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

Table des matières

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 : $x_1, \neg x_1, \ldots, x_n, \neg x_n$
- Disjonction : $C = x_1 \vee \neg x_3 \vee \neg x_4$ (clause)
- Conjonction : $C_1 \wedge C_2 \wedge \cdots \wedge C_k$

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

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

Complexité

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. $\quad A \leftarrow$ AffSuivante(A)
4. $\quad$ 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 $O(|\varphi|2^n)$.

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

Outil : propagation de valeur

Definition

- $\varphi_{|\ell}$: « φ sachant ℓ »
- Calcul : *propagation* de $\ell \to$ simplification de φ :
  - on supprime les clauses avec ℓ
  - on supprime $\neg\ell$ des clauses qui le contiennent

ℓ : littéral

Propagation(φ, ℓ)

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

Exemple

$\varphi_{|x_1, \neg x_4, x_2}$

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

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

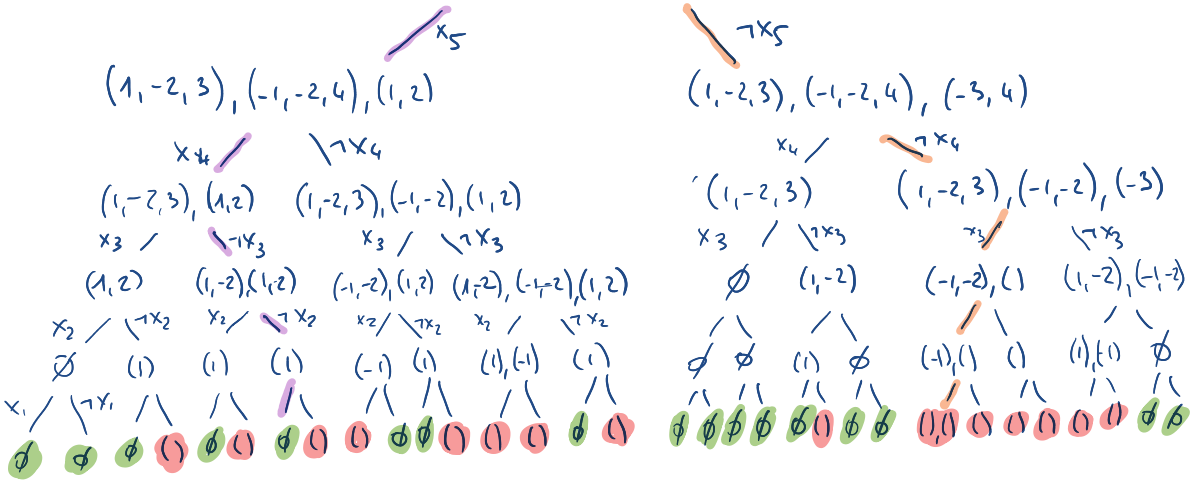
$$T(n) \leq 2T(n-1) + \Theta(|\varphi|) \rightsquigarrow T(n) = \Theta(2^n|\varphi|)$$

Exemple

$x_1, \neg x_2, \neg x_3, x_4, x_5$ → satisfait

$x_1, x_2, x_3, \neg x_4, \neg x_5$ → ne satisfait pas.

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



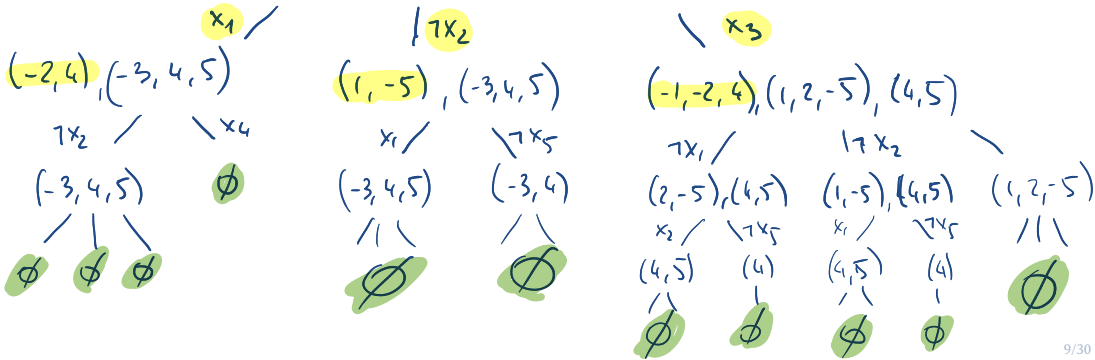
Une autre approche récursive

Idée

- Si $\varphi = (\ell_1 \vee \ell_2 \vee \ell_3) \wedge \varphi'$ est satisfiable, $\ell_1 =$ VRAI ou $\ell_2 =$ VRAI ou $\ell_3 =$ VRAI
- Appels récursifs sur $\varphi'_{|\ell_1}$, $\varphi'_{|\ell_2}$ et $\varphi'_{|\ell_3}$

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. $(\ell_1 \vee \ell_2 \vee \ell_3) \wedge \varphi' \leftarrow \varphi$
4. Si 3-Sat-élim($\varphi_{|\ell_1}$) : renvoyer vrai
5. Si 3-Sat-élim($\varphi_{|\ell_2}$) : renvoyer vrai
6. Si 3-Sat-élim($\varphi_{|\ell_3}$) : renvoyer vrai
7. Renvoyer faux

Complexité

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

Amélioration

Idée

Si $\ell_1 \wedge \varphi'_{|\ell_1}$ n'est pas satisfiable, on peut supposer $\neg\ell_1$!

3-Sat-élim2(φ) :

1. S'il existe une clause vide : renvoyer FAUX
2. Si $\varphi = \emptyset$: renvoyer VRAI
3. $(\ell_1 \vee \ell_2 \vee \ell_3) \wedge \varphi' \leftarrow \varphi$
4. Si 3-Sat-élim2($\varphi_{|\ell_1}$) : renvoyer VRAI
5. Si 3-Sat-élim2($\varphi_{|\ell_2,\neg\ell_1}$) : renvoyer VRAI
6. Si 3-Sat-élim2($\varphi_{|\ell_3,\neg\ell_1,\neg\ell_2}$) : renvoyer VRAI
7. Renvoyer FAUX

Complexité

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

Résolution de récurrences

Théorème

Soit $a_1, \ldots, a_{k-1} \geq 0$, $a_k > 0$, $f(n) \nearrow$ et $T(n)$ satisfiant 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 de $x^k - \sum_{i=1}^{k} a_i x^{k-i}$

Corollaire

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

$x^3 - x^2 - x - 1 \rightsquigarrow$ racine ≥ 0 est $1,83929\ldots$

Preuve du théorème

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

① Poly. caractéristique $\chi = X^k - \sum_{i=1}^{k} a_i X^{k-i}$ a une unique racine réelle positive

Réc. sur k : $\chi' = k \cdot X^{k-1} - \sum_{i=1}^{k-1} a_i (k-i) X^{k-1-i}$ donc $\chi'/_k = X^{k-1} - \sum_{i=1}^{k-1} a_i \frac{k-i}{k} X^{k-1-i}$

Par hyp. de réc., $\chi'/_k$ (donc χ') a une unique racine réelle positive x_0

Donc χ décroît de 0 à x_0 puis croît de x_0 à $+\infty$.

Or $\chi(0) = -a_k < 0$ et $\lim_{+\infty} \chi = +\infty$.

$\Rightarrow$ χ a une unique racine réelle positive ✓

Preuve du théorème

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

① Poly. caractéristique $\chi = X^k - \sum_{i=1}^{k} a_i X^{k-i}$ a une unique racine réelle positive

② $T(n) \leq c \cdot \lambda^n + \frac{1}{1-A} f(n)$ où λ est la racine de χ et $A = \sum_{i=1}^{k} a_i$.

$$T(n) \leq \sum_{i=1}^{k} a_i T(n-i) + f(n) \leq \sum_{i=1}^{k} a_i \left(c\lambda^{n-i} + \frac{1}{1-A} f(n-i) \right) + f(n) \quad (*)$$

Or $\lambda^k = \sum_i a_i \lambda^{k-i}$ donc $\sum_i a_i \lambda^{n-i} = \lambda^n$ et $f \nearrow$ donc $f(n-i) \leq f(n)$.

$$(*) \leq c\lambda^n + \sum_{i=1}^{k} a_i \frac{1}{1-A} f(n) + f(n) = c \cdot \lambda^n + A \frac{1}{1-A} f(n) + f(n)$$

$$= c\lambda^n + \frac{1}{1-A} f(n)$$

Nouvelle amélioration

Idée

- Si $\neg\ell$ n'apparaît pas dans φ, on peut supposer $\ell \to$ remplacer φ par $\varphi_{|\ell}$
- On peut supposer que toute variable x apparaît positivement et négativement dans φ

Lemme

Pour que $\varphi = (x \vee \ell_1 \vee \ell_2) \wedge (\neg x \vee \ell_3 \vee \ell_4) \wedge \varphi'$ soit satisfiable, quatre possibilités :

1. x et ℓ_3 sont VRAIS et $\varphi'_{|x,\ell_3}$ est satisfiable
2. x et ℓ_4 sont VRAIS et $\varphi'_{|x,\ell_4,\neg\ell_3}$ est satisfiable — *on peut supposer ℓ_3 FAUX*
3. $\neg x$ et ℓ_1 sont VRAIS et $\varphi'_{|\neg x,\ell_1}$ est satisfiable
4. $\neg x$ et ℓ_2 sont VRAIS et $\varphi'_{|\neg x,\ell_2,\neg\ell_1}$ est satisfiable — *on peut supposer ℓ_1 FAUX*

Algorithme et complexité

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

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 $C = \ell_1 \vee \ell_2 \vee \ell_3$ 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

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

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

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

Comment utiliser l'algorithme ?

Affectation de départ

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

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 $A_{\text{FAUX}} = (\text{FAUX}, \ldots, \text{FAUX})$

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

Et pour ($k-$)SAT?

- λ^n avec $1,307 < \lambda < 2$

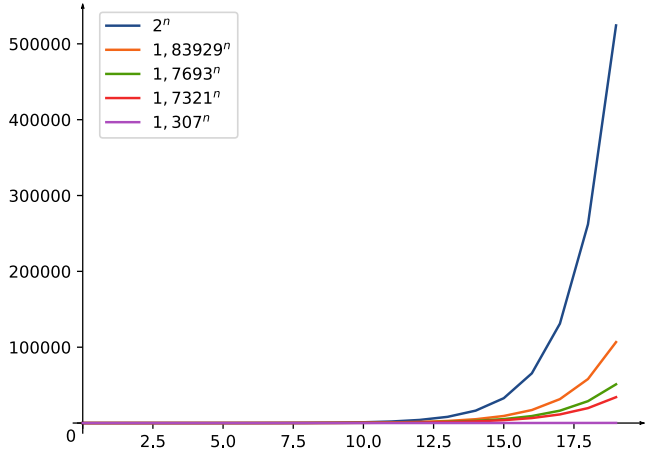


Table des matières

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

Rappel des algorithmes

Distances entre préfixes

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

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										

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 $O(mn)$

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 $i \to O(m)$
- 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

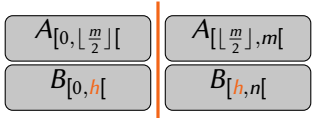
Une stratégie « diviser pour régner »

HA GOR | RYT 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 $A_{[0,\lfloor \frac{m}{2} \rfloor[} \| B_{[0,h[}$ et $A_{[\lfloor \frac{m}{2} \rfloor,m[} \| B_{[h,n[}$

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

Calculer le point central h

Généralisation aux préfixes

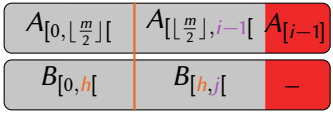
- $H_{i,j}$: $A_{[0,\lfloor \frac{m}{2} \rfloor[}$ est aligné avec $B_{[0,H_{i,j}[}$ dans un alignement optimal $A_{[0,i[} \| B_{[0,j[}$
  - $H_{i,j}$ défini uniquement pour $i \geq \lfloor \frac{m}{2} \rfloor$
  - $H_{i,j} = j$ si $i = \lfloor \frac{m}{2} \rfloor$
  - $H_{i,0} = 0$
- $h = H_{m,n}$

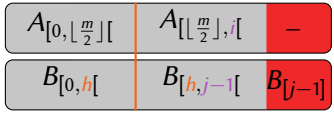
$A_{[0,\lfloor \frac{m}{2} \rfloor[} \| B_{[0,j[}$

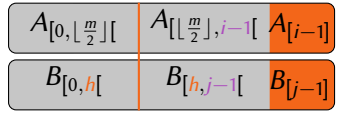
Lemme

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

Preuve : alignements possibles







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	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	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	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	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	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	1	1	2	4	4	5	5	5	5	5
Y	0	1	1	1	4	4	4	5	5	5	5
T	0	1	1	1	4	4	4	4	4	4	4
N	0	1	1	1	4	4	4	4	4	4	4
E	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	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	1	1	2	4	4	5	5	5	5	5
Y	0	1	1	1	4	4	4	5	5	5	5
T	0	1	1	1	4	4	4	4	4	4	4
N	0	1	1	1	4	4	4	4	4	4	4
E	0	1	1	1	4	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 : $\Delta^{-1}_{[j]} \leftarrow j$; $H^{-1}_{[j]} \leftarrow j$
4. Pour $i = 1$ à m :
5. $\quad \Delta^{0}_{[0]} \leftarrow i$; $H^{0}_{[0]} \leftarrow 0$
6. $\quad$ Pour $j = 1$ à n :
7. $\quad\quad \Delta^{0}_{[j]} \leftarrow \min(\Delta^{-1}_{[j]} + 1, \Delta^{0}_{[j-1]} + 1, \Delta^{-1}_{[j-1]} + \delta_{A_{[i-1]} \neq B_{[j-1]}})$
8. $\quad\quad$ Si $i > \lfloor \frac{m}{2} \rfloor$:
9. $\quad\quad\quad$ Si $\Delta^{0}_{[j]} = \Delta^{-1}_{[j]} + 1$: $H^{0}_{[j]} \leftarrow H^{-1}_{[j]}$
10. $\quad\quad\quad$ Sinon si $\Delta^{0}_{[j]} = \Delta^{0}_{[j-1]} + 1$: $H^{0}_{[j]} \leftarrow H^{0}_{[j-1]}$
11. $\quad\quad\quad$ Sinon : $H^{0}_{[j]} \leftarrow H^{-1}_{[j-1]}$
12. $\quad$ Pour $j = 0$ à n : $\Delta^{-1}_{[j]} \leftarrow \Delta^{0}_{[j]}$; $H^{-1}_{[j]} \leftarrow H^{0}_{[j]}$ *(courante → précédente)*
13. Renvoyer $H^{0}_{[n]}$

L'algorithme récursif

AlignementDpR(A, B) :

1. $(m, n) \leftarrow$ tailles de A et B
2. Si $m = 0$: renvoyer $(\text{"}_{-(n)-}\text{"}, B)$ *n blancs*
3. Si $n = 0$: renvoyer $(A, \text{"}_{-(m)-}\text{"})$ *m blancs*
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 $(A_1 \cdot A_2, B_1 \cdot B_2)$ (concaténations)

Complexité

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

Tq $T(m,n) \leq 2cmn$: $T(m,n) \leq T(\lfloor \frac{m}{2} \rfloor, h) + T(\lceil \frac{m}{2} \rceil, n-h) + cmn \leq 2c\frac{m}{2}h + 2c\frac{m}{2}(n-h) + cmn$

$\leq 2cmn$

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