

Algorithmique – 2. Techniques algorithmiques
10. Deux techniques avancées

Bruno Grenet



<https://membres-ljk.imag.fr/Bruno.Grenet/Algorithmique.html>

Université Grenoble Alpes – IM²AG
L3 Mathématiques et Informatique

Table des matières

1. Recherche exhaustive rapide : exemple de 3-SAT

2. Diviser-pour-régner + programmation dynamique : exemple de la distance d'édition

Table des matières

1. Recherche exhaustive rapide : exemple de 3-SAT

2. Diviser-pour-régner + programmation dynamique : exemple de la distance d'édition

Définition du problème

Le problème SAT

Définition

Entrée : une formule logique φ à n variables booléennes, sous *forme normale conjonctive* (CNF)

Sortie : une affectation des variables qui satisfasse φ ; « insatisfiable » sinon

Formule logique CNF : *conjonction de disjonction de littéraux*

- ▶ **Littéraux :** $x_1, \neg x_1, \dots, x_n, \neg x_n$
- ▶ **Disjonction :** $C = x_1 \vee \neg x_3 \vee \neg x_4$ (clause)
- ▶ **Conjonction :** $C_1 \wedge C_2 \wedge \dots \wedge C_k$

$$\varphi(x_1, x_2, x_3) = (\neg x_1 \vee x_2) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge \neg x_2$$

Affectation satisfaisante ou non

- ▶ $(x_1, x_2, x_3) = (\text{FAUX}, \text{FAUX}, \text{FAUX})$ satisfait φ
- ▶ $(x_1, x_2, x_3) = (\text{VRAI}, \text{FAUX}, \text{VRAI})$ ne satisfait pas φ

- ▶ Formule 3-CNF : chaque clause possède exactement 3 littéraux

SAT : algorithme de recherche exhaustive

RECHERCHEEXHAUSTIVE(φ) :

1. $A \leftarrow$ tableau de longueur n , initialisé à -1
2. Tant que NON(SATISFAIT(φ, A)):
3. $A \leftarrow$ AFFSUIVANTE(A)
4. Si AFFSUIVANTE a renvoyé « Fin » : Renvoyer « Insatisfiable »
5. Renvoyer A

Propriétés

Correction : conséquence de la correction de SATISFAIT et AFFSUIVANTE

Complexité : nombre d'itérations $\leq 2^n$; coût d'une itération : $O(|\varphi| + n) = O(|\varphi|)$

Remarque : permet d'obtenir toutes les affectations satisfaisantes, ou leur nombre, ...

Théorème

L'algorithme RECHERCHEEXHAUSTIVE trouve une affectation satisfaisante s'il en existe une, et renvoie « Insatisfiable » sinon, en temps $O(|\varphi|2^n)$.

9/24

- Peut-on faire mieux que $O(|\varphi|2^n)$?

Outil : propagation de valeur

Definition

- ▶ $\varphi|_l$: « φ sachant l »
- ▶ Calcul : *propagation* de $l \rightarrow$ simplification de φ :
 - ▶ on supprime les clauses avec l
 - ▶ on supprime $\neg l$ des clauses qui le contiennent

l : littéral

PROPAGATION(φ, l)

1. $\psi \leftarrow \emptyset$ (formule vide)
2. Pour C dans φ :
3. Si $\neg l \in C$: ajouter $C \setminus \{\neg l\}$ à ψ
4. Sinon si $l \notin C$: ajouter C à ψ
5. Renvoyer ψ

Exemple

$x_1 = \text{VRAI}$
 $x_2 = \text{FAUX}$

$$\begin{aligned} & \text{---} (x_1 \vee \neg x_2 \vee x_3) \text{---} \\ \wedge & (\neg x_1 \vee \neg x_2 \vee x_4) \\ \wedge & \text{---} (x_1 \vee x_2 \vee \neg x_5) \text{---} \\ \wedge & (\neg x_3 \vee x_4 \vee x_5) \end{aligned}$$

Complexité : $\Theta(|\varphi|)$

Version récursive de la recherche exhaustive

Idée

- ▶ Pour chaque x_i , essayer $x_i = \text{VRAI}$ et $x_i = \text{FAUX}$ en propageant
- ▶ Appels récursifs sur $\varphi|_{x_i}$ et $\varphi|_{\neg x_i}$

3-SAT-EXHAU(φ, n):

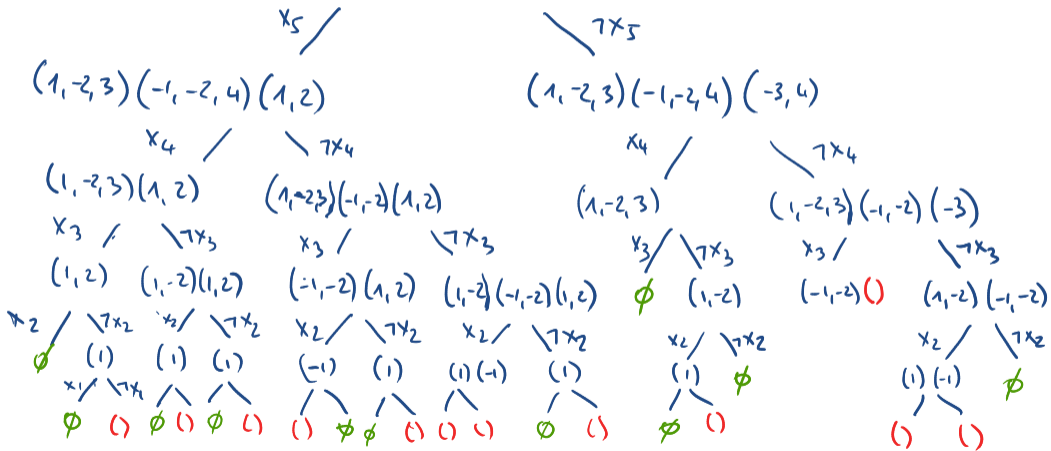
1. S'il existe une clause vide : renvoyer FAUX
2. Si $n = 0$: renvoyer VRAI
3. Si 3-SAT-EXHAU($\varphi|_{x_n}, n - 1$) : renvoyer VRAI
4. Si 3-SAT-EXHAU($\varphi|_{\neg x_n}, n - 1$) : renvoyer VRAI
5. Renvoyer FAUX

Complexité

$$T(n) = 2T(n-1) + \Theta(|\varphi|) = \Theta(|\varphi|2^n)$$

Exemple

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (x_1 \vee x_2 \vee \neg x_5) \wedge (\neg x_3 \vee x_4 \vee x_5)$$



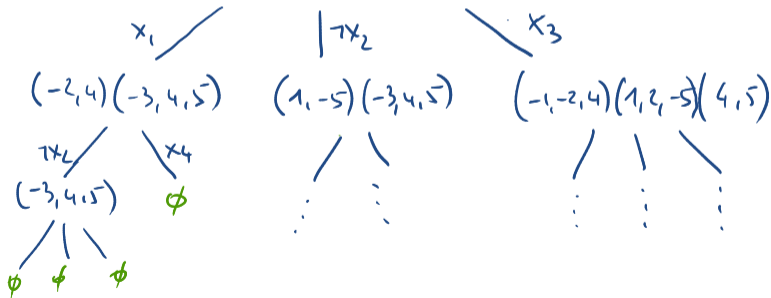
Une autre approche récursive

Idée

- ▶ Si $\varphi = (l_1 \vee l_2 \vee l_3) \wedge \varphi'$ est satisfiable, $l_1 = \text{VRAI}$ ou $l_2 = \text{VRAI}$ ou $l_3 = \text{VRAI}$
- ▶ Appels récursifs sur $\varphi|_{l_1}$, $\varphi|_{l_2}$ et $\varphi|_{l_3}$

Exemple

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (x_1 \vee x_2 \vee \neg x_5) \wedge (\neg x_3 \vee x_4 \vee x_5)$$



Algorithme par élimination

3-SAT-ÉLIM(φ):

1. S'il existe une clause vide: renvoyer FAUX
2. Si $\varphi = \emptyset$: renvoyer VRAI
3. $(l_1 \vee l_2 \vee l_3) \wedge \varphi' \leftarrow \varphi$
4. Si 3-SAT-ÉLIM($\varphi|_{l_1}$): renvoyer VRAI
5. Si 3-SAT-ÉLIM($\varphi|_{l_2}$): renvoyer VRAI
6. Si 3-SAT-ÉLIM($\varphi|_{l_3}$): renvoyer VRAI
7. Renvoyer FAUX

Complexité

$$T(n) = 3T(n-1) + \Theta(|\varphi|) = \Theta(3^n |\varphi|)$$

Amélioration

Idée

Si $l_1 \wedge \varphi'_{|l_1}$ n'est pas satisfiable, on peut supposer $\neg l_1$!

3-SAT-ÉLIM2(φ):

1. S'il existe une clause vide : renvoyer FAUX
2. Si $\varphi = \emptyset$: renvoyer VRAI
3. $(l_1 \vee l_2 \vee l_3) \wedge \varphi' \leftarrow \varphi$
4. Si 3-SAT-ÉLIM2($\varphi_{|l_1}$) : renvoyer VRAI
5. Si 3-SAT-ÉLIM2($\varphi_{|l_2, \neg l_1}$) : renvoyer VRAI
6. Si 3-SAT-ÉLIM2($\varphi_{|l_3, \neg l_1, \neg l_2}$) : renvoyer VRAI
7. Renvoyer FAUX

Complexité

$$T(n) = T(n-1) + T(n-2) + T(n-3) + \Theta(|\varphi|) = ???$$

Résolution de récurrences

Théorème admis

Soit $a_1, \dots, a_k \geq 0$ et $T(n)$ satisfaisant pour $n \geq k$

$$T(n) \leq \sum_{i=1}^k a_i T(n-i) + f(n)$$

Alors $T(n) = O(\lambda^n f(n))$ où λ est l'unique racine réelle ≥ 0 du polynôme $x^k - \sum_{i=1}^k a_i x^{n-i}$

Corollaire

Si $T(n) \leq T(n-1) + T(n-2) + T(n-3) + O(|\varphi|)$, alors $T(n) = O(1,83929^n \cdot |\varphi|)$

Nouvelle amélioration

Idée

- ▶ Si $\neg l$ n'apparaît pas dans φ , on peut supposer $l \rightarrow$ remplacer φ par $\varphi|_l$
- ▶ On peut supposer que toute variable x apparaît positivement et négativement dans φ

Lemme

Pour que $\varphi = (x \vee l_1 \vee l_2) \wedge (\neg x \vee l_3 \vee l_4) \wedge \varphi'$ soit satisfiable, quatre possibilités :

1. x et l_3 sont VRAIS et $\varphi'|_{x,l_3}$ est satisfiable
2. x et l_4 sont VRAIS et $\varphi'|_{x,l_4,\neg l_3}$ est satisfiable *on peut supposer l_3 FAUX*
3. $\neg x$ et l_1 sont VRAIS et $\varphi'|_{\neg x,l_1}$ est satisfiable
4. $\neg x$ et l_2 sont VRAIS et $\varphi'|_{\neg x,l_2,\neg l_1}$ est satisfiable *on peut supposer l_1 FAUX*

Algorithme et complexité

↳ 4 appels récursifs avec $n-2$ ou $n-3$ variables

$$\text{Complexité : } T(n) = 2T(n-2) + 2T(n-3) + \Theta(|\varphi|) = \Theta(1,7693^n |\varphi|)$$

Autre approche : la recherche locale

Idée de base

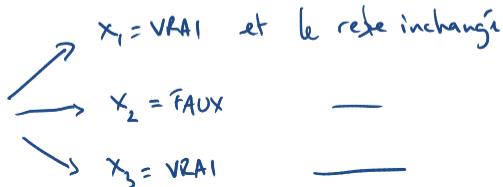
- ▶ Constat :
 - ▶ Si une affectation A ne satisfait pas φ , il existe une clause C de φ non satisfaite
 - ▶ Si $C = l_1 \vee l_2 \vee l_3$ n'est pas satisfaite, aucun de ses littéraux n'est satisfait
- ▶ Algorithme :
 - ▶ Partir d'une affectation A ; si elle satisfait φ , renvoyer A
 - ▶ Créer trois nouvelles affectations en modifiant l_1 ou l_2 ou l_3 et recommencer avec chacune

Exemple

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (x_1 \vee x_2 \vee \neg x_5) \wedge (\neg x_3 \vee x_4 \vee x_5)$$

$$\left. \begin{array}{l} x_1 = \text{FAUX} \\ x_2 = \text{VRAI} \\ x_3 = \text{FAUX} \\ x_4 = \text{VRAI} \\ x_5 = \text{FAUX} \end{array} \right\}$$

$(x_1 \vee \neg x_2 \vee x_3)$ non sat.



Algorithme de recherche locale

3-SAT-LOCAL(φ, A):

1. Si A satisfait φ : renvoyer VRAI
2. $C \leftarrow$ clause non satisfaite de φ
3. $x, y, z \leftarrow$ variables de C
4. $A_x \leftarrow$ affectation obtenue en changeant la valeur de x dans A
5. $A_y \leftarrow$ affectation obtenue en changeant la valeur de y dans A
6. $A_z \leftarrow$ affectation obtenue en changeant la valeur de z dans A
7. Si 3-SAT-LOCAL(φ, A_x): renvoyer VRAI
8. Si 3-SAT-LOCAL(φ, A_y): renvoyer VRAI
9. Si 3-SAT-LOCAL(φ, A_z): renvoyer VRAI
10. Renvoyer FAUX

Correction ?

L'algorithme boucle: $(x \vee \neg y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg z)$ et $A = (\text{VRAI}, \text{VRAI}, \text{VRAI})$

Algorithme de recherche locale bornée

3-SAT-LOCAL(φ, A, r):

1. Si A satisfait φ : renvoyer VRAI
2. Si $r = 0$: renvoyer FAUX
3. $C \leftarrow$ clause non satisfaite de φ
4. $x, y, z \leftarrow$ variables de C
5. $A_x \leftarrow$ affectation obtenue en changeant la valeur de x dans A
6. $A_y \leftarrow$ affectation obtenue en changeant la valeur de y dans A
7. $A_z \leftarrow$ affectation obtenue en changeant la valeur de z dans A
8. Si 3-SAT-LOCAL($\varphi, A_x, r - 1$): renvoyer VRAI
9. Si 3-SAT-LOCAL($\varphi, A_y, r - 1$): renvoyer VRAI
10. Si 3-SAT-LOCAL($\varphi, A_z, r - 1$): renvoyer VRAI
11. Renvoyer FAUX

Analyse de la recherche locale bornée

Distance entre deux affectations

Nombre de variables avec affectations différentes

- ▶ distance entre (VRAI, FAUX, FAUX) et (FAUX, VRAI, FAUX): 2
- ▶ distance entre (VRAI, FAUX, FAUX) et (VRAI, VRAI, FAUX): 1

Correction

3-SAT-LOCAL(φ, A, r) renvoie

- ▶ VRAI si φ est satisfaite par une affectation à distance $\leq r$ de A
- ▶ FAUX sinon

Complexité

$$T(n, r) \leq 3T(n, r - 1) + O(|\varphi|) \rightarrow \Theta(3^r \cdot |\varphi|)$$

Comment utiliser l'algorithme ?

Affectation de départ

- ▶ Par exemple $A_{\text{VRAI}} = (\text{VRAI}, \dots, \text{VRAI})$
- ▶ Si A satisfait φ , A et A_{VRAI} sont à distance $\leq n \rightarrow r = n$
- ▶ Complexité: $O(3^n \cdot |\varphi|)$

$(V, F, \bar{F}, V, V, \bar{F}, V, V)$

Observation

- ▶ Toute affectation a $\geq \frac{n}{2}$ VRAIS ou $\geq \frac{n}{2}$ FAUX
- ▶ Si A satisfait φ , alors A est à distance $\leq \frac{n}{2}$ de A_{VRAI} ou de $A_{\text{FAUX}} = (\text{FAUX}, \dots, \text{FAUX})$

Algorithme et complexité

- ▶ Deux appels à 3-SAT-LOCAL($\varphi, A, \frac{n}{2}$) avec $A = A_{\text{VRAI}}$ et $A = A_{\text{FAUX}}$
- ▶ Complexité: $O(3^{n/2} \cdot |\varphi|) = O(1,7321^n \cdot |\varphi|)$

Conclusion

Trois approches

- ▶ Variable par variable : 2^n
- ▶ Clause par clause : $1,7693^n$
- ▶ Recherche locale : $1,7321^n$

Autres algorithmes

- ▶ TD : $1,618^n$
- ▶ Théorie : $1,307^n$
- ▶ Pratique : autres approches
très efficaces

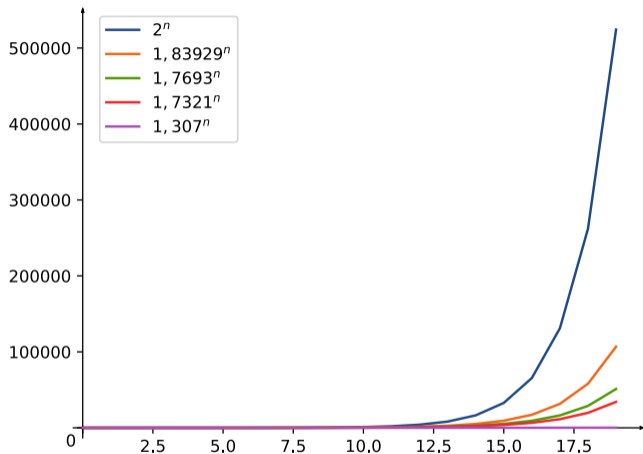


Table des matières

1. Recherche exhaustive rapide : exemple de 3-SAT

2. Diviser-pour-régner + programmation dynamique : exemple de la distance d'édition

Rappel du problème

Distance d'édition

Entrées: deux mots A et B de longueur m et n respectivement

Sortie 1: la *distance d'édition* $\Delta(A, B)$ entre A et B

Sortie 2: un *alignement* de A et B avec $\Delta(A, B)$ désaccords

Alignement et distance

HA GORRYT NE
ALGO RITHME

↳ distance 6

Rappel des algorithmes

Distances entre préfixes

- ▶ $\Delta_{i,j}$: distance entre $A_{[0,i]}$ et $B_{[0,j]}$ où
 - ▶ $A_{[0,i]} = A_{[0]}A_{[1]} \cdots A_{[i-1]}$
 - ▶ $B_{[0,j]} = B_{[0]}B_{[1]} \cdots B_{[j-1]}$
- ▶ $\Delta(A, B) = \Delta_{m,n}$

Lemme

$$\Delta_{i,j} = \min \begin{cases} \Delta_{i-1,j} + 1 \\ \Delta_{i,j-1} + 1 \\ \Delta_{i-1,j-1} + \delta_{A_{[i-1]} \neq B_{[j-1]}} \end{cases}$$

Théorème

On peut calculer $\Delta(A, B)$ et un alignement optimal en temps et espace $O(mn)$

Δ		A	L	G	O	R	I	T	H	M	E
	0	1	2	3	4	5	6	7	8	9	10
H	1										
A	2										
G	3										
O	4										
R	5										
R	6										
Y	7										
T	8										
N	9										
E	10										

Rappel des algorithmes

Distances entre préfixes

- ▶ $\Delta_{i,j}$: distance entre $A_{[0,i[}$ et $B_{[0,j[}$ où
 - ▶ $A_{[0,i[} = A_{[0}A_{[1} \cdots A_{[i-1}$
 - ▶ $B_{[0,j[} = B_{[0}B_{[1} \cdots B_{[j-1}$
- ▶ $\Delta(A, B) = \Delta_{m,n}$

Lemme

$$\Delta_{i,j} = \min \begin{cases} \Delta_{i-1,j} + 1 \\ \Delta_{i,j-1} + 1 \\ \Delta_{i-1,j-1} + \delta_{A_{[i-1]} \neq B_{[j-1]} \end{cases}$$

Δ		A	L	G	O	R	I	T	H	M	E
	0	1	2	3	4	5	6	7	8	9	10
H	1	1	2	3	4	5	6	7	7	8	9
A	2	1	2	3	4	5	6	7	8	8	9
G	3	2	2	2	3	4	5	6	7	8	9
O	4	3	3	3	2	3	4	5	6	7	8
R	5	4	4	4	3	2	3	4	5	6	7
R	6	5	5	5	4	3	3	4	5	6	7
Y	7	6	6	6	5	4	4	4	5	6	7
T	8	7	7	7	6	5	5	4	5	6	7
N	9	8	8	8	7	6	6	5	5	6	7
E	10	9	9	9	8	7	7	6	6	6	6

Théorème

On peut calculer $\Delta(A, B)$ et un alignement optimal en temps et espace $O(mn)$

Problématique de la mémoire

Et si on veut *moins* qu'un espace $O(mn)$?

Calcul de la distance

- ▶ Pour remplir la ligne i de Δ , on n'a besoin que de la ligne $i - 1$
- ▶ On ne retient que les lignes $i - 1$ et $i \rightarrow O(m)$
- ▶ Remarque : on peut échanger m et n pour avoir $m \leq n$

Calcul de l'alignement

- ▶ Besoin de toutes les lignes...
- ▶ Reconstruction impossible avec uniquement les deux dernières lignes

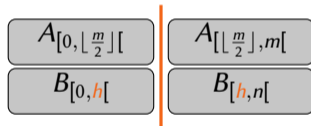
Une stratégie « diviser pour régner »

HA GORRYT NE
ALGO RITHME

Si on connaît...

... la partie de B à aligner avec $A_{[0, \lfloor \frac{m}{2} \rfloor[}$ et celle avec $A_{[\lfloor \frac{m}{2} \rfloor, m[}$

$h = 4$



On peut calculer...

... deux alignements $A_{[0, \lfloor \frac{m}{2} \rfloor[} \parallel B_{[0, h[}$ et $A_{[\lfloor \frac{m}{2} \rfloor, m[} \parallel B_{[h, n[}$

→ appels récurrents en tailles $(\lfloor \frac{m}{2} \rfloor, h)$ et $(\lceil \frac{m}{2} \rceil, n - h)$

Calculer le point central h

Généralisation aux préfixes

- ▶ $H_{i,j}$: $A_{[0, \lfloor \frac{m}{2} \rfloor]}$ est aligné avec $B_{[0, H_{i,j}]}$ dans un alignement optimal $A_{[0, i]} \parallel B_{[0, j]}$
 - ▶ $H_{i,j}$ défini uniquement pour $i \geq \lfloor \frac{m}{2} \rfloor$
 - ▶ $H_{i,j} = j$ si $i = \lfloor \frac{m}{2} \rfloor$
 - ▶ $H_{i,0} = 0$
- ▶ $h = H_{m,n}$

$$A_{[0, \lfloor \frac{m}{2} \rfloor]} \parallel B_{[0, j]}$$

Lemme

$$\text{Si } i > \lfloor \frac{m}{2} \rfloor, H_{i,j} = \begin{cases} H_{i-1,j} & \text{si } \Delta_{i,j} = \Delta_{i-1,j} + 1 \\ H_{i,j-1} & \text{si } \Delta_{i,j} = \Delta_{i,j-1} + 1 \\ H_{i-1,j-1} & \text{si } \Delta_{i,j} = \Delta_{i-1,j-1} + \delta_{A_{[i-1]} \neq B_{[j-1]}} \end{cases}$$

Preuve : alignements possibles

$A_{[0, \lfloor \frac{m}{2} \rfloor]}$	$A_{[\lfloor \frac{m}{2} \rfloor, i-1]}$	$A_{[i-1]}$
$B_{[0, h]}$	$B_{[h, j]}$	-

$A_{[0, \lfloor \frac{m}{2} \rfloor]}$	$A_{[\lfloor \frac{m}{2} \rfloor, i]}$	-
$B_{[0, h]}$	$B_{[h, j-1]}$	$B_{[j-1]}$

$A_{[0, \lfloor \frac{m}{2} \rfloor]}$	$A_{[\lfloor \frac{m}{2} \rfloor, i-1]}$	$A_{[i-1]}$
$B_{[0, h]}$	$B_{[h, j-1]}$	$B_{[j-1]}$

Exemple

Δ		A	L	G	O	R	I	T	H	M	E
	0	1	2	3	4	5	6	7	8	9	10
H	1										
A	2										
G	3										
O	4										
R	5										
R	6										
Y	7										
T	8										
N	9										
E	10										

H		A	L	G	O	R	I	T	H	M	E
	x	x	x	x	x	x	x	x	x	x	x
H	x	x	x	x	x	x	x	x	x	x	x
A	x	x	x	x	x	x	x	x	x	x	x
G	x	x	x	x	x	x	x	x	x	x	x
O	x	x	x	x	x	x	x	x	x	x	x
R	0	1	2	3	4	5	6	7	8	9	10
R	0										
Y	0										
T	0										
N	0										
E	0										

Exemple

Δ		A	L	G	O	R	I	T	H	M	E
	0	1	2	3	4	5	6	7	8	9	10
H	1	1	2	3	4	5	6	7	7	8	9
A	2	1	2	3	4	5	6	7	8	8	9
G	3	2	2	2	3	4	5	6	7	8	9
O	4	3	3	3	2	3	4	5	6	7	8
R	5	4	4	4	3	2	3	4	5	6	7
R	6										
Y	7										
T	8										
N	9										
E	10										

H		A	L	G	O	R	I	T	H	M	E
		x	x	x	x	x	x	x	x	x	x
H		x	x	x	x	x	x	x	x	x	x
A		x	x	x	x	x	x	x	x	x	x
G		x	x	x	x	x	x	x	x	x	x
O		x	x	x	x	x	x	x	x	x	x
R	0	1	2	3	4	5	6	7	8	9	10
R	0										
Y	0										
T	0										
N	0										
E	0										

Exemple

Δ	A	L	G	O	R	I	T	H	M	E	
	0	1	2	3	4	5	6	7	8	9	10
H	1	1	2	3	4	5	6	7	7	8	9
A	2	1	2	3	4	5	6	7	8	8	9
G	3	2	2	2	3	4	5	6	7	8	9
O	4	3	3	3	2	3	4	5	6	7	8
R	5	4	4	4	3	2	3	4	5	6	7
R	6	5	5	5	4	3	3	4	5	6	7
Y	7										
T	8										
N	9										
E	10										

H	A	L	G	O	R	I	T	H	M	E	
	x	x	x	x	x	x	x	x	x	x	
H	x	x	x	x	x	x	x	x	x	x	
A	x	x	x	x	x	x	x	x	x	x	
G	x	x	x	x	x	x	x	x	x	x	
O	x	x	x	x	x	x	x	x	x	x	
R	0	1	2	3	4	5	6	7	8	9	10
R	0	1	2	2	...	-					
Y	0										
T	0										
N	0										
E	0										

Exemple

Δ	A	L	G	O	R	I	T	H	M	E	H	A	L	G	O	R	I	T	H	M	E	
	0	1	2	3	4	5	6	7	8	9	10	x	x	x	x	x	x	x	x	x	x	x
H	1	1	2	3	4	5	6	7	7	8	9	x	x	x	x	x	x	x	x	x	x	x
A	2	1	2	3	4	5	6	7	8	8	9	x	x	x	x	x	x	x	x	x	x	x
G	3	2	2	2	3	4	5	6	7	8	9	x	x	x	x	x	x	x	x	x	x	x
O	4	3	3	3	2	3	4	5	6	7	8	x	x	x	x	x	x	x	x	x	x	x
R	5	4	4	4	3	2	3	4	5	6	7	0	1	2	3	4	5	6	7	8	9	10
R	6	5	5	5	4	3	3	4	5	6	7	0	1	1	2	4	4	5	5	5	5	5
Y	7	6	6	6	5	4	4	4	5	6	7	0	1	1	1	4	4	4	5	5	5	5
T	8	7	7	7	6	5	5	4	5	6	7	0	1	1	1	4	4	4	4	4	4	4
N	9	8	8	8	7	6	6	5	5	6	7	0	1	1	1	4	4	4	4	4	4	4
E	10	9	9	9	8	7	7	6	6	6	6	0	1	1	1	4	4	4	4	4	4	4

Exemple

Δ		A	L	G	O	R	I	T	H	M	E
	0	1	2	3	4	5	6	7	8	9	10
H	1	1	2	3	4	5	6	7	7	8	9
A	2	1	2	3	4	5	6	7	8	8	9
G	3	2	2	2	3	4	5	6	7	8	9
O	4	3	3	3	2	3	4	5	6	7	8
R	5	4	4	4	3	2	3	4	5	6	7
R	6	5	5	5	4	3	3	4	5	6	7
Y	7	6	6	6	5	4	4	4	5	6	7
T	8	7	7	7	6	5	5	4	5	6	7
N	9	8	8	8	7	6	6	5	5	6	7
E	10	9	9	9	8	7	7	6	6	6	6

H		A	L	G	O	R	I	T	H	M	E
		x	x	x	x	x	x	x	x	x	x
H		x	x	x	x	x	x	x	x	x	x
A		x	x	x	x	x	x	x	x	x	x
G		x	x	x	x	x	x	x	x	x	x
O		x	x	x	x	x	x	x	x	x	x
R		0	1	2	3	4	5	6	7	8	9
R		0	1	1	2	4	4	5	5	5	5
Y		0	1	1	1	4	4	4	5	5	5
T		0	1	1	1	4	4	4	4	4	4
N		0	1	1	1	4	4	4	4	4	4
E		0	1	1	1	4	4	4	4	4	4

Remarque : on peut ne retenir que deux lignes de Δ et H

Calcul de h : l'algorithme

CALCULH(A, B):

1. $(m, n) \leftarrow$ tailles de A et B
2. $\Delta^{-1}, \Delta^0, H^{-1}, H^0 \leftarrow$ tableaux de taille $n + 1$ *(lignes précédente et courante)*
3. Pour $j = 0$ à n : $\Delta_{[j]}^{-1} \leftarrow j$; $H_{[j]}^0 \leftarrow j$
4. Pour $i = 1$ à m :
 5. $\Delta_{[0]}^0 \leftarrow i$; $H_{[0]}^0 \leftarrow 0$
 6. Pour $j = 1$ à n :
 7. $\Delta_{[j]}^0 \leftarrow \min(\Delta_{[j]}^{-1} + 1, \Delta_{[j-1]}^0 + 1, \Delta_{[j-1]}^{-1} + \delta_{A_{[i-1]} \neq B_{[j-1]}})$
 8. Si $i > \lfloor \frac{m}{2} \rfloor$:
 9. Si $\Delta_{[j]}^0 = \Delta_{[j]}^{-1} + 1$: $H_{[j]}^0 \leftarrow H_{[j]}^{-1}$
 10. Sinon si $\Delta_{[j]}^0 = \Delta_{[j-1]}^0 + 1$: $H_{[j]}^0 \leftarrow H_{[j-1]}^0$
 11. Sinon: $H_{[j]}^0 \leftarrow H_{[j-1]}^{-1}$
 12. Pour $j = 0$ à n : $\Delta_{[j]}^{-1} \leftarrow \Delta_{[j]}^0$; $H_{[j]}^{-1} \leftarrow H_{[j]}^0$ *(courante \rightarrow précédente)*
13. Renvoyer $H_{[n]}^0$

L'algorithme récursif

ALIGNEMENTDPR(A, B):

1. $(m, n) \leftarrow$ tailles de A et B
2. Si $m = 0$: renvoyer (" $_{(n)}$ ", B)
3. Si $n = 0$: renvoyer (A , " $_{(m)}$ ")
4. $h \leftarrow$ CALCULH(A, B)
5. $(A_1, B_1) \leftarrow$ ALIGNEMENTDPR($A_{[0, \lfloor \frac{m}{2} \rfloor]}$, $B_{[0, h]}$)
6. $(A_2, B_2) \leftarrow$ ALIGNEMENTDPR($A_{[\lfloor \frac{m}{2} \rfloor, m]}$, $B_{[h, n]}$)
7. Renvoyer ($A_1 \cdot A_2, B_1 \cdot B_2$)

n blancs

m blancs

(concaténations)

Complexité

- ▶ Espace: $O(\min(m, n))$ pour CALCULH et réutilisation de l'espace $\rightarrow O(\min(m, n))$
- ▶ Temps: $T(m, n) \leq T(\lfloor \frac{m}{2} \rfloor, h) + T(\lceil \frac{m}{2} \rceil, n - h) + O(mn) \leq c \cdot mn$

$$T(m, n) \leq 2 \cdot c \cdot mn: \quad T(m, n) \leq \cancel{2c} \cdot \cancel{m/2} \cdot h + \cancel{2c} \cdot \cancel{m/2} \cdot (n-h) + c \cdot mn = 2c \cdot mn.$$

si $m = 2^k$

$$T(m, n) = \Theta(mn)$$

Conclusion

Distance d'édition

- ▶ Programmation dynamique en temps $O(mn)$ et espace $O(mn)$
- ▶ Version économe en mémoire $O(m)$ pour la distance, sans alignement
- ▶ Ajout de « diviser pour régner » : version économe en mémoire pour l'alignement

De manière plus générale

- ▶ Programmation dynamique gourmande en mémoire
- ▶ Très souvent : version économe en mémoire sans reconstruction
- ▶ Reconstruction avec « diviser pour régner » :
 - ▶ « même » temps que l'algorithme de base multiplié par une constante, voire $\log(n)$
 - ▶ « même » espace que la version économe multiplié par une constante