ALG3 : récursivité et programmation dynamique

loig.jezequel@univ-nantes.fr

Suites récursives

Suite

Une suite est une famille d'éléments (typiquement des réels) appelés termes, indexée par les entiers naturels. On la note en général (u_n) et on note son n-ième terme u_n.

Définition par le terme général

On définit u_n en fonction de n.

Exemples

$u_n = n$, $u_n = n!$, $u_n = 1/n$, $u_n = 2 * n$, etc.

Définition récursive

On définit u_n en fonction des termes précédents $(u_0, u_1, \ldots, u_{n-1})$. Dans ce cas on doit initialiser la suite : donner la valeur de ses premiers termes

Exemples

- $u_n = n \times u_{n-1}$ avec $u_0 = 1$,
- $u_n = u_{n-1} + u_{n-2}$ avec $u_0 = 0$ et $u_1 = 1$, etc.

Calcul itératif du n-ième terme d'une suite récursive simple

Suite simple

On considère une suite (u_n) telle que le terme u_n ne dépend que du terme u_{n-1} : $u_n = f(u_{n-1})$.

Algorithme (calcul de u_n)

Soit $i = 0$ et soit $u_i = u_0$. Tant que $i < n$, poser $u_i = f(u_i)$ puis ajouter 1 à i. Retourner u_i.

Exemple

On veut calculer u_5 lorsque $u_n = 2 \times u_{n-1} + 1$ et $u_0 = 0$.

Calcul itératif du n-ième terme d'une suite récursive simple

Suite simple

On considère une suite (u_n) telle que le terme u_n ne dépend que du terme u_{n-1} : $u_n = f(u_{n-1})$.

Algorithme (calcul de u_n)

Soit $i = 0$ et soit $u_i = u_0$. Tant que $i < n$, poser $u_i = f(u_i)$ puis ajouter 1 à i. Retourner u_i.

Exemple

On veut calculer u_5 lorsque $u_n = 2 \times u_{n-1} + 1$ et $u_0 = 0$.

1. $u_i = 2 \times 0 + 1 = 1$, $i = 1$

Calcul itératif du n-ième terme d'une suite récursive simple

Suite simple

On considère une suite (u_n) telle que le terme u_n ne dépend que du terme u_{n-1} : $u_n = f(u_{n-1})$.

Algorithme (calcul de u_n)

Soit $i = 0$ et soit $u_i = u_0$. Tant que $i < n$, poser $u_i = f(u_i)$ puis ajouter 1 à i. Retourner u_i.

Exemple

On veut calculer u_5 lorsque $u_n = 2 \times u_{n-1} + 1$ et $u_0 = 0$.

1. $u_i = 2 \times 0 + 1 = 1$, $i = 1$
2. $u_i = 2 \times 1 + 1 = 3$, $i = 2$

Calcul itératif du n-ième terme d'une suite récursive simple

Suite simple

On considère une suite (u_n) telle que le terme u_n ne dépend que du terme u_{n-1} : $u_n = f(u_{n-1})$.

Algorithme (calcul de u_n)

Soit $i = 0$ et soit $u_i = u_0$. Tant que $i < n$, poser $u_i = f(u_i)$ puis ajouter 1 à i. Retourner u_i.

Exemple

On veut calculer u_5 lorsque $u_n = 2 \times u_{n-1} + 1$ et $u_0 = 0$.

1. $u_i = 2 \times 0 + 1 = 1$, $i = 1$
2. $u_i = 2 \times 1 + 1 = 3$, $i = 2$
3. $u_i = 2 \times 3 + 1 = 7$, $i = 3$

Calcul itératif du n-ième terme d'une suite récursive simple

Suite simple

On considère une suite (u_n) telle que le terme u_n ne dépend que du terme u_{n-1} : $u_n = f(u_{n-1})$.

Algorithme (calcul de u_n)

Soit $i = 0$ et soit $u_i = u_0$. Tant que $i < n$, poser $u_i = f(u_i)$ puis ajouter 1 à i. Retourner u_i.

Exemple

On veut calculer u_5 lorsque $u_n = 2 \times u_{n-1} + 1$ et $u_0 = 0$.

1. $u_i = 2 \times 0 + 1 = 1$, $i = 1$
2. $u_i = 2 \times 1 + 1 = 3$, $i = 2$
3. $u_i = 2 \times 3 + 1 = 7$, $i = 3$
4. $u_i = 2 \times 7 + 1 = 15$, $i = 4$

Calcul itératif du n-ième terme d'une suite récursive simple

Suite simple

On considère une suite (u_n) telle que le terme u_n ne dépend que du terme u_{n-1} : $u_n = f(u_{n-1})$.

Algorithme (calcul de u_n)

Soit $i = 0$ et soit $u_i = u_0$. Tant que $i < n$, poser $u_i = f(u_i)$ puis ajouter 1 à i. Retourner u_i.

Exemple

On veut calculer u_5 lorsque $u_n = 2 \times u_{n-1} + 1$ et $u_0 = 0$.

1. $u_i = 2 \times 0 + 1 = 1$, $i = 1$
2. $u_i = 2 \times 1 + 1 = 3$, $i = 2$
3. $u_i = 2 \times 3 + 1 = 7$, $i = 3$
4. $u_i = 2 \times 7 + 1 = 15$, $i = 4$
5. $u_i = 2 \times 15 + 1 = 31$, $i = 5$

Calcul itératif du n-ième terme d'une suite récursive simple

Suite simple

On considère une suite (u_n) telle que le terme u_n ne dépend que du terme u_{n-1} : $u_n = f(u_{n-1})$.

Algorithme (calcul de u_n)

Soit $i = 0$ et soit $u_i = u_0$. Tant que $i < n$, poser $u_i = f(u_i)$ puis ajouter 1 à i. Retourner u_i.

Exemple

On veut calculer u_5 lorsque $u_n = 2 \times u_{n-1} + 1$ et $u_0 = 0$.

1. $u_i = 2 \times 0 + 1 = 1$, $i = 1$
2. $u_i = 2 \times 1 + 1 = 3$, $i = 2$
3. $u_i = 2 \times 3 + 1 = 7$, $i = 3$
4. $u_i = 2 \times 7 + 1 = 15$, $i = 4$
5. $u_i = 2 \times 15 + 1 = 31$, $i = 5$
6. $i \geq 5$, on retourne $u_i = 31$

Calcul récursif du n-ième terme d'une suite récursive simple

Suite simple

On considère une suite (u_n) telle que le terme u_n ne dépend que du terme u_{n-1} : $u_n = f(u_{n-1})$.

Fonction récursive

Une fonction récursive est une fonction qui s'appelle elle-même.

Algorithme (calcul récursif de u_n)

On appelle $u(i)$ la fonction qui retourne u_0 si $i = 0$ et qui retourne $f(u(i-1))$ sinon. Retourner $u(n)$.

Exemple

On veut calculer u_5 lorsque $u_n = 2 \times u_{n-1} + 1$ et $u_0 = 0$.

Calcul récursif du n-ième terme d'une suite récursive simple

Suite simple

On considère une suite (u_n) telle que le terme u_n ne dépend que du terme u_{n-1} : $u_n = f(u_{n-1})$.

Fonction récursive

Une fonction récursive est une fonction qui s'appelle elle-même.

Algorithme (calcul récursif de u_n)

On appelle $u(i)$ la fonction qui retourne u_0 si $i = 0$ et qui retourne $f(u(i-1))$ sinon. Retourner $u(n)$.

Exemple

On veut calculer u_5 lorsque $u_n = 2 \times u_{n-1} + 1$ et $u_0 = 0$.

ret.

Calcul récursif du n-ième terme d'une suite récursive simple

Suite simple

On considère une suite (u_n) telle que le terme u_n ne dépend que du terme u_{n-1} : $u_n = f(u_{n-1})$.

Fonction récursive

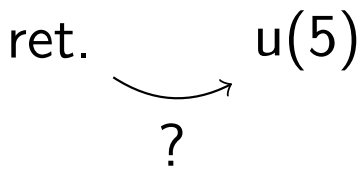
Une fonction récursive est une fonction qui s'appelle elle-même.

Algorithme (calcul récursif de u_n)

On appelle $u(i)$ la fonction qui retourne u_0 si $i = 0$ et qui retourne $f(u(i-1))$ sinon. Retourner $u(n)$.

Exemple

On veut calculer u_5 lorsque $u_n = 2 \times u_{n-1} + 1$ et $u_0 = 0$.



Calcul récursif du n-ième terme d'une suite récursive simple

Suite simple

On considère une suite (u_n) telle que le terme u_n ne dépend que du terme u_{n-1} : $u_n = f(u_{n-1})$.

Fonction récursive

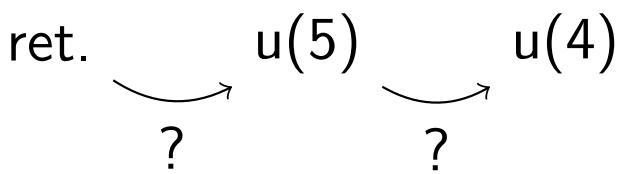
Une fonction récursive est une fonction qui s'appelle elle-même.

Algorithme (calcul récursif de u_n)

On appelle $u(i)$ la fonction qui retourne u_0 si $i = 0$ et qui retourne $f(u(i-1))$ sinon. Retourner $u(n)$.

Exemple

On veut calculer u_5 lorsque $u_n = 2 \times u_{n-1} + 1$ et $u_0 = 0$.



Calcul récursif du n-ième terme d'une suite récursive simple

Suite simple

On considère une suite (u_n) telle que le terme u_n ne dépend que du terme u_{n-1} : $u_n = f(u_{n-1})$.

Fonction récursive

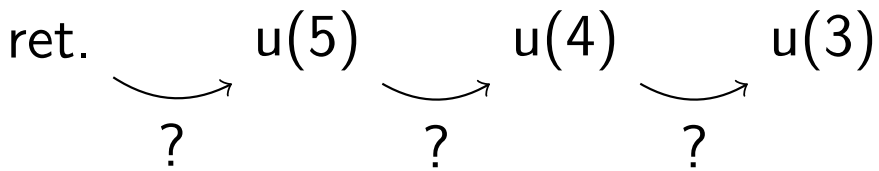
Une fonction récursive est une fonction qui s'appelle elle-même.

Algorithme (calcul récursif de u_n)

On appelle $u(i)$ la fonction qui retourne u_0 si $i = 0$ et qui retourne $f(u(i-1))$ sinon. Retourner $u(n)$.

Exemple

On veut calculer u_5 lorsque $u_n = 2 \times u_{n-1} + 1$ et $u_0 = 0$.



Calcul récursif du n-ième terme d'une suite récursive simple

Suite simple

On considère une suite (u_n) telle que le terme u_n ne dépend que du terme u_{n-1} : $u_n = f(u_{n-1})$.

Fonction récursive

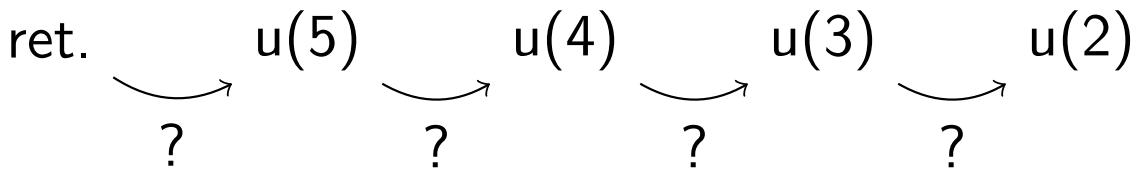
Une fonction récursive est une fonction qui s'appelle elle-même.

Algorithme (calcul récursif de u_n)

On appelle $u(i)$ la fonction qui retourne u_0 si $i = 0$ et qui retourne $f(u(i-1))$ sinon. Retourner $u(n)$.

Exemple

On veut calculer u_5 lorsque $u_n = 2 \times u_{n-1} + 1$ et $u_0 = 0$.



Calcul récursif du n-ième terme d'une suite récursive simple

Suite simple

On considère une suite (u_n) telle que le terme u_n ne dépend que du terme u_{n-1} : $u_n = f(u_{n-1})$.

Fonction récursive

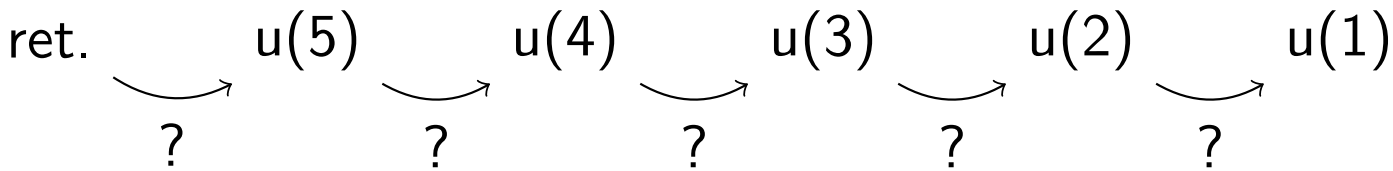
Une fonction récursive est une fonction qui s'appelle elle-même.

Algorithme (calcul récursif de u_n)

On appelle $u(i)$ la fonction qui retourne u_0 si $i = 0$ et qui retourne $f(u(i-1))$ sinon. Retourner $u(n)$.

Exemple

On veut calculer u_5 lorsque $u_n = 2 \times u_{n-1} + 1$ et $u_0 = 0$.



Calcul récursif du n-ième terme d'une suite récursive simple

Suite simple

On considère une suite (u_n) telle que le terme u_n ne dépend que du terme u_{n-1} : $u_n = f(u_{n-1})$.

Fonction récursive

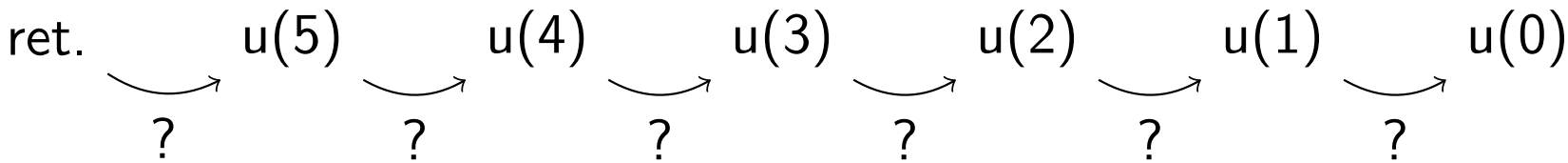
Une fonction récursive est une fonction qui s'appelle elle-même.

Algorithme (calcul récursif de u_n)

On appelle $u(i)$ la fonction qui retourne u_0 si $i = 0$ et qui retourne $f(u(i-1))$ sinon. Retourner $u(n)$.

Exemple

On veut calculer u_5 lorsque $u_n = 2 \times u_{n-1} + 1$ et $u_0 = 0$.



Calcul récursif du n-ième terme d'une suite récursive simple

Suite simple

On considère une suite (u_n) telle que le terme u_n ne dépend que du terme u_{n-1} : $u_n = f(u_{n-1})$.

Fonction récursive

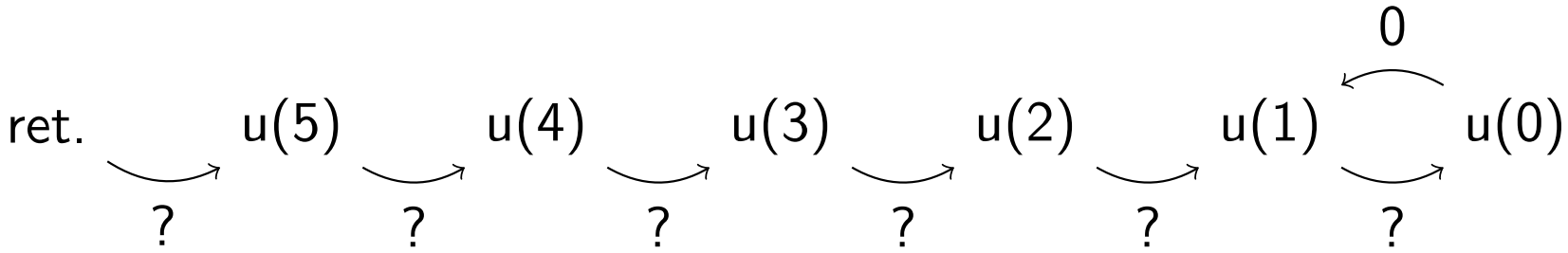
Une fonction récursive est une fonction qui s'appelle elle-même.

Algorithme (calcul récursif de u_n)

On appelle $u(i)$ la fonction qui retourne u_0 si $i = 0$ et qui retourne $f(u(i-1))$ sinon. Retourner $u(n)$.

Exemple

On veut calculer u_5 lorsque $u_n = 2 \times u_{n-1} + 1$ et $u_0 = 0$.



Calcul récursif du n-ième terme d'une suite récursive simple

Suite simple

On considère une suite (u_n) telle que le terme u_n ne dépend que du terme u_{n-1} : $u_n = f(u_{n-1})$.

Fonction récursive

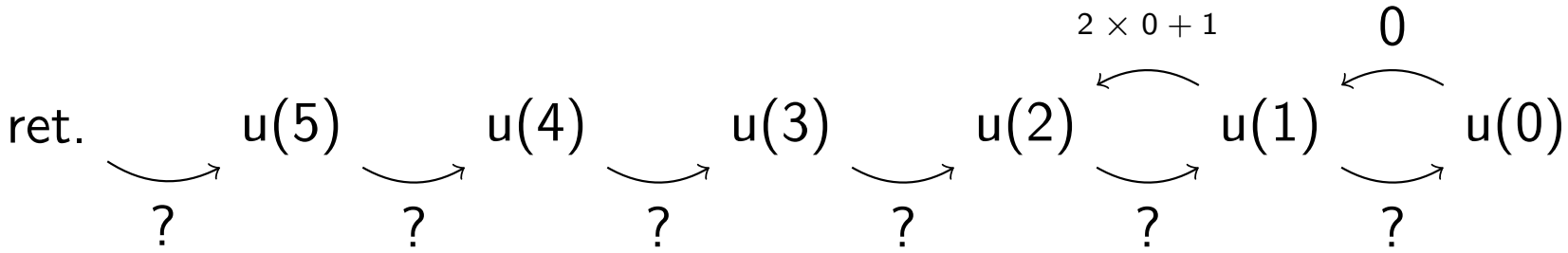
Une fonction récursive est une fonction qui s'appelle elle-même.

Algorithme (calcul récursif de u_n)

On appelle $u(i)$ la fonction qui retourne u_0 si $i = 0$ et qui retourne $f(u(i-1))$ sinon. Retourner $u(n)$.

Exemple

On veut calculer u_5 lorsque $u_n = 2 \times u_{n-1} + 1$ et $u_0 = 0$.



Calcul récursif du n-ième terme d'une suite récursive simple

Suite simple

On considère une suite (u_n) telle que le terme u_n ne dépend que du terme u_{n-1} : $u_n = f(u_{n-1})$.

Fonction récursive

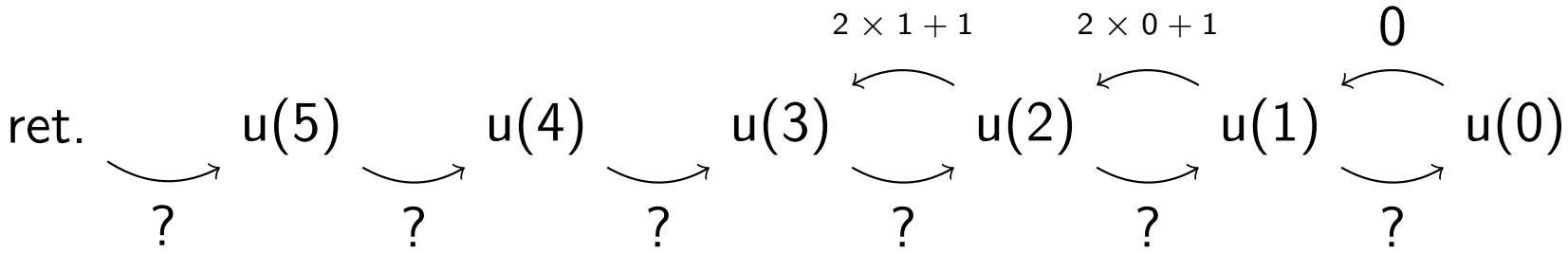
Une fonction récursive est une fonction qui s'appelle elle-même.

Algorithme (calcul récursif de u_n)

On appelle $u(i)$ la fonction qui retourne u_0 si $i = 0$ et qui retourne $f(u(i-1))$ sinon. Retourner $u(n)$.

Exemple

On veut calculer u_5 lorsque $u_n = 2 \times u_{n-1} + 1$ et $u_0 = 0$.



Calcul récursif du n-ième terme d'une suite récursive simple

Suite simple

On considère une suite (u_n) telle que le terme u_n ne dépend que du terme u_{n-1} : $u_n = f(u_{n-1})$.

Fonction récursive

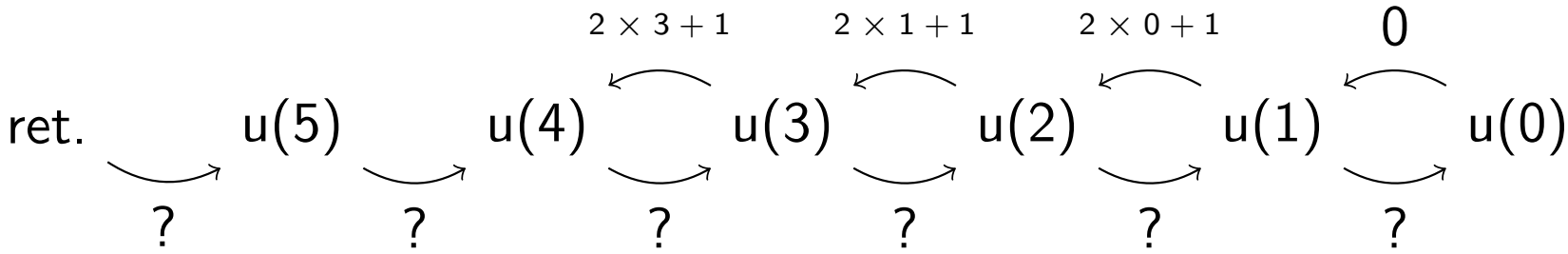
Une fonction récursive est une fonction qui s'appelle elle-même.

Algorithme (calcul récursif de u_n)

On appelle $u(i)$ la fonction qui retourne u_0 si $i = 0$ et qui retourne $f(u(i-1))$ sinon. Retourner $u(n)$.

Exemple

On veut calculer u_5 lorsque $u_n = 2 \times u_{n-1} + 1$ et $u_0 = 0$.



Calcul récursif du n-ième terme d'une suite récursive simple

Suite simple

On considère une suite (u_n) telle que le terme u_n ne dépend que du terme u_{n-1} : $u_n = f(u_{n-1})$.

Fonction récursive

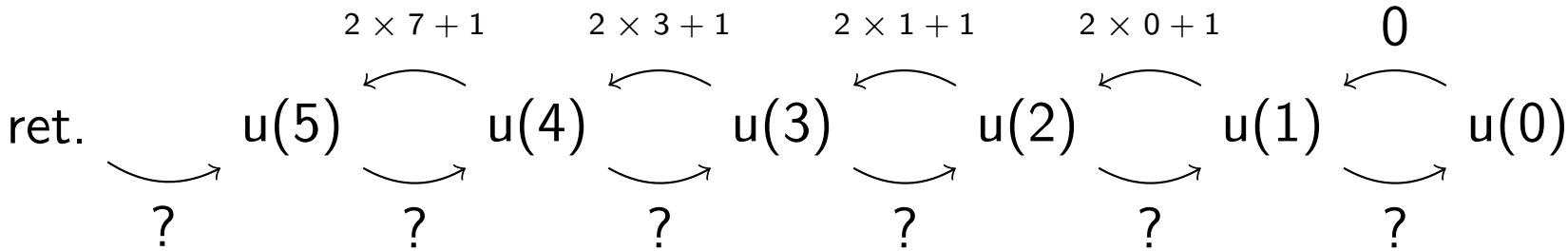
Une fonction récursive est une fonction qui s'appelle elle-même.

Algorithme (calcul récursif de u_n)

On appelle $u(i)$ la fonction qui retourne u_0 si $i = 0$ et qui retourne $f(u(i-1))$ sinon. Retourner $u(n)$.

Exemple

On veut calculer u_5 lorsque $u_n = 2 \times u_{n-1} + 1$ et $u_0 = 0$.



Calcul récursif du n-ième terme d'une suite récursive simple

Suite simple

On considère une suite (u_n) telle que le terme u_n ne dépend que du terme u_{n-1} : $u_n = f(u_{n-1})$.

Fonction récursive

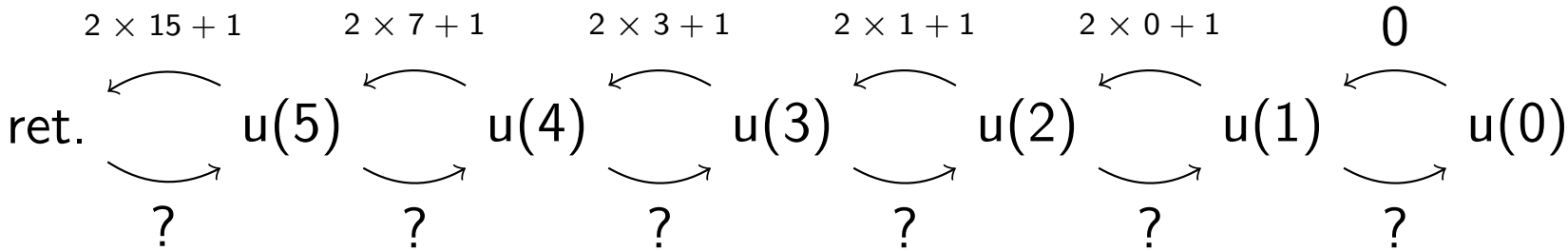
Une fonction récursive est une fonction qui s'appelle elle-même.

Algorithme (calcul récursif de u_n)

On appelle $u(i)$ la fonction qui retourne u_0 si $i = 0$ et qui retourne $f(u(i-1))$ sinon. Retourner $u(n)$.

Exemple

On veut calculer u_5 lorsque $u_n = 2 \times u_{n-1} + 1$ et $u_0 = 0$.



À propos de la récursivité

Gros avantage

Les solutions à certains problèmes s'écrivent très simplement de manière récursive mais sont compliquées à écrire correctement de manière itérative (par des boucles).

Exemple (code Go)

Étant donné un tableau t contenant des caractères et un entier $k \leq len(t)$, donner toutes les chaînes de k caractères que l'on peut construire en utilisant au plus une fois chaque caractère de t.

- Pour t = | a | b | c | d | et $k = 2$ on doit trouver *ab*, *ac*, *ad*, *ba*, *bc*, *bd*, *ca*, *cb*, *cd*, *da*, *db* et *dc*.
- Pour t = | a | b | c | d | et $k = 3$ on doit trouver *abc*, *abd*, *acb*, *acd*, *adb*, *adc*, *bac*, *bad*, *bca*, *bcd*, *bda*, *bdc*, *cab*, *cad*, *cba*, *cbd*, *cda*, *cdb*, *dab*, *dac*, *dba*, *dbc*, *dca*, *dcb*.

Autres exemples plus tard

Tri fusion, algorithmes sur les arbres

À propos de la récursivité, suite

Soyez prudents

- Un appel de fonction est plus long qu'un tour de boucle.
- Les appels de fonctions non résolus occupent la mémoire.

Récursivité terminale

Si l'appel récursif est la dernière chose qui se passe dans une fonction, le compilateur peut optimiser et éviter les appels de fonctions non résolus.

Remarque

Une fonction récursive peut toujours être rendue récursive terminale en utilisant la notion de continuation. Cependant, c'est un peu compliqué et on n'en parlera pas dans ce cours.

Un exemple un peu plus compliqué : la suite de Fibonacci

Histoire

Cette suite est tirée d'un problème proposé en 1202 par le mathématicien italien Fibonacci, mais elle avait été étudiée avant par les mathématiciens indiens pour résoudre d'autres problèmes.

Définition d'origine

Quelqu'un a déposé un couple de lapins dans un certain lieu, clos de toutes parts, pour savoir combien de couples seraient issus de cette paire en une année, car il est dans leur nature de générer un autre couple en un seul mois, et qu'ils enfantent dans le second mois après leur naissance.

Définition sous forme de suite

La suite de Fibonacci est la suite (u_n) telle que $u_n = u_{n-1} + u_{n-2}$ avec $u_0 = 0$ et $u_1 = 1$.

La suite de Fibonacci, suite

Implantation naïve (code Go)

Si on calcule simplement le n-ième terme de cette suite en appliquant la définition récursive, ce n'est pas très efficace.

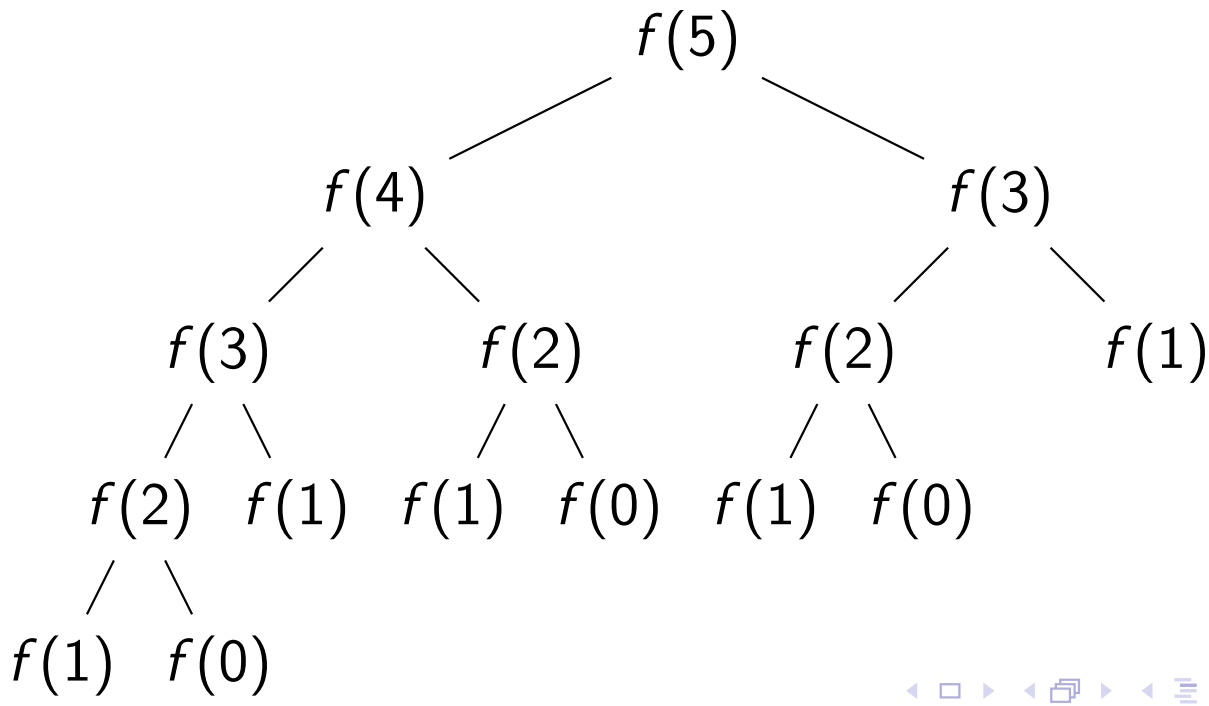
La suite de Fibonacci, suite

Implantation naïve (code Go)

Si on calcule simplement le n-ième terme de cette suite en appliquant la définition récursive, ce n'est pas très efficace.

Arbre des appels récursifs

On peut comprendre pourquoi en représentant l'arbre des appels récursifs correspondant (ici pour le calcul de $f(5)$).

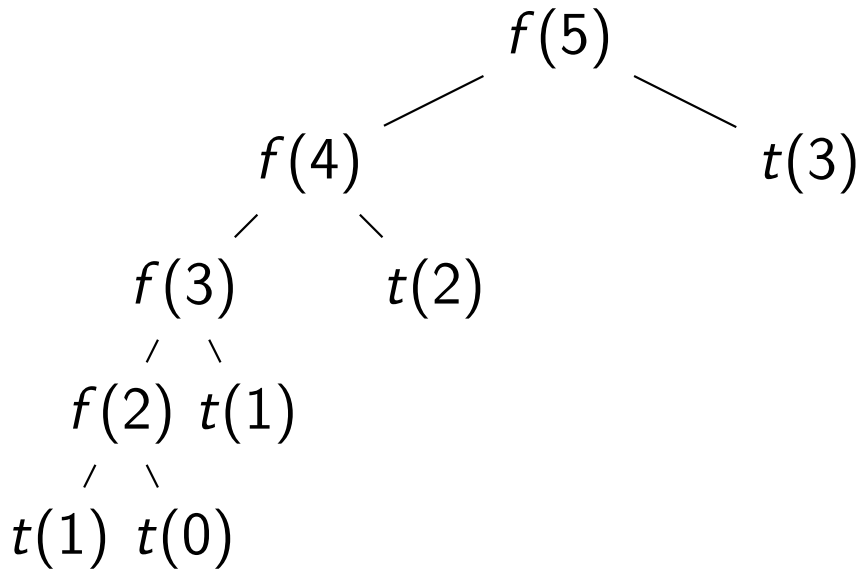


La suite de Fibonacci, le retour

Programmation dynamique

Dans une telle situation, on peut éviter les appels récursifs redondants en stockant dans un tableau la valeur obtenue lors du premier appel et en la récupérant lors des appels suivants.

Arbre des appels récursifs et des accès au tableau (code Go)



Il existe aussi une version sans tableau (qui ne stocke que deux valeurs à chaque instant) et récursive terminale. (code Go)

Un exemple plus compliqué de programmation dynamique

Sous-chaînes

Étant donnée une chaîne de caractères s on appelle sous-chaîne de s toute chaîne de caractères s' telle que $len(s') \leq len(s)$ et telle qu'il existe des entiers $i_0 < i_1 < \cdots < i_{len(s')-1}$ tels que quel que soit $0 \leq j < len(s')$ on a $0 \leq i_j < len(s)$ et $s'[j] = s[i_j]$.

Exemples

bisou, *ours*, *bison*, *bonus*, *sono*, et *iso* sont des quelques-unes des sous-chaînes de *bisounours*.

Problème de la plus longue sous-chaîne commune

Étant données deux chaînes de caractères s_1 et s_2, trouver la plus longue sous-chaîne de s_1 qui est aussi une sous-chaîne de s_2.

Exemple

Les sous-chaînes communes de *bisounours* et *souris* sont *s*, *o*, *u*, *r*, *i*, *so*, *su*, *sr*, *ss*, *ou*, *or*, *os*, *ur*, *us*, *rs*, *is*, *sou*, *sor*, *sos*, *sur*, *sus*, *srs*, *our*, *ous*, *ors*, *urs*, *sour*, *sous*, *ours*, et *sours*. La plus longue est *sours*.

Sous-chaînes et force brute

Première approche

Pour résoudre le problème on pourrait calculer toutes les sous-chaînes de s_1, puis vérifier une-à-une (en commençant par la plus grande) si ce sont des sous-chaînes de s_2.

Nombre de sous-chaînes

Une chaîne de caractères s de longueur n a jusqu'à 2^n sous-chaînes.

Vérifier qu'une sous-chaîne est contenue dans une chaîne

Étant données une chaîne s et une chaîne s', vérifier que s' est une sous chaîne de s peut impliquer de regarder un à un tous les caractères de s.

Ordre de grandeur du nombre d'opérations dans le pire cas

Si s_1 a pour longueur n et s_2 a pour longueur m, on fera de l'ordre de $2^n \times m$ opérations pour résoudre le problème.

Sous-chaînes, version récursive

Remarques fondamentales

Soient s_1' et s_2' des chaînes de caractères, soient c_1 et c_2 des caractères et soit s une plus grande sous-chaîne commune à $s_1 = s_1'c_1$ et $s_2 = s_2'c_2$.

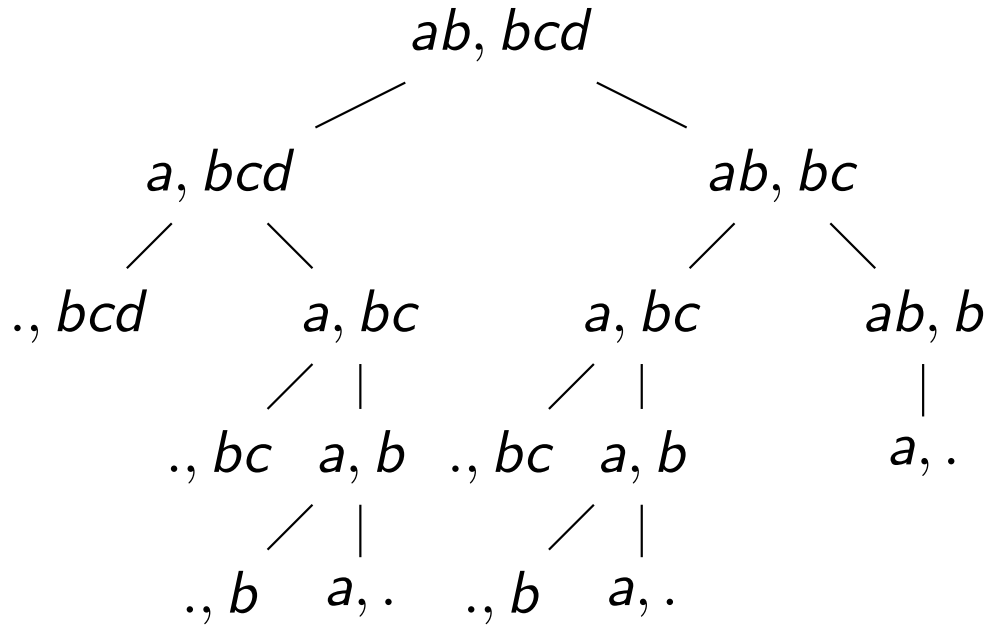
- Si $c_1 = c_2$ alors $s = s'c_1$ avec s' une plus grande sous-chaîne commune à s_1' et s_2'.
- Si $c_1 \neq c_2$ alors s est une plus grande sous-chaîne commune à s_1 et s_2' ou à s_1' et s_2.

Définition récursive d'une plus grande sous-chaîne commune

$$sc(s_1, s_2) = \begin{cases} \varepsilon & \texttt{si } len(s_1) = 0 \texttt{ ou } len(s_2) = 0 \\ sc(s_1', s_2')c & \texttt{si } s_1 = s_1'c \texttt{ et } s_2 = s_2'c \\ sc(s_1', s_2) & \texttt{si } s_1 = s_1'c_1, s_2 = s_2'c_2 \texttt{ avec } c_1 \neq c_2 \\ & \texttt{et } len(sc(s_1', s_2)) \geq len(sc(s_1, s_2')) \\ sc(s_1, s_2') & \texttt{si } s_1 = s_1'c_1, s_2 = s_2'c_2 \texttt{ avec } c_1 \neq c_2 \\ & \texttt{et } len(sc(s_1', s_2)) < len(sc(s_1, s_2')) \end{cases}$$

Plus grande sous-chaîne et programmation dynamique

Exemple d'arbre des appels récursifs pour $s_1 = ab$ et $s_2 = bcd$



Réduire les appels récursifs (code Go)

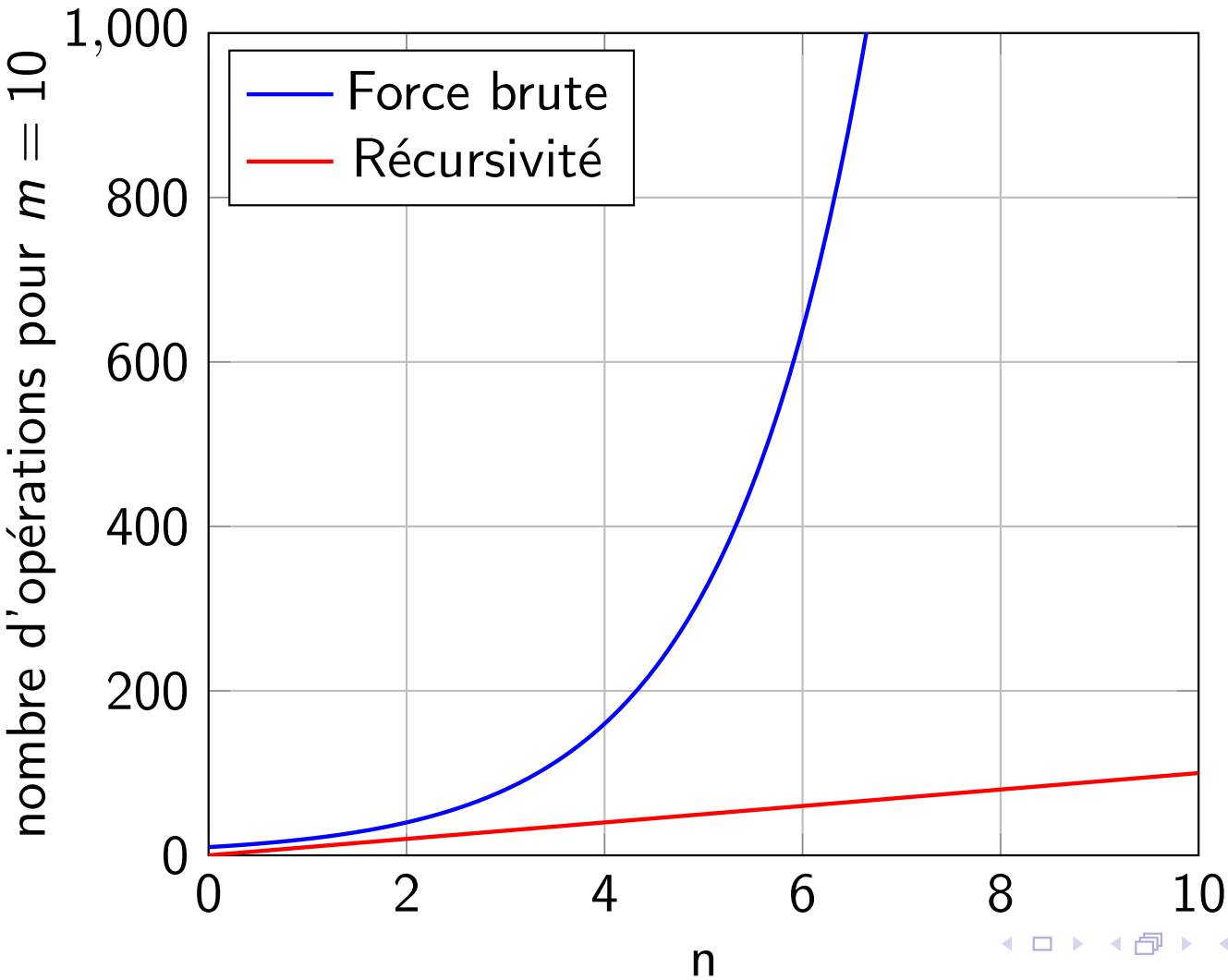
Utilisation d'un tableau t à deux dimensions, la case $t[i][j]$ indique la solution du problème pour les chaînes constituées des i premiers caractères de s_1 et des j premiers caractères de s_2.

Plus grande sous-chaîne et nombres d'opérations

On considère que $len(s_1) = n \leq len(s_2) = m$.

Force brute : de l'ordre de $2^n \times m$ opérations

Récursivité : de l'ordre de $n \times m$ opérations (remplir le tableau t)



Quand utiliser la programmation dynamique pour résoudre un problème ?

Sous-problèmes de même nature

Si on peut résoudre le problème en combinant les solutions de plusieurs problèmes plus petits mais de même nature que le problème d'origine (qui peuvent se résoudre de la même façon, en appliquant le même algorithme) alors on pourra le résoudre récursivement.

Chevauchement des sous-problèmes

Si le nombre de sous-problèmes différents résolus au cours d'une résolution récursive du problème et petit par rapport au nombres de sous-problèmes résolus, donc si on résout souvent les mêmes sous-problèmes, on pourra améliorer les performances grace à la programmation dynamique.