Programmation dynamique

Yann Vaxès et Benjamin Monmege

Rappel : Définition de suites récurrentes

Exemple de la factorielle:

$$u_0 = 1 \qquad \text{et} \qquad \forall n \geq 1 \quad u_n = n \times u_{n-1}$$

```
def factorielle(n: int) -> int:
    assert n >= 0
    if n == 0:
        return 1
    return n * factorielle(n - 1)
```

Rappel : Définition de suites récurrentes

Exemple de la factorielle:

$$u_0 = 1 \qquad \text{et} \qquad \forall n \geq 1 \quad u_n = n \times u_{n-1}$$

```
def factorielle(n: int) -> int:
    assert n >= 0
    if n == 0:
        return 1
    return n * factorielle(n - 1)
```

Correction et complexité s'étudient par des preuves par récurrence

($a, b \in \mathbf{N}$)

$$C_0 = a \qquad \text{et} \qquad \forall n \geq 1 \quad C_n = b + C_{n-1}$$

Rappel : Définition de suites récurrentes

Exemple de la factorielle:

$$u_0 = 1 \qquad \text{et} \qquad \forall n \geq 1 \quad u_n = n \times u_{n-1}$$

```
def factorielle(n: int) -> int:
    assert n >= 0
    if n == 0:
        return 1
    return n * factorielle(n - 1)
```

Correction et complexité s'étudient par des preuves par récurrence

($a, b \in \mathbf{N}$)

$$C_0 = a \qquad \text{et} \qquad \forall n \geq 1 \quad C_n = b + C_{n-1}$$

$$C_n = O(n)$$

Fonctions récursives : attention aux écueils

Suite de Fibonacci:

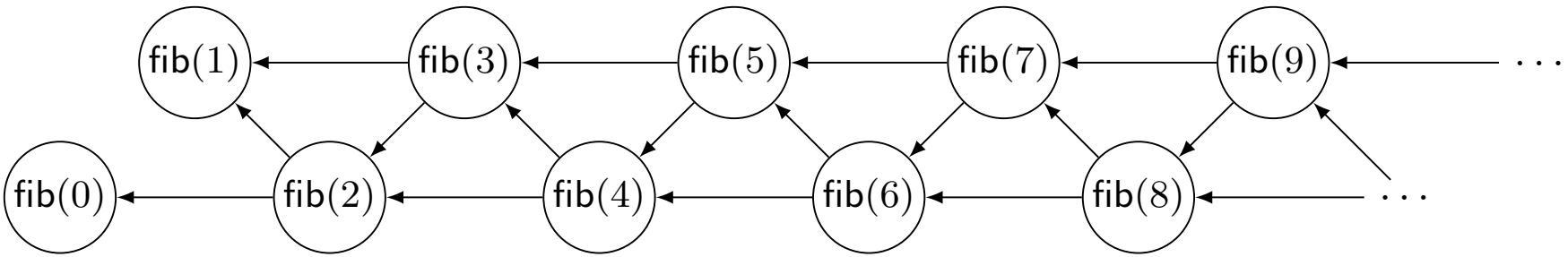
$$F_0 = F_1 = 1 \qquad \text{et} \qquad \forall n \geq 2 \quad F_n = F_{n-1} + F_{n-2}$$

```
def fibonacci(n):
    if n <= 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```

Quelle complexité ? À vous de jouer !

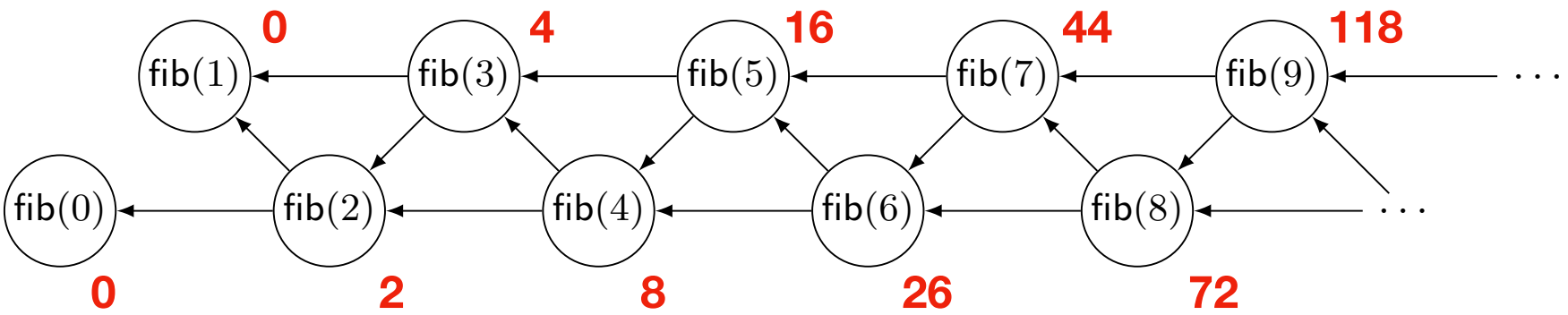
Graphe d'appels

```
def fibonacci(n):
    if n <= 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```

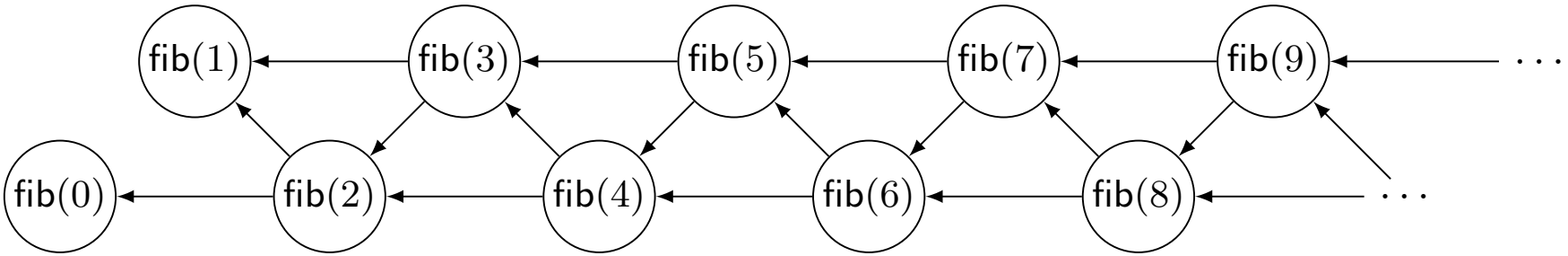


Graphe d'appels et nombre total d'appels nécessaires

```
def fibonacci(n):
    if n <= 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```



Pour faire mieux : calcul ascendant



- Calculer l'ensemble S des sommets accessibles depuis le sommet v dont on cherche à calculer la valeur
- Ordonner S selon un tri topologique $<$ (c'est-à-dire tel que si (u, v) est un arc du graphe d'appels alors $v < u$) : cf séance sur les parcours de graphes
- Par ordre croissant, calculer et stocker l'image de chaque sommet de S

Calcul ascendant pour Fibonacci

- Si on veut calculer F_n, l'ordre topologique nous dit de calculer tous les F_i pour i de 0 à n...
- On les stocke dans une liste !

```
def fibonacci(n):
    resultats = [1, 1] # on connait les valeurs de F0 et F1
    for i in range(2, n + 1):
        resultats.append(resultats[-1] + resultats[-2])
    return resultats[n]
```

Calcul ascendant pour Fibonacci

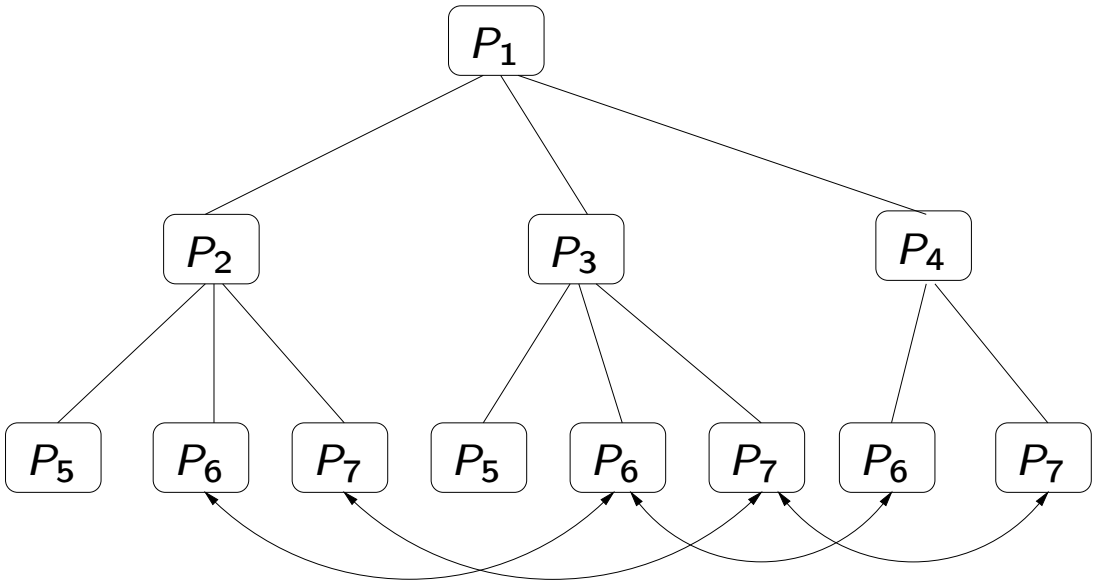
- Si on veut calculer F_n, l'ordre topologique nous dit de calculer tous les F_i pour i de 0 à n...
- On les stocke dans une liste !

```
def fibonacci(n):
    resultats = [1, 1] # on connait les valeurs de F0 et F1
    for i in range(2, n + 1):
        resultats.append(resultats[-1] + resultats[-2])
    return resultats[n]
```

Quelle complexité en temps ? Quelle complexité en espace ?
Peut-on faire mieux ?

Programmation dynamique

Une méthodologie pour aborder certains problèmes. . .



Lorsque les sous-problèmes ont des sous-sous-problèmes en commun, il faut les résoudre une seule fois et mettre le résultat dans une table !

Méthodologie

Problèmes d'optimisation

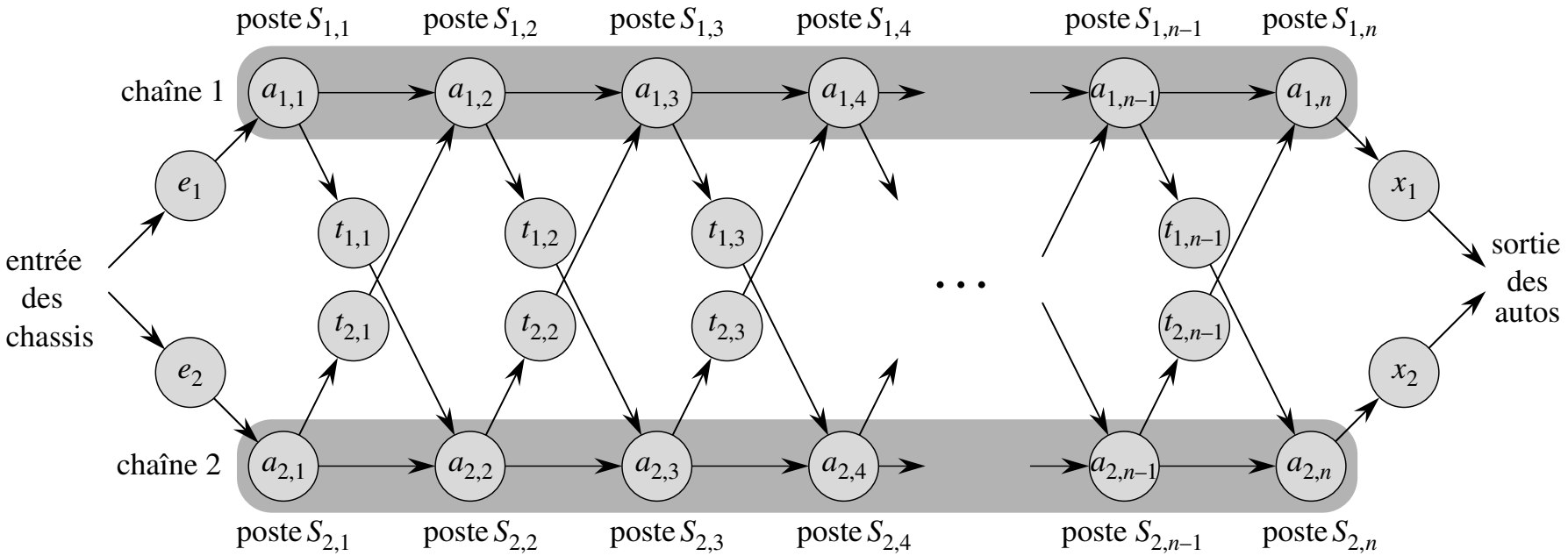
- des problèmes qui admettent un ensemble de solutions $\mathcal{S}$
- $c : \mathcal{S} \to \mathbf{R}$ fonction objectif
- on cherche une solution s^* telle que $c(s^*) = \min\{c(s) \mid s \in \mathcal{S}\}$

Conception d'un algorithme de programmation dynamique

1. Caractérisation de la structure des solutions optimales
2. Définition récursive de la valeur optimale
3. Calcul ascendant de la valeur optimale
4. Construction d'une solution optimale à partir des informations obtenues en 3 (facultatif)

Exemple : ordonnancement de chaînes de montage

- Un constructeur automobile possède un atelier avec deux chaînes de montage (1 et 2) comportant chacune n postes.
- Chaque véhicule doit passer par les n postes dans l'ordre.



Exemple : ordonnancement de chaînes de montage

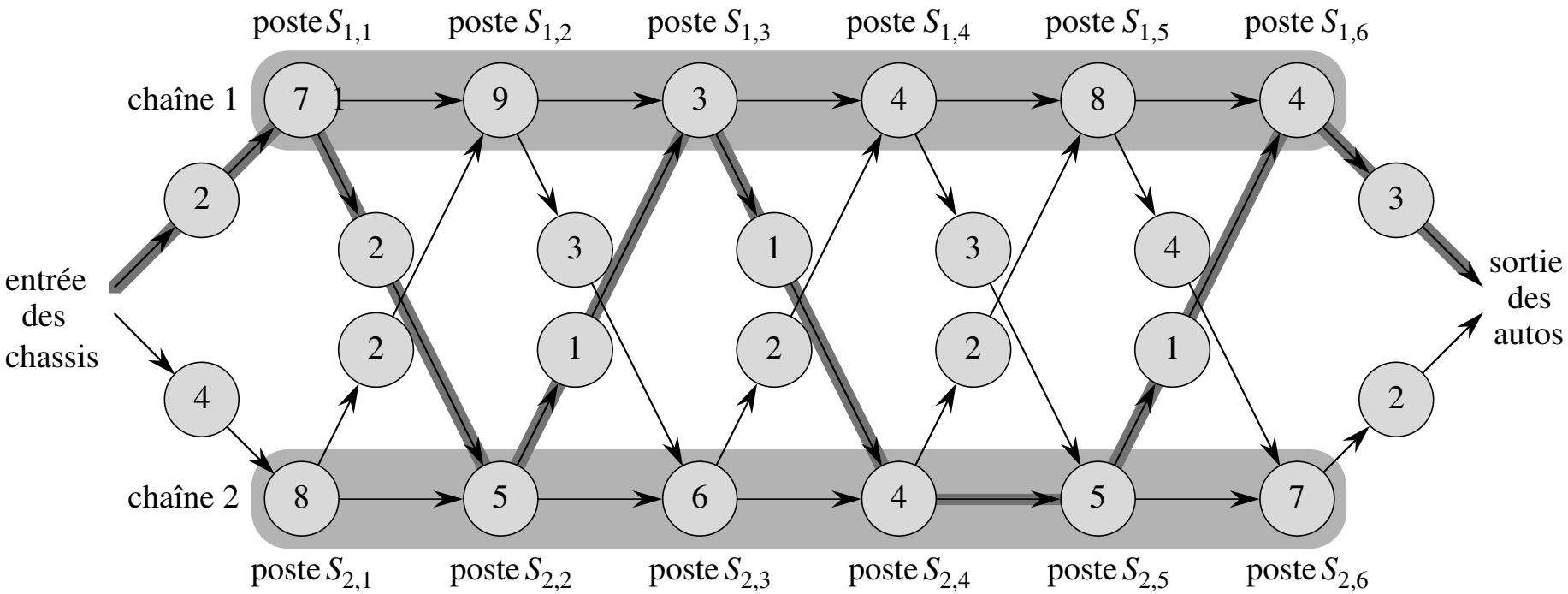
Données du problème

- $S_{i,j}$ le j-ième poste de la chaîne i
- e_i le *temps d'entrée* d'un véhicule sur la chaîne i
- $a_{i,j}$ le *temps de montage* pour le poste j sur la chaîne i (on peut avoir $a_{1,j} \neq a_{2,j}$)
- $t_{i,j}$ le *temps de transfert* d'un véhicule de la chaîne i vers l'autre chaîne après le poste $S_{i,j}$
- x_i le *temps de sortie* d'un véhicule de la chaîne i

Problème

Déterminer quels sont les postes à sélectionner sur la chaîne 1 et sur la chaîne 2 pour minimiser le délai de transit d'une auto à travers l'atelier

Exemple


poste $S_{1,1}$
poste $S_{1,2}$
poste $S_{1,3}$
poste $S_{1,4}$
poste $S_{1,5}$
poste $S_{1,6}$
chaîne 1
chaîne 2
entrée des chassis
sortie des autos
7 1 9 3 4 8 4
2 2 3 1 3 4 3
2 1 2 2 1 2
4
8 5 6 4 5 7
poste $S_{2,1}$
poste $S_{2,2}$
poste $S_{2,3}$
poste $S_{2,4}$
poste $S_{2,5}$
poste $S_{2,6}$

Espace des solutions

- Chaque solution est définie par le sous-ensemble de postes de la chaîne 1 utilisés (les postes restants sont choisis dans la chaîne 2)

Combien de solutions possibles ?

Espace des solutions

- Chaque solution est définie par le sous-ensemble de postes de la chaîne 1 utilisés (les postes restants sont choisis dans la chaîne 2)

Combien de solutions possibles ?

- 2^n solutions possibles : l'approche naïve consistant à parcourir tous les chemins possibles est inefficace

Étape 1 : sous-structures optimales

Les sous-problèmes à considérer consistent à calculer un itinéraire optimal jusqu'au poste $S_{i,j}$ pour $i = 1, 2$ et $j = 1, \ldots, n$

Par exemple, considérons un itinéraire optimal jusqu'au poste $S_{1,j}$:

- Si $j = 1$, il n'y a qu'un seul chemin possible
- Pour $j = 2, \ldots, n$, il y a deux possibilités:
  - ou bien l'itinéraire optimal jusqu'à $S_{1,j-1}$ suivi du poste $S_{1,j}$
  - ou bien l'itinéraire optimal jusqu'à $S_{2,j-1}$ suivi d'un changement de chaîne et du poste $S_{1,j}$

Étape 2 : solution récursive

$f_i[j]$: délai optimal jusqu'à $S_{i,j}$

f^* : délai optimal pour traverser l'atelier

$$f^* = \min(f_1[n] + x_1, f_2[n] + x_2)$$

Étape 2 : solution récursive

$f_i[j]$: délai optimal jusqu'à $S_{i,j}$

f^* : délai optimal pour traverser l'atelier

$$f^* = \min(f_1[n] + x_1, f_2[n] + x_2)$$

Pour le poste 1 de chaque chaîne, pas de choix:

$$f_1[1] = e_1 + a_{1,1} \qquad\qquad f_2[1] = e_2 + a_{2,1}$$

Étape 2 : solution récursive

$f_i[j]$: délai optimal jusqu'à $S_{i,j}$

f^* : délai optimal pour traverser l'atelier

$$f^* = \min(f_1[n] + x_1, f_2[n] + x_2)$$

Pour le poste 1 de chaque chaîne, pas de choix:

$$f_1[1] = e_1 + a_{1,1} \qquad\qquad f_2[1] = e_2 + a_{2,1}$$

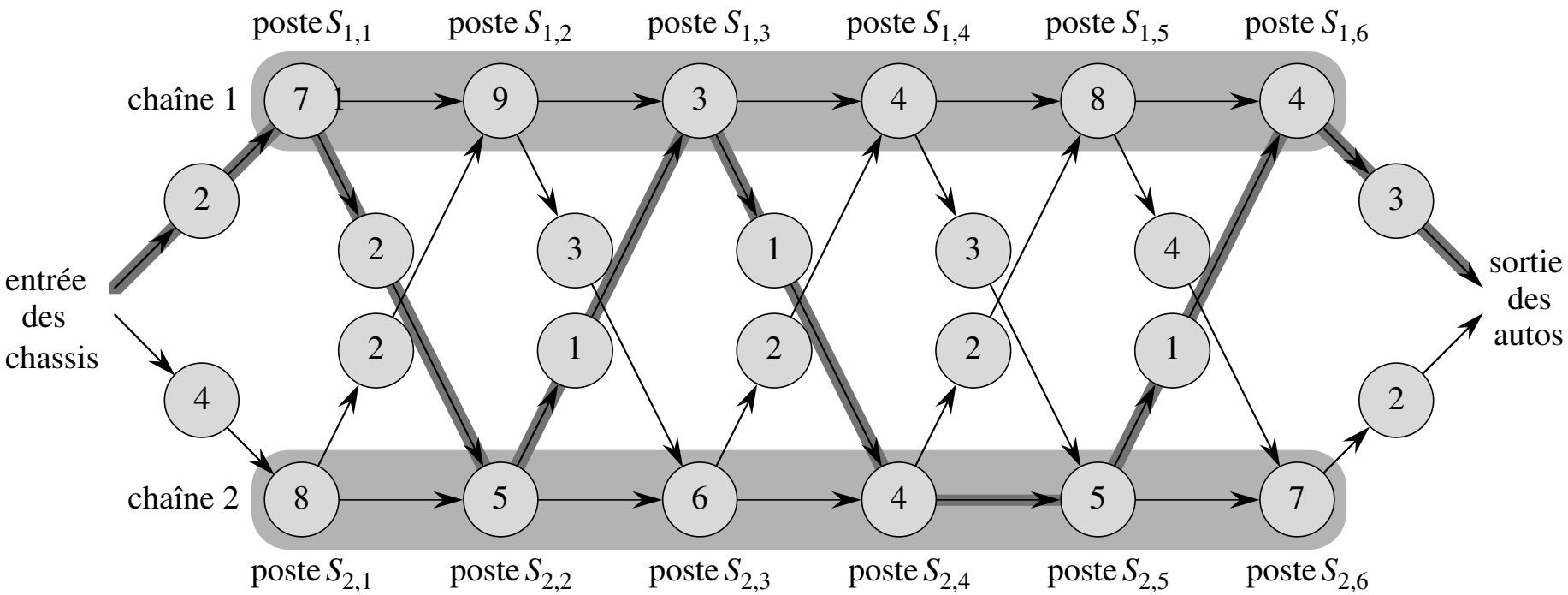
Pour le poste $j = 2, \ldots, n$ de la chaîne 1, deux possibilités:

$$f_1[j] = \min(f_1[j-1] + a_{1,j}, f_2[j-1] + t_{2,j-1} + a_{1,j})$$

et idem pour la chaîne 2:

$$f_2[j] = \min(f_2[j-1] + a_{2,j}, f_1[j-1] + t_{1,j-1} + a_{2,j})$$

Exemple



j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

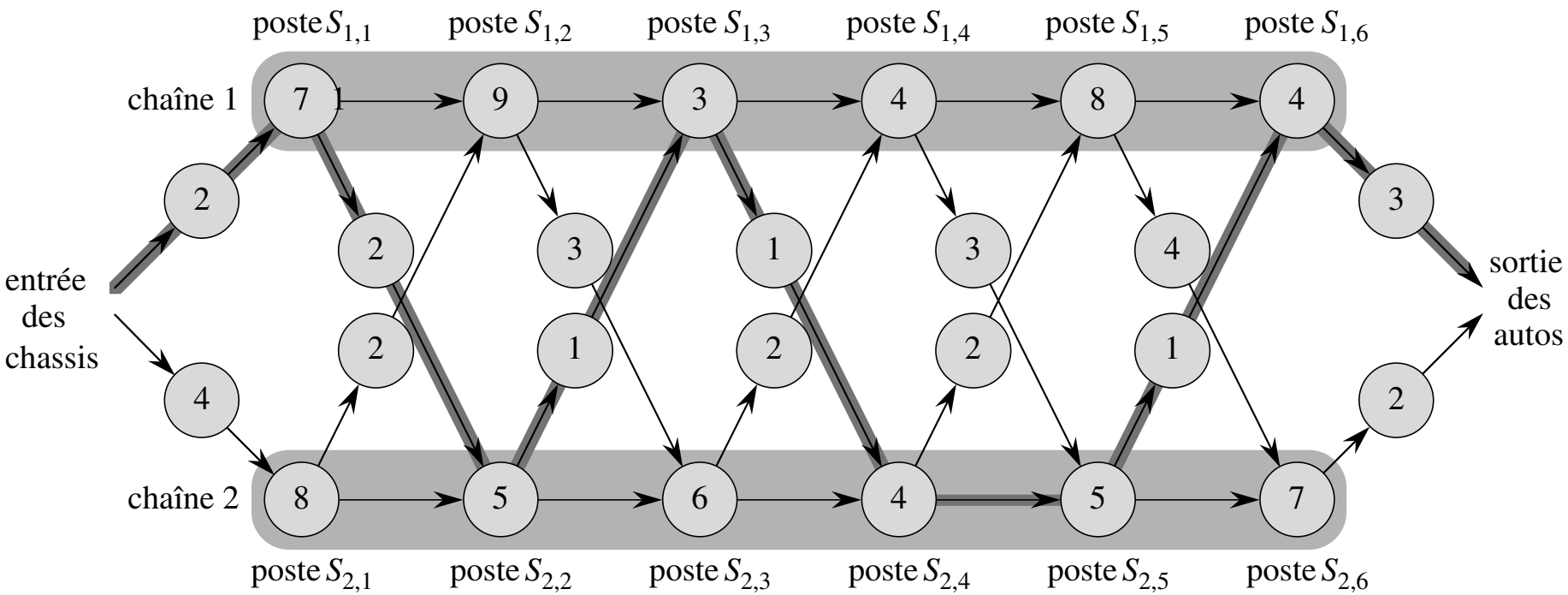
Information pour la construction de la solution

Les $f_i[j]$ sont les *valeurs* des solutions optimales des sous-problèmes

Pour pouvoir reconstruire les *solutions optimales* elles-mêmes, on définit

- $l_i[j]$ le numéro de la chaîne (1 ou 2) dont le poste $j-1$ est utilisé par un chemin optimal jusqu'au poste $S_{i,j}$
- $l_1[j]$ n'est pas défini car aucun poste ne vient avant le poste 1

Exemple



j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

j	2	3	4	5	6
$l_1[j]$	1	2	1	1	2
$l_2[j]$	1	2	1	2	2

$l^* = 1$

Étape 3 : calcul des temps optimaux de manière ascendante

Plus-Rapide-Chemin(a, t, e, x, n)

1. $f_1[1] \leftarrow e_1 + a_{1,1}$
2. $f_2[1] \leftarrow e_2 + a_{2,1}$
3. **pour** $j \leftarrow 2$ à n
4. **faire si** $f_1[j-1] + a_{1,j} \leq f_2[j-1] + t_{2,j-1} + a_{1,j}$
5. **alors** $f_1[j] \leftarrow f_1[j-1] + a_{1,j}$
6. $l_1[j] \leftarrow 1$
7. **sinon** $f_1[j] \leftarrow f_2[j-1] + t_{2,j-1} + a_{1,j}$
8. $l_1[j] \leftarrow 2$
9. **si** $f_2[j-1] + a_{2,j} \leq f_1[j-1] + t_{1,j-1} + a_{2,j}$
10. **alors** $f_2[j] \leftarrow f_2[j-1] + a_{2,j}$
11. $l_2[j] \leftarrow 2$
12. **sinon** $f_2[j] \leftarrow f_1[j-1] + t_{1,j-1} + a_{2,j}$
13. $l_2[j] \leftarrow 1$
14. **si** $f_1[n] + x_1 \leq f_2[n] + x_2$
15. **alors** $f^* = f_1[n] + x_1$
16. $l^* = 1$
17. **sinon** $f^* = f_2[n] + x_2$
18. $l^* = 2$

Étape 3 : calcul des temps optimaux de manière ascendante

Plus-Rapide-Chemin(a, t, e, x, n)

1. $f_1[1] \leftarrow e_1 + a_{1,1}$
2. $f_2[1] \leftarrow e_2 + a_{2,1}$
3. **pour** $j \leftarrow 2$ à n
4. **faire si** $f_1[j-1] + a_{1,j} \leq f_2[j-1] + t_{2,j-1} + a_{1,j}$
5. **alors** $f_1[j] \leftarrow f_1[j-1] + a_{1,j}$
6. $l_1[j] \leftarrow 1$
7. **sinon** $f_1[j] \leftarrow f_2[j-1] + t_{2,j-1} + a_{1,j}$
8. $l_1[j] \leftarrow 2$
9. **si** $f_2[j-1] + a_{2,j} \leq f_1[j-1] + t_{1,j-1} + a_{2,j}$
10. **alors** $f_2[j] \leftarrow f_2[j-1] + a_{2,j}$
11. $l_2[j] \leftarrow 2$
12. **sinon** $f_2[j] \leftarrow f_1[j-1] + t_{1,j-1} + a_{2,j}$
13. $l_2[j] \leftarrow 1$
14. **si** $f_1[n] + x_1 \leq f_2[n] + x_2$
15. **alors** $f^* = f_1[n] + x_1$
16. $l^* = 1$
17. **sinon** $f^* = f_2[n] + x_2$
18. $l^* = 2$

Complexité : $O(n)$

Étape 4 : construction du chemin optimal

Affichage par ordre décroissant des numéros de poste :

Afficher-Postes(l, n)

1. $i \leftarrow l^*$
2. afficher "chaîne" i, "poste" n
3. **pour** $j \leftarrow n$ **jusqu'à** 2
4. $\quad$ **faire** $i \leftarrow l_i[j]$
5. $\quad$ afficher "chaîne" i, "poste" $j - 1$