Fun with strings: Maximal Common Subsequences

What is a Subsequence?

It’s a new sequence created by deleting elements from the original sequence and keeping the relative order of the remaining elements

1 8 3 9 28 4 1 39 198324 2 43 2 42 99 -1

What is a Subsequence?

It’s a new sequence created by deleting elements from the original sequence and keeping the relative order of the remaining elements

1 8 3 9 28 4 1 39 198324 2 43 2 42 99 -1

What is a Subsequence?

It’s a new string created by deleting characters from the original string and keeping the relative order of the remaining characters

`sdfjlvasdjvaiuew`

What is a Subsequence?

It’s a new string created by deleting characters from the original string and keeping the relative order of the remaining characters

sdfjlvas**d**j**va**iu**e**w

What is a Subsequence?

`a is a Subsequence`

What is a Subsequence?

`uq isnt a Subsequence`

no crossing allowed!

Indeed

‘a’ is a subsequence of ‘What is a Subsequence?’

‘uq’ is not a subsequence of ‘What is a Subsequence?’

Subsequences are not necessarily contiguous

Substrings are contiguous

sdfjlvasdjvaiuew

Subsequences:

sdfjlvasdjvaiuew

Position doesn't matter in subsequences

Only the relative position of the characters is important

no crss

Only the relative position of the characters is important

The actual positions the subsequence maps to don't matter

^ called "embeddings" or "mappings"

The subsequence relation is transitive

$W \subset W' \subset W''$

$\Rightarrow W \subset W''$

e.g.: s ⊂ seqn ⊂ subsequence

=> s ⊂ subsequence

Then, what is a common subsequence?

Well, you need something to make it common with

It’s a subsequence that appears in multiple strings you are analyzing

Then, what is a common subsequence?

Well, you need something to make it common with

It’s a subsequence that appears in multiple strings you are analyzing

Let’s start easy with 2 strings

What is a Maximal Common Subsequence?

First, take two strings

Then, find a common subsequence

What is a Maximal Common Subsequence?

```
1. first, take two strings

2. then find a common subsequence
```

What is a Maximal Common Subsequence?

1. first, take two strings

2. then find a common subsequence

What is a Maximal Common Subsequence?

1. first, take two strings

2. then find a common subsequence

What is a Maximal Common Subsequence?

1. first, take two strings

2. then find a common subsequence

-> fian

This is a common subsequence; is it maximal?

What is a Maximal Common Subsequence?

1. first, take two strings

2. then find a common subsequence

-> fian

This is a common subsequence; is it maximal? No: fiaon

Maximality

Subsequence W of X is maximal if it is not subsequence of other subsequences

Maximality

Subsequence W of X is maximal if it is not subsequence of other subsequences

$\texttt{fian} \subset \texttt{fiaon}$

Maximality is subtle

Is it `adcc` maximal?

`abdcacd`

`badbcdcc`

Maximality is subtle

Is it `adcc` maximal?

no!

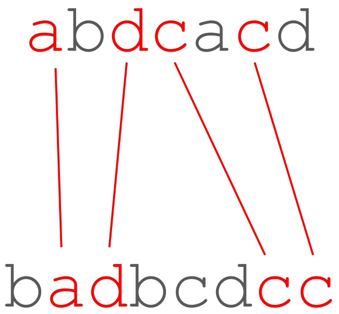
`abdcacd`

`badbcdcc`

Maximality is subtle

Is it `adcc` maximal?

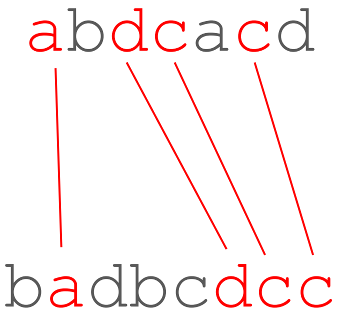
no!



Maximality is subtle

Is it `adcc` maximal?

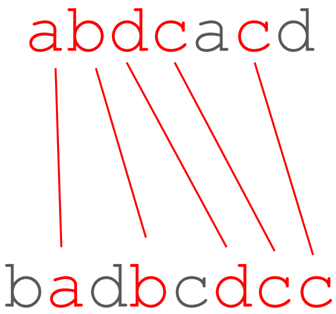
no!



Maximality is subtle

Is it `adcc` maximal?

no! `abdcc`



Maximality is weird

Is it `e` maximal?

`eabdcacd`

`badbcdcce`

Maximality is weird

Is it e maximal?

Yes!!

```
eabdcacd

badbcdcce
```

Our problem

Find all MCS between two strings X and Y

Trivial algorithm for finding all MCSs

1. Find all common subsequences
2. Filter out non-maximal ones by applying the definition

Trivial algorithm for finding all MCSs

1. Find all common subsequences (quite a lot!)
2. Filter out non-maximal ones (by checking if they are subseq. of other subseq.)

Trivial algorithm for finding all MCSs

1. Find all common subsequences (quite a lot! Is it feasible?)
2. Filter out non-maximal ones (by checking if they are subseq. of other subseq.)

Can we do better?

to be continued…

Sanity check

Questions? You still with me?

Longest Common Subsequences

Let’s take a detour

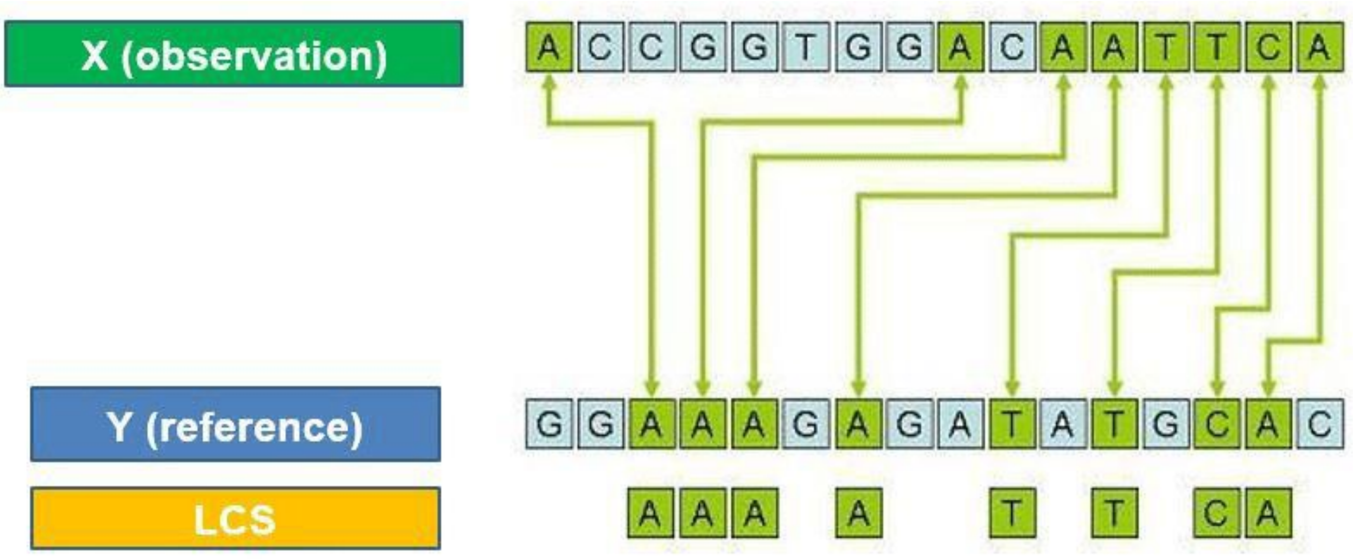
Easy definition:

the set of common subsequences whose length is maximum

The LCS problem consists in finding the length of an LCS

Use cases

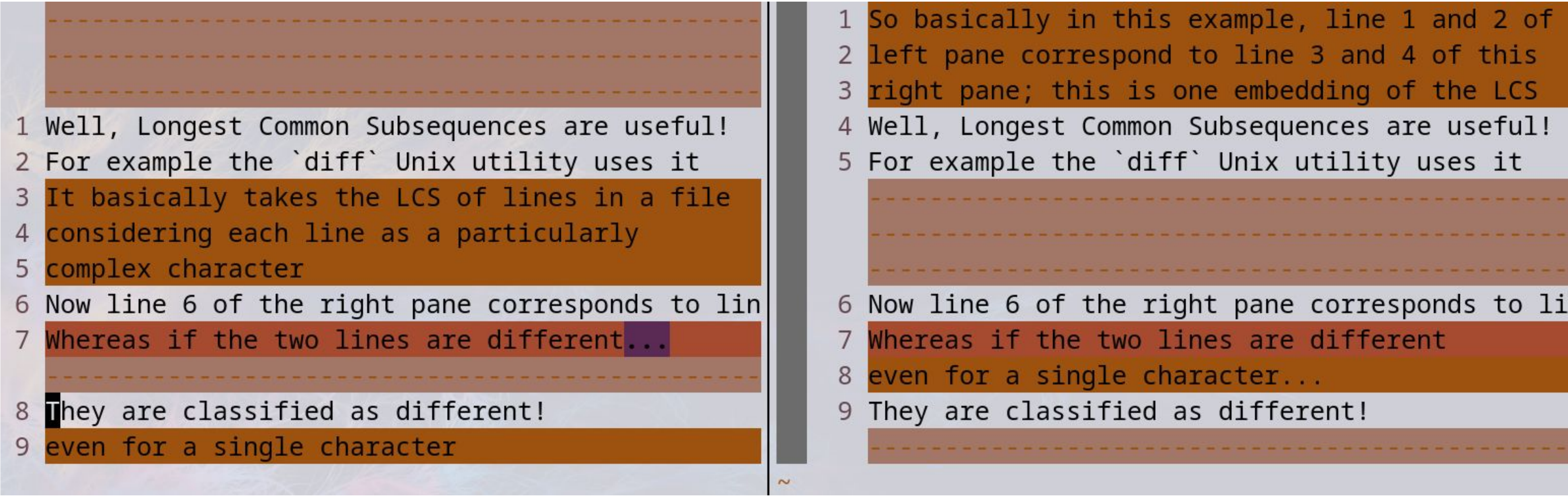
DNA sequence alignment



https://doi.org/10.1787/19939019

Use cases

diff



Use cases

DNA sequence alignment

diff

Spelling error correction

Plagiarism detection

…

$LCS \subseteq MCS$

If a common subsequence is longest

then it is maximal

$$LCS \subseteq MCS$$

If a common subsequence is longest then it is maximal

can you see why?

$LCS \subseteq MCS$

If a common subsequence is longest then it is maximal

Let $W \in LCS(X, Y)$, suppose by contradiction W is not maximal

Then there exists some $c \in \Sigma$ and some $i \in [|W|]$: $W' = W[0, i) \cdot c \cdot W[i, |W|)$

is still a subsequence of X and Y

But $|W'| = |W| + 1$

A contradiction, as we supposed $W \in LCS(X, Y)$

$LCS \leq MCS$

The LCS problem reduces to the MCS problem

If we find all MCS we can list all LCS (keep the ones with maximum length)

$LCS \leq MCS$

The LCS problem reduces to the MCS problem

If we find all MCS we can list all LCS (keep the ones with maximum length)

… and of course know the maximal length

How do we compute one LCS for two strings?

Classical dynamic programming approach O(mn) where m=|X|, n=|Y|

How do we compute one LCS for two strings?

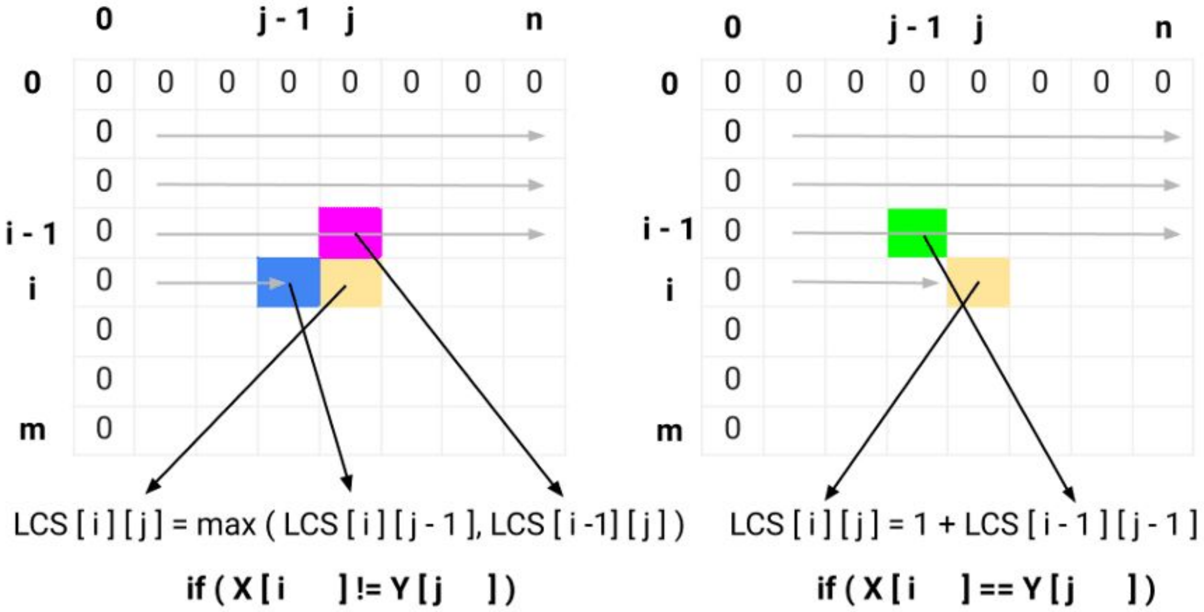
Classical dynamic programming approach O(mn) where m=|X|, n=|Y|

		0	1	2	3	4	5	6	7
Y[ ] ↓	X[ ] →	Ø	M	Z	J	A	W	X	U
0	Ø	0	0	0	0	0	0	0	0
1	X	**0**	0	0	0	0	0	1	1
2	M	0	**1**	**1**	1	1	1	1	1
3	J	0	1	1	**2**	2	2	2	2
4	Y	0	1	1	**2**	2	2	2	2
5	A	0	1	1	2	**3**	**3**	**3**	3
6	U	0	1	1	2	3	3	3	**4**
7	Z	0	1	2	2	3	3	3	**4**

https://www.enjoyalgorithms.com/blog/longest-common-subsequence

How do we compute one LCS for two strings?

Classical dynamic programming approach O(mn) where m=|X|, n=|Y|



https://www.enjoyalgorithms.com/blog/longest-common-subsequence

How many LCS are there?

let $t = |X| + |Y|$

let $0 < |\Sigma| \leq t$

let $D(t) = max_{X,Y}|LCS(X, Y)|$

$$\boxed{\begin{array}{c} D(t) > 1.2^t \\ D(t) < 1.32^t \end{array}}$$

(if t mod 6 = 0)

Exponential even for two strings X and Y!!

https://arxiv.org/abs/cs/0301030

How many LCS embeddings are there?

let $t = |X| + |Y|$

let $0 < |\Sigma| \leq t$

let $D(t) = max_{X,Y} |LCS(X, Y)|$

$$\boxed{\begin{array}{c} D(t) > 1.2^t \\ D(t) < 1.32^t \end{array}}$$

(if t mod 6 = 0)

The number of embeddings is even greater!

https://arxiv.org/abs/cs/0301030

Since $LCS \subseteq MCS$

Also the number of distinct MCS is exponential even for two strings!!

Complexity

LCS problem

Given a set of strings S, print the length of a LCS of S

For k strings

The LCS problem is NP-Hard [Maier 1978]

For 2 strings

The LCS problem has a conditional lower bound of $O(n^2)$ [Abboud et al. 2015]

https://doi.org/10.1145/322063.322075

https://doi.org/10.1109/FOCS.2015.14

Complexity

LCS problem

Given a set of strings S, print the length of a LCS of S

For 2 strings

The LCS problem has a conditional lower bound of $O(n^2)$ [Abboud et al. 2015]

-> based on the Strong Exponential Time Hypothesis (SETH).

-> it states that $\lim_{k\to\infty} s_k = 1$, where $s_k = \inf\{\delta | k\text{-SAT can be solved in } O(2^{\delta n}) \text{ time}\}$.

https://doi.org/10.1109/FOCS.2015.14

Enough theory

Let’s play with some example

Back to our plan

Too many!

1. ~~Find all common subsequences (quite a lot! Is it feasible?)~~
2. Filter out non-maximal ones (~~by checking if they are subseq. of other subseq.~~)

We cannot do it without knowing the other subsequences

How do we check maximality?

If we don’t have other subsequences?

How do we check maximality?

The curtain algorithm

https://www.interior-images.ca/theater-curtains/

How do we check maximality?

The curtain algorithm

Sorry, low budget

https://www.greenme.it/casa-e-giardino/diy-upcycling/10-tende-interne-riciclo-creativo/

The curtain algorithm

Take any input embedding

The curtain algorithm

Take any input embedding

Make it leftmost*

The curtain algorithm

Take any input embedding

Make it leftmost*

*the embedding of W is leftmost if its last match is (g_W, h_W) where $X[0, g_W)$ and $Y[0, h_W)$ are the shortest prefixes of X and Y that contain W

The curtain algorithm

Take any input embedding

Make it leftmost*

*the embedding of W is leftmost if its last match is (g_W, h_W) where $X[0, g_W)$ and $Y[0, h_W)$ are the shortest prefixes of X and Y that contain W

-> the definition of rightmost is analogous and uses the shortest suffixes

The curtain algorithm

Take any input embedding

Make it leftmost

Make it rightmost

one piece at a time

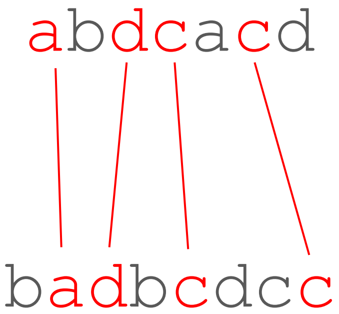
and check for insertions

(the substrings in between should be non-overlapping)

The curtain algorithm

Is it `adcc` maximal?

Take any input embedding

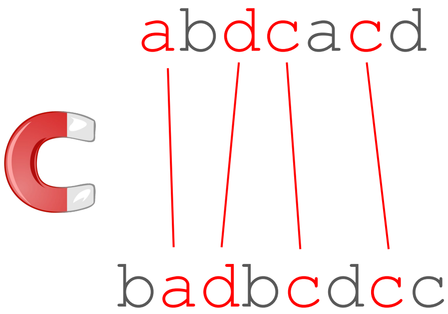


The curtain algorithm

Is it adcc maximal?

Take any input embedding

Make it leftmost



The curtain algorithm

Is it adcc maximal?

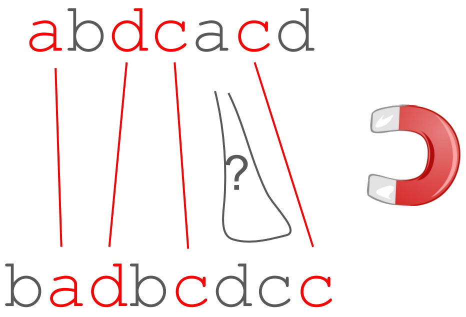
Take any input embedding

Make it leftmost

Make it rightmost

one piece at a time

and check for insertions



The curtain algorithm

Is it `adcc` maximal?

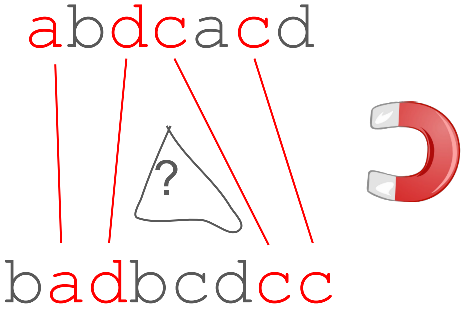
Take any input embedding

Make it leftmost

Make it rightmost

one piece at a time

and check for insertions



The curtain algorithm

Is it adcc maximal?

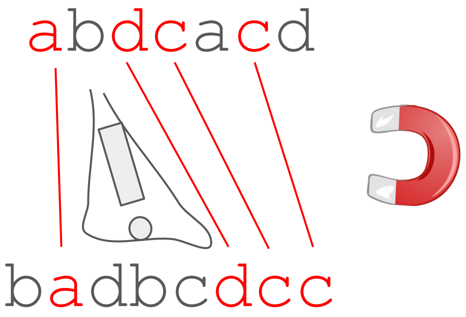
Take any input embedding

Make it leftmost

Make it rightmost

one piece at a time

and check for insertions



The curtain algorithm

Is it `adcc` maximal?

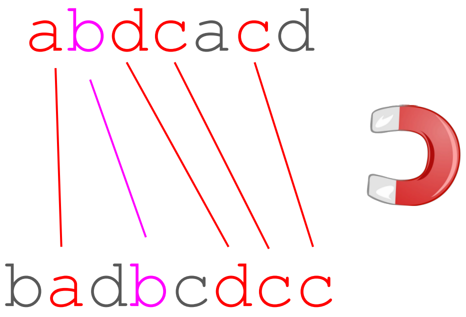
Take any input embedding

Make it leftmost

Make it rightmost

one piece at a time

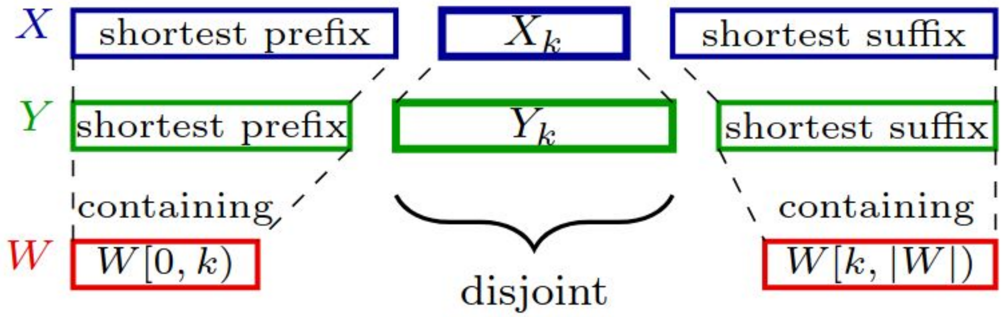
and check for insertions



The curtain algorithm

Take any input embedding

Make it leftmost



Make it rightmost

one piece at a time

https://doi.org/10.1007/s00453-021-00898-5

and check for insertions $W \in MCS(X, Y) \Leftrightarrow \forall k \in [|W|], X_k \cap Y_k = \emptyset$

(the substrings in between should be non-overlapping);

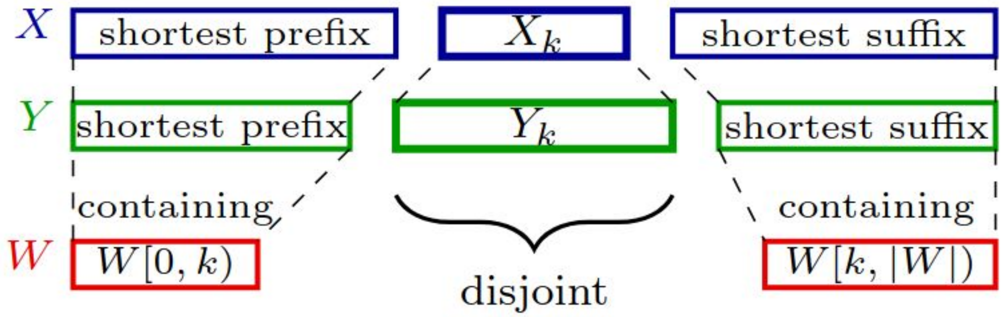
The curtain algorithm - why does it work?

$$W \in MCS(X,Y) \Leftrightarrow \forall k \in [|W|], X_k \cap Y_k = \emptyset$$

(=>) contrapositive

Suppose $\exists k^* \in [|W|],$

$$c \in \Sigma : c \in X_{k^*} \cap Y_{k^*}$$



https://doi.org/10.1007/s00453-021-00898-5

The curtain algorithm - why does it work?

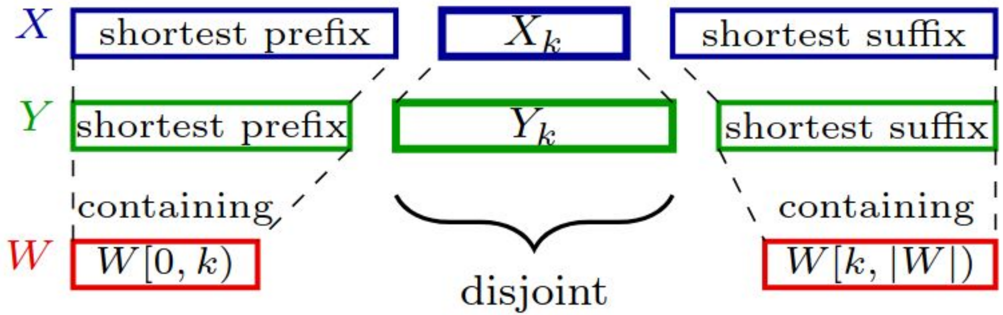
$$W \in MCS(X,Y) \Leftrightarrow \forall k \in [|W|], X_k \cap Y_k = \emptyset$$

(=>) contrapositive

Suppose $\exists k^* \in [|W|],$

$$c \in \Sigma : c \in X_{k*} \cap Y_{k*}$$

then $W[0,k^*) \cdot c \cdot W[k^*, |W|)$
is common subsequence



https://doi.org/10.1007/s00453-021-00898-5

The curtain algorithm - why does it work?

$$W \in MCS(X,Y) \Leftrightarrow \forall k \in [|W|], X_k \cap Y_k = \emptyset$$

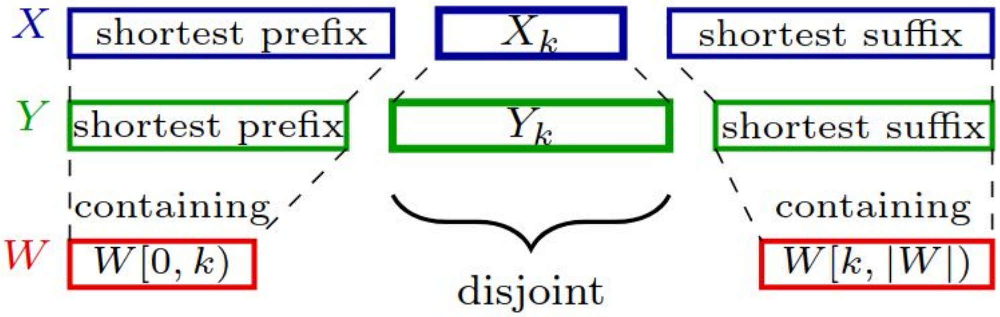
(=>) contrapositive

Suppose $\exists k^* \in [|W|],$

$$c \in \Sigma : c \in X_{k^*} \cap Y_{k^*}$$

then $W[0,k^*) \cdot c \cdot W[k^*,|W|)$
is common subsequence

But $W \subset W[0,k^*) \cdot c \cdot W[k^*,|W|)$
=> $W \notin MCS(X,Y)$



https://doi.org/10.1007/s00453-021-00898-5

How can we generate all distinct MCS?

How can we generate even one MCS?

How can we generate all distinct MCS?

How can we generate even one MCS?

-> There's an algorithm for this [Sakai 2018]

-> Not easily extendable to our problem

-> $O(n\sqrt{\log n / \log\log n})$

-> $(n = \max(|X|, |Y|))$

https://doi.org/10.1016/j.tcs.2019.06.020

Can you find one MCS?

What's the simplest way?

```
abcacd
```

```
badcda
```

Can you find one MCS?

Let’s try a greedy approach

```
abcacd
```

```
badcda
```

Can you find one MCS?

Let's try a greedy approach

Read the top string

left-to-right

find matches

abcacd

badcda

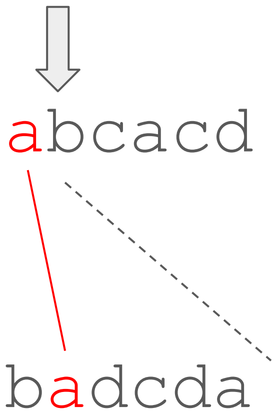
Can you find one MCS?

Let’s try a greedy approach

Read the top string

left-to-right

find matches



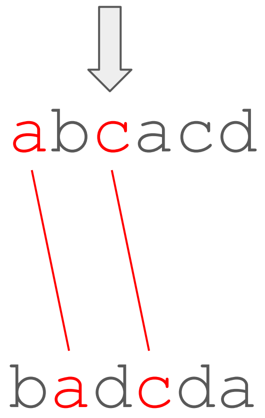
Can you find one MCS?

Let's try a greedy approach

Read the top string

left-to-right

find matches



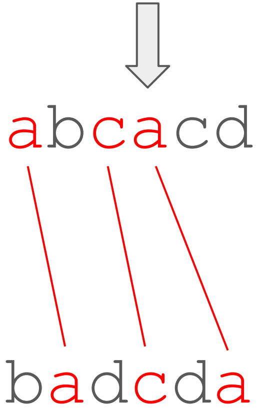
Can you find one MCS?

Let’s try a greedy approach

Read the top string

left-to-right

find matches



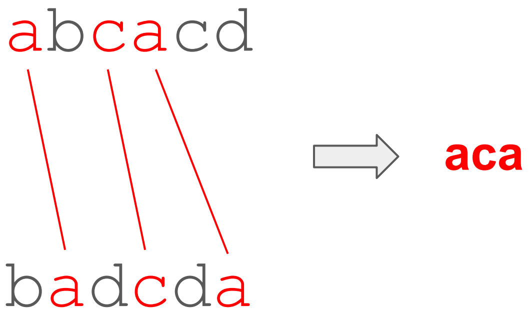
Can you find one MCS?

Let’s try a greedy approach

Read the top string

left-to-right

find matches



Can you find one MCS?

Let's try a greedy approach

Read the top string

left-to-right

find matches

abcacd

badcda

$\Rightarrow$ **aca**

Is this maximal?

Can you find one MCS?

Let’s try a greedy approach

Read the top string

left-to-right

find matches

abcacd

badcda

⇨ **aca**

Is this maximal?

We can check!

Checking maximality

Take any input embedding

`abcacd`

`badcda`

Checking maximality

Take any input embedding

Make it leftmost

abcacd

badcda

Checking maximality

Take any input embedding

Make it leftmost

Done by construction

abcacd

badcda

Checking maximality

Take any input embedding

Make it leftmost

Done by construction

abcacd

badcda

Make it rightmost

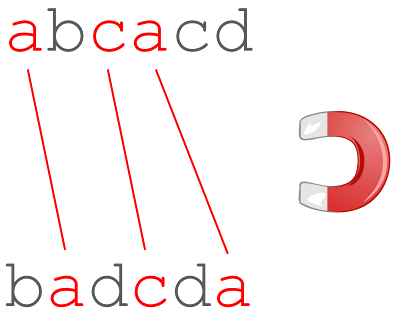
one piece at a time and check for insertions

Checking maximality

Take any input embedding

Make it leftmost

Done by construction



Make it rightmost

one piece at a time and check for insertions

It's also rightmost! -> No possible insertions

Checking maximality

Take any input embedding

Make it leftmost

Done by construction

abcacd

badcda

Make it rightmost

one piece at a time and check for insertions

It’s also rightmost! -> No possible insertions -> It is maximal!

“aca” is Maximal

Good job everyone!

MCS problem has a greedy solution

Seminar’s over

Open the chips

Not so fast

Read the top string

left-to-right

find matches

```
dcacab

bdacda
```

Not so fast

Read the top string

left-to-right

find matches

dcacab

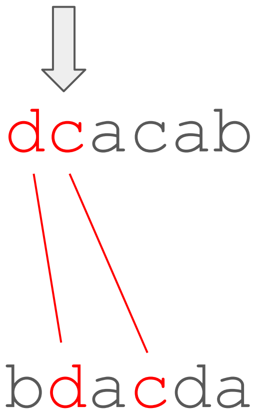
bdacda

Not so fast

Read the top string

left-to-right

find matches

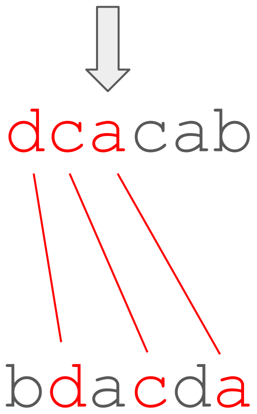


Not so fast

Read the top string

left-to-right

find matches

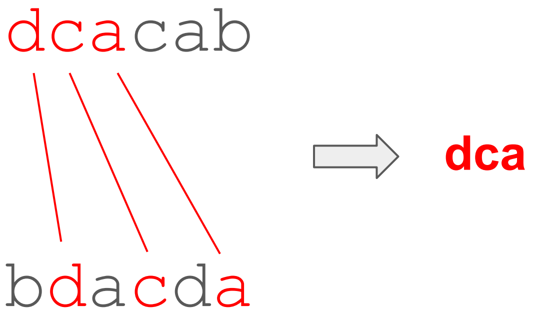


Not so fast

Read the top string

left-to-right

find matches



Is this maximal?

Not so fast

dcacab

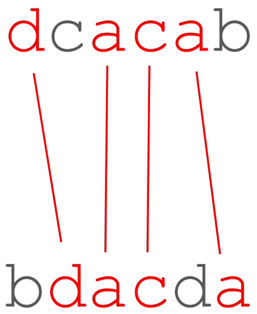
bdacda

⇒ **dca**

Is this maximal?

Not really! daca

Not so fast



$dca \subset daca$

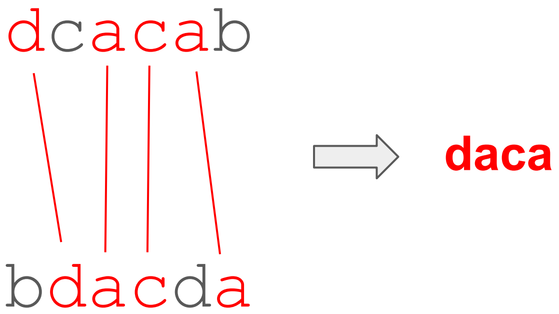
Is this maximal?

Not really! daca

What can we do?

Greedy doesn't work

Why?



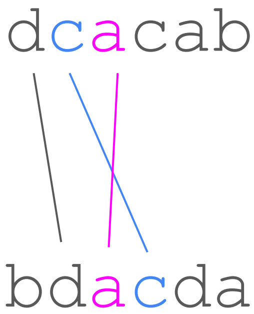
What can we do?

Greedy doesn't work

Why?

When choosing this

we ignored this



Idea1: We shouldn't choose c if it one end can be shifted to insert another char

Idea2: Maybe we should keep all possible embeddings found so far

Idea1

We shouldn’t choose a match if it one end can be shifted to insert another char

abadcbc

abbcdc

ab is the prefix of

abdc

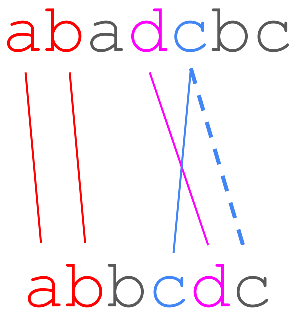
which is an MCS

Idea1

We shouldn’t choose a match if it one end can be shifted to insert another char

One end of c can be shifted

to insert d



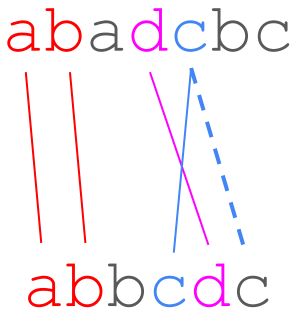
Idea1

We shouldn’t choose a match if it one end can be shifted to insert another char

One end of c can be shifted

to insert d

This is not enough to discard c!



~~Idea 1~~

We shouldn't choose a match if it one end can be shifted to insert another char

abadcbc

One end of c can be shifted

to insert d

abbcdc

This is not enough to discard c!

There's an MCS abcc that uses that match!

Idea2: keep all possible embeddings found so far

We can clearly see that

$Y \subset X$

X: `babababab`

Y: `baabaab`

Watch out for complexity!

We can clearly see that

$$Y \subset X$$

Hence

$$MCS(X,Y) = \{Y\}$$

X: babababab

Y: baabaab

Watch out for complexity!

We can clearly see that

$$Y \subset X$$

Hence

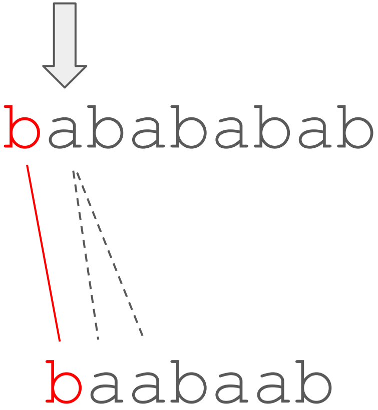
$$MCS(X, Y) = \{Y\}$$

But we don't know it yet!

X: `babab`

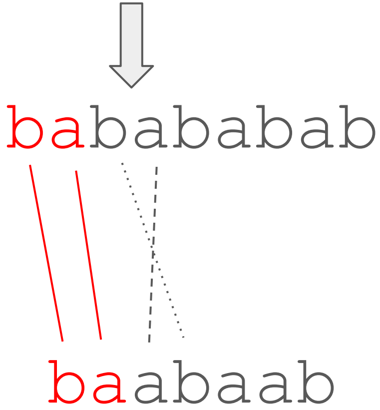
Y: `baab`

Watch out for complexity!



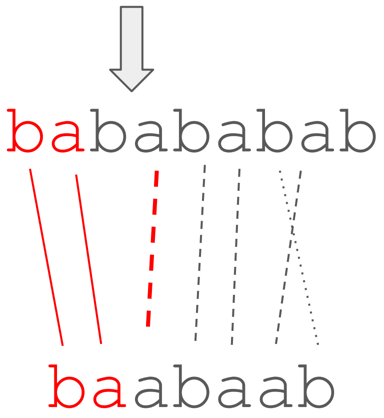
2 choices
surely the left one
is the better one

Watch out for complexity!



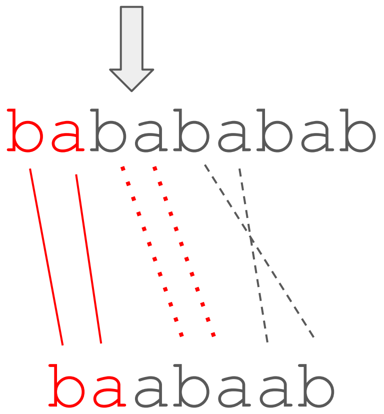
2 choices
which is the
better one?

Watch out for complexity!



What about here??

Watch out for complexity!

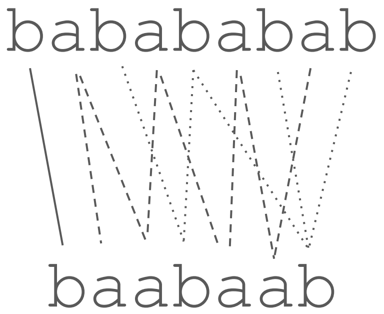


and here???

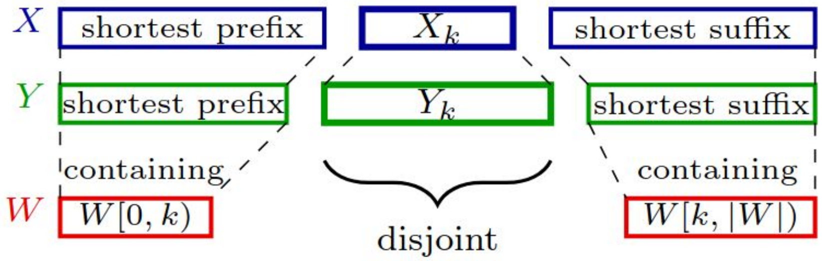
Watch out for complexity!

The number of embeddings is exponential

We cannot explore all configurations in reasonable time



Other too complex ideas



Use the curtain algorithm:

Input: W common subseq.

$\forall k \in [|W|]:$

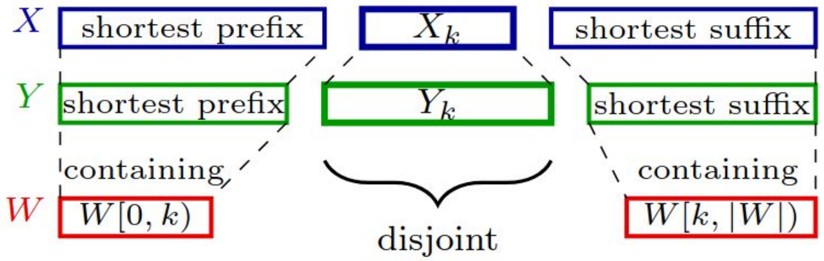
$I_k \leftarrow X_k \cap Y_k$

$\forall c \in I_k:$

recur on $W[0, k) \cdot c \cdot W[k, |W|)$

if $\sum_k |I_k| = 0 \Rightarrow W \in MCS(X, Y)$

Other too complex ideas



Use the curtain algorithm:

Input: W common subseq.

bababab

$\forall k \in [|W|]:$

$I_k \leftarrow X_k \cap Y_k$

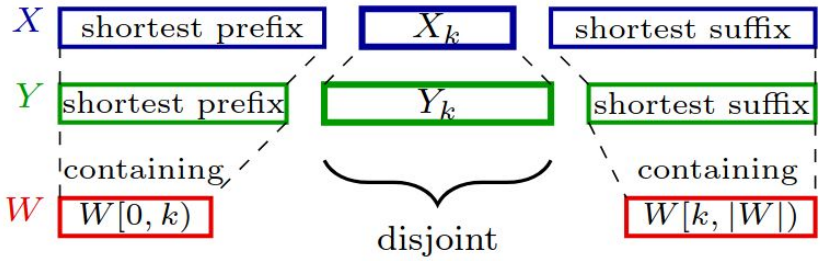
$\forall c \in I_k:$

babab

recur on $W[0,k) \cdot c \cdot W[k,|W|)$

if $\sum_k |I_k| = 0 \Rightarrow W \in MCS(X,Y)$

Other too complex ideas



Use the curtain algorithm:

Input: W common subseq.

bababab

$$\forall k \in [|W|] :$$

$$I_k \leftarrow X_k \cap Y_k$$

$$\forall c \in I_k :$$

babab

recur on $W[0, k) \cdot c \cdot W[k, |W|)$

if $\sum_k |I_k| = 0 \Rightarrow W \in MCS(X, Y)$

In this case we start with
W=a and W=b

We would analyze almost all common subseq.

just to output “babab”, i.e. the only MCS

Promising ideas: Divide and Conquer?

$MCS(X, Y) = \{abba, abad, dba\}$

X: `abadba`

Y: `dabbad`

Promising ideas: Divide and Conquer?

$$MCS(X, Y) = \{abba, abad, dba\}$$

X': `aba | dba` :X''

Y': `dab | bad` :Y''

Promising ideas: Divide and Conquer?

$$MCS(X,Y) = \{abba, abad, dba\}$$

X': aba | dba :X''

$$MCS(X',Y') = \{ab\}$$

Y': dab | bad :Y''

$$MCS(X'',Y'') = \{ba, d\}$$

~~Promising~~ ideas: Divide and Conquer?

$$MCS(X,Y) = \{abba, abad, dba\}$$

X': aba | dba :X''

$$MCS(X',Y') = \{ab\}$$

Y': dab | bad :Y''

$$MCS(X'',Y'') = \{ba, d\}$$

$$MCS(X',Y') \times MCS(X'',Y'') = \{abba, abd\}$$

~~Promising~~ ideas: Divide and Conquer?

$$MCS(X,Y) = \{abba, abad, dba\}$$

X': aba | dba :X''

$$MCS(X',Y') = \{ab\}$$

Y': dab | bad :Y''

$$MCS(X'',Y'') = \{ba, d\}$$

$$MCS(X',Y') \times MCS(X'',Y'') = \{\boxed{abba}, \boxed{abd}\}$$:-($abd \subset abad$

But maybe we are getting closer

Better ideas for incremental construction

Let P be a prefix of an MCS

Better ideas for incremental construction

Let P be a prefix of an MCS

We want to extend it to $P' = Pc$, such that P' is still a prefix of some MCS

Better ideas for incremental construction

Let P be a prefix of an MCS

We want to extend it to $P' = Pc$, such that P' is still a prefix of some MCS

But how do we know P is a prefix of an MCS if we do not know the MCS?

Prefixes of MCS and MCS of prefixes

The prefix of an MCS is an MCS of a prefix of the two strings

Prefixes of MCS and MCS of prefixes

The prefix of an MCS is an MCS of a prefix of the two strings

????????????
????????????
?

Prefixes of MCS and MCS of prefixes

The prefix of an MCS is an MCS of a prefix of the two strings

$MCS(X, Y) = \{abba, abad, dba\}$

```
abadba
```

```
dabbad
```

Prefixes of MCS and MCS of prefixes

The prefix of an MCS is an MCS of a prefix of the two strings

$$MCS(X, Y) = \{\boxed{ab}ba, \boxed{ab}ad, dba\}$$

abadba

The prefix "ab"

dabbad

Prefixes of MCS and MCS of prefixes

The prefix of an MCS is an MCS of a prefix of the two strings

$MCS(X, Y) = \{abba, abad, dba\}$

aba|dba

The prefix "ab"

is an MCS of X[0,3) and Y[0,4)

dabb|ad

Prefixes of MCS and MCS of prefixes

The prefix of an MCS is an MCS of a prefix of the two strings

Formally:

$$\forall s \in MCS(X,Y), p_s \in [|s|], \exists p_X, p_Y \in [n] :$$

$$s[0,p_s) \in MCS(X[0,p_X),Y[0,p_Y))$$

Prefixes of MCS and MCS of prefixes

The prefix of an MCS is an MCS of a prefix of the two strings

Formally:

$$\forall s \in MCS(X,Y), p_s \in [|s|], \exists p_X, p_Y \in [n] :$$

$$s[0, p_s) \in MCS(X[0, p_X), Y[0, p_Y))$$

The converse doesn't hold!

MCS of prefixes and prefixes of MCS

An MCS of a prefix of the two strings is not necessarily the prefix of an MCS

MCS of prefixes and prefixes of MCS

An MCS of a prefix of the two strings is not necessarily the prefix of an MCS

$MCS(X[0,6), Y[0,2)) = \{da\}$

abadba |

da | bbad

MCS of prefixes and prefixes of MCS

An MCS of a prefix of the two strings is not necessarily the prefix of an MCS

$$MCS(X[0,6), Y[0,2)) = \{da\}$$

abadba

The MCS "da"

is not a prefix of any MCS

dabbad

$$MCS(X, Y) = \{abba, abad, dba\}$$

Recap for incremental construction

- P prefix of $s \in MCS(X, Y) \Rightarrow P \in MCS(X[0, p_X), Y[0, p_Y))$

- $P \in MCS(X[0, p_X), Y[0, p_Y)) \not\Rightarrow P$ prefix of $s \in MCS(X, Y)$

But our goal is exactly finding MCS through prefixes!

We needed the second implication!

- $P \in MCS(X[0, p_X), Y[0, p_Y)) \not\Rightarrow P$ prefix of $s \in MCS(X, Y)$

We need something stronger

Let P be the prefix of an MCS

We need something stronger

Let P be the prefix of an MCS

(base case: empty string ε)

We need something stronger

Let P be the prefix of an MCS

(base case: empty string ε)

It can be proven that for some $c \in \Sigma$:

Pc is a valid prefix $\Leftrightarrow$

1. $\exists(i,j) \in Ext_P : X[i] = c \wedge Y[j] = c$
2. $P \in MCS(X[0,i), Y[0,j))$

We need something stronger

Let P be the prefix of an MCS

(base case: empty string ε)

It can be proven that for some $c \in \Sigma$:

Pc is a valid prefix $\Leftrightarrow$

1. $\exists (i, j) \in Ext_P : X[i] = c \land Y[j] = c$
2. $P \in MCS(X[0, i), Y[0, j))$

So if P is an MCS of a prefix AND the c is in Ext_P

=> Pc is the prefix of an MCS!

We need something stronger

Let P be the prefix of an MCS

(base case: empty string ε)

It can be proven that for some $c \in \Sigma$:

Pc is a valid prefix $\Leftrightarrow$

1. $\exists (i,j) \in Ext_P : X[i] = c \wedge Y[j] = c$
2. $P \in MCS(X[0,i), Y[0,j))$

So if P is an MCS of a prefix AND the c is in Ext_P

=> Pc is the prefix of an MCS! => Pc is the MCS of a prefix!

We need something stronger

Let P be the prefix of an MCS

(base case: empty string ε)

It can be proven that for some $c \in \Sigma$:

Pc is a valid prefix $\Leftrightarrow$

1. $\exists (i,j) \in Ext_P : X[i] = c \wedge Y[j] = c$
2. $P \in MCS(X[0,i), Y[0,j))$

So if P is an MCS of a prefix AND the c is in Ext_P

=> Pc is the prefix of an MCS! => Pc is the MCS of a prefix!

We can iterate!

An example

Let $P = ab$

`abadba`

`dabbad`

An example

Let $P = ab$

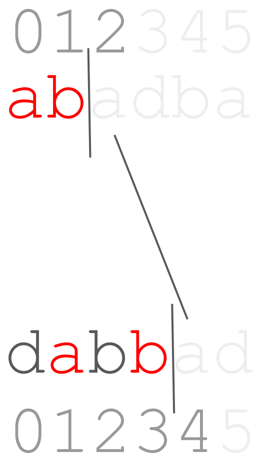
$$Ext_P = \{(4,3),(2,4)\}$$

An example

Let $P = ab$

$Ext_P = \{(4,3), \boxed{(2,4)}\}$

'a' = X[2] = Y[4]



An example

Let $P = ab$

$Ext_P = \{(4,3), \boxed{(2,4)}\}$

‘a’ = X[2] = Y[4]

$P \in MCS(X[0,2), Y[0,4))$!

```
012345
abadba

dabbad
012345
```

An example

Let $P = ab$

$Ext_P = \{(4, 3), \boxed{(2, 4)}\}$

'a' = X[2] = Y[4]

$P \in MCS(X[0, 2), Y[0, 4))$!

$\Rightarrow P \cdot a$ is prefix of an MCS

012345
abadba

dabbad
012345

$MCS(X, Y) = \{abba, abad, dba\}$

An example

Let $P = ab$

$Ext_P = \{(4,3), \boxed{(2,4)}\}$

'a' = X[2] = Y[4]

$P \in MCS(X[0,2), Y[0,4))$!

012345
abadba

$MCS(X,Y) = \{abba, abad, dba\}$

dabbad
012345

$\Rightarrow P \cdot a$ is prefix of an MCS

-> 'aba' is one MCS of the prefix identified by (2,4)!!

An example

Let $P = ab$

$Ext_P = \{(4,3), \boxed{(2,4)}\}$

'a' = X[2] = Y[4]

$P \in MCS(X[0,2), Y[0,4))$!

012345
abadba

$MCS(X,Y) = \{abba, abad, dba\}$

dabbad
012345

$\Rightarrow P \cdot a$ is prefix of an MCS

-> 'aba' is one MCS of the prefix identified by (2,4)!!

-> Keep going! $Ext_{aba} = \ldots$

What’s Ext?

What's Ext?

Don't worry about it

We would just need
40 more minutes

What's Ext?

Don't worry about it

Just know that it needs

$O(|\Sigma|n^2 log(n))$

preprocessing time and

the whole algorithm takes

$O(|\Sigma|n^3)$ delay and $O(n^2)$ space

Going further

Open problems

Improving complexity

- We have a conditional lower bound of $O(n^2)$ from LCS

Open problems

Improving complexity

We have a conditional lower bound of $O(n^2)$ from LCS

Testing applications

All applications of LCS can be adapted to MCS, with better (?) performances!

Open problems

Improving complexity

We have a conditional lower bound of $O(n^2)$ from LCS

Testing applications

All applications of LCS can be adapted to MCS, with better (?) performances!

Special cases

Redefining maximality to fit applications such as DNA sequence alignment

Open problems

Improving complexity

We have a conditional lower bound of $O(n^2)$ from LCS

Testing applications

All applications of LCS can be adapted to MCS, with better (?) performances!

Special cases

Redefining maximality to fit applications such as DNA sequence alignment

Or anything that comes to mind!

Open problems

Improving complexity

We have a conditional lower bound of $O(n^2)$ from LCS

Testing applications

All applications of LCS can be adapted to MCS, with better (?) performances!

Special cases

Redefining maximality to fit applications such as DNA sequence alignment

Or anything that comes to mind!

Generalizing to $k > 2$ strings

Takeaways

MCS are quite slippery to solve

    deceivingly simple

Takeaways

MCS are quite slippery to solve

deceivingly simple, but fun!

I only gave you ONE way to solve the problem

if you have some ideas we could have a chat

Takeaways

MCS are quite slippery to solve

deceivingly simple, but fun!

I only gave you ONE way to solve the problem

if you have some ideas we could have a chat

Great opportunity for research!

Tons of paper on LCS, less than 10 on MCS!

Takeaways

MCS are quite slippery to solve

deceivingly simple, but fun!

I only gave you ONE way to solve the problem

if you have some ideas we could have a chat

Great opportunity for research!

Tons of paper on LCS, less than 10 on MCS!

Great potential, not quite known

Thank you