Chapitre 1 : Programmation dynamique

ENSIIE - Module de Recherche Opérationnelle

Dimitri Watel (dimitri.watel@ensiie.fr)

2022

Principe

Définition

La programmation dynamique est une approche de résolution ; une manière de concevoir un algorithme pour résoudre un problème. Ce n'est pas un algorithme.

Avertissement

La méthode est importante. **Les exemples ne le sont pas.** Vous verrez dans ce cours et en TD des exercices permettant d'appliquer la programmation dynamique sur des problèmes différents de ceux que vous aurez à résoudre en examen.
Dans ce cours, vous verrez 3 exemples :

- La suite de Fibonacci
- La somme de sous-ensembles
- Les plus courts-chemins dans un cas particulier

Trouver une formule de récurrence

Etape 1

La programmation dynamique est une manière de coder une formule de récurrence. Pour résoudre un problème avec la méthode de programmation dynamique, la première étape consiste à trouver la formule de récurrence.

Suite de Fibonacci

Définition du problème

Soit un entier n, trouver $f(n)$ défini par $f(n) = f(n-1) + f(n-2)$ si $n > 1$, et par $f(0) = 0$ et $f(1) = 1$.

La formule de récurrence est déjà décrite dans le problème.

Suite de Fibonacci : version naïve

Fonction $f(i)$
  Si $i \leq 1$ **Alors**
    Renvoyer i
  Renvoyer $f(i-1) + f(i-2)$

Problème de somme de sous-ensemble

$$P = \{1, 1, 1, 5, 7, 8, 8, 9, 17\}$$

Existe-t-il un sous-ensemble de P de somme 40 ?

$$\exists ? P' \subset P | \sum_{p \in P'} p = 40$$

Problème de somme de sous-ensemble

$$P = \{\mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{5}, \mathbf{7}, \mathbf{8}, 8, 9, \mathbf{17}\}$$

Existe-t-il un sous-ensemble de P de somme 40 ?

$$\exists ? P' \subset P | \sum_{p \in P'} p = 40$$

Formule de récurrence

$$P = \{1, 1, 1, 5, 7, 8, 8, 9, 17\}$$

Posons $P = \{p_1, p_2, \ldots, p_9\}$ et $B = 40$. Le problème devient :

Somme de sous-ensemble

$$\exists ? P' \subset \{p_1, p_2, \ldots, p_9\} | \sum_{p \in P'} p = B.$$

Version plus générale

Soit $i \leq 9$ et $b \leq B$, calculer
$f(i, b) = \exists ? P' \subset \{p_1, p_2, \ldots, p_i\} | \sum_{p \in P'} p = b.$

Formule de récurrence

Supposons que $\exists P' \subset \{p_1, p_2, \ldots, p_i\} | \sum_{p \in P'} p = b$. Dans ce cas :

- soit $p_i \in P'$ et donc $\exists P'' \subset \{p_1, p_2, \ldots, p_{i-1}\} | \sum_{p \in P''} p = b - p_i$
- soit $p_i \notin P'$ et donc $\exists P'' \subset \{p_1, p_2, \ldots, p_{i-1}\} | \sum_{p \in P''} p = b$

$$f(i, b) = f(i-1, b-p_i) \vee f(i-1, b)$$

Formule de récurrence

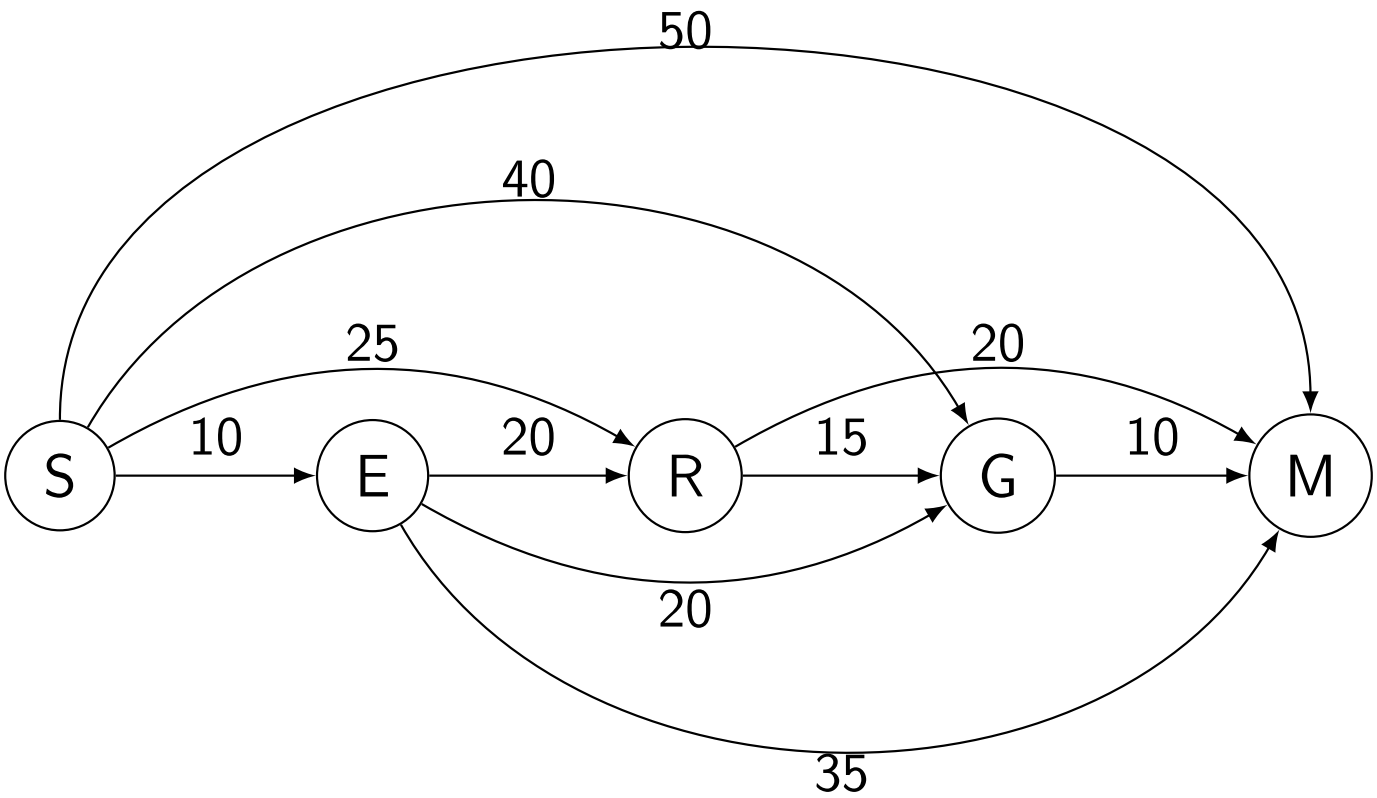
Plus exactement :

$$f(i, b \geq 0) = f(i-1, b-p_i) \vee f(i-1, b)$$
$$f(i, b < 0) = \bot$$
$$f(0, b \neq 0) = \bot$$
$$f(0, 0) = \top$$

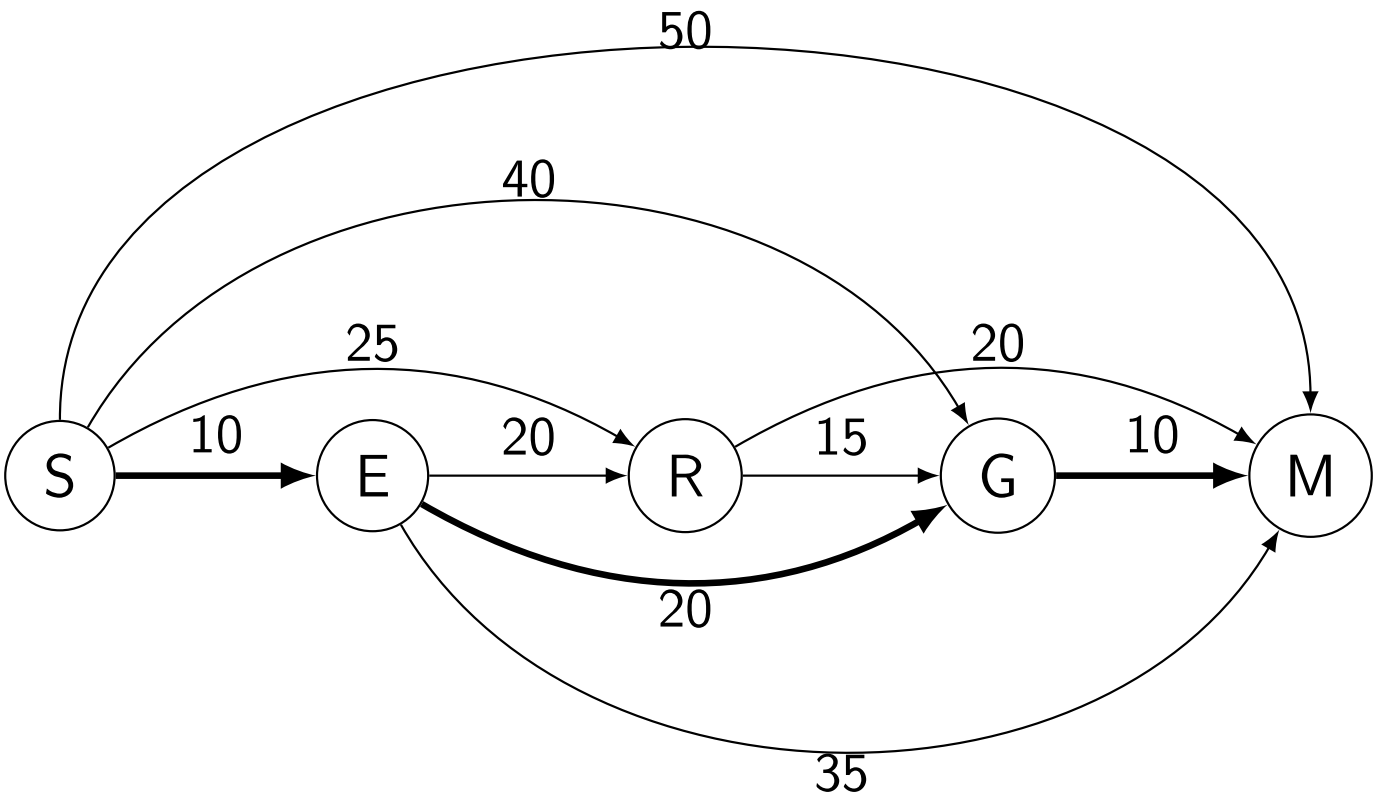
Somme de sous-ensemble : version naïve

Fonction $f(i, b)$
  Si $b < 0$ **Alors**
    Renvoyer $\bot$
  Si $i = 0$ **Alors**
    Renvoyer $(b = 0)$
  Renvoyer $f(i-1, b-p_i) \vee f(i-1, b)$

Quel est le plus court chemin de S à M ?



Quel est le plus court chemin de S à M ?



Formule de récurrence

On note $d(v)$ le poids d'un plus court chemin de v vers M. On recherche $d(S)$. On note aussi $\omega(u, v)$ le poids de l'arc (u, v)

Version générale

Soit v un nœud de G, calculer $d(v)$.

Relation de récurrence

Soit u^* le successeur de v dans un plus court chemin P de v vers M alors $d(v) = d(u^*) + \omega(v, u^*)$. Si u est un autre successeur de v, alors $d(u) + \omega(v, u) \geq d(u^*) + \omega(v, u^*)$.

$$d(v) = \min_{u \in \Gamma^+(v)} (d(u) + \omega(v, u))$$
$$d(M) = 0$$

Remarque : Ça fonctionne car le graphe n'a pas de circuit !

Plus court chemin : version naïve

Fonction $d(v)$
  Si $v = M$ **Alors**
    Renvoyer 0
  Renvoyer $\min_{u \in \Gamma^+(v)} (d(u) + \omega(v, u))$

Impraticabilité

Aucune de ces relations de récurrence ne doit être programmée naïvement : on répèterait les mêmes calculs.

Fibonacci

$f(n) = f(n-1) + \mathbf{f(n-2)} = \mathbf{f(n-2)} + \mathbf{f(n-3)} + \mathbf{f(n-3)} + f(n-4) = \dots$

Somme de sous-ensemble

Si $p_{i-1} = p_i = 1$, $f(i, B) = f(i-1, B-1) \vee f(i-1, B) =$
$f(i-2, B-2) \vee \mathbf{f(i-2, B-1)} \vee \mathbf{f(i-2, B-1)} \vee f(i-2, B)$

Plus court chemin

Si les arcs (u, v), (u, w) et (v, w) existent.
$d(u) = \min(d(v) + \omega(u, v), \mathbf{d(w)} + \omega(u, w), \dots) =$
$\min(\min(\mathbf{d(w)} + \omega(v, w), \dots) + \omega(u, v), \min(\dots) + \omega(u, w), \dots)$

Programmation dynamique : version Mémoïsation

Définition

Utiliser la programmation dynamique sous forme de mémoïsation pour coder une fonction récursive consiste à enregistrer au fur et à mesure les résultats des appels récursifs, de sorte que le calcul n'est pas refait une seconde fois si l'appel est répété.

Suite de fibonacci : Mémoïsation

Soit $T : \mathbb{N} \to \mathbb{N}$ un tableau où initialement toutes les cases sont vides.

Fonction $f(i)$
  Si $i \leq 1$ **Alors**
    Renvoyer i
  Renvoyer $f(i-1) + f(i-2)$

Fonction $f_{Memo}(i)$
  Si $T(i)$ est vide **Alors**
    Si $i \leq 1$ **Alors**
      Renvoyer i
    $T(i) \leftarrow f_{Memo}(i-1) + f_{Memo}(i-2)$
  Renvoyer $T(i)$

Somme de sous-ensemble

Soit $T : \mathbb{N}^2 \to \{\top, \bot\}$ un tableau où initialement toutes les cases sont vides.

Fonction $f(i, b)$
  Si $b < 0$ **Alors**
    Renvoyer $\bot$
  Si $i = 0$ **Alors**
    Renvoyer $(b = 0)$
  Renvoyer $f(i-1, b-p_i) \vee f(i-1, b)$

Fonction $f_{Memo}(i, b)$
  Si $T(i, b)$ est vide **Alors**
    Si $b < 0$ **Alors**
      Renvoyer $\bot$
    Si $i = 0$ **Alors**
      Renvoyer $(b = 0)$
    $T(i, b) \leftarrow f_{Memo}(i-1, b-p_i) \vee f_{Memo}(i-1, b)$
  Renvoyer $T(i, b)$

Plus court chemin

Soit $T : V \to \mathbb{N}$ un tableau où initialement toutes les cases sont vides.

Fonction $d(v)$
  Si $v = M$ **Alors**
    Renvoyer 0
  Renvoyer
$$\min_{u \in \Gamma^+(v)} (d(u) + \omega(v,u))$$

Fonction $d_{Memo}(v)$
  Si $T(v)$ est vide **Alors**
    Si $v = M$ **Alors**
      Renvoyer 0
    $T(v) \leftarrow$
$$\min_{u \in \Gamma^+(v)} (d_{Memo}(u) + \omega(v,u))$$
  Renvoyer $T(v)$

De manière générale

Programmation dynamique : version memoïsation

Si on souhaite calculer une fonction récursive $f : X \to Y$,

- créer un tableau $T : X \to Y$
- Avant de faire un calcul, vérifier si le résultat n'est pas dans T
- Enregistrer tout résultat dans T avant de le renvoyer.

Programmation dynamique : version itérative

Définition

Utiliser la programmation dynamique sous forme iterative pour coder une fonction récursive consiste à résoudre tous les sous-problèmes, en commençant par les cas terminaux et en remontant à l'envers les appels récursifs, jusqu'à ce que le problème initial soit résolu. Les résultats intermédiaires sont stockés dans un tableau.

Suite de fibonacci : Version iterative

- Liste de tous les sous-problèmes : Calculer $f(i), \forall i \leq n$
- Appels récursifs : $f(i) \rightarrow f(i-1), f(i-2)$
- Cas terminaux : $f(0), f(1)$

$$\Rightarrow \text{Calculer } f(i-1) \text{ avant} f(i)$$

$$\Rightarrow \text{Calculer } f(i) \text{ pour } i \text{ de } 0 \text{ à } n$$

Suite de fibonacci : Version iterative

Soit $T : [\![0, n]\!] \to \mathbb{N}$ un tableau où initialement toutes les cases sont vides.

Fonction $f(i)$
  Si $i \leq 1$ **Alors**
    Renvoyer i
  Renvoyer
$f(i-1) + f(i-2)$

Fonction $f_{Iter}(n)$
  $T(0) \leftarrow 0$
  $T(1) \leftarrow 1$
  Pour i de 2 à n **Faire**
    $T(i) \leftarrow T(i-1) + T(i-2)$
  Renvoyer $T(n)$

Somme de sous-ensemble : Version itérative

- Liste de tous les sous-problèmes :
  Calculer $f(i, b), \forall i \leq n, b \leq B$
- Appels récursifs : $f(i, b) \rightarrow f(i-1, b-p_i), f(i-1, b)$
- Cas terminaux : $f(0, b), \forall b$

$\Rightarrow$ Calculer $f(i-1, b), \forall b \leq B,$ avant$f(i, b), \forall b \leq B$

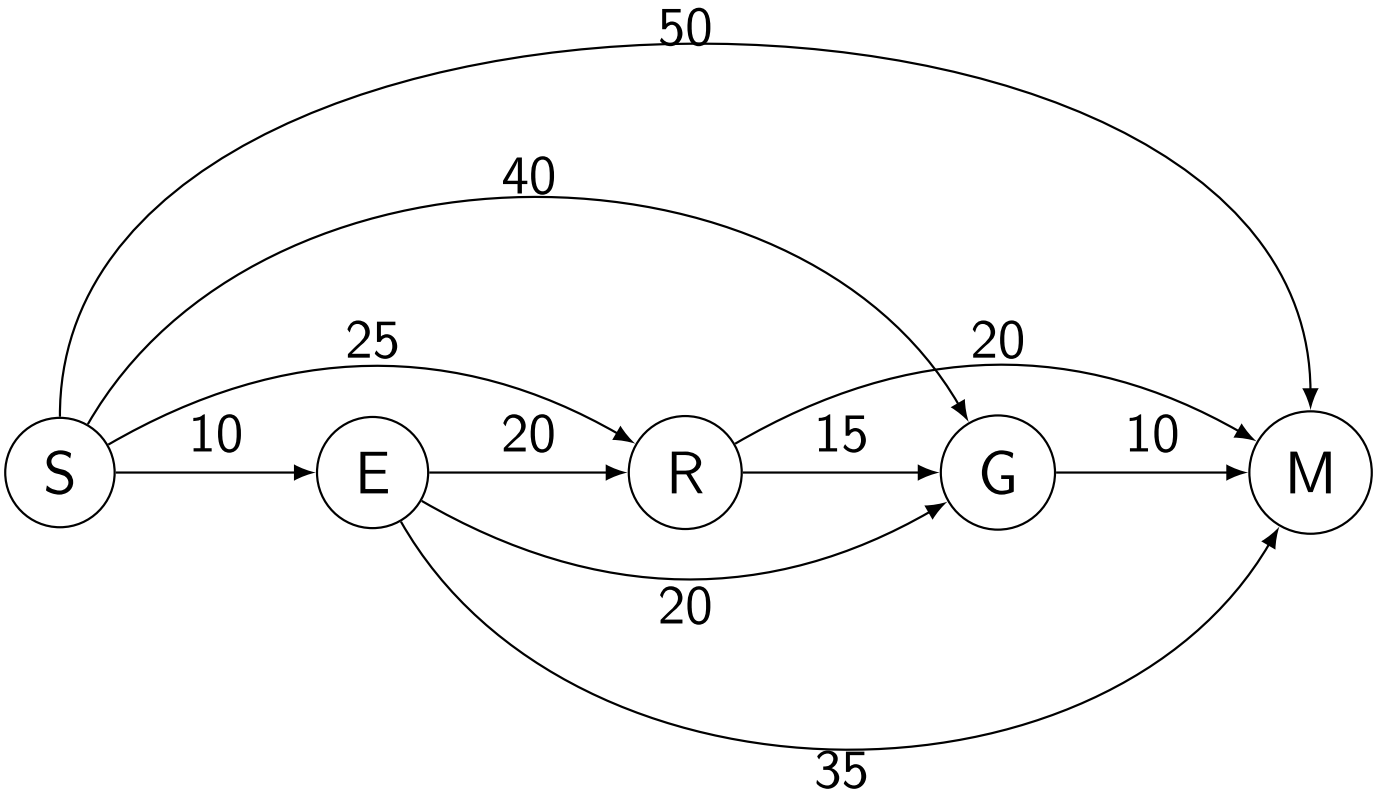
$\Rightarrow$ Calculer $f(i, b)$ pour i de 0 à $n,$ pour b de 0 à B

Somme de sous-ensemble : Version itérative

Soit $T : [\![0; n]\!] \times [\![0; B]\!] \to \{\top, \bot\}$ un tableau où initialement toutes les cases sont vides.

Fonction $f(i, b)$
  Si $b < 0$ **Alors**
    Renvoyer $\bot$
  Si $i = 0$ **Alors**
    Renvoyer $(b = 0)$
  Renvoyer
$f(i-1, b-p_i) \vee f(i-1, b)$

Fonction $f_{Iter}(n, B)$
  Pour b de 0 à B **Faire**
    $T(0, b) \leftarrow (b = 0)$
  Pour i de 1 à n **Faire**
    Pour b de 0 à B **Faire**
      Si $b \geq p_i$ **Alors**
        $T(i, b) \leftarrow T(i-1, b-p_i) \vee T(i-1, b)$
      Sinon
        $T(i, b) \leftarrow T(i-1, b)$
  Renvoyer $T(n, B)$


50
40
25
20
10
20
15
10
S
E
R
G
M
20
35

Plus court chemin : Version itérative

- Liste de tous les sous-problèmes

$$\text{Calculer } d(v), \forall v$$

- Appels récursifs

$$d(v) \rightarrow d(u), \forall u \in \Gamma^+(v)$$

- Cas terminaux

$$d(M)$$

$$\Rightarrow \text{Calculer } d(u), \forall u \in \Gamma^+(v), \text{ avant} d(v)$$

$$\Rightarrow \text{Calculer } d(u) \text{ dans l'ordre topologique inverse.}$$

Plus court chemin : Version itérative

Soit $T : V \to \mathbb{N}$ un tableau où initialement toutes les cases sont vides.

Fonction $d(v)$
  Si $v = M$ **Alors**
    Renvoyer 0
  Renvoyer
$\min_{u \in \Gamma^+(v)} (d(u) + \omega(v,u))$

Fonction $d_{Iter}(S)$
  $T(M) = 0$
  $L \leftarrow$ Ordre topologique inverse de $G \backslash M$
  Pour $v \in L$ **Faire**
    $T(v) \leftarrow \min_{u \in \Gamma^+(v)} (T(u) + \omega(v,u))$
  Renvoyer $T(S)$

De manière générale

Programmation dynamique : version itérative

Si on a une fonction $f : X \to Y$ récursive,

- lister tous les sous-problèmes de $f(x)$
- lister tous les types d'appels récursifs de f
- lister tous les cas terminaux de f
- Calculer en premier les cas terminaux, stocker les résultats dans un tableau T.
- Remonter les appels récursifs jusqu'à avoir résolu le problème initial en stockant dans T au fur et à mesure.

Complexité : Mémoïsation Fibonacci

1: **Fonction** $f_{Memo}(i)$
2: **Si** $T(i)$ est vide **Alors**
3: **Si** $i \leq 1$ **Alors**
4: **Renvoyer** i
5: $T(i) \leftarrow f_{Memo}(i-1) + f_{Memo}(i-2)$
6: **Renvoyer** $T(i)$

- Lignes 3 à 5 faites au plus une fois par $i \leq n$
- $\Rightarrow$ Au plus $2n$ appels récursifs à f_{Memo}, deux par $i \leq n$.
- $\Rightarrow$ Au plus n appels avec les lignes 3 à 5 en temps $O(1)$ et au plus n sans ces lignes, en temps $O(1)$ aussi.

$\Rightarrow$ Complexité : $O(n+n) = O(n)$

Complexité : Mémoïsation Somme de sous ensembles

1: **Fonction** $f_{Memo}(i, b)$
2: **Si** $T(i, b)$ est vide **Alors**
3: **Si** $b < 0$ **Alors**
4: **Renvoyer** $\perp$
5: **Si** $i = 0$ **Alors**
6: **Renvoyer** $(b = 0)$
7: $T(i, b) \leftarrow f_{Memo}(i - 1, b - p_i) \vee f_{Memo}(i - 1, b)$
8: **Renvoyer** $T(i, b)$

- Lignes 3 à 7 faites au plus une fois par $i \leq n, b \leq B$
- $\Rightarrow$ Au plus $2nB$ appels récursifs à f_{Memo}, deux par $i \leq n, b \leq B$.
- $\Rightarrow$ Au plus nB appels avec les lignes 3 à 7 en temps $O(1)$ et au plus nB sans ces lignes, en temps $O(1)$ aussi.

$\Rightarrow$ Complexité : $O(nB + nB) = O(nB)$

Complexité : Mémoïsation Somme de sous ensembles

On rappelle que $G = (V, E)$, on note $n = |V|$ et $m = |E|$.

1: **Fonction** $d_{Memo}(v)$
2: **Si** $T(v)$ est vide **Alors**
3: **Si** $v = M$ **Alors**
4: **Renvoyer** 0
5: $T(v) \leftarrow \min_{u \in \Gamma^+(v)} (d_{Memo}(u) + \omega(v, u))$
6: **Renvoyer** $T(v)$

- Lignes 3 à 5 faites au plus une fois par nœud $v \in V$
- $\Rightarrow$ Au plus $m = \sum_{v \in V} deg(v)$ appels récursifs à d_{Memo}, $deg(v)$ par $v \in V$.
- $\Rightarrow$ Au plus n appels avec les lignes 3 à 5 en temps $O(deg(v))$ et au plus $m - n$ sans ces lignes, en temps $O(1)$.

$\Rightarrow$ Complexité : $O(\sum_{v \in V} deg(v) + m - n) = O(2m - n) = O(m)$

Version itératives

Les versions itératives et mémoïsation ont (souvent) la même complexité : dans le pire cas, le tableau T est rempli complètement.

Attention aux cas particuliers.

Des fois, ça ne marche pas.

Explosion combinatoire récursive

On dispose de règles de transformations de nombre binaire en un autre nombre binaire **plus grand**. Par exemple

- $0 \xrightarrow{(1)} 10$
- $1 \xrightarrow{(2)} 01$
- $101 \xrightarrow{(3)} 01001$
- $010 \xrightarrow{(4)} 0111$

On peut ensuite transformer des nombres en d'autres à l'aide des règles (1) à (4) :

$\underline{0} \xrightarrow{(1)} \underline{1}0 \xrightarrow{(2)} \underline{010} \xrightarrow{(4)} 0\underline{11} \xrightarrow{(2)} 0\underline{1011} \xrightarrow{(3)} \ldots$

Question : connaissant deux nombres x et y binaires et un ensemble de m règles de transformation $\mathcal{T}$, puis-je transformer x en y à l'aide des règles de $\mathcal{T}$?

Explosion combinatoire récursive

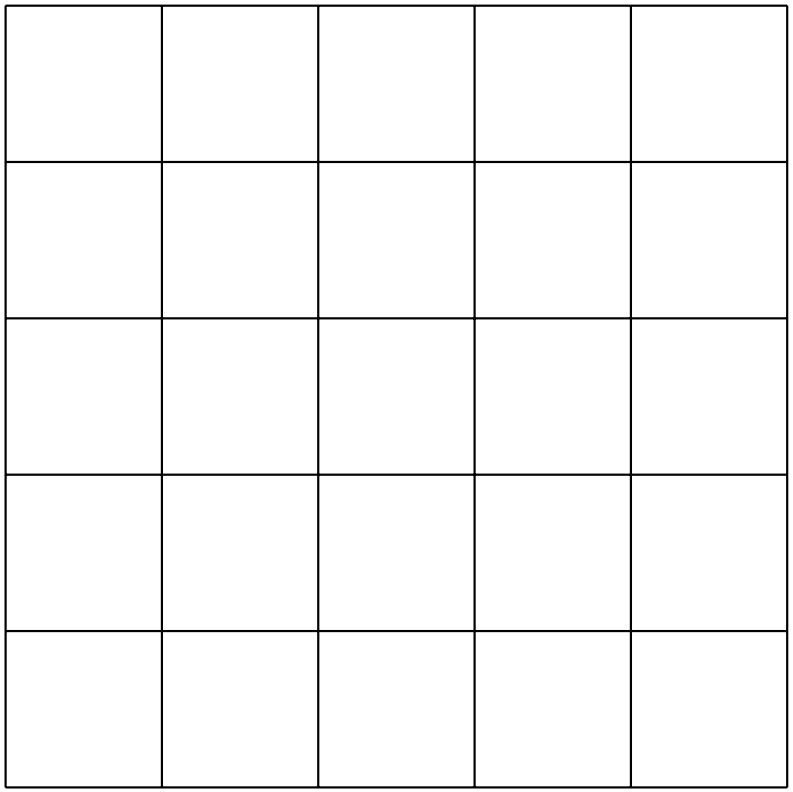
Il existe bien un algorithme de programmation dynamique pour résoudre ce problème.

Fonction $f(x, y)$
  Si $|x| > |y|$ **Alors**
    Renvoyer FAUX
  Sinon Si x = y **Alors**
    Renvoyer VRAI
  Si $T(x, y)$ est vide **Alors**
    $T(x, y) \leftarrow$ FAUX
    Pour tout règle $t \in \mathcal{T}$ **Faire**
      Pour tout manière d'appliquer t à x **Faire**
        $T(x, y) \leftarrow T(x, y) \vee f(t(x), y)$
  Renvoyer $T(x, y)$

Explosion combinatoire récursive

Mais cet algorithme nous oblige à enregistrer dans T jusqu'à $2^{|y|-|x|}$ informations.

Récursion ratée



Problème des carrés latins

Entrée

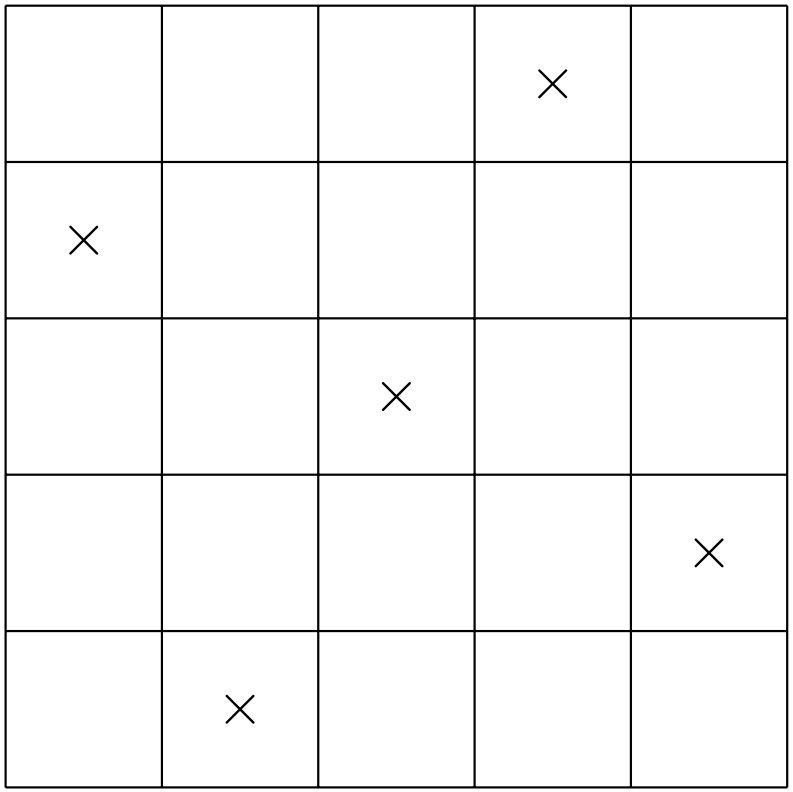
Une grille vide de $n \times n$ cases

Sortie

n cases telles que

- exactement une case par ligne et par colonne
- **maximiser** la distance minimale entre deux cases choisies

Récursion ratée



Problème des carrés latins

Entrée

Une grille vide de $n \times n$ cases

Sortie

n cases telles que

- exactement une case par ligne et par colonne
- **maximiser** la distance minimale entre deux cases choisies

Récursion ratée

Problème pour générer une fonction récursive...
Rien d'évident n'est apparent.

Au programme, mais vu en TD

- Comment retrouver la solution et non pas uniquement la valeur de la solution ? (par exemple, comment trouver le plus court chemin et pas uniquement la valeur du plus court chemin ; comment trouver le sous-ensemble et pas uniquement prouver son existence . . .)
- Algorithme de Roy-Floyd-Warshall : Plus cours chemins de tous les nœuds vers tous les nœuds.