Optimisation des algorithmes d’alignement de séquences

Nadia El-Mabrouk

Algorithmes de programmation dynamique

- Alignement local ou global
  - *O(mn)* en temps et en espace (pire des cas et meilleur des cas).

Une grande variété de méthodes d'optimisation en temps et en espace. Dans ce cours:

I. **Algorithme de Hirschberg** pour l'alignement global en espace linéaire; application à l'alignement local;

II. **Alignement par bande** pour une optimisation en temps et en espace de l'alignement global;

III. **Algorithme des quatre russes** pour une amélioration asymptotique en temps de l'alignement global;

IV. **Algorithme de Crochemore et Landau** pour une amélioration asymptotique en temps de l'alignement global et local.

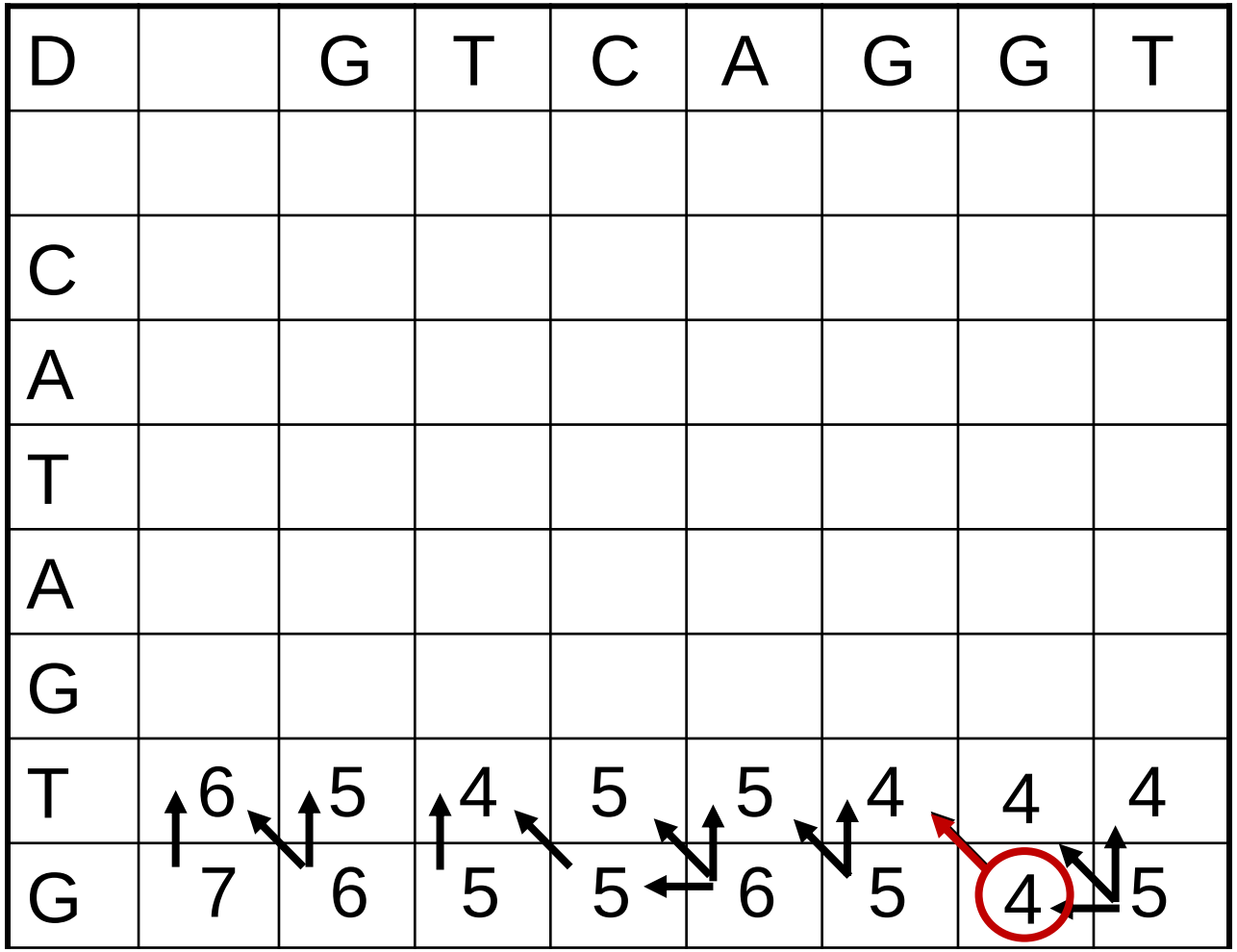
I- Algorithme de Hirschberg

Complexité en espace de l'alignement de séquences

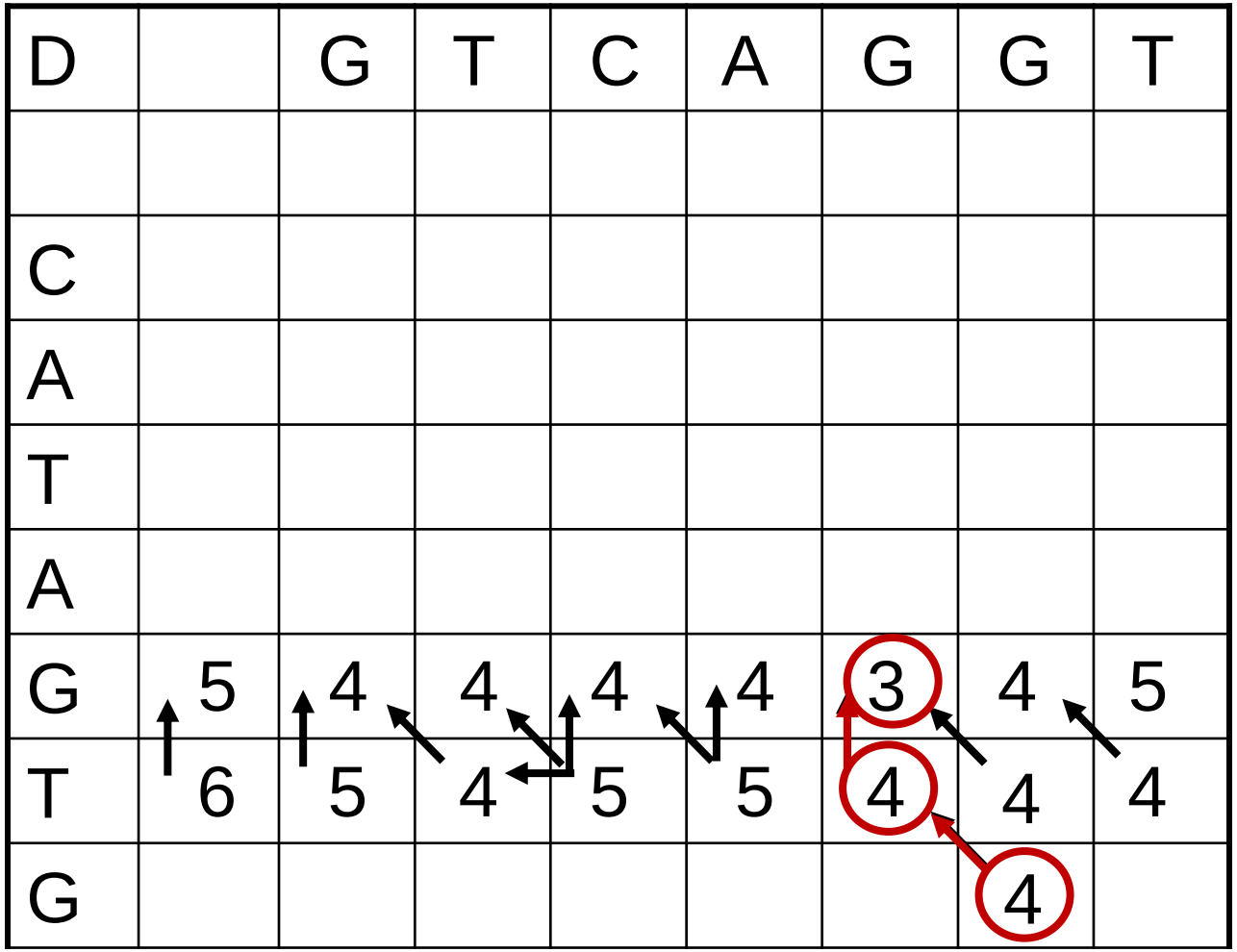
- S_1 de taille m; S_2 de taille n (m lignes, n colonnes)
- Valeur d'un alignement global optimal: $V(S,T)$
- Algorithme classique pour l'alignement global: Espace O(mn)
  - Par exemple, deux séquences de taille 20000 nécessitera environ 1.6GB de RAM pour la table de programmation dynamique (si chaque case est représentée par un entier sur 4 bits)
  - Impossible de considérer de longues séquences d'ADN.
- Calcul de $V(S,T)$ **sans alignement**:
  - Pour le calcul des valeurs d'une ligne i, on a besoin uniquement de la ligne $i-1$. A chaque étape, remplir la ligne courante a l'aide de la ligne précédente → Espace nécessaire : $2n$
  - On peut encore réduire l'espace a ***n+1***
- Une utilisation naïve de cette stratégie permet de retrouver un alignement en temps $O(n^2m)$

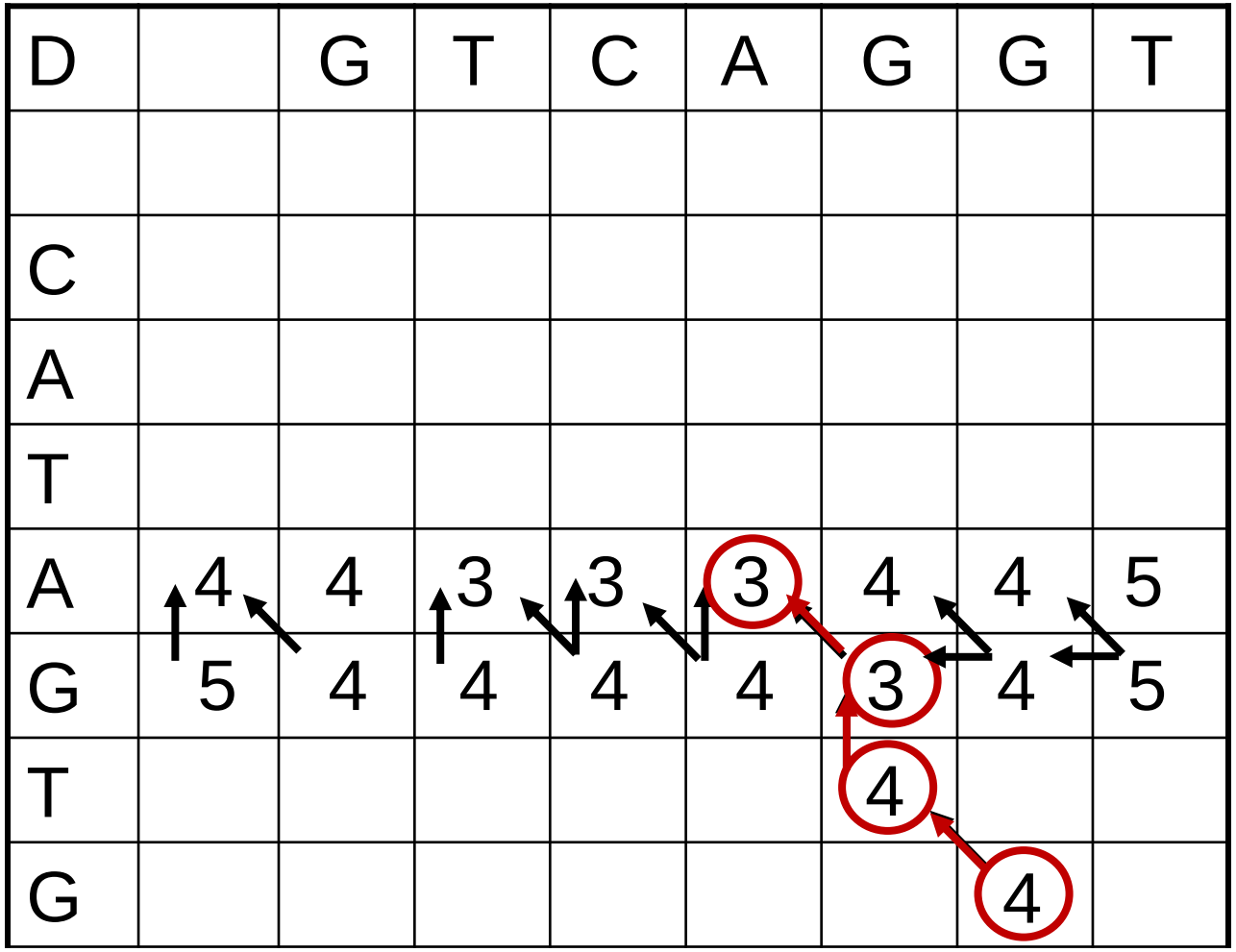
D		G	T	C	A	G	G	T
	0	1	2	3	4	5	6	7
C	1	1	2	2	3	4	5	6
A	2	2	2	3	2	3	4	5
T	3	3	2	3	3	3	4	4
A								
G								
T								
G								

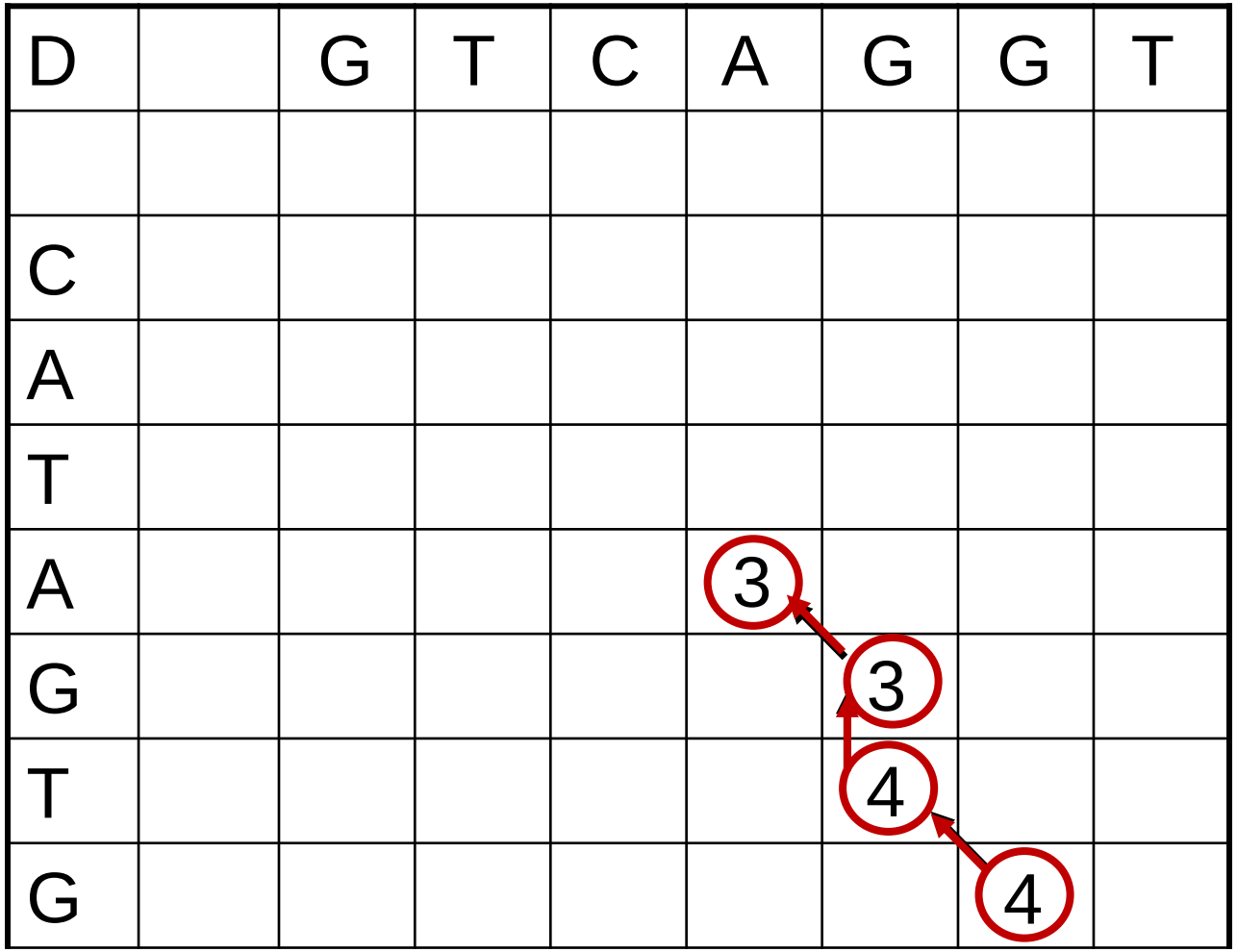
D		G	T	C	A	G	G	T
C								
A								
T	3	3	2	3	3	3	4	4
A	4	4	3	3	3	4	4	5
G	5	4	4	4	4	3	4	5
T	6	5	4	5	5	4	4	4
G	7	6	5	5	6	5	4	5


D G T C A G G T
C
A
T
A
G
T 6 5 4 5 5 4 4 4
G 7 6 5 5 6 5 4 5

D		G	T	C	A	G	G	T
	0	1	2	3	4	5	6	7
C	1	1	2	2	3	4	5	6
A	2	2	2	3	2	3	4	5
T	3	3	2	3	3	3	4	4
A	4	4	3	3	3	4	4	5
G	5	4	4	4	4	3	4	5
T	6	5	4	5	5	4	4	4
G							4	


D G T C A G G T
C
A
T
A
G 5 4 4 4 4 3 4 5
T 6 5 4 5 5 4 4 4
G 4

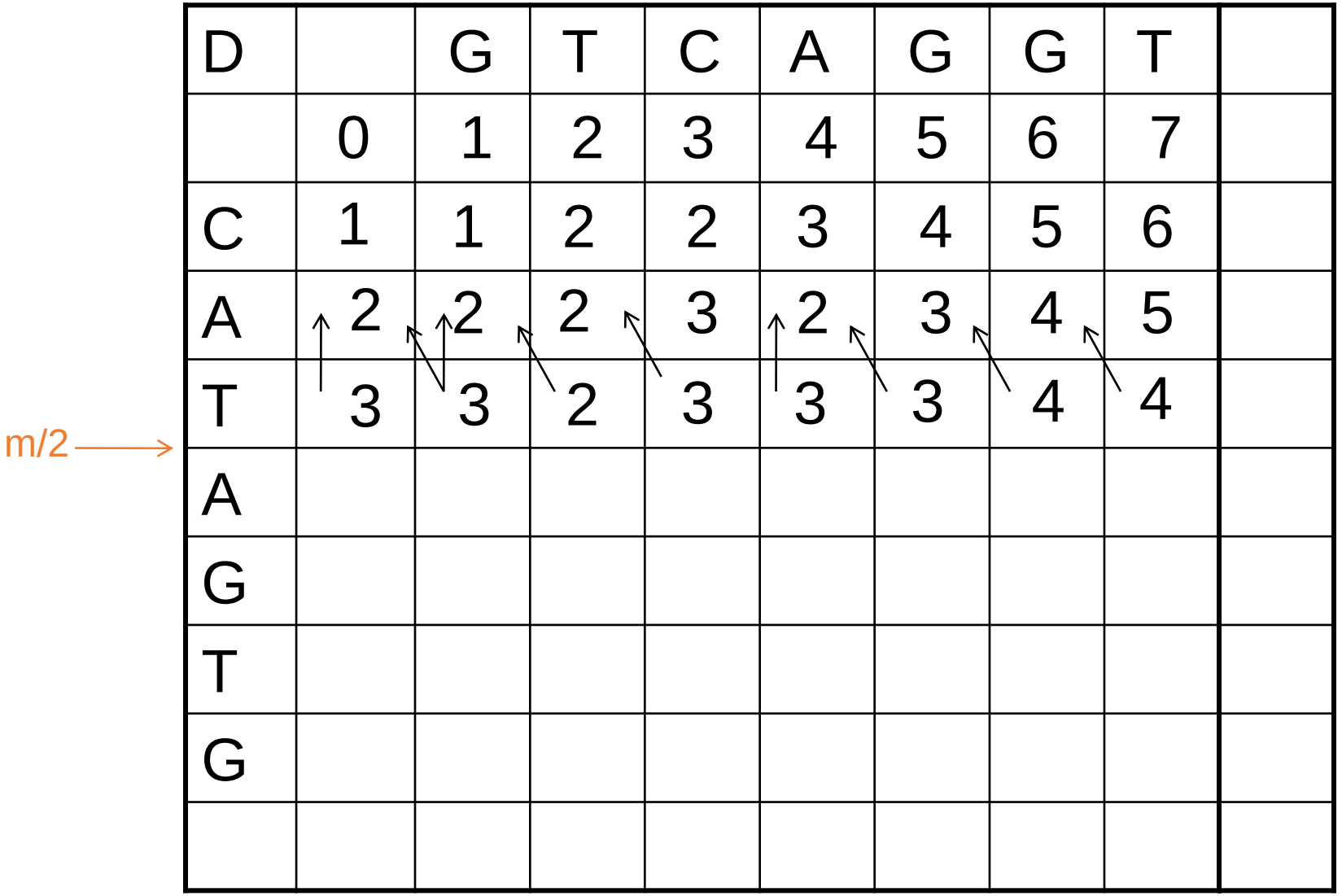

D G T C A G G T
C
A
T
A 4 4 3 3 3 4 4 5
G 5 4 4 4 4 3 4 5
T 4
G 4

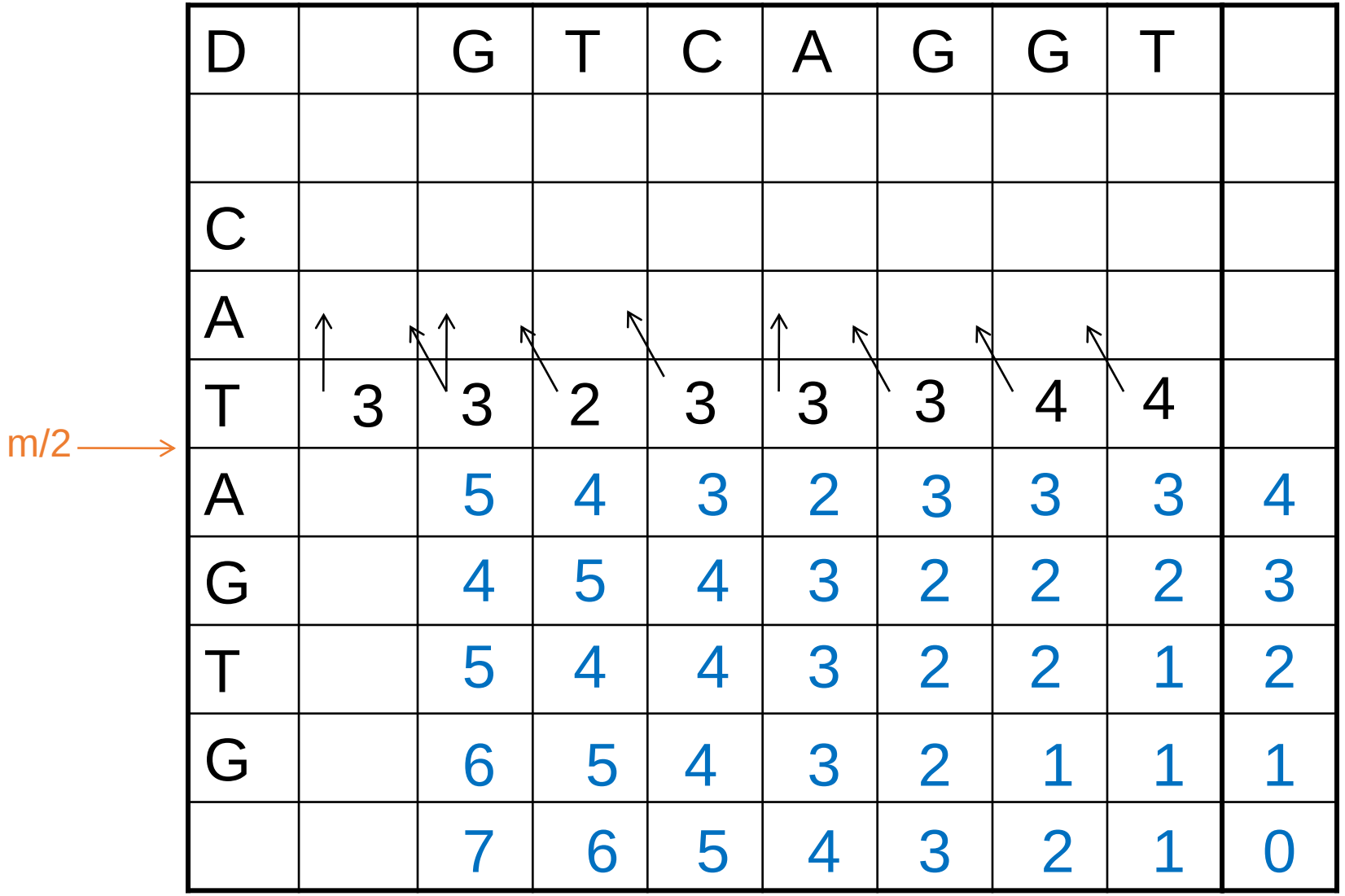

D
G
T
C
A
G
G
T
C
A
T
A
G
T
G
3
3
4
4

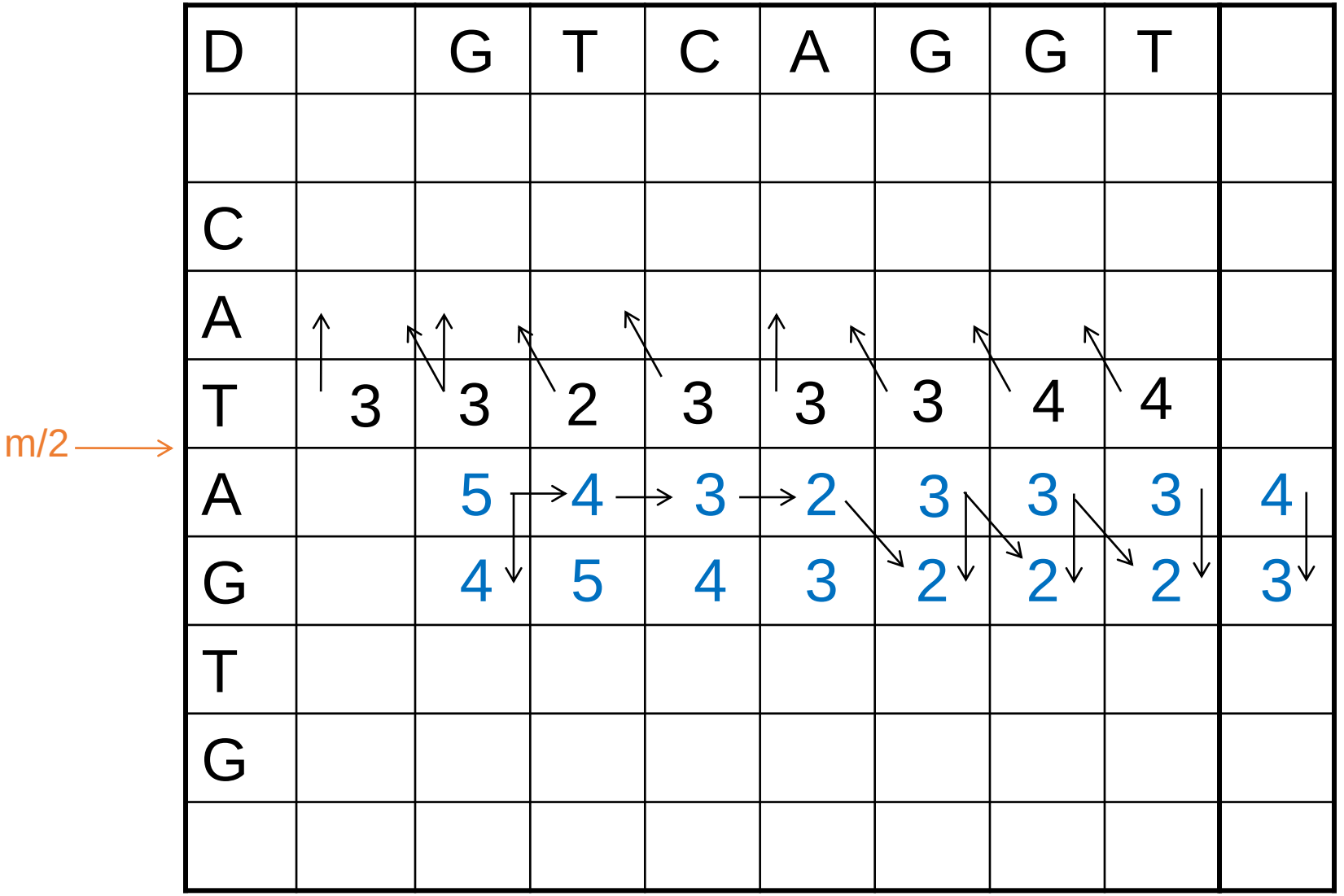
Algorithme de Hirschberg

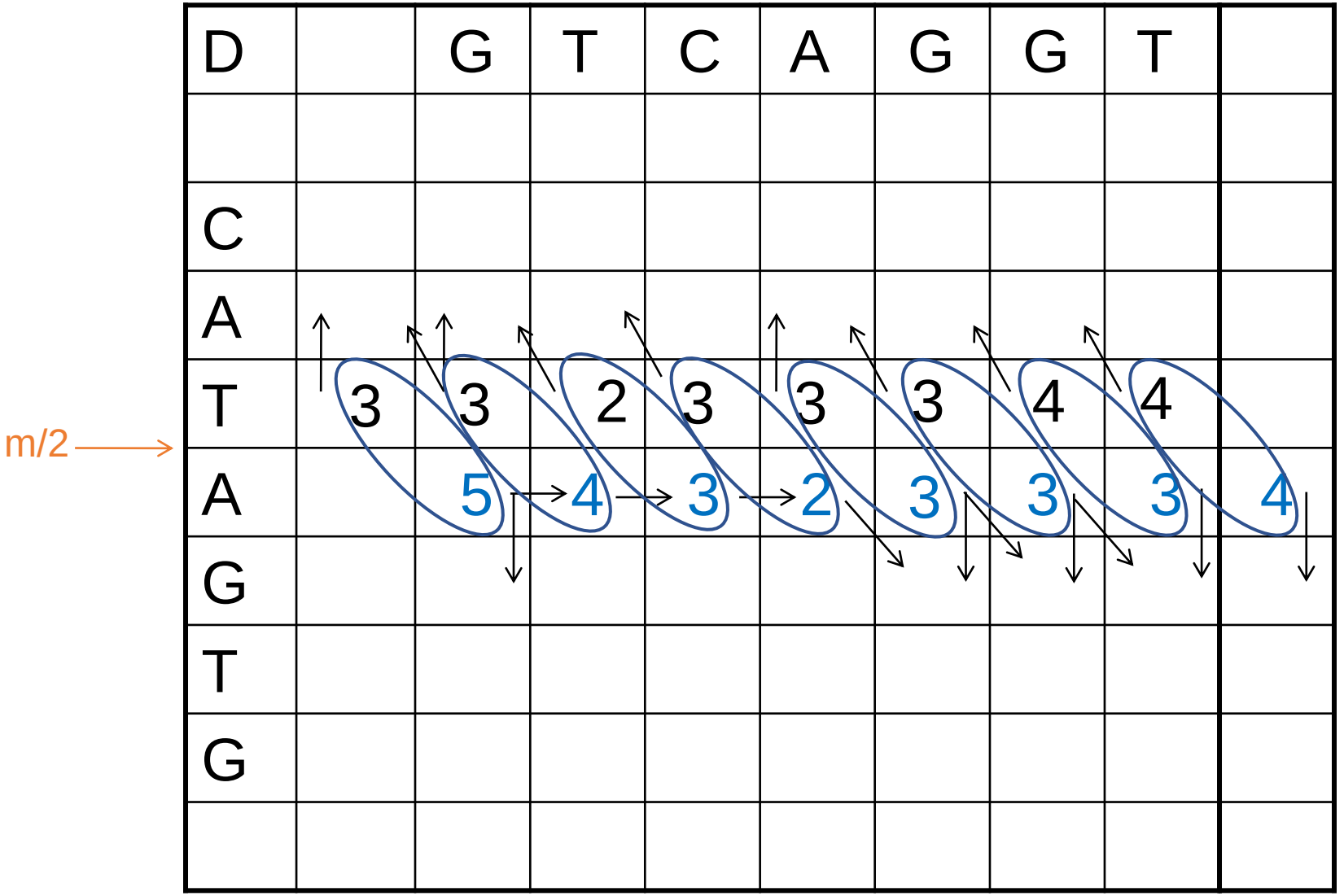
Comment utiliser un espace linéaire pour trouver un alignement, sans trop augmenter le temps de calcul?

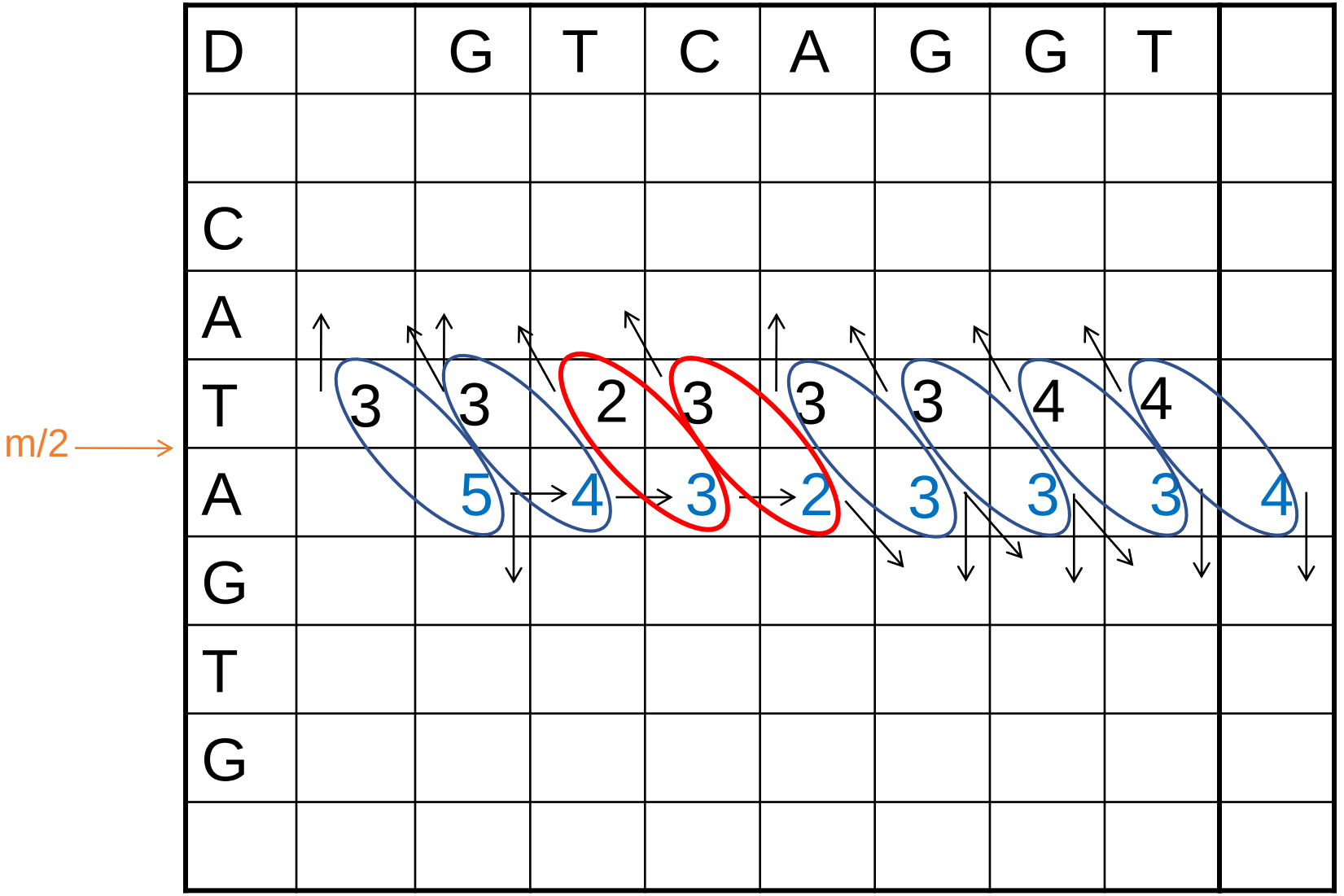
- **Complexité en espace en O(m+n)**, et complexité en temps dans le pire des cas double par rapport a l'algorithme classique pour l'alignement global, donc, toujours en O(mn).
  - Par exemple, deux séquences de taille 20000 nécessitera environ 160KB(10^3) plutôt que 1.6 GB (10^9) de RAM.
- Idée générale: Utiliser une stratégie « diviser pour régner » - Plutôt que de reconstituer le chemin à partir du bas de la table, le reconstituer à partir du centre.

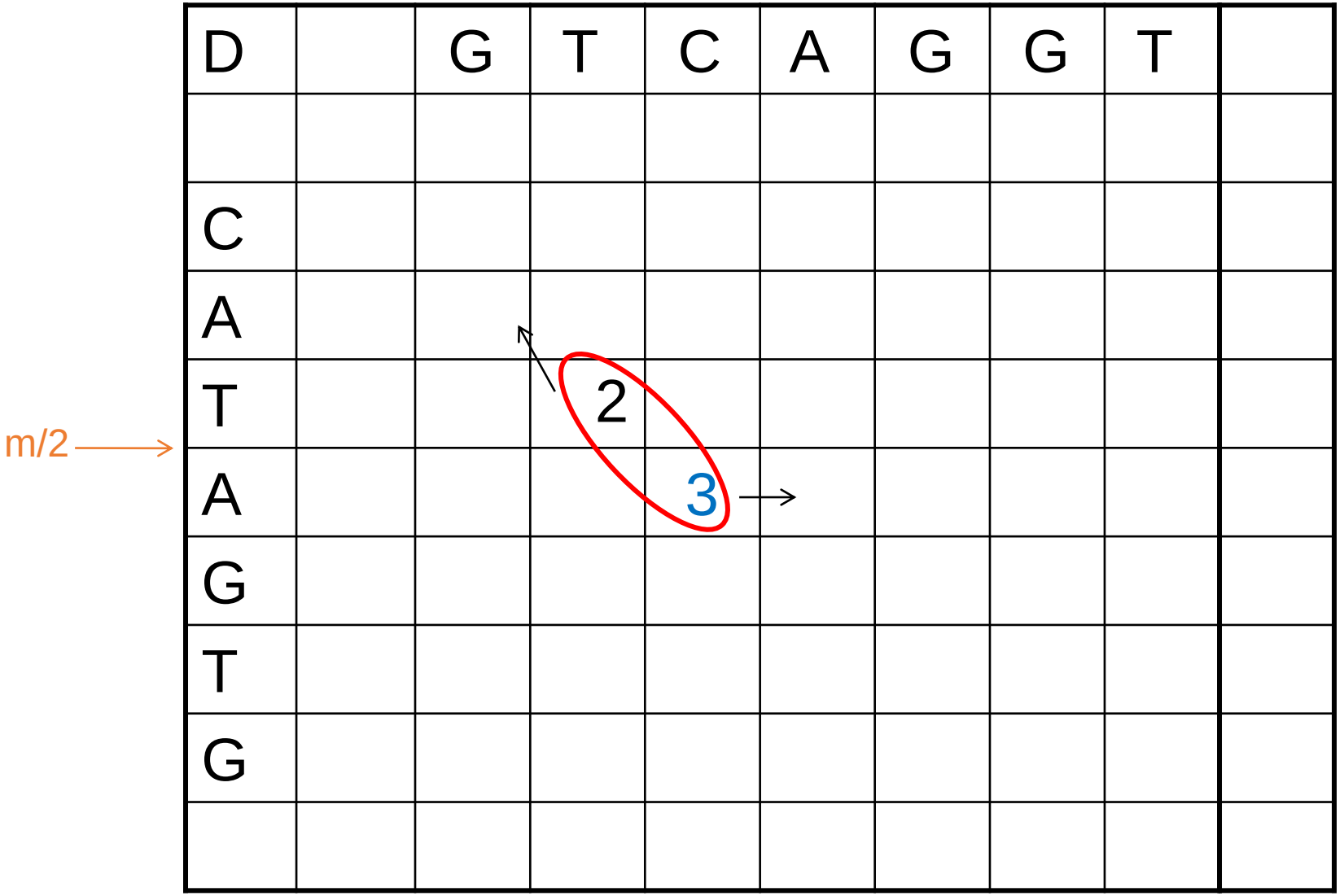

D G T C A G G T
0 1 2 3 4 5 6 7
C 1 1 2 2 3 4 5 6
A 2 2 2 3 2 3 4 5
T 3 3 2 3 3 3 4 4
m/2
A
G
T
G

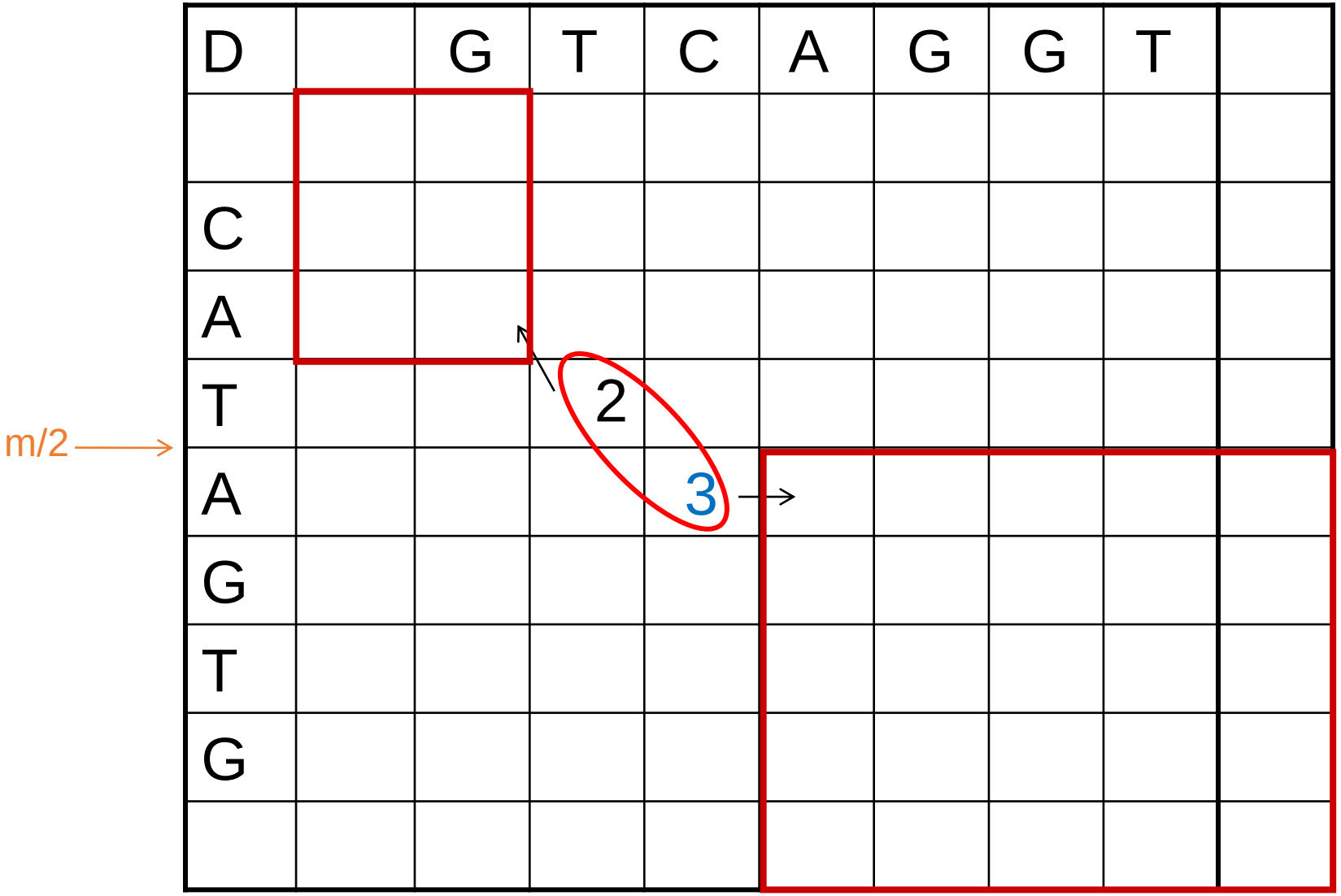

D G T C A G G T
C
A
T 3 3 2 3 3 3 4 4
m/2
A 5 4 3 2 3 3 3 4
G 4 5 4 3 2 2 2 3
T 5 4 4 3 2 2 1 2
G 6 5 4 3 2 1 1 1
7 6 5 4 3 2 1 0

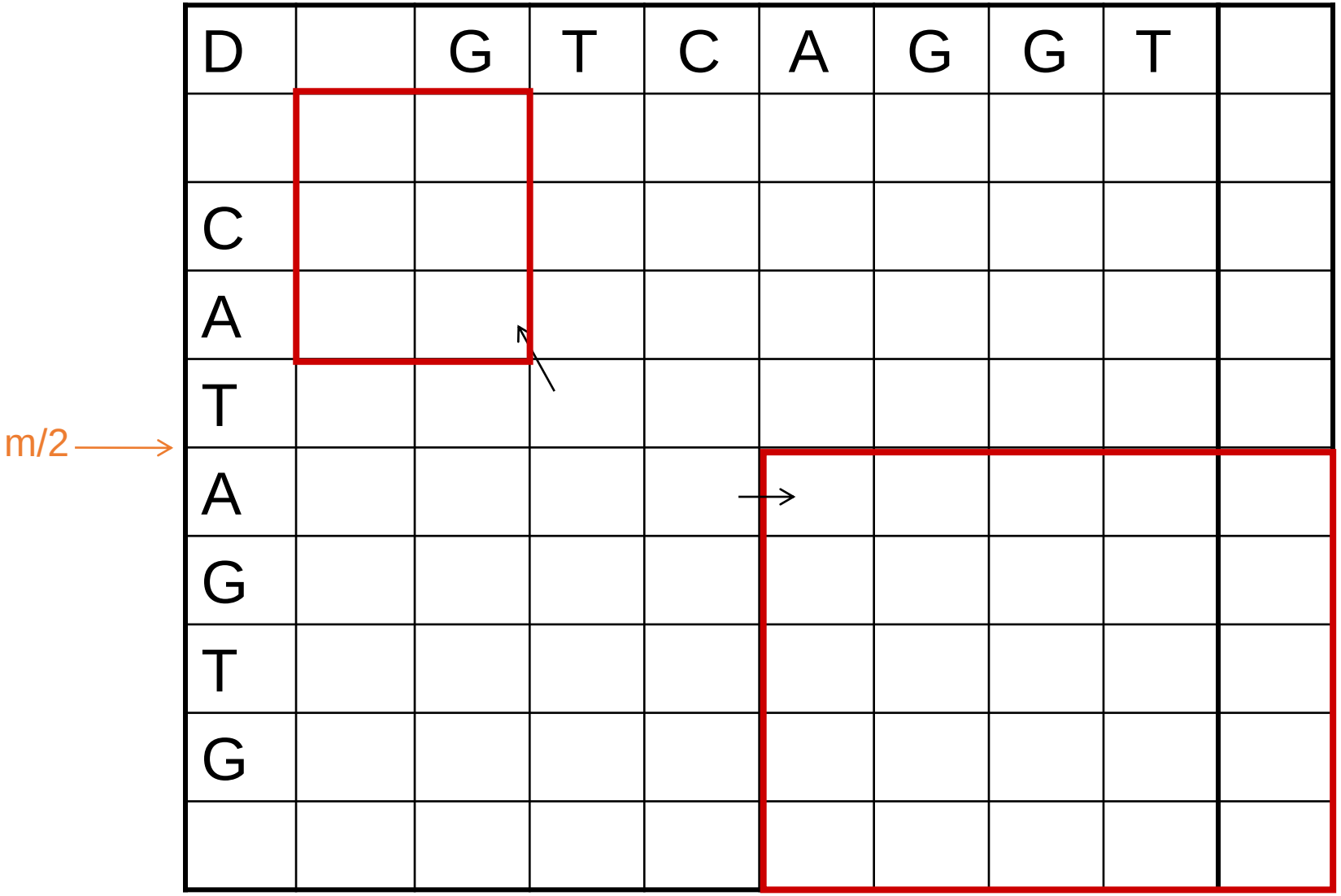

D G T C A G G T
C
A
T 3 3 2 3 3 3 4 4
m/2
A 5 4 3 2 3 3 3 4
G 4 5 4 3 2 2 2 3
T
G


D G T C A G G T
C
A
T
A
G
T
G
m/2
3 3 2 3 3 3 4 4
5 4 3 2 3 3 3 4


D G T C A G G T
C
A
T
A
G
T
G
m/2
3 3 2 3 3 3 4 4
5 4 3 2 3 3 3 4


D G T C A G G T
C
A
T
A
G
T
G
m/2
2
3


D G T C A G G T
C
A
T
A
G
T
G
2
3
m/2


D G T C A G G T
C
A
T
A
G
T
G
m/2

Algorithme de Hirschberg

Notation:

- Pour une séquence S, on note S^r l'inverse de S, i.e. si t est la taille de S, $S^r[i]=S[t-i+1]$

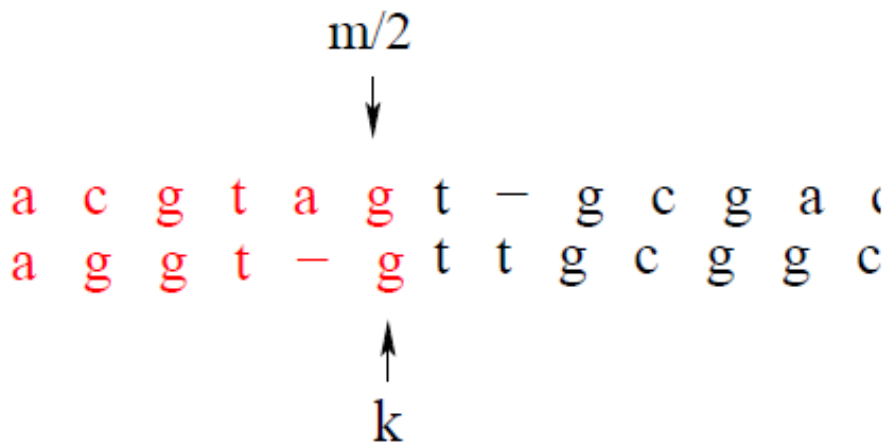
Exemple: Si S = GTCAGGT, S^r = TGGACTG

- *$V^r(i,j)$: Valeur de l'alignement optimal entre $S_1^r[1..i]=S_2^r[1..j]$,* autrement dit entre $S_1[m-i+1..m]=S_2^r[n-j+1..n]$

- *k^* maximise $V(m/2, k) + V^r(m/2,n-k)$, pour tout k, $1 \leq k \leq n$*

Lemme 1:

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



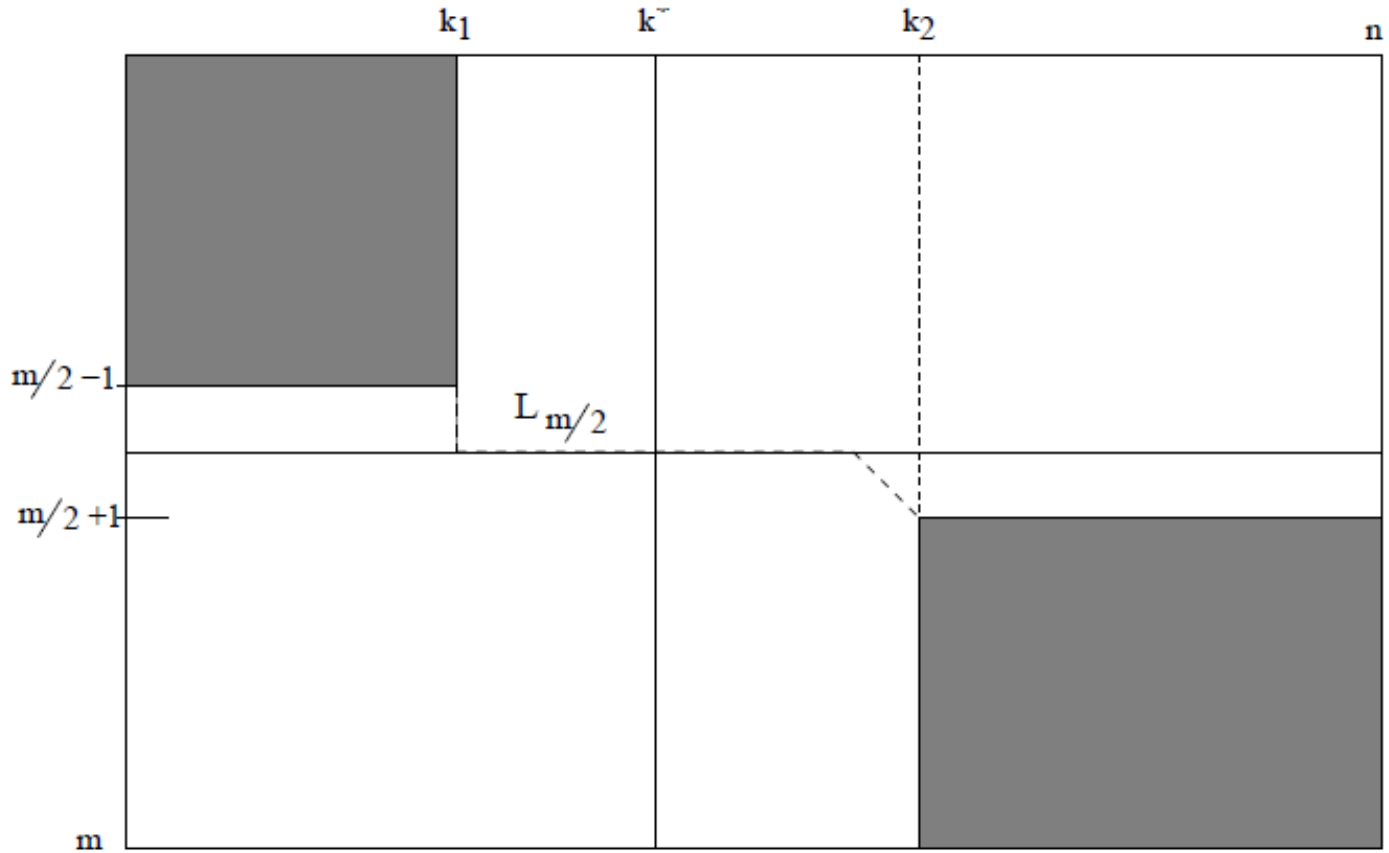
Preuve: Pour tout k, $V(m,n) =$

$$\text{Optimal}\begin{pmatrix} S[1..m/2] \\ T[1..k] \end{pmatrix} + \text{Optimal}\begin{pmatrix} S[m/2+1..m] \\ T[k+1..n] \end{pmatrix}$$

$$V(m/2,k) \qquad\qquad V^r(m/2, n-k)$$

Il existe un chemin optimal L allant de $(0,0)$ à (m,n) qui passe par $(m/2, k^*)$

$L_{m/2}$: sous-chemin de L qui commence à la dernière case de la ligne $m/2-1$ de L et fini à la $1^{ère}$ case de la ligne $m/2+1$ de L.



Lemme 2: k^* peut être trouvé en un temps $O(mn)$ et en espace $O(m)$. Même complexité pour produire le sous-chemin $L_{m/2}$.

Preuve:

- Remplir la table pour S et T, mais s'arrêter après la ligne $m/2$. Conserver les pointeurs à la ligne $m/2 \longrightarrow$ temps $O(mn)$, espace $O(m)$. $V(m/2, k)$ connu pour tout k.
- Même chose pour S^r et T^r.
- Trouver la valeur maximale $V(m/2, k) + V(m/2, n-k) \hookrightarrow k^*$ $\longrightarrow$ temps $O(n)$.
- Produire le chemin $L_{m/2}$ en suivant les pointeurs, d'abord de $(m/2, k^*)$ à la case $(m/2-1, k_1)$, ensuite de $(m/2, k^*)$ à la case $(m/2+1, k_2)$.

Deux sous-problèmes: Alignement de $S[1, m/2-1]$ et $T[1..k_1]$, et alignement de $S[m/2+1..m]$ et $T[k_2..n]$

Procedure $OPTA(l, l', r, r')$;

Début

$h = (l' - l)/2$;

Trouver k^*, k_1, k_2 et L_h;

$OPTA(l, h-1, r, k_1)$; (problème supérieur)

Produire L_h;

$OPTA(h+1, l', k_2, r')$; (problème inférieur)

Fin.

Appel initial: $OPTA(1, m, 1, n)$

À la sortie de la procédure, le chemin L produit est un alignement optimal de S et T.

Valeur $V(m, n)$ obtenue dès la premier appel de la procédure, i.e. pour $h = m/2$.

Complexité en temps: On note cpq, au lieu de $O(pq)$, le temps nécessaire pour remplir une table $p \times q$. c: constante.

- 1^{er} niveau de récursion: cmn
- $2^{ème}$ niveau de récursion, 2 sous-problèmes:

$$ck^*m/2 + c(n - k^*)m/2 = cnm/2$$

- $i^{ème}$ niveau de récursion, 2^{i-1} sous-problèmes.
  Chaque sous-problème a $m/2^{i-1}$ lignes et un nombre variables de colonnes. Mais chaque sous-problème a des colonnes distinctes des autres $\Rightarrow$ Au plus $cnm/2^{i-1}$

Théorème: Complexité en espace en $O(m)$ et complexité en temps

$$\sum_{i=1}^{\log n} cmn/2^{i-1} \leq 2cnm$$

Algorithme classique de l'alignement global en temps cmn.

Application à l'alignement local

L'idée de l'algorithme d'Hirschberg peut s'appliquer à un alignement local. J'explique ici l'idée dans le cas d'un seul chemin optimal dans la table. L'idée peut être étendue au cas général.

(voir X. Huang, R.C. Hardison and W. Miller. A space-efficient algorithm for local similarities. *Computer applications in biosciences*, 6(4):373-31, 1990).

- Exécuter l'alignement local en espace linéaire (conserver une ligne à chaque fois) pour les séquences S_1 et S_2 et conserver la case de valeur max. Cette case correspond au coin sud-est de la sous-table.
- Exécuter l'alignement local en espace linéaire pour les séquences inverses S_1^r et S_2^r à partir du coin sud-est identifié à l'étape précédente et conserver la case de valeur max. Cette case correspond au coin nord-ouest de la sous-table.
- Exécuter l'algorithme de Hirschberg dans la sous-table déterminée par ces deux coins.

II - Alignement global par bandes

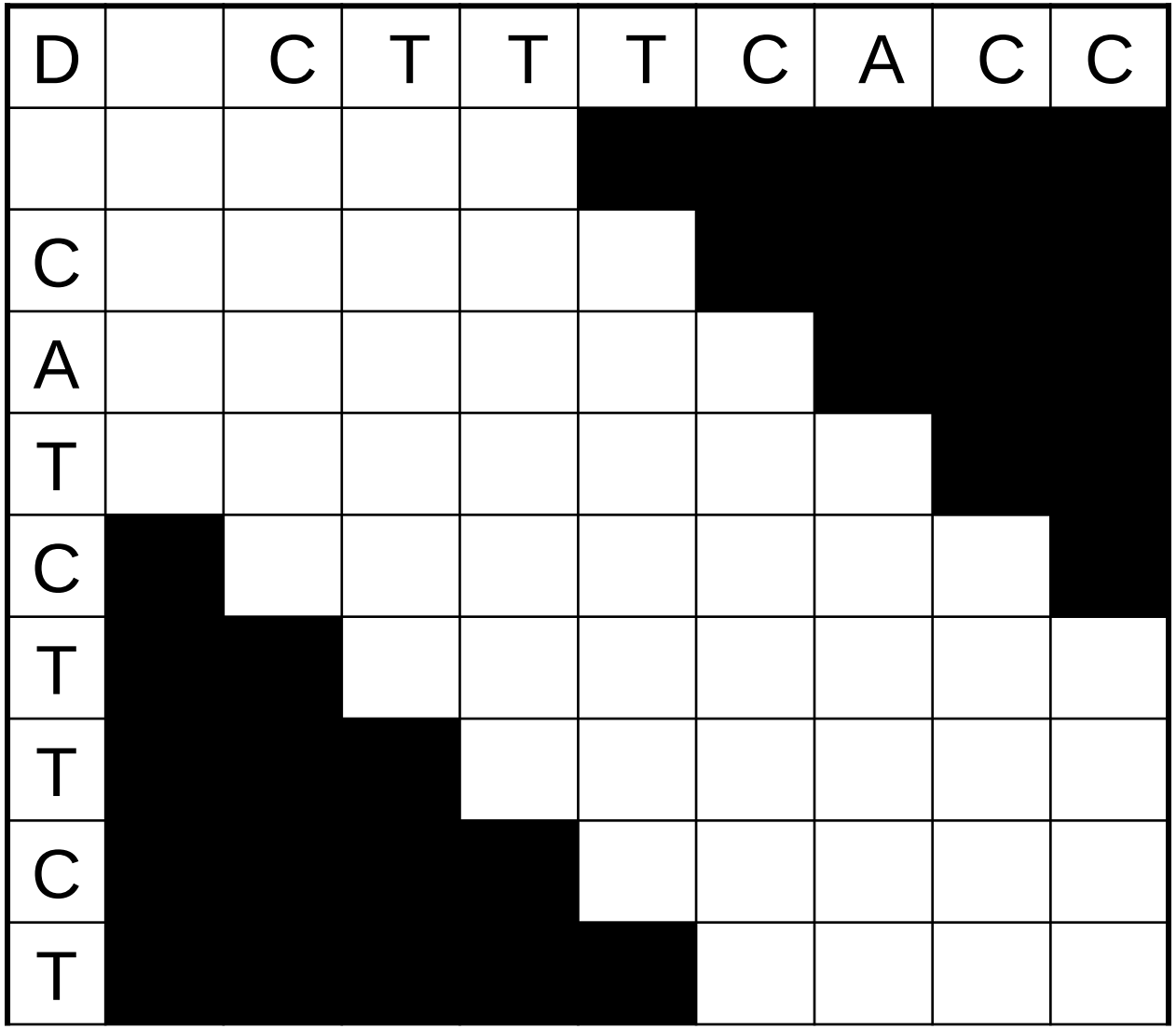
- Supposons qu'on aligne deux gènes orthologues, autrement dit deux gènes de deux espèces différentes provenant du même ancêtre commun. On s'attend à peu de divergence entre le deux gènes.

  ➔ L'alignement va rester proche de la diagonale principale.

- On s'inspire de cette idée pour limiter l'exploration autour de la diagonale principale: gain de temps et d'espace. Soient S_1 et S_2 deux séquences de taille *n et m avec n<m* et *k* une constante << *n*
  - Si S_1 et S_2 sont suffisamment similaires (nb d'erreurs au plus *k*), algo en *O(kn)* en temps et en espace.
  - *O(nm)* dans le pire des cas, mais avec une plus grande constante de multiplicité.

D		C	T	T	T	C	A	C	C
C									
A									
T									
C									
T									
T									
C									
T									

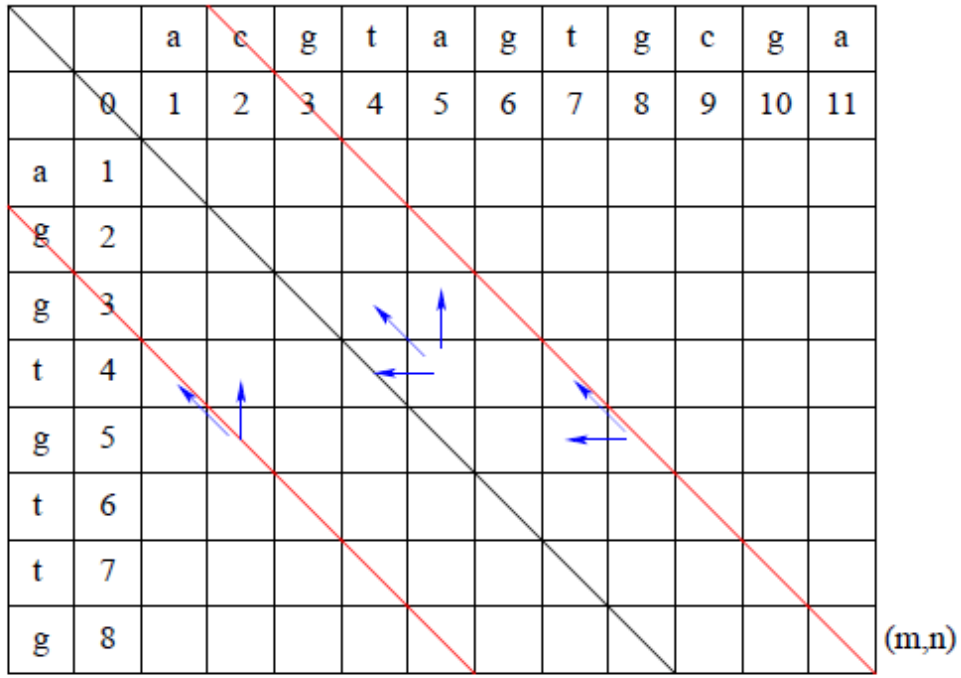
On veut savoir si on peut aligner à 3 erreurs près



Observation: Tout chemin définissant un alignement contenant moins de k erreurs ne passe par aucune case $(i, i+l)$ ou $(i, i-l)$, avec $l > k$

$\Rightarrow$ compléter la table dans une bande centrée sur la diagonale principale, déterminée par les deux diagonales $+k$ et $-k$. Bande qui contient $2k+1$ cases à chaque ligne

k=3



Si $m = n$, la taille de la bande peut être réduite de moitié

Si (m, n) contient une valeur $v < k$, alors un alignement optimal avec v erreurs existe. Alignement trouvé en suivant les pointeurs de (m, n) à $(0, 0)$.

Réciproquement, si un alignement optimal avec moins de k erreurs existe, alors (m, n) contient une valeur $< k$.

Si (m, n) contient une valeur plus grande que k, alors cette valeur est supérieure à la valeur d'un alignement optimal.

Complexité: en temps et en espace $O(km)$.

Nombre d'erreurs non spécifié

S et T deux séquences, et k^* valeur d'un alignement optimal (k^* non connu, mais petit).

"Deviner" la valeur de k^* en appliquant l'algorithme pour nombre d'erreurs borné avec différentes valeurs de k

Commencer avec $k = 1$, et doubler k jusqu'à obtenir un alignement global de valeur $\leq k$.

Complexité: Soit k' la plus grande valeur de k utilisée. $k' \leq 2k^*$.

Complexité totale en temps et en espace:

$$O(k'm + k'm/2 + k'm/4 + \cdots + m) = O(k'm) = O(k^*m)$$

III- Algorithme des quatre russes

Amélioration asymptotique en temps des algorithmes d'alignement global

➢ Méthode des quatre russes pour l'alignement global

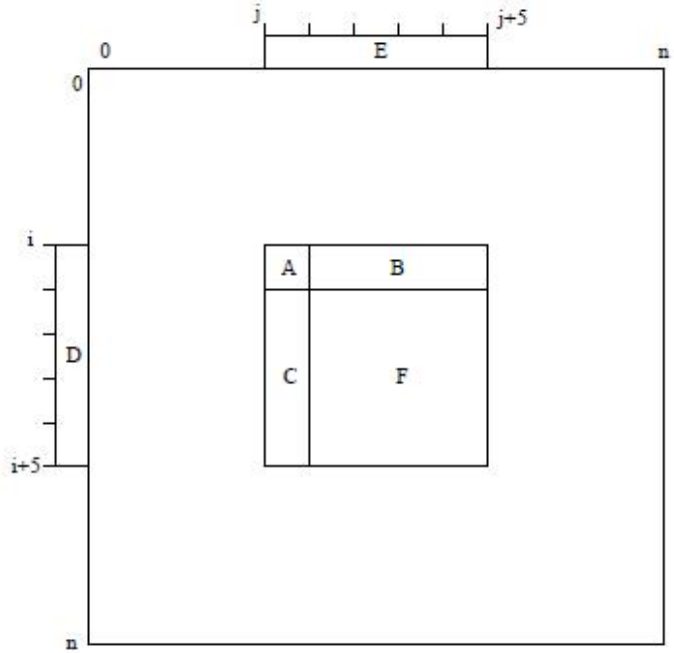
W.J. Masek et M.S. Paterson, A faster algorithm for computing string edit distances. *Journal of Computer and System Sciences*, 20 (1): 18-31, 1980.

Méthode d'accélération d'algorithmes basés sur la programmation dynamique.
Comparaison de deux séquences de taille n en temps

$$O(n^2 / \log n)$$

t-bloc: sous-tableau de taille t x t.

Idée générale: Partitionner la table de prog. dyn. en t-blocs « pas trop » chevauchants.
Calcul élémentaire effectué pour chaque t-bloc, en temps O(t).



Observation: Valeurs d'un t-bloc débutant à la case (i, j) fonction de la première ligne et de la première colonne du t-bloc, et des deux sous-mots $S_1[i..i+t-1]$ et $S_2[j..j+t-1]$.

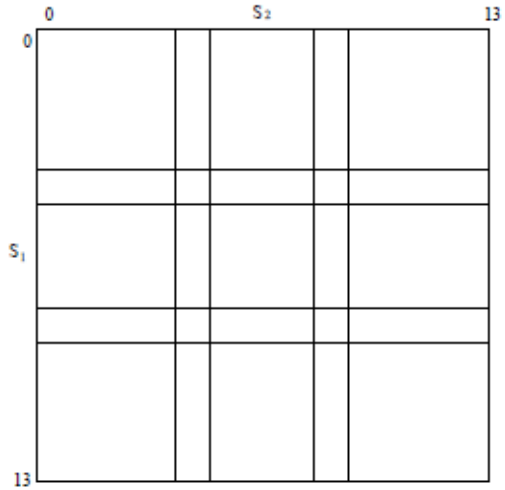
Fonction "bloc": Calcul des valeurs d'un $t - bloc$. Paramètres (A, B, C, D, E).

Fonction "bloc restreint": Calcul de la dernière ligne et de la dernière colonne d'un t-bloc.

Algorithme général

Pour simplifier, S_1 et S_2 sont de taille $n = k(t-1)$.

1. Subdiviser le tableau $(n+1) \times (n+1)$ en t-blocs:



2. Initialiser la première ligne et la première colonne du tableau
3. Examiner le tableau ligne par ligne. Fonction "bloc restreint" détermine les valeurs de la dernière ligne colonne de chaque t-bloc.
4. Valeur d'édition de l'alignement global optimal en (n, n).

Cœur du problème: Étape (3).

Possibilité: Précalculer tous les $t - blocs$ possibles. Énumérer tous les inputs possibles, calculer les outputs correspondant (ligne et colonne de taille t). Conserver les output indexés par les inputs.

Complexité:

- Inputs et outputs de taille $O(t)$. Chaque input peut être retrouvé en un temps $O(t)$. Il y a (n^2/t^2) t-blocs $\Rightarrow$ algorithme en $O(n^2/t)$. Si $t \sim \log n$, algorithme en $O(n^2/\log n)$.
- Phase de prétraitement: Combien de choix d'inputs? Chaque case peut contenir les valeurs de 0 à $n \Rightarrow (n+1)^t$ valeurs possibles pour chaque ligne ou colonne de taille t. D'autre part, $|\Sigma|^t$ mots possibles de taille t

  $\Longrightarrow (n+1)^{2t}|\Sigma|^{2t}$ inputs différents

  $\Longrightarrow$ Temps de prétraitement en $O((n+1)^{2t}|\Sigma|^{2t}t^2)$, et comme $t \geq 1$, temps au moins en $O(n^2)$ $\Rightarrow$ Pas de progrès !!!

Optimisation

Terme prédominant: $(n+1)^{2t}$, nombre de choix possibles pour chaque ligne et colonne. Mais ces choix pas tous possibles:

Lemme: Deux cases adjacentes d'une ligne, colonne ou diagonale du tableau de la prog. dyn. peuvent avoir des valeurs qui diffèrent d'au plus 1

Chaque ligne (ou colonne) d'un t-bloc peut être représentée par la valeur la plus à gauche de la ligne, suivie d'un vecteur de compensation, de valeurs $\{-1, 0, 1\}$.

Exemple: ligne 5, 4, 4, 5 $\longrightarrow$ 5, -1, 0, +1.

Comment se débarasser du premier terme?

Vecteur offset: Vecteur de taille t, de valeurs $\{-1, 0, 1\}$, où la première valeur doit être 0.

Théorème: t-bloc commençant à la position (i, j). Vecteurs offset de la dernière ligne et colonne du t-bloc, peuvent être déterminés à partir des vecteurs offset correspondant à la première ligne et colonne du t-bloc, et des mots $S_1[i..i + t - 1]$ et $S_2[j..j + t - 1]$.

Algorithme:

1. Subdiviser le tableau en t-blocs.
2. Initialiser la première ligne et colonne. Calculer les vecteurs offset de la première ligne et colonne.
3. Examiner le tableau ligne par ligne, et déterminer les vecteurs offset à la dernière ligne et à la dernière colonne de chaque bloc.
4. Si Q est la valeur totale des offsets calculés à la dernière ligne, alors $D(n, n) = D(n, 0) + Q = n + Q$.

		C	A	G	T	C	A
	0 0	1 +1	2 +1	3 +1	4 +1	5 +1	6 +1
A	1 +1			-1			-1
T	2 +1			0			-1
G	3 +1	0	0	-1 0	+1	0	+1 0
T	4 +1			+1			0
C	5 +1			+1			-1
A	6 +1	-1	-1	+1 +1	-1	-1	-1 -1

Vecteurs offset 1ere ligne et colonne

Subdiviser en t-blocs

		C	A	G	
	0	1	2	3	0
A	1	1	1	2	-1
T	2	2	2	2	0
G	3	3	3	2	0
	0	0	0	-1	

		T	C	A	
	0	1	2	3	0
A	-1	0	1	2	-1
T	-1	-1	0	1	-1
G	-1	0	0	1	0
	0	+1	0	+1	

D(6,6)= D(6,0) -1 -1 +1 -1 -1 -1 =2

Complexité: Il existe maintenant $3^{2(t-1)}|\Sigma|^{2t}$ inputs possibles. Pour chaque input, le output correspondant est calculé en temps $O(t^2)$.

$\Longrightarrow$ si $t = (\log_{3|\Sigma|} n)/2$, prétraitement effectué en un temps

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

Théorème: La distance d'édition entre deux séquences de taille n peut être calculée en un temps $O(n^2/\log n)$

Dans la pratique, valeur de t choisie telle que calcul des vecteur offsets et recherche d'un input en temps constant. Par exemple, choisir la taille d'un mot machine pour t. Également, utiliser une table de hashage pour ranger les inputs.

III- Algorithme de Crochemore et Landau

Amélioration asymptotique en temps des algorithmes d'alignement global et local

- Méthode des quatre russes pour l'alignement global
  - Limité à l'alignement global
  - Système de pondération limité à des nombres rationnels
- Méthode de Crochemore et Landau

M. Crochemore, G.M. Landau and Z. Ziv-Ukelson. A sub-quadratic sequence alignment algorithm for unrestricted cost matrices. Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms, pp 679-688, 2002.

  - Remédie à ces limitations
  - $O(n^2 / \log n)$
  - Peut traiter des textes compressés. Même plus rapide dans ce cas:

$O(h.n^2 / \log n)$ pour $0 < h \leq 1$, où h représente l'entropie des mots, i.e. leur degré « compressibilité » ou de répétitivité, ratio entre la taille de la séquence compressée et la taille de la séquence non-compressée. Plus le mot est répétitif, plus h est petit.

Le reste de ce document est donné en supplément. Il permet aux étudiants gradués du cours d'avoir un exemple de présentation d'article à laquelle je m'attends.

L'algorithme de Crochemore et Landau n'est pas requis pour les évaluations du cours.

Notions préliminaires

- Contrairement à la méthode des quatre russes, les bocs ne sont pas nécessairement carrés, et pas de taille égale.
- Utilise la factorisation LZ78 (Lempel-Ziv)

Le texte est divisé en mots où chaque mot est le plus long mot déjà vu + le prochain caractère dans le texte.

Exemple : S = aacgacg

- 1er mot : a
- 2e mot : ac
- 3e mot : g
- 4e mot : acg

-Chaque mot est encodé comme un indice allant vers son préfixe (un mot précédent) plus le caractère extra.

-Chaque nouveau mot est ajouté à la liste des mots qui peuvent se faire référencer.

-On peut utiliser un arbre préfixe.

Notions préliminaires

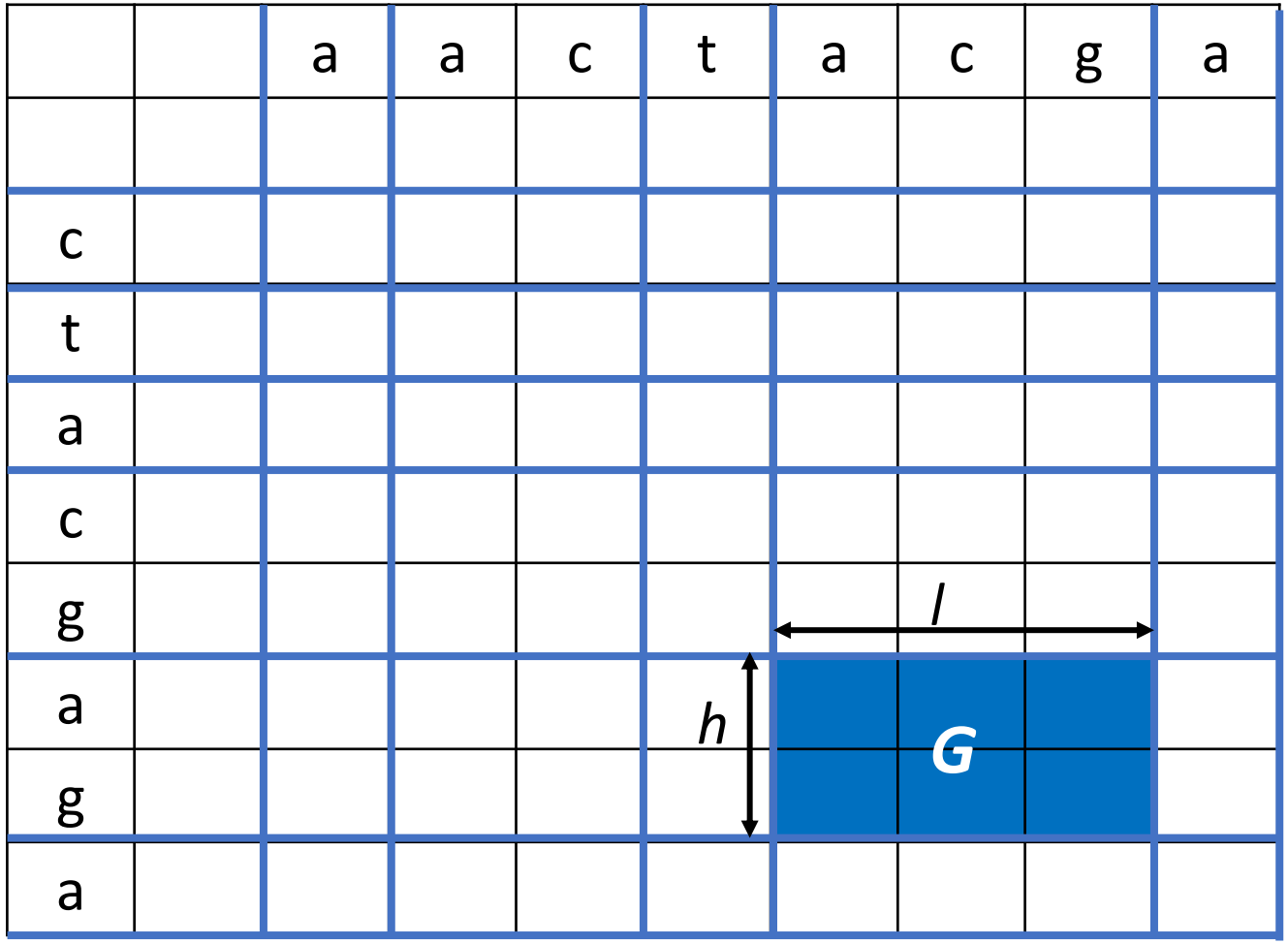
- Comme chaque mot est distinct, on peut montrer que le nombre maximal de mots distincts dans un texte S de taille n sur un alphabet fini est $O(n/\log(n))$.
- En général, le nombre de mots distincts d'un texte est dans $O(hn/\log(n))$ où $0 \leq h \leq 1$ est l'entropie du texte.

Subdivision de la table en blocs

		a	a	c	t	a	c	g	a
c									
t									
a									
c									
g									
a									
g									
a									

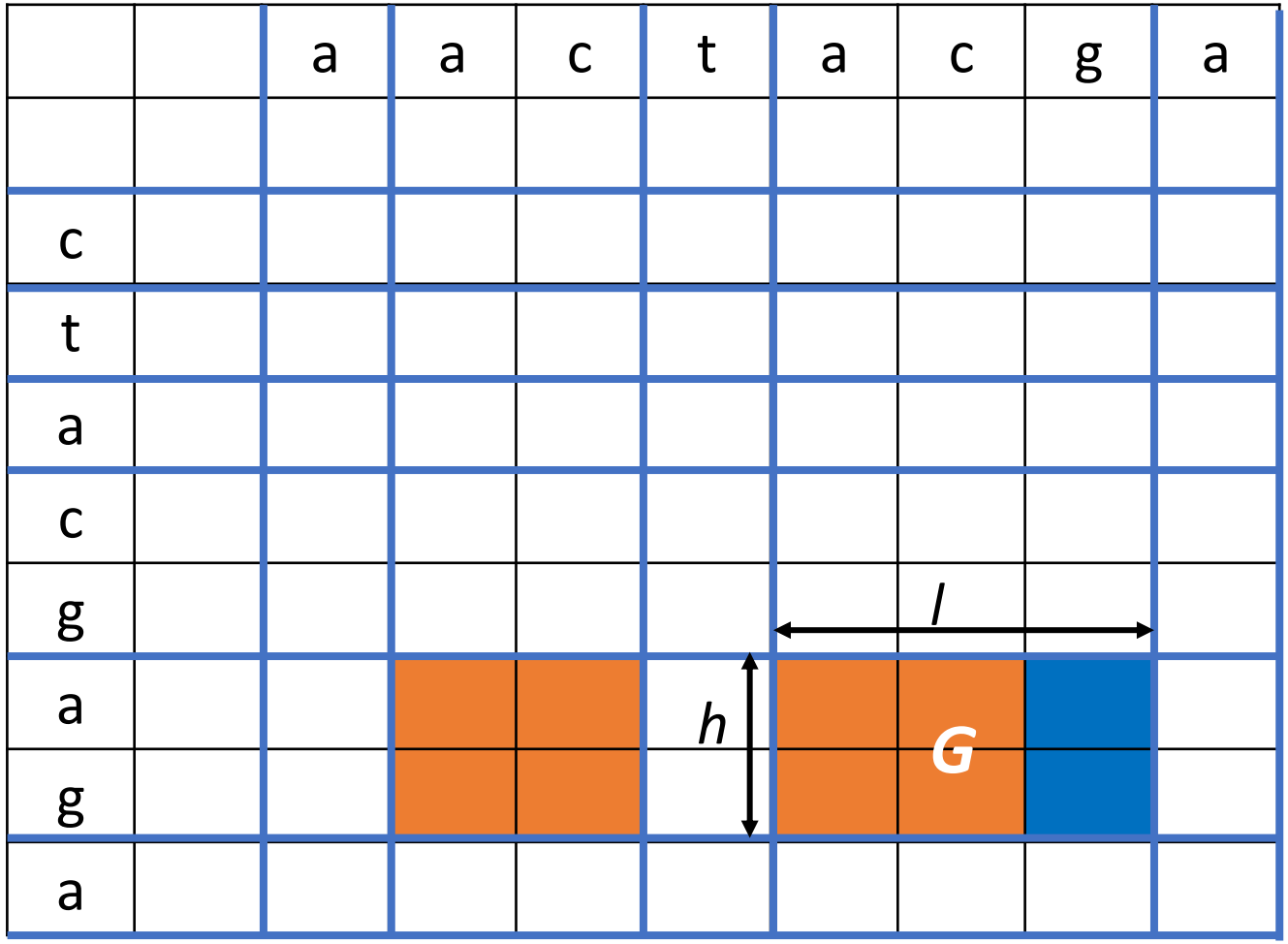
Subdivision de la table en blocs

- Pour chaque bloc *G* de hauteur *h* et de largeur *l*



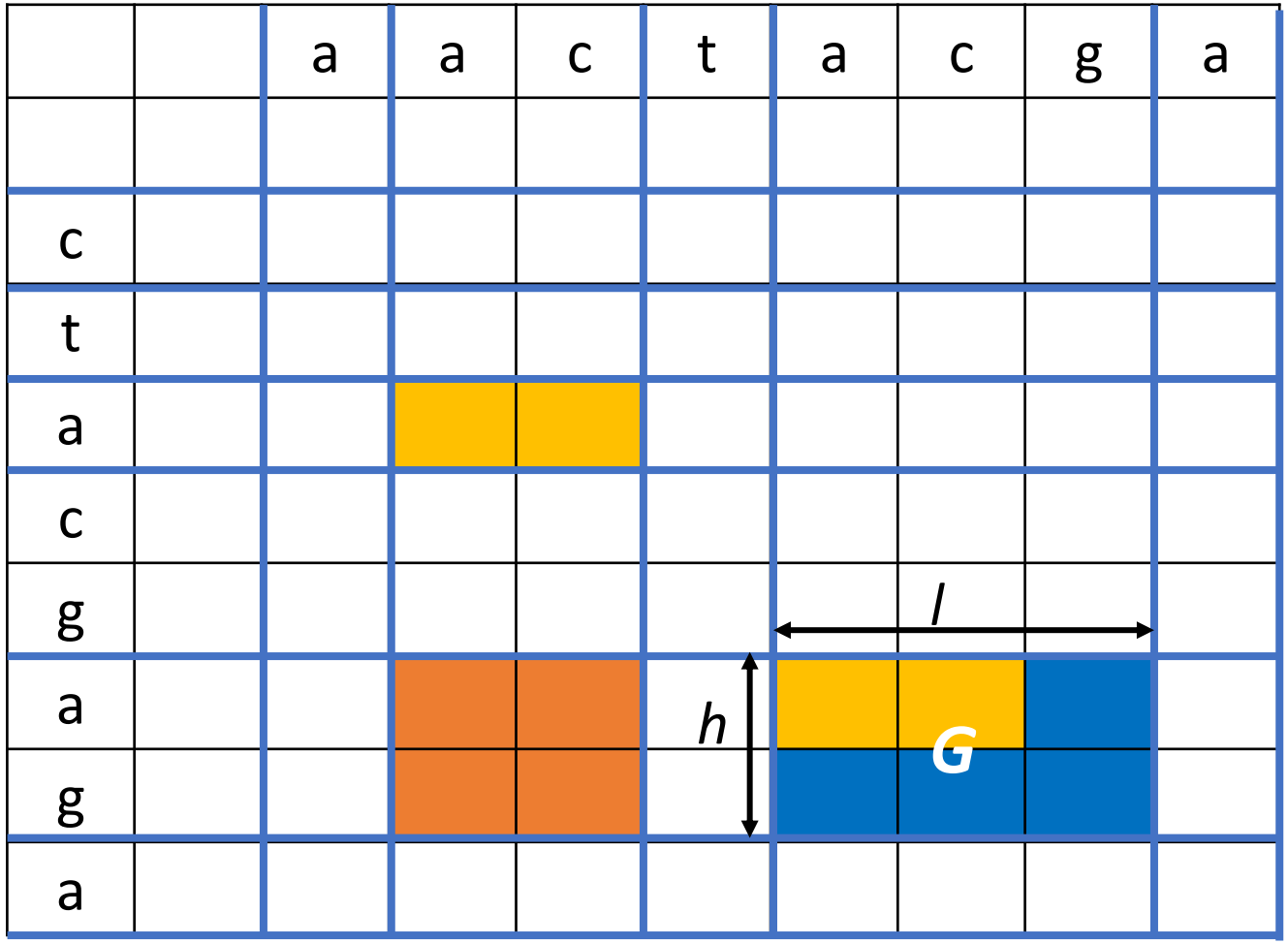
Subdivision de la table en blocs

- Pour chaque bloc *G* de hauteur *h* et de largeur *l*
  - Préfixe gauche



Subdivision de la table en blocs

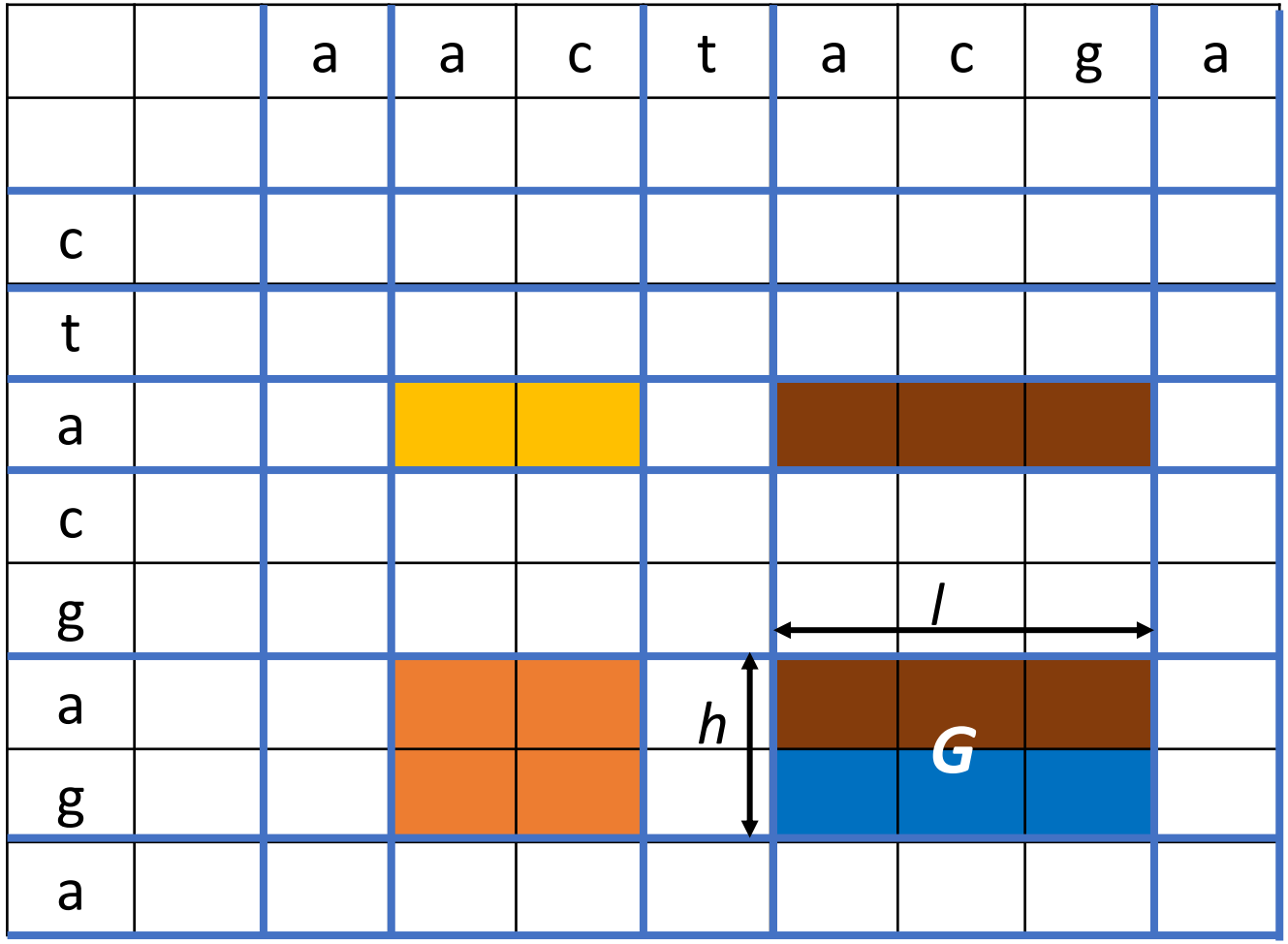
- Pour chaque bloc *G* de hauteur *h* et de largeur *l*
  - Préfixe gauche
  - Préfixe diagonal



Subdivision de la table en blocs

- Pour chaque bloc *G* de hauteur *h* et de largeur *l*
  - Préfixe gauche
  - Préfixe diagonal
  - Préfixe haut

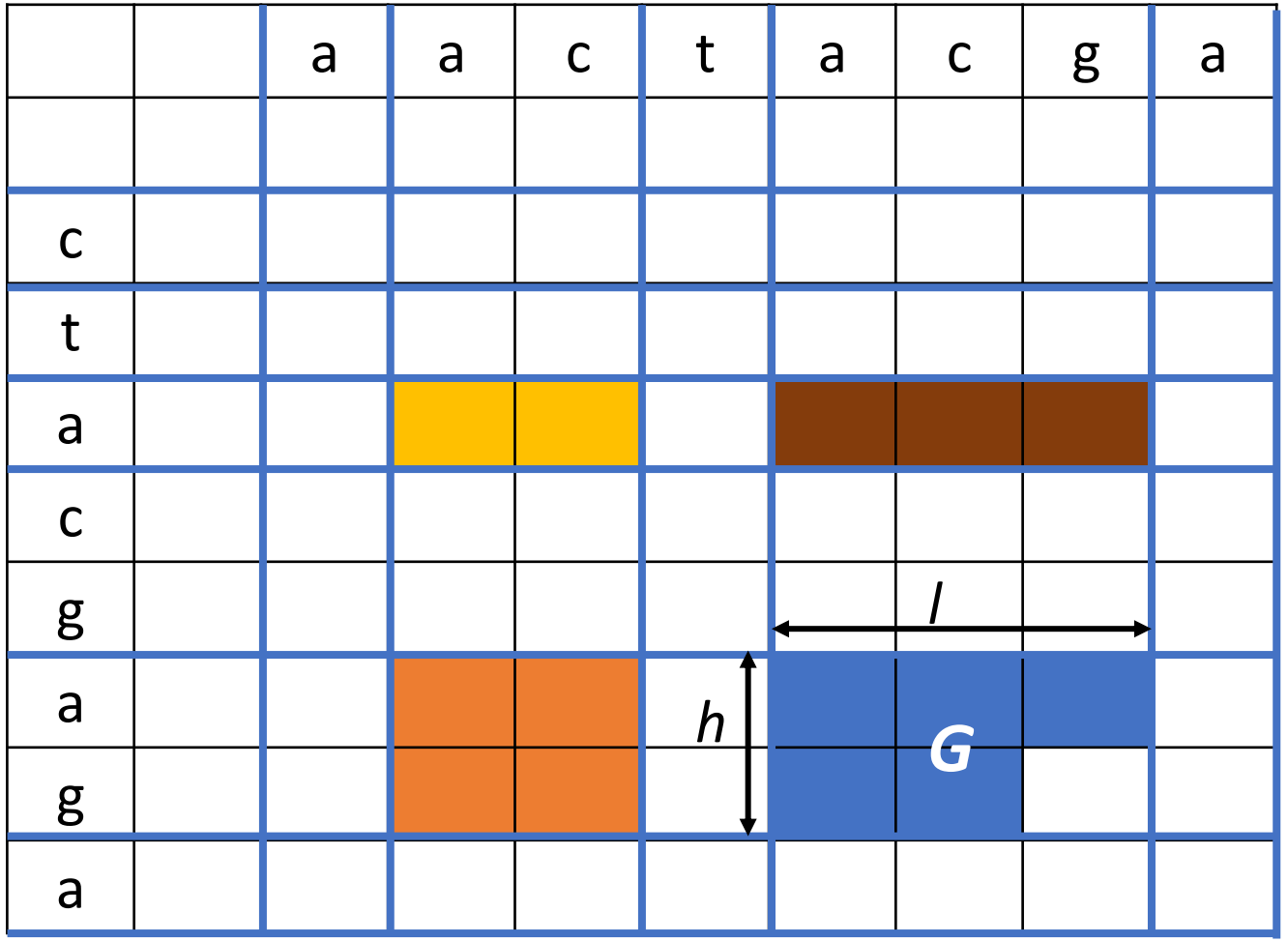
Tous les trois calculés avant G.



Subdivision de la table en blocs

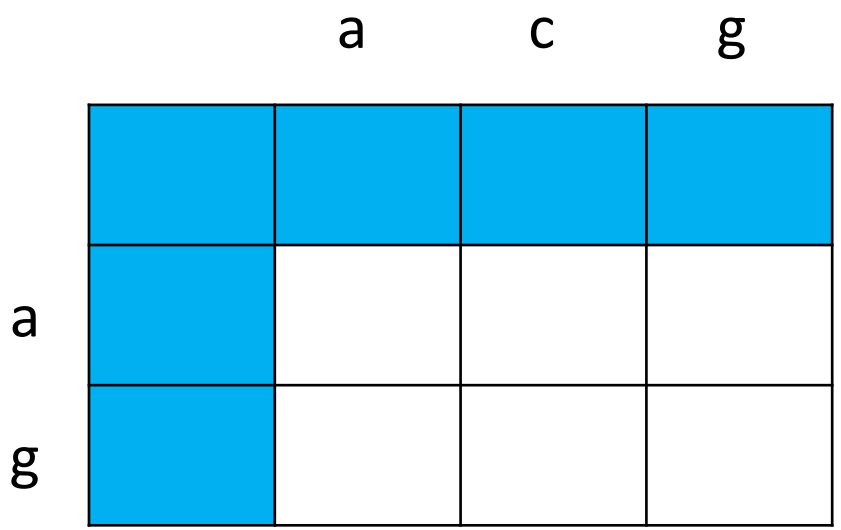
- Pour chaque bloc *G* de hauteur *h* et de largeur *l*
  - Préfixe gauche
  - Préfixe diagonal
  - Préfixe haut

Tous les trois calculés avant G (la seule case pas calculée est (g,g)). Mais résultat de G dépend des valeurs en entrées de G:



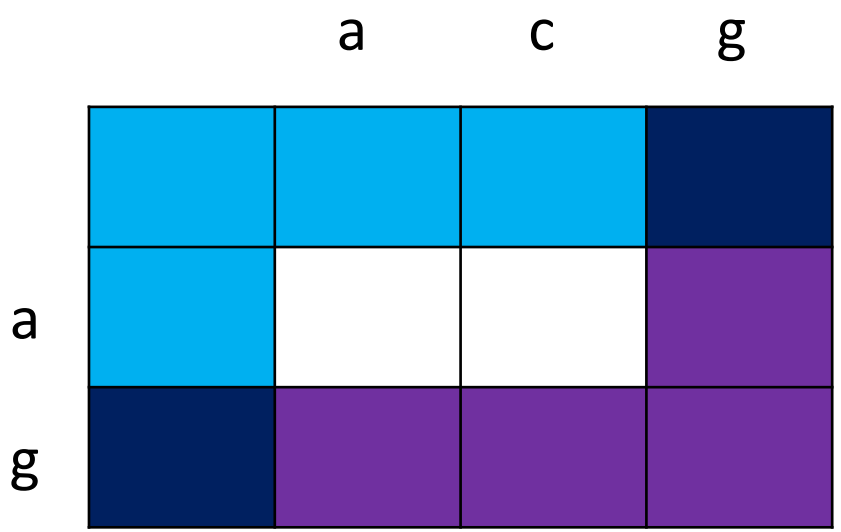
Subdivision de la table en blocs

- Pour chaque bloc *G* de hauteur *h* et de largeur *l*
  - *p = h+l-1* le périmètre de *G.*
  - Entrée *l* de taille *p*



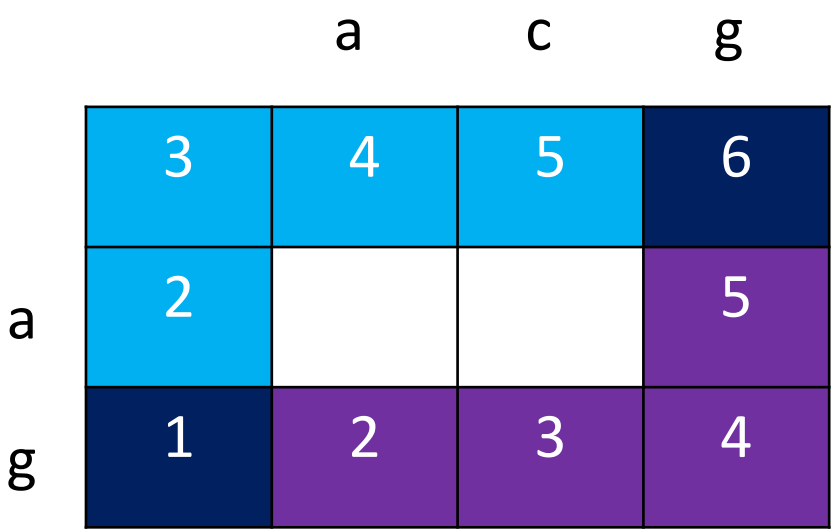
Subdivision de la table en blocs

- Pour chaque bloc *G* de hauteur *h* et de largeur *l*
  - *p = h+l-1* le périmètre de *G.*
  - Entrée *I* de taille *p*
  - Sortie *O* de taille *p*



Subdivision de la table en blocs

- Pour chaque bloc *G* de hauteur *h* et de largeur *l*
  - *p = h+l-1* le périmètre de *G.*
  - Entrée *I* de taille *p*
  - Sortie *O* de taille *p*



Subdivision de la table en blocs

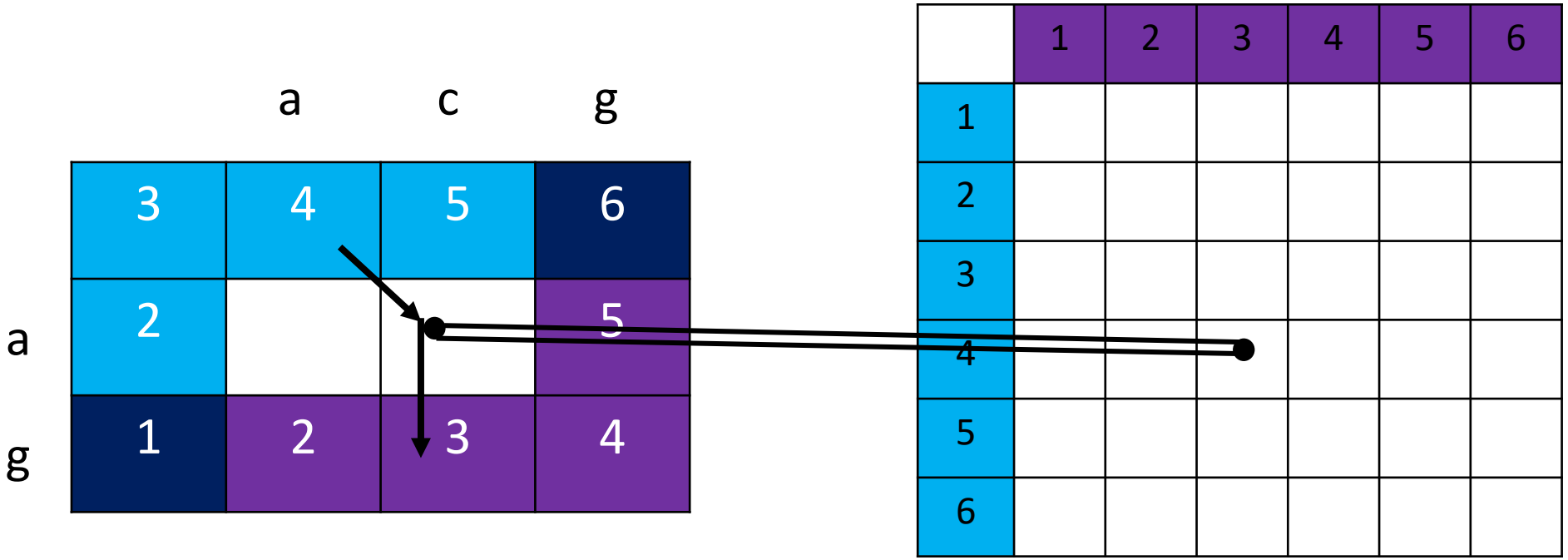
- Créer une matrice *DIST* p x p : lignes correspondant aux cellules Entrée (I) et colonnes aux cellules Sortie (O) tel que *DIST[i,j]* contient la valeur d'un alignement optimal entre les cellules correspondantes.

		a	c	g
	3	4	5	6
a	2			5
g	1	2	3	4

	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						

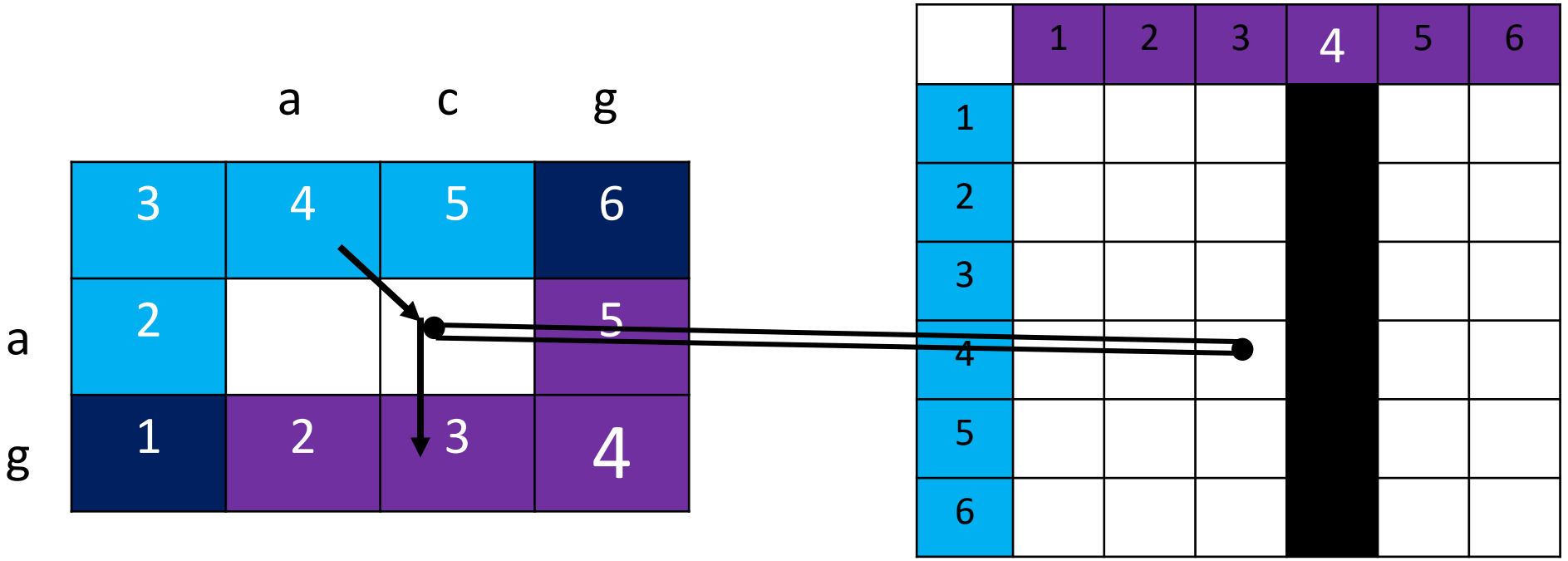
Subdivision de la table en blocs

- Créer une matrice *DIST* p x p : lignes correspondant aux cellules Entrée (I) et colonnes aux cellules Sortie (O) tel que *DIST[i,j]* contient la valeur d'un alignement optimal entre les cellules correspondantes.



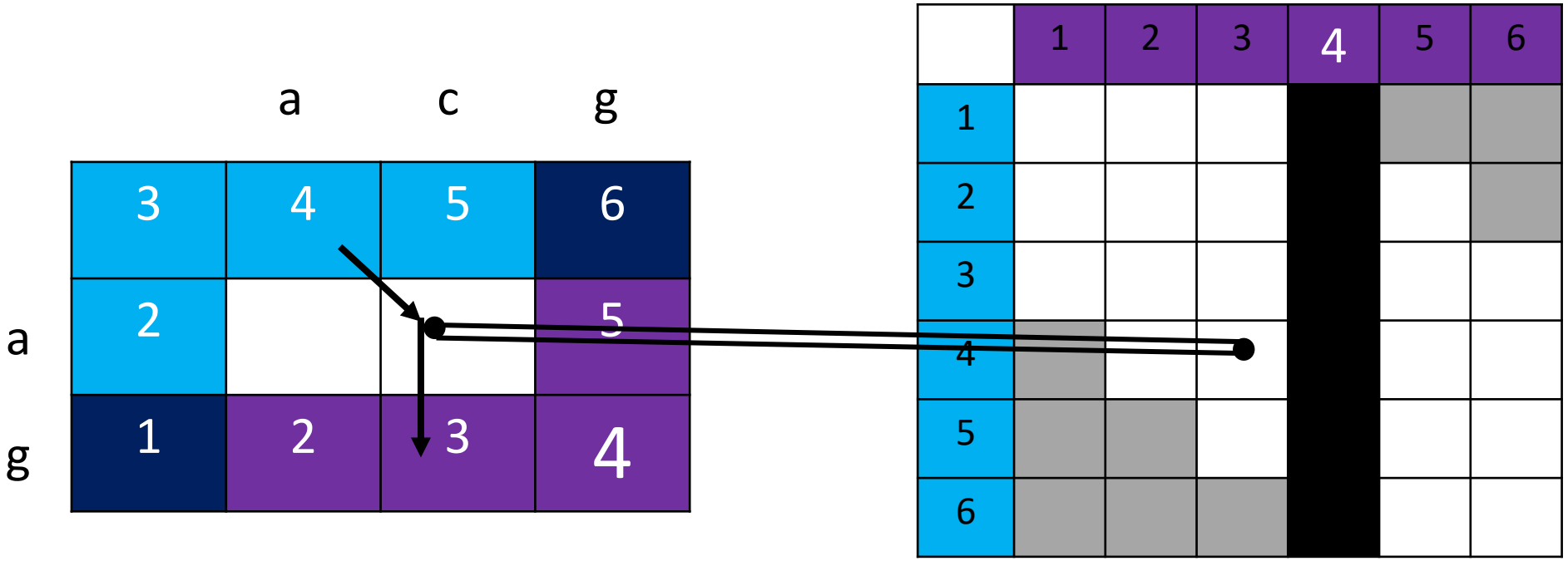
Subdivision de la table en blocs

- La seule colonne de la table à compléter est celle pour la case 4.



Subdivision de la table en blocs

- La seule colonne de la table à compléter est celle pour la case 4.
- Certaines cases n'ont pas de valeurs.



Subdivision de la table en blocs

Remplissage de la table pour une valeur de similarité :

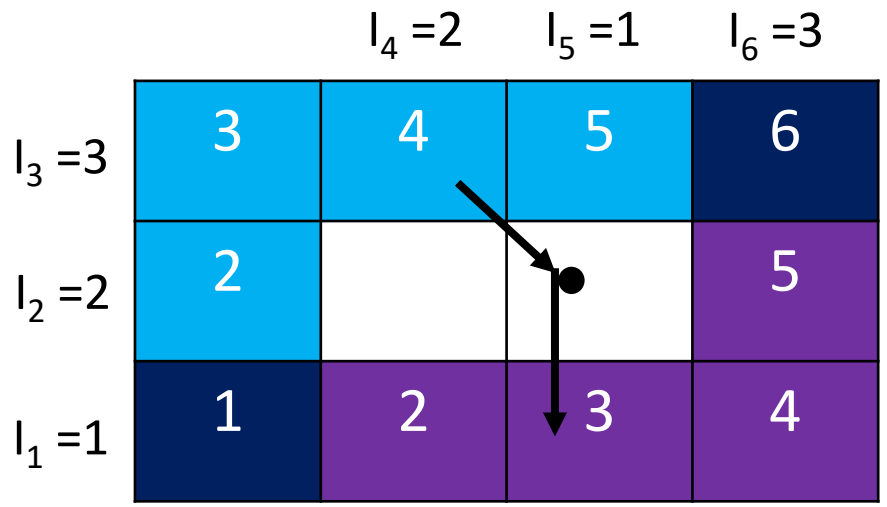
- δ (x,-) = δ (-,y) = -1
- δ (x,y) = -1 si x ≠ y
- δ (x,y) = 1 si x = y

	a	c	g
a	2		5
g	1	2	3	4

	1	2	3	4	5	6
1	0	-1	-2	-3		
2	-1	-1	-2	-1	-3	
3	-2	0	0	1	-1	-3
4		-2	-2	0	-2	-2
5			-2	0	-1	-1
6				-2	-1	0

Subdivision de la table en blocs

- Étant donnée une entrée $I_1, I_2, \ldots I_p$,
  - $OUT[i,j] = I_i + DIST[i,j]$



OUT (4,3)
= I_4 + DIST (4,3) = 2-2=0

		1	2	3	4	5	6
	1	0	-1	-2	-3		
	2	-1	-1	-2	-1	-3	
	3	-2	0	0	1	-1	-3
	4		-2	-2	0	-2	-2
	5			-2	0	-1	-1
	6				-2	-1	0

I_1 =1
I_2 =2
I_3 =3
I_4 =2
I_5 =1
I_6 =3

Subdivision de la table en blocs

- Étant donnée une entrée $I_1, I_2, \ldots I_p$,
  - $OUT[i,j] = I_i + DIST[i,j]$

DIST

		1	2	3	4	5	6
I_1 =1	1	0	-1	-2	-3		
I_2 =2	2	-1	-1	-2	-1	-3	
I_3 =3	3	-2	0	0	1	-1	-3
I_4 =2	4		-2	-2	0	-2	-2
I_5 =1	5			-2	0	-1	-1
I_6 =3	6				-2	-1	0

OUT

	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						

Subdivision de la table en blocs

- Étant donnée une entrée $I_1, I_2, \ldots I_p$,
  - $OUT[i,j] = I_i + DIST[i,j]$

DIST

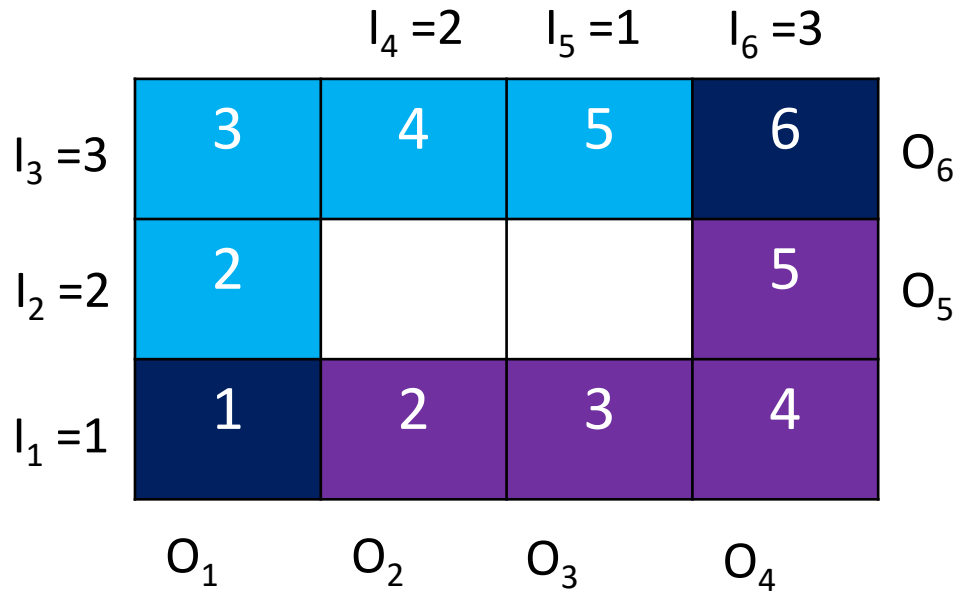
		1	2	3	4	5	6
I_1 =1	1	0	-1	-2	-3		
I_2 =2	2	-1	-1	-2	-1	-3	
I_3 =3	3	-2	0	0	1	-1	-3
I_4 =2	4		-2	-2	0	-2	-2
I_5 =1	5			-2	0	-1	-1
I_6 =3	6				-2	-1	0

OUT

	1	2	3	4	5	6
1	1	0	-1	-2		
2	1	1	0	1	-1	
3	1	3	3	4	2	0
4		0	0	2	0	0
5			-1	1	0	0
6				1	2	3

Subdivision de la table en blocs

- Étant donnée une entrée I_1, I_2, … I_p,
  - $OUT[i,j] = I_i + DIST[i,j]$
  - $O_j = \max_{\{0 \le r \le p\}} \{I_r + DIST[r,j]\}$



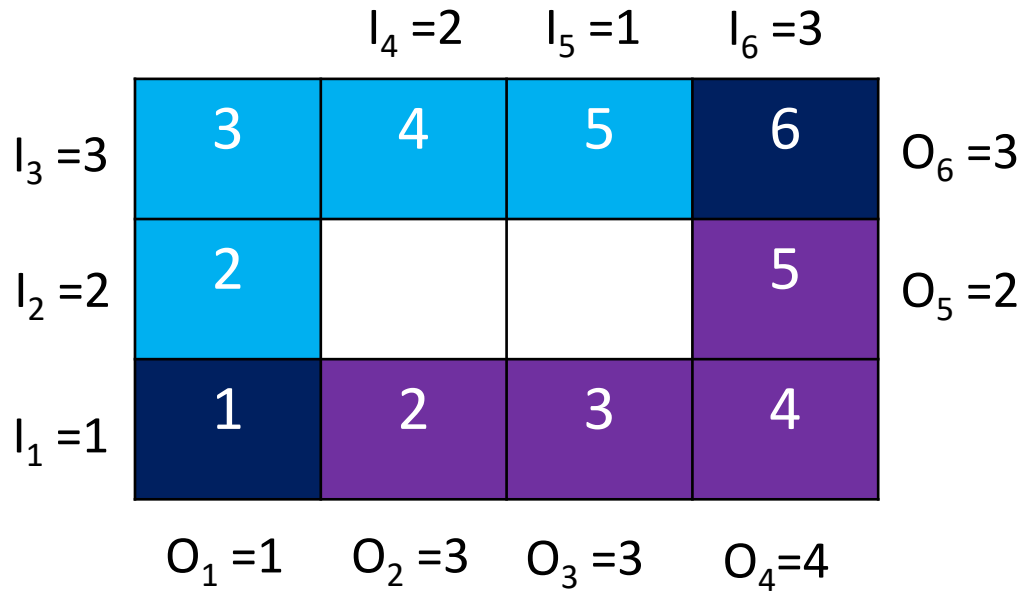
OUT

	1	2	3	4	5	6
1	1	0	-1	-2		
2	1	1	0	1	-1	
3	1	3	3	4	2	0
4		0	0	2	0	0
5			-1	1	0	0
6				1	2	3
	O_1	**O_2**	**O_3**	**O_4**	**O_5**	**O_6**

Subdivision de la table en blocs

- Étant donnée une entrée $I_1, I_2, \ldots I_p$,
  - $OUT[i,j] = I_i + DIST[i,j]$
  - $O_j = \max_{\{0 \le r \le p\}} \{I_r + DIST[r,j]\}$

Comment calculer path et OUT de façon efficace?

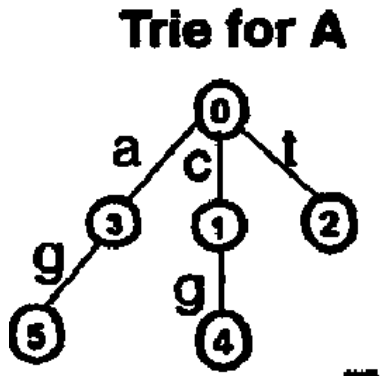


OUT

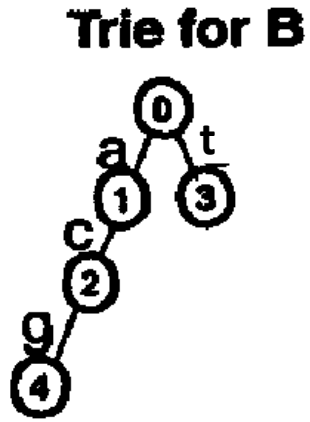
	1	2	3	4	5	6
1	**1**	0	-1	-2		
2	1	1	0	1	-1	
3	1	**3**	**3**	**4**	**2**	0
4		0	0	2	0	0
5			-1	1	0	0
6				1	2	**3**
	O_1	**O_2**	**O_3**	**O_4**	**O_5**	**O_6**

Structures de données

- Représenter les deux séquences par des indexes:
  - A = c t a c g a g a → c | t | a | cg | ag | a |



- Représenter les deux séquences par des indexes:
  - A = a a c t a c g a → a | ac | t | acg | a |



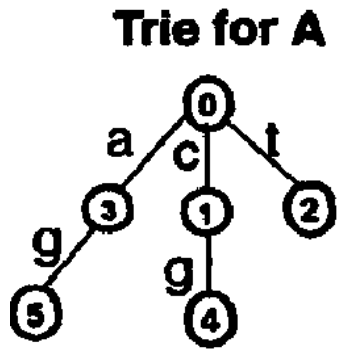
Structures de données

- Une « table de blocs » contenant une entrée par bloc.

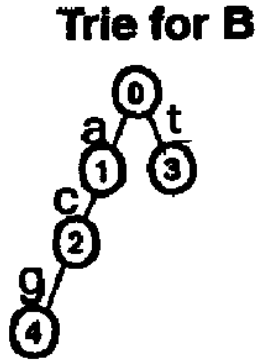
B

		a	a	c	t	a	c	g	a
c									
t									
a									
c									
g									
a									
g									
a									

A



c | t | a | cg | ag | a |

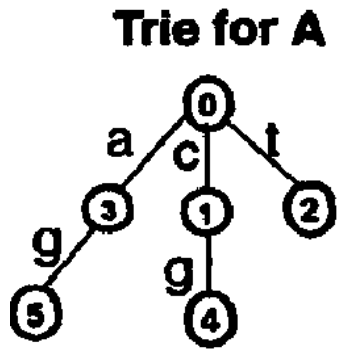


a | ac | t | acg | a |

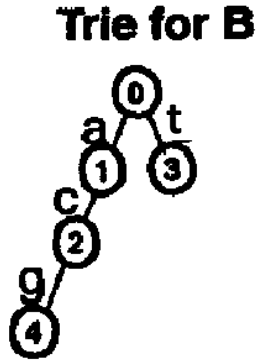
Structures de données

- Une « table de blocs » contenant une entrée par bloc.

		B							
		a	a	c	t	a	c	g	a
c									
t									
A a									
c									
g									
a									
g									
a									



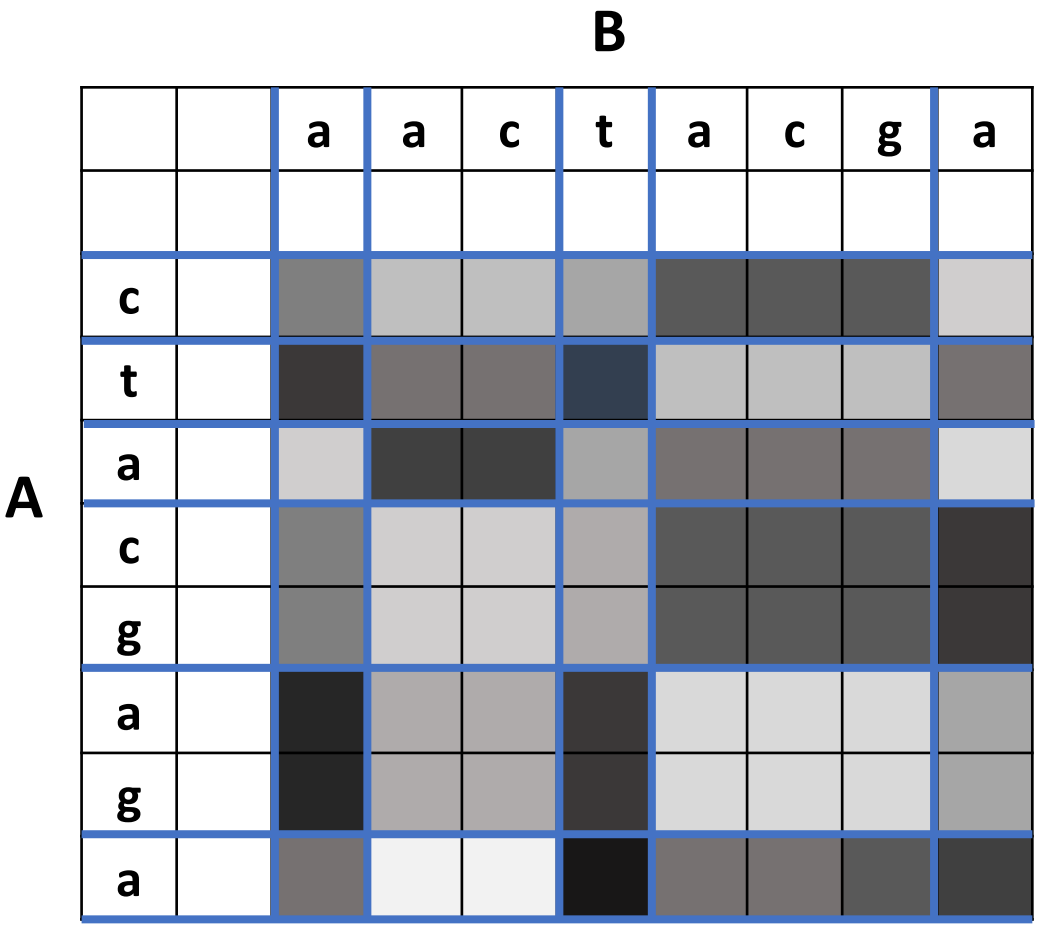
c | t | a | cg | ag | a |

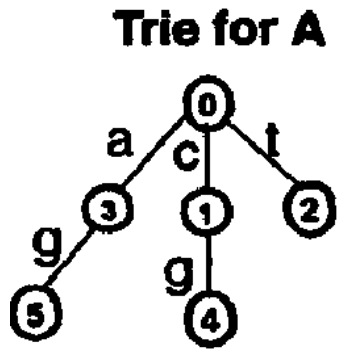


a | ac | t | acg | a |

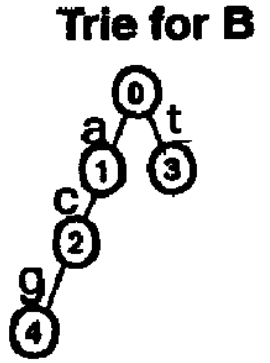
Structures de données

- Une « table de blocs » contenant une entrée par bloc.





c | t | a | cg | ag | a |



a | ac | t | acg | a |

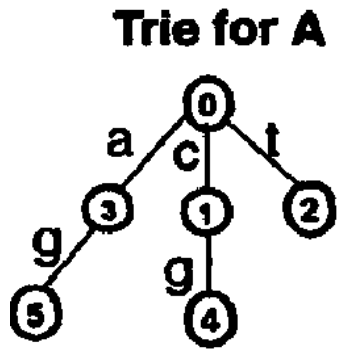
Structures de données

- Une « table de blocs » contenant une entrée par bloc.

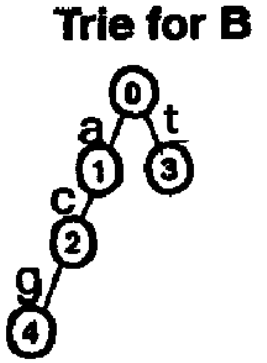
B

		a	a	c	t	a	c	g	a
c		(1,1)	(1,2)		(1,3)		(1,4)		(1,1)
t		(2,1)	(2,2)		(2,3)		(2,4)		(2,1)
a									
c									
g									
a									
g									
a									

A



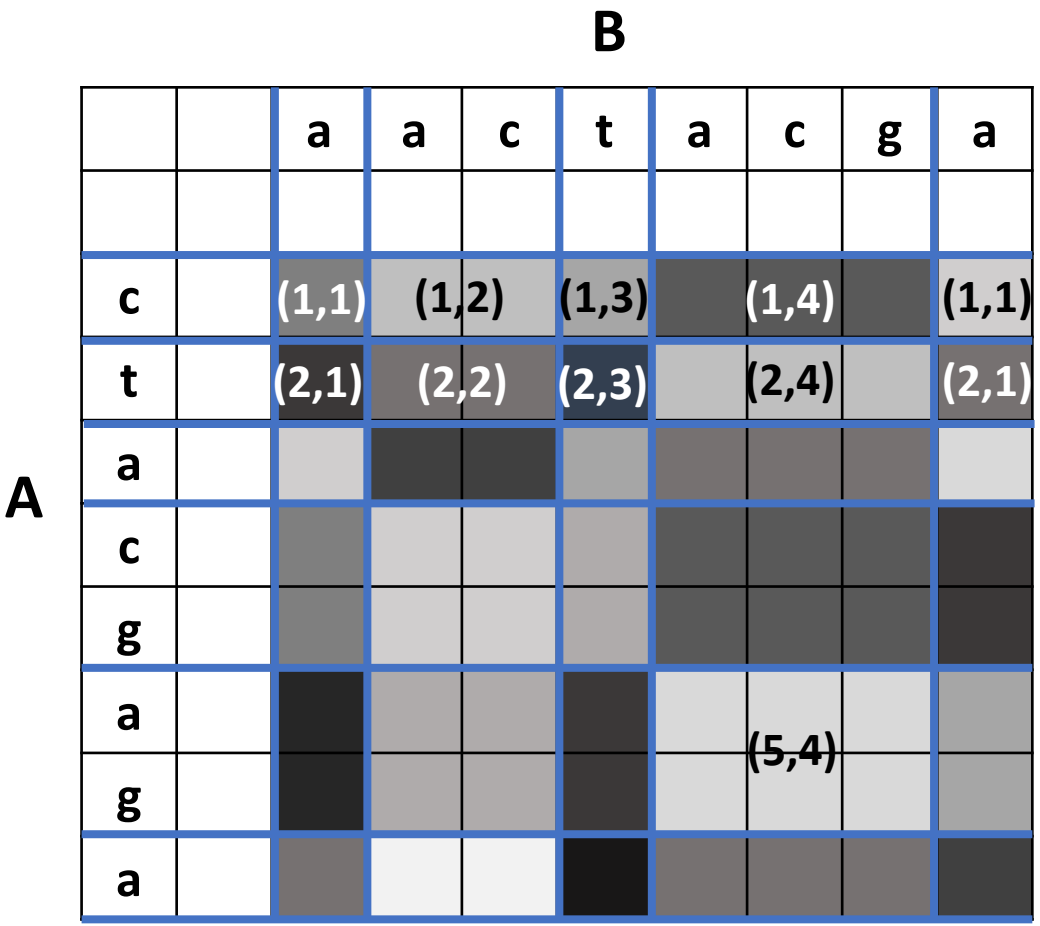
c | t | a | cg | ag | a |

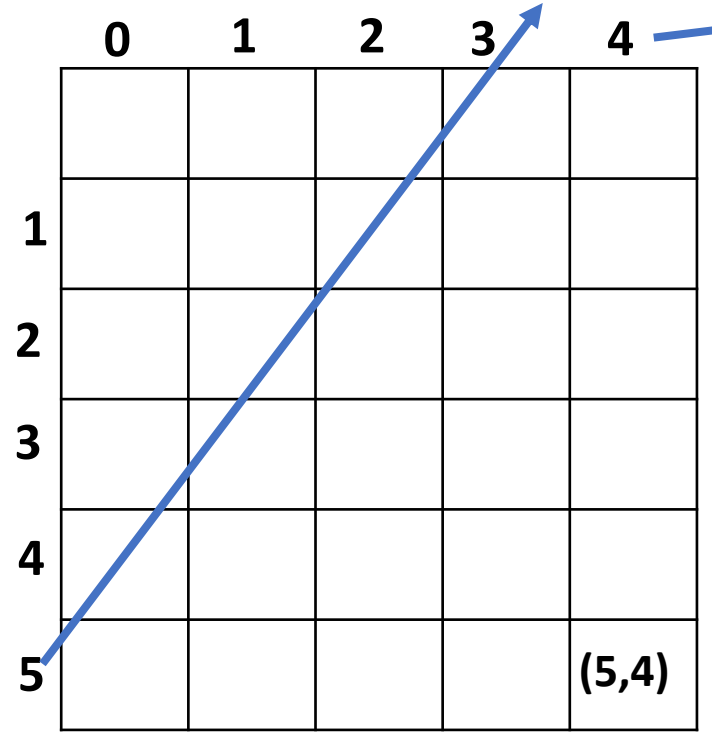


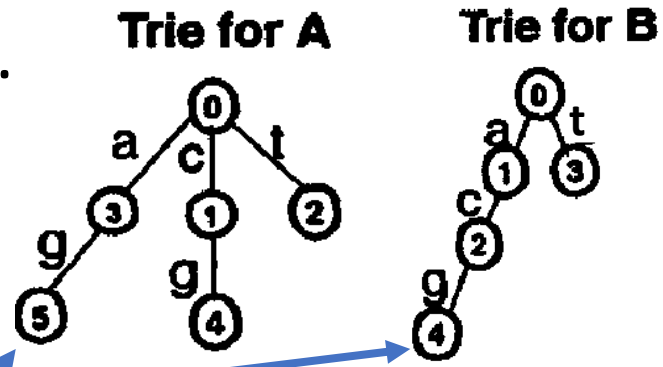
a | ac | t | acg | a |

Structures de données

- Une « table de blocs » contenant une entrée par bloc.



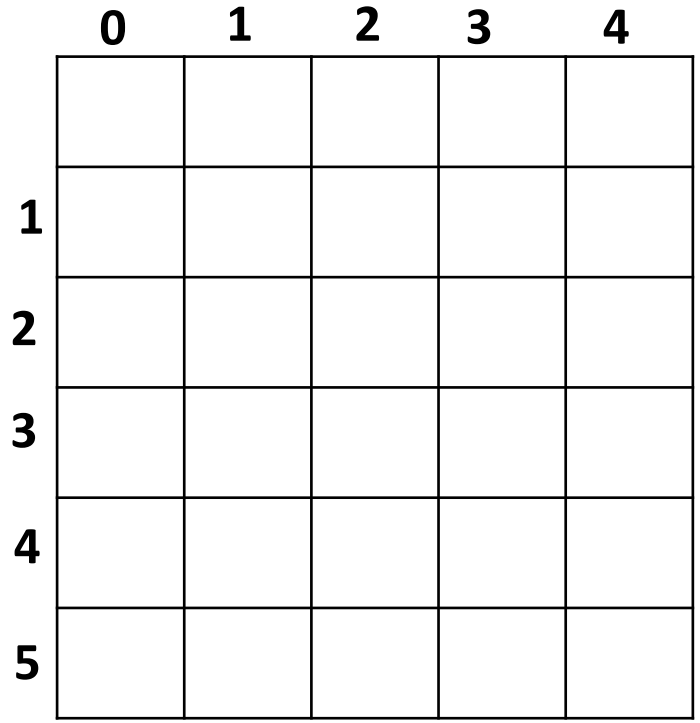




Taille de la table:
$O((h.n / \log n)^2)$

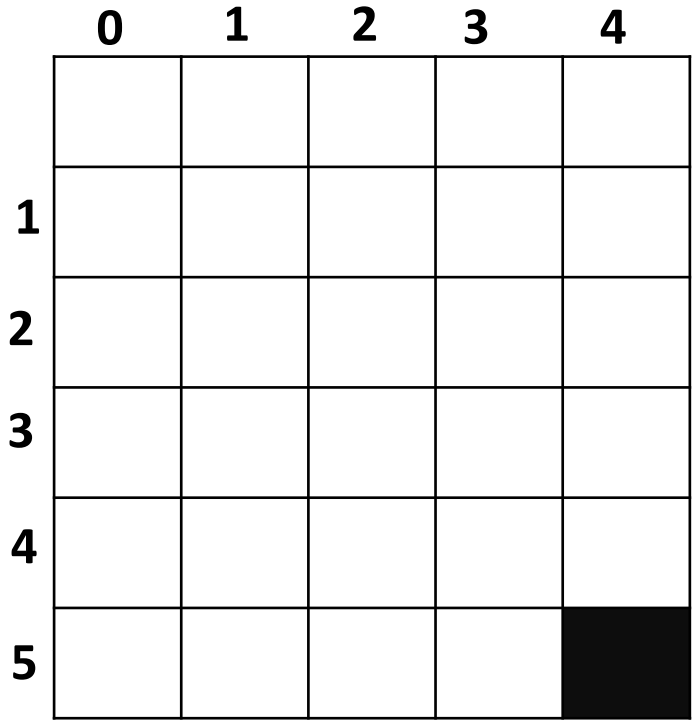
Structures de données

- À chaque case de la table te blocs, associer une seule colonne de la matrice *DIST* du bloc : la colonne correspondant à la cellule en bas à droite du bloc



Structures de données

- À chaque case de la table de blocs, associer une seule colonne de la matrice *DIST* du bloc : la colonne correspondant à la cellule en bas à droite du bloc



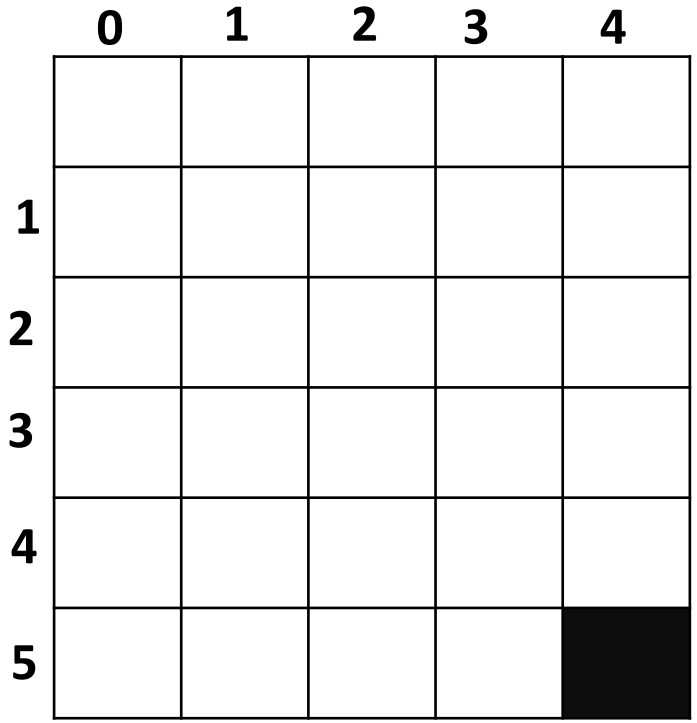
	a	c	g
	3	4	5	6
a	2			5
g	1	2	3	4

DIST

	1	2	3	4	5	6
1	0	-1	-2	-3		
2	-1	-1	-2	-1	-3	
3	-2	0	0	1	-1	-3
4		-2	-2	0	-2	-2
5			-2	0	-1	-1
6				-2	-1	0

Structures de données

- À chaque case de la table te blocs, associer une seule colonne de la matrice *DIST* du bloc : la colonne correspondant à la cellule en bas à droite du bloc



	a	c	g
	3	4	5	6
a	2			5
g	1	2	3	4

DIST

	1	2	3	4	5	6
1	0	-1	-2	-3		
2	-1	-1	-2	-1	-3	
3	-2	0	0	1	-1	-3
4		-2	-2	0	-2	-2
5			-2	0	-1	-1
6				-2	-1	0

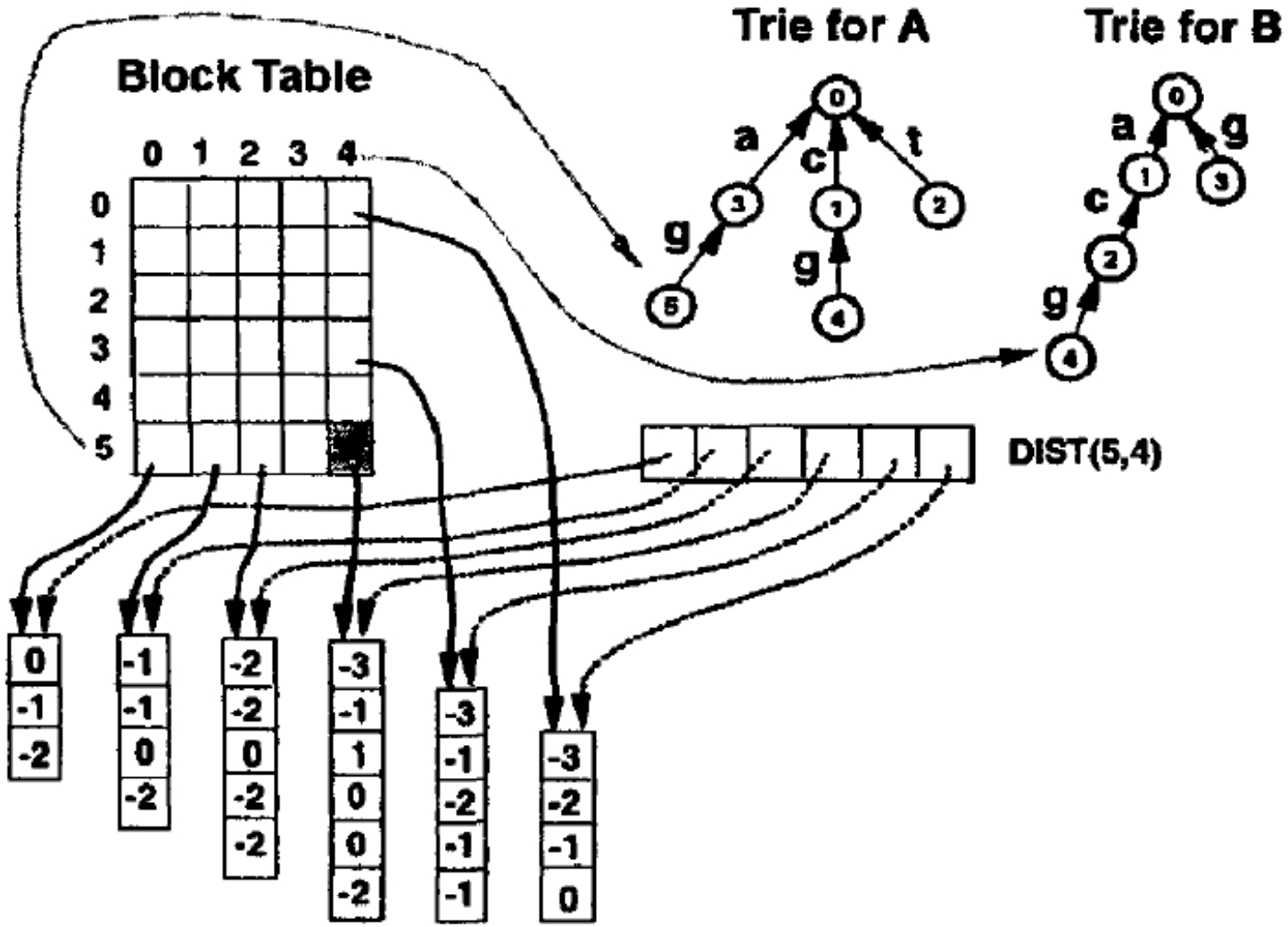
Algorithme

Pour chaque bloc G de périmètre p:

1. Calculer la colonne de la matrice *DIST* dont on a besoin (voir plus haut); se fait *O(p)* en utilisant l'info sur les blocs voisins (qui peuvent être trouvées en temps constant dans la table de blocs);
2. Compléter la matrice *DIST* en allant rechercher les colonnes correspondantes dans la table de blocs; se fait en temps *O(p);*
3. Calculer la matrice OUT et la Sortie. Se fait en temps *O(p).*

- Et donc le temps total est proportionnel au périmètre total des cellules, ce qui correspond à $O(h.n^2 / \log n)$

Pour l’étape 2



Pour l'étape 3: Propriétés des matrices path et OUT

- Ce sont des matrices Monge
  - Donc totalement monotones
- Algorithme SMAWK peut calculer tous les maximas des lignes et des colonnes d'une matrice totalement monotone en *O(p)*.

	1	2	3	4	5	6
1	1	0	-1	-2		
2	1	1	0	1	-1	
3	1	3	3	4	2	0
4		0	0	2	0	0
5			-1	1	0	0
6				1	2	3

DEFINITION 3.3. *A matrix* $M[0 \ldots m, 0 \ldots n]$ *is* **Monge** *if either condition 1 or 2 below holds for all* $a, b = 0 \ldots m;\ c, d = 0 \ldots n$*:*

1. **convex condition:** $M[a, c] + M[b, d] \leq M[b, c] + M[a, d]$ *for all* $a < b$ *and* $c < d$.
2. **concave condition:** $M[a, c] + M[b, d] \geq M[b, c] + M[a, d]$ *for all* $a < b$ *and* $c < d$.

DEFINITION 3.4. *A matrix* $M[0 \ldots m, 0 \ldots n]$ *is* **totally monotone** *if either condition 1 or 2 below holds for all* $a, b = 0 \ldots m;\ c, d = 0 \ldots n$*:*

1. **convex condition:** $M[a, c] \geq M[b, c] \Longrightarrow M[a, d] \geq M[b, d]$ *for all* $a < b$ *and* $c < d$.
2. **concave condition:** $M[a, c] \leq M[b, c] \Longrightarrow M[a, d] \leq M[b, d]$ *for all* $a < b$ *and* $c < d$.

Conclusion

- L'algorithme présenté permet une comparaison de deux séquences en temps $O(hn^2/\log(n))$.
- La matrice des scores n'est pas restreinte à des nombres rationnels.
- L'algorithme peut être étendu pour le problème de l'alignement local.
- La complexité en espace peut être réduite à $O(h^2n^2/(\log(n))^2)$ si la matrice des scores est restreinte à des scores rationnels.
- Il est possible, autant pour l'alignement global que l'alignement local, de retrouver l'alignement optimal avec une complexité en temps qui est linéaire en la taille du chemin correspondant à l'alignement dans le graphe, donc sans augmenter la complexité de l'algorithme.

Références

- ***Algorithms on Strings, Trees and Sequences – Computer science and Computational biology***, Dan Gusfield, Cambridge University Press, 1997.
- **Handbook of Computational Molecular Biology**, Srinivas Aluru ed., Chapman & Hall/CRC Computer and Information Science Series, 2005.
- **Pour l'algorithme de Crochemore et Landau:**
  - M. Crochemore, G.M. Landau and Z. Ziv-Ukelson. A sub-quadratic sequence alignment algorithm for unrestricted cost matrices. Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms, pp 679-688, 2002.