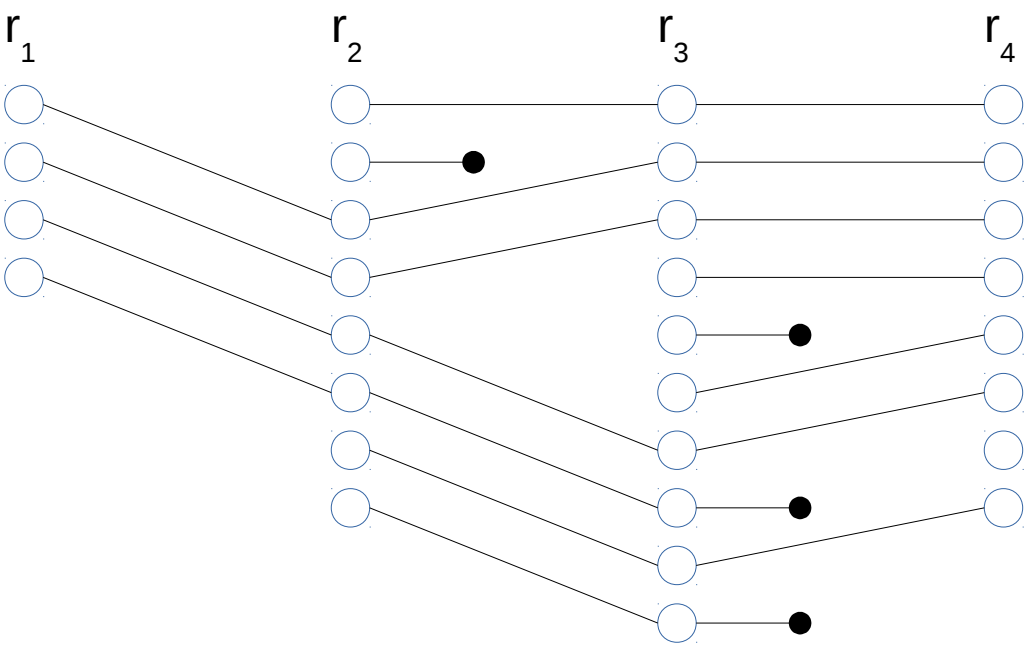


Blame

- Blaming a file is basically **tracking its lines** across all its revisions

Zimmermann, Thomas et al. (May 2006). "*Mining Version Archives for Co-changed Lines*." In proceedings of Mining Software Repositories Workshop, Shanghai, China.

Blame

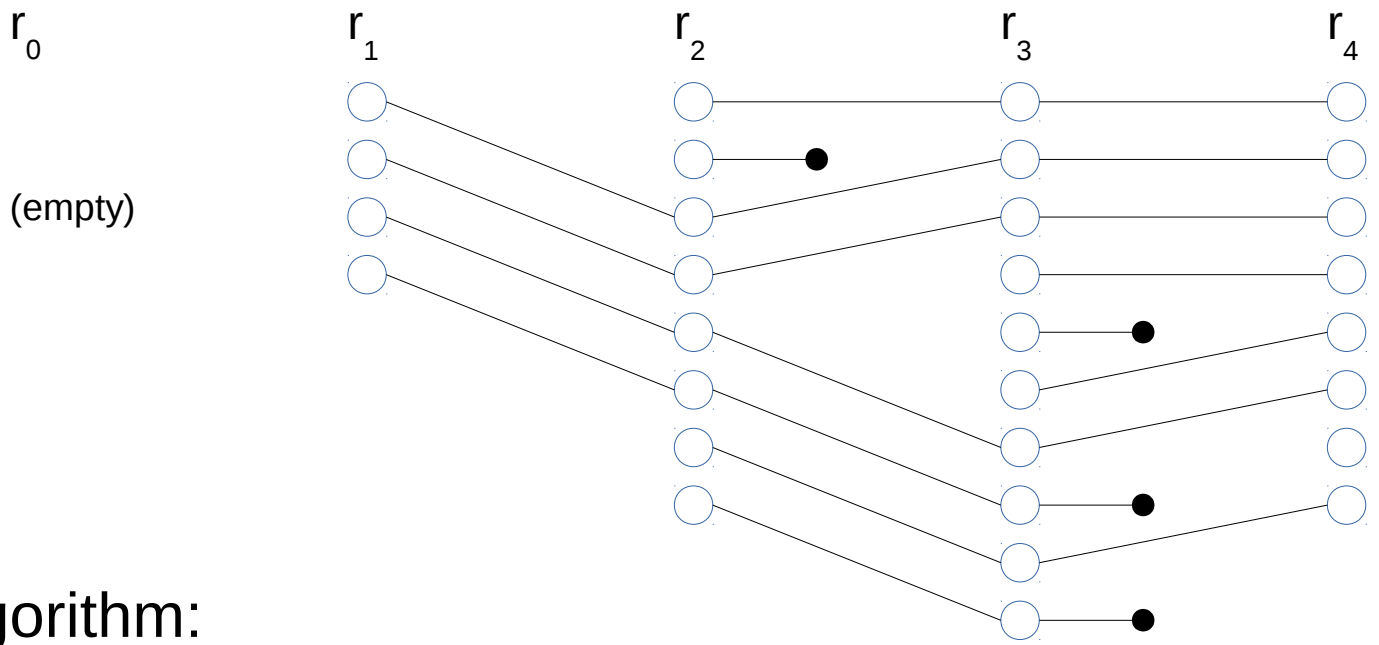


- Easily solved by building a graph:
  - Each node is a line in a file (for all revisions)
  - Each edge came from the diff between two revisions

Blame

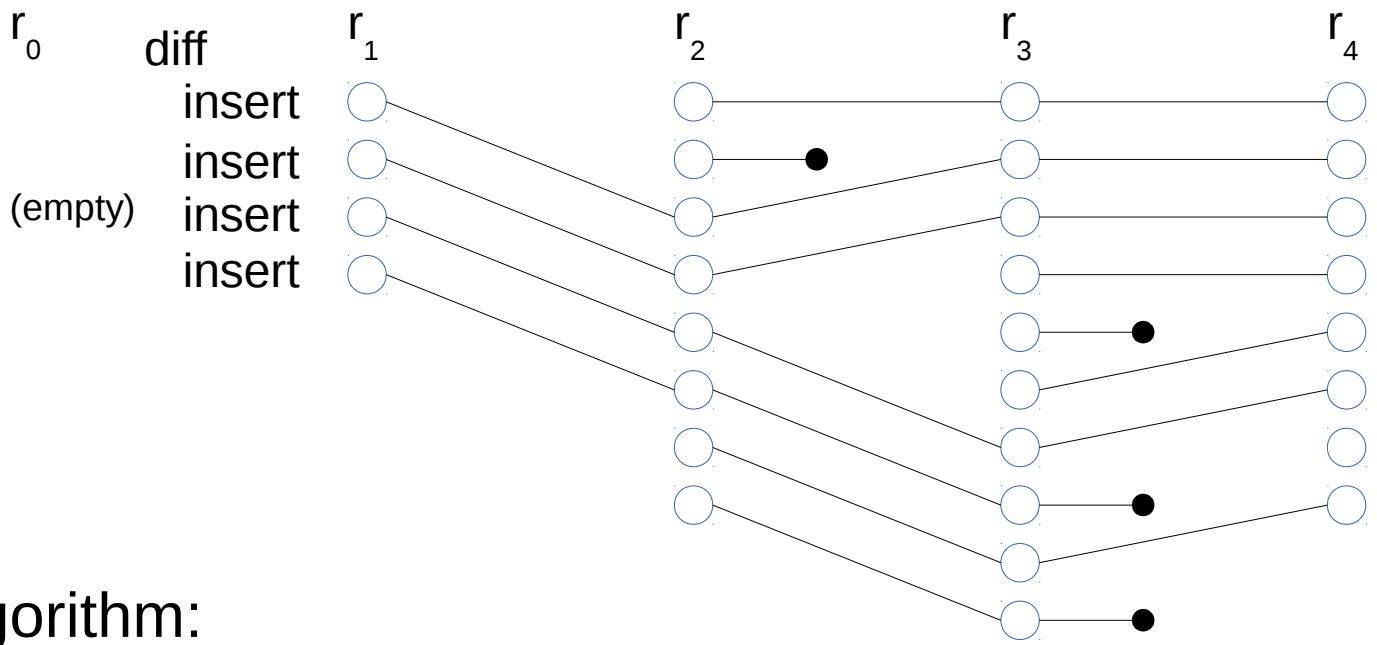
Forward Algorithm

Blame



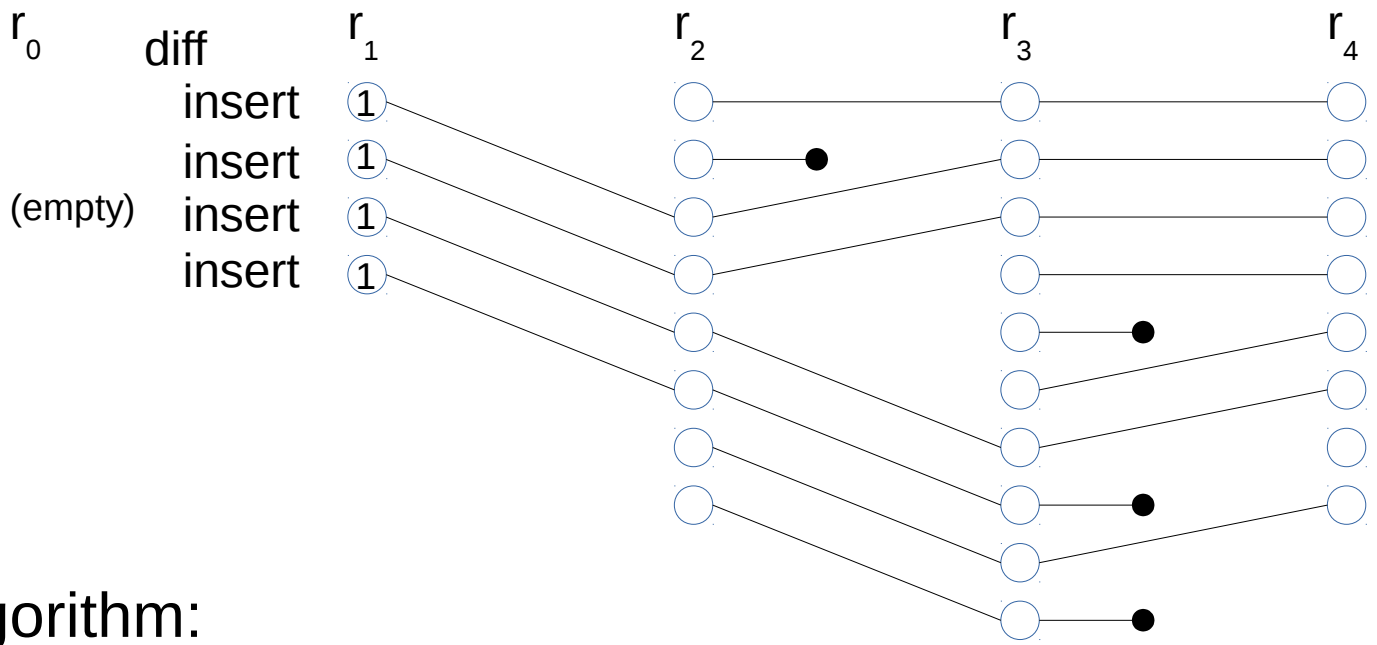
- Forward algorithm:
  - For all revisions (0...last):
    - Calculate diff of current and next revision
    - For all lines in the revision:
      - Assign new author to next node if the line was **insert**ed
      - Assign old author to next node if the line was **keep**ed

Blame



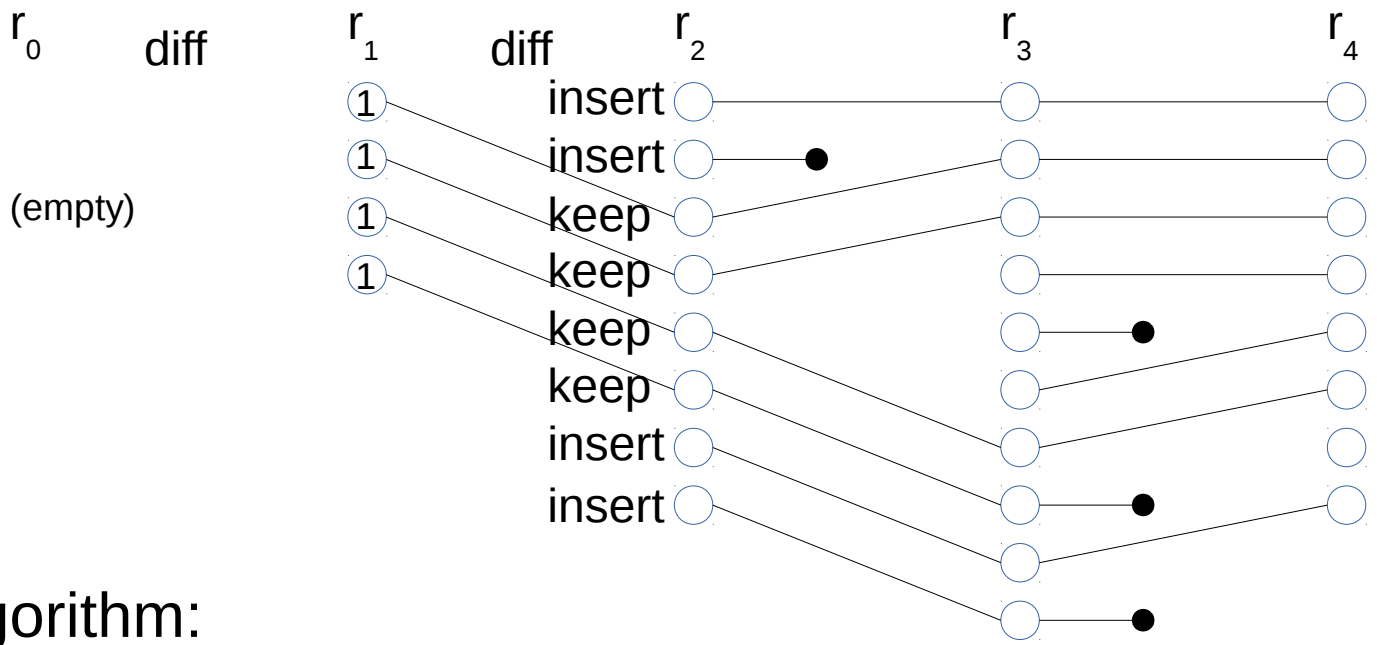
- Forward algorithm:
  - For all revisions (0...last):
    - Calculate diff of current and next revision
    - For all lines in the revision:
      - Assign new author to next node if the line was **insert**ed
      - Assign old author to next node if the line was **keep**ed

Blame



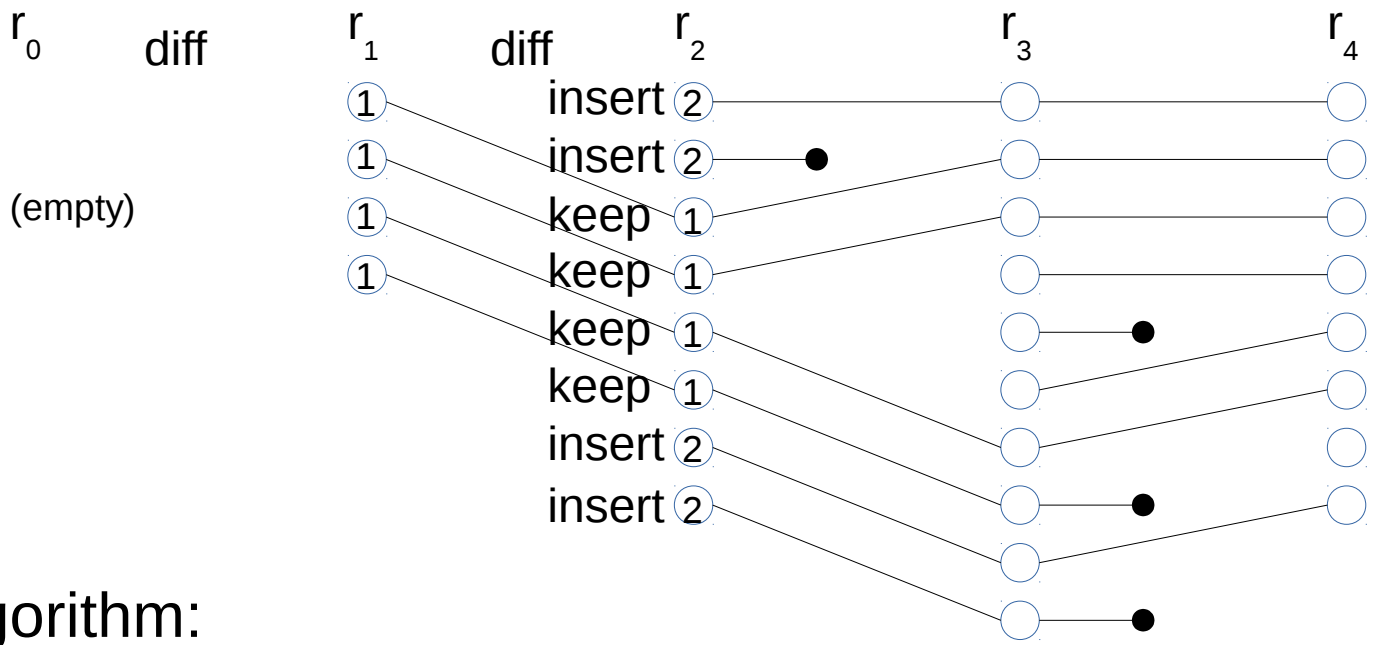
- Forward algorithm:
  - For all revisions (0...last):
    - Calculate diff of current and next revision
    - For all lines in the revision:
      - Assign new author to next node if the line was **insert**ed
      - Assign old author to next node if the line was **keep**ed

Blame



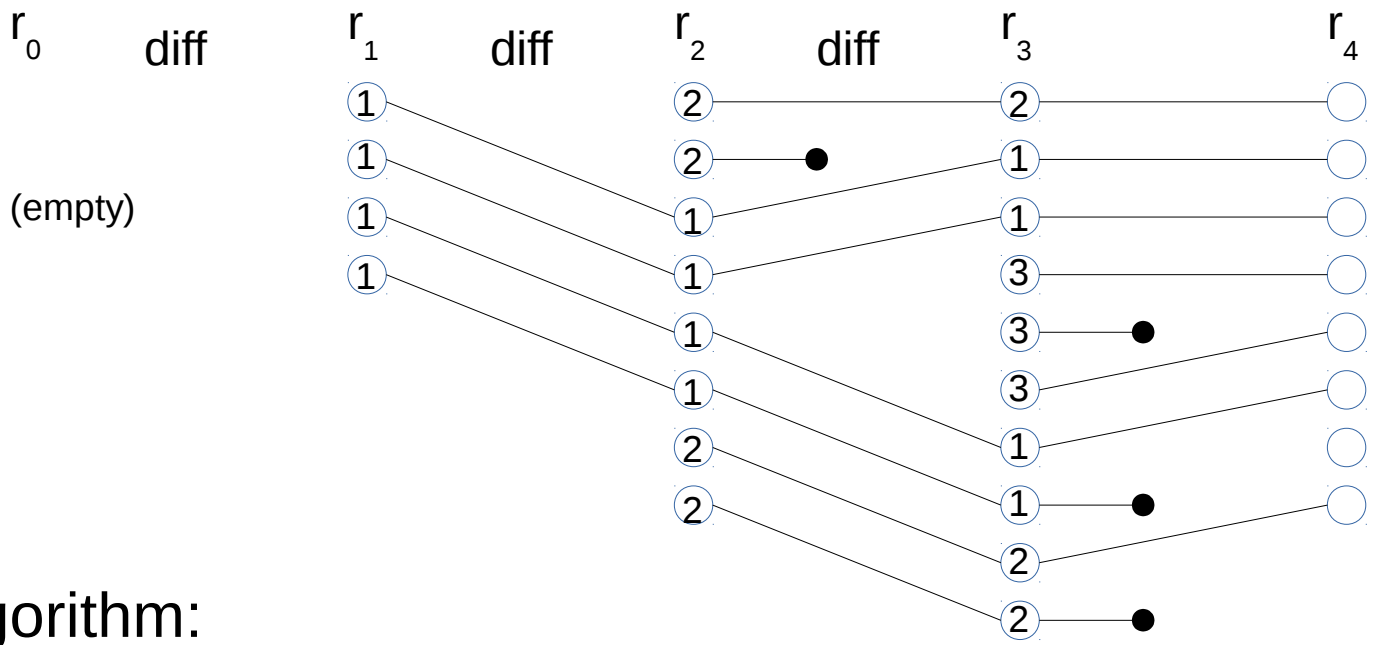
- Forward algorithm:
  - For all revisions (0...last):
    - Calculate diff of current and next revision
    - For all lines in the revision:
      - Assign new author to next node if the line was **insert**ed
      - Assign old author to next node if the line was **keep**ed

Blame



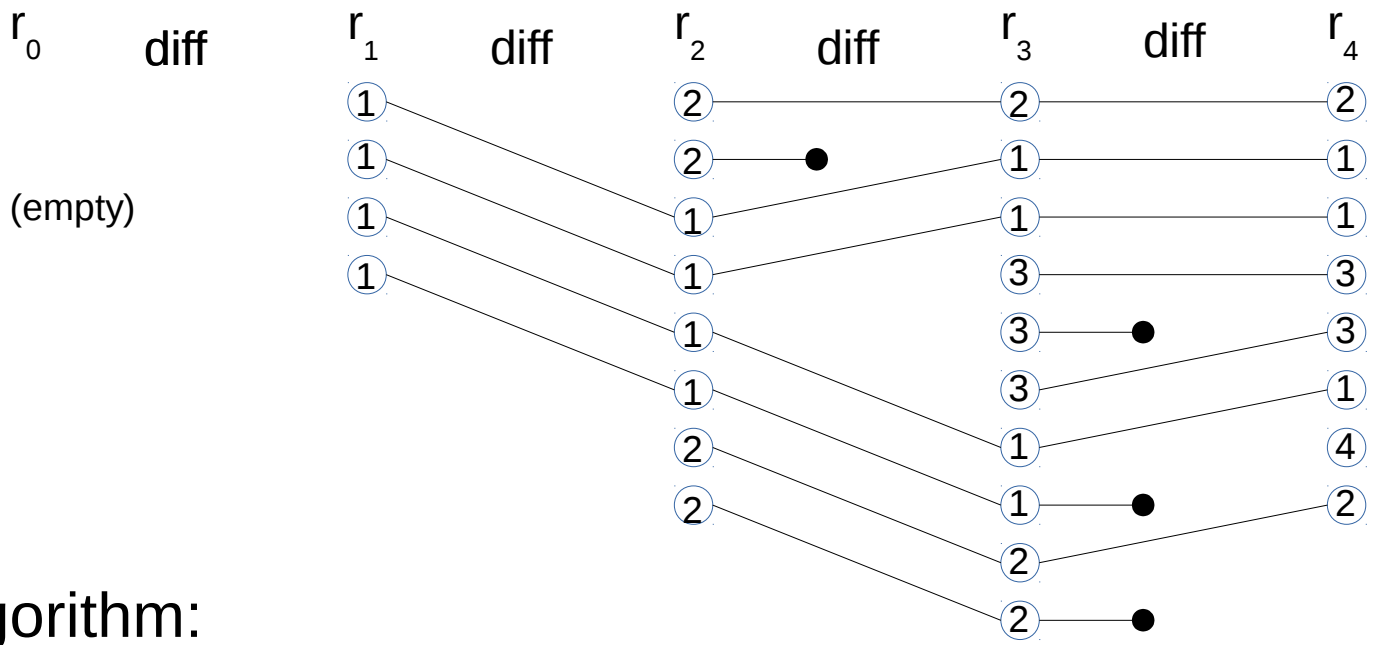
- Forward algorithm:
  - For all revisions (0...last):
    - Calculate diff of current and next revision
    - For all lines in the revision:
      - Assign new author to next node if the line was **insert**ed
      - Assign old author to next node if the line was **keep**ed

Blame



- Forward algorithm:
  - For all revisions (0...last):
    - Calculate diff of current and next revision
    - For all lines in the revision:
      - Assign new author to next node if the line was **insert**ed
      - Assign old author to next node if the line was **keep**ed

Blame

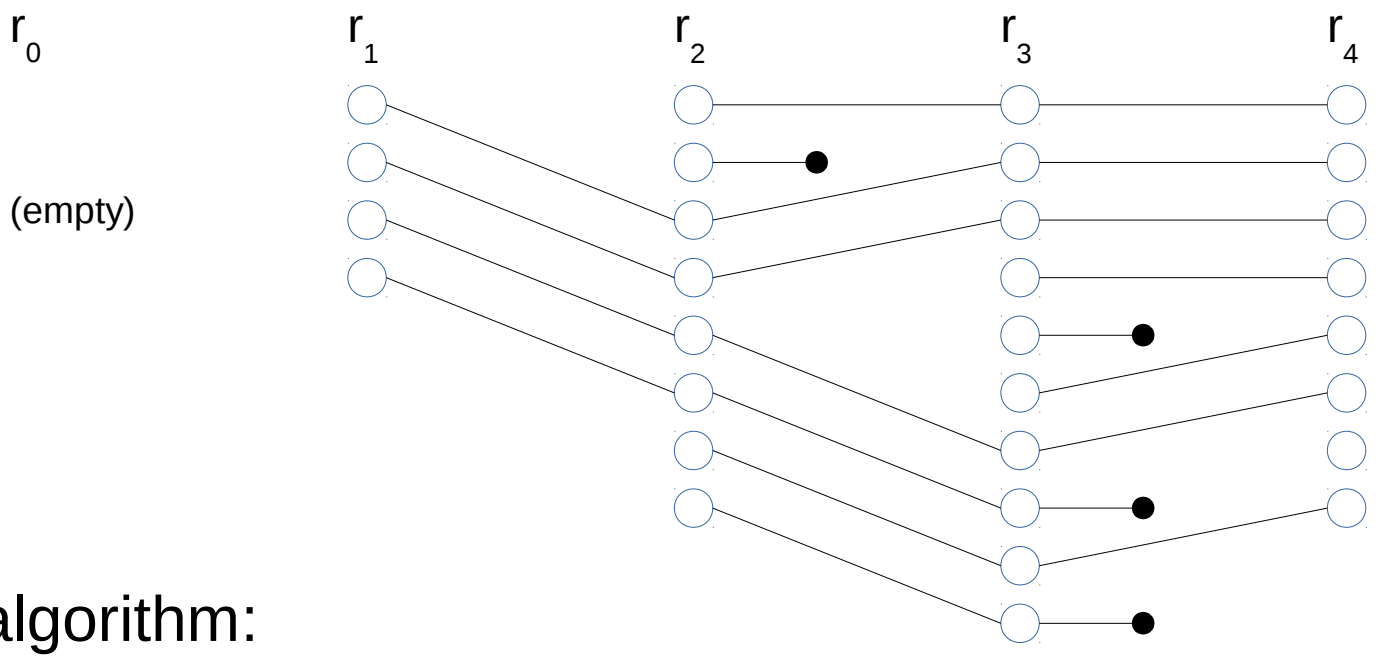


- Forward algorithm:
  - For all revisions (0...last):
    - Calculate diff of current and next revision
    - For all lines in the revision:
      - Assign new author to next node if the line was **insert**ed
      - Assign old author to next node if the line was **keep**ed

Blame

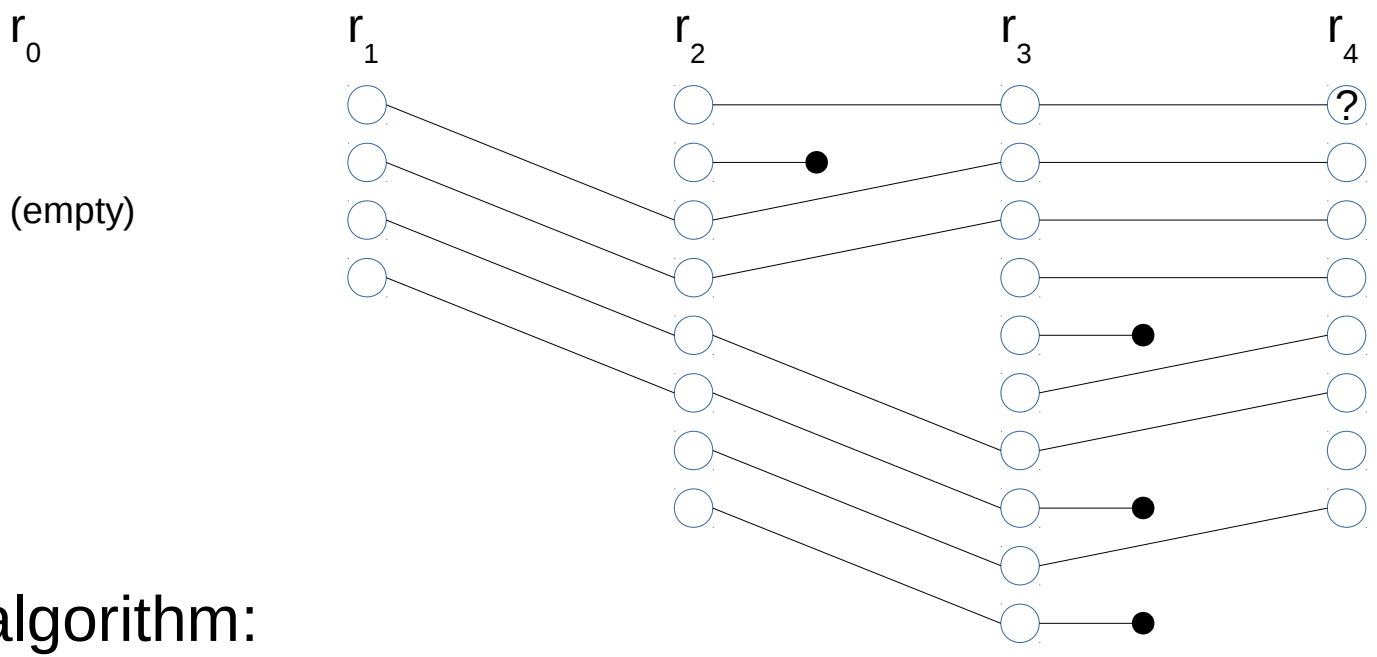
Backward Algorithm

Blame



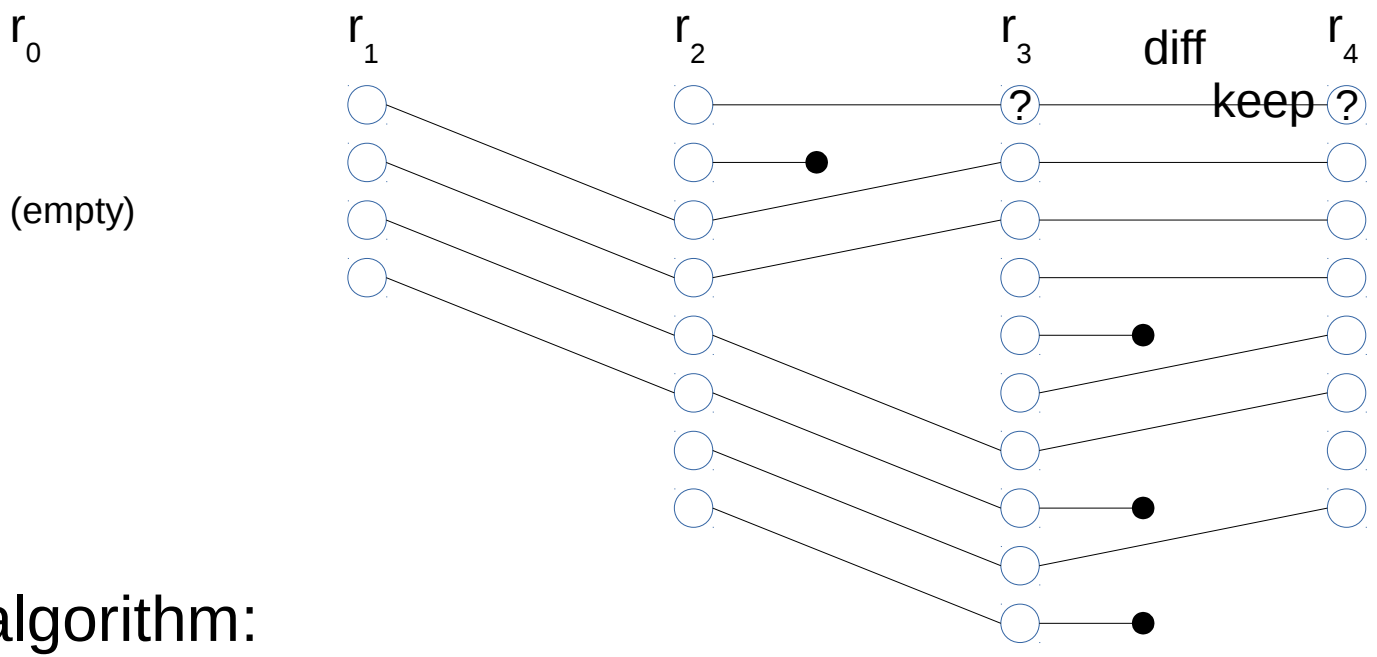
- Backward algorithm:
  - For all lines in last revision:
    - For all revisions
      - Calculate diff if it has not been already calculated
      - Trace the line back until an **insert** is found
        - Assign author to line

Blame



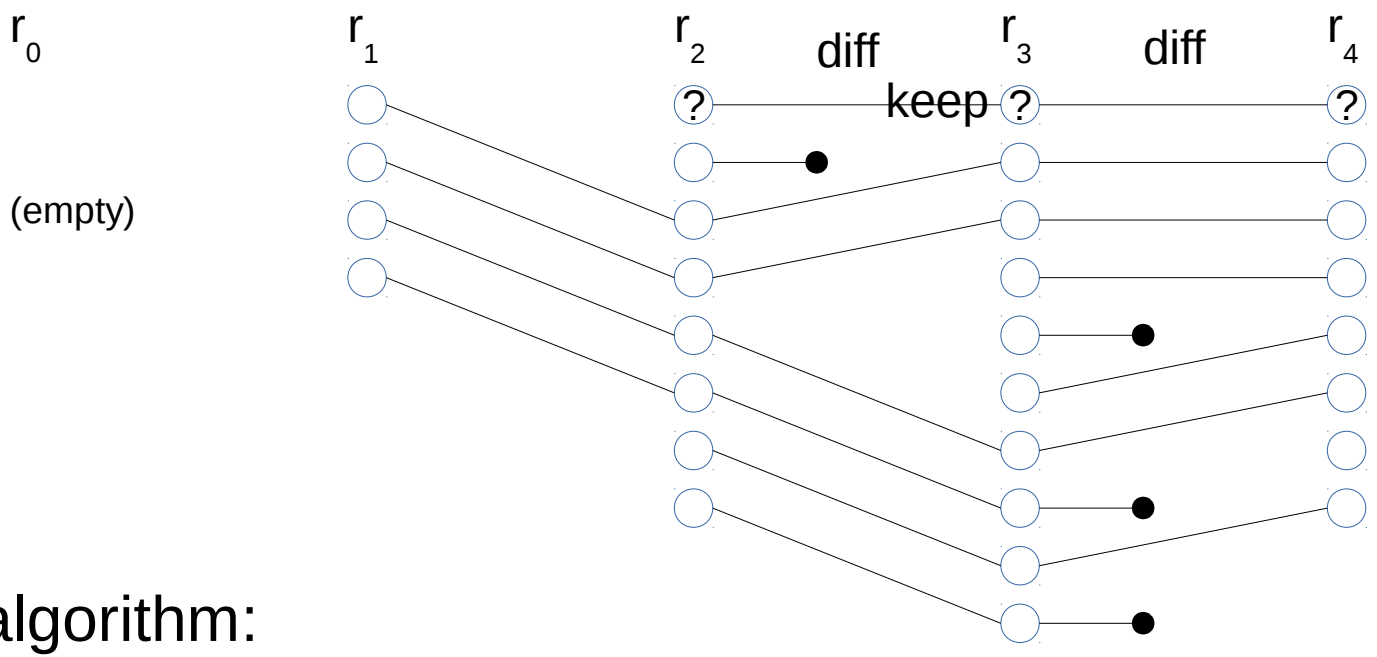
- Backward algorithm:
  - For all lines in last revision:
    - For all revisions
      - Calculate diff if it has not been already calculated
      - Trace the line back until an **insert** is found
        - Assign author to line

Blame



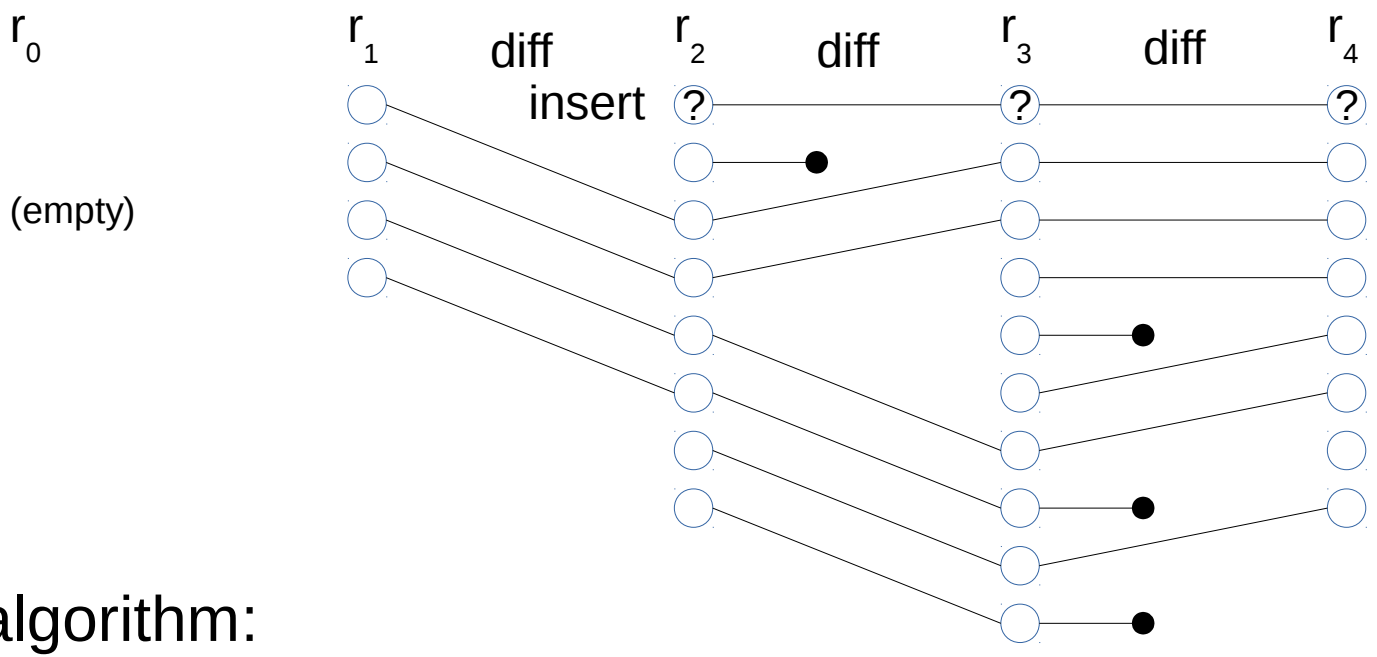
- Backward algorithm:
  - For all lines in last revision:
    - For all revisions
      - Calculate diff if it has not been already calculated
      - Trace the line back until an **insert** is found
        - Assign author to line

Blame



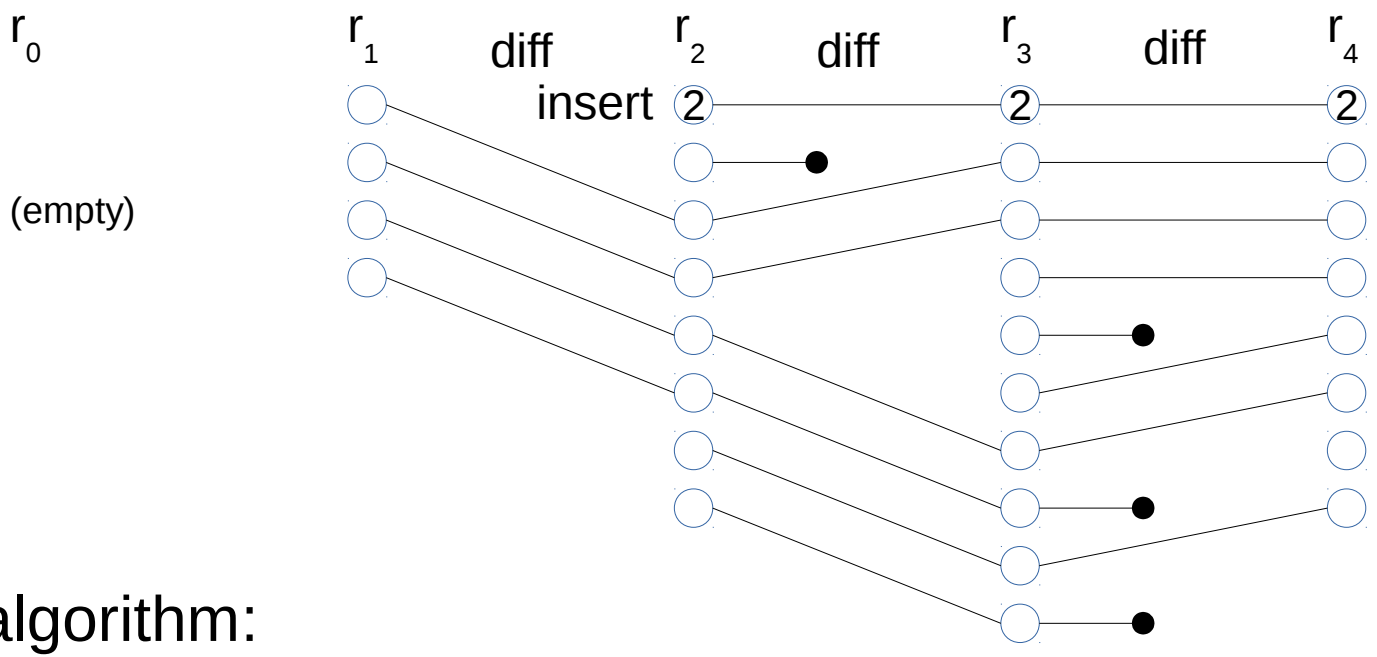
- Backward algorithm:
  - For all lines in last revision:
    - For all revisions
      - Calculate diff if it has not been already calculated
      - Trace the line back until an **insert** is found
        - Assign author to line

Blame



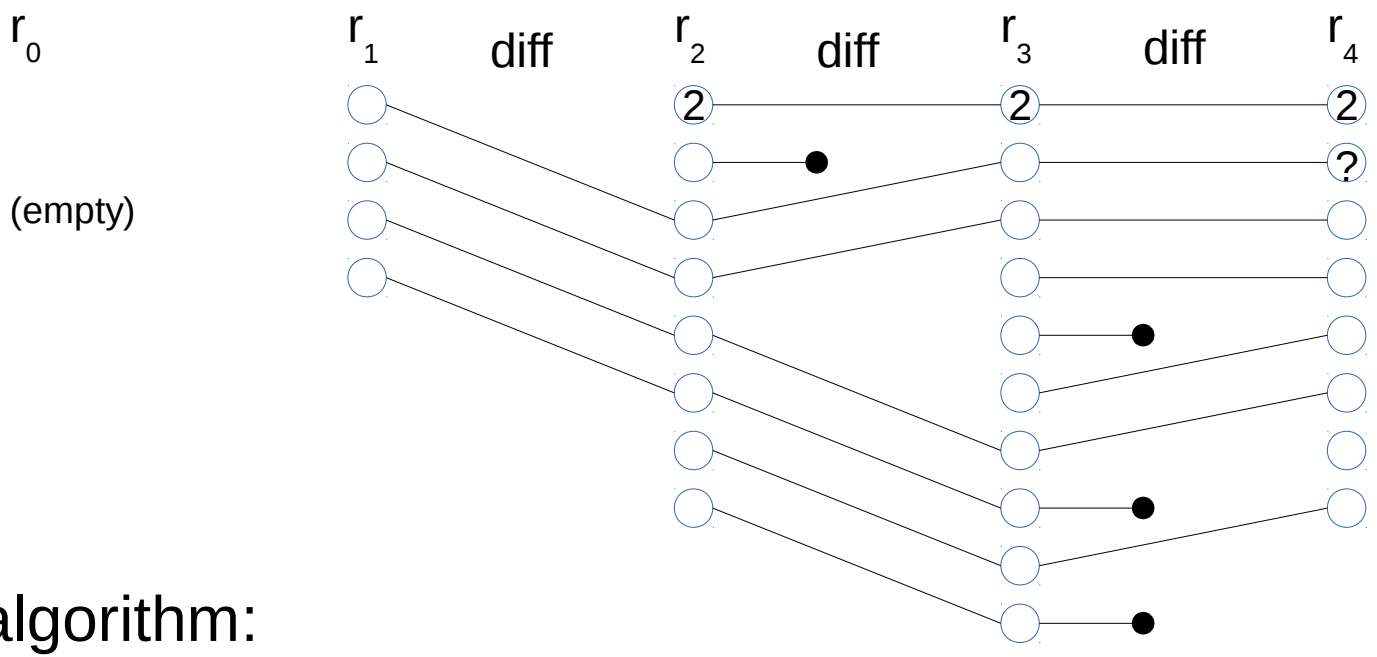
- Backward algorithm:
  - For all lines in last revision:
    - For all revisions
      - Calculate diff if it has not been already calculated
      - Trace the line back until an **insert** is found
        - Assign author to line

Blame



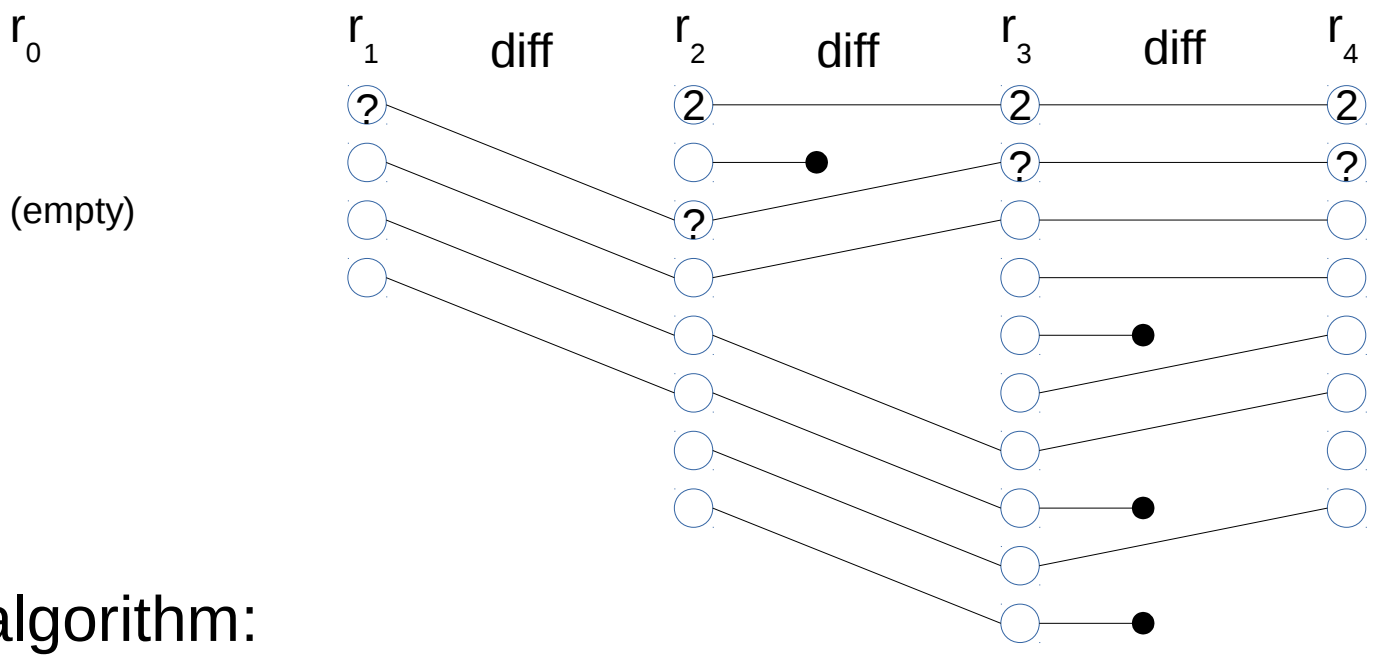
- Backward algorithm:
  - For all lines in last revision:
    - For all revisions
      - Calculate diff if it has not been already calculated
      - Trace the line back until an **insert** is found
        - Assign author to line

Blame



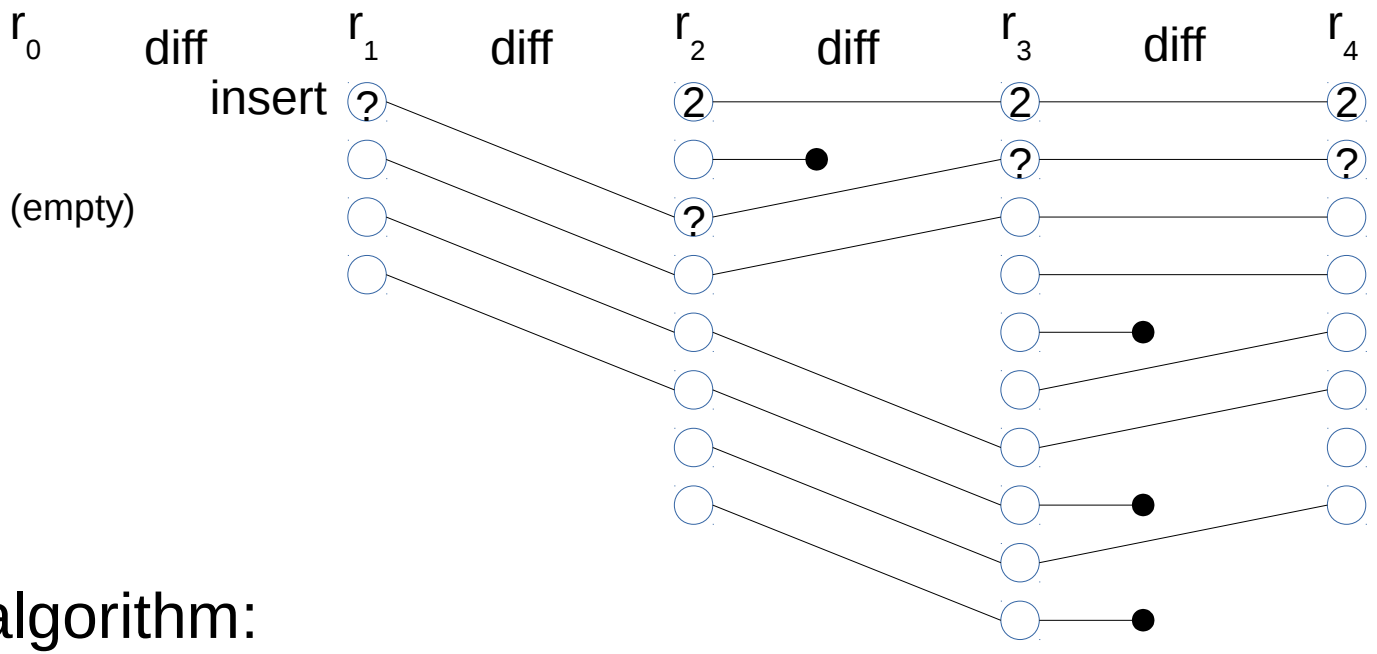
- Backward algorithm:
  - For all lines in last revision:
    - For all revisions
      - Calculate diff if it has not been already calculated
      - Trace the line back until an **insert** is found
        - Assign author to line

Blame



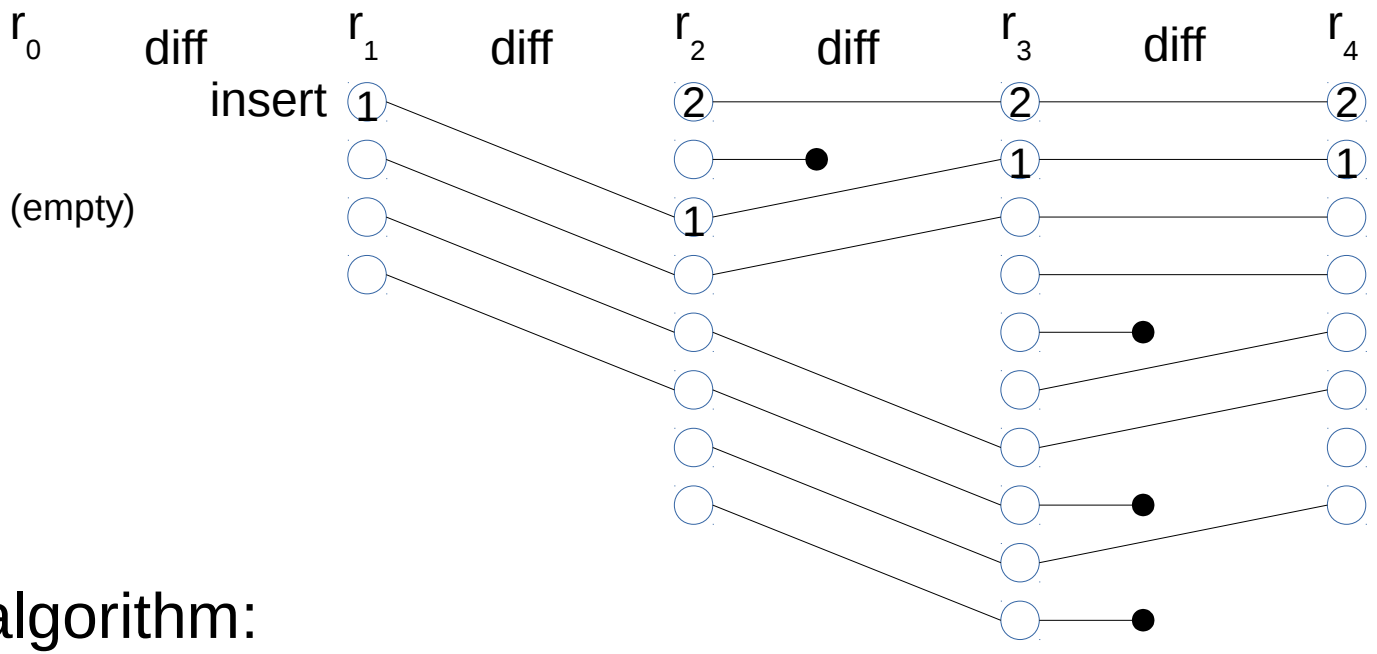
- Backward algorithm:
  - For all lines in last revision:
    - For all revisions
      - Calculate diff if it has not been already calculated
      - Trace the line back until an **insert** is found
        - Assign author to line

Blame



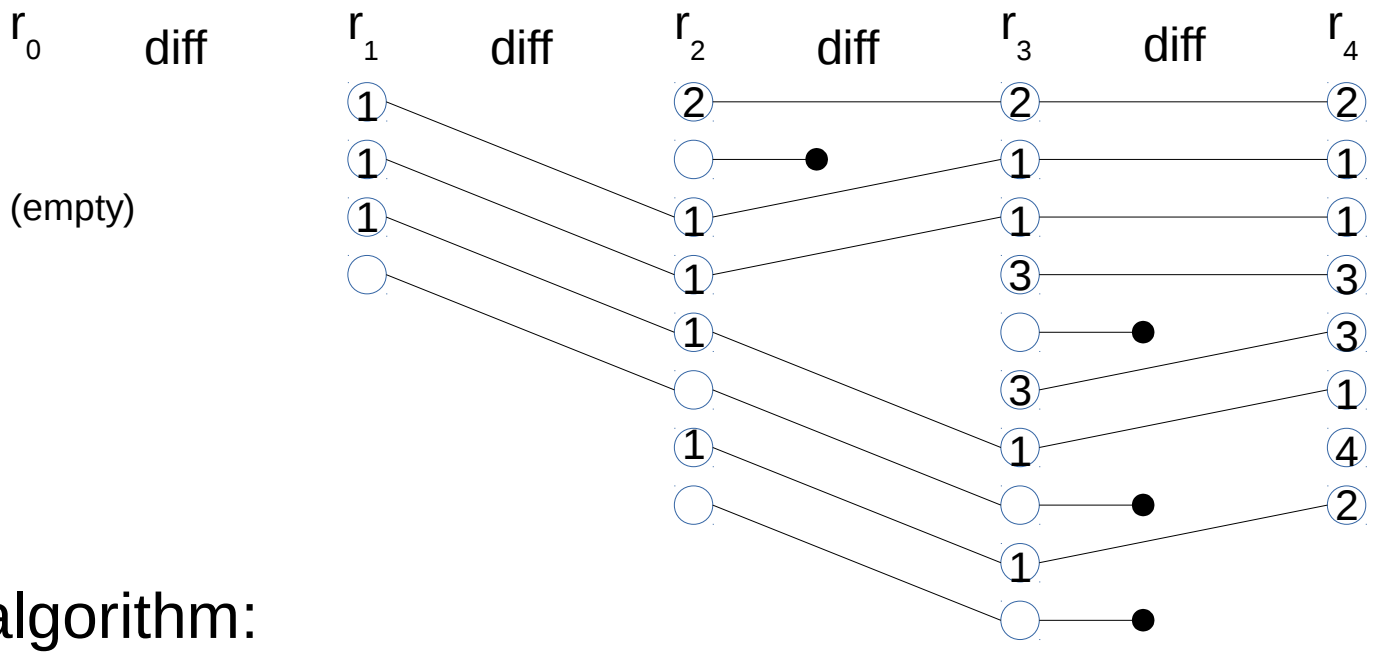
- Backward algorithm:
  - For all lines in last revision:
    - For all revisions
      - Calculate diff if it has not been already calculated
      - Trace the line back until an **insert** is found
        - Assign author to line

Blame



- Backward algorithm:
  - For all lines in last revision:
    - For all revisions
      - Calculate diff if it has not been already calculated
      - Trace the line back until an **insert** is found
        - Assign author to line

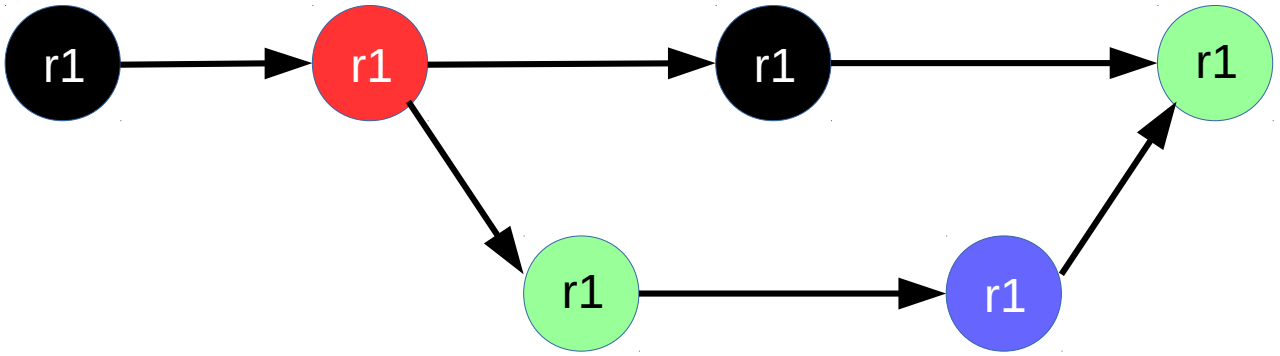
Blame



- Backward algorithm:
  - For all lines in last revision:
    - For all revisions
      - Calculate diff if it has not been already calculated
      - Trace the line back until an **insert** is found
        - Assign author to line

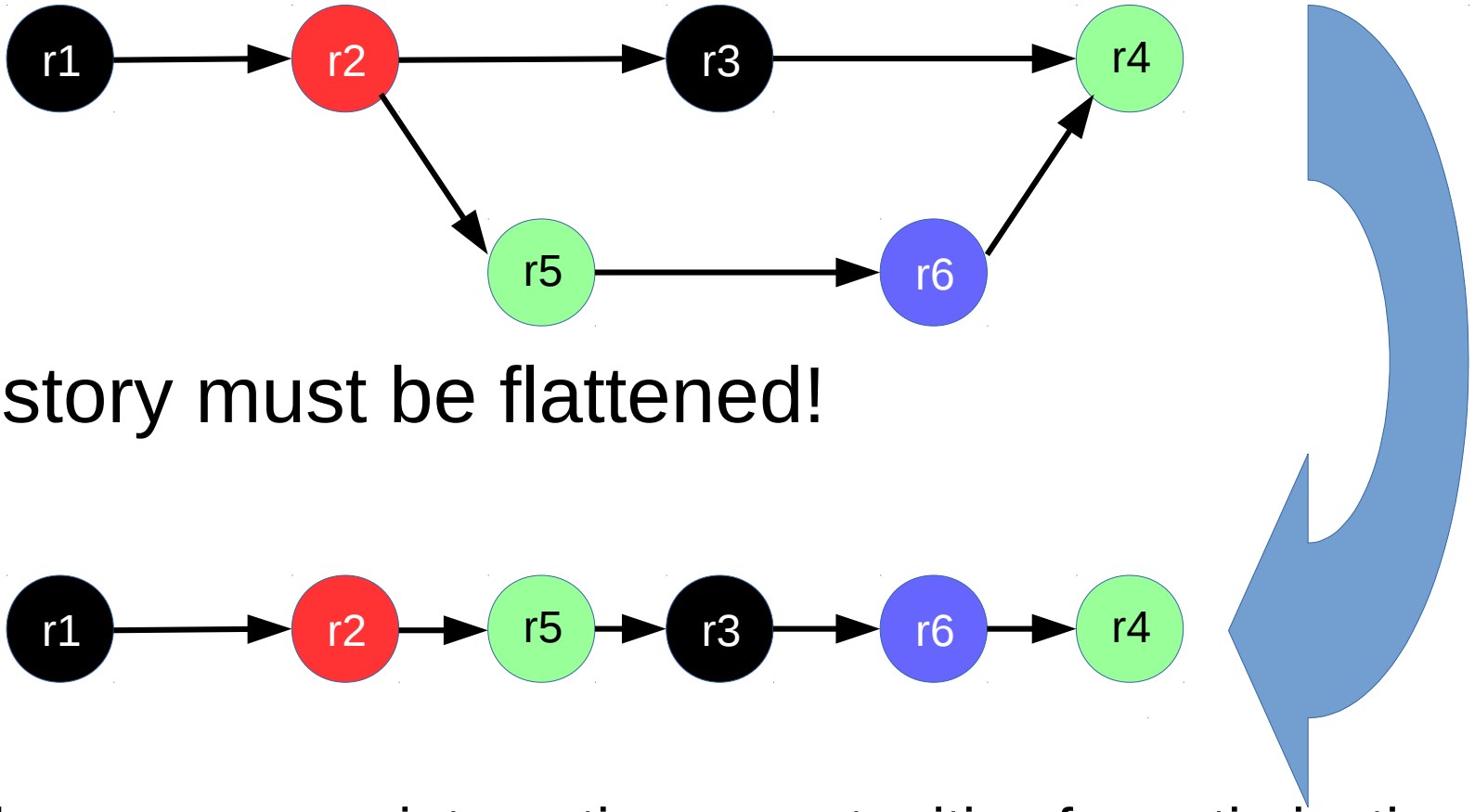
Blame

- Merges:



Blame

- Merges:



- History must be flattened!
  - there are some interesting opportunities for optimization in how you flatten the history

Blame

- Conclusions:
  - You must flatten the history
  - Build a graph with all the lines in all the revisions
  - Calculate the diff between (maybe) all revisions
  - Walk the graph building the edges and assigning authorship

Questions?