Programmation dynamique

DIU Enseignement de l'informatique au lycée
Bloc 5 : Algorithmique avancée

Université de Montpellier
Juin 2021

Définition du problème

Entrées Un ensemble P d'entiers, une somme s (entière)

Sortie 1 Le nombre minimal de pièces de P dont la somme vaut s

Sortie 2 Une liste minimale de pièces de P, de somme s

Définition du problème

Entrées Un ensemble P d'entiers, une somme s (entière)

Sortie 1 Le nombre minimal de pièces de P dont la somme vaut s

Sortie 2 Une liste minimale de pièces de P, de somme s

Exemple

Entrées 

somme 14€53

Définition du problème

Entrées Un ensemble P d'entiers, une somme s (entière)

Sortie 1 Le nombre minimal de pièces de P dont la somme vaut s

Sortie 2 Une liste minimale de pièces de P, de somme s

Exemple

Entrées 

somme 14€53

Sortie 1 6

Sortie 2 

Algorithme glouton

1. Rendre la plus grande pièce (ou billet) possible
2. Mettre à jour la somme à rendre
3. Recommencer tant que la somme est > 0

Algorithme glouton

1. Rendre la plus grande pièce (ou billet) possible
2. Mettre à jour la somme à rendre
3. Recommencer tant que la somme est > 0

- Optimal pour (quasiment) toutes les monnaies fiduciaires...
  - Bloc 2, TD 2, exercice 7
- ... mais pas toujours : $P = \{1, 4, 6\}$ et $s = 8$
  - $6 + 1 + 1$ au lieu de $4 + 4$

Une formule récursive

$$\text{pieces}_P(s) = \text{nombre minimal de pièces de } P \text{ dont la somme vaut } s$$

- Si $s = 0$: $\text{pieces}_P(0) = 0$
- Sinon, pour $p \leq s$ dans P : $\text{pieces}_P(s) \leq 1 + \text{pieces}_P(s - p)$

Une formule récursive

$$\text{pieces}_P(s) = \text{nombre minimal de pièces de } P \text{ dont la somme vaut } s$$

- Si $s = 0$: $\text{pieces}_P(0) = 0$
- Sinon, pour $p \leq s$ dans P : $\text{pieces}_P(s) \leq 1 + \text{pieces}_P(s - p)$

$$\text{pieces}_P(s) = \begin{cases} 0 \text{ si } s = 0 \\ 1 + \min\{\text{pieces}_P(s - p) : p \in P, p \leq s\} & \text{sinon} \end{cases}$$

Une formule récursive

$$\text{pieces}_P(s) = \text{nombre minimal de pièces de } P \text{ dont la somme vaut } s$$

- Si $s = 0$: $\text{pieces}_P(0) = 0$
- Sinon, pour $p \leq s$ dans P : $\text{pieces}_P(s) \leq 1 + \text{pieces}_P(s - p)$

$$\text{pieces}_P(s) = \begin{cases} 0 \text{ si } s = 0 \\ 1 + \min\{\text{pieces}_P(s - p) : p \in P, p \leq s\} \quad \text{sinon} \end{cases}$$

Exemple Si $P = \{1, 6, 14, 23\}$ et $s = 17$

- Soit on rend 14, et il reste 3 : $\text{pieces}_P(17) \leq 1 + \text{pieces}_P(3)$
- Soit on rend 6, et il reste 11 : $\text{pieces}_P(17) \leq 1 + \text{pieces}_P(11)$
- Soit on rend 1, et il reste 16 : $\text{pieces}_P(17) \leq 1 + \text{pieces}_P(16)$

$\rightsquigarrow$ $\text{pieces}_P(17) = 1 + \min\{\text{pieces}_P(3), \text{pieces}_P(11), \text{pieces}_P(16)\}$

Mauvaise idée : algorithme récursif !

$$\text{pieces}_P(s) = \begin{cases} 0 \text{ si } s = 0 \\ 1 + \min\{\text{pieces}_P(s-p) : p \in P, p \leq s\} & \text{sinon} \end{cases}$$

RenduNaïf(P, s)

1. Si $s = 0$: renvoyer 0
2. $n \leftarrow +\infty$
3. Pour chaque pièce $p \in P$:
4. Si $p \leq s$:
5. $n_p \leftarrow$ RenduNaïf$(P, s - p)$
6. Si $n_p \leq n$: $n \leftarrow n_p$
7. Renvoyer $1 + n$

Mauvaise idée : algorithme récursif !

$$\text{pieces}_P(s) = \begin{cases} 0 \text{ si } s = 0 \\ 1 + \min\{\text{pieces}_P(s-p) : p \in P, p \leq s\} & \text{sinon} \end{cases}$$

RENDUNAÏF(P, s)

1. Si $s = 0$: renvoyer 0
2. $n \leftarrow +\infty$
3. Pour chaque pièce $p \in P$:
4. Si $p \leq s$:
5. $n_p \leftarrow$ RENDUNAÏF$(P, s-p)$
6. Si $n_p \leq n$: $n \leftarrow n_p$
7. Renvoyer $1 + n$

- $t_P(s) = O(|P|) + \sum_{p \leq s} t_P(s-p)$
  $= O(|P|^s)$

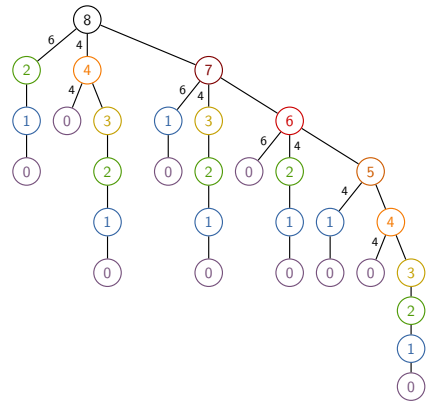
Mauvaise idée : algorithme récursif !

$$\text{pieces}_P(s) = \begin{cases} 0 \text{ si } s = 0 \\ 1 + \min\{\text{pieces}_P(s-p) : p \in P, p \leq s\} & \text{sinon} \end{cases}$$

RENDUNAÏF(P, s)

1. Si $s = 0$: renvoyer 0
2. $n \leftarrow +\infty$
3. Pour chaque pièce $p \in P$:
4. Si $p \leq s$:
5. $n_p \leftarrow$ RENDUNAÏF$(P, s-p)$
6. Si $n_p \leq n$: $n \leftarrow n_p$
7. Renvoyer $1 + n$

- $t_P(s) = O(|P|) + \sum_{p \leq s} t_P(s-p)$
  $= O(|P|^s)$



Idée moyenne : mémoïsation

≪ Mémoïsation ≫ : écrire des informations dans un mémo.

RenduMemo(P, s, V)

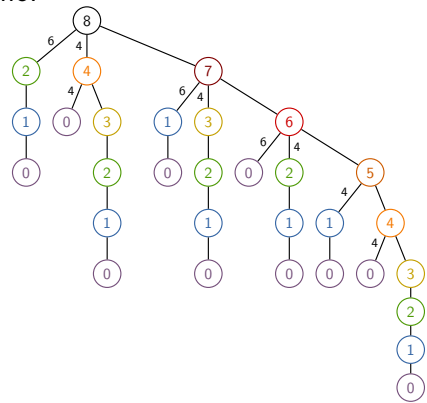
1. Si s est dans V : renvoyer V_s
2. $n \leftarrow +\infty$
3. Pour chaque pièce $p \in P$:
4. Si $p \leq s$:
5. $n_p \leftarrow$ RenduMemo$(P, s - p, V)$
6. Si $n_p \leq n$: $n \leftarrow n_p$
7. $V_s \leftarrow 1 + n$
8. Renvoyer $1 + n$

Idée moyenne : mémoïsation

≪ Mémoïsation ≫ : écrire des informations dans un mémo.

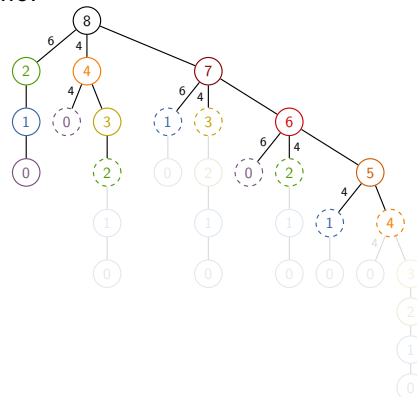
RenduMemo(P, s, V)

1. Si s est dans V : renvoyer V_s
2. $n \leftarrow +\infty$
3. Pour chaque pièce $p \in P$:
4. Si $p \leq s$:
5. $n_p \leftarrow$ RenduMemo$(P, s - p, V)$
6. Si $n_p \leq n$: $n \leftarrow n_p$
7. $V_s \leftarrow 1 + n$
8. Renvoyer $1 + n$



Idée moyenne : mémoïsation

≪ Mémoïsation ≫ : écrire des informations dans un mémo.



RenduMemo(P, s, V)

1. Si s est dans V : renvoyer V_s
2. $n \leftarrow +\infty$
3. Pour chaque pièce $p \in P$:
4. Si $p \leq s$:
5. $n_p \leftarrow$ RenduMemo$(P, s - p, V)$
6. Si $n_p \leq n$: $n \leftarrow n_p$
7. $V_s \leftarrow 1 + n$
8. Renvoyer $1 + n$

Bonne idée : programmation dynamique

Calculer $\text{pieces}_P(s)$ par valeurs de s croissantes

RenduProgdyn(P, s)

1. $L \leftarrow$ tableau de longueur $s+1$, initialisé à $+\infty$
2. $L_0 \leftarrow 0$
3. Pour $i = 1$ à s :
4. Pour chaque pièce $p \in P$:
5. Si $p \leq i$ et $1 + L_{i-p} < L_i$:
6. $L_i \leftarrow 1 + L_{i-p}$
7. Renvoyer L_s

Bonne idée : programmation dynamique

Calculer pieces$_P(s)$ par valeurs de s croissantes

RENDUPROGDYN(P, s)

1. $L \leftarrow$ tableau de longueur $s+1$, initialisé à $+\infty$
2. $L_0 \leftarrow 0$
3. Pour $i = 1$ à s :
4. Pour chaque pièce $p \in P$:
5. Si $p \leq i$ et $1 + L_{i-p} < L_i$:
6. $L_i \leftarrow 1 + L_{i-p}$
7. Renvoyer L_s

$P = \{1, 4, 6\}$, $s = 8$

$L = [0, \infty, \infty, \infty, \infty, \infty, \infty, \infty, \infty]$

Bonne idée : programmation dynamique

Calculer $\text{pieces}_P(s)$ par valeurs de s croissantes

RenduProgdyn(P, s)

1. $L \leftarrow$ tableau de longueur $s+1$, initialisé à $+\infty$
2. $L_0 \leftarrow 0$
3. Pour $i = 1$ à s :
4. Pour chaque pièce $p \in P$:
5. Si $p \leq i$ et $1 + L_{i-p} < L_i$:
6. $L_i \leftarrow 1 + L_{i-p}$
7. Renvoyer L_s

$P = \{1, 4, 6\}$, $s = 8$

$L = [0, \ 1, \infty, \infty, \infty, \infty, \infty, \infty, \infty]$

Bonne idée : programmation dynamique

Calculer $\text{pieces}_P(s)$ par valeurs de s croissantes

RenduProgdyn(P, s)

1. $L \leftarrow$ tableau de longueur $s+1$, initialisé à $+\infty$
2. $L_0 \leftarrow 0$
3. Pour $i = 1$ à s :
4. Pour chaque pièce $p \in P$:
5. Si $p \leq i$ et $1 + L_{i-p} < L_i$:
6. $L_i \leftarrow 1 + L_{i-p}$
7. Renvoyer L_s

$P = \{1, 4, 6\}$, $s = 8$

$L = [0, \ 1, \ 2, \infty, \infty, \infty, \infty, \infty, \infty]$

Bonne idée : programmation dynamique

Calculer $\text{pieces}_P(s)$ par valeurs de s croissantes

RenduProgdyn(P, s)

1. $L \leftarrow$ tableau de longueur $s+1$, initialisé à $+\infty$
2. $L_0 \leftarrow 0$
3. Pour $i = 1$ à s :
4. Pour chaque pièce $p \in P$:
5. Si $p \leq i$ et $1 + L_{i-p} < L_i$:
6. $L_i \leftarrow 1 + L_{i-p}$
7. Renvoyer L_s

$P = \{1, 4, 6\}$, $s = 8$

$L = [0, \ 1, \ 2, \ 3, \infty, \infty, \infty, \infty, \infty]$

Bonne idée : programmation dynamique

Calculer $\text{pieces}_P(s)$ par valeurs de s croissantes

RENDUPROGDYN(P, s)

1. $L \leftarrow$ tableau de longueur $s+1$, initialisé à $+\infty$
2. $L_0 \leftarrow 0$
3. Pour $i = 1$ à s :
4. Pour chaque pièce $p \in P$:
5. Si $p \leq i$ et $1 + L_{i-p} < L_i$:
6. $L_i \leftarrow 1 + L_{i-p}$
7. Renvoyer L_s

$P = \{1, 4, 6\}$, $s = 8$

$L = [0, \ 1, \ 2, \ 3, \ 4, \infty, \infty, \infty, \infty]$

Bonne idée : programmation dynamique

Calculer $\text{pieces}_P(s)$ par valeurs de s croissantes

RenduProgdyn(P, s)

1. $L \leftarrow$ tableau de longueur $s+1$, initialisé à $+\infty$
2. $L_0 \leftarrow 0$
3. Pour $i = 1$ à s :
4. Pour chaque pièce $p \in P$:
5. Si $p \leq i$ et $1 + L_{i-p} < L_i$:
6. $L_i \leftarrow 1 + L_{i-p}$
7. Renvoyer L_s

$P = \{1, 4, 6\}$, $s = 8$

$L = [0, \ 1, \ 2, \ 3, \ 1, \infty, \infty, \infty, \infty]$

Bonne idée : programmation dynamique

Calculer $\text{pieces}_P(s)$ par valeurs de s croissantes

RenduProgdyn(P, s)

1. $L \leftarrow$ tableau de longueur $s+1$, initialisé à $+\infty$
2. $L_0 \leftarrow 0$
3. Pour $i = 1$ à s :
4. Pour chaque pièce $p \in P$:
5. Si $p \leq i$ et $1 + L_{i-p} < L_i$:
6. $L_i \leftarrow 1 + L_{i-p}$
7. Renvoyer L_s

$P = \{1, 4, 6\}$, $s = 8$

$L = [0,\ 1,\ 2,\ 3,\ 1,\ 2, \infty, \infty, \infty]$

Bonne idée : programmation dynamique

Calculer $\text{pieces}_P(s)$ par valeurs de s croissantes

RenduProgdyn(P, s)

1. $L \leftarrow$ tableau de longueur $s+1$, initialisé à $+\infty$
2. $L_0 \leftarrow 0$
3. Pour $i = 1$ à s :
4. Pour chaque pièce $p \in P$:
5. Si $p \leq i$ et $1 + L_{i-p} < L_i$:
6. $L_i \leftarrow 1 + L_{i-p}$
7. Renvoyer L_s

$P = \{1, 4, 6\}$, $s = 8$

$L = [0, \ 1, \ 2, \ 3, \ 1, \ 2, \ 3, \infty, \infty]$

Bonne idée : programmation dynamique

Calculer $\text{pieces}_P(s)$ par valeurs de s croissantes

RENDUPROGDYN(P, s)

1. $L \leftarrow$ tableau de longueur $s+1$, initialisé à $+\infty$
2. $L_0 \leftarrow 0$
3. Pour $i = 1$ à s :
4. Pour chaque pièce $p \in P$:
5. Si $p \leq i$ et $1 + L_{i-p} < L_i$:
6. $L_i \leftarrow 1 + L_{i-p}$
7. Renvoyer L_s

$P = \{1, 4, 6\}$, $s = 8$

$L = [0,\ 1,\ 2,\ 3,\ 1,\ 2,\ 1, \infty, \infty]$

Bonne idée : programmation dynamique

Calculer $\text{pieces}_P(s)$ par valeurs de s croissantes

RenduProgdyn(P, s)

1. $L \leftarrow$ tableau de longueur $s+1$, initialisé à $+\infty$
2. $L_0 \leftarrow 0$
3. Pour $i = 1$ à s :
4. Pour chaque pièce $p \in P$:
5. Si $p \leq i$ et $1 + L_{i-p} < L_i$:
6. $L_i \leftarrow 1 + L_{i-p}$
7. Renvoyer L_s

$P = \{1, 4, 6\}$, $s = 8$

$L = [0,\ 1,\ 2,\ 3,\ 1,\ 2,\ 1,\ 2, \infty]$

Bonne idée : programmation dynamique

Calculer pieces$_P(s)$ par valeurs de s croissantes

RenduProgdyn(P, s)

1. $L \leftarrow$ tableau de longueur $s+1$, initialisé à $+\infty$
2. $L_0 \leftarrow 0$
3. Pour $i = 1$ à s :
4. Pour chaque pièce $p \in P$:
5. Si $p \leq i$ et $1 + L_{i-p} < L_i$:
6. $L_i \leftarrow 1 + L_{i-p}$
7. Renvoyer L_s

$P = \{1, 4, 6\}$, $s = 8$

$L = [0,\ 1,\ 2,\ 3,\ 1,\ 2,\ 1,\ 2,\ 3]$

Bonne idée : programmation dynamique

Calculer $\text{pieces}_P(s)$ par valeurs de s croissantes

RenduProgdyn(P, s)

1. $L \leftarrow$ tableau de longueur $s+1$, initialisé à $+\infty$
2. $L_0 \leftarrow 0$
3. Pour $i = 1$ à s :
4. Pour chaque pièce $p \in P$:
5. Si $p \leq i$ et $1 + L_{i-p} < L_i$:
6. $L_i \leftarrow 1 + L_{i-p}$
7. Renvoyer L_s

$P = \{1, 4, 6\}$, $s = 8$

$L = [0,\ 1,\ 2,\ 3,\ 1,\ 2,\ 1,\ 2,\ 2]$

Correction et complexité

Théorème

L'algorithme RENDUPROGDYN *calcule* $\text{pieces}_P(s)$ *en temps* $O(s|P|)$.

RENDUPROGDYN(P, s)

1. $L \leftarrow$ tableau de longueur $s+1$ initialisé à $+\infty$
2. $L_0 \leftarrow 0$
3. Pour $i = 1$ à s :
4. Pour chaque pièce $p \in P$:
5. Si $p \leq i$ et $1 + L_{i-p} < L_i$:
6. $L_i \leftarrow 1 + L_{i-p}$
7. Renvoyer L_s

Correction et complexité

Théorème
L'algorithme RENDUPROGDYN *calcule* $\text{pieces}_P(s)$ *en temps* $O(s|P|)$.

RENDUPROGDYN(P, s)
1. $L \leftarrow$ tableau de longueur $s+1$ initialisé à $+\infty$
2. $L_0 \leftarrow 0$
3. Pour $i = 1$ à s :
4. Pour chaque pièce $p \in P$:
5. Si $p \leq i$ et $1 + L_{i-p} < L_i$:
6. $L_i \leftarrow 1 + L_{i-p}$
7. Renvoyer L_s

Preuve de correction : utilisation de la formule récursive
Preuve de complexité : double boucle

Correction et complexité

Théorème
L'algorithme RENDUPROGDYN *calcule* $\text{pieces}_P(s)$ *en temps* $O(s|P|)$.

RENDUPROGDYN(P, s)
1. $L \leftarrow$ tableau de longueur $s+1$ initialisé à $+\infty$
2. $L_0 \leftarrow 0$
3. Pour $i = 1$ à s :
4. Pour chaque pièce $p \in P$:
5. Si $p \leq i$ et $1 + L_{i-p} < L_i$:
6. $L_i \leftarrow 1 + L_{i-p}$
7. Renvoyer L_s

Preuve de correction : utilisation de la formule récursive
Preuve de complexité : double boucle

Comment calculer la liste des pièces ?
- Retenir le tableau L en entier
- Le parcourir pour reconstruire une solution

Reconstruction

RenduReconstruction(P, s, L)

1. $S \leftarrow$ liste vide
2. Tant que $s > 0$:
3. Pour chaque pièce $p \in P$:
4. Si $p \leq s$ et $L_{s-p} = L_s - 1$:
5. $s \leftarrow s - p$
6. Ajouter p à S
7. Sortir de la boucle Pour
8. Renvoyer S

Reconstruction

RenduReconstruction(P, s, L)

1. $S \leftarrow$ liste vide
2. Tant que $s > 0$:
3. Pour chaque pièce $p \in P$:
4. Si $p \leq s$ et $L_{s-p} = L_s - 1$:
5. $s \leftarrow s - p$
6. Ajouter p à S
7. Sortir de la boucle Pour
8. Renvoyer S

$P = \{1, 4, 6\}$, $s = 8$

$L = [0, 1, 2, 3, 1, 2, 1, 2, 2]$

Reconstruction

RenduReconstruction(P, s, L)

1. $S \leftarrow$ liste vide
2. Tant que $s > 0$:
3. Pour chaque pièce $p \in P$:
4. Si $p \leq s$ et $L_{s-p} = L_s - 1$:
5. $s \leftarrow s - p$
6. Ajouter p à S
7. Sortir de la boucle Pour
8. Renvoyer S

$P = \{1, 4, 6\}$, $s = 8$

$L = [0, 1, 2, 3, 1, 2, 1, 2, 2]$

$s = 8$ $L_{s-1} = 2$

Reconstruction

RenduReconstruction(P, s, L)

1. $S \leftarrow$ liste vide
2. Tant que $s > 0$:
3. Pour chaque pièce $p \in P$:
4. Si $p \leq s$ et $L_{s-p} = L_s - 1$:
5. $s \leftarrow s - p$
6. Ajouter p à S
7. Sortir de la boucle Pour
8. Renvoyer S

$P = \{1, 4, 6\}$, $s = 8$

$L = [0, 1, 2, 3, 1, 2, 1, 2, 2]$

$s = 8$ $L_{s-4} = 1$

Reconstruction

RenduReconstruction(P, s, L)

1. $S \leftarrow$ liste vide
2. Tant que $s > 0$:
3. Pour chaque pièce $p \in P$:
4. Si $p \leq s$ et $L_{s-p} = L_s - 1$:
5. $s \leftarrow s - p$
6. Ajouter p à S
7. Sortir de la boucle Pour
8. Renvoyer S

$P = \{1, 4, 6\}$, $s = 8$

$L = [0, 1, 2, 3, 1, 2, 1, 2, 2]$

$s = 8$ $L_{s-4} = 1$

$\rightsquigarrow$ ajouter 4 à S

Reconstruction

RenduReconstruction(P, s, L)

1. $S \leftarrow$ liste vide
2. Tant que $s > 0$:
3. Pour chaque pièce $p \in P$:
4. Si $p \leq s$ et $L_{s-p} = L_s - 1$:
5. $s \leftarrow s - p$
6. Ajouter p à S
7. Sortir de la boucle Pour
8. Renvoyer S

$P = \{1, 4, 6\}$, $s = 8$

$L = [0, 1, 2, 3, 1, 2, 1, 2, 2]$

$s = 8$ $L_{s-4} = 1$

$\rightsquigarrow$ ajouter 4 à S

$s = 4$ $L_{s-1} = 3$

Reconstruction

RenduReconstruction(P, s, L)

1. $S \leftarrow$ liste vide
2. Tant que $s > 0$:
3. Pour chaque pièce $p \in P$:
4. Si $p \leq s$ et $L_{s-p} = L_s - 1$:
5. $s \leftarrow s - p$
6. Ajouter p à S
7. Sortir de la boucle Pour
8. Renvoyer S

$P = \{1, 4, 6\}$, $s = 8$

$L = [0, 1, 2, 3, 1, 2, 1, 2, 2]$

$s = 8$ $L_{s-4} = 1$

$\rightsquigarrow$ ajouter 4 à S

$s = 4$ $L_{s-4} = 0$

Reconstruction

RenduReconstruction(P, s, L)

1. $S \leftarrow$ liste vide
2. Tant que $s > 0$:
3. Pour chaque pièce $p \in P$:
4. Si $p \leq s$ et $L_{s-p} = L_s - 1$:
5. $s \leftarrow s - p$
6. Ajouter p à S
7. Sortir de la boucle Pour
8. Renvoyer S

$P = \{1, 4, 6\}$, $s = 8$

$L = [0, 1, 2, 3, 1, 2, 1, 2, 2]$

$s = 8$ $L_{s-4} = 1$

$\rightsquigarrow$ ajouter 4 à S

$s = 4$ $L_{s-4} = 0$

$\rightsquigarrow$ ajouter 4 à S

Reconstruction

RenduReconstruction(P, s, L)

1. $S \leftarrow$ liste vide
2. Tant que $s > 0$:
3. Pour chaque pièce $p \in P$:
4. Si $p \leq s$ et $L_{s-p} = L_s - 1$:
5. $s \leftarrow s - p$
6. Ajouter p à S
7. Sortir de la boucle Pour
8. Renvoyer S

$P = \{1, 4, 6\}$, $s = 8$

$L = [0, 1, 2, 3, 1, 2, 1, 2, 2]$

$s = 8$ $L_{s-4} = 1$

$\rightsquigarrow$ ajouter 4 à S

$s = 4$ $L_{s-4} = 0$

$\rightsquigarrow$ ajouter 4 à S

$s = 0$ $S = [4, 4]$

Reconstruction

RenduReconstruction(P, s, L)

1. $S \leftarrow$ liste vide
2. Tant que $s > 0$:
3. Pour chaque pièce $p \in P$:
4. Si $p \leq s$ et $L_{s-p} = L_s - 1$:
5. $s \leftarrow s - p$
6. Ajouter p à S
7. Sortir de la boucle Pour
8. Renvoyer S

$P = \{1, 4, 6\}$, $s = 8$

$L = [0, 1, 2, 3, 1, 2, 1, 2, 2]$

$s = 8$ $L_{s-4} = 1$

$\rightsquigarrow$ ajouter 4 à S

$s = 4$ $L_{s-4} = 0$

$\rightsquigarrow$ ajouter 4 à S

$s = 0$ $S = [4, 4]$

Lemme

L'algorithme RenduReconstruction *renvoie une liste minimale de pièces en temps* $O(n|P|)$ *où* $n = \text{pieces}_P(s)$.

1. Premier exemple : le rendu de monnaie

2. Qu'est-ce que la programmation dynamique ?

3. Deuxième exemple : la distance d'édition

Idée générale

Programmation dynamique = récursion sans répétition

Idée générale

Programmation dynamique = récursion sans répétition

Ingrédients

1. Formule **récursive** pour la valeur optimale
   - en fonction des valeurs de sous-problèmes
   - sous-problèmes possiblement nombreux et non disjoints

Idée générale

Programmation dynamique = récursion sans répétition

Ingrédients

1. Formule **récursive** pour la valeur optimale
   - en fonction des valeurs de sous-problèmes
   - sous-problèmes possiblement nombreux et non disjoints
2. Algorithme **itératif** pour la valeur optimale
   - en commençant par les plus petits sous-problèmes
   - approche « *bottom-up* »

Idée générale

Programmation dynamique = récursion sans répétition

Ingrédients

1. Formule **récursive** pour la valeur optimale
   - en fonction des valeurs de sous-problèmes
   - sous-problèmes possiblement nombreux et non disjoints
2. Algorithme **itératif** pour la valeur optimale
   - en commençant par les plus petits sous-problèmes
   - approche « *bottom-up* »
3. Reconstruction de la solution ***a posteriori***
   - ajout d'informations à l'algo. pour la valeur
   - algorithme de reconstruction indépendant

Idée générale

Programmation dynamique = récursion sans répétition

Ingrédients

1. Formule **récursive** pour la valeur optimale
   - en fonction des valeurs de sous-problèmes
   - sous-problèmes possiblement nombreux et non disjoints
2. Algorithme **itératif** pour la valeur optimale
   - en commençant par les plus petits sous-problèmes
   - approche « *bottom-up* »
3. Reconstruction de la solution ***a posteriori***
   - ajout d'informations à l'algo. pour la valeur
   - algorithme de reconstruction indépendant

« Diviser pour régner »

Sous-problèmes disjoints, approche « *top-down* »

Ingrédient 1 : Formule récursive

Partie la plus importante (et difficile) !

Ingrédient 1 : Formule récursive

Partie la plus importante (et difficile) !

Étapes

1. Spécification **précise** du problème
2. Formule récursive basée sur les solutions d'instances plus petites du **même problème exactement**

Ingrédient 1 : Formule récursive

Partie la plus importante (et difficile) !

Étapes

1. Spécification **précise** du problème
2. Formule récursive basée sur les solutions d'instances plus petites du **même problème exactement**

Rendu de monnaie

Expression de $\text{pieces}_P(s)$ à partir des $\text{pieces}_P(s-p)$, $p \in P$

Ingrédient 1 : Formule récursive

Partie la plus importante (et difficile) !

Étapes

1. Spécification **précise** du problème
2. Formule récursive basée sur les solutions d'instances plus petites du **même problème exactement**

Rendu de monnaie

Expression de $\text{pieces}_P(s)$ à partir des $\text{pieces}_P(s-p)$, $p \in P$

En pratique, étape souvent (très) guidée

Ingrédient 2 : Algorithme itératif

Partie plutôt facile... mais attention quand même !

Ingrédient 2 : Algorithme itératif

Partie plutôt facile... mais attention quand même !

Étapes

1. choix d'une structure de données (*très* souvent un tableau)
2. **ordre de calcul** si tableau multi-dimensionnel *cf. ex. suivant*
3. écriture effective de l'algorithme
4. analyse de complexité

Ingrédient 2 : Algorithme itératif

Partie plutôt facile... mais attention quand même !

Étapes

1. choix d'une structure de données (*très* souvent un tableau)
2. **ordre de calcul** si tableau multi-dimensionnel *cf. ex. suivant*
3. écriture effective de l'algorithme
4. analyse de complexité

Rendu de monnaie

1. Tableau L
2. Ordre croissant

Ingrédient 2 : Algorithme itératif

Partie plutôt facile... mais attention quand même !

Étapes

1. choix d'une structure de données (*très* souvent un tableau)
2. **ordre de calcul** si tableau multi-dimensionnel *cf. ex. suivant*
3. écriture effective de l'algorithme
4. analyse de complexité

Rendu de monnaie

1. Tableau L
2. Ordre croissant

En pratique, étape souvent non guidée

Ingrédient 3 : Reconstruction

Partie de difficulté très variable !

Ingrédient 3 : Reconstruction

Partie de difficulté très variable !

Étapes

1. ajout d'informations supplémentaires à l'algo. précédent
2. *redescente* depuis la solution générale vers les instances petites

Ingrédient 3 : Reconstruction

Partie de difficulté très variable !

Étapes

1. ajout d'informations supplémentaires à l'algo. précédent
2. *redescente* depuis la solution générale vers les instances petites

Rendu de monnaie

1. renvoyer L et non seulement L_s
2. descente depuis L_s

Ingrédient 3 : Reconstruction

Partie de difficulté très variable !

Étapes

1. ajout d'informations supplémentaires à l'algo. précédent
2. *redescente* depuis la solution générale vers les instances petites

Rendu de monnaie

1. renvoyer L et non seulement L_s
2. descente depuis L_s

En pratique, étape pas toujours effectuée

Problématique de la mémoire

- Dans RENDU DE MONNAIE, tableau L de taille s
  - Complexité *en espace* $O(s)$
  - Si $s = 2^{25}$: env. 1Mo de mémoire

Problématique de la mémoire

- Dans RENDU DE MONNAIE, tableau L de taille s
  - Complexité *en espace* $O(s)$
  - Si $s = 2^{25}$: env. 1Mo de mémoire
- En général : la prog. dyn. est **gourmande en mémoire**
  - parfois le *réactif limitant*
  - souvent : possible de limiter l'espace mémoire si pas de reconstruction
  - plus de détails dans l'exemple suivant

Conclusion sur la programmation dynamique

Comparaison avec les algorithmes gloutons :

- Algo. glouton : cas particulier de programmation dynamique avec un seul sous-problème
- Souvent pas suffisant

Conclusion sur la programmation dynamique

Comparaison avec les algorithmes gloutons :

- Algo. glouton : cas particulier de programmation dynamique avec un seul sous-problème
- Souvent pas suffisant

Les algorithmes gloutons fonctionnent rarement !

Conclusion sur la programmation dynamique

Comparaison avec les algorithmes gloutons :

- Algo. glouton : cas particulier de programmation dynamique avec un seul sous-problème
- Souvent pas suffisant

Les algorithmes gloutons fonctionnent rarement !

D'où vient ce nom ?

- Bellman (1940) : travaux en optimisation mathématique
  - Programmation : planification, ordonnancement
  - Dynamique : « *it's impossible to use the word dynamic in a pejorative sense* »
- Origine du mot peu claire : référence à la programmation linéaire, et/ou problèmes de financements (*cf* Wikipédia)

Corrigeons les erreurs

AGORRYTNES

Corrigeons les erreurs

Corrigeons les erreurs

ALGORITHMES

Corrigeons les erreurs

À quelle **distance** se trouve-t-on du mot correct ?

Corrigeons les erreurs

À quelle **distance** se trouve-t-on du mot correct ?

Distance 5

A	_	G	O	R	R	Y	T	_	N	E	S
A	L	G	O	_	R	I	T	H	M	E	S

La distance d'édition

Définition

La **distance d'édition** (ou de **Levenshtein**, ou d'**Ulam**) entre deux mots u et v est le **nombre minimal de désaccords** dans un **alignement** de u et v

A	_	G	O	R	R	Y	T	_	N	E	S
A	L	G	O	_	R	I	T	H	M	E	S

La distance d'édition

Définition

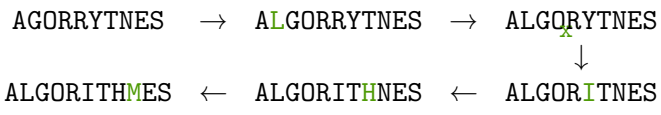
La **distance d'édition** (ou de **Levenshtein**, ou d'**Ulam**) entre deux mots u et v est le **nombre minimal de désaccords** dans un **alignement** de u et v

A	_	G	O	R	R	Y	T	_	N	E	S
A	L	G	O	_	R	I	T	H	M	E	S

Définition alternative

Longueur de la **plus courte suite de transformations** pour passer de u à v, avec les transformations suivantes :

- **insertion** d'une nouvelle lettre
- **suppression** d'une lettre
- **remplacement** d'une lettre par une autre



Définition du problème

Entrée Deux mots u et v sur un alphabet
(mot : chaîne de caractère ou tableau de caractères ou ...)

Sortie 1 La distance d'édition entre u et v

Sortie 2 Un alignement optimal de u et v

Définition du problème

Entrée Deux mots u et v sur un alphabet
(mot : chaîne de caractère ou tableau de caractères ou ...)

Sortie 1 La distance d'édition entre u et v

Sortie 2 Un alignement optimal de u et v

Utilité

- Orthographe :
  - Correcteur orthographique
  - Reconnaissance optique de caractères
- Linguistique (proximité de langues)
- Bioinformatique :
  - similarité de séquences ADN
  - similarité d'arbres phylogénétiques
- ...

Formalisation

- $u_{[0,i[} = u_0 u_1 \cdots u_{i-1}$ et $v_{[0,j[} = v_0 v_1 \cdots v_{j-1}$
- $\text{edit}(i, j)$: distance entre $u_{[0,i[}$ et $v_{[0,j[}$
  - $\text{edit}(i, 0) = i$
  - $\text{edit}(0, j) = j$

Formalisation

- $u_{[0,i[} = u_0 u_1 \cdots u_{i-1}$ et $v_{[0,j[} = v_0 v_1 \cdots v_{j-1}$
- $\text{edit}(i, j)$: distance entre $u_{[0,i[}$ et $v_{[0,j[}$
  - $\text{edit}(i, 0) = i$
  - $\text{edit}(0, j) = j$
- si $|u| = m$ et $|v| = n$, on cherche $\text{edit}(m, n)$

Formalisation

- $u_{[0,i[} = u_0 u_1 \cdots u_{i-1}$ et $v_{[0,j[} = v_0 v_1 \cdots v_{j-1}$
- $\text{edit}(i, j)$: distance entre $u_{[0,i[}$ et $v_{[0,j[}$
  - $\text{edit}(i, 0) = i$
  - $\text{edit}(0, j) = j$

- si $|u| = m$ et $|v| = n$, on cherche $\text{edit}(m, n)$

Trois alignements possibles

$u_{[0,i-1[}$	u_{i-1}		$u_{[0,i[}$	-		$u_{[0,i-1[}$	u_{i-1}
$v_{[0,j[}$	-		$v_{[0,j-1[}$	v_{j-1}		$v_{[0,j-1[}$	v_{j-1}

Formalisation

- $u_{[0,i[} = u_0 u_1 \cdots u_{i-1}$ et $v_{[0,j[} = v_0 v_1 \cdots v_{j-1}$
- $\text{edit}(i, j)$: distance entre $u_{[0,i[}$ et $v_{[0,j[}$
  - $\text{edit}(i, 0) = i$
  - $\text{edit}(0, j) = j$

- si $|u| = m$ et $|v| = n$, on cherche $\text{edit}(m, n)$

Trois alignements possibles

$u_{[0,i-1[}$	u_{i-1}		$u_{[0,i[}$	-		$u_{[0,i-1[}$	u_{i-1}
$v_{[0,j[}$	-		$v_{[0,j-1[}$	v_{j-1}		$v_{[0,j-1[}$	v_{j-1}

Alignements AGORR et ALGOR

AGOR	R		AGORR	_		AGOR	R		
ALGOR	_		ALGO	R		ALGO	R		
1	1		3	1		2	0	$\rightarrow$	2

Formule récursive

Lemme

$$\text{edit}(i,j) = \min \begin{cases} \text{edit}(i-1,j)+1 \\ \text{edit}(i,j-1)+1 \\ \text{edit}(i-1,j-1)+\mathbf{1}_{[u_{i-1}\neq v_{j-1}]} \end{cases}$$

où $\mathbf{1}_{[u_{i-1}\neq v_{j-1}]} = 1$ *si* $u_{i-1} \neq v_{j-1}$, 0 *sinon*

Formule récursive

Lemme

$$\text{edit}(i,j) = \min \begin{cases} \text{edit}(i-1,j)+1 \\ \text{edit}(i,j-1)+1 \\ \text{edit}(i-1,j-1)+\mathbf{1}_{[u_{i-1}\neq v_{j-1}]} \end{cases}$$

où $\mathbf{1}_{[u_{i-1}\neq v_{j-1}]} = 1$ *si* $u_{i-1} \neq v_{j-1}$, 0 *sinon*

Preuve :

Formule récursive

Lemme

$$\text{edit}(i,j) = \min \begin{cases} \text{edit}(i-1,j)+1 \\ \text{edit}(i,j-1)+1 \\ \text{edit}(i-1,j-1)+\mathbf{1}_{[u_{i-1}\neq v_{j-1}]} \end{cases}$$

où $\mathbf{1}_{[u_{i-1}\neq v_{j-1}]} = 1$ *si* $u_{i-1} \neq v_{j-1}$, 0 *sinon*

Preuve :

- $\text{edit}(i,j) \leq \text{edit}(i-1,j)+1$:

$u_{[0,i-1[}$	u_{i-1}
$v_{[0,j[}$	-

Formule récursive

Lemme

$$\text{edit}(i,j) = \min \begin{cases} \text{edit}(i-1,j)+1 \\ \text{edit}(i,j-1)+1 \\ \text{edit}(i-1,j-1)+\mathbf{1}_{[u_{i-1}\neq v_{j-1}]} \end{cases}$$

où $\mathbf{1}_{[u_{i-1}\neq v_{j-1}]} = 1$ *si* $u_{i-1} \neq v_{j-1}$, 0 *sinon*

Preuve :

- $\text{edit}(i,j) \leq \text{edit}(i-1,j)+1$:

$u_{[0,i-1[}$	u_{i-1}
$v_{[0,j[}$	-

- $\text{edit}(i,j) \leq \text{edit}(i,j-1)+1$:

$u_{[0,i[}$	-
$v_{[0,j-1[}$	v_{j-1}

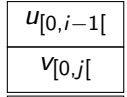
Formule récursive

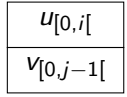
Lemme

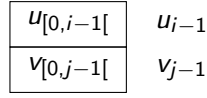
$$\text{edit}(i,j) = \min \begin{cases} \text{edit}(i-1,j)+1 \\ \text{edit}(i,j-1)+1 \\ \text{edit}(i-1,j-1)+\mathbf{1}_{[u_{i-1}\neq v_{j-1}]} \end{cases}$$

où $\mathbf{1}_{[u_{i-1}\neq v_{j-1}]} = 1$ *si* $u_{i-1} \neq v_{j-1}$, 0 *sinon*

Preuve :

- $\text{edit}(i,j) \leq \text{edit}(i-1,j)+1$: 

- $\text{edit}(i,j) \leq \text{edit}(i,j-1)+1$: 

- $\text{edit}(i,j) \leq \text{edit}(i-1,j-1)+\mathbf{1}_{[u_{i-1}\neq v_{j-1}]}$: 

Formule récursive

Lemme

$$\text{edit}(i,j) = \min \begin{cases} \text{edit}(i-1,j)+1 \\ \text{edit}(i,j-1)+1 \\ \text{edit}(i-1,j-1)+\mathbf{1}_{[u_{i-1}\neq v_{j-1}]} \end{cases}$$

où $\mathbf{1}_{[u_{i-1}\neq v_{j-1}]} = 1$ *si* $u_{i-1} \neq v_{j-1}$, 0 *sinon*

Preuve :

- $\text{edit}(i,j) \leq \text{edit}(i-1,j)+1$: $\begin{array}{|c|}\hline u_{[0,i-1[} \\ \hline v_{[0,j[} \\ \hline \end{array}$ $\begin{array}{c} u_{i-1} \\ - \end{array}$
- $\text{edit}(i,j) \leq \text{edit}(i,j-1)+1$: $\begin{array}{|c|}\hline u_{[0,i[} \\ \hline v_{[0,j-1[} \\ \hline \end{array}$ $\begin{array}{c} - \\ v_{j-1} \end{array}$
- $\text{edit}(i,j) \leq \text{edit}(i-1,j-1)+\mathbf{1}_{[u_{i-1}\neq v_{j-1}]}$: $\begin{array}{|c|}\hline u_{[0,i-1[} \\ \hline v_{[0,j-1[} \\ \hline \end{array}$ $\begin{array}{c} u_{i-1} \\ v_{j-1} \end{array}$

$$\implies \text{edit}(i,j) \leq \min\{\dots\}$$

Formule récursive

Lemme

$$\text{edit}(i,j) = \min \begin{cases} \text{edit}(i-1,j)+1 \\ \text{edit}(i,j-1)+1 \\ \text{edit}(i-1,j-1)+\mathbf{1}_{[u_{i-1}\neq v_{j-1}]} \end{cases}$$

où $\mathbf{1}_{[u_{i-1}\neq v_{j-1}]} = 1$ *si* $u_{i-1} \neq v_{j-1}$, 0 *sinon*

Preuve Si $\text{edit}(i,j) = d$,

Formule récursive

Lemme

$$\text{edit}(i,j) = \min \begin{cases} \text{edit}(i-1,j)+1 \\ \text{edit}(i,j-1)+1 \\ \text{edit}(i-1,j-1)+\mathbf{1}_{[u_{i-1}\neq v_{j-1}]} \end{cases}$$

où $\mathbf{1}_{[u_{i-1}\neq v_{j-1}]} = 1$ *si* $u_{i-1} \neq v_{j-1}$, 0 *sinon*

Preuve Si $\text{edit}(i,j) = d$,

- $u_{i-1} = v_{j-1} \implies \text{edit}(i-1,j-1) = d$

Formule récursive

Lemme

$$\text{edit}(i,j) = \min \begin{cases} \text{edit}(i-1,j)+1 \\ \text{edit}(i,j-1)+1 \\ \text{edit}(i-1,j-1)+\mathbf{1}_{[u_{i-1}\neq v_{j-1}]} \end{cases}$$

où $\mathbf{1}_{[u_{i-1}\neq v_{j-1}]} = 1$ *si* $u_{i-1} \neq v_{j-1}$, 0 *sinon*

Preuve Si $\text{edit}(i,j) = d$,

- $u_{i-1} = v_{j-1} \implies \text{edit}(i-1,j-1) = d$
- $u_{i-1} \neq v_{j-1}$:
  - $\text{edit}(i-1,j-1) \geq d-1$
  - $\text{edit}(i,j-1) \geq d-1$
  - $\text{edit}(i-1,j) \geq d-1$

Formule récursive

Lemme

$$\text{edit}(i,j) = \min \begin{cases} \text{edit}(i-1,j)+1 \\ \text{edit}(i,j-1)+1 \\ \text{edit}(i-1,j-1)+\mathbf{1}_{[u_{i-1}\neq v_{j-1}]} \end{cases}$$

où $\mathbf{1}_{[u_{i-1}\neq v_{j-1}]} = 1$ *si* $u_{i-1} \neq v_{j-1}$, 0 *sinon*

Preuve Si $\text{edit}(i,j) = d$,

- $u_{i-1} = v_{j-1} \implies \text{edit}(i-1,j-1) = d$
- $u_{i-1} \neq v_{j-1}$:
  - $\text{edit}(i-1,j-1) \geq d-1$
  - $\text{edit}(i,j-1) \geq d-1$
  - $\text{edit}(i-1,j) \geq d-1$

$$\implies \text{edit}(i,j) \geq \min\{\dots\}$$

Algorithme : exemple

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

Algorithme : exemple

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

Algorithme : exemple

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

Algorithme : exemple

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

Algorithme : exemple

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

Algorithme : exemple

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

Algorithme : exemple

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

Algorithme : exemple

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

Algorithme : exemple

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

Algorithme : exemple

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

Algorithme : exemple

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

Algorithme : exemple

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

Algorithme : exemple

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

Algorithme : exemple

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

Algorithme : exemple

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

Algorithme : exemple

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

L'algorithme

EDITION(u, v)

1. $(m, n) \leftarrow$ tailles de u et v
2. $E \leftarrow$ tableau de dimensions $m+1$ par $n+1$
3. Pour $i = 0$ à m : $E_{[i,0]} \leftarrow i$ `# cas de`
4. Pour $j = 0$ à n : $E_{[0,j]} \leftarrow j$ `# base`
5. Pour $i = 1$ à m :
6. Pour $j = 1$ à n :
7. $E_{[i,j]} \leftarrow \min(E_{[i-1,j]} + 1, E_{[i,j-1]} + 1, E_{[i-1,j-1]} + \mathbf{1}_{[u_{i-1} \neq v_{j-1}]})$
8. Renvoyer $E_{[m,n]}$

L'algorithme

EDITION(u, v)

1. $(m, n) \leftarrow$ tailles de u et v
2. $E \leftarrow$ tableau de dimensions $m+1$ par $n+1$
3. Pour $i = 0$ à m : $E_{[i,0]} \leftarrow i$ `# cas de`
4. Pour $j = 0$ à n : $E_{[0,j]} \leftarrow j$ `# base`
5. Pour $i = 1$ à m :
6. Pour $j = 1$ à n :
7. $E_{[i,j]} \leftarrow \min(E_{[i-1,j]} + 1, E_{[i,j-1]} + 1, E_{[i-1,j-1]} + \mathbf{1}_{[u_{i-1} \neq v_{j-1}]})$
8. Renvoyer $E_{[m,n]}$

Lemme

L'algorithme EDITION *renvoie la distance entre u et v en temps $O(mn)$, en utilisant un espace mémoire $O(mn)$.*

Version efficace en mémoire

Pour remplir la ligne i, on n'a besoin que des lignes i et $i-1$!

Version efficace en mémoire

Pour remplir la ligne i, on n'a besoin que des lignes i et $i-1$!

EditionMinMemoire(u, v)

1. $(m, n) \leftarrow$ tailles de u et v `# Hyp. : m ≥ n`
2. $P \leftarrow$ tableau de taille $n+1$ `# Ligne précédente`
   $C \leftarrow$ tableau de taille $n+1$ `# Ligne courante`
3. Pour $j = 0$ à n : $P_j = j$ `# cas de base`
4. Pour $i = 1$ à m :
5. $\quad C_0 \leftarrow i$ `# cas de base`
6. $\quad$ Pour $j = 1$ à n : $C_j \leftarrow \min(P_j + 1, C_{j-1} + 1, P_{j-1} + \mathbf{1}_{[u_{i-1} - 1 \neq v_{j-1}]})$
7. $\quad$ Pour $j = 0$ à n : $P_j \leftarrow C_j$
8. Renvoyer C_n

Version efficace en mémoire

Pour remplir la ligne i, on n'a besoin que des lignes i et $i-1$!

EditionMinMemoire(u, v)

1. $(m, n) \leftarrow$ tailles de u et v `# Hyp. : m ≥ n`
2. $P \leftarrow$ tableau de taille $n+1$ `# Ligne précédente`
   $C \leftarrow$ tableau de taille $n+1$ `# Ligne courante`
3. Pour $j = 0$ à n : $P_j = j$ `# cas de base`
4. Pour $i = 1$ à m :
5. $\quad C_0 \leftarrow i$ `# cas de base`
6. $\quad$ Pour $j = 1$ à n : $C_j \leftarrow \min(P_j + 1, C_{j-1} + 1, P_{j-1} + \mathbf{1}_{[u_{i-1} - 1 \neq v_{j-1}]})$
7. $\quad$ Pour $j = 0$ à n : $P_j \leftarrow C_j$
8. Renvoyer C_n

Lemme

EditionMinMemoire *calcule la distance entre u et v en temps $O(mn)$, en utilisant un* ***espace mémoire $O(n)$***.

Reconstruction : exemple

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

Reconstruction : exemple

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

Reconstruction : exemple

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

Reconstruction : exemple

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

Reconstruction : exemple

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

Reconstruction : exemple

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

Reconstruction : exemple

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

Reconstruction : exemple

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

Reconstruction : exemple

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

Reconstruction : exemple

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

Reconstruction : exemple

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

Reconstruction : exemple

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

Reconstruction : exemple

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

A	_	G	O	R	R	Y	T	_	N	E
A	L	G	O	R	_	I	T	H	M	E

Algorithme de reconstruction

ALIGNEMENT(u, v, E)

1. $(i, j) \leftarrow$ tailles de u et v
2. Tant que $i > 0$ et $j > 0$:
3. Si $E_{[i,j]} = E_{[i-1,j-1]}$ et $u_{i-1} = v_{j-1}$: # u_{i-1}/v_{j-1}
4. $(i, j) \leftarrow (i - 1, j - 1)$
5. Sinon si $E_{[i,j]} = E_{[i-1,j-1]} + 1$ et $u_{i-1} \neq v_{j-1}$: # u_{i-1}/v_{j-1}
6. $(i, j) \leftarrow (i - 1, j - 1)$
7. Sinon si $E_{[i,j]} = E_{[i-1,j]} + 1$: # $u_{i-1}/\_$
8. Insérer _ en $j^{\text{ème}}$ position dans v
9. $i \leftarrow i - 1$
10. Sinon : # si $E_{[i,j]} = E_{[i,j-1]} + 1$: # $\_/v_{j-1}$
11. Insérer _ en $i^{\text{ème}}$ position dans u
12. $j \leftarrow j - 1$
13. Insérer j symboles _ en tête de u # l'un ou
14. Insérer i symboles _ en tête de v # l'autre
15. Renvoyer u et v

Correction et complexité

Lemme

L'algorithme ALIGNEMENT *aligne les mots u et v avec* edit(m, n) *désaccords, en temps $O(m + n)$.*

ALIGNEMENT(u, v, E)

1. $(i, j) \leftarrow$ tailles de u et v
2. Tant que $i > 0$ et $j > 0$:
3. Si $E_{[i,j]} = E_{[i-1,j-1]}$ et $u_{i-1} = v_{j-1}$:
4. $(i, j) \leftarrow (i - 1, j - 1)$
5. Sinon si $E_{[i,j]} = E_{[i-1,j-1]} + 1$:
6. $(i, j) \leftarrow (i - 1, j - 1)$
7. Sinon si $E_{[i,j]} = E_{[i-1,j]} + 1$:
8. Insérer _ en $j^{\text{ème}}$ position dans v
9. $i \leftarrow i - 1$
10. Sinon :
11. Insérer _ en $i^{\text{ème}}$ position dans u
12. $j \leftarrow j - 1$
13. Insérer j symboles _ en tête de u
14. Insérer i symboles _ en tête de v
15. Renvoyer u et v

Correction et complexité

Lemme

L'algorithme ALIGNEMENT *aligne les mots u et v avec* edit(m, n) *désaccords, en temps $O(m+n)$.*

```
ALIGNEMENT(u, v, E)
1.  (i, j) ← tailles de u et v
2.  Tant que i > 0 et j > 0 :
3.      Si E[i,j] = E[i−1,j−1] et u_{i−1} = v_{j−1} :
4.          (i, j) ← (i − 1, j − 1)
5.      Sinon si E[i,j] = E[i−1,j−1] + 1 :
6.          (i, j) ← (i − 1, j − 1)
7.      Sinon si E[i,j] = E[i−1,j] + 1 :
8.          Insérer _ en j^ème position dans v
9.          i ← i − 1
10.     Sinon :
11.         Insérer _ en i^ème position dans u
12.         j ← j − 1
13. Insérer j symboles _ en tête de u
14. Insérer i symboles _ en tête de v
15. Renvoyer u et v
```

Preuve de complexité : à chaque tour, $i + j$ diminue de ≥ 1

Correction et complexité

Lemme

L'algorithme ALIGNEMENT *aligne les mots u et v avec* edit(m, n) *désaccords, en temps $O(m+n)$.*

```
ALIGNEMENT(u, v, E)
1.  (i, j) ← tailles de u et v
2.  Tant que i > 0 et j > 0 :
3.      Si E[i,j] = E[i−1,j−1] et u_{i−1} = v_{j−1} :
4.          (i, j) ← (i − 1, j − 1)
5.      Sinon si E[i,j] = E[i−1,j−1] + 1 :
6.          (i, j) ← (i − 1, j − 1)
7.      Sinon si E[i,j] = E[i−1,j] + 1 :
8.          Insérer _ en j^ème position dans v
9.          i ← i − 1
10.     Sinon :
11.         Insérer _ en i^ème position dans u
12.         j ← j − 1
13. Insérer j symboles _ en tête de u
14. Insérer i symboles _ en tête de v
15. Renvoyer u et v
```

Preuve de complexité : à chaque tour, $i+j$ diminue de ≥ 1

Preuve de correction : « en entrant dans la boucle, $u_{[i,m[}$ et $v_{[j,n[}$ sont *alignés* de manière optimale »

Conclusion

Théorème

La distance d'édition entre deux mots u et v de tailles respectives m et n peut être calculée en temps et espace $O(mn)$. Leur alignement peut être calculé en temps $O(m+n)$ supplémentaire.
Pour ne calculer que la distance d'édition, un espace $O(\min(m,n))$ est suffisant.

Conclusion

Théorème

La distance d'édition entre deux mots u et v de tailles respectives m et n peut être calculée en temps et espace $O(mn)$. Leur alignement peut être calculé en temps $O(m+n)$ supplémentaire.
Pour ne calculer que la distance d'édition, un espace $O(\min(m,n))$ est suffisant.

Peut-on faire mieux qu'un temps $O(mn)$?

Conclusion

Théorème

La distance d'édition entre deux mots u et v de tailles respectives m et n peut être calculée en temps et espace $O(mn)$. Leur alignement peut être calculé en temps $O(m+n)$ supplémentaire.
Pour ne calculer que la distance d'édition, un espace $O(\min(m,n))$ est suffisant.

Peut-on faire mieux qu'un temps $O(mn)$?

$\rightsquigarrow$ un peu...

Conclusion

Théorème

La distance d'édition entre deux mots u et v de tailles respectives m et n peut être calculée en temps et espace $O(mn)$. Leur alignement peut être calculé en temps $O(m+n)$ supplémentaire.
Pour ne calculer que la distance d'édition, un espace $O(\min(m,n))$ est suffisant.

Peut-on faire mieux qu'un temps $O(mn)$?

$\rightsquigarrow$ un peu...

Peut-on faire beaucoup mieux que $O(mn)$?

Conclusion

Théorème

La distance d'édition entre deux mots u et v de tailles respectives m et n peut être calculée en temps et espace $O(mn)$. Leur alignement peut être calculé en temps $O(m+n)$ supplémentaire.
Pour ne calculer que la distance d'édition, un espace $O(\min(m,n))$ est suffisant.

Peut-on faire mieux qu'un temps $O(mn)$?

$\rightsquigarrow$ un peu...

Peut-on faire beaucoup mieux que $O(mn)$?

$\rightsquigarrow$ sans doute pas (si on veut le résultat exact)

Conclusion

Théorème

La distance d'édition entre deux mots u et v de tailles respectives m et n peut être calculée en temps et espace $O(mn)$. Leur alignement peut être calculé en temps $O(m+n)$ supplémentaire.
Pour ne calculer que la distance d'édition, un espace $O(\min(m,n))$ est suffisant.

Peut-on faire mieux qu'un temps $O(mn)$?

$\rightsquigarrow$ un peu...

Peut-on faire beaucoup mieux que $O(mn)$?

$\rightsquigarrow$ sans doute pas (si on veut le résultat exact)

- Problème très important en pratique... et en théorie !
- Beaucoup de résultats très récents (2021 !) sur le sujet