

# Chapitre 5

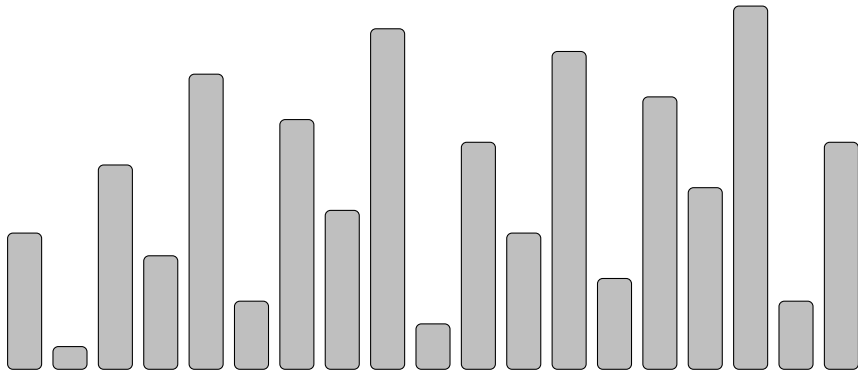
## Programmation dynamique

HLIN401 : Algorithmique et complexité

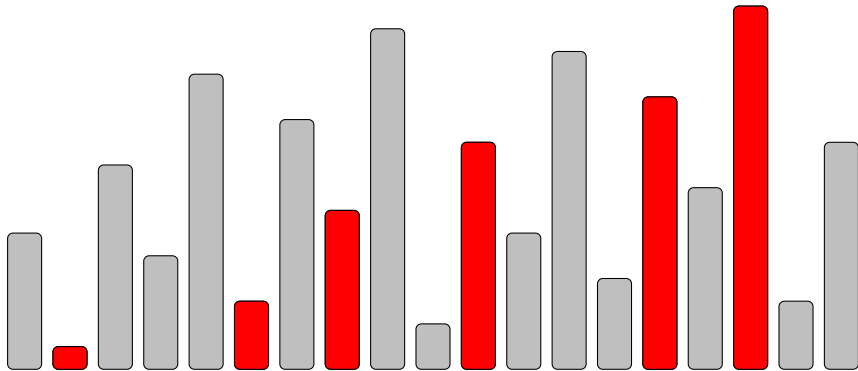
L2 Informatique  
Université de Montpellier  
2020 – 2021

1. Premier exemple : plus longue sous-suite croissante
2. Qu'est-ce que la programmation dynamique ?
3. Deuxième exemple : choix de cours, le retour
4. Troisième exemple : la distance d'édition
5. Quatrième exemple : calcul de plus courts chemins
6. Bonus : le voyageur de commerce

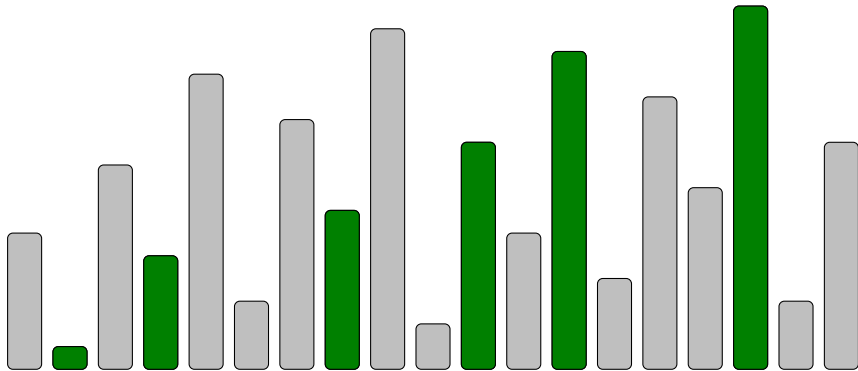
## Plus longue sous-suite croissante



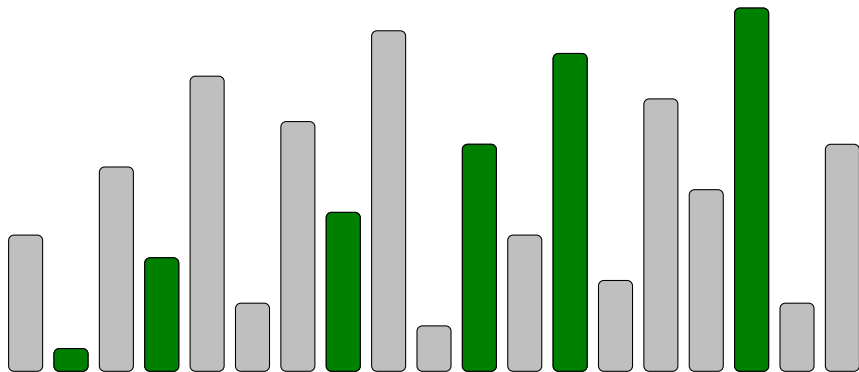
## Plus longue sous-suite croissante



## Plus longue sous-suite croissante



## Plus longue sous-suite croissante



### Définition

Une **plus longue sous-suite croissante (PLSSC)** d'un tableau  $T$  d'entiers est une suite la plus grande possible d'indices  $0 \leq i_1 < i_2 < \dots < i_k \leq n - 1$  telle que  $T[i_1] \leq T[i_2] \leq \dots \leq T[i_k]$ .

# Formalisation du problème

**Entrée** Un tableau  $T$  de  $n$  entiers

**Sortie 1** Une PLSSC de  $T$

# Formalisation du problème

**Entrée** Un tableau  $T$  de  $n$  entiers

**Sortie 1** Une PLSSC de  $T$

**Sortie 2** La longueur d'une PLSSC de  $T$



# Formalisation du problème

**Entrée** Un tableau  $T$  de  $n$  entiers

**Sortie 1** Une PLSSC de  $T$

**Sortie 2** La longueur d'une PLSSC de  $T$

## Exemple précédent

**Entrée**  $T = [6, 1, 9, 5, 13, 3, 11, 7, 15, 2, 10, 6, 14, 4, 12, 8, 16, 3, 10]$

**Sortie 1**  $[6, 1, 9, 5, 13, 3, 11, 7, 15, 2, 10, 6, 14, 4, 12, 8, 16, 3, 10]$

ou  $[6, 1, 9, 5, 13, 3, 11, 7, 15, 2, 10, 6, 14, 4, 12, 8, 16, 3, 10]$

ou ...

**Sortie 2** 6

# Formalisation du problème

**Entrée** Un tableau  $T$  de  $n$  entiers

**Sortie 1** Une PLSSC de  $T$

**Sortie 2** La longueur d'une PLSSC de  $T$

## Exemple précédent

**Entrée**  $T = [6, 1, 9, 5, 13, 3, 11, 7, 15, 2, 10, 6, 14, 4, 12, 8, 16, 3, 10]$

**Sortie 1**  $[6, 1, 9, 5, 13, 3, 11, 7, 15, 2, 10, 6, 14, 4, 12, 8, 16, 3, 10]$

ou  $[6, 1, 9, 5, 13, 3, 11, 7, 15, 2, 10, 6, 14, 4, 12, 8, 16, 3, 10]$

ou ...

**Sortie 2** 6

► Algo. naïf : considérer toutes les sous-suites  $\rightsquigarrow O(2^n)$

# Formalisation du problème

**Entrée** Un tableau  $T$  de  $n$  entiers

**Sortie 1** Une PLSSC de  $T$

**Sortie 2** La longueur d'une PLSSC de  $T$

## Exemple précédent

**Entrée**  $T = [6, 1, 9, 5, 13, 3, 11, 7, 15, 2, 10, 6, 14, 4, 12, 8, 16, 3, 10]$

**Sortie 1**  $[6, 1, 9, 5, 13, 3, 11, 7, 15, 2, 10, 6, 14, 4, 12, 8, 16, 3, 10]$

ou  $[6, 1, 9, 5, 13, 3, 11, 7, 15, 2, 10, 6, 14, 4, 12, 8, 16, 3, 10]$

ou ...

**Sortie 2** 6

► Algo. naïf : considérer toutes les sous-suites  $\rightsquigarrow O(2^n)$

**Algorithme de complexité polynomiale ?**

## Formule récursive pour PLSSC

- ▶  $\ell(T)$  : longueur des PLSSC de  $T$
- ▶  $\ell_i$  : longueur des PLSSC de  $T$  **finissant en case**  $T_{[i]}$

## Formule récursive pour PLSSC

- ▶  $\ell(T)$  : longueur des PLSSC de  $T$
- ▶  $\ell_i$  : longueur des PLSSC de  $T$  **finissant en case**  $T_{[i]}$

Remarque

$$\ell(T) = \max_{0 \leq i < n} \ell_i$$

## Formule récursive pour PLSSC

- ▶  $\ell(T)$  : longueur des PLSSC de  $T$
- ▶  $\ell_i$  : longueur des PLSSC de  $T$  **finissant en case**  $T_{[i]}$

Remarque

$$\ell(T) = \max_{0 \leq i < n} \ell_i$$

Lemme

$$\ell_i = \begin{cases} 1 & \text{si } i = 0 \\ 1 + \max\{\ell_j : j < i \text{ tels que } T_{[j]} \leq T_{[i]}\} & \text{pour } 1 \leq i < n \end{cases}$$

## Formule récursive pour PLSSC

- ▶  $\ell(T)$  : longueur des PLSSC de  $T$
- ▶  $\ell_i$  : longueur des PLSSC de  $T$  **finissant en case**  $T_{[i]}$

Remarque

$$\ell(T) = \max_{0 \leq i < n} \ell_i$$

Lemme

$$\ell_i = \begin{cases} 1 & \text{si } i = 0 \\ 1 + \max\{\ell_j : j < i \text{ tels que } T_{[j]} \leq T_{[i]}\} & \text{pour } 1 \leq i < n \end{cases}$$

Preuve On montre deux inégalités :

- ≥ si  $\max = \ell_{j_m}$ , il existe une SSC de lgr  $\ell_{j_m} : T_{[j_0]} \leq \dots \leq T_{[j_m]}$  ; donc  $T_{[j_0]} \leq \dots \leq T_{[j_m]} \leq T_{[i]}$  est une SSC de lgr  $\ell_{j_m} + 1$

## Formule récursive pour PLSSC

- ▶  $\ell(T)$  : longueur des PLSSC de  $T$
- ▶  $\ell_i$  : longueur des PLSSC de  $T$  **finissant en case**  $T_{[i]}$

Remarque

$$\ell(T) = \max_{0 \leq i < n} \ell_i$$

Lemme

$$\ell_i = \begin{cases} 1 & \text{si } i = 0 \\ 1 + \max\{\ell_j : j < i \text{ tels que } T_{[j]} \leq T_{[i]}\} & \text{pour } 1 \leq i < n \end{cases}$$

Preuve On montre deux inégalités :

- $\geq$  si  $\max = \ell_{j_m}$ , il existe une SSC de lgr  $\ell_{j_m} : T_{[j_0]} \leq \dots \leq T_{[j_m]}$  ; donc  $T_{[j_0]} \leq \dots \leq T_{[j_m]} \leq T_{[i]}$  est une SSC de lgr  $\ell_{j_m} + 1$
- $\leq$  il existe une SSC de lgr  $\ell_i : T_{[j_0]} \leq \dots \leq T_{[j]} \leq T_{[i]}$  ; alors  $T_{[j_0]} \leq \dots \leq T_{[j]}$  est une SSC de lgr  $\ell_i - 1 : \ell_j \geq \ell_i - 1$  ; donc  $\max_{j < i, T_{[j]} \leq T_{[i]}} \ell_j \geq \ell_i - 1$



## Mauvaise idée : algorithme récursif !

$$\ell_i = \begin{cases} 1 & \text{si } i = 0 \\ 1 + \max\{\ell_j : j < i \text{ tels que } T_{[j]} \leq T_{[i]}\} & \text{pour } 1 \leq i < n \end{cases}$$

## Mauvaise idée : algorithme récursif !

$$\ell_i = \begin{cases} 1 & \text{si } i = 0 \\ 1 + \max\{\ell_j : j < i \text{ tels que } T_{[j]} \leq T_{[i]}\} & \text{pour } 1 \leq i < n \end{cases}$$

**Algorithme** :  $L(T, i)$

**si**  $i = 0$  : **renvoyer** 1

$\max \leftarrow 0$

**pour**  $j = 0$  à  $i - 1$  :

**si**  $T_{[j]} \leq T_{[i]}$  :

$L \leftarrow L(T, j)$

**si**  $L > \max$  :  $\max \leftarrow L$

**renvoyer**  $1 + \max$

## Mauvaise idée : algorithme récursif !

$$\ell_i = \begin{cases} 1 & \text{si } i = 0 \\ 1 + \max\{\ell_j : j < i \text{ tels que } T_{[j]} \leq T_{[i]}\} & \text{pour } 1 \leq i < n \end{cases}$$

- ▶  $t_L(i) \leq \sum_{j < i} t_L(j)$   
 $\rightsquigarrow t_L(i) = O(2^i)$

**Algorithme :  $L(T, i)$**

**si  $i = 0$  : renvoyer 1**

$$\text{max} \leftarrow 0$$

**pour**  $j = 0$  à  $i - 1$  :

**si**  $T_{[j]} \leq T_{[i]}$  :

$$L \leftarrow L(T, j)$$

**si**  $L > \text{max}$  :  $\text{max} \leftarrow L$

**renvoyer**  $1 + \max$

## Mauvaise idée : algorithme récursif !

$$\ell_i = \begin{cases} 1 & \text{si } i = 0 \\ 1 + \max\{\ell_j : j < i \text{ tels que } T_{[j]} \leq T_{[i]}\} & \text{pour } 1 \leq i < n \end{cases}$$

**Algorithme** :  $L(T, i)$

**si**  $i = 0$  : renvoyer 1

$\max \leftarrow 0$

**pour**  $j = 0$  à  $i - 1$  :

**si**  $T_{[j]} \leq T_{[i]}$  :

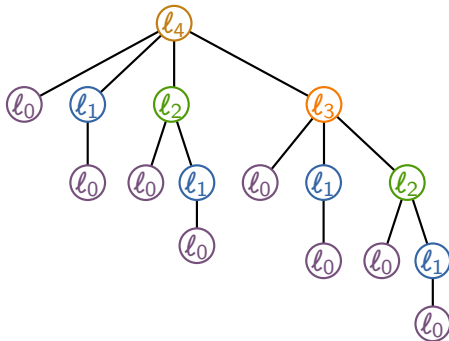
$L \leftarrow L(T, j)$

**si**  $L > \max$  :  $\max \leftarrow L$

**renvoyer**  $1 + \max$

►  $t_L(i) \leq \sum_{j < i} t_L(j)$

$\rightsquigarrow t_L(i) = O(2^i)$



## Mauvaise idée : algorithme récursif !

$$\ell_i = \begin{cases} 1 & \text{si } i = 0 \\ 1 + \max\{\ell_j : j < i \text{ tels que } T_{[j]} \leq T_{[i]}\} & \text{pour } 1 \leq i < n \end{cases}$$

**Algorithme :  $L(T, i)$**

**si**  $i = 0$  : renvoyer 1

$\max \leftarrow 0$

**pour**  $j = 0$  à  $i - 1$  :

**si**  $T_{[j]} \leq T_{[i]}$  :

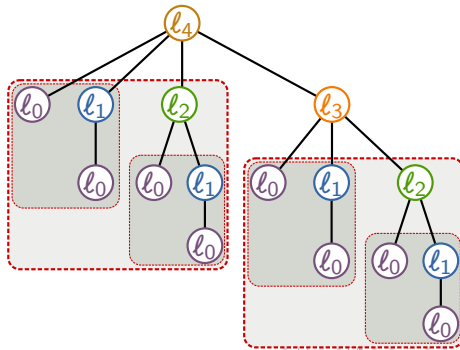
$L \leftarrow L(T, j)$

**si**  $L > \max$  :  $\max \leftarrow L$

**renvoyer**  $1 + \max$

►  $t_L(i) \leq \sum_{j < i} t_L(j)$

$\rightsquigarrow t_L(i) = O(2^i)$



## Idée moyenne : mémorisation

- Retenir les valeurs calculées



## Idée moyenne : mémorisation

- Retenir les valeurs calculées

**Algorithme :** MEMO( $T, i, L$ )

**si**  $i = 0$  : renvoyer 1

$\max \leftarrow 0$

**pour**  $j = 0$  à  $i - 1$  :

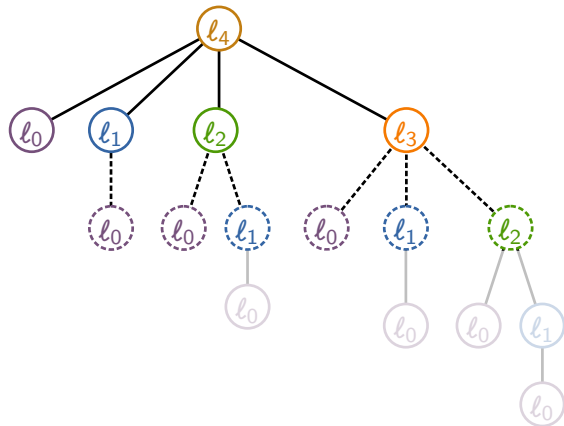
**si**  $T[j] \leq T[i]$  :

**si**  $L[j] = \text{None}$  :

$L[j] \leftarrow \text{MEMO}(T, j, L)$

**si**  $L[j] > \max$  :  $\max \leftarrow L[j]$

**renvoyer**  $1 + \max$







# Algorithme PLSSC

**Algorithme :** PLSSC( $T$ )

$L \leftarrow$  tableau de taille  $n$ , initialisé à 1

$M \leftarrow L_{[0]}$  //  $M = \max_i L_{[i]}$

**pour**  $i = 1$  à  $n - 1$  :

$\max \leftarrow 0$

//  $\max = \max\{L_{[j]} : j < i, T_{[j]} \leq T_{[i]}\}$

**pour**  $j = 0$  à  $i - 1$  :

**si**  $T_{[j]} \leq T_{[i]}$  et  $L_{[j]} > \max$  :

$\max \leftarrow L_{[j]}$

$L_{[i]} \leftarrow 1 + \max$

**si**  $L_{[i]} > M$  :  $M \leftarrow L_{[i]}$

**renvoyer**  $M$

# Correction et complexité

## Théorème

*L'algorithme PLSSC calcule  $\ell(T)$  en temps  $O(n^2)$ .*

**Algorithme :** PLSSC( $T$ )

$L \leftarrow$  tableau de taille  $n$ , initialisé à 1

$M \leftarrow L[0]$

**pour**  $i = 1$  à  $n - 1$  :

$\max \leftarrow 0$

**pour**  $j = 0$  à  $i - 1$  :

**si**  $T[j] \leq T[i]$  et  $L[j] > \max$  :

$\max \leftarrow L[j]$

$L[i] \leftarrow 1 + \max$

**si**  $L[i] > M$  :  $M \leftarrow L[i]$

**renvoyer**  $M$

# Correction et complexité

## Théorème

*L'algorithme PLSSC calcule  $\ell(T)$  en temps  $O(n^2)$ .*

Preuve de correction : utilisation de la formule récursive

Preuve de complexité : double boucle

**Algorithme** : PLSSC( $T$ )

$L \leftarrow$  tableau de taille  $n$ , initialisé à 1

$M \leftarrow L[0]$

**pour**  $i = 1$  à  $n - 1$  :

$\max \leftarrow 0$

**pour**  $j = 0$  à  $i - 1$  :

**si**  $T[j] \leq T[i]$  et  $L[j] > \max$  :

$\max \leftarrow L[j]$

$L[i] \leftarrow 1 + \max$

**si**  $L[i] > M$  :  $M \leftarrow L[i]$

**renvoyer**  $M$

# Correction et complexité

## Théorème

*L'algorithme PLSSC calcule  $\ell(T)$  en temps  $O(n^2)$ .*

**Preuve de correction :** utilisation de la formule récursive

**Preuve de complexité :** double boucle

**Algorithme :** PLSSC( $T$ )

$L \leftarrow$  tableau de taille  $n$ , initialisé à 1

$M \leftarrow L[0]$

**pour**  $i = 1$  à  $n - 1$  :

$\max \leftarrow 0$

**pour**  $j = 0$  à  $i - 1$  :

**si**  $T[j] \leq T[i]$  et  $L[j] > \max$  :

$\max \leftarrow L[j]$

$L[i] \leftarrow 1 + \max$

**si**  $L[i] > M$  :  $M \leftarrow L[i]$

**renvoyer**  $M$

## Comment calculer une PLSSC (en plus de sa longueur) ?

- ▶ Retenir les indices des max
- ▶ Reconstruire *a posteriori*

# Algorithme PLSSC avec reconstruction

**Algorithme** : PLSSC( $T$ )

$L \leftarrow$  tableau de taille  $n$ , initialisé à 1

$Prec \leftarrow$  tableau de taille  $n$ , initialisé à  $-1$

$i_M \leftarrow 0$  // fin de la PLSSC courante

**pour**  $i = 1$  à  $n - 1$  :

$max \leftarrow 0$

**pour**  $j = 0$  à  $i - 1$  :

**si**  $T[j] \leq T[i]$  et  $L[j] > max$  :

$max \leftarrow L[j]$  ;  $Prec[i] \leftarrow j$

$L[i] \leftarrow 1 + max$

**si**  $L[i] > L[i_M]$  :  $i_M \leftarrow i$

**renvoyer**  $L[i_M]$ ,  $i_M$  et  $Prec$

## Algorithme PLSSC avec reconstruction

**Algorithme** : PLSSC( $T$ )

$L \leftarrow$  tableau de taille  $n$ , initialisé à 1

$Prec \leftarrow$  tableau de taille  $n$ , initialisé à  $-1$

$i_M \leftarrow 0$  // fin de la PLSSC courante

**pour**  $i = 1$  à  $n - 1$  :

$max \leftarrow 0$

**pour**  $j = 0$  à  $i - 1$  :

**si**  $T[j] \leq T[i]$  et  $L[j] > max$  :

$max \leftarrow L[j]$  ;  $Prec[i] \leftarrow j$

$L[i] \leftarrow 1 + max$

**si**  $L[i] > L[i_M]$  :  $i_M \leftarrow i$

**renvoyer**  $L[i_M]$ ,  $i_M$  et  $Prec$

**Algorithme** : PLSSC\_REC( $T, i_M, Prec$ )

$S \leftarrow$  tab. de taille  $n$ , initialisé à 0

$i \leftarrow i_M$

**tant que**  $i \neq -1$  :

$S[i] \leftarrow 1$

$i \leftarrow Prec[i]$

**renvoyer**  $S$

# Algorithme PLSSC avec reconstruction

**Algorithme** : PLSSC( $T$ )

$L \leftarrow$  tableau de taille  $n$ , initialisé à 1

$Prec \leftarrow$  tableau de taille  $n$ , initialisé à  $-1$

$i_M \leftarrow 0$  // fin de la PLSSC courante

**pour**  $i = 1$  à  $n - 1$  :

$max \leftarrow 0$

**pour**  $j = 0$  à  $i - 1$  :

**si**  $T[j] \leq T[i]$  et  $L[j] > max$  :

$max \leftarrow L[j]$  ;  $Prec[i] \leftarrow j$

$L[i] \leftarrow 1 + max$

**si**  $L[i] > L[i_M]$  :  $i_M \leftarrow i$

**renvoyer**  $L[i_M]$ ,  $i_M$  et  $Prec$

**Algorithme** : PLSSC\_REC( $T, i_M, Prec$ )

$S \leftarrow$  tab. de taille  $n$ , initialisé à 0

$i \leftarrow i_M$

**tant que**  $i \neq -1$  :

$S[i] \leftarrow 1$

$i \leftarrow Prec[i]$

**renvoyer**  $S$

**Lemme**

*L'algo. PLSSC\_REC reconstruit une PLSSC de  $T$  en temps  $O(n)$ .*



1. Premier exemple : plus longue sous-suite croissante
2. Qu'est-ce que la programmation dynamique ?
3. Deuxième exemple : choix de cours, le retour
4. Troisième exemple : la distance d'édition
5. Quatrième exemple : calcul de plus courts chemins
6. Bonus : le voyageur de commerce

## Idée générale

**Programmation dynamique = récursion sans répétition**

# Idée générale

**Programmation dynamique = récursion sans répétition**

## Ingrédients

1. Formule **réursive** 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 **réursive** 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**

## Plus longue sous-suite croissante

1. Définition de  $\ell_i$  en pas seulement  $\ell(T)$
2. Expression de  $\ell_i$  en fonction des  $\ell_j, j < i$

# Ingédient 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**

## Plus longue sous-suite croissante

1. Définition de  $\ell_i$  en pas seulement  $\ell(T)$
2. Expression de  $\ell_i$  en fonction des  $\ell_j, j < i$

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
3. écriture effective de l'algorithme
4. analyse de complexité

*cf. ex. suivants*

## 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
3. écriture effective de l'algorithme
4. analyse de complexité

*cf. ex. suivants*

### Plus longue sous-suite croissante

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
3. écriture effective de l'algorithme
4. analyse de complexité

*cf. ex. suivants*

### Plus longue sous-suite croissante

1. Tableau  $L$
2. Ordre croissant

En pratique, étape souvent non guidée

## Ingédient 3 : Reconstruction

Partie de difficulté très variable !

## Ingédient 3 : Reconstruction

Partie de difficulté très variable !

### Étapes

1. ajout d'informations supplémentaires à l'algo. précédent
2. *redescende* 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

### Plus longue sous-suite croissante

1. tableau Prec, indice  $i_M$
2. descente depuis  $T_{[i_M]}$ , en suivant Prec

## Ingédient 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

### Plus longue sous-suite croissante

1. tableau Prec, indice  $i_M$
2. descente depuis  $T_{[i_M]}$ , en suivant Prec

En pratique, étape pas toujours effectuée

# Problématique de la mémoire

- ▶ Dans PLSSC, tableau Prec de taille  $n$ 
  - ▶ Complexité *en espace*  $O(n)$
  - ▶ Si  $n = 2^{25}$  : env. 1Mo de mémoire

# Problématique de la mémoire

- ▶ Dans PLSSC, tableau Prec de taille  $n$ 
  - ▶ Complexité *en espace*  $O(n)$
  - ▶ Si  $n = 2^{25}$  : 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 les exemples suivants

# 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)

1. Premier exemple : plus longue sous-suite croissante
2. Qu'est-ce que la programmation dynamique ?
3. Deuxième exemple : choix de cours, le retour
4. Troisième exemple : la distance d'édition
5. Quatrième exemple : calcul de plus courts chemins
6. Bonus : le voyageur de commerce



# Revisitons nos classiques

## Choix de cours **valué**

**Entrée** cours  $C_i = (d_i, f_i, e_i)$  [début, fin, crédits ECTS],  $0 \leq i < n$

**Sortie** un sous-ensemble de cours compatibles qui maximise le nombre total de crédits ECTS

# Revisitons nos classiques

## Choix de cours **valué**

**Entrée** cours  $C_i = (d_i, f_i, e_i)$  [début, fin, crédits ECTS],  $0 \leq i < n$

**Sortie** un sous-ensemble de cours compatibles qui maximise le nombre total de crédits ECTS

## Lemme

*L'algorithme CHOIXCOURSGROUTON vu pour le choix de cours non valué est arbitrairement mauvais sur ce problème !*

# Revisitons nos classiques

## Choix de cours **valué**

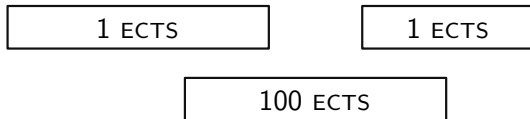
**Entrée** cours  $C_i = (d_i, f_i, e_i)$  [début, fin, crédits ECTS],  $0 \leq i < n$

**Sortie** un sous-ensemble de cours compatibles qui maximise le nombre total de crédits ECTS

## Lemme

*L'algorithme CHOIXCOURSGROUTON vu pour le choix de cours non valué est arbitrairement mauvais sur ce problème !*

## Exemple



- CHOIXCOURSGROUTON renvoie une solution à 2 ECTS
- L'optimal est 50 fois meilleur !

# Formule récursive pour le choix de cours valué

Cours triés par ordre de fin croissante  $C_0, C_1, \dots, C_{n-1}$

## Notations

- ▶  $\text{pred}(k) = \max\{j : f_j \leq d_k\}$  (avec  $\max(\emptyset) = -1$ )
- ▶  $\text{maxECTS}(k)$  : nb. maximal d'ECTS avec les cours  $C_0, \dots, C_k$  ( $\text{maxECTS}(-1) = 0$ )

# Formule récursive pour le choix de cours valué

Cours triés par ordre de fin croissante  $C_0, C_1, \dots, C_{n-1}$

## Notations

- ▶  $\text{pred}(k) = \max\{j : f_j \leq d_k\}$  (avec  $\max(\emptyset) = -1$ )
- ▶  $\text{maxECTS}(k)$  : nb. maximal d'ECTS avec les cours  $C_0, \dots, C_k$  ( $\text{maxECTS}(-1) = 0$ )

## Lemme

$\text{maxECTS}(0) = e_0$  et pour  $1 \leq k < n$ ,

$\text{maxECTS}(k) = \max(\text{maxECTS}(k-1), e_k + \text{maxECTS}(\text{pred}(k)))$

# Formule récursive pour le choix de cours valué

Cours triés par ordre de fin croissante  $C_0, C_1, \dots, C_{n-1}$

## Notations

- ▶  $\text{pred}(k) = \max\{j : f_j \leq d_k\}$  (avec  $\max(\emptyset) = -1$ )
- ▶  $\text{maxECTS}(k)$  : nb. maximal d'ECTS avec les cours  $C_0, \dots, C_k$   
( $\text{maxECTS}(-1) = 0$ )

## Lemme

$\text{maxECTS}(0) = e_0$  et pour  $1 \leq k < n$ ,

$$\text{maxECTS}(k) = \max(\text{maxECTS}(k-1), e_k + \text{maxECTS}(\text{pred}(k)))$$

**Preuve :** la solution optimale pour  $C_0, \dots, C_k$  contient-elle  $C_k$  ?

- ▶ Si oui, les autres cours choisis sont parmi  $C_0, \dots, C_{\text{pred}(k)} \rightsquigarrow$  nb d'ECTS :  
 $e_k + \text{maxECTS}(\text{pred}(k))$
- ▶ Si non, nb d'ECTS :  $\text{maxECTS}(k-1)$

## Algorithme pour le choix de cours valué

**Algorithme** : MAXECTS( $C$ )

Trier  $C$  par dates de fin croissantes

$P \leftarrow$  tableau de taille  $n$ , initialisé à  $-1$

// prédécesseurs

**pour**  $k = 1$  à  $n - 1$  :

**pour**  $j = 0$  à  $k - 1$  :  
        **si**  $f_j \leq d_k$  :  $P_{[k]} \leftarrow j$

$M \leftarrow$  tableau de taille  $n$ , initialisé à  $0$

$M_{[0]} \leftarrow e_0$

**pour**  $k = 1$  à  $n - 1$  :

**si**  $P_{[k]} \neq -1$  :  $M_{[k]} \leftarrow \max(M_{[k-1]}, e_k + M_{[P_{[k]}]})$   
    **sinon** :  $M_{[k]} \leftarrow \max(M_{[k-1]}, e_k)$

**renvoyer**  $M_{[n-1]}$

## Algorithme pour le choix de cours valué

**Algorithme :** MAXECTS( $C$ )

Trier  $C$  par dates de fin croissantes

$P \leftarrow$  tableau de taille  $n$ , initialisé à  $-1$

// prédécesseurs

**pour**  $k = 1$  à  $n - 1$  :

**pour**  $j = 0$  à  $k - 1$  :  
        **si**  $f_j \leq d_k$  :  $P_{[k]} \leftarrow j$

$M \leftarrow$  tableau de taille  $n$ , initialisé à  $0$

$M_{[0]} \leftarrow e_0$

**pour**  $k = 1$  à  $n - 1$  :

**si**  $P_{[k]} \neq -1$  :  $M_{[k]} \leftarrow \max(M_{[k-1]}, e_k + M_{[P_{[k]}]})$   
    **sinon** :  $M_{[k]} \leftarrow \max(M_{[k-1]}, e_k)$

**renvoyer**  $M_{[n-1]}$

Complexité :  $O(n^2)$  (calcul de  $P$ , calcul de  $M$  en  $O(n)$ )



# Et pour le sac-à-dos ?

## Sac-à-dos (non fractionnaire)

**Entrée** un ensemble d'objets, ayant une taille  $t_i$  et une valeur  $v_i$   
une taille  $T$  de sac-à-dos

**Sortie** un sous-ensemble d'objets qui rentre dans le sac, et maximise la valeur totale

# Et pour le sac-à-dos ?

## Sac-à-dos (non fractionnaire)

**Entrée** un ensemble d'objets, ayant une taille  $t_i$  et une valeur  $v_i$   
une taille  $T$  de sac-à-dos

**Sortie** un sous-ensemble d'objets qui rentre dans le sac, et maximise la valeur totale

## Lemme

*L'algorithme SÀDFRACGLOUTON n'est pas optimal pour le sac-à-dos non fractionnaire.*

# Et pour le sac-à-dos ?

## Sac-à-dos (non fractionnaire)

**Entrée** un ensemble d'objets, ayant une taille  $t_i$  et une valeur  $v_i$   
une taille  $T$  de sac-à-dos

**Sortie** un sous-ensemble d'objets qui rentre dans le sac, et maximise la valeur totale

## Lemme

*L'algorithme SÀDFRACGLOUTON n'est pas optimal pour le sac-à-dos non fractionnaire.*

- ▶ Exemples vus en cours et TD
- ▶ Algos de programmation dynamique optimaux  $\rightsquigarrow$  cf. TD

1. Premier exemple : plus longue sous-suite croissante
2. Qu'est-ce que la programmation dynamique ?
3. Deuxième exemple : choix de cours, le retour
- 4. Troisième exemple : la distance d'édition**
5. Quatrième exemple : calcul de plus courts chemins
6. Bonus : le voyageur de commerce

AGORRYTNES

## Corrigeons les erreurs

↓ ↓

AGORRYTNS

## Corrigeons les erreurs

↓ ↓  
AGORRYTNES

ALGORITHMES  
x

## Corrigeons les erreurs

↓ ↓

AGORRYTNS

# ALGORITHMES

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



## Corrigeons les erreurs

AGORRYTNES

ALGORITHMES

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  $A$  et  $B$  est le **nombre minimal de désaccords** dans un **alignement** de  $A$  et  $B$

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



# Définition du problème

**Entrée** Deux mots  $A$  et  $B$  sur un alphabet  
(mot : chaîne de caractère ou tableau de caractères ou ...)

**Sortie 1** La distance d'édition entre  $A$  et  $B$

**Sortie 2** Un alignement optimal de  $A$  et  $B$

# Définition du problème

**Entrée** Deux mots  $A$  et  $B$  sur un alphabet  
(mot : chaîne de caractère ou tableau de caractères ou ...)

**Sortie 1** La distance d'édition entre  $A$  et  $B$

**Sortie 2** Un alignement optimal de  $A$  et  $B$

## Utilité

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

# Formalisation

- ▶  $A_{[0,i[} = A_{[0]}A_{[1]} \cdots A_{[i-1]}$  et  $B_{[0,j[} = B_{[0]}B_{[1]} \cdots B_{[j-1]}$
- ▶  $\text{edit}(i, j)$  : distance entre  $A_{[0,i[}$  et  $B_{[0,j[}$ 
  - ▶  $\text{edit}(i, 0) = i$
  - ▶  $\text{edit}(0, j) = j$

# Formalisation

- ▶  $A_{[0,i[} = A_{[0]}A_{[1]} \cdots A_{[i-1]}$  et  $B_{[0,j[} = B_{[0]}B_{[1]} \cdots B_{[j-1]}$
- ▶  $\text{edit}(i, j)$  : distance entre  $A_{[0,i[}$  et  $B_{[0,j[}$ 
  - ▶  $\text{edit}(i, 0) = i$
  - ▶  $\text{edit}(0, j) = j$
- ▶ si  $|A| = m$  et  $|B| = n$ , on cherche  $\text{edit}(m, n)$

# Formalisation

- ▶  $A_{[0,i[} = A_{[0]}A_{[1]} \cdots A_{[i-1]}$  et  $B_{[0,j[} = B_{[0]}B_{[1]} \cdots B_{[j-1]}$
- ▶  $\text{edit}(i, j)$  : distance entre  $A_{[0,i[}$  et  $B_{[0,j[}$ 
  - ▶  $\text{edit}(i, 0) = i$
  - ▶  $\text{edit}(0, j) = j$
- ▶ si  $|A| = m$  et  $|B| = n$ , on cherche  $\text{edit}(m, n)$

## Trois alignements possibles

<table><tr><td><math>A_{[0,i-1[}</math></td></tr><tr><td><math>B_{[0,j[}</math></td></tr></table> $A_{[i-1]}$	$A_{[0,i-1[}$	$B_{[0,j[}$		<table><tr><td><math>A_{[0,i[}</math></td></tr><tr><td><math>B_{[0,j-1[}</math></td></tr></table> $-$ $B_{[j-1]}$	$A_{[0,i[}$	$B_{[0,j-1[}$		<table><tr><td><math>A_{[0,i-1[}</math></td></tr><tr><td><math>B_{[0,j-1[}</math></td></tr></table> $A_{[i-1]}$ $B_{[j-1]}$	$A_{[0,i-1[}$	$B_{[0,j-1[}$
$A_{[0,i-1[}$										
$B_{[0,j[}$										
$A_{[0,i[}$										
$B_{[0,j-1[}$										
$A_{[0,i-1[}$										
$B_{[0,j-1[}$										



# Formalisation

- ▶  $A_{[0,i[} = A_{[0]}A_{[1]} \cdots A_{[i-1]}$  et  $B_{[0,j[} = B_{[0]}B_{[1]} \cdots B_{[j-1]}$
- ▶  $\text{edit}(i, j)$  : distance entre  $A_{[0,i[}$  et  $B_{[0,j[}$ 
  - ▶  $\text{edit}(i, 0) = i$
  - ▶  $\text{edit}(0, j) = j$
- ▶ si  $|A| = m$  et  $|B| = n$ , on cherche  $\text{edit}(m, n)$

## Trois alignements possibles

$A_{[0,i-1[}$	$A_{[i-1]}$		$A_{[0,i[}$	-		$A_{[0,i-1[}$	$A_{[i-1]}$
$B_{[0,j[}$	-		$B_{[0,j-1[}$	$B_{[j-1]}$		$B_{[0,j-1[}$	$B_{[j-1]}$

## Alignements AGORR et ALGOR

AGORR	-		AGOR	R		AGOR	R
ALGO	R		ALGOR	-		ALGO	R
3	1		1	1		2	0
							→ 2

# Formule récursive

Lemme

$$\text{edit}(i, j) = \min \begin{cases} \text{edit}(i-1, j) + 1 \\ \text{edit}(i, j-1) + 1 \\ \text{edit}(i-1, j-1) + \epsilon_{ij} \end{cases}$$

où  $\epsilon_{ij} = 1$  si  $A_{[i-1]} \neq B_{[j-1]}$ , 0 sinon

# Formule récursive

Lemme

$$\text{edit}(i, j) = \min \begin{cases} \text{edit}(i-1, j) + 1 \\ \text{edit}(i, j-1) + 1 \\ \text{edit}(i-1, j-1) + \epsilon_{ij} \end{cases}$$

où  $\epsilon_{ij} = 1$  si  $A_{[i-1]} \neq B_{[j-1]}$ , 0 sinon

Preuve :

# Formule récursive

## Lemme

$$\text{edit}(i, j) = \min \begin{cases} \text{edit}(i-1, j) + 1 \\ \text{edit}(i, j-1) + 1 \\ \text{edit}(i-1, j-1) + \epsilon_{ij} \end{cases}$$

où  $\epsilon_{ij} = 1$  si  $A_{[i-1]} \neq B_{[j-1]}$ , 0 sinon

## Preuve :

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

$A_{[0, i-1]}$
$B_{[0, j]}$

 $A_{[i-1]}$   
-

# Formule récursive

## Lemme

$$\text{edit}(i, j) = \min \begin{cases} \text{edit}(i-1, j) + 1 \\ \text{edit}(i, j-1) + 1 \\ \text{edit}(i-1, j-1) + \epsilon_{ij} \end{cases}$$

où  $\epsilon_{ij} = 1$  si  $A_{[i-1]} \neq B_{[j-1]}$ , 0 sinon

## Preuve :

► $\text{edit}(i, j) \leq \text{edit}(i-1, j) + 1 :$	<table><tbody><tr><td><math>A_{[0, i-1]}</math></td><td><math>A_{[i-1]}</math></td></tr><tr><td><math>B_{[0, j]}</math></td><td>-</td></tr></tbody></table>	$A_{[0, i-1]}$	$A_{[i-1]}$	$B_{[0, j]}$	-
$A_{[0, i-1]}$	$A_{[i-1]}$				
$B_{[0, j]}$	-				
► $\text{edit}(i, j) \leq \text{edit}(i, j-1) + 1 :$	<table><tbody><tr><td><math>A_{[0, i]}</math></td><td>-</td></tr><tr><td><math>B_{[0, j-1]}</math></td><td><math>B_{[j-1]}</math></td></tr></tbody></table>	$A_{[0, i]}$	-	$B_{[0, j-1]}$	$B_{[j-1]}$
$A_{[0, i]}$	-				
$B_{[0, j-1]}$	$B_{[j-1]}$				

# Formule récursive

## Lemme

$$\text{edit}(i, j) = \min \begin{cases} \text{edit}(i-1, j) + 1 \\ \text{edit}(i, j-1) + 1 \\ \text{edit}(i-1, j-1) + \epsilon_{ij} \end{cases}$$

où  $\epsilon_{ij} = 1$  si  $A_{[i-1]} \neq B_{[j-1]}$ , 0 sinon

## Preuve :

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

$A_{[0, i-1[}$
$B_{[0, j[}$

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

$A_{[0, i[}$
$B_{[0, j-1[}$

 -  
 $B_{[j-1]}$
- ▶  $\text{edit}(i, j) \leq \text{edit}(i-1, j-1) + \epsilon_{ij}$  : 

$A_{[0, i-1[}$
$B_{[0, j-1[}$

 $A_{[i-1]}$   
 $B_{[j-1]}$

# Formule récursive

## Lemme

$$\text{edit}(i, j) = \min \begin{cases} \text{edit}(i-1, j) + 1 \\ \text{edit}(i, j-1) + 1 \\ \text{edit}(i-1, j-1) + \epsilon_{ij} \end{cases}$$

où  $\epsilon_{ij} = 1$  si  $A_{[i-1]} \neq B_{[j-1]}$ , 0 sinon

## Preuve :

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

$A_{[0, i-1[}$	$A_{[i-1]}$
$B_{[0, j]}$	-

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

$A_{[0, i]}$	-
$B_{[0, j-1]}$	$B_{[j-1]}$

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

$A_{[0, i-1]}$	$A_{[i-1]}$
$B_{[0, j-1]}$	$B_{[j-1]}$

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

# Formule récursive

Lemme

$$\text{edit}(i, j) = \min \begin{cases} \text{edit}(i-1, j) + 1 \\ \text{edit}(i, j-1) + 1 \\ \text{edit}(i-1, j-1) + \epsilon_{ij} \end{cases}$$

où  $\epsilon_{ij} = 1$  si  $A_{[i-1]} \neq B_{[j-1]}$ , 0 sinon

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



# Formule récursive

## Lemme

$$\text{edit}(i, j) = \min \begin{cases} \text{edit}(i-1, j) + 1 \\ \text{edit}(i, j-1) + 1 \\ \text{edit}(i-1, j-1) + \epsilon_{ij} \end{cases}$$

où  $\epsilon_{ij} = 1$  si  $A_{[i-1]} \neq B_{[j-1]}$ , 0 sinon

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

$$\blacktriangleright A_{[i-1]} = B_{[j-1]} \implies \text{edit}(i-1, j-1) = d$$

# Formule récursive

## Lemme

$$\text{edit}(i, j) = \min \begin{cases} \text{edit}(i-1, j) + 1 \\ \text{edit}(i, j-1) + 1 \\ \text{edit}(i-1, j-1) + \epsilon_{ij} \end{cases}$$

où  $\epsilon_{ij} = 1$  si  $A_{[i-1]} \neq B_{[j-1]}$ , 0 sinon

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

- ▶  $A_{[i-1]} = B_{[j-1]} \implies \text{edit}(i-1, j-1) = d$
- ▶  $A_{[i-1]} \neq B_{[j-1]}$  :
  - ▶  $\text{edit}(i-1, j-1) \geq d-1$
  - ▶  $\text{edit}(i, j-1) \geq d-1$
  - ▶  $\text{edit}(i-1, j) \geq d-1$

# Formule récursive

## Lemme

$$\text{edit}(i, j) = \min \begin{cases} \text{edit}(i-1, j) + 1 \\ \text{edit}(i, j-1) + 1 \\ \text{edit}(i-1, j-1) + \epsilon_{ij} \end{cases}$$

où  $\epsilon_{ij} = 1$  si  $A_{[i-1]} \neq B_{[j-1]}$ , 0 sinon

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

- ▶  $A_{[i-1]} = B_{[j-1]} \implies \text{edit}(i-1, j-1) = d$
- ▶  $A_{[i-1]} \neq B_{[j-1]}$  :
  - ▶  $\text{edit}(i-1, j-1) \geq d-1$
  - ▶  $\text{edit}(i, j-1) \geq d-1$
  - ▶  $\text{edit}(i-1, j) \geq d-1$

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

## 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

**Algorithme** : EDITION( $A, B$ )

$(m, n) \leftarrow$  tailles de  $A$  et  $B$

$E \leftarrow$  tableau de dimensions  $m + 1$  par  $n + 1$

**pour**  $i = 0$  à  $m$  :  $E_{[i,0]} \leftarrow i$  // cas de

**pour**  $j = 0$  à  $n$  :  $E_{[0,j]} \leftarrow j$  // base

**pour**  $i = 1$  à  $m$  :

**pour**  $j = 1$  à  $n$  :

$\epsilon \leftarrow 0$

**si**  $A_{[i-1]} \neq B_{[j-1]}$  :  $\epsilon \leftarrow 1$

$E_{[i,j]} \leftarrow \min(E_{[i-1,j]} + 1, E_{[i,j-1]} + 1, E_{[i-1,j-1]} + \epsilon)$

**renvoyer**  $E_{[m,n]}$

## Lemme

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

## 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  $i - 1$  !

**Algorithme :** EDITIONMINMEMOIRE( $A, B$ )

$(m, n) \leftarrow$  tailles de  $A$  et  $B$  // Hyp. :  $m \geq n$

$P \leftarrow$  tableau de dimension  $n + 1$  // Ligne précédente

$C \leftarrow$  tableau de dimension  $n + 1$  // Ligne courante

**pour**  $j = 0$  à  $n$  :  $P[j] \leftarrow j$  // cas de base

**pour**  $i = 1$  à  $m$  :

$C[0] \leftarrow i$  // cas de base

**pour**  $j = 1$  à  $n$  :

$\epsilon \leftarrow 0$  si  $A[i-1] = B[j-1]$ , 1 sinon

$C[j] \leftarrow \min(P[j] + 1, C[j-1] + 1, P[j-1] + \epsilon)$

**pour**  $j = 0$  à  $n$  :  $P[j] \leftarrow C[j]$

**renvoyer**  $C[n]$

## Version efficace en mémoire

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

**Algorithme :** EDITIONMINMEMOIRE( $A, B$ )

$(m, n) \leftarrow$  tailles de  $A$  et  $B$  // Hyp. :  $m \geq n$

$P \leftarrow$  tableau de dimension  $n + 1$  // Ligne précédente

$C \leftarrow$  tableau de dimension  $n + 1$  // Ligne courante

**pour**  $j = 0$  à  $n$  :  $P[j] \leftarrow j$  // cas de base

**pour**  $i = 1$  à  $m$  :

$C[0] \leftarrow i$  // cas de base

**pour**  $j = 1$  à  $n$  :

$\epsilon \leftarrow 0$  si  $A[i-1] = B[j-1]$ , 1 sinon

$C[j] \leftarrow \min(P[j] + 1, C[j-1] + 1, P[j-1] + \epsilon)$

**pour**  $j = 0$  à  $n$  :  $P[j] \leftarrow C[j]$

**renvoyer**  $C[n]$

### Lemme

EDITIONMINMEMOIRE  
calcule la distance entre  
 $A$  et  $B$  en temps  $O(mn)$   
et **espace  $O(n)$** .



## 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	4	5	6
H	8	7	6	5	4	4	4	4	5	6
M	9	8	7	6	5	5	5	5	5	6
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 : example

		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

**Algorithme :** ALIGNEMENT( $A, B, E$ )

```
(i, j) ← (m, n)                                // tailles de A et B
tant que i > 0 et j > 0 :
    si E[i,j] = E[i-1,j-1] et A[i-1] = B[j-1] :           // A[i-1]/B[j-1]
        | (i, j) ← (i - 1, j - 1)
    sinon si E[i,j] = E[i-1,j-1] + 1 :                   // A[i-1]/B[j-1]
        | (i, j) ← (i - 1, j - 1)
    sinon si E[i,j] = E[i-1,j] + 1 :                       // A[i-1]/-
        | Insérer « _ » en jème position dans B ; i ← i - 1
    sinon si E[i,j] = E[i,j-1] + 1 :                       // -/B[j-1]
        | Insérer « _ » en ième position dans A ; j ← j - 1
Insérer j symboles « _ » en tête de A              // 1'un ou
Insérer i symboles « _ » en tête de B              // 1'autre
renvoyer A et B
```

# Correction et complexité

## Lemme

*L'algorithme ALIGNEMENT aligne les mots  $A$  et  $B$  de manière optimale, en temps  $O(m + n)$ .*

**Algorithme :** ALIGNEMENT( $A, B, E$ )

$(i, j) \leftarrow (m, n)$

**tant que**  $i > 0$  et  $j > 0$  :

**si**  $E_{[i,j]} = E_{[i-1,j-1]}$  et  $A_{[i-1]} = B_{[j-1]}$  :

$(i, j) \leftarrow (i - 1, j - 1)$

**sinon si**  $E_{[i,j]} = E_{[i-1,j-1]} + 1$  :

$(i, j) \leftarrow (i - 1, j - 1)$

**sinon si**  $E_{[i,j]} = E_{[i-1,j]} + 1$  :

        Insérer « \_ » en  $j^{\text{ème}}$  position dans  $B$  ;  $i \leftarrow i - 1$

**sinon si**  $E_{[i,j]} = E_{[i,j-1]} + 1$  :

        Insérer « \_ » en  $i^{\text{ème}}$  position dans  $A$  ;  $j \leftarrow j - 1$

Insérer  $j$  symboles « \_ » en tête de  $A$

Insérer  $i$  symboles « \_ » en tête de  $B$

**renvoyer**  $A$  et  $B$

# Correction et complexité

## Lemme

*L'algorithme ALIGNEMENT aligne les mots  $A$  et  $B$  de manière optimale, en temps  $O(m + n)$ .*

**Algorithme :** ALIGNEMENT( $A, B, E$ )

$(i, j) \leftarrow (m, n)$

**tant que**  $i > 0$  et  $j > 0$  :

**si**  $E_{[i,j]} = E_{[i-1,j-1]}$  et  $A_{[i-1]} = B_{[j-1]}$  :

$(i, j) \leftarrow (i - 1, j - 1)$

**sinon si**  $E_{[i,j]} = E_{[i-1,j-1]} + 1$  :

$(i, j) \leftarrow (i - 1, j - 1)$

**sinon si**  $E_{[i,j]} = E_{[i-1,j]} + 1$  :

        Insérer « \_ » en  $j^{\text{ème}}$  position dans  $B$  ;  $i \leftarrow i - 1$

**sinon si**  $E_{[i,j]} = E_{[i,j-1]} + 1$  :

        Insérer « \_ » en  $i^{\text{ème}}$  position dans  $A$  ;  $j \leftarrow j - 1$

Insérer  $j$  symboles « \_ » en tête de  $A$

Insérer  $i$  symboles « \_ » en tête de  $B$

**renvoyer**  $A$  et  $B$

**Preuve de complexité :** à chaque tour,  $i + j$  diminue de  $\geq 1$



# Correction et complexité

## Lemme

*L'algorithme ALIGNEMENT aligne les mots  $A$  et  $B$  de manière optimale, en temps  $O(m + n)$ .*

**Algorithme :** ALIGNEMENT( $A, B, E$ )

$(i, j) \leftarrow (m, n)$

**tant que**  $i > 0$  et  $j > 0$  :

**si**  $E[i, j] = E[i-1, j-1]$  et  $A[i-1] = B[j-1]$  :

$(i, j) \leftarrow (i-1, j-1)$

**sinon si**  $E[i, j] = E[i-1, j-1] + 1$  :

$(i, j) \leftarrow (i-1, j-1)$

**sinon si**  $E[i, j] = E[i-1, j] + 1$  :

        Insérer « \_ » en  $j^{\text{ème}}$  position dans  $B$  ;  $i \leftarrow i-1$

**sinon si**  $E[i, j] = E[i, j-1] + 1$  :

        Insérer « \_ » en  $i^{\text{ème}}$  position dans  $A$  ;  $j \leftarrow j-1$

Insérer  $j$  symboles « \_ » en tête de  $A$

Insérer  $i$  symboles « \_ » en tête de  $B$

**renvoyer**  $A$  et  $B$

**Preuve de complexité :** à chaque tour,  $i + j$  diminue de  $\geq 1$

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

# Conclusion

## Théorème

*La distance d'édition entre deux mots  $A$  et  $B$  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.*

# Conclusion

## Théorème

*La distance d'édition entre deux mots  $A$  et  $B$  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.*

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

# Conclusion

## Théorème

*La distance d'édition entre deux mots  $A$  et  $B$  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.*

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

↪ un peu...

# Conclusion

## Théorème

*La distance d'édition entre deux mots  $A$  et  $B$  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.*

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

↪ un peu...

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

# Conclusion

## Théorème

*La distance d'édition entre deux mots  $A$  et  $B$  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.*

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

↪ un peu...

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

↪ sans doute pas (si on veut le résultat exact)

# Conclusion

## Théorème

*La distance d'édition entre deux mots  $A$  et  $B$  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.*

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

↪ un peu...

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

↪ sans doute pas (si on veut le résultat exact)

- ▶ Problème très important en pratique... et en théorie !
- ▶ Beaucoup de résultats très récents (2020 !) sur le sujet

1. Premier exemple : plus longue sous-suite croissante
2. Qu'est-ce que la programmation dynamique ?
3. Deuxième exemple : choix de cours, le retour
4. Troisième exemple : la distance d'édition
5. Quatrième exemple : calcul de plus courts chemins
6. Bonus : le voyageur de commerce



## Rappel : parcours en largeur d'un graphe

**Algorithme :** PARCOURS LARGEUR( $G, s$ )

$F \leftarrow$  file vide

Ajouter  $s$  à  $F$  et marquer  $s$

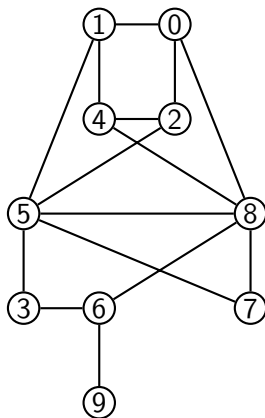
**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

    Afficher  $u$

**pour** tout voisin non marqué  $v$  de  $u$  :

        Ajouter  $v$  à  $F$  et marquer  $v$



## Rappel : parcours en largeur d'un graphe

**Algorithme :** PARCOURS LARGEUR( $G, s$ )

$F \leftarrow$  file vide

Ajouter  $s$  à  $F$  et marquer  $s$

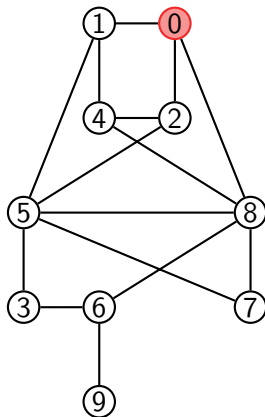
**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

    Afficher  $u$

**pour** tout voisin non marqué  $v$  de  $u$  :

        Ajouter  $v$  à  $F$  et marquer  $v$



► File : 0

► Affichage :

## Rappel : parcours en largeur d'un graphe

**Algorithme** : PARCOURS LARGEUR( $G, s$ )

$F \leftarrow$  file vide

Ajouter  $s$  à  $F$  et marquer  $s$

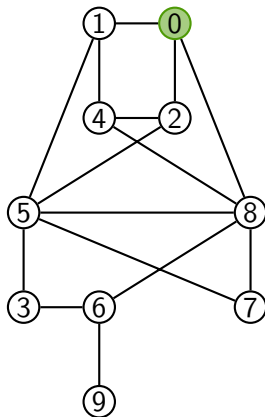
**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

    Afficher  $u$

**pour** tout voisin non marqué  $v$  de  $u$  :

        Ajouter  $v$  à  $F$  et marquer  $v$



► File :

► Affichage : 0

## Rappel : parcours en largeur d'un graphe

**Algorithme :** PARCOURS LARGEUR( $G, s$ )

$F \leftarrow$  file vide

Ajouter  $s$  à  $F$  et marquer  $s$

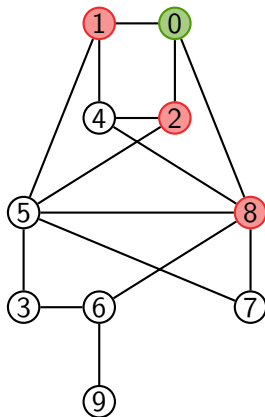
**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

    Afficher  $u$

**pour** tout voisin non marqué  $v$  de  $u$  :

        Ajouter  $v$  à  $F$  et marquer  $v$



► File : 1 2 8

► Affichage : 0

## Rappel : parcours en largeur d'un graphe

**Algorithme :** PARCOURS LARGEUR( $G, s$ )

$F \leftarrow$  file vide

Ajouter  $s$  à  $F$  et marquer  $s$

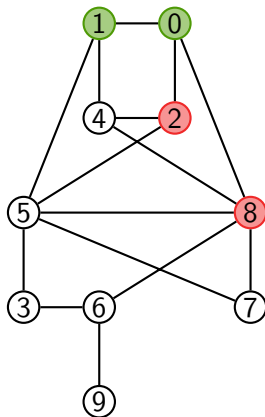
**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

    Afficher  $u$

**pour** tout voisin non marqué  $v$  de  $u$  :

        Ajouter  $v$  à  $F$  et marquer  $v$



► File : 2 8

► Affichage : 0 1

## Rappel : parcours en largeur d'un graphe

**Algorithme :** PARCOURS LARGEUR( $G, s$ )

$F \leftarrow$  file vide

Ajouter  $s$  à  $F$  et marquer  $s$

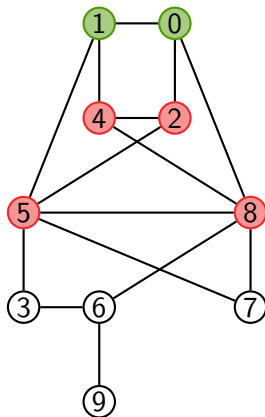
**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

    Afficher  $u$

**pour** tout voisin non marqué  $v$  de  $u$  :

        Ajouter  $v$  à  $F$  et marquer  $v$



► File : 2 8 4 5

► Affichage : 0 1

## Rappel : parcours en largeur d'un graphe

**Algorithme** : PARCOURS LARGEUR( $G, s$ )

$F \leftarrow$  file vide

Ajouter  $s$  à  $F$  et marquer  $s$

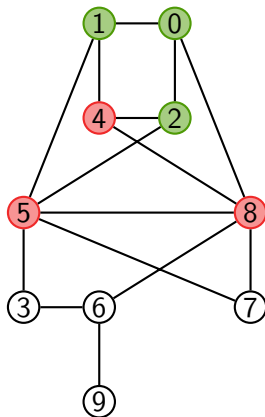
**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

    Afficher  $u$

**pour** tout voisin non marqué  $v$  de  $u$  :

        Ajouter  $v$  à  $F$  et marquer  $v$



► File : 2 8 4 5

► Affichage : 0 1 2

## Rappel : parcours en largeur d'un graphe

**Algorithme :** PARCOURS LARGEUR( $G, s$ )

$F \leftarrow$  file vide

Ajouter  $s$  à  $F$  et marquer  $s$

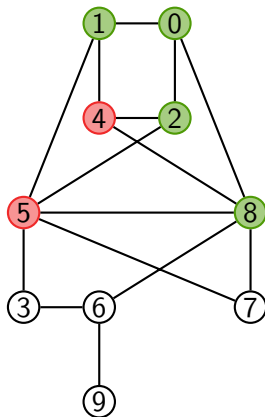
**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

    Afficher  $u$

**pour** tout voisin non marqué  $v$  de  $u$  :

        Ajouter  $v$  à  $F$  et marquer  $v$



► File : 8 4 5

► Affichage : 0 1 2 8



## Rappel : parcours en largeur d'un graphe

**Algorithme :** PARCOURS LARGEUR( $G, s$ )

$F \leftarrow$  file vide

Ajouter  $s$  à  $F$  et marquer  $s$

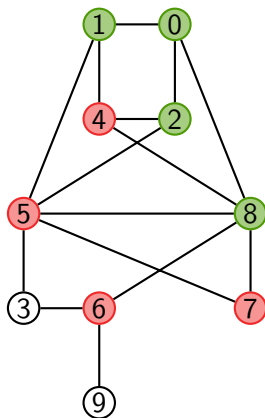
**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

    Afficher  $u$

**pour** tout voisin non marqué  $v$  de  $u$  :

        Ajouter  $v$  à  $F$  et marquer  $v$



► File : 4 5 6 7

► Affichage : 0 1 2 8

## Rappel : parcours en largeur d'un graphe

**Algorithme :** PARCOURS LARGEUR( $G, s$ )

$F \leftarrow$  file vide

Ajouter  $s$  à  $F$  et marquer  $s$

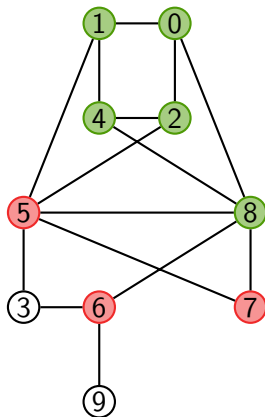
**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

    Afficher  $u$

**pour** tout voisin non marqué  $v$  de  $u$  :

        Ajouter  $v$  à  $F$  et marquer  $v$



► File : 4 5 6 7

► Affichage : 0 1 2 8 4

## Rappel : parcours en largeur d'un graphe

**Algorithme :** PARCOURS LARGEUR( $G, s$ )

$F \leftarrow$  file vide

Ajouter  $s$  à  $F$  et marquer  $s$

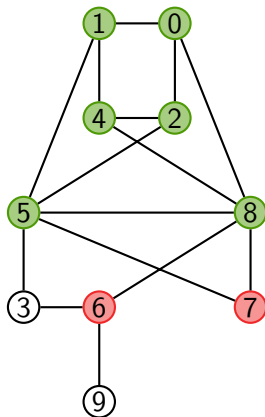
**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

    Afficher  $u$

**pour** tout voisin non marqué  $v$  de  $u$  :

        Ajouter  $v$  à  $F$  et marquer  $v$



► File : 5 6 7

► Affichage : 0 1 2 8 4 5

## Rappel : parcours en largeur d'un graphe

**Algorithme :** PARCOURS LARGEUR( $G, s$ )

$F \leftarrow$  file vide

Ajouter  $s$  à  $F$  et marquer  $s$

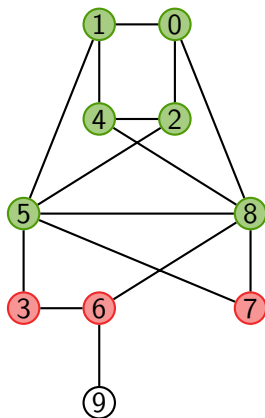
**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

    Afficher  $u$

**pour** tout voisin non marqué  $v$  de  $u$  :

        Ajouter  $v$  à  $F$  et marquer  $v$



► File : 6 7 3

► Affichage : 0 1 2 8 4 5

## Rappel : parcours en largeur d'un graphe

**Algorithme :** PARCOURS LARGEUR( $G, s$ )

$F \leftarrow$  file vide

Ajouter  $s$  à  $F$  et marquer  $s$

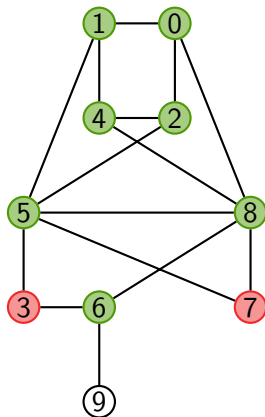
**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

    Afficher  $u$

**pour** tout voisin non marqué  $v$  de  $u$  :

        Ajouter  $v$  à  $F$  et marquer  $v$



► File : 6 7 3

► Affichage : 0 1 2 8 4 5 6

## Rappel : parcours en largeur d'un graphe

**Algorithme :** PARCOURS LARGEUR( $G, s$ )

$F \leftarrow$  file vide

Ajouter  $s$  à  $F$  et marquer  $s$

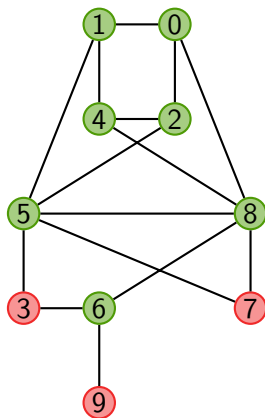
**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

    Afficher  $u$

**pour** tout voisin non marqué  $v$  de  $u$  :

        Ajouter  $v$  à  $F$  et marquer  $v$



► File : 7 3 9

► Affichage : 0 1 2 8 4 5 6

## Rappel : parcours en largeur d'un graphe

**Algorithme :** PARCOURS LARGEUR( $G, s$ )

$F \leftarrow$  file vide

Ajouter  $s$  à  $F$  et marquer  $s$

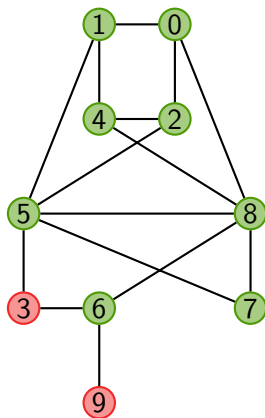
**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

    Afficher  $u$

**pour** tout voisin non marqué  $v$  de  $u$  :

        Ajouter  $v$  à  $F$  et marquer  $v$



► File : 7 3 9

► Affichage : 0 1 2 8 4 5 6 7

## Rappel : parcours en largeur d'un graphe

**Algorithme :** PARCOURS LARGEUR( $G, s$ )

$F \leftarrow$  file vide

Ajouter  $s$  à  $F$  et marquer  $s$

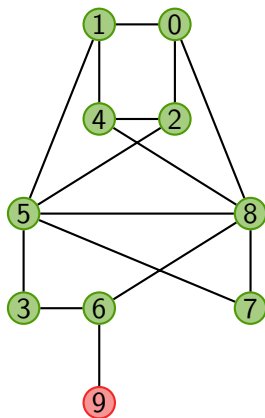
**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

    Afficher  $u$

**pour** tout voisin non marqué  $v$  de  $u$  :

        Ajouter  $v$  à  $F$  et marquer  $v$



► File : 3 9

► Affichage : 0 1 2 8 4 5 6 7 3



## Rappel : parcours en largeur d'un graphe

**Algorithme :** PARCOURS LARGEUR( $G, s$ )

$F \leftarrow$  file vide

Ajouter  $s$  à  $F$  et marquer  $s$

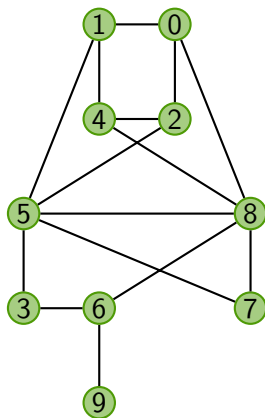
**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

    Afficher  $u$

**pour** tout voisin non marqué  $v$  de  $u$  :

        Ajouter  $v$  à  $F$  et marquer  $v$



► File : 9

► Affichage : 0 1 2 8 4 5 6 7 3 9

## Parcours en largeur et calcul des distances

**Algorithme :** PARCOURS LARGEUR( $G, s$ )

$F \leftarrow$  file vide

Ajouter  $s$  à  $F$  et marquer  $s$

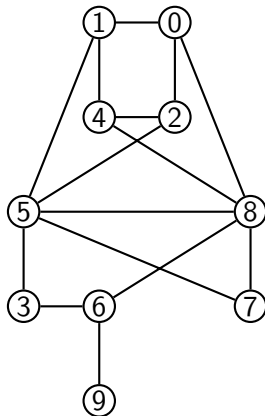
**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

    Afficher  $u$

**pour** tout voisin non marqué  $v$  de  $u$  :

        Ajouter  $v$  à  $F$  et marquer  $v$



## Parcours en largeur et calcul des distances

**Algorithme :** DISTANCES( $G, s$ )

$F \leftarrow$  file vide;  $D[u] \leftarrow +\infty$  pour tout  $u$

Ajouter  $s$  à  $F$ ;  $D[s] \leftarrow 0$

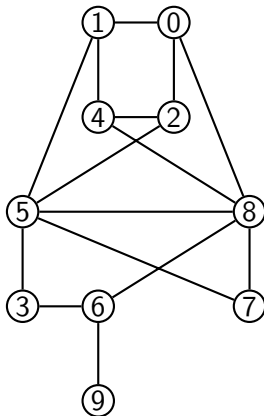
**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

**pour** tout voisin  $v$  de  $u$  tq  $D[v] = +\infty$  :

        Ajouter  $v$  à  $F$ ;  $D[v] \leftarrow D[u] + 1$

**renvoyer**  $D$



## Parcours en largeur et calcul des distances

**Algorithme :** DISTANCES( $G, s$ )

$F \leftarrow$  file vide;  $D[u] \leftarrow +\infty$  pour tout  $u$

Ajouter  $s$  à  $F$ ;  $D[s] \leftarrow 0$

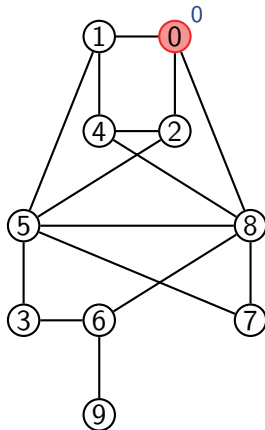
**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

**pour** tout voisin  $v$  de  $u$  tq  $D[v] = +\infty$  :

        Ajouter  $v$  à  $F$ ;  $D[v] \leftarrow D[u] + 1$

**renvoyer**  $D$



► File : 0

# Parcours en largeur et calcul des distances

**Algorithme :** DISTANCES( $G, s$ )

$F \leftarrow$  file vide;  $D[u] \leftarrow +\infty$  pour tout  $u$

Ajouter  $s$  à  $F$ ;  $D[s] \leftarrow 0$

**tant que**  $F$  est non vide :

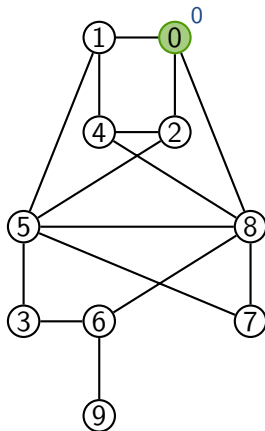
$u \leftarrow$  défiler un élément de  $F$

**pour** tout voisin  $v$  de  $u$  tq  $D[v] = +\infty$  :

        Ajouter  $v$  à  $F$ ;  $D[v] \leftarrow D[u] + 1$

**renvoyer**  $D$

► File :



# Parcours en largeur et calcul des distances

**Algorithme :** DISTANCES( $G, s$ )

$F \leftarrow$  file vide;  $D[u] \leftarrow +\infty$  pour tout  $u$

Ajouter  $s$  à  $F$ ;  $D[s] \leftarrow 0$

**tant que**  $F$  est non vide :

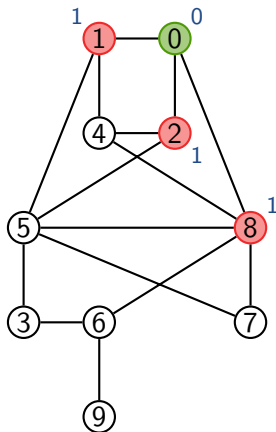
$u \leftarrow$  défiler un élément de  $F$

**pour** tout voisin  $v$  de  $u$  tq  $D[v] = +\infty$  :

            Ajouter  $v$  à  $F$ ;  $D[v] \leftarrow D[u] + 1$

**renvoyer**  $D$

► File : 1 2 8



## Parcours en largeur et calcul des distances

**Algorithme :** DISTANCES( $G, s$ )

$F \leftarrow$  file vide;  $D[u] \leftarrow +\infty$  pour tout  $u$

Ajouter  $s$  à  $F$ ;  $D[s] \leftarrow 0$

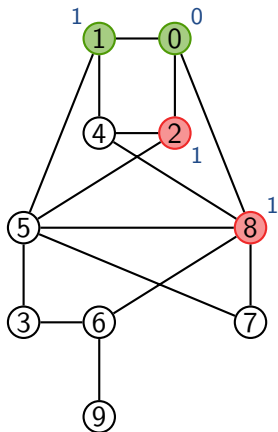
**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

**pour** tout voisin  $v$  de  $u$  tq  $D[v] = +\infty$  :

        Ajouter  $v$  à  $F$ ;  $D[v] \leftarrow D[u] + 1$

**renvoyer**  $D$



► File : 2 8

# Parcours en largeur et calcul des distances

**Algorithme :** DISTANCES( $G, s$ )

$F \leftarrow$  file vide;  $D[u] \leftarrow +\infty$  pour tout  $u$

Ajouter  $s$  à  $F$ ;  $D[s] \leftarrow 0$

**tant que**  $F$  est non vide :

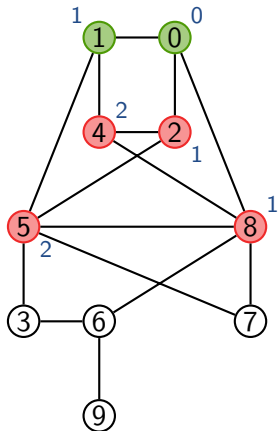
$u \leftarrow$  défiler un élément de  $F$

**pour** tout voisin  $v$  de  $u$  tq  $D[v] = +\infty$  :

            Ajouter  $v$  à  $F$ ;  $D[v] \leftarrow D[u] + 1$

**renvoyer**  $D$

► File : 2 8 4 5





# Parcours en largeur et calcul des distances

**Algorithme :** DISTANCES( $G, s$ )

$F \leftarrow$  file vide;  $D[u] \leftarrow +\infty$  pour tout  $u$

Ajouter  $s$  à  $F$ ;  $D[s] \leftarrow 0$

**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

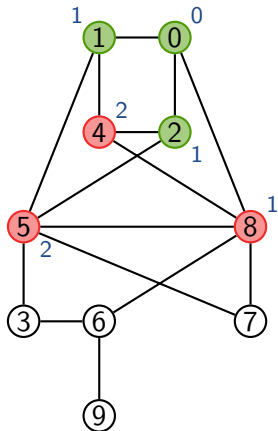
**pour** tout voisin  $v$  de  $u$  tq  $D[v] = +\infty$  :

        Ajouter  $v$  à  $F$ ;  $D[v] \leftarrow D[u] + 1$

**renvoyer**  $D$

► File :

8 4 5



# Parcours en largeur et calcul des distances

**Algorithme :** DISTANCES( $G, s$ )

$F \leftarrow$  file vide;  $D[u] \leftarrow +\infty$  pour tout  $u$

Ajouter  $s$  à  $F$ ;  $D[s] \leftarrow 0$

**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

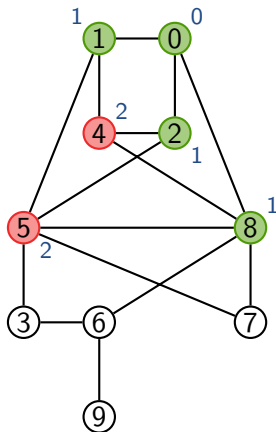
**pour** tout voisin  $v$  de  $u$  tq  $D[v] = +\infty$  :

        Ajouter  $v$  à  $F$ ;  $D[v] \leftarrow D[u] + 1$

**renvoyer**  $D$

► File :

4 5



## Parcours en largeur et calcul des distances

**Algorithme :** DISTANCES( $G, s$ )

$F \leftarrow$  file vide;  $D[u] \leftarrow +\infty$  pour tout  $u$

Ajouter  $s$  à  $F$ ;  $D[s] \leftarrow 0$

**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

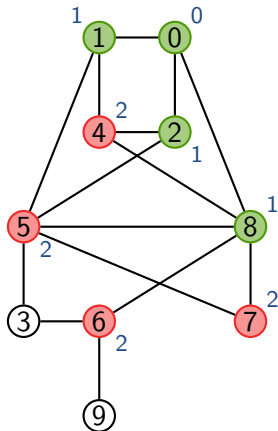
**pour** tout voisin  $v$  de  $u$  tq  $D[v] = +\infty$  :

        Ajouter  $v$  à  $F$ ;  $D[v] \leftarrow D[u] + 1$

**renvoyer**  $D$

► File :

4 5 6 7



## Parcours en largeur et calcul des distances

**Algorithme :** DISTANCES( $G, s$ )

$F \leftarrow$  file vide;  $D[u] \leftarrow +\infty$  pour tout  $u$

Ajouter  $s$  à  $F$ ;  $D[s] \leftarrow 0$

**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

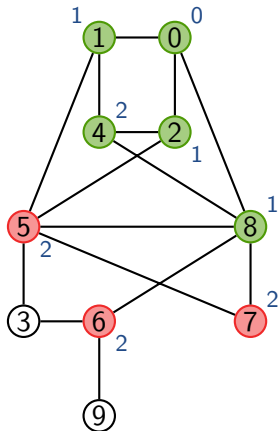
**pour** tout voisin  $v$  de  $u$  tq  $D[v] = +\infty$  :

            Ajouter  $v$  à  $F$ ;  $D[v] \leftarrow D[u] + 1$

**renvoyer**  $D$

► File :

5 6 7



# Parcours en largeur et calcul des distances

**Algorithme :** DISTANCES( $G, s$ )

$F \leftarrow$  file vide;  $D[u] \leftarrow +\infty$  pour tout  $u$

Ajouter  $s$  à  $F$ ;  $D[s] \leftarrow 0$

**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

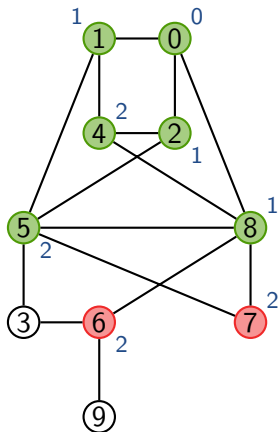
**pour** tout voisin  $v$  de  $u$  tq  $D[v] = +\infty$  :

            Ajouter  $v$  à  $F$ ;  $D[v] \leftarrow D[u] + 1$

**renvoyer**  $D$

► File :

6 7



# Parcours en largeur et calcul des distances

**Algorithme :** DISTANCES( $G, s$ )

$F \leftarrow$  file vide;  $D[u] \leftarrow +\infty$  pour tout  $u$

Ajouter  $s$  à  $F$ ;  $D[s] \leftarrow 0$

**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

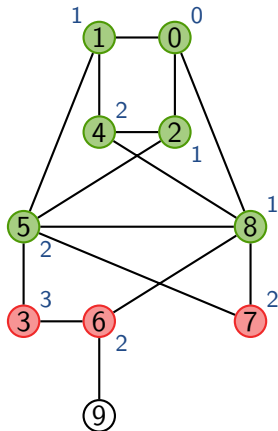
**pour** tout voisin  $v$  de  $u$  tq  $D[v] = +\infty$  :

            Ajouter  $v$  à  $F$ ;  $D[v] \leftarrow D[u] + 1$

**renvoyer**  $D$

► File :

6 7 3



# Parcours en largeur et calcul des distances

**Algorithme :** DISTANCES( $G, s$ )

$F \leftarrow$  file vide;  $D[u] \leftarrow +\infty$  pour tout  $u$

Ajouter  $s$  à  $F$ ;  $D[s] \leftarrow 0$

**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

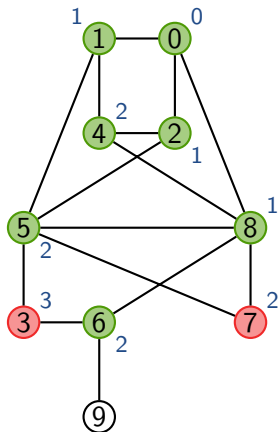
**pour** tout voisin  $v$  de  $u$  tq  $D[v] = +\infty$  :

            Ajouter  $v$  à  $F$ ;  $D[v] \leftarrow D[u] + 1$

**renvoyer**  $D$

► File :

7 3



# Parcours en largeur et calcul des distances

**Algorithme :** DISTANCES( $G, s$ )

$F \leftarrow$  file vide;  $D[u] \leftarrow +\infty$  pour tout  $u$

Ajouter  $s$  à  $F$ ;  $D[s] \leftarrow 0$

**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

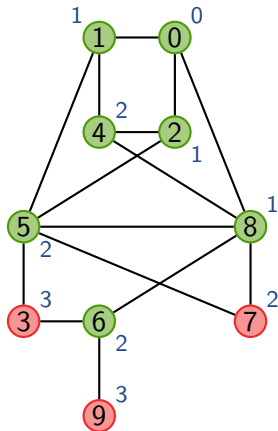
**pour** tout voisin  $v$  de  $u$  tq  $D[v] = +\infty$  :

            Ajouter  $v$  à  $F$ ;  $D[v] \leftarrow D[u] + 1$

**renvoyer**  $D$

► File :

7 3 9





# Parcours en largeur et calcul des distances

**Algorithme :** DISTANCES( $G, s$ )

$F \leftarrow$  file vide;  $D[u] \leftarrow +\infty$  pour tout  $u$

Ajouter  $s$  à  $F$ ;  $D[s] \leftarrow 0$

**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

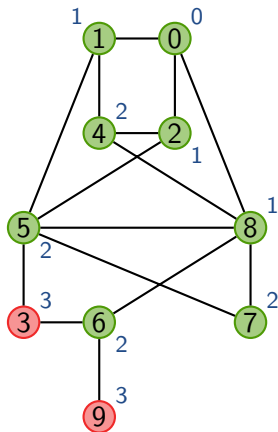
**pour** tout voisin  $v$  de  $u$  tq  $D[v] = +\infty$  :

        Ajouter  $v$  à  $F$ ;  $D[v] \leftarrow D[u] + 1$

**renvoyer**  $D$

► File :

3 9



# Parcours en largeur et calcul des distances

**Algorithme :** DISTANCES( $G, s$ )

$F \leftarrow$  file vide;  $D[u] \leftarrow +\infty$  pour tout  $u$

Ajouter  $s$  à  $F$ ;  $D[s] \leftarrow 0$

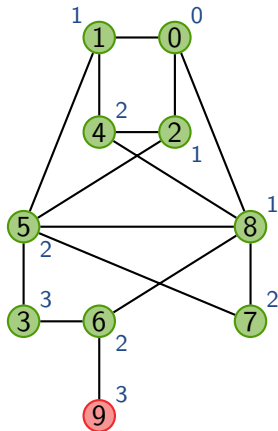
**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

**pour** tout voisin  $v$  de  $u$  tq  $D[v] = +\infty$  :

            Ajouter  $v$  à  $F$ ;  $D[v] \leftarrow D[u] + 1$

**renvoyer**  $D$



► File :

9

# Parcours en largeur et calcul des distances

**Algorithme :** DISTANCES( $G, s$ )

$F \leftarrow$  file vide;  $D[u] \leftarrow +\infty$  pour tout  $u$

Ajouter  $s$  à  $F$ ;  $D[s] \leftarrow 0$

**tant que**  $F$  est non vide :

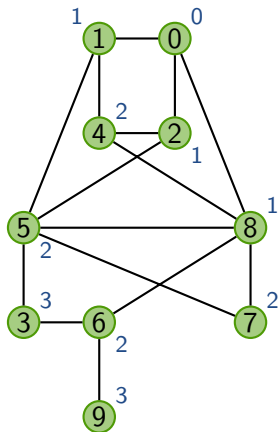
$u \leftarrow$  défiler un élément de  $F$

**pour** tout voisin  $v$  de  $u$  tq  $D[v] = +\infty$  :

            Ajouter  $v$  à  $F$ ;  $D[v] \leftarrow D[u] + 1$

**renvoyer**  $D$

► File :



# Parcours en largeur et calcul des distances

**Algorithme :** DISTANCES( $G, s$ )

$F \leftarrow$  file vide;  $D[u] \leftarrow +\infty$  pour tout  $u$

Ajouter  $s$  à  $F$ ;  $D[s] \leftarrow 0$

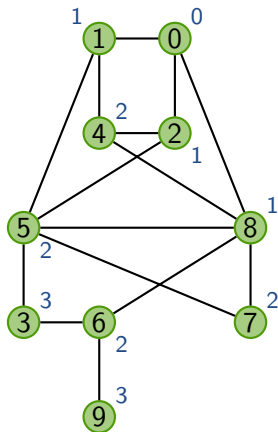
**tant que**  $F$  est non vide :

$u \leftarrow$  défiler un élément de  $F$

**pour** tout voisin  $v$  de  $u$  tq  $D[v] = +\infty$  :

            Ajouter  $v$  à  $F$ ;  $D[v] \leftarrow D[u] + 1$

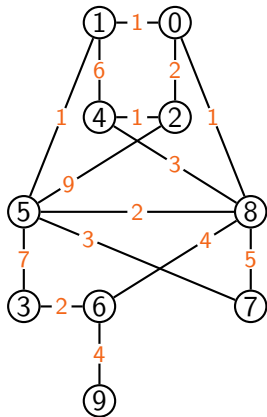
**renvoyer**  $D$



- File :
- DISTANCES calcule les distances entre  $s$  et tous les autres sommets

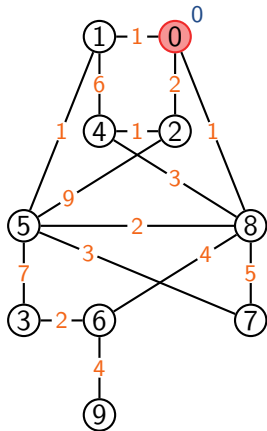
# Algorithme de Dijkstra

Et si le graphe est *pondéré*?



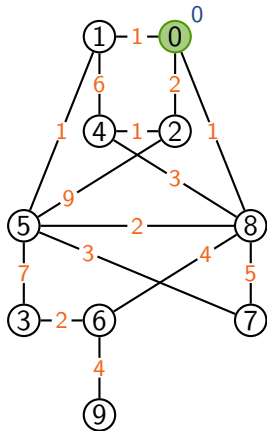
# Algorithme de Dijkstra

Et si le graphe est *pondéré*?



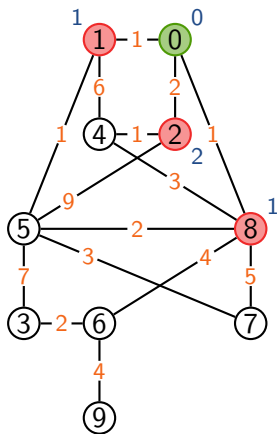
# Algorithme de Dijkstra

Et si le graphe est *pondéré*?



# Algorithme de Dijkstra

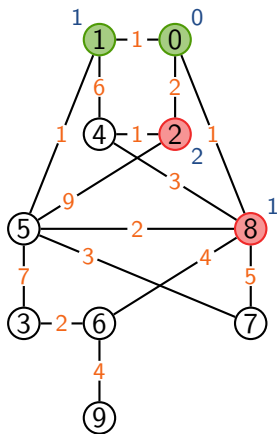
Et si le graphe est *pondéré*?





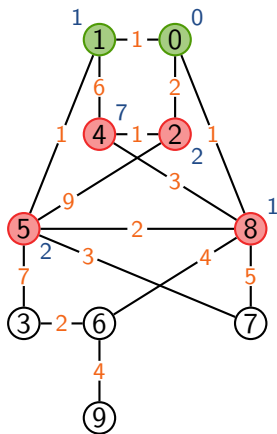
# Algorithme de Dijkstra

Et si le graphe est *pondéré*?



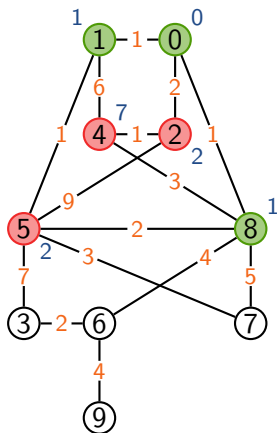
# Algorithme de Dijkstra

Et si le graphe est *pondéré*?



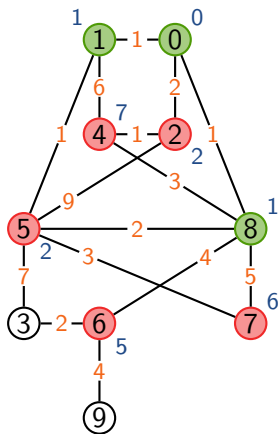
# Algorithme de Dijkstra

Et si le graphe est *pondéré*?



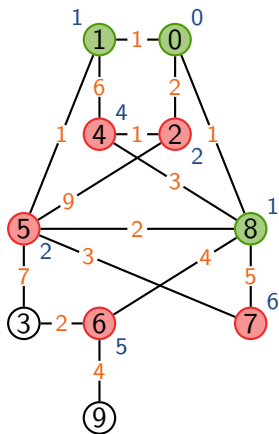
# Algorithme de Dijkstra

Et si le graphe est *pondéré*?



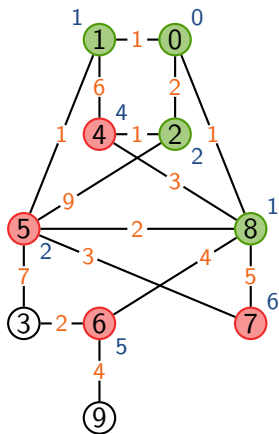
# Algorithme de Dijkstra

Et si le graphe est *pondéré*?



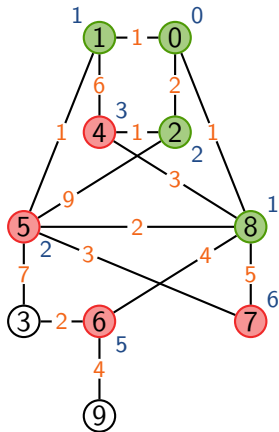
# Algorithme de Dijkstra

Et si le graphe est *pondéré*?



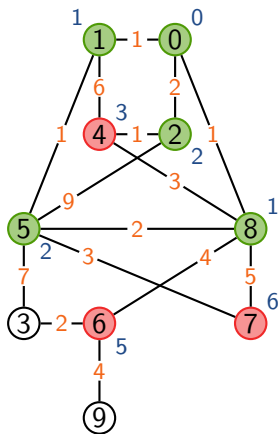
# Algorithme de Dijkstra

Et si le graphe est *pondéré*?



# Algorithme de Dijkstra

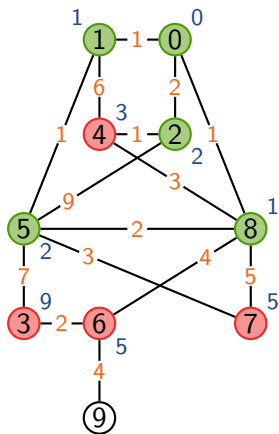
Et si le graphe est *pondéré*?





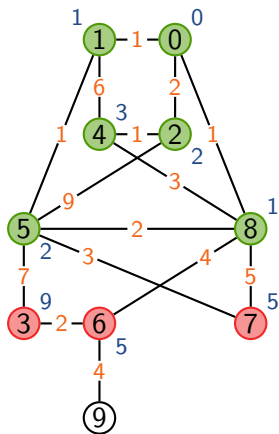
# Algorithme de Dijkstra

Et si le graphe est *pondéré*?



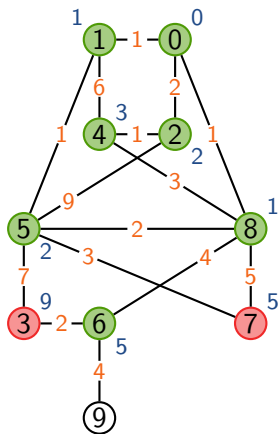
# Algorithme de Dijkstra

Et si le graphe est *pondéré*?



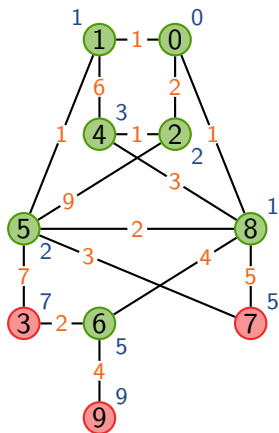
# Algorithme de Dijkstra

Et si le graphe est *pondéré*?



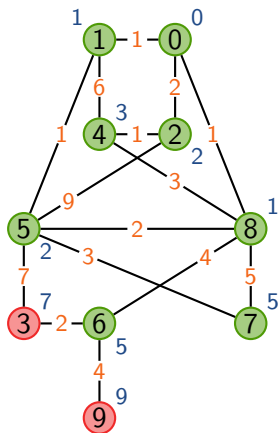
# Algorithme de Dijkstra

Et si le graphe est *pondéré*?



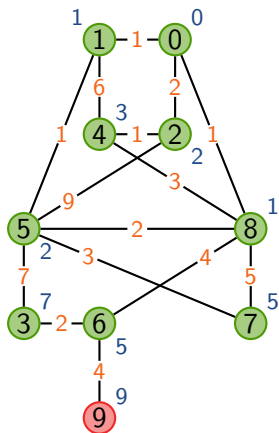
# Algorithme de Dijkstra

Et si le graphe est *pondéré*?



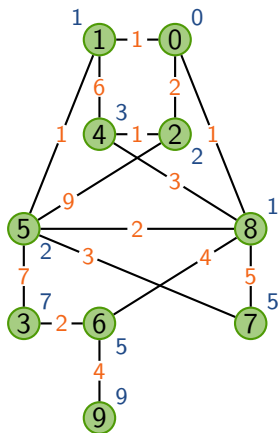
# Algorithme de Dijkstra

Et si le graphe est *pondéré*?



# Algorithme de Dijkstra

Et si le graphe est *pondéré*?



# Algorithme de Dijkstra

Et si le graphe est *pondéré*?

**Algorithme :** DISTANCES( $G, s$ )

$F \leftarrow$  file vide

$D[u] \leftarrow +\infty$  pour tout  $u$

Ajouter  $s$  à  $F$  ;  $D[s] \leftarrow 0$

**tant que**  $F$  est non vide :

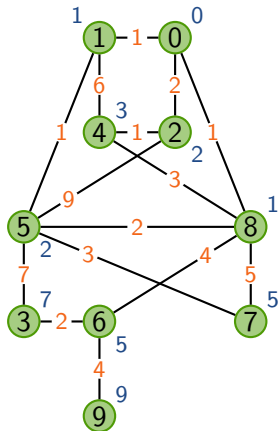
$u \leftarrow$  défiler un élément de  $F$

**pour** tout voisin  $v$  de  $u$  tq  $D[v] = +\infty$  :

        Ajouter  $v$  à  $F$

$D[v] \leftarrow D[u] + 1$

**renvoyer**  $D$





# Algorithme de Dijkstra

Et si le graphe est *pondéré*?

**Algorithme :** DIJKSTRA( $G, s, p$ )

$F \leftarrow$  file de priorité vide

$D[u] \leftarrow +\infty$  pour tout  $u$

Ajouter  $s$  à  $F$  ;  $D[s] \leftarrow 0$

**tant que**  $F$  est non vide :

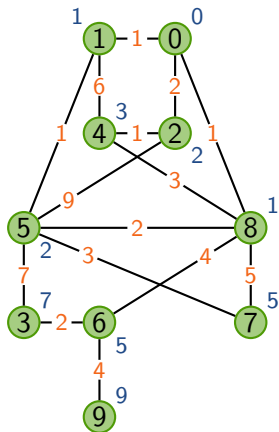
$u \leftarrow$  défiler un élément de  $F$  de distance minimale

**pour** tout voisin  $v$  de  $u$  :

        Ajouter  $v$  à  $F$  **si**  $D[v] = +\infty$

$D[v] \leftarrow \min(D[v], D[u] + p(u, v))$

**renvoyer**  $D$



## L'algorithme en détail

**Algorithme : DIJKSTRA( $G, s, p$ )**

$F \leftarrow$  file de priorité vide

```
D ← tableau de  $n$  entiers, initialisé à  $+\infty$  // Tableau des distances
```

**pour** *chaque* sommet  $u$  de  $G$  : AJOUTER( $F, u, +\infty$ )    // Insertion dans la file

CHANGERPRIORITÉ( $F, s, 0$ );  $D_{[s]} \leftarrow 0$  // Distance 0 pour  $s$

**tant que  $F$  est non vide :**

```

u ← EXTRAIREMIN(F)           // Élément de distance minimale

```

**pour tout voisin  $v$  de  $u$  :**

**si**  $D_{[u]} + p(u, v) < D_{[v]}$  :

$$D_{[v]} \leftarrow D_{[u]} + p(u, v) \quad // \text{ Mise à jour de la distance}$$
CHANGERPRIORITÉ( $F, v, D_{[v]}$ ) // Mise à jour de la filerenvoyer  $D$

# Propriétés de l'algorithme de Dijkstra

## Théorème

*Si  $G$  est connexe,  $\text{DIJKTRA}(G, s, p)$  calcule les longueurs des plus courts chemins de  $s$  à chaque sommet de  $G$ . Sa complexité est*

- ▶  *$O(m \log n)$  avec des listes d'adjacence*
- ▶  *$O(n^2)$  avec une matrice d'adjacence*

*où  $n$  est le nombre de sommets,  $m$  le nombre d'arêtes.*

# Propriétés de l'algorithme de Dijkstra

## Théorème

*Si  $G$  est connexe,  $\text{DIJKTRA}(G, s, p)$  calcule les longueurs des plus courts chemins de  $s$  à chaque sommet de  $G$ . Sa complexité est*

- ▶  $O(m \log n)$  avec des listes d'adjacence
- ▶  $O(n^2)$  avec une matrice d'adjacence

*où  $n$  est le nombre de sommets,  $m$  le nombre d'arêtes.*

## Preuve de complexité

- ▶ Listes d'adjacence :
  - ▶ Liste de priorité : AJOUTER/EXTRAIREMIN/CHANGERPRIORITÉ en  $O(\log n)$
  - ▶ Chaque sommet est extrait une fois  $\rightsquigarrow O(n \log n)$
  - ▶ Chaque arête *peut* déclencher un CHANGERPRIORITÉ  $\rightsquigarrow O(m \log n)$

# Propriétés de l'algorithme de Dijkstra

## Théorème

Si  $G$  est connexe,  $\text{DIJKSTRA}(G, s, p)$  calcule les longueurs des plus courts chemins de  $s$  à chaque sommet de  $G$ . Sa complexité est

- ▶  $O(m \log n)$  avec des listes d'adjacence
- ▶  $O(n^2)$  avec une matrice d'adjacence

où  $n$  est le nombre de sommets,  $m$  le nombre d'arêtes.

## Preuve de complexité

- ▶ Listes d'adjacence :
  - ▶ Liste de priorité : AJOUTER/EXTRAIREMIN/CHANGERPRIORITÉ en  $O(\log n)$
  - ▶ Chaque sommet est extrait une fois  $\rightsquigarrow O(n \log n)$
  - ▶ Chaque arête *peut* déclencher un CHANGERPRIORITÉ  $\rightsquigarrow O(m \log n)$
- ▶ Matrices d'adjacence
  - ▶ Pas de liste de priorité : parcours de tous les sommets à chaque fois
  - ▶ « pour tout voisin  $v$  de  $u$  »  $\rightsquigarrow n \times O(n)$



# Propriétés de l'algorithme de Dijkstra

## Théorème

*Si  $G$  est connexe,  $\text{DIJKTRA}(G, s, p)$  calcule les longueurs des plus courts chemins de  $s$  à chaque sommet de  $G$ . Sa complexité est*

- ▶  $O(m \log n)$  avec des listes d'adjacence
- ▶  $O(n^2)$  avec une matrice d'adjacence

*où  $n$  est le nombre de sommets,  $m$  le nombre d'arêtes.*

**Preuve de correction.** On note  $d_u$  la distance minimale de  $u$  à  $s$

Invariant : « quand  $u$  est extrait de  $F$  à l'itération  $k$ ,  $D_{[u]} = d_u$  »

- ▶  $k = 1$  : extraction de  $s$  et  $D_{[s]} = d_s = 0 \rightsquigarrow \text{OK}$

# Propriétés de l'algorithme de Dijkstra

## Théorème

Si  $G$  est connexe,  $\text{DIJKTRA}(G, s, p)$  calcule les longueurs des plus courts chemins de  $s$  à chaque sommet de  $G$ . Sa complexité est

- ▶  $O(m \log n)$  avec des listes d'adjacence
- ▶  $O(n^2)$  avec une matrice d'adjacence

où  $n$  est le nombre de sommets,  $m$  le nombre d'arêtes.

**Preuve de correction.** On note  $d_u$  la distance minimale de  $u$  à  $s$

Invariant : « quand  $u$  est extrait de  $F$  à l'itération  $k$ ,  $D_{[u]} = d_u$  »

- ▶  $k = 1$  : extraction de  $s$  et  $D_{[s]} = d_s = 0 \rightsquigarrow \text{OK}$
- ▶ Soit  $u$  le  $k^{\text{ème}}$  sommet extrait,  $k > 0$ , et  $d = D_{[u]}$  à ce moment-là
  - ▶ Supposons qu'il existe un chemin de longueur  $d_u < d$  de  $s$  à  $u$
  - ▶ Il existe une arête  $xy$  sur ce chemin tq  $x$  est extrait avant l'itération  $k$  mais pas  $y$
  - ▶ À l'itération  $k$ ,  $D_{[x]} = d_x$  et comme  $y$  est voisin de  $x$ ,  $D_{[y]} \leq d_x + p(x, y)$
  - ▶ Mais  $d_x + p(x, y) \leq d_u < d$  :  $y$  devrait avoir été choisi avant  $u$



## Algorithme de Dijkstra : reconstruction

**Algorithme** : DIJKSTRA( $G, s, p$ )

$F \leftarrow$  file de priorité vide

$D \leftarrow$  tableau de  $n$  entiers, initialisé à  $+\infty$

$P \leftarrow$  tableau de  $n$  entiers // Prédecesseurs

**pour** chaque sommet  $u$  de  $G$  : AJOUTER( $F, u, +\infty$ )

CHANGERPRIORITÉ( $F, s, 0$ );  $D[s] \leftarrow 0$

**tant que**  $F$  est non vide :

$u \leftarrow$  EXTRAIREMIN( $F$ )

**pour** tout voisin  $v$  de  $u$  :

**si**  $D[u] + p(u, v) < D[v]$  :

$D[v] \leftarrow D[u] + p(u, v)$ ;  $P[v] \leftarrow u$

            CHANGERPRIORITÉ( $F, v, D[v]$ )

**renvoyer**  $D$  et  $P$



## Algorithme de Dijkstra : reconstruction

**Algorithme** : DIJKSTRA( $G, s, p$ )

$F \leftarrow$  file de priorité vide

$D \leftarrow$  tableau de  $n$  entiers, initialisé à  $+\infty$

$P \leftarrow$  tableau de  $n$  entiers // Prédecesseurs

**pour** chaque sommet  $u$  de  $G$  : AJOUTER( $F, u, +\infty$ )

CHANGERPRIORITÉ( $F, s, 0$ );  $D[s] \leftarrow 0$

**tant que**  $F$  est non vide :

$u \leftarrow$  EXTRAIREMIN( $F$ )

**pour** tout voisin  $v$  de  $u$  :

**si**  $D[u] + p(u, v) < D[v]$  :

$D[v] \leftarrow D[u] + p(u, v)$ ;  $P[v] \leftarrow u$

            CHANGERPRIORITÉ( $F, v, D[v]$ )

**renvoyer**  $D$  et  $P$

**Algorithme** : DIJKSTRA\_REC( $G, P, u$ )

$C \leftarrow \{u\}$  // Chemin

**tant que**  $u \neq s$  :

$u \leftarrow P[u]$

    Ajouter  $u$  à  $C$

**renvoyer**  $C$

## Algorithme de Dijkstra : reconstruction

**Algorithme** : DIJKSTRA( $G, s, p$ )

$F \leftarrow$  file de priorité vide

$D \leftarrow$  tableau de  $n$  entiers, initialisé à  $+\infty$

$P \leftarrow$  tableau de  $n$  entiers // Prédecesseurs

**pour** chaque sommet  $u$  de  $G$  : AJOUTER( $F, u, +\infty$ )

CHANGERPRIORITÉ( $F, s, 0$ );  $D[s] \leftarrow 0$

**tant que**  $F$  est non vide :

$u \leftarrow$  EXTRAIREMIN( $F$ )

**pour** tout voisin  $v$  de  $u$  :

**si**  $D[u] + p(u, v) < D[v]$  :

$D[v] \leftarrow D[u] + p(u, v)$ ;  $P[v] \leftarrow u$

            CHANGERPRIORITÉ( $F, v, D[v]$ )

**renvoyer**  $D$  et  $P$

**Algorithme** : DIJKSTRA\_REC( $G, P, u$ )

$C \leftarrow \{u\}$  // Chemin

**tant que**  $u \neq s$  :

$u \leftarrow P[u]$

    Ajouter  $u$  à  $C$

**renvoyer**  $C$

**Algorithme** : DIJKSTRA\_ARBRE( $G, P$ )

$T \leftarrow$  arbre vide

**pour** tout sommet  $u \neq s$  :

    Ajouter l'arête  $P[u]u$  à  $T$

**renvoyer**  $T$

# Calcul de plus court chemins

## Théorème

L'algorithme DIJKSTRA\_ARBRE construit un **arbre des plus courts chemins** depuis  $s$ , en temps  $O(n)$ .

**Preuve (idée) :** Chaque sommet a un unique prédecesseur  $\rightsquigarrow$  arbre

# Calcul de plus court chemins

## Théorème

L'algorithme DIJKSTRA\_ARBRE construit un **arbre des plus courts chemins** depuis  $s$ , en temps  $O(n)$ .

Preuve (idée) : Chaque sommet a un unique prédecesseur  $\rightsquigarrow$  arbre

## Pour aller plus loin

- ▶ Si arêtes de poids négatif (*raccourcis*), mais pas de cycle strictement négatif
  - ▶ Algorithme de Bellman-Ford (adaptation de DIJKSTRA)

# Calcul de plus court chemins

## Théorème

L'algorithme DIJKSTRA\_ARBRE construit un **arbre des plus courts chemins** depuis  $s$ , en temps  $O(n)$ .

Preuve (idée) : Chaque sommet a un unique prédecesseur  $\rightsquigarrow$  arbre

## Pour aller plus loin

- ▶ Si arêtes de poids négatif (*raccourcis*), mais pas de cycle strictement négatif
  - ▶ Algorithme de Bellman-Ford (adaptation de DIJKSTRA)
- ▶ Chemins les plus courts entre toute paire de sommets
  - ▶  $n$  répétitions de DIJKSTRA  $\rightsquigarrow O(mn \log n)$  ou  $O(n^3)$
  - ▶ Algorithme de Floyd-Warshall  $\rightsquigarrow$  TD

# Calcul de plus court chemins

## Théorème

L'algorithme DIJKSTRA\_ARBRE construit un **arbre des plus courts chemins** depuis  $s$ , en temps  $O(n)$ .

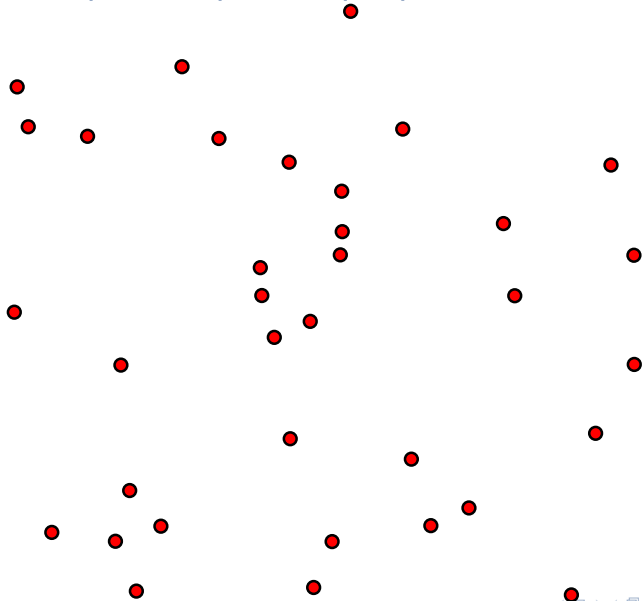
**Preuve (idée) :** Chaque sommet a un unique prédecesseur  $\rightsquigarrow$  arbre

## Pour aller plus loin

- ▶ Si arêtes de poids négatif (*raccourcis*), mais pas de cycle strictement négatif
  - ▶ Algorithme de Bellman-Ford (adaptation de DIJKSTRA)
- ▶ Chemins les plus courts entre toute paire de sommets
  - ▶  $n$  répétitions de DIJKSTRA  $\rightsquigarrow O(mn \log n)$  ou  $O(n^3)$
  - ▶ Algorithme de Floyd-Warshall  $\rightsquigarrow$  TD
- ▶ En pratique (ex. cartes routières) : algorithme  $A^*$ 
  - ▶ distance par la route  $\geq$  distance à vol d'oiseau (calcul facile)
  - ▶ certains sommets peuvent être ignorés  $\rightsquigarrow$  calcul plus rapide !

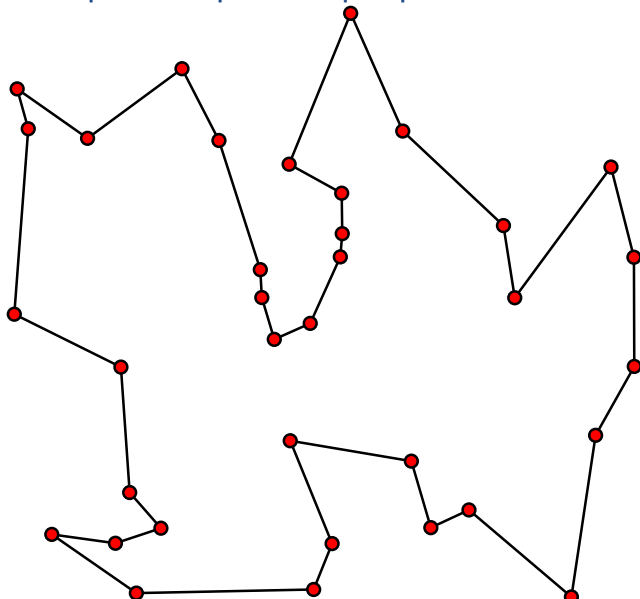
1. Premier exemple : plus longue sous-suite croissante
2. Qu'est-ce que la programmation dynamique ?
3. Deuxième exemple : choix de cours, le retour
4. Troisième exemple : la distance d'édition
5. Quatrième exemple : calcul de plus courts chemins
6. Bonus : le voyageur de commerce

Chemin le plus court passant par chaque point ?

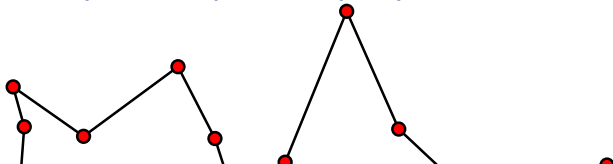




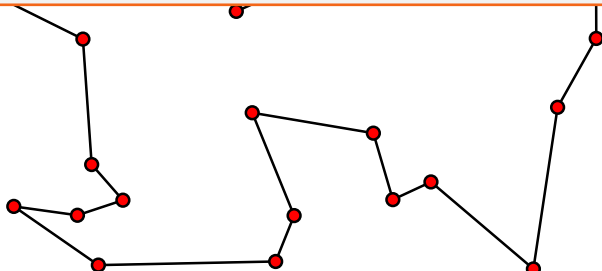
Chemin le plus court passant par chaque point ?



Chemin le plus court passant par chaque point ?



- ▶ Algorithme naïf : essayer toutes les permutations  $\rightsquigarrow O(n!)$







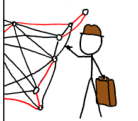
# Chemin le plus court passant par chaque point ?

- ▶ Algorithme naïf :  $O(n!)$
  - ▶ Algorithme polynôme :  $O(n^2 2^n)$
- Objectif : algorithme**

BRUTE-FORCE  
SOLUTION:  
 $O(n!)$



DYNAMIC  
PROGRAMMING  
ALGORITHMS:  
 $O(n^2 2^n)$



SELLING ON EBAY:  
 $O(1)$

STILL WORKING  
ON YOUR ROUTE?

SHUT THE  
HELL UP.



<https://xkcd.com/399/>

# Formalisation

**Entrée** Ensemble  $S = \{s_0, \dots, s_{n-1}\}$  de points dans le plan

**Sortie 1** Chemin  $s_{i_0} \rightarrow s_{i_1} \rightarrow \dots \rightarrow s_{i_{n-1}} \rightarrow s_{i_0}$  qui minimise la distance totale, avec  $\delta_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$  si  $s_i = (x_i, y_i)$  et  $s_j = (x_j, y_j)$

**Sortie 2** Longueur  $\ell_S = \sum_{j=0}^{n-1} \delta_{i_j i_{j+1}}$  du chemin le plus court ( $i_n = i_0$ )

# Formalisation

**Entrée** Ensemble  $S = \{s_0, \dots, s_{n-1}\}$  de points dans le plan

**Sortie 1** Chemin  $s_{i_0} \rightarrow s_{i_1} \rightarrow \dots \rightarrow s_{i_{n-1}} \rightarrow s_{i_0}$  qui minimise la distance totale, avec  $\delta_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$  si  $s_i = (x_i, y_i)$  et  $s_j = (x_j, y_j)$

**Sortie 2** Longueur  $\ell_S = \sum_{j=0}^{n-1} \delta_{i_j i_{j+1}}$  du chemin le plus court ( $i_n = i_0$ )

► On peut fixer  $s_{i_0} = s_0$

# Formalisation

**Entrée** Ensemble  $S = \{s_0, \dots, s_{n-1}\}$  de points dans le plan

**Sortie 1** Chemin  $s_{i_0} \rightarrow s_{i_1} \rightarrow \dots \rightarrow s_{i_{n-1}} \rightarrow s_{i_0}$  qui minimise la distance totale, avec  $\delta_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$  si  $s_i = (x_i, y_i)$  et  $s_j = (x_j, y_j)$

**Sortie 2** Longueur  $\ell_S = \sum_{j=0}^{n-1} \delta_{i_j i_{j+1}}$  du chemin le plus court ( $i_n = i_0$ )

- ▶ On peut fixer  $s_{i_0} = s_0$
- ▶ Si  $U \subset S$  avec  $s_0, s_j \in U$ , on note  $\Delta(U, s_j)$  la longueur du plus court chemin de  $s_0$  à  $s_j$  visitant chaque  $s_i \in U$  une fois exactement



# Formalisation

**Entrée** Ensemble  $S = \{s_0, \dots, s_{n-1}\}$  de points dans le plan

**Sortie 1** Chemin  $s_{i_0} \rightarrow s_{i_1} \rightarrow \dots \rightarrow s_{i_{n-1}} \rightarrow s_{i_0}$  qui minimise la distance totale, avec  $\delta_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$  si  $s_i = (x_i, y_i)$  et  $s_j = (x_j, y_j)$

**Sortie 2** Longueur  $\ell_S = \sum_{j=0}^{n-1} \delta_{i_j i_{j+1}}$  du chemin le plus court ( $i_n = i_0$ )

- ▶ On peut fixer  $s_{i_0} = s_0$
- ▶ Si  $U \subset S$  avec  $s_0, s_j \in U$ , on note  $\Delta(U, s_j)$  la longueur du plus court chemin de  $s_0$  à  $s_j$  visitant chaque  $s_i \in U$  une fois exactement

$$\ell_S = \min_j \Delta(\{s_0, \dots, s_{n-1}\}, s_j) + \delta_{j,0}$$

## Formule récursive pour $\Delta$

$\Delta(U, s_j)$  : longueur du plus court chemin de  $s_0$  à  $s_j$  visitant chaque  $s_i \in U$  une fois exactement

## Formule récursive pour $\Delta$

$\Delta(U, s_j)$  : longueur du plus court chemin de  $s_0$  à  $s_j$  visitant chaque  $s_i \in U$  une fois exactement

### Lemme

- ▶  $\Delta(\{s_0\}, s_0) = 0$  et  $\Delta(U, s_0) = +\infty$  si  $|U| > 1$
- ▶  $\Delta(U, s_j) = \min_{i \in U: i \neq j} \Delta(U \setminus \{s_j\}, s_i) + \delta_{ij}$

## Formule récursive pour $\Delta$

$\Delta(U, s_j)$  : longueur du plus court chemin de  $s_0$  à  $s_j$  visitant chaque  $s_i \in U$  une fois exactement

### Lemme

- ▶  $\Delta(\{s_0\}, s_0) = 0$  et  $\Delta(U, s_0) = +\infty$  si  $|U| > 1$
- ▶  $\Delta(U, s_j) = \min_{i \in U: i \neq j} \Delta(U \setminus \{s_j\}, s_i) + \delta_{ij}$

**Preuve :** pour aller de  $s_0$  à  $s_j$ ,

- ▶ on choisit un point  $s_i$
- ▶ on va de  $s_0$  à  $s_i$
- ▶ puis directement de  $s_i$  à  $s_j$

## Formule récursive pour $\Delta$

$\Delta(U, s_j)$  : longueur du plus court chemin de  $s_0$  à  $s_j$  visitant chaque  $s_i \in U$  une fois exactement

### Lemme

- ▶  $\Delta(\{s_0\}, s_0) = 0$  et  $\Delta(U, s_0) = +\infty$  si  $|U| > 1$
- ▶  $\Delta(U, s_j) = \min_{i \in U: i \neq j} \Delta(U \setminus \{s_j\}, s_i) + \delta_{ij}$

**Preuve :** pour aller de  $s_0$  à  $s_j$ ,

- ▶ on choisit un point  $s_i$
- ▶ on va de  $s_0$  à  $s_i$
- ▶ puis directement de  $s_i$  à  $s_j$   
 $\rightsquigarrow$  il *suffit* de trouver le meilleur  $s_i$  !

## Algorithme TSP

**Algorithme :** TSP( $S$ )

$\Delta \leftarrow$  tableau à deux dimensions, indexé par les sous-ensembles de  $S$  contenant  $\{s_0\}$ , et par les sommets  $s_0$  à  $s_{n-1}$

$\Delta_{[\{s_0\}, s_0]} = 0$

**pour**  $t = 2$  à  $n$  :

**pour** tous les  $U \subset S$  de taille  $t$  tels que  $s_0 \in U$  :

$\Delta_{[U, s_0]} = +\infty$

**pour** tout  $s_j \in U, j \neq 0$  :

$\Delta_{[U, s_j]} = \min\{\Delta_{[U \setminus \{s_j\}, s_i]} + \delta_{ij} : s_i \in U, i \neq j\}$

**renvoyer**  $\min_j (\Delta_{[\{s_0, \dots, s_n\}, s_j]} + \delta_{j0})$

# Algorithme TSP

## Algorithme : $\text{TSP}(S)$

$\Delta \leftarrow$  tableau à deux dimensions, indexé par les sous-ensembles de  $S$  contenant  $\{s_0\}$ , et par les sommets  $s_0$  à  $s_{n-1}$

$$\Delta_{[\{s_0\}, s_0]} = 0$$

**pour**  $t = 2$  à  $n$  :

**pour** tous les  $U \subset S$  de taille  $t$  tels que  $s_0 \in U$  :

$$\Delta_{[U, s_0]} = +\infty$$

**pour** tout  $s_j \in U, j \neq 0$  :

$$\Delta_{[U, s_j]} = \min\{\Delta_{[U \setminus \{s_j\}, s_i]} + \delta_{ij} : s_i \in U, i \neq j\}$$

**renvoyer**  $\min_j (\Delta_{[\{s_0, \dots, s_n\}, s_j]} + \delta_{j0})$

## Lemme

*L'algorithme TSP calcule  $\ell_S$  en temps  $O(n^2 2^n)$*

# Preuve de la complexité

```
1  Algorithme : TSP( $S$ )
2   $\Delta \leftarrow$  tableau à deux dimensions, indexé par les sous-ensembles de
3       $S$  contenant  $\{s_0\}$ , et par les sommets  $s_0$  à  $s_{n-1}$ 
4   $\Delta_{[\{s_0\}, s_0]} = 0$ 
5  pour  $t = 2$  à  $n$  :
6      pour tous les  $U \subset S$  de taille  $t$  tels que  $s_0 \in U$  :
7           $\Delta_{[U, s_0]} = +\infty$ 
8          pour tout  $s_j \in U, j \neq 0$  :
9               $\Delta_{[U, s_j]} = \min\{\Delta_{[U \setminus \{s_j\}, s_i]} + \delta_{ij} : s_i \in U, i \neq j\}$ 
10 renvoyer  $\min_j (\Delta_{[\{s_0, \dots, s_n\}, s_j]} + \delta_{j0})$ 
```

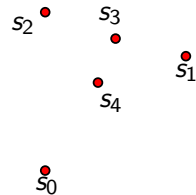
## Preuve

- Calcul du min :  $O(t)$  car  $|U| = t$
- Boucle sur  $s_j$  :  $O(t^2)$
- Boucle sur  $U$  :  $\binom{n-1}{t-1} O(t^2)$  car  $\binom{n-1}{t-1}$  sous-ensembles
- Boucle sur  $t$  :  $\sum_{t=2}^n \binom{n-1}{t-1} O(t^2) \leq O(n^2) \sum_t \binom{n}{t} = O(n^2 2^n)$



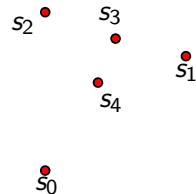
# Example

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$				
$\{s_0, s_2\}$	$+\infty$				
$\{s_0, s_3\}$	$+\infty$				
$\{s_0, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2\}$	$+\infty$				
$\{s_0, s_1, s_3\}$	$+\infty$				
$\{s_0, s_1, s_4\}$	$+\infty$				
$\{s_0, s_2, s_3\}$	$+\infty$				
$\{s_0, s_2, s_4\}$	$+\infty$				
$\{s_0, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3\}$	$+\infty$				
$\{s_0, s_1, s_2, s_4\}$	$+\infty$				
$\{s_0, s_1, s_3, s_4\}$	$+\infty$				
$\{s_0, s_2, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$				
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$					



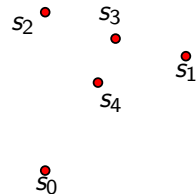
# Example

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$				
$\{s_0, s_3\}$	$+\infty$				
$\{s_0, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2\}$	$+\infty$				
$\{s_0, s_1, s_3\}$	$+\infty$				
$\{s_0, s_1, s_4\}$	$+\infty$				
$\{s_0, s_2, s_3\}$	$+\infty$				
$\{s_0, s_2, s_4\}$	$+\infty$				
$\{s_0, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3\}$	$+\infty$				
$\{s_0, s_1, s_2, s_4\}$	$+\infty$				
$\{s_0, s_1, s_3, s_4\}$	$+\infty$				
$\{s_0, s_2, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$				
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$					



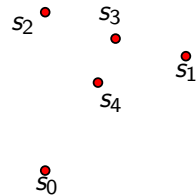
# Example

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$				
$\{s_0, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2\}$	$+\infty$				
$\{s_0, s_1, s_3\}$	$+\infty$				
$\{s_0, s_1, s_4\}$	$+\infty$				
$\{s_0, s_2, s_3\}$	$+\infty$				
$\{s_0, s_2, s_4\}$	$+\infty$				
$\{s_0, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3\}$	$+\infty$				
$\{s_0, s_1, s_2, s_4\}$	$+\infty$				
$\{s_0, s_1, s_3, s_4\}$	$+\infty$				
$\{s_0, s_2, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$				
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$					



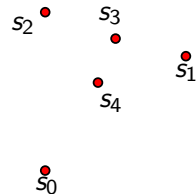
# Example

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2\}$	$+\infty$				
$\{s_0, s_1, s_3\}$	$+\infty$				
$\{s_0, s_1, s_4\}$	$+\infty$				
$\{s_0, s_2, s_3\}$	$+\infty$				
$\{s_0, s_2, s_4\}$	$+\infty$				
$\{s_0, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3\}$	$+\infty$				
$\{s_0, s_1, s_2, s_4\}$	$+\infty$				
$\{s_0, s_1, s_3, s_4\}$	$+\infty$				
$\{s_0, s_2, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$				
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$					



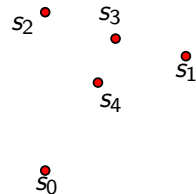
# Example

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				11, 7
$\{s_0, s_1, s_2\}$	$+\infty$				
$\{s_0, s_1, s_3\}$	$+\infty$				
$\{s_0, s_1, s_4\}$	$+\infty$				
$\{s_0, s_2, s_3\}$	$+\infty$				
$\{s_0, s_2, s_4\}$	$+\infty$				
$\{s_0, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3\}$	$+\infty$				
$\{s_0, s_1, s_2, s_4\}$	$+\infty$				
$\{s_0, s_1, s_3, s_4\}$	$+\infty$				
$\{s_0, s_2, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$				
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$					



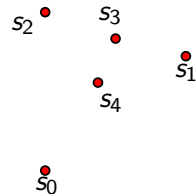
# Example

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				11, 7
$\{s_0, s_1, s_2\}$	$+\infty$	34, 7			
$\{s_0, s_1, s_3\}$	$+\infty$				
$\{s_0, s_1, s_4\}$	$+\infty$				
$\{s_0, s_2, s_3\}$	$+\infty$				
$\{s_0, s_2, s_4\}$	$+\infty$				
$\{s_0, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3\}$	$+\infty$				
$\{s_0, s_1, s_2, s_4\}$	$+\infty$				
$\{s_0, s_1, s_3, s_4\}$	$+\infty$				
$\{s_0, s_2, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$				
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$					



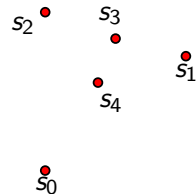
# Example

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				11, 7
$\{s_0, s_1, s_2\}$	$+\infty$	34, 7	37, 4		
$\{s_0, s_1, s_3\}$	$+\infty$				
$\{s_0, s_1, s_4\}$	$+\infty$				
$\{s_0, s_2, s_3\}$	$+\infty$				
$\{s_0, s_2, s_4\}$	$+\infty$				
$\{s_0, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3\}$	$+\infty$				
$\{s_0, s_1, s_2, s_4\}$	$+\infty$				
$\{s_0, s_1, s_3, s_4\}$	$+\infty$				
$\{s_0, s_2, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$				
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$					



# Example

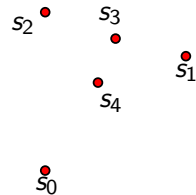
	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				11, 7
$\{s_0, s_1, s_2\}$	$+\infty$	34, 7	37, 4		
$\{s_0, s_1, s_3\}$	$+\infty$	25, 2		28, 9	
$\{s_0, s_1, s_4\}$	$+\infty$				
$\{s_0, s_2, s_3\}$	$+\infty$				
$\{s_0, s_2, s_4\}$	$+\infty$				
$\{s_0, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3\}$	$+\infty$				
$\{s_0, s_1, s_2, s_4\}$	$+\infty$				
$\{s_0, s_1, s_3, s_4\}$	$+\infty$				
$\{s_0, s_2, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$				
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$					





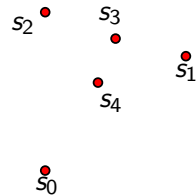
# Example

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				11, 7
$\{s_0, s_1, s_2\}$	$+\infty$	34, 7	37, 4		
$\{s_0, s_1, s_3\}$	$+\infty$	25, 2		28, 9	
$\{s_0, s_1, s_4\}$	$+\infty$	22, 1			31, 1
$\{s_0, s_2, s_3\}$	$+\infty$				
$\{s_0, s_2, s_4\}$	$+\infty$				
$\{s_0, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3\}$	$+\infty$				
$\{s_0, s_1, s_2, s_4\}$	$+\infty$				
$\{s_0, s_1, s_3, s_4\}$	$+\infty$				
$\{s_0, s_2, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$				
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$					



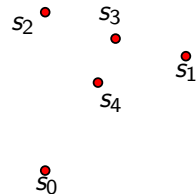
# Example

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				11, 7
$\{s_0, s_1, s_2\}$	$+\infty$	34, 7	37, 4		
$\{s_0, s_1, s_3\}$	$+\infty$	25, 2		28, 9	
$\{s_0, s_1, s_4\}$	$+\infty$	22, 1			31, 1
$\{s_0, s_2, s_3\}$	$+\infty$		25, 5	26, 5	
$\{s_0, s_2, s_4\}$	$+\infty$				
$\{s_0, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3\}$	$+\infty$				
$\{s_0, s_1, s_2, s_4\}$	$+\infty$				
$\{s_0, s_1, s_3, s_4\}$	$+\infty$				
$\{s_0, s_2, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$				
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$					



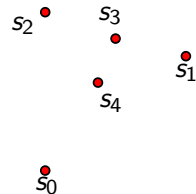
# Exemple

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				11, 7
$\{s_0, s_1, s_2\}$	$+\infty$	34, 7	37, 4		
$\{s_0, s_1, s_3\}$	$+\infty$	25, 2		28, 9	
$\{s_0, s_1, s_4\}$	$+\infty$	22, 1			31, 1
$\{s_0, s_2, s_3\}$	$+\infty$		25, 5	26, 5	
$\{s_0, s_2, s_4\}$	$+\infty$		21, 7		28
$\{s_0, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3\}$	$+\infty$				
$\{s_0, s_1, s_2, s_4\}$	$+\infty$				
$\{s_0, s_1, s_3, s_4\}$	$+\infty$				
$\{s_0, s_2, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$				
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$					



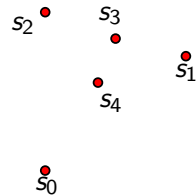
# Exemple

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				11, 7
$\{s_0, s_1, s_2\}$	$+\infty$	34, 7	37, 4		
$\{s_0, s_1, s_3\}$	$+\infty$	25, 2		28, 9	
$\{s_0, s_1, s_4\}$	$+\infty$	22, 1			31, 1
$\{s_0, s_2, s_3\}$	$+\infty$		25, 5	26, 5	
$\{s_0, s_2, s_4\}$	$+\infty$		21, 7		28
$\{s_0, s_3, s_4\}$	$+\infty$			17, 0	22, 4
$\{s_0, s_1, s_2, s_3\}$	$+\infty$				
$\{s_0, s_1, s_2, s_4\}$	$+\infty$				
$\{s_0, s_1, s_3, s_4\}$	$+\infty$				
$\{s_0, s_2, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$				
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$					



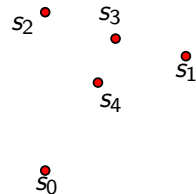
# Exemple

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				11, 7
$\{s_0, s_1, s_2\}$	$+\infty$	34, 7	37, 4		
$\{s_0, s_1, s_3\}$	$+\infty$	25, 2		28, 9	
$\{s_0, s_1, s_4\}$	$+\infty$	22, 1			31, 1
$\{s_0, s_2, s_3\}$	$+\infty$		25, 5	26, 5	
$\{s_0, s_2, s_4\}$	$+\infty$		21, 7		28
$\{s_0, s_3, s_4\}$	$+\infty$			17, 0	22, 4
$\{s_0, s_1, s_2, s_3\}$	$+\infty$	34, 8	37, 4	43, 0	
$\{s_0, s_1, s_2, s_4\}$	$+\infty$				
$\{s_0, s_1, s_3, s_4\}$	$+\infty$				
$\{s_0, s_2, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$				
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$					



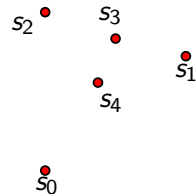
# Exemple

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				11, 7
$\{s_0, s_1, s_2\}$	$+\infty$	34, 7	37, 4		
$\{s_0, s_1, s_3\}$	$+\infty$	25, 2		28, 9	
$\{s_0, s_1, s_4\}$	$+\infty$	22, 1			31, 1
$\{s_0, s_2, s_3\}$	$+\infty$		25, 5	26, 5	
$\{s_0, s_2, s_4\}$	$+\infty$		21, 7		28
$\{s_0, s_3, s_4\}$	$+\infty$			17, 0	22, 4
$\{s_0, s_1, s_2, s_3\}$	$+\infty$	34, 8	37, 4	43, 0	
$\{s_0, s_1, s_2, s_4\}$	$+\infty$	38, 4	38, 9		45, 2
$\{s_0, s_1, s_3, s_4\}$	$+\infty$				
$\{s_0, s_2, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$				
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$					



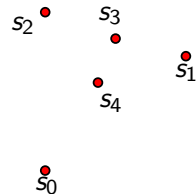
# Exemple

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				11, 7
$\{s_0, s_1, s_2\}$	$+\infty$	34, 7	37, 4		
$\{s_0, s_1, s_3\}$	$+\infty$	25, 2		28, 9	
$\{s_0, s_1, s_4\}$	$+\infty$	22, 1			31, 1
$\{s_0, s_2, s_3\}$	$+\infty$		25, 5	26, 5	
$\{s_0, s_2, s_4\}$	$+\infty$		21, 7		28
$\{s_0, s_3, s_4\}$	$+\infty$			17, 0	22, 4
$\{s_0, s_1, s_2, s_3\}$	$+\infty$	34, 8	37, 4	43, 0	
$\{s_0, s_1, s_2, s_4\}$	$+\infty$	38, 4	38, 9		45, 2
$\{s_0, s_1, s_3, s_4\}$	$+\infty$	25, 3		30, 3	34, 2
$\{s_0, s_2, s_3, s_4\}$	$+\infty$				
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$				
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$					



# Exemple

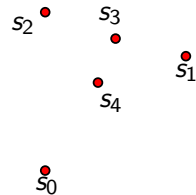
	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				11, 7
$\{s_0, s_1, s_2\}$	$+\infty$	34, 7	37, 4		
$\{s_0, s_1, s_3\}$	$+\infty$	25, 2		28, 9	
$\{s_0, s_1, s_4\}$	$+\infty$	22, 1			31, 1
$\{s_0, s_2, s_3\}$	$+\infty$		25, 5	26, 5	
$\{s_0, s_2, s_4\}$	$+\infty$		21, 7		28
$\{s_0, s_3, s_4\}$	$+\infty$			17, 0	22, 4
$\{s_0, s_1, s_2, s_3\}$	$+\infty$	34, 8	37, 4	43, 0	
$\{s_0, s_1, s_2, s_4\}$	$+\infty$	38, 4	38, 9		45, 2
$\{s_0, s_1, s_3, s_4\}$	$+\infty$	25, 3		30, 3	34, 2
$\{s_0, s_2, s_3, s_4\}$	$+\infty$		25, 6	30, 2	31, 9
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$				
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$					





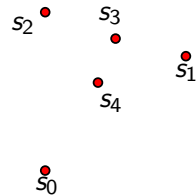
# Exemple

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				11, 7
$\{s_0, s_1, s_2\}$	$+\infty$	34, 7	37, 4		
$\{s_0, s_1, s_3\}$	$+\infty$	25, 2		28, 9	
$\{s_0, s_1, s_4\}$	$+\infty$	22, 1			31, 1
$\{s_0, s_2, s_3\}$	$+\infty$		25, 5	26, 5	
$\{s_0, s_2, s_4\}$	$+\infty$		21, 7		28
$\{s_0, s_3, s_4\}$	$+\infty$			17, 0	22, 4
$\{s_0, s_1, s_2, s_3\}$	$+\infty$	34, 8	37, 4	43, 0	
$\{s_0, s_1, s_2, s_4\}$	$+\infty$	38, 4	38, 9		45, 2
$\{s_0, s_1, s_3, s_4\}$	$+\infty$	25, 3		30, 3	34, 2
$\{s_0, s_2, s_3, s_4\}$	$+\infty$		25, 6	30, 2	31, 9
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$	38, 5	38, 9	46, 7	45, 2
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$					



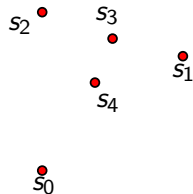
# Exemple

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				11, 7
$\{s_0, s_1, s_2\}$	$+\infty$	34, 7	37, 4		
$\{s_0, s_1, s_3\}$	$+\infty$	25, 2		28, 9	
$\{s_0, s_1, s_4\}$	$+\infty$	22, 1			31, 1
$\{s_0, s_2, s_3\}$	$+\infty$		25, 5	26, 5	
$\{s_0, s_2, s_4\}$	$+\infty$		21, 7		28
$\{s_0, s_3, s_4\}$	$+\infty$			17, 0	22, 4
$\{s_0, s_1, s_2, s_3\}$	$+\infty$	34, 8	37, 4	43, 0	
$\{s_0, s_1, s_2, s_4\}$	$+\infty$	38, 4	38, 9		45, 2
$\{s_0, s_1, s_3, s_4\}$	$+\infty$	25, 3		30, 3	34, 2
$\{s_0, s_2, s_3, s_4\}$	$+\infty$		25, 6	30, 2	31, 9
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$	38, 5	38, 9	46, 7	45, 2
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$		59, 1	56, 9	63, 7	56, 9



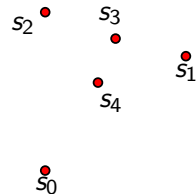
# Exemple

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				11, 7
$\{s_0, s_1, s_2\}$	$+\infty$	34, 7	37, 4		
$\{s_0, s_1, s_3\}$	$+\infty$	25, 2		28, 9	
$\{s_0, s_1, s_4\}$	$+\infty$	22, 1			31, 1
$\{s_0, s_2, s_3\}$	$+\infty$		25, 5	26, 5	
$\{s_0, s_2, s_4\}$	$+\infty$		21, 7		28
$\{s_0, s_3, s_4\}$	$+\infty$			17, 0	22, 4
$\{s_0, s_1, s_2, s_3\}$	$+\infty$	34, 8	37, 4	43, 0	
$\{s_0, s_1, s_2, s_4\}$	$+\infty$	38, 4	38, 9		45, 2
$\{s_0, s_1, s_3, s_4\}$	$+\infty$	25, 3		30, 3	34, 2
$\{s_0, s_2, s_3, s_4\}$	$+\infty$		25, 6	30, 2	31, 9
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$	38, 5	38, 9	46, 7	45, 2
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$		59, 1	56, 9	63, 7	56, 9



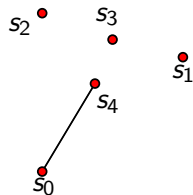
## Exemple : reconstruction

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				11, 7
$\{s_0, s_1, s_2\}$	$+\infty$	34, 7	37, 4		
$\{s_0, s_1, s_3\}$	$+\infty$	25, 2		28, 9	
$\{s_0, s_1, s_4\}$	$+\infty$	22, 1			31, 1
$\{s_0, s_2, s_3\}$	$+\infty$		25, 5	26, 5	
$\{s_0, s_2, s_4\}$	$+\infty$		21, 7		28
$\{s_0, s_3, s_4\}$	$+\infty$			17, 0	22, 4
$\{s_0, s_1, s_2, s_3\}$	$+\infty$	34, 8	37, 4	43, 0	
$\{s_0, s_1, s_2, s_4\}$	$+\infty$	38, 4	38, 9		45, 2
$\{s_0, s_1, s_3, s_4\}$	$+\infty$	25, 3		30, 3	34, 2
$\{s_0, s_2, s_3, s_4\}$	$+\infty$		25, 6	30, 2	31, 9
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$	38, 5	38, 9	46, 7	45, 2
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$		59, 1	56, 9	63, 7	56, 9



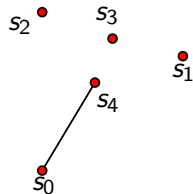
## Exemple : reconstruction

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				11, 7
$\{s_0, s_1, s_2\}$	$+\infty$	34, 7	37, 4		
$\{s_0, s_1, s_3\}$	$+\infty$	25, 2		28, 9	
$\{s_0, s_1, s_4\}$	$+\infty$	22, 1			31, 1
$\{s_0, s_2, s_3\}$	$+\infty$		25, 5	26, 5	
$\{s_0, s_2, s_4\}$	$+\infty$		21, 7		28
$\{s_0, s_3, s_4\}$	$+\infty$			17, 0	22, 4
$\{s_0, s_1, s_2, s_3\}$	$+\infty$	34, 8	37, 4	43, 0	
$\{s_0, s_1, s_2, s_4\}$	$+\infty$	38, 4	38, 9		45, 2
$\{s_0, s_1, s_3, s_4\}$	$+\infty$	25, 3		30, 3	34, 2
$\{s_0, s_2, s_3, s_4\}$	$+\infty$		25, 6	30, 2	31, 9
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$	38, 5	38, 9	46, 7	45, 2
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$		59, 1	56, 9	63, 7	56, 9



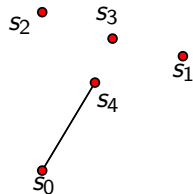
## Exemple : reconstruction

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				11, 7
$\{s_0, s_1, s_2\}$	$+\infty$	34, 7	37, 4		
$\{s_0, s_1, s_3\}$	$+\infty$	25, 2		28, 9	
$\{s_0, s_1, s_4\}$	$+\infty$	22, 1			31, 1
$\{s_0, s_2, s_3\}$	$+\infty$		25, 5	26, 5	
$\{s_0, s_2, s_4\}$	$+\infty$		21, 7		28
$\{s_0, s_3, s_4\}$	$+\infty$			17, 0	22, 4
$\{s_0, s_1, s_2, s_3\}$	$+\infty$	34, 8	37, 4	43, 0	
$\{s_0, s_1, s_2, s_4\}$	$+\infty$	38, 4	38, 9		45, 2
$\{s_0, s_1, s_3, s_4\}$	$+\infty$	25, 3		30, 3	34, 2
$\{s_0, s_2, s_3, s_4\}$	$+\infty$		25, 6	30, 2	31, 9
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$	38, 5	38, 9	46, 7	45, 2
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$		59, 1	56, 9	63, 7	56, 9



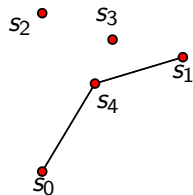
## Exemple : reconstruction

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				11, 7
$\{s_0, s_1, s_2\}$	$+\infty$	34, 7	37, 4		
$\{s_0, s_1, s_3\}$	$+\infty$	25, 2		28, 9	
$\{s_0, s_1, s_4\}$	$+\infty$	22, 1			31, 1
$\{s_0, s_2, s_3\}$	$+\infty$		25, 5	26, 5	
$\{s_0, s_2, s_4\}$	$+\infty$		21, 7		28
$\{s_0, s_3, s_4\}$	$+\infty$			17, 0	22, 4
$\{s_0, s_1, s_2, s_3\}$	$+\infty$	34, 8	37, 4	43, 0	
$\{s_0, s_1, s_2, s_4\}$	$+\infty$	38, 4	38, 9		45, 2
$\{s_0, s_1, s_3, s_4\}$	$+\infty$	25, 3		30, 3	34, 2
$\{s_0, s_2, s_3, s_4\}$	$+\infty$		25, 6	30, 2	31, 9
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$	38, 5	38, 9	46, 7	45, 2
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$		59, 1	56, 9	63, 7	56, 9



## Exemple : reconstruction

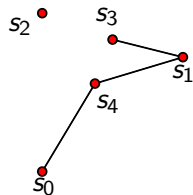
	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				11, 7
$\{s_0, s_1, s_2\}$	$+\infty$	34, 7	37, 4		
$\{s_0, s_1, s_3\}$	$+\infty$	25, 2		28, 9	
$\{s_0, s_1, s_4\}$	$+\infty$	22, 1			31, 1
$\{s_0, s_2, s_3\}$	$+\infty$		25, 5	26, 5	
$\{s_0, s_2, s_4\}$	$+\infty$		21, 7		28
$\{s_0, s_3, s_4\}$	$+\infty$			17, 0	22, 4
$\{s_0, s_1, s_2, s_3\}$	$+\infty$	34, 8	37, 4	43, 0	
$\{s_0, s_1, s_2, s_4\}$	$+\infty$	38, 4	38, 9		45, 2
$\{s_0, s_1, s_3, s_4\}$	$+\infty$	25, 3		30, 3	34, 2
$\{s_0, s_2, s_3, s_4\}$	$+\infty$		25, 6	30, 2	31, 9
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$	38, 5	38, 9	46, 7	45, 2
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$		59, 1	56, 9	63, 7	56, 9





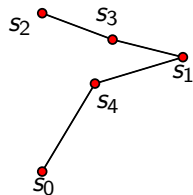
## Exemple : reconstruction

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				11, 7
$\{s_0, s_1, s_2\}$	$+\infty$	34, 7	37, 4		
$\{s_0, s_1, s_3\}$	$+\infty$	25, 2		28, 9	
$\{s_0, s_1, s_4\}$	$+\infty$	22, 1			31, 1
$\{s_0, s_2, s_3\}$	$+\infty$		25, 5	26, 5	
$\{s_0, s_2, s_4\}$	$+\infty$		21, 7		28
$\{s_0, s_3, s_4\}$	$+\infty$			17, 0	22, 4
$\{s_0, s_1, s_2, s_3\}$	$+\infty$	34, 8	37, 4	43, 0	
$\{s_0, s_1, s_2, s_4\}$	$+\infty$	38, 4	38, 9		45, 2
$\{s_0, s_1, s_3, s_4\}$	$+\infty$	25, 3		30, 3	34, 2
$\{s_0, s_2, s_3, s_4\}$	$+\infty$		25, 6	30, 2	31, 9
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$	38, 5	38, 9	46, 7	45, 2
$\min_j (\Delta_{[S, s_j]} + \delta_{j0})$		59, 1	56, 9	63, 7	56, 9



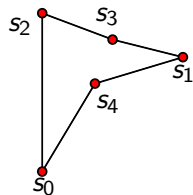
## Exemple : reconstruction

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				11, 7
$\{s_0, s_1, s_2\}$	$+\infty$	34, 7	37, 4		
$\{s_0, s_1, s_3\}$	$+\infty$	25, 2		28, 9	
$\{s_0, s_1, s_4\}$	$+\infty$	22, 1			31, 1
$\{s_0, s_2, s_3\}$	$+\infty$		25, 5	26, 5	
$\{s_0, s_2, s_4\}$	$+\infty$		21, 7		28
$\{s_0, s_3, s_4\}$	$+\infty$			17, 0	22, 4
$\{s_0, s_1, s_2, s_3\}$	$+\infty$	34, 8	37, 4	43, 0	
$\{s_0, s_1, s_2, s_4\}$	$+\infty$	38, 4	38, 9		45, 2
$\{s_0, s_1, s_3, s_4\}$	$+\infty$	25, 3		30, 3	34, 2
$\{s_0, s_2, s_3, s_4\}$	$+\infty$		25, 6	30, 2	31, 9
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$	38, 5	38, 9	46, 7	45, 2
$\min_j(\Delta_{[S, s_j]} + \delta_{j0})$		59, 1	56, 9	63, 7	56, 9



## Exemple : reconstruction

	$s_0$	$s_1$	$s_2$	$s_3$	$s_4$
$\{s_0\}$	0				
$\{s_0, s_1\}$	$+\infty$	20, 6			
$\{s_0, s_2\}$	$+\infty$		18		
$\{s_0, s_3\}$	$+\infty$			17	
$\{s_0, s_4\}$	$+\infty$				11, 7
$\{s_0, s_1, s_2\}$	$+\infty$	34, 7	37, 4		
$\{s_0, s_1, s_3\}$	$+\infty$	25, 2		28, 9	
$\{s_0, s_1, s_4\}$	$+\infty$	22, 1			31, 1
$\{s_0, s_2, s_3\}$	$+\infty$		25, 5	26, 5	
$\{s_0, s_2, s_4\}$	$+\infty$		21, 7		28
$\{s_0, s_3, s_4\}$	$+\infty$			17, 0	22, 4
$\{s_0, s_1, s_2, s_3\}$	$+\infty$	34, 8	37, 4	43, 0	
$\{s_0, s_1, s_2, s_4\}$	$+\infty$	38, 4	38, 9		45, 2
$\{s_0, s_1, s_3, s_4\}$	$+\infty$	25, 3		30, 3	34, 2
$\{s_0, s_2, s_3, s_4\}$	$+\infty$		25, 6	30, 2	31, 9
$\{s_0, s_1, s_2, s_3, s_4\}$	$+\infty$	38, 5	38, 9	46, 7	45, 2
$\min_j(\Delta_{[S, s_j]} + \delta_{j0})$		59, 1	56, 9	63, 7	56, 9



# Algorithme de reconstruction

**Algorithme :**  $\text{TSP}(S)$

$\Delta \leftarrow$  tableau à deux dimensions, indexé par les sous-ensembles de  $S$  contenant  $\{s_0\}$ , et par les sommets  $s_0$  à  $s_{n-1}$

$\text{Prec} \leftarrow$  tableau de mêmes dimensions

$\Delta[\{s_0\}, s_0] = 0$

**pour**  $t = 2$  à  $n$  :

**pour** tous les  $U \subset S$  de taille  $t$  tels que  $s_0 \in U$  :

$\Delta[U, s_0] = +\infty$

**pour** tout  $s_j \in U, j \neq 0$  :

$\Delta[U, s_j] = \min\{\Delta[U \setminus \{s_j\}, s_i] + \delta_{ij} : s_i \in U, i \neq j\}$

$\text{Prec}[U, s_j] \leftarrow$  indice du minimum

**renvoyer**  $\min_j (\Delta[\{s_0, \dots, s_n\}, s_j] + \delta_{j0}),$  indice du min et  $\text{Prec}$

# Algorithme de reconstruction

## Algorithme : $\text{TSP}(S)$

$\Delta \leftarrow$  tableau à deux dimensions, indexé par les sous-ensembles de  $S$  contenant  $\{s_0\}$ , et par les sommets  $s_0$  à  $s_{n-1}$

$\text{Prec} \leftarrow$  tableau de mêmes dimensions

$\Delta[\{s_0\}, s_0] = 0$

**pour**  $t = 2$  à  $n$  :

**pour** tous les  $U \subset S$  de taille  $t$  tels que  $s_0 \in U$  :

$\Delta[U, s_0] = +\infty$

**pour** tout  $s_j \in U, j \neq 0$  :

$\Delta[U, s_j] = \min\{\Delta[U \setminus \{s_j\}, s_i] + \delta_{ij} : s_i \in U, i \neq j\}$

$\text{Prec}[U, s_j] \leftarrow$  indice du minimum

**renvoyer**  $\min_j (\Delta[\{s_0, \dots, s_n\}, s_j] + \delta_{j0}),$  indice du min et  $\text{Prec}$

## Algorithme : $\text{TSP-REC}(S, \Delta, \text{Prec}, j)$

$i_0 \leftarrow 0$

$i_1 \leftarrow j$

$U \leftarrow S$

**pour**  $k = 2$  à  $n - 1$  :

$i_k \leftarrow \text{Prec}[U, i_{k-1}]$

$U \leftarrow U \setminus \{s_{i_{k-1}}\}$

**renvoyer**  $(i_0, i_1, \dots, i_{n-1}, i_0)$

# Algorithme de reconstruction

## Algorithme : TSP( $S$ )

$\Delta \leftarrow$  tableau à deux dimensions, indexé par les sous-ensembles de  $S$  contenant  $\{s_0\}$ , et par les sommets  $s_0$  à  $s_{n-1}$

$\text{Prec} \leftarrow$  tableau de mêmes dimensions

$\Delta[\{s_0\}, s_0] = 0$

**pour**  $t = 2$  à  $n$  :

**pour** tous les  $U \subset S$  de taille  $t$  tels que  $s_0 \in U$  :

$\Delta[U, s_0] = +\infty$

**pour** tout  $s_j \in U, j \neq 0$  :

$\Delta[U, s_j] = \min\{\Delta[U \setminus \{s_j\}, s_i] + \delta_{ij} : s_i \in U, i \neq j\}$

$\text{Prec}[U, s_j] \leftarrow$  indice du minimum

**renvoyer**  $\min_j (\Delta[\{s_0, \dots, s_n\}, s_j] + \delta_{j0}),$  indice du min et  $\text{Prec}$

## Algorithme : TSP-REC( $S, \Delta, \text{Prec}, j$ )

$i_0 \leftarrow 0$

$i_1 \leftarrow j$

$U \leftarrow S$

**pour**  $k = 2$  à  $n - 1$  :

$i_k \leftarrow \text{Prec}[U, i_{k-1}]$

$U \leftarrow U \setminus \{s_{i_{k-1}}\}$

**renvoyer**  $(i_0, i_1, \dots, i_{n-1}, i_0)$

## Lemme

*L'algorithme TSP-REC construit un chemin de longueur minimale en temps  $O(n)$*

## Théorème

*La longueur minimale d'un chemin passant par  $n$  points peut être calculée en temps  $O(n^2 2^n)$ . Le chemin lui-même peut être calculé en temps  $O(n)$  supplémentaire.*

## Théorème

*La longueur minimale d'un chemin passant par  $n$  points peut être calculée en temps  $O(n^2 2^n)$ . Le chemin lui-même peut être calculé en temps  $O(n)$  supplémentaire.*

- ▶ Détails à régler : gestion des ensembles
  - ▶ théorie : pas de problème (bonne complexité)
  - ▶ pratique : pas si facile !



## Théorème

*La longueur minimale d'un chemin passant par  $n$  points peut être calculée en temps  $O(n^2 2^n)$ . Le chemin lui-même peut être calculé en temps  $O(n)$  supplémentaire.*

- ▶ Détails à régler : gestion des ensembles
  - ▶ théorie : pas de problème (bonne complexité)
  - ▶ pratique : pas si facile !
- ▶ Fonctionne aussi hors d'un plan euclidien (carte, plans, graphes, ...)

## Théorème

*La longueur minimale d'un chemin passant par  $n$  points peut être calculée en temps  $O(n^2 2^n)$ . Le chemin lui-même peut être calculé en temps  $O(n)$  supplémentaire.*

- ▶ Détails à régler : gestion des ensembles
  - ▶ théorie : pas de problème (bonne complexité)
  - ▶ pratique : pas si facile !
- ▶ Fonctionne aussi hors d'un plan euclidien (carte, plans, graphes, ...)
- ▶ Très utile en pratique comme en théorie !

## Théorème

*La longueur minimale d'un chemin passant par  $n$  points peut être calculée en temps  $O(n^2 2^n)$ . Le chemin lui-même peut être calculé en temps  $O(n)$  supplémentaire.*

- ▶ Détails à régler : gestion des ensembles
  - ▶ théorie : pas de problème (bonne complexité)
  - ▶ pratique : pas si facile !
- ▶ Fonctionne aussi hors d'un plan euclidien (carte, plans, graphes, ...)
- ▶ Très utile en pratique comme en théorie !
- ▶ Un exemple : <http://map.vroom-project.org/>

# Conclusion sur la programmation dynamique

- ▶ Méthode assez systématique pour de nombreux problèmes
- ▶ Souvent :
  - ▶ version simple en *bonne* complexité
  - ▶ améliorations possibles
- ▶ Parfois (trop) gourmande en mémoire

# Conclusion sur la programmation dynamique

- ▶ Méthode assez systématique pour de nombreux problèmes
- ▶ Souvent :
  - ▶ version simple en *bonne* complexité
  - ▶ améliorations possibles
- ▶ Parfois (trop) gourmande en mémoire

## Également aperçu dans ce cours

- ▶ Algorithme exact exponentiel (TSP)
- ▶ Plus courts chemins dans des graphes (DIJKSTRA)