An input sensitive online algorithm for LCS computation

Heikki Hyyrö

Department of Computer Sciences
University of Tampere, Finland

The basic problem setting:

The basic problem setting:

- Input: strings A and B from an alphabet of size σ
  - ▷ The lengths are n and m with $n \geq m$

The basic problem setting:

- Input: strings A and B from an alphabet of size σ
  - ▷ The lengths are n and m with $n \geq m$
- The task: compute the length of a **longest common subsequence** (LCS) of A and B

The basic problem setting:

- Input: strings A and B from an alphabet of size σ
  - ▷ The lengths are n and m with $n \geq m$
- The task: compute the length of a **longest common subsequence** (LCS) of A and B
  - ▷ We denote the length by L
  - ▷ E.g. if $A =$ "Prague" and $B =$ "charge"

The basic problem setting:

- Input: strings A and B from an alphabet of size σ
  - ▷ The lengths are n and m with $n \geq m$
- The task: compute the length of a **longest common subsequence** (LCS) of A and B
  - ▷ We denote the length by L
  - ▷ E.g. if $A =$ "Prague" and $B =$ "charge", then $L =$ LLCS(A,B) $= 3$

The basic problem setting:

- Input: strings A and B from an alphabet of size σ
  - ▷ The lengths are n and m with $n \geq m$
- The task: compute the length of a **longest common subsequence** (LCS) of A and B
  - ▷ We denote the length by L
  - ▷ E.g. if $A =$ "Prague" and $B =$ "charge", then $L = \text{LLCS}(A,B) = 3$

LLCS is a dual of indel edit distance:

- $ed_{id}(A, B) = n + m - 2\text{LLCS}(A, B)$

An online algorithm:

An online algorithm:

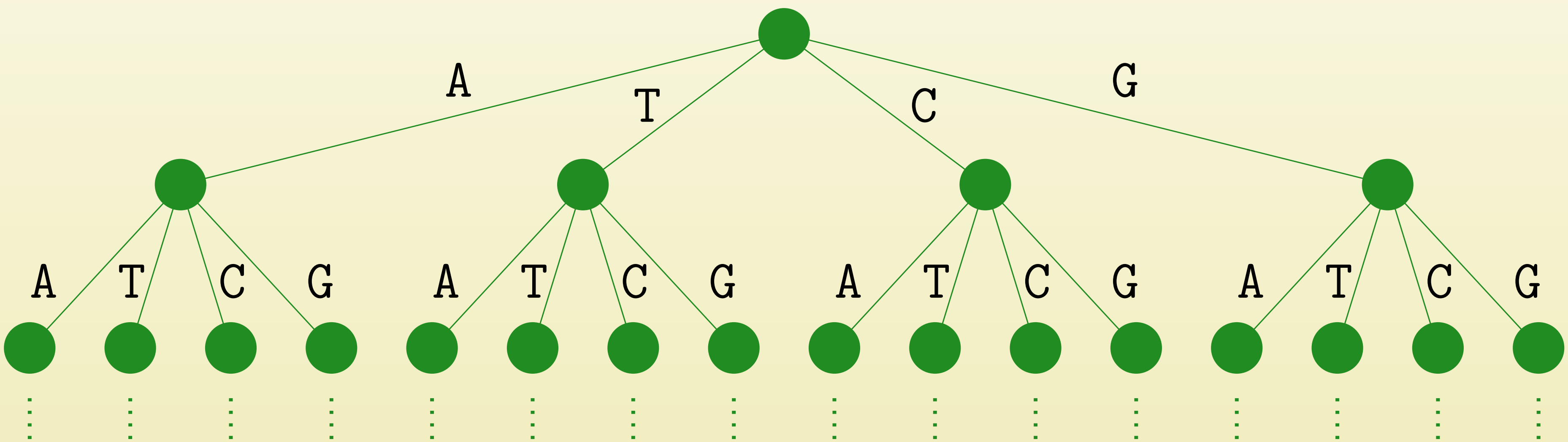
- Preprocesses only A, and can then read B one character at a time

An online algorithm:

- Preprocesses only A, and can then read B one character at a time
- Useful e.g. in one-against-many comparison or neighborhood generation

An online algorithm:

- Preprocesses only A, and can then read B one character at a time
- Useful e.g. in one-against-many comparison or neighborhood generation



Many input-sensitive algorithms exist

Many input-sensitive algorithms exist, e.g.:

- $\mathcal{O}(n \log n + nL)$: Hirschberg: Algorithms for the Longest Common Subsequence Problem, *Journal of the ACM* (1977)

Many input-sensitive algorithms exist, e.g.:

- $\mathcal{O}(n \log n + nL)$: Hirschberg: Algorithms for the Longest Common Subsequence Problem, *Journal of the ACM* (1977)
- $\mathcal{O}(n(m - L))$: Nakatsu et al.: A Longest Common Subsequence Algorithm Suitable for Similar Texts, *Acta Informatica* (1982)

Many input-sensitive algorithms exist, e.g.:

- $\mathcal{O}(n \log n + nL)$: Hirschberg: Algorithms for the Longest Common Subsequence Problem, *Journal of the ACM* (1977)
- $\mathcal{O}(n(m - L))$: Nakatsu et al.: A Longest Common Subsequence Algorithm Suitable for Similar Texts, *Acta Informatica* (1982)
- $\mathcal{O}(\sigma n + \min\{mL, L(n-L)\})$: Rick: A New Flexible Algorithm for the Longest Common Subsequence Problem, *CPM 1995*

Many input-sensitive algorithms exist, e.g.:

- $\mathcal{O}(n \log n + nL)$: Hirschberg: Algorithms for the Longest Common Subsequence Problem, *Journal of the ACM* (1977)
- $\mathcal{O}(n(m - L))$: Nakatsu et al.: A Longest Common Subsequence Algorithm Suitable for Similar Texts, *Acta Informatica* (1982)
- $\mathcal{O}(\sigma n + \min\{mL, L(n - L)\})$: Rick: A New Flexible Algorithm for the Longest Common Subsequence Problem, *CPM 1995*
- $\mathcal{O}(\sigma n + \min\{mL, L(n - L)\})$: Goeman & Clausen: A New Practical Linear Space Algorithm for the Longest Common Subsequence Problem, *Kybernetika* (2002)

Classic solution: $\mathcal{O}(mn)$ dynamic programming

Classic solution: $\mathcal{O}(mn)$ dynamic programming

- A table D, where $D[i,j]$ =LLCS$(A_{1..i}, B_{1..j})$ and

$$D[i,j] = \begin{cases} 1 + D[i\text{-}1, j\text{-}1], \text{ if } A_i = B_j, \\ \text{else } \max\{D[i, j\text{-}1], D[i\text{-}1, j]\} \end{cases}$$

		c	h	a	r	g	e
	0	**0**	**0**	**0**	**0**	**0**	**0**
P	**0**	0	0	0	0	0	0
r	**0**	0	0	0	1	1	1
a	**0**	0	0	1	1	1	1
g	**0**	0	0	1	1	2	2
u	**0**	0	0	1	1	2	2
e	**0**	0	0	1	1	2	3

Incremental encoding of columns of D

Incremental encoding of columns of D

Regular:

		A	T	C
	0	**0**	**0**	**0**
T	**0**	0	1	1
A	**0**	1	1	1
C	**0**	1	1	2

Incremental:

		A	T	C
	0	**0**	**0**	**0**
T	**0**	0	1	1
A	**0**	1	0	0
C	**0**	0	0	1

- $\Delta[i, j] = D[i, j] - D[i\text{-}1, j]$
- $D[i, j] = \Sigma_{k=1}^{i} \Delta[k, j]$
  - ▷ $|\{\Delta[k, j] : \ 1 \leq k \leq j \ \wedge \ \Delta[k, j] = 1\}|$

Incremental encoding of columns of D

Regular:

		A	T	C
	0	**0**	**0**	**0**
T	**0**	0	1	1
A	**0**	1	1	1
C	**0**	1	1	2

Incremental:

		A	T	C
T			1	1
A		1		
C				1

- $\Delta[i,j] = D[i,j] - D[i\text{-}1,j]$
- $D[i,j] = \Sigma_{k=1}^{i} \Delta[k,j]$
  - ▷ $|\{\Delta[k,j] : \; 1 \le k \le j \;\wedge\; \Delta[k,j] = 1\}|$
- Store only increment points i where $\Delta[i,j] = 1$

Incremental encoding of columns of D

Regular:

		A	T	C
	0	**0**	**0**	**0**
T	**0**	0	1	1
A	**0**	1	1	1
C	**0**	1	1	2

Incremental:

		A	T	C
T			1	1
A		1		
C				1

- $\Delta[i,j] = D[i,j] - D[i\text{-}1,j]$
- $D[i,j] = \Sigma_{k=1}^{i} \Delta[k,j]$
  - ▷ $|\{\Delta[k,j] : 1 \leq k \leq j \ \wedge \ \Delta[k,j] = 1\}|$
- Store only increment points i where $\Delta[i,j] = 1 \Rightarrow$ each column j takes $D[n,j] \leq L =$ LLCS(A,B) space

Incremental encoding of columns of D

Regular:

		A	T	C
	0	**0**	**0**	**0**
T	**0**	0	1	1
A	**0**	1	1	1
C	**0**	1	1	2

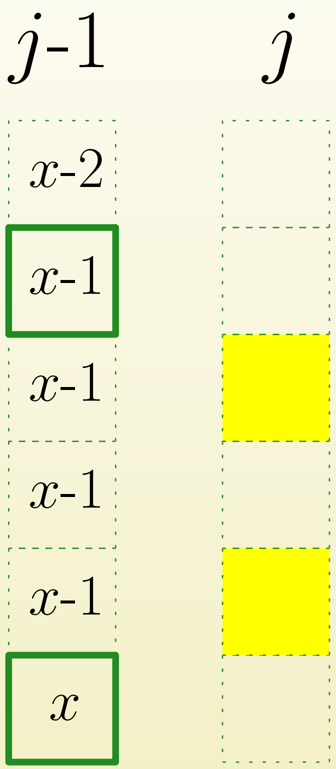
Incremental:

		A	T	C
T			1	1
A		1		
C				1

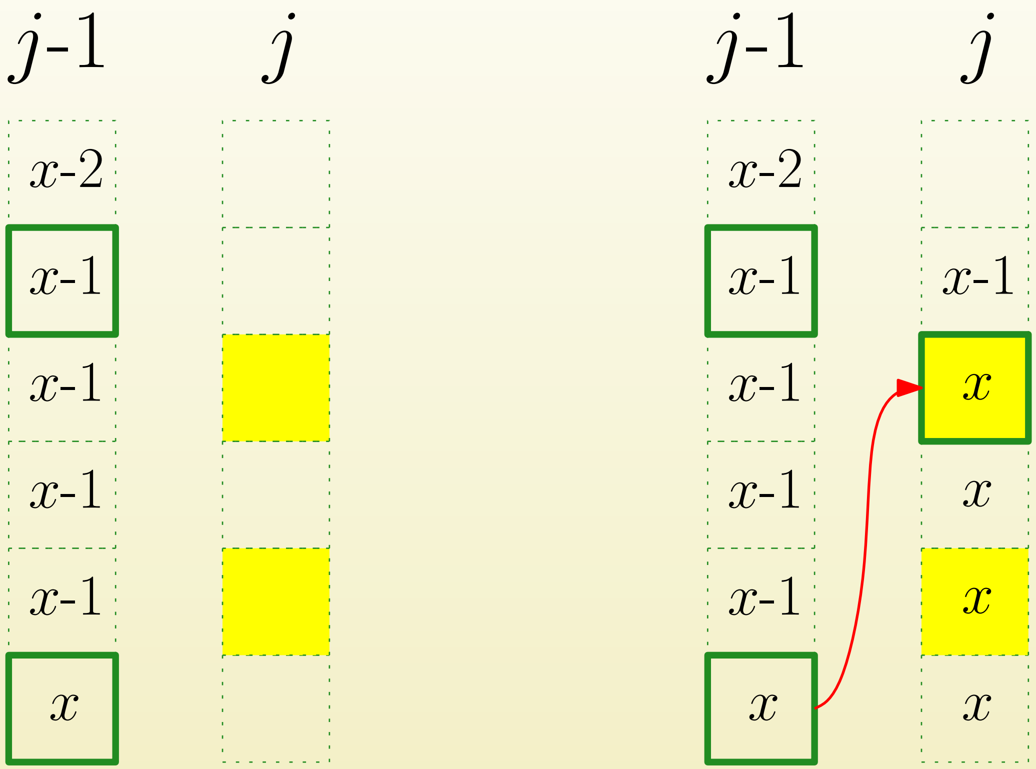
- $\Delta[i,j] = D[i,j] - D[i\text{-}1,j]$
- $D[i,j] = \Sigma_{k=1}^{i} \Delta[k,j]$
  - ▷ $|\{\Delta[k,j] :\ 1 \leq k \leq j\ \wedge\ \Delta[k,j] = 1\}|$
- Store only increment points i where $\Delta[i,j] = 1 \Rightarrow$ each column j takes $D[n,j] \leq L =$ LLCS(A,B) space
- let $I_x[j]$ denote the xth increment point in column j

How to compute increment points for column j?

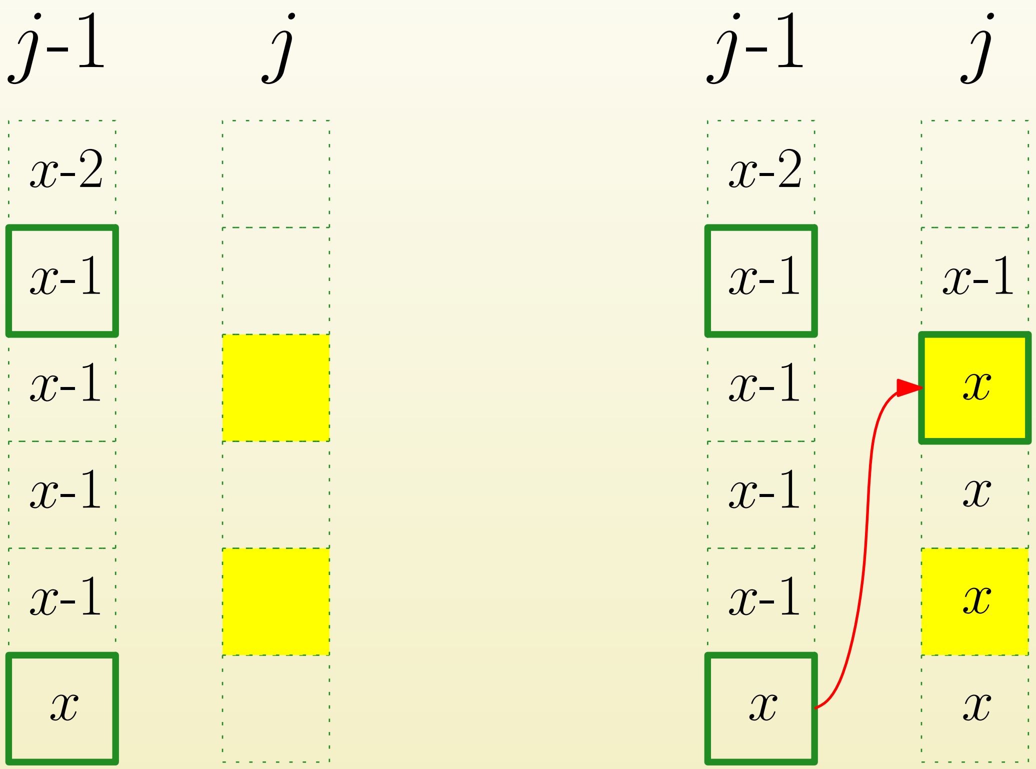
How to compute increment points for column j?



How to compute increment points for column j?



How to compute increment points for column j?



$$I_x[j] = \min\{i : \; i > I_{x\text{-}1}[j\text{-}1] \;\wedge\; (A_i = B_j \vee i = I_x[j\text{-}1])$$

$$I_x[j] = \min\{i : \; i > I_{x\text{-}1}[j\text{-}1] \;\wedge\; (A_i = B_j \vee i = I_x[j\text{-}1])\}$$

$$I_x[j] = \min\{i : \ i > I_{x\text{-}1}[j\text{-}1] \ \wedge \ (A_i = B_j \vee i = I_x[j\text{-}1])$$

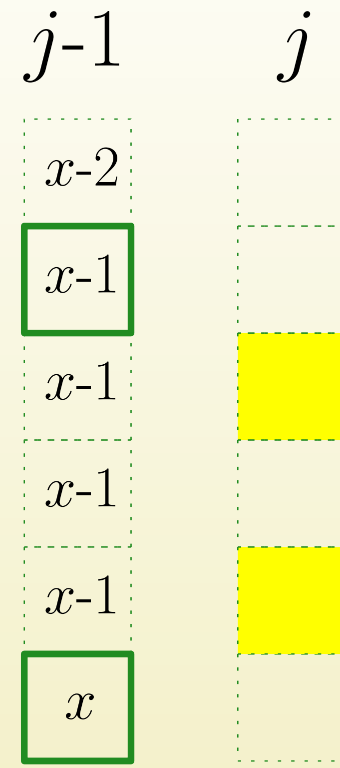
How to locate the relevant match $A_i = B_j$ quickly?

An input sensitive online algorithm for LCS computation

$$I_x[j] = \min\{i : \; i > I_{x\text{-}1}[j\text{-}1] \;\wedge\; (A_i = B_j \vee i = I_x[j\text{-}1])$$

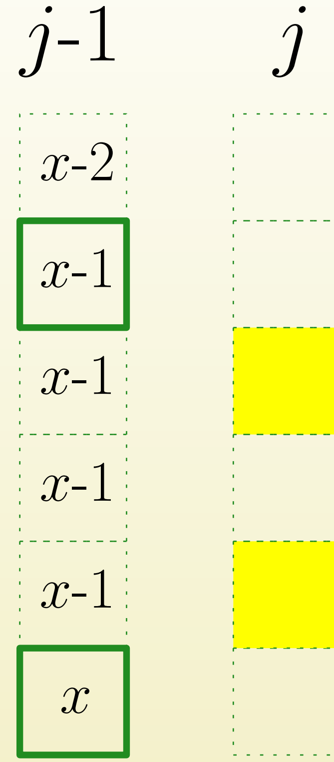
How to locate the relevant match $A_i = B_j$ quickly?

- Precompute a $\sigma \times n$ table NM, where $NM[\lambda, k] = \min\{i : i > k \wedge (A_i = \lambda \vee i = n+1)\}$



An input sensitive online algorithm for LCS computation

$$I_x[j] = \min\{i : \ i > I_{x\text{-}1}[j\text{-}1] \ \wedge \ (A_i = B_j \vee i = I_x[j\text{-}1])$$



How to locate the relevant match $A_i = B_j$ quickly?

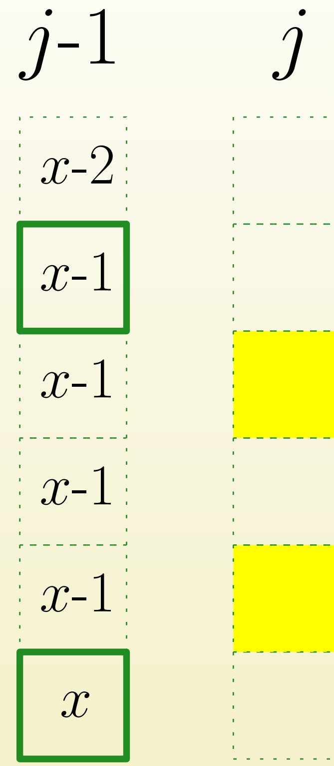
- Precompute a $\sigma \times n$ table NM, where $NM[\lambda, k] = \min\{i : i > k \wedge (A_i = \lambda \vee i = n + 1)\}$

▷ E.g. if $A =$ “oklahoma”, then NM[‘a’,1] $=$ 4 and NM[‘h’,5] $=$ 9

$I_x[j] = \min\{i : \ i > I_{x\text{-}1}[j\text{-}1] \ \wedge \ (A_i = B_j \vee i = I_x[j\text{-}1])$

How to locate the relevant match $A_i = B_j$ quickly?

- Precompute a $\sigma \times n$ table NM, where $NM[\lambda, k] = \min\{i : i > k \wedge (A_i = \lambda \vee i = n+1)\}$

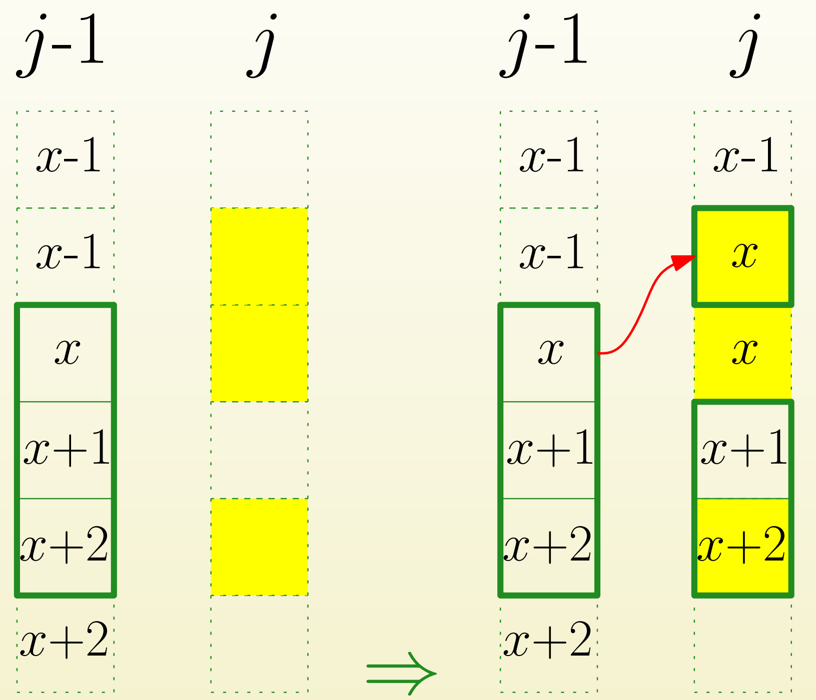


  ▷ E.g. if $A =$ "oklahoma", then NM['a',1] $=$ 4 and NM['h',5] $=$ 9

Resulting time to compute $L =$ LLCS(A, B): $\mathcal{O}(\sigma n + mL)$

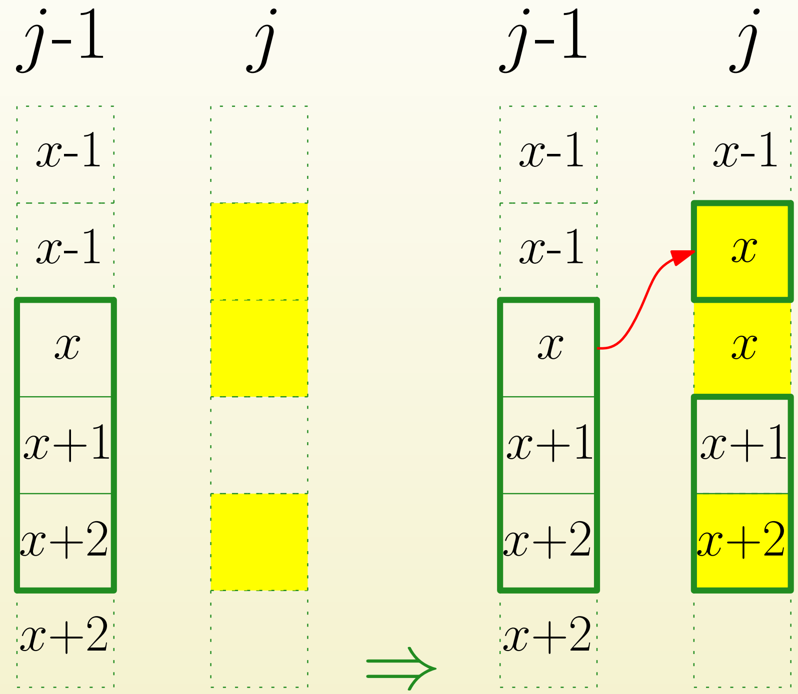
Consider again the computation of the increment points

- The changes occur only to the first increment points among blocks of consecutive increments



Consider again the computation of the increment points

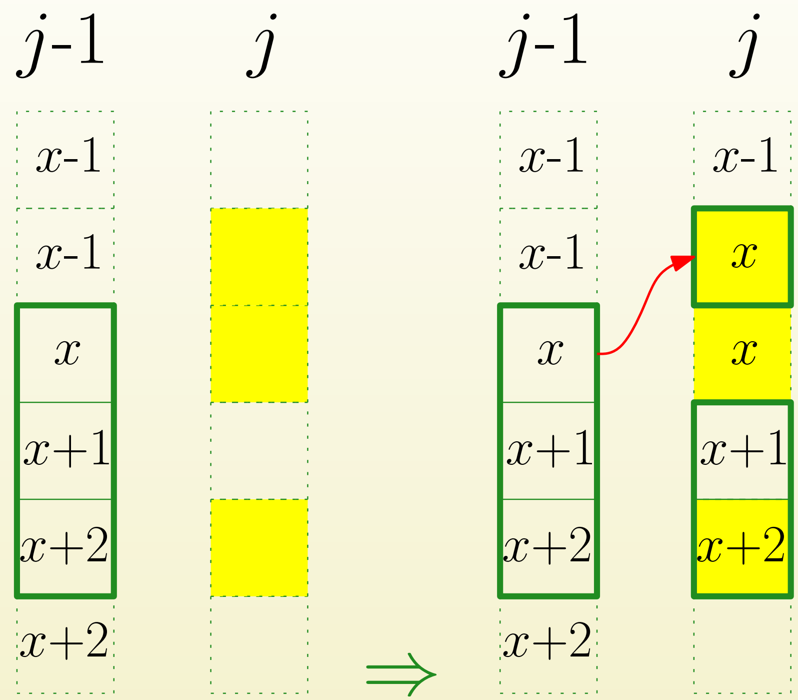
- The changes occur only to the first increment points among blocks of consecutive increments



A “block” encoding:

Consider again the computation of the increment points

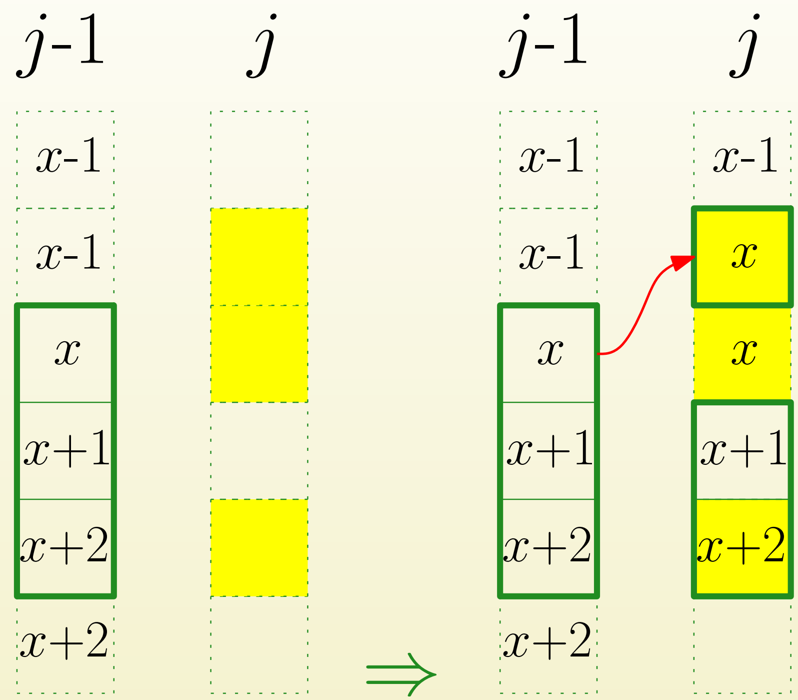
- The changes occur only to the first increment points among blocks of consecutive increments



A "block" encoding: let $S_y[j]$ and $E_y[j]$ be the positions of the first and last points in the yth maximal segment of consecutive increment points

Consider again the computation of the increment points

- The changes occur only to the first increment points among blocks of consecutive increments

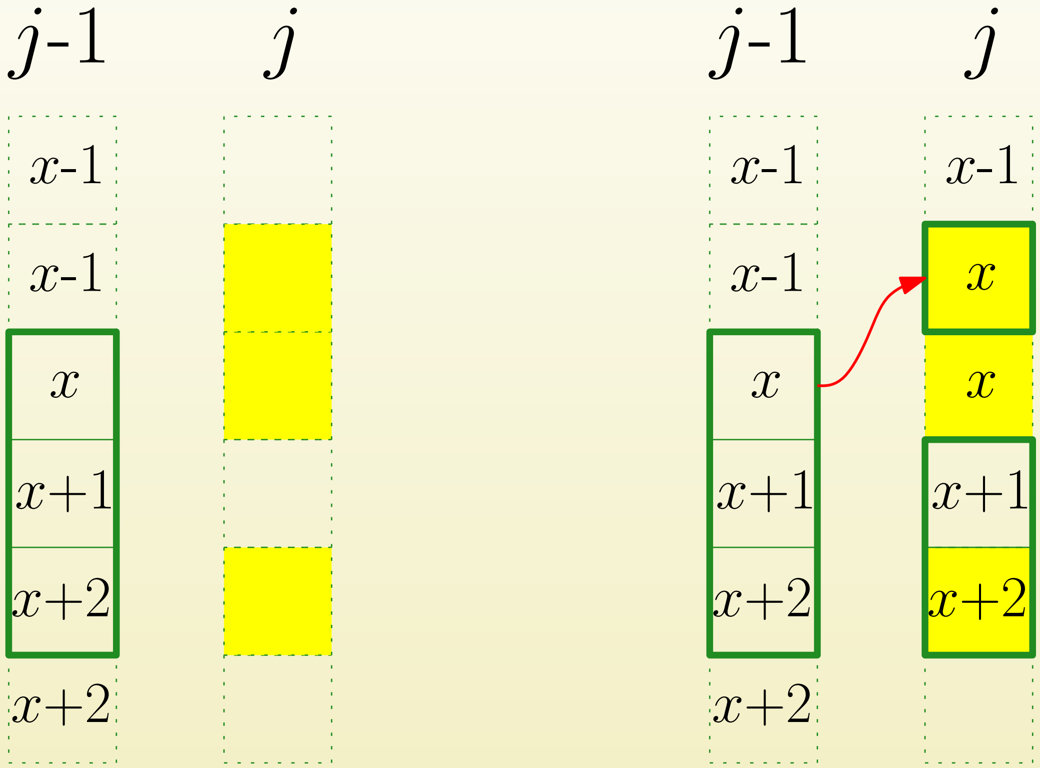


A "block" encoding: let $S_y[j]$ and $E_y[j]$ be the positions of the first and last points in the yth maximal segment of consecutive increment points

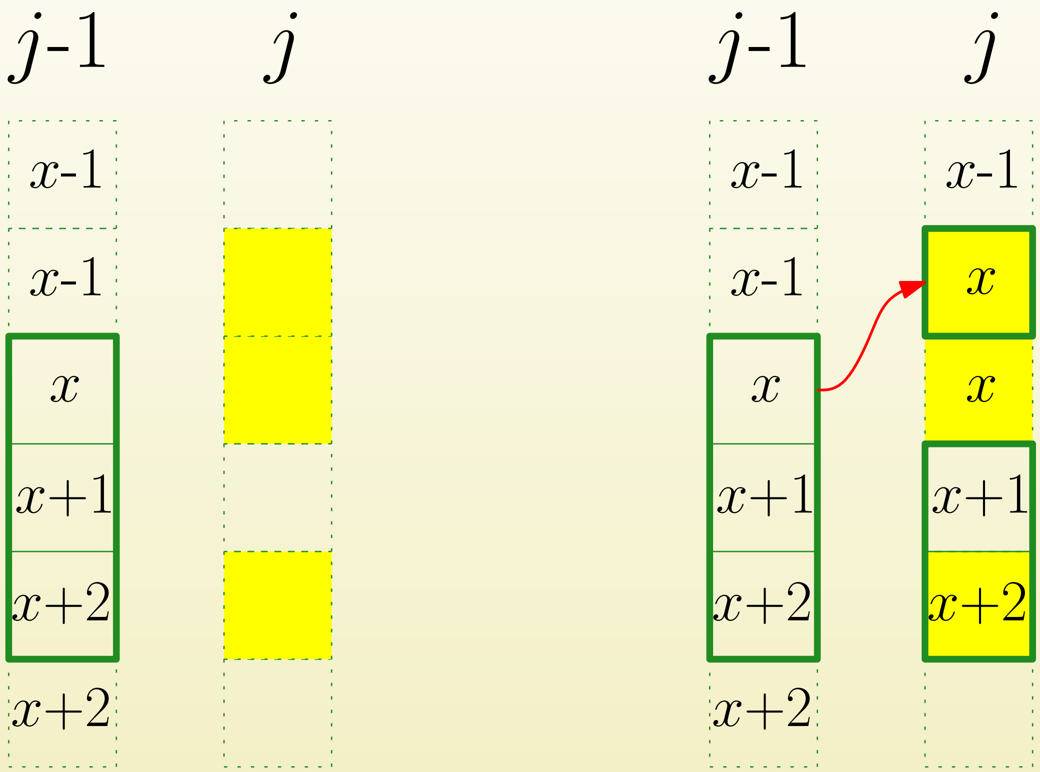
- $\Delta[k, j] = 1$ for $k = S_y[j]..E_y[j]$
- $\Delta[k, j] \neq 0$ for $k = S_y[j]$-1 and $k = E_y[j] + 1$

Create the list of increment blocks for column j incrementally from column $j-1$

Create the list of increment blocks for column j incrementally from column $j-1$

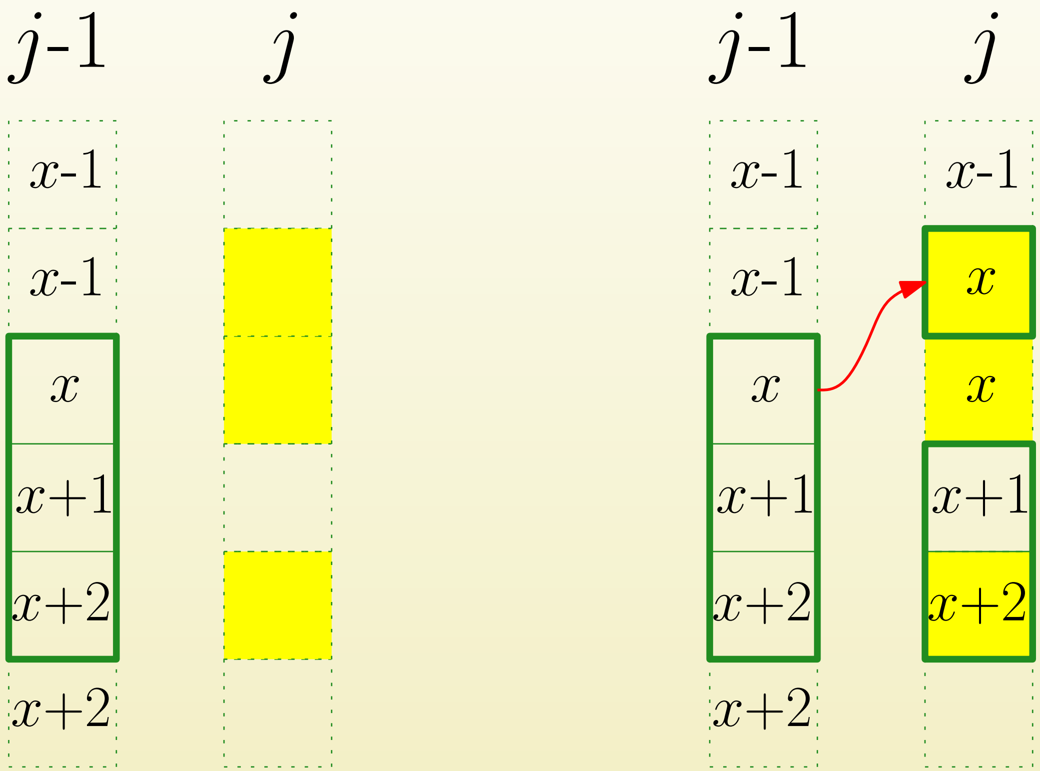


Create the list of increment blocks for column j incrementally from column $j-1$



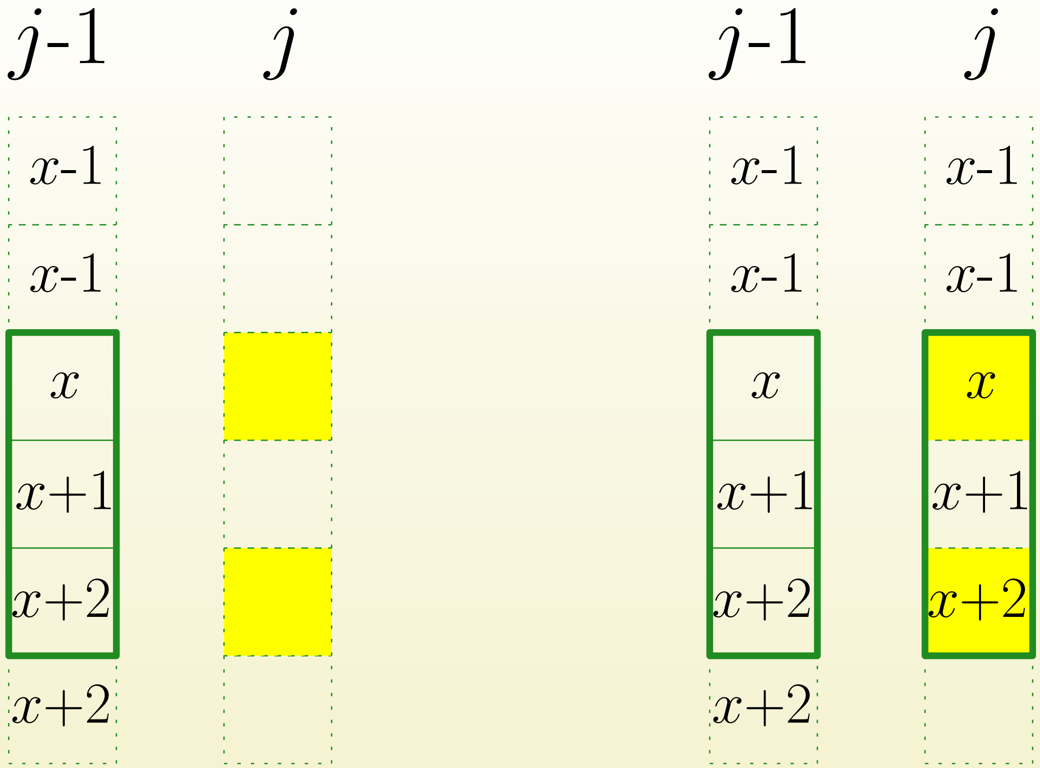
$$\text{if } i = NM[B_j, E_{y-1}[j\text{-}1]] < S_y[j\text{-}1]$$

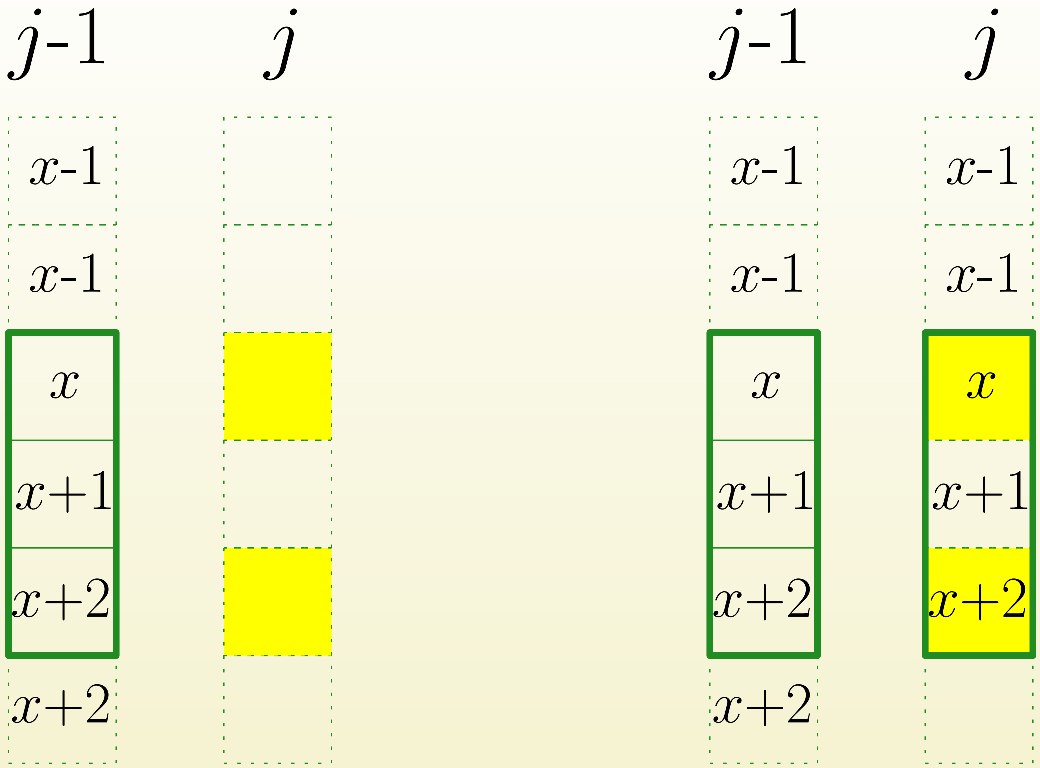
Create the list of increment blocks for column j incrementally from column $j-1$



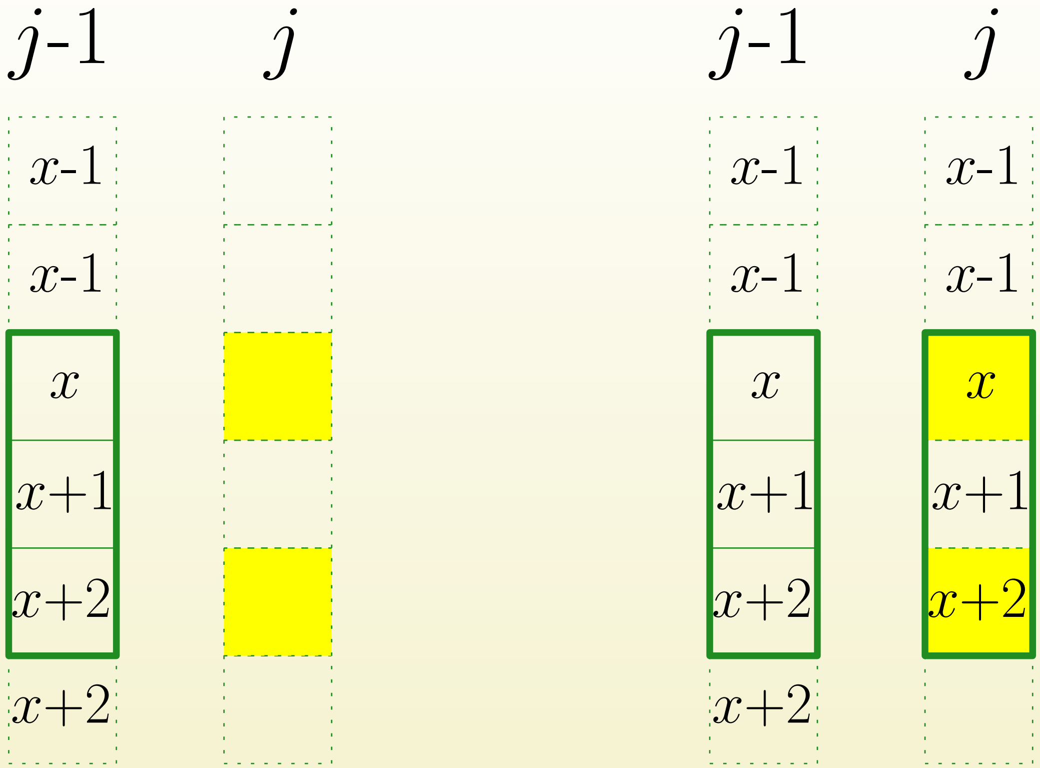
if $i = NM[B_j, E_{y-1}[j\text{-}1]] < S_y[j\text{-}1]$, add increment point i and increment block $S_y[j\text{-}1]+1..E_y[j\text{-}1]$ to column j (possibly merging point i into a block)

An input sensitive online algorithm for LCS computation

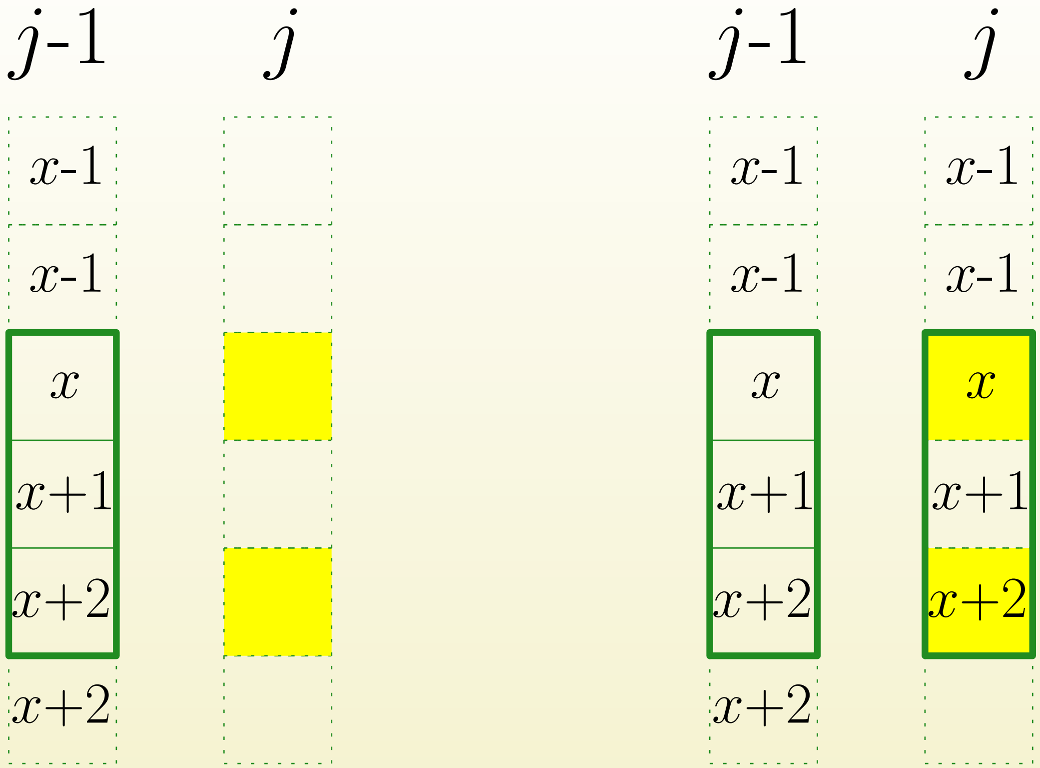




if $i = NM[B_j, E_{y-1}[j\text{-}1]] \geq S_y[j\text{-}1]$

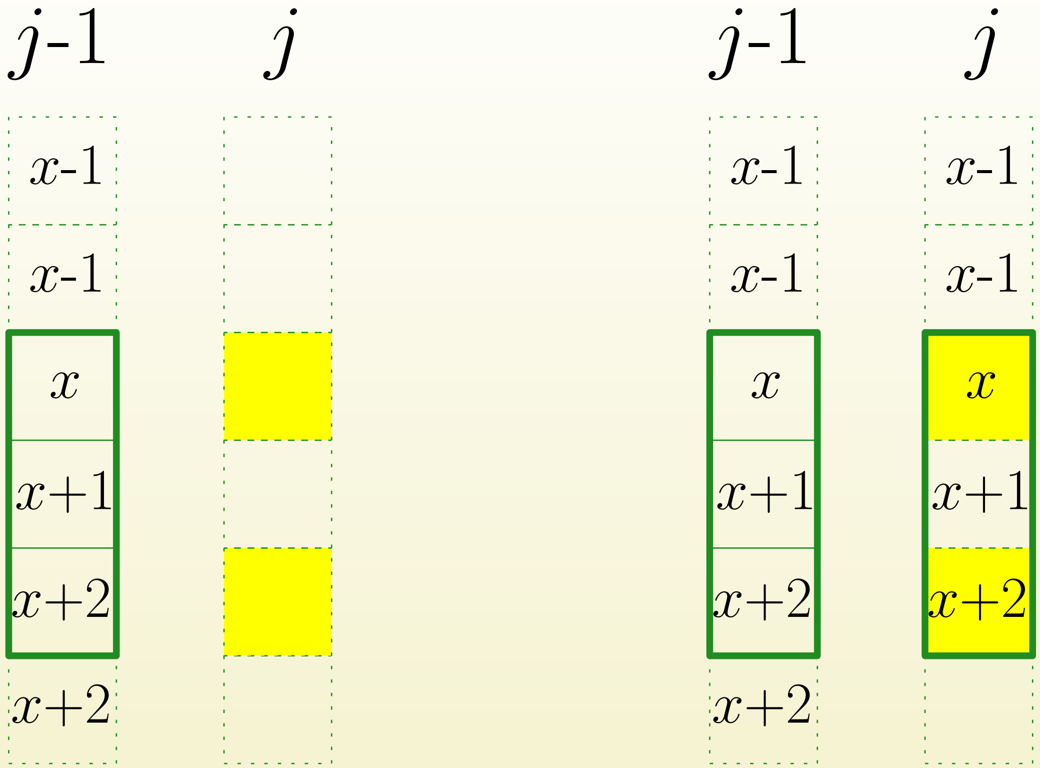


if $i = NM[B_j, E_{y-1}[j\text{-}1]] \geq S_y[j\text{-}1]$, add increment block $S_y[j\text{-}1]..E_y[j\text{-}1]$ as such to column j



if $i = NM[B_j, E_{y-1}[j\text{-}1]] \geq S_y[j\text{-}1]$, add increment block $S_y[j\text{-}1]..E_y[j\text{-}1]$ as such to column j

Amount of work



if $i = NM[B_j, E_{y-1}[j\text{-}1]] \geq S_y[j\text{-}1]$, add increment block $S_y[j\text{-}1]..E_y[j\text{-}1]$ as such to column j

Amount of work $\approx$ the number of increment blocks

Analysis of the encoding

Analysis of the encoding

- Consider a column of size l in the table Δ

Analysis of the encoding

- Consider a column of size l in the table Δ
- Let $0_{\#}$ be the number of non-increment points

Analysis of the encoding

- Consider a column of size l in the table Δ
- Let $0_{\#}$ be the number of non-increment points and $1_{\#}$ the number of increment points

Analysis of the encoding

- Consider a column of size l in the table Δ
- Let $0_{\#}$ be the number of non-increment points and $1_{\#}$ the number of increment points
  - ▷ $l = 0_{\#} + 1_{\#}$

Analysis of the encoding

- Consider a column of size l in the table Δ
- Let $0_{\#}$ be the number of non-increment points and $1_{\#}$ the number of increment points
  - ▷ $l = 0_{\#} + 1_{\#}$
- Also let $block_{\#}$ be the number of maximal increment blocks

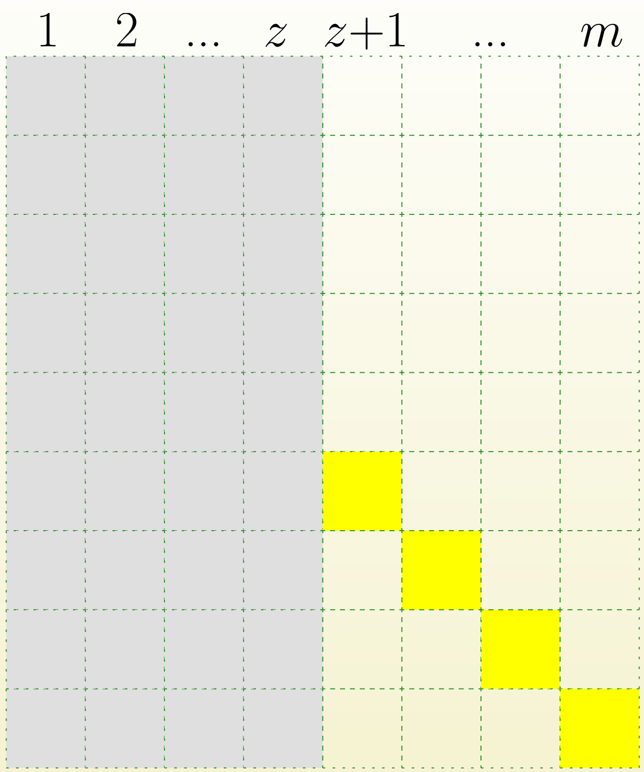
Analysis of the encoding

- Consider a column of size l in the table Δ
- Let $0_{\#}$ be the number of non-increment points and $1_{\#}$ the number of increment points
  - ▷ $l = 0_{\#} + 1_{\#}$
- Also let $block_{\#}$ be the number of maximal increment blocks
- Now it holds that
  - ▷ $block_{\#} \leq 1_{\#}$

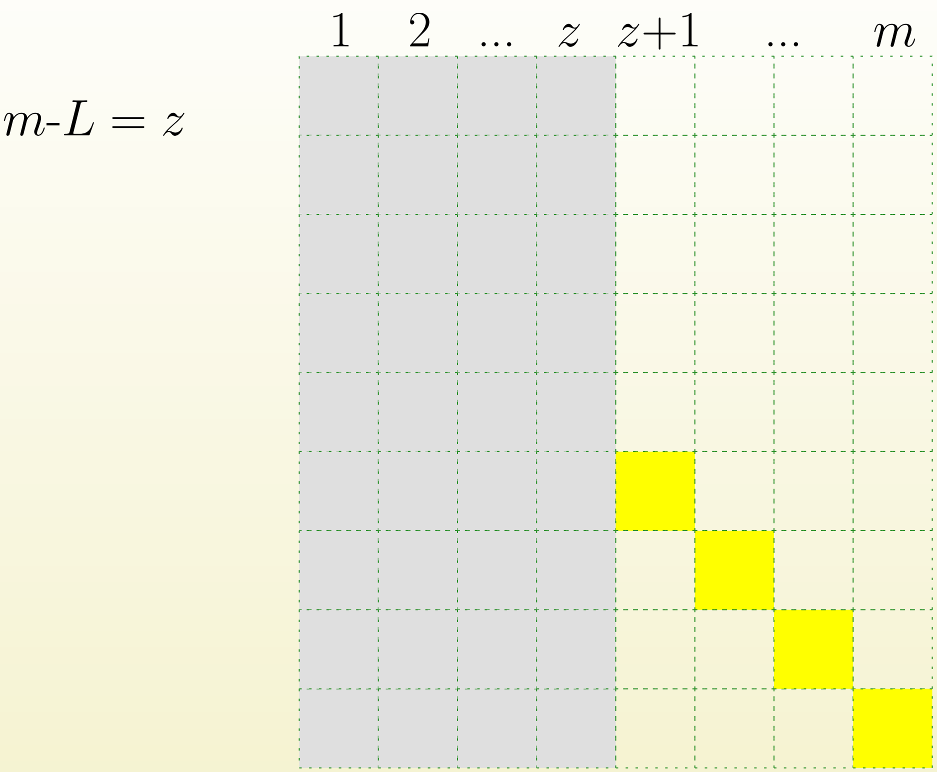
Analysis of the encoding

- Consider a column of size l in the table Δ
- Let $0_\#$ be the number of non-increment points and $1_\#$ the number of increment points
  - ▷ $l = 0_\# + 1_\#$
- Also let $block_\#$ be the number of maximal increment blocks
- Now it holds that
  - ▷ $block_\# \leq 1_\#$
  - ▷ $block_\# \leq 1 + 0_\# = l - 1_\# + 1$

m-$L = z$

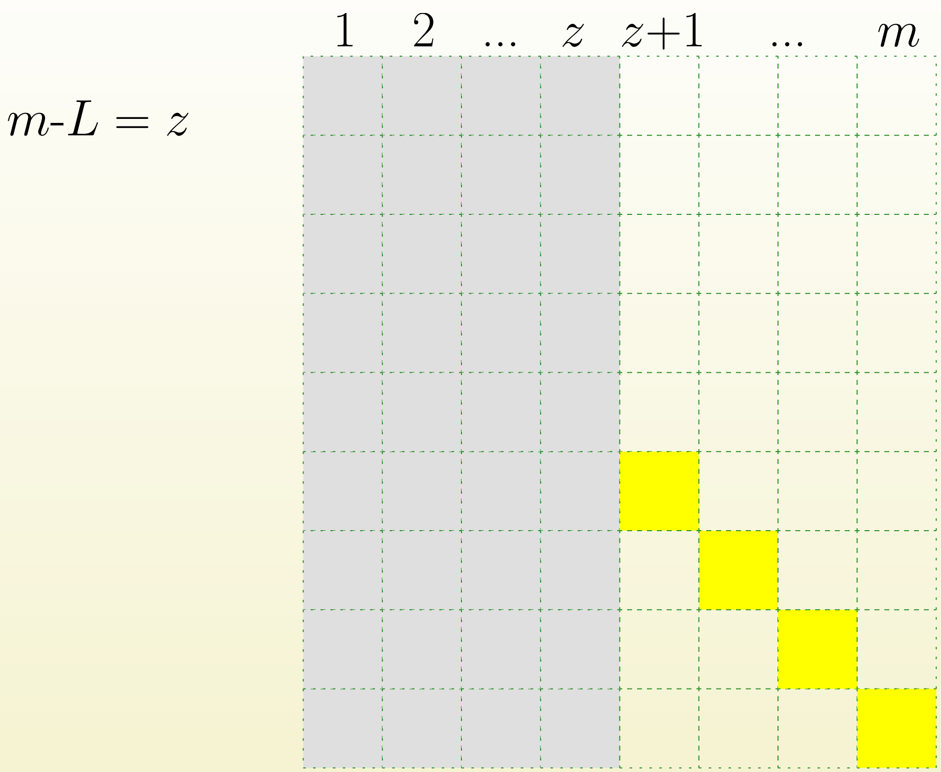


Consider the figure



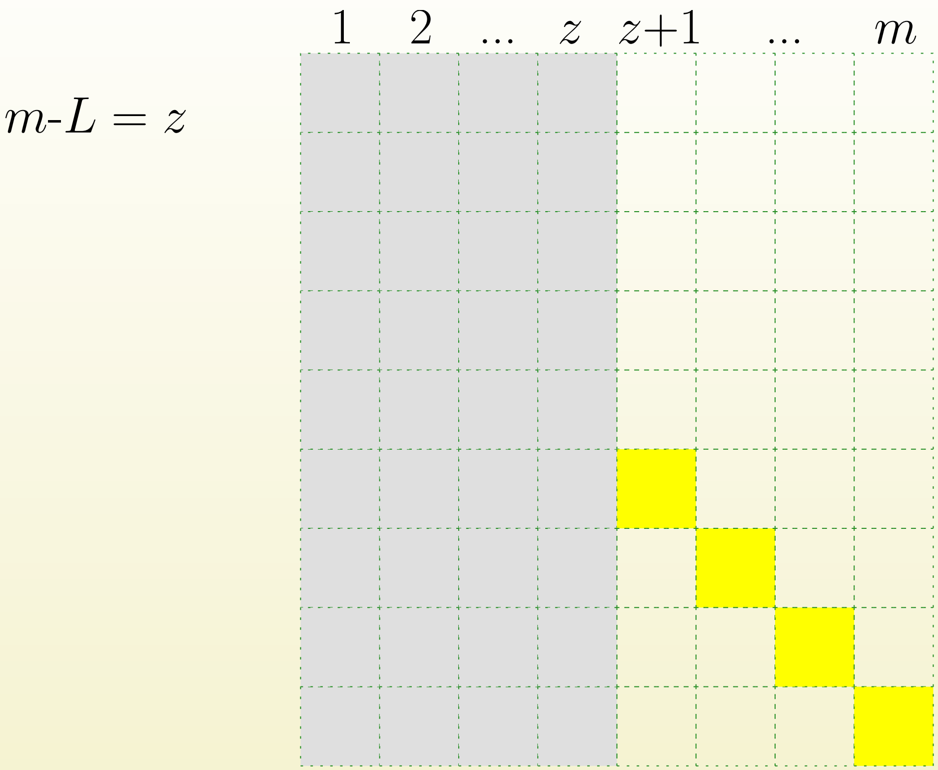
Consider the figure

- Each of the first $z = m - L$ columns



Consider the figure

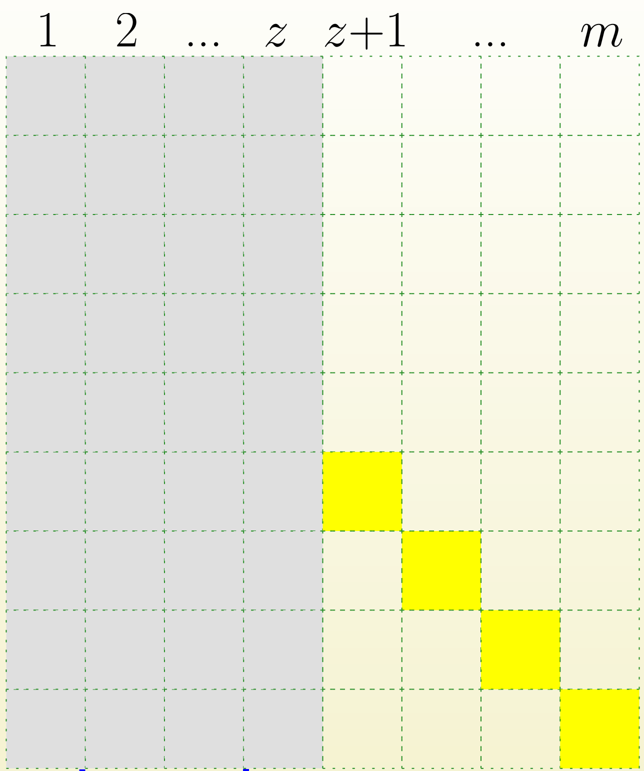
- Each of the first $z = m - L$ columns has at most z increment points (because $D[n, j] \leq j$)



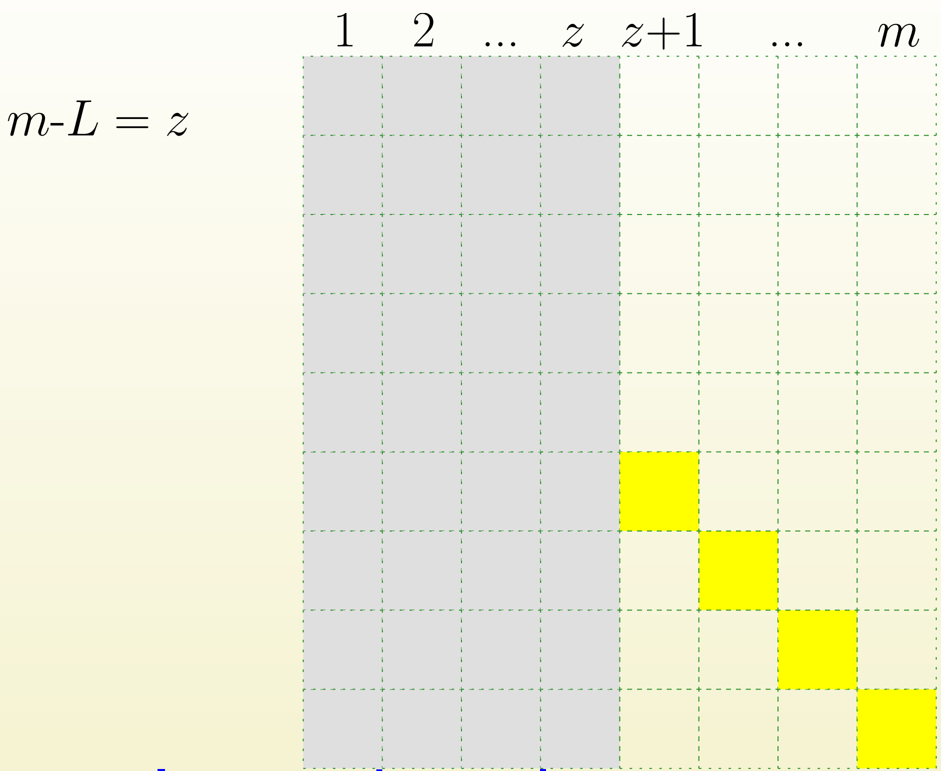
Consider the figure

- Each of the first $z = m - L$ columns has at most z increment points (because $D[n, j] \leq j$)
  - ▷ Work for this part: $\mathcal{O}(z^2)$

m-$L = z$

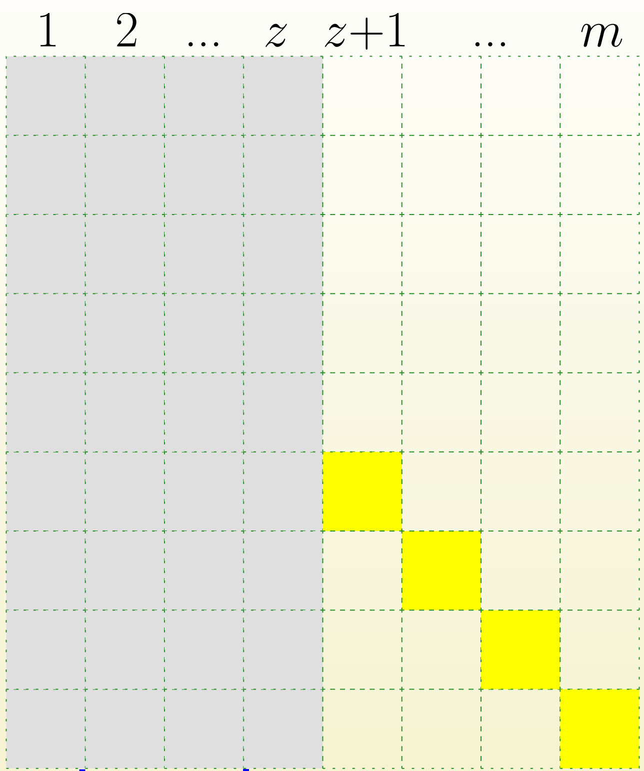


- In each column $z + i$:

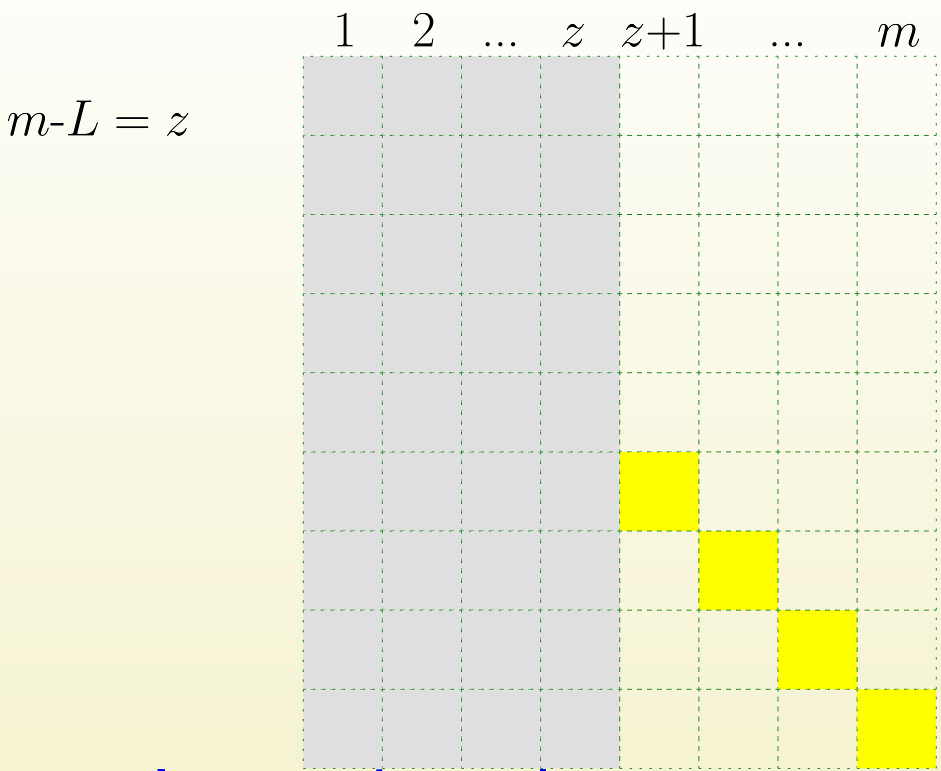


- In each column $z + i$: there are at most $z + i$ increment points

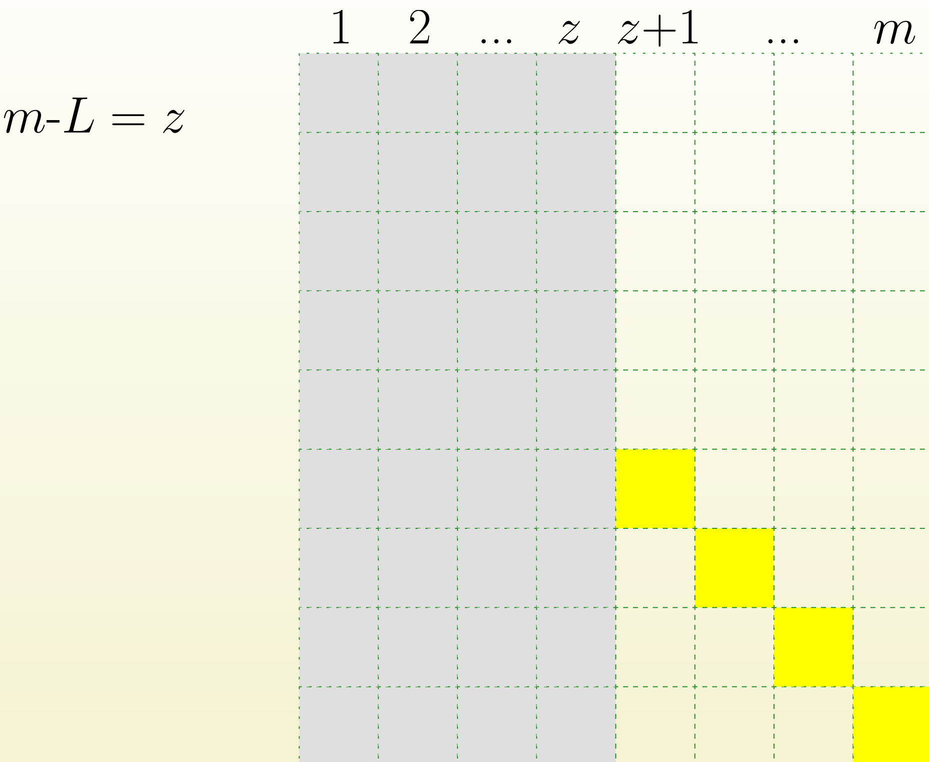
m-$L = z$



- In each column $z+i$: there are at most $z+i$ increment points and the first n-L+i rows hold at least i increment points

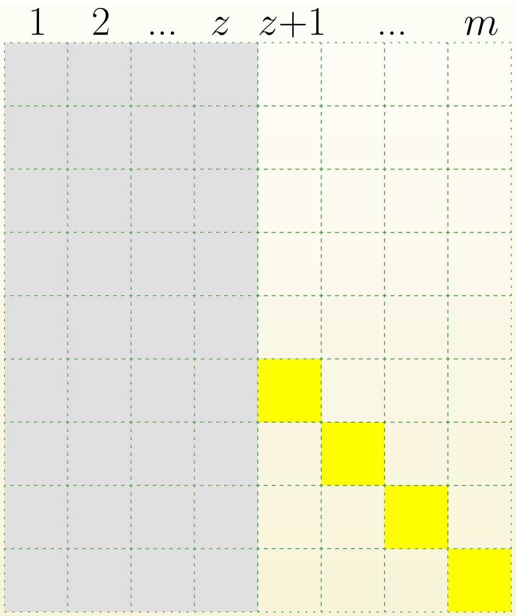


- In each column $z + i$: there are at most $z + i$ increment points and the first n-L+i rows hold at least i increment points
  - ▷ $block_{\#}$ for first n-L+i rows $\leq n$-L+1



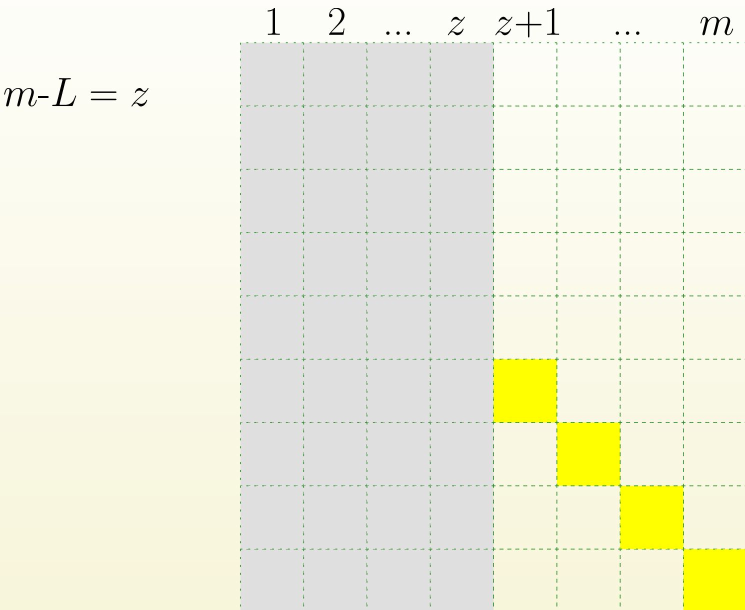
- In each column $z+i$: there are at most $z+i$ increment points and the first n-L+i rows hold at least i increment points
  - ▷ $block_{\#}$ for first n-L+i rows $\leq n$-L+1
  - ▷ $block_{\#}$ for remaining rows $\leq z$+i-$i = z$

m-$L = z$

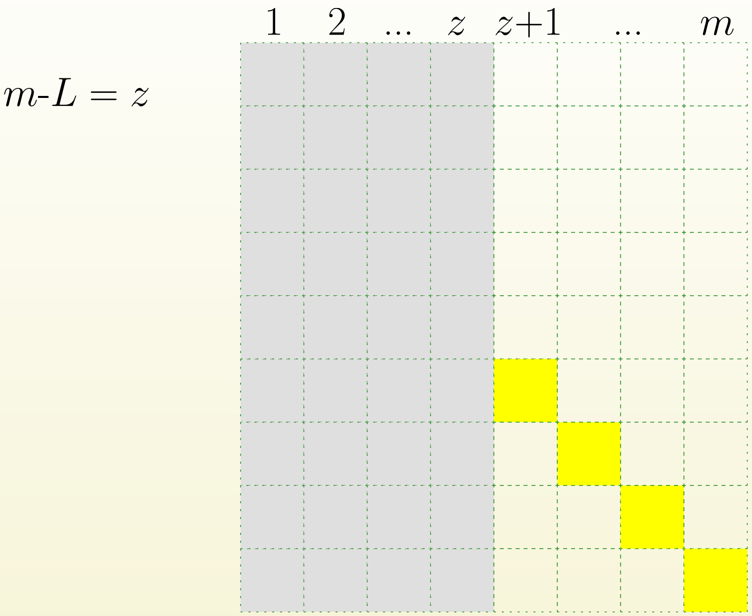


Total work $\approx z^2 + (m-z)(n-L+z+1) = \mathcal{O}(m(n-L))$

An input sensitive online algorithm for LCS computation

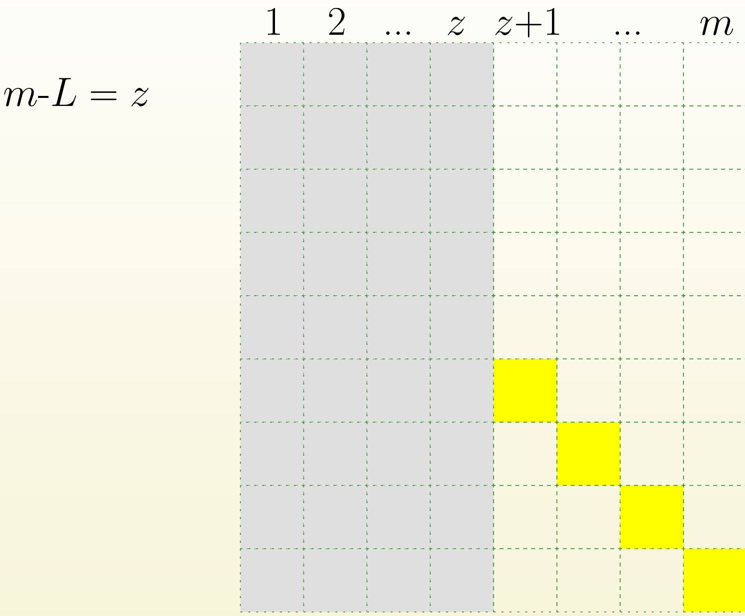


Total work $\approx z^2 + (m-z)(n-L+z+1) = \mathcal{O}(m(n-L))$,
and also bounded by $\mathcal{O}(mL)$



Total work $\approx z^2 + (m-z)(n-L+z+1) = \mathcal{O}(m(n-L))$, and also bounded by $\mathcal{O}(mL)$

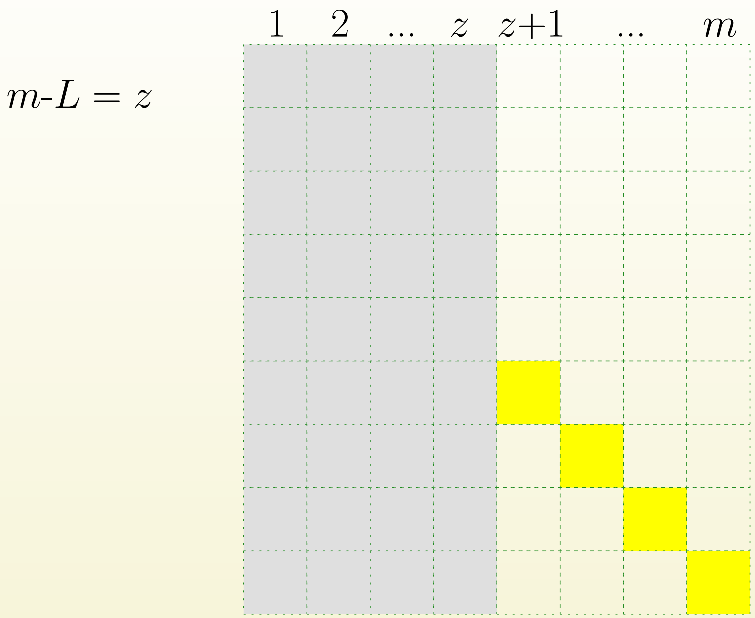
Total time complexity $\mathcal{O}(\sigma n + \min\{mL, L(n-L)\})$



Total work $\approx z^2 + (m-z)(n-L+z+1) = \mathcal{O}(m(n-L))$, and also bounded by $\mathcal{O}(mL)$

Total time complexity $\mathcal{O}(\sigma n + \min\{mL, L(n-L)\})$

- If $L < \frac{m}{2}$, then $\mathcal{O}(mL) = \mathcal{O}(L(n-L))$



Total work $\approx z^2 + (m-z)(n-L+z+1) = \mathcal{O}(m(n-L))$,
and also bounded by $\mathcal{O}(mL)$

Total time complexity $\mathcal{O}(\sigma n + \min\{mL, L(n-L)\})$

- If $L < \frac{m}{2}$, then $\mathcal{O}(mL) = \mathcal{O}(L(n-L))$
- If $L \geq \frac{m}{2}$, then $\mathcal{O}(m(n-L)) = \mathcal{O}(L(n-L))$