12.1 Computing alignments in only linear space

20160329

Chanyoung Song

Computing alignment in O(nm) space

- S_1 = cacd
- S_2 = cadb

s	a	b	c	d	-
a	1	-3	-2	0	-1
b		3	-2	-1	0
c			0	-4	-2
d				3	-1
-					0

V(I,j)	-	c	a	c	d
-					
c					
a					
d					
b					

V(i,j) = max[V(i-1, j-1) +s(S_1(i), S_2(j)), V(i-1,j) + s(S_1(i), -), V(i,j-1)_s(-,S_2(j))]

Computing alignment in O(nm) space

- S_1 = cacd
- S_2 = cadb

s	a	b	c	d	-
a	1	-3	-2	0	-1
b		3	-2	-1	0
c			0	-4	-2
d				3	-1
-					0

V(I,j)	-	c	a	c	d
-	0	-2	-3	-5	-6
c	-2				
a					
d					
b					

-2-2 = -4

0+0 = 0

-2-2 = -4

$V(i,j) = \max[\, V(i-1, j-1) + s(S_1(i), S_2(j)),\ V(i-1,j) + s(S_1(i), -),\ V(i,j-1)\_s(-,S_2(j))\,]$

Computing alignment in O(nm) space

- S_1 = cacd
- S_2 = cadb

s	a	b	c	d	-
a	1	-3	-2	0	-1
b		3	-2	-1	0
c			0	-4	-2
d				3	-1
-					0

V(I,j)	-	c	a	c	d
-	0	-2	-3	-5	-6
c	-2	0			
a					
d					
b					

- Compute 3-time
- Space for 1 value, max 3 pointers

$V(i,j) = \max[\ V(i-1, j-1) + s(S_1(i), S_2(j)),\ V(i-1,j) + s(S_1(i), -),\ V(i,j-1)\_s(-,S_2(j))\ ]$

Computing alignment in O(nm) space

- S_1 = cacd
- S_2 = cadb

s	a	b	c	d	-
a	1	-3	-2	0	-1
b		3	-2	-1	0
c			0	-4	-2
d				3	-1
-					0

V(I,j)	-	c	a	c	d
-	0	-2	-3	-5	-6
c	-2	0	-1	-3	-4
a	-3	-1	1	-1	-2
d	-4	-2	0	-2	2
b	-4	-2	0	-2	2

V(i,j) = max[V(i-1, j-1) +s(S_1(i), S_2(j)), V(i-1,j) + s(S_1(i), -), V(i,j-1)_s(-,S_2(j))]

Computing Alignments

- Dynamic Programming ($|S_1| = n$, $|S_2| = m$)
  - Time Complexity : O(nm)
  - Space Complexity : ~~O(nm)~~ O(m)

How about computing only V(n,m),
Not an alignment ?

Space Reduction for Computing Similarity in O(m) space

- S_1 = cacd
- S_2 = cadb

s	a	b	c	d	-
a	1	-3	-2	0	-1
b		3	-2	-1	0
c			0	-4	-2
d				3	-1
-					0

V(I,j)	-	c	a	c	d
-	0	-2	-3	-5	-6
c					
a					
d					
b					

V(i,j) = max[V(i-1, j-1) +s(S_1(i), S_2(j)), V(i-1,j) + s(S_1(i), -), V(i,j-1)_s(-,S_2(j))]

Space Reduction for Computing Similarity in O(m) space

- S_1 = cacd
- S_2 = cadb

s	a	b	c	d	-
a	1	-3	-2	0	-1
b		3	-2	-1	0
c			0	-4	-2
d				3	-1
-					0

V(I,j)	-	c	a	c	d
-	0	-2	-3	-5	-6
c	-2				
a					
d					
b					

V(i,j) = max[V(i-1, j-1) +s(S_1(i), S_2(j)), V(i-1,j) + s(S_1(i), -), V(i,j-1)_s(-,S_2(j))]

Space Reduction for Computing Similarity in O(m) space

- S_1 = cacd
- S_2 = cadb

s	a	b	c	d	-
a	1	-3	-2	0	-1
b		3	-2	-1	0
c			0	-4	-2
d				3	-1
-					0

V(I,j)	-	c	a	c	d
-	0	-2	-3	-5	-6
c	-2				
a					
d					
b					

V(i,j) = max[V(i-1, j-1) +s(S_1(i), S_2(j)), V(i-1,j) + s(S_1(i), -), V(i,j-1)_s(-,S_2(j))]

Space Reduction for Computing Similarity in O(m) space

- S_1 = cacd
- S_2 = cadb

s	a	b	c	d	-
a	1	-3	-2	0	-1
b		3	-2	-1	0
c			0	-4	-2
d				3	-1
-					0

V(I,j)	-	c	a	c	d
-	0	-2	-3	-5	-6
c	-2	0			
a					
d					
b					

V(i,j) = max[V(i-1, j-1) +s(S_1(i), S_2(j)), V(i-1,j) + s(S_1(i), -), V(i,j-1)_s(-,S_2(j))]

Space Reduction for Computing Similarity in O(m) space

- S_1 = cacd
- S_2 = cadb

s	a	b	c	d	-
a	1	-3	-2	0	-1
b		3	-2	-1	0
c			0	-4	-2
d				3	-1
-					0

V(I,j)	-	c	a	c	d
-	0	-2	-3	-5	-6
c	-2	0	-1	-3	-4
a					
d					
b					

i-th row

ONLY NEED i-1-th ROW
FOR COMPUTING i-th ROW

V(i,j) = max[V(i-1, j-1) +s(S_1(i), S_2(j)), V(i-1,j) + s(S_1(i), -), V(i,j-1)_s(-,S_2(j))]

Definition

- For any string α,

  let α^r denote the **reverse of string $\boldsymbol{\alpha}$**

α = abcde

α^r = edcba

Definition

- Given strings S_1 and S_2,
- Define $V^r(i, j)$ as the similarity of
  1) the string consisting of the first i characters of S_1^r, and
  2) the string consisting of the first j characters of S_2^r.

- Equivalently $V^r(i, j)$ is the similarity of
  1) the last i characters of S_1 and
  2) the last j characters of S_2

S_1= abdddd
S_2= abcdcd

reverse

S_1^r= ddddba
S_2^r= dcdcba

$V^r(3, 3)$?

Lemma 12.1.1

- $V(n, m) = \max_{0<=k<=m}[\ V(n/2, k) + V^r(n/2, m\text{-}k)\ ]$

Ex> S_1= abdddd, S_2= abcdcd

- $V(6, 6) = \max_{0<=k<=m}[\ V(3, k) + V^r(3, m\text{-}k)\ ]$
- $V(3, k)$: S_1= $\underbrace{\boxed{\text{a b d}}}_{3개}$ d d d S_2= $\underbrace{\boxed{\text{a b c d}}}_{k개}$ c d
- $V^r(3, m\text{-}k)$: S_1= a b d $\underbrace{\boxed{\text{d d d}}}_{3개}$ S_2= a b c d $\underbrace{\boxed{\text{c d}}}_{m\text{-}k개}$
- $V(3, k) + V^r(3, m\text{-}k)$

Lemma 12.1.1

- $V(n, m) = \max_{0<=k<=m}[\ V(n/2, k) + V^r(n/2, m-k)\ ]$

Proof>

- $S_t[1,..,i]$: the prefix of S_t **(the first i characters)**
- $S_t^r[1,..,i]$: the reverse of the suffix of S_t **(the last i characters)**

t = {1, 2}

For fixed any position k' in S_2,

An alignment of $S_1[1,..,n/2]$ and $S_2[1,..,k']$ ➡ $V(n/2, k')$

An alignment of $S_1[n/2+1,..,n]$ and $S_2[k'+1,..,m]$ ➡ $V^r(n/2, m - k')$

$V(n/2, k') + V^r(n/2, m - k') \leq \max_{0<=k<=m}[\ V(n/2, k) + V^r(n/2, m-k)\ ] \leq V(n, m)$

Lemma 12.1.1

- $V(n, m) = \max_{0<=k<=m}[\, V(n/2, k) + V^r(n/2, m-k)\,]$

n/2

S_1	c	a	c	d	-
S_2	c	a	-	d	b

k'

Proof>

An optimal alignment of S_1 and S_2.

Let k' be the right-most position in S_2 that is aligned with a character at or before position n/2 in S_1.

$S_1[1,..,n/2]$ and $S_2[1,..,k']$ ➡ $p \leq V(n/2,k')$

$S_1[n/2+1,..,n]$ and $S_2[k'+1,..,m]$ ➡ $q \leq V^r(n/2, m - k')$

$V(n, m) \leq V(n/2, k') + V^r(n/2, m - k') \leq \max_{0<=k<=m}[\, V(n/2, k) + V^r(n/2, m-k)\,]$

Definition

- Let k^* be a position k

  that maximizes [$V(n/2, k) + V^r(n/2, m-k)$]

By Lemma 12.1.1,

There is an optimal alignment whose traceback path in the full dynamic programming table goes through cell $(n/2, k^*)$

n/2

S_1	c	a	c	d	-
S_2	c	a	-	d	b

k^*

V(I,j)	-	c	a	c	d
-	0	-2	-3	-5	-6
c	-2	0	-1	-3	-4
a	-3	-1	1	-1	-2
d	-4	-2	0	-2	2
b	-4	-2	0	-2	2

Definition

- Let $L_{n/2}$ be the subpath of L that

  starts with the last node of L in row n/2-1 and
  ends with the first node of L in row n/2 + 1

	V(I,j)	-	c	a	c	d
	-	0	-2	-3	-5	-6
	c	-2		-1	-3	-4
n/2→	a	-3	-1			-2
	d	-4	-2	0	-2	
	b	-4	-2	0	-2	2

Lemma 12.1.2

- A position k* in row $n/2$ can be found in $O(nm)$ time and $O(m)$ space.
- Moreover, a subpath $L_{n/2}$ can be found and stored in those time and space bounds.

1) Find a position k* in row n/2 in O(nm) time and O(m) space

	V(I,j)	-	c	a	c	d	b	a	a
	-								
	c								
	a								
	b								
n/2	d								
	c								
	a								
	a								

1) For $V(n/2, k)$, $0 \leq k \leq m$
   - Time : $O(nm/2)$
   - Space : $O(m)$
   - Pointer set at row n/2

- Let k^* be a position k

  that maximizes $[\ V(n/2, k) + V^r(n/2, m-k)\ ]$

1) Find a position k* in row n/2 in O(nm) time and O(m) space

V(I,j)	-	c	a	c	d	b	a	a
-								
c								
a								
b								
d	-4	-2	0	-2	2	2	1	0
c								
a								
a								

n/2

1) For V(n/2, k), $0 \leq k \leq m$
- Time : O(nm/2)
- Space : O(m)
- Pointer set at row n/2

- Let k^* be a position k that maximizes [$V(n/2, k) + V^r(n/2, m-k)$]

1) Find a position k* in row n/2 in O(nm) time and O(m) space

V(I,j)	-	c	a	c	d	b	a	a
-								
c								
a								
b								
d								
c								
a								
a								

n/2

1) For $V(n/2, k)$, $0 \leq k \leq m$
   - Time : $O(nm/2)$
   - Space : $O(m)$
   - Pointer set at row n/2

2) For $V^r(n/2, m-k)$, $0 \leq k \leq m$
   - Time : $O(nm/2)$
   - Space : $O(m)$
   - Pointer set at row n/2

- Let k^* be a position k

  that maximizes $[\ V(n/2, k) + V^r(n/2, m-k)\ ]$

1) Find a position k* in row n/2 in O(nm) time and O(m) space

k^*

V(I,j)	-	c	a	c	d	b	a	a
-								
c								
a								
b								
n/2 d								
c								
a								
a								

1) For $V(n/2, k)$, $0 \le k \le m$
   - Time : $O(nm/2)$
   - Space : $O(m)$
   - Pointer set at row n/2

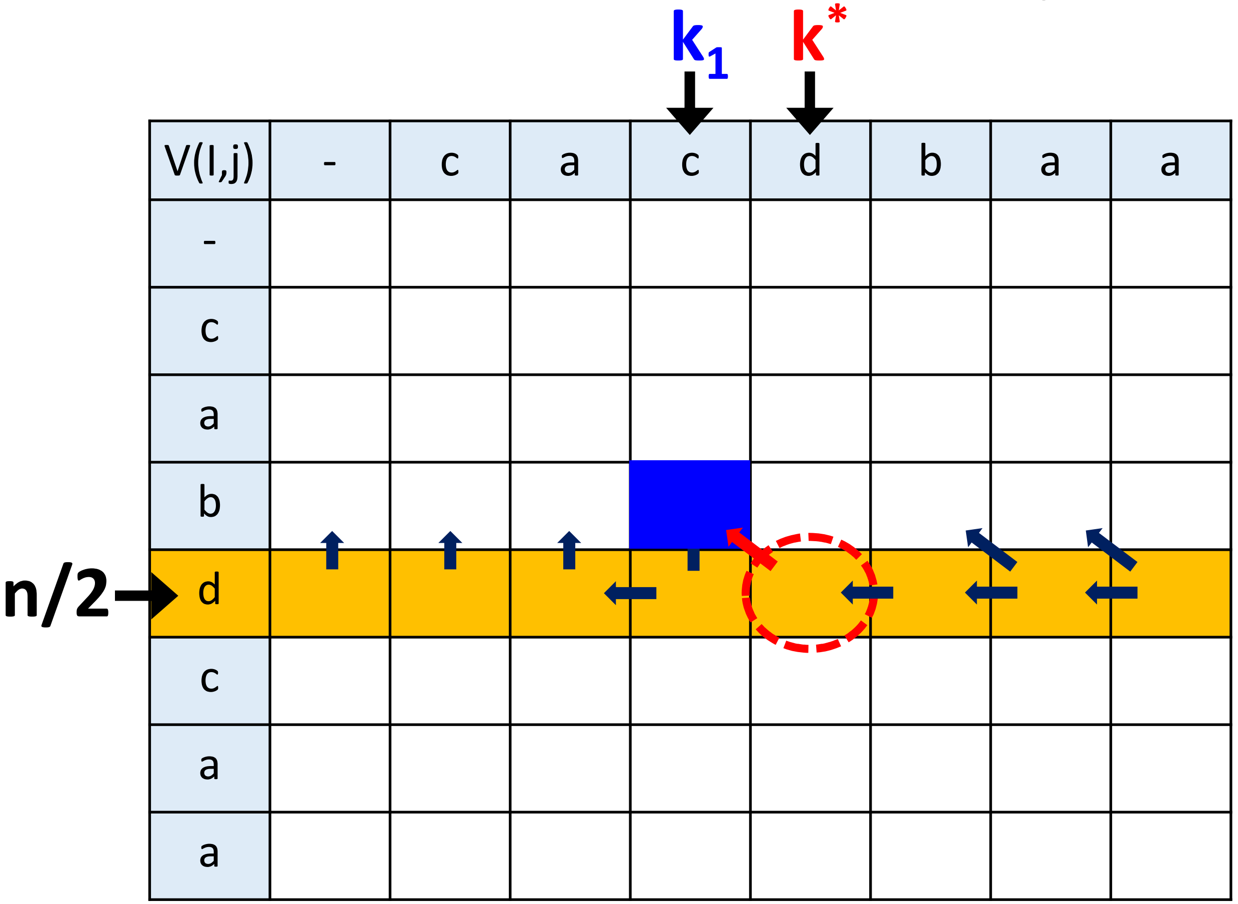
2) For $V^r(n/2, m\text{-}k)$, $0 \le k \le m$
   - Time : $O(nm/2)$
   - Space : $O(m)$
   - Pointer set at row n/2

A position k* in row n/2 can be found in O(nm) time and O(m) space.

- Let k^* be a position k

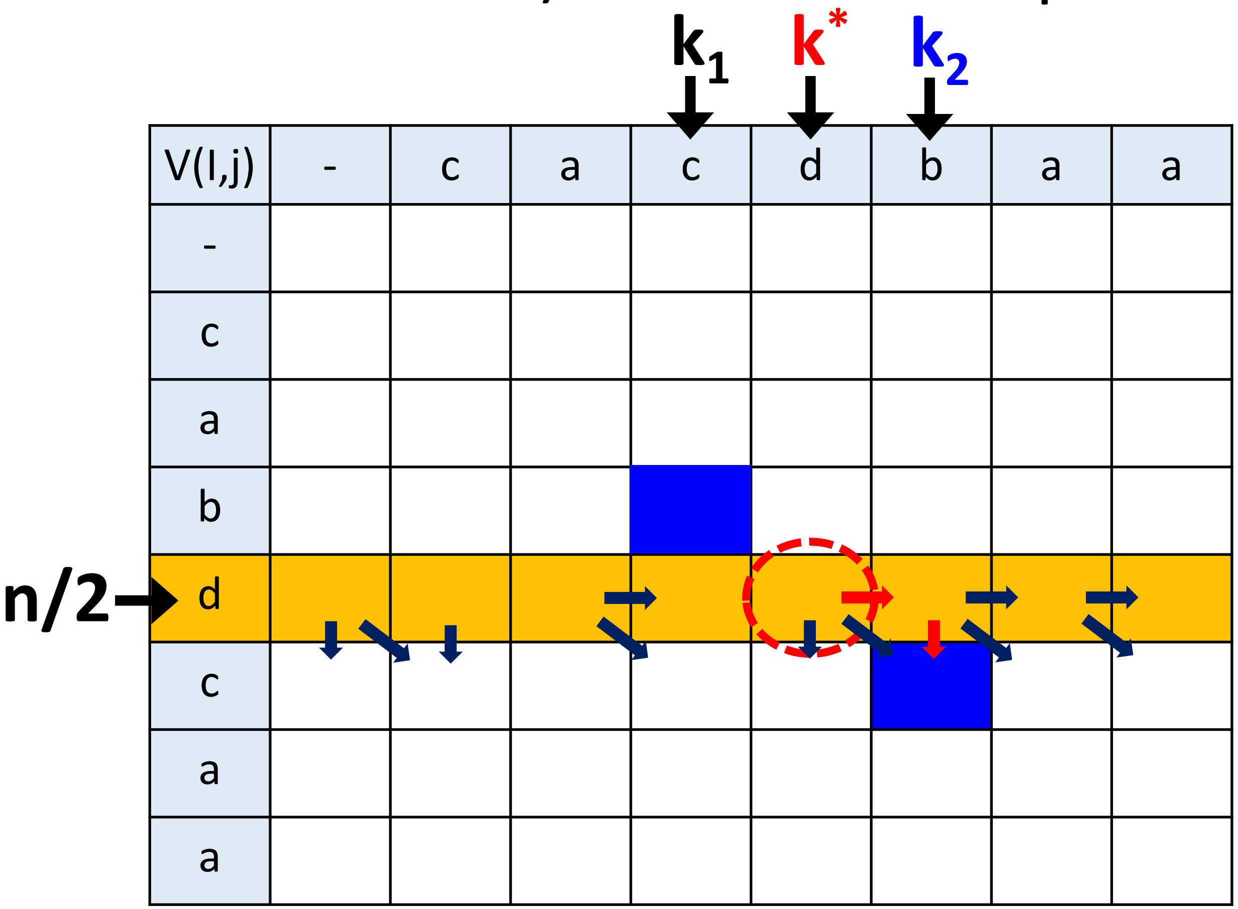
  that maximizes $[\, V(n/2, k) + V^r(n/2, m\text{-}k) \,]$

2) Find a Subpath $L_{n/2}$



k_1 : Col. Idx of 1st cell in n/2 - 1 row from cell (n/2, k*)

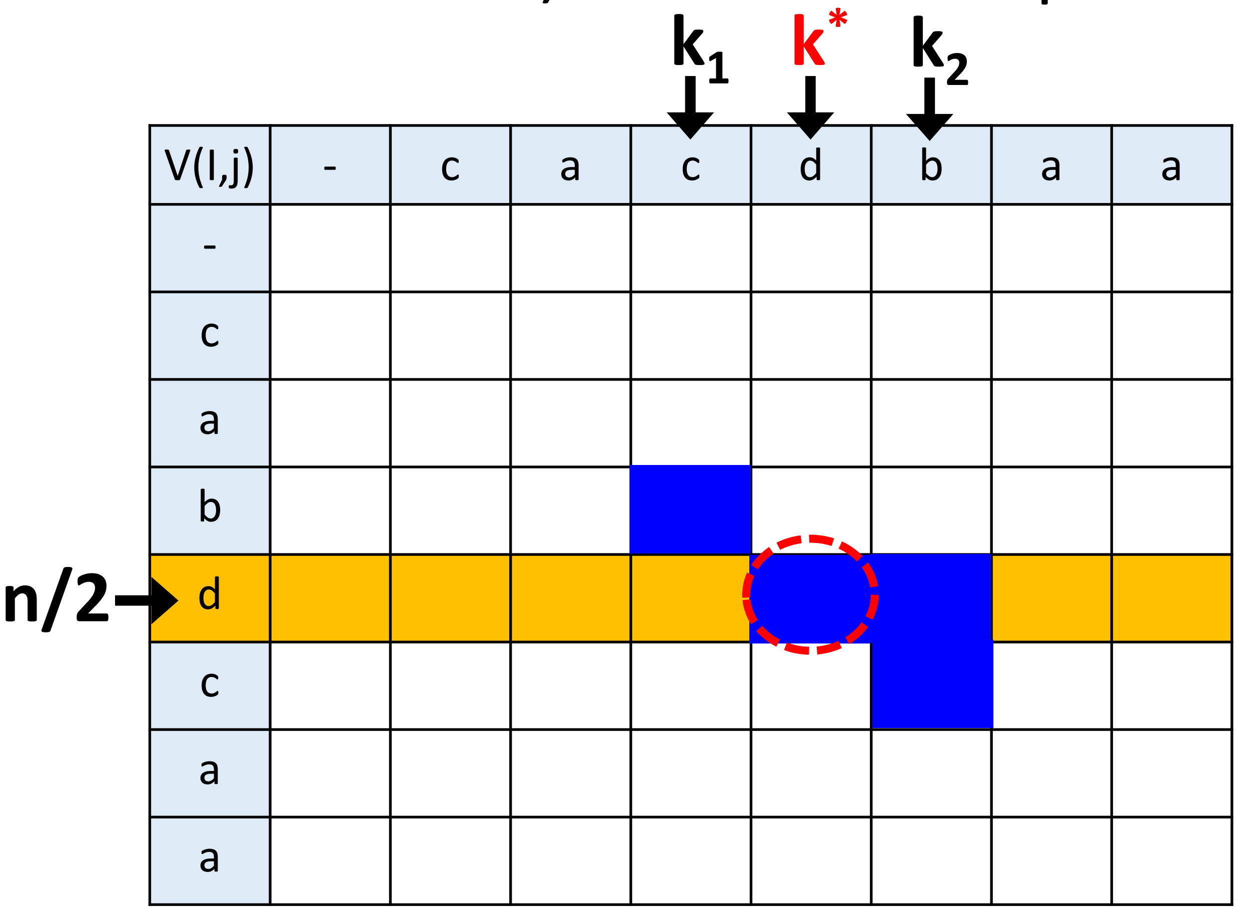
2) Find a Subpath $L_{n/2}$



k_1 : Col. Idx of 1st cell in n/2 - 1 row from cell (n/2, k^*)

k_2 : Col. Idx of 1st cell in n/2 + 1 row from cell (n/2, k^*)

2) Find a Subpath $L_{n/2}$



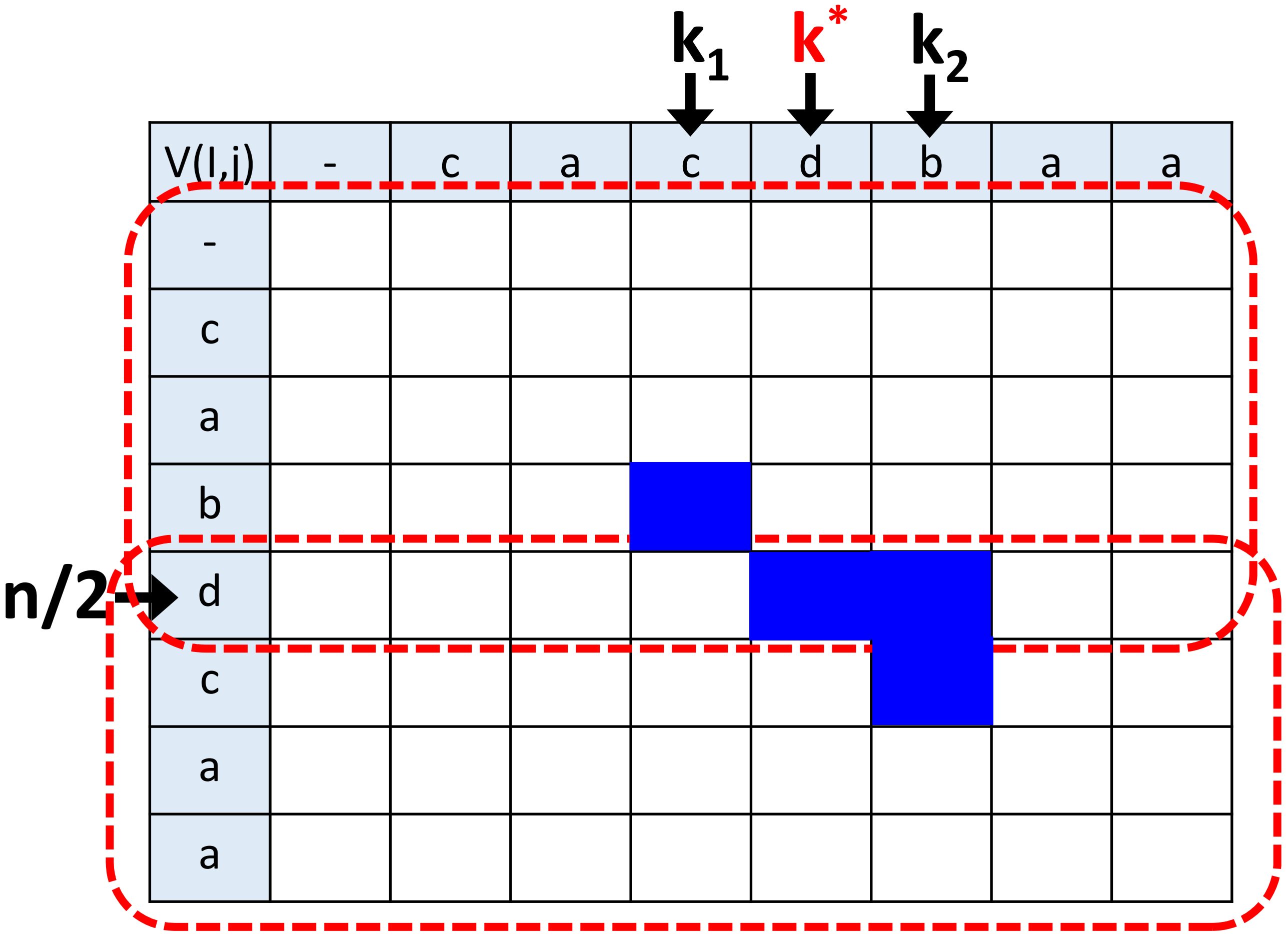
- $L_{n/2}$ (subpath)

A subpath $L_{n/2}$ can be found and stored in O(nm) time and O(m) space.

k_1 : Col. Idx of 1st cell in n/2 - 1 row from cell (n/2, k^*)

k_2 : Col. Idx of 1st cell in n/2 + 1 row from cell (n/2, k^*)

Full Idea : Use Recursion



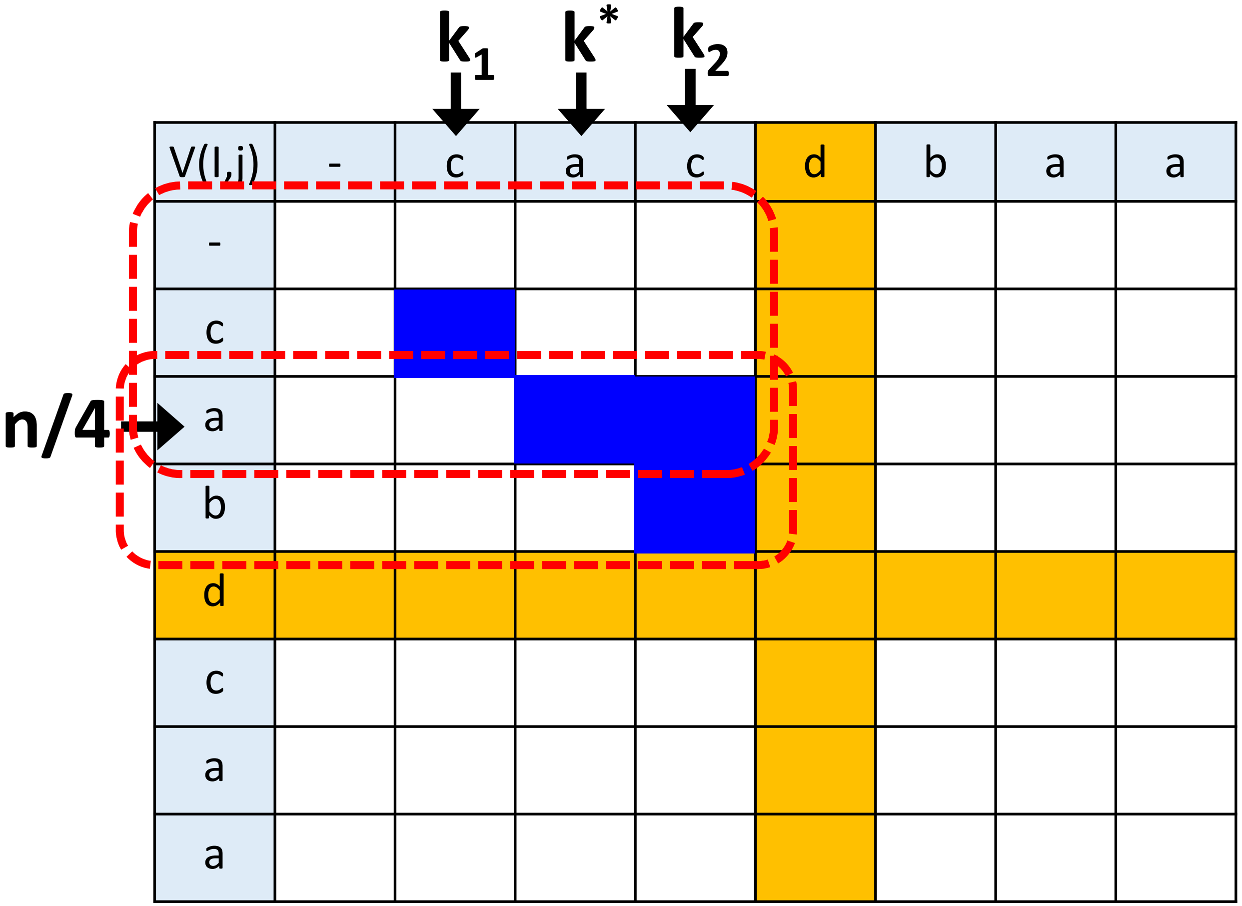
- $L_{n/2}$ (subpath)
- k^*, k_1, k_2

- Time : cnm

  $= c(n/2)m \times 2$

- Space : O(m)

Full Idea : Use Recursion



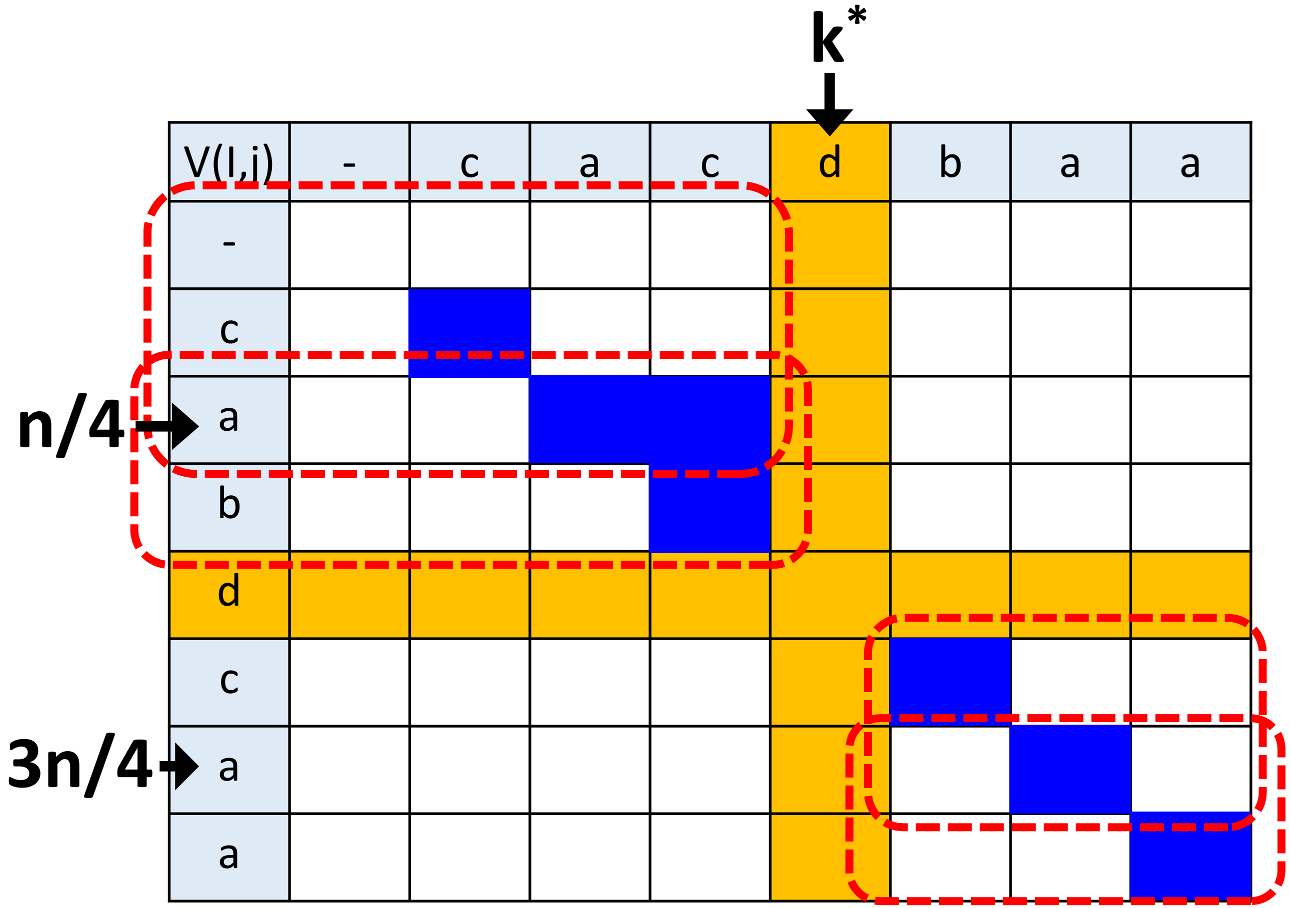
- $L_{n/4}$ (subpath)
- k^*, k_1, k_2

Full Idea : Use Recursion

							k_1	k^*	k_2
	V(I,j)	-	c	a	c	d	b	a	a
	-								
	c								
	a								
	b								
	d								
	c								
3n/4 →	a								
	a								

- $L_{3n/4}$ (subpath)
- k^*, k_1, k_2

Full Idea : Use Recursion



- $L_{n/4}$, $L_{3n/4}$ (subpath)
- k …

- Time : (1/2)cnm

$$= c(n/4)(k^*)\times 2$$
$$+ c(n/4)(m - k^*) \times 2$$

- Space : O(m)

Algorithm Summary

- Procedure OPTA(r, r', c, c')

  Begin

  h := (r + r') /2

  Find index k^*, k_1, k_2, and subpath $L_{h.}$ (By Dynamic Programing)

  Call OPTA(r, h-1, c, k_1)
  Output subpath L_h
  Call OPTA(h+1, r', k_2, c')

  End

Row Size : n
-> depth : logn

Theorem 12.1.1

- Using the procedure OPTA, an optimal alignment of two strings of length n and m can be found in $\sum_{i=1}^{logn} cnm/2^{i-1} \leq 2cnm$ time and O(m) space.