Introduction à l'algorithmique, structures de contrôle et de données

Stage IREM – Nov./Déc. 2010

Plan

Plan

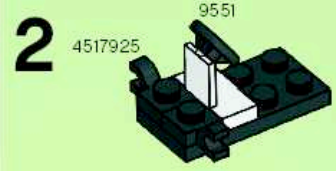
Vocabulaire

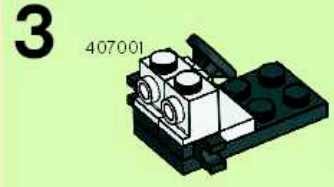
- un ordinateur = un objet, des ressources matérielles (le hardware).
- un algorithme = une recette, une méthode pour obtenir un résultat.
- un programme = un algorithme écrit dans un langage de programmation.
- un problème algorithmique = un problème à résoudre.

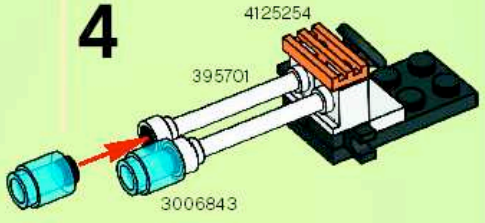
Vocabulaire

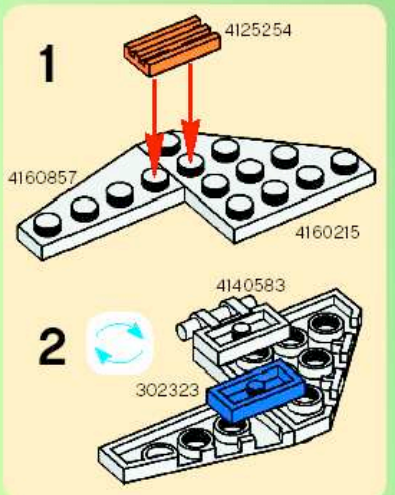
- un ordinateur = un objet, des ressources matérielles (le hardware).
- un algorithme = une recette, une méthode pour obtenir un résultat.
- un programme = un algorithme écrit dans un langage de programmation.
- un problème algorithmique = un problème à résoudre.

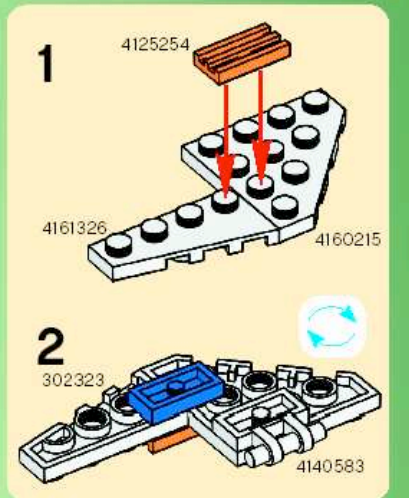
Des exemples !

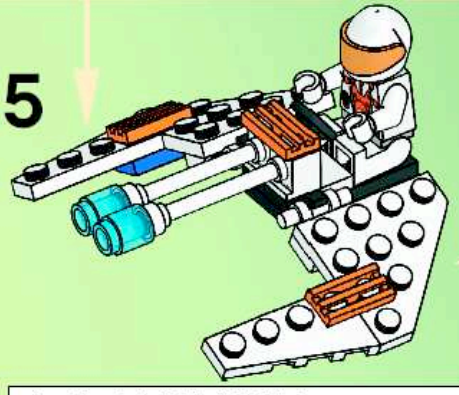












⚠ **AVERTISSEMENT :** RISQUE D'ÉTOUFFEMENT.
Contient des petites pièces. Ne convient pas aux enfants de moins de 3 ans.

⚠ **ADVERTENCIA:** PELIGRO DE ASFIXIA.
No recomendado para niños menores de 3 años.
Contiene piezas pequeñas.

SERVICE
www.LEGO.com

484 Émincé de rouget au pistou

24 heures à l'avance
Préparation : 40 mn - Cuisson : 30 à 35 mn

1 kg 500 rougets.
400 g pâtes.
500 g fenouil.
2 dl huile olive.
0 dl 5 vinaigre xérès.
Coulis de tomates.
1 citron.
Basilic.
Herbes de Provence.
Thym.
Vinaigrette.
Sel.
Poivre.

Écailler les rougets, les laver, puis soulever délicatement les filets et enlever les arêtes qui restent avec une pince à épiler. Mettre les filets dans un plat creux ; arroser avec de l'huile d'olive ; saupoudrer d'herbes de Provence et laisser mariner au frais, pendant 24 heures, le tout couvert par un torchon.
Cuire le fenouil à l'eau bouillante salée, citronnée et parfumée à l'huile d'olive et avec une brindille de thym (15 à 20 mn). Égoutter.
Cuire les pâtes à l'eau bouillante salée, huilée, pendant 12 mn. Égoutter. Rafraîchir. Tenir au chaud.
Pendant ces cuissons préparer le coulis de tomates (41).
Assaisonner le fenouil coupé en tranches avec la vinaigrette. Tenir au chaud. Assaisonner les pâtes avec le basilic (à volonté) haché et un mélange d'huile d'olive et de vinaigre de xérès. Disposer les pâtes sur le plat, recouvrir avec la fondue de fenouil.
Rapidement, cuire à la poêle Téfal, à feu très vif, les filets marinés pour qu'ils deviennent dorés et croustillants. Saler. Poivrer et disposer en étoile les filets de rougets, sur le plat de légumes. Servir avec le coulis de tomates en saucière.

Algorithme

Un algorithme est la description univoque d'une méthode effective pour résoudre un problème, exprimée à l'aide d'une suite d'instructions élémentaires.

Problèmes algorithmiques

- Données : un entier a
  Résultat : la somme $1 + 2 + 3 + \ldots + a$
- Données : une liste de mots
  Résultat : la liste triée dans l'ordre alphabétique
- Données : une liste de mots
  Résultat : une liste de pages Web contenant ces mots
- Données : une carte, deux villes A et B
  Résultat : un plus court chemin entre A et B
- Données : un programme P, une donnée a
  Résultat : réponse à "est-il vrai que le résultat de P appliqué à a vaut 8a+5" ?
- ...

Un algorithme doit résoudre toutes les instances d'un problème.

Exemple

La recherche du minimum dans une liste (non vide) d'entiers...

Exemple

La recherche du minimum dans une liste (non vide) d'entiers...

1. Noter le premier entier de la liste
2. Pour tous les entiers suivants, faire :
   Si l'entier est inférieur à celui noté
   Alors remplacer celui-ci par le nouvel entier
3. Renvoyer le nombre noté.

Exemple

La recherche du minimum dans une liste (non vide) d'entiers...
En pseudo-code :

Recherche du minimum

Données : liste non vide d'entiers L

Résultat : entier m le plus petit de L

$m \leftarrow L(1)$
pour chaque i *allant de* 2 *à* $|L|$ **faire**
  si $L(i) < m$ **alors** $m \leftarrow L(i)$
retourner m

Exemple

La recherche du minimum dans une liste (non vide) d'entiers...
En pseudo-code :

Recherche du minimum

Données : liste non vide d'entiers L

Résultat : entier m le plus petit de L

$m \leftarrow L(1)$
pour chaque i *allant de* 2 *à* $|L|$ **faire**
  si $L(i) < m$ **alors** $m \leftarrow L(i)$
retourner m

Des instances :

- [3, 10, 5, 24, 2, 12]
- [10]

Un problème. . . des algorithmes ?

Pour un problème donné, il peut y avoir plusieurs algorithmes différents. . . ou aucun !

Un problème. . . des algorithmes ?

Pour un problème donné, il peut y avoir plusieurs algorithmes différents. . . ou aucun !

Lorsqu'il existe plusieurs algorithmes, on peut les comparer selon plusieurs critères :

- les idées sous-jacentes, leur structure (récursif / itératif, glouton, prog. dynamique, diviser pour régner,. . .),
- les structures de données utilisées,
- la complexité algorithmique (*i.e.* les ressources – temps, mémoire– nécessaires à son exécution).

Complexité d'un algorithme

Complexité en temps ou en espace mémoire.

Complexité d'un algorithme

Complexité en temps ou en espace mémoire.

Objectif : trouver un ordre de grandeur du nombre d'opérations élémentaires nécessaires à l'exécution de l'algorithme.

Pas en minutes ou microsecondes !

On veut une notion robuste : indépendante d'un ordinateur donné, d'un compilateur, d'un langage de programmation, *etc.* et exprimée en fonction de la taille de la donnée à traiter.

Complexité d'un algorithme

Complexité en temps ou en espace mémoire.

Objectif : trouver un ordre de grandeur du nombre d'opérations élémentaires nécessaires à l'exécution de l'algorithme.

Pas en minutes ou microsecondes !

On veut une notion robuste : indépendante d'un ordinateur donné, d'un compilateur, d'un langage de programmation, *etc.* et exprimée en fonction de la taille de la donnée à traiter.

opération élémentaire : opération qui prend un temps constant (ou presque).

(Recherche du minimum : $n - 1$ comparaisons sont faites.)

Complexité d'un algorithme

Coût de $\mathcal{A}$ sur x : l'exécution de l'algorithme $\mathcal{A}$ sur la donnée x requiert $C_{\mathcal{A}}(x)$ opérations élémentaires.

Complexité d'un algorithme

Coût de $\mathcal{A}$ sur x : l'exécution de l'algorithme $\mathcal{A}$ sur la donnée x requiert $C_{\mathcal{A}}(x)$ opérations élémentaires.

Complexité dans le pire cas :

$$\mathcal{C}_{\mathcal{A}}(n) \stackrel{\text{def}}{=} \max_{x.|x|=n} C_{\mathcal{A}}(x)$$

Complexité d'un algorithme

Coût de $\mathcal{A}$ sur x : l'exécution de l'algorithme $\mathcal{A}$ sur la donnée x requiert $C_{\mathcal{A}}(x)$ opérations élémentaires.

Complexité dans le pire cas :

$$\mathcal{C}_{\mathcal{A}}(n) \stackrel{\text{def}}{=} \max_{x.|x|=n} C_{\mathcal{A}}(x)$$

Complexité en moyenne

$$\mathcal{C}^{moy}_{\mathcal{A}}(n) \stackrel{\text{def}}{=} \sum_{x.|x|=n} p(x) \cdot C_{\mathcal{A}}(x)$$

p : distribution de probabilités sur les données de taille n.

Algorithmes efficaces. . . et au delà !

Quelques familles d'algorithmes selon leur complexité :

- les algorithmes sous-linéaires : par exemple, en $O(\log(n))$
- les algorithmes linéaires : $O(n)$
  et quasi-linaires : $O(n \cdot \log(n))$
- les algorithmes polynomiaux : $O(n^k)$

Algorithmes efficaces. . . et au delà !

Quelques familles d'algorithmes selon leur complexité :

- les algorithmes sous-linéaires : par exemple, en $O(\log(n))$
- les algorithmes linéaires : $O(n)$
  et quasi-linaires : $O(n \cdot \log(n))$
- les algorithmes polynomiaux : $O(n^k)$
- les algorithmes exponentiels : $O(2^{p(n)})$
- . . . doublement exponentiels : $O(2^{2^{p(n)}})$
- . . .

NB : $O(g(n))$ contient l'ensemble des fonctions majorées par un $c \cdot g(n)$ avec $c > 0$ au delà d'un certain n_0. . .

Algorithmes efficaces. . . ou pas !

Considérons un algorithme de complexité $f(n)$ qui permette de résoudre en une heure les instances de taille X d'un problème sur un ordinateur aujourd'hui.
Alors un ordinateur 1000 fois plus rapide permettrait en une heure de résoudre les instances de taille. . .

- $1000 \cdot X$ si $f(n) = n$,

Algorithmes efficaces. . . ou pas !

Considérons un algorithme de complexité $f(n)$ qui permette de résoudre en une heure les instances de taille X d'un problème sur un ordinateur aujourd'hui.
Alors un ordinateur 1000 fois plus rapide permettrait en une heure de résoudre les instances de taille. . .

- $1000 \cdot X$ si $f(n) = n$,
- $31,6 \cdot X$ si $f(n) = n^2$,

Algorithmes efficaces. . . ou pas !

Considérons un algorithme de complexité $f(n)$ qui permette de résoudre en une heure les instances de taille X d'un problème sur un ordinateur aujourd'hui.
Alors un ordinateur 1000 fois plus rapide permettrait en une heure de résoudre les instances de taille. . .

- $1000 \cdot X$ si $f(n) = n$,
- $31,6 \cdot X$ si $f(n) = n^2$,
- $X + 9,97$ si $f(n) = 2^n$.

(voir "Algorithmics, the spirit of computing", D. Harel)

A la recherche du bon algorithme

Étant donné un problème, on cherche donc des algorithmes :

- corrects : qu'ils calculent bien ce que l'on veut,
- efficaces : nécessitant des ressources (temps, mémoire) raisonnables.

A la recherche du bon algorithme

Étant donné un problème, on cherche donc des algorithmes :

- corrects : qu'ils calculent bien ce que l'on veut,
- efficaces : nécessitant des ressources (temps, mémoire) raisonnables.

Peu de problèmes admettent des algorithmes utilisables en pratique.

Il est important de savoir concevoir, vérifier et analyser des algorithmes.

Un exemple d'algorithme très efficace...

Problème :

Rechercher une définition dans un dictionnaire, un numéro dans un annuaire,...

On commence avec la zone de recherche $(1 \to \text{fin})$, puis :

1. si la zone de recherche est vide, alors c'est fini
2. on ouvre le dictionnaire au "milieu" m de la zone de recherche $(d \to f)$...
3. si le mot recherche y est, c'est fini !
4. si on recommence (1) sur la zone réduite $(d \to m-1)$ ou $(m+1 \to f)$.

Problème :

Rechercher une définition dans un dictionnaire, un numéro dans un annuaire,...

On commence avec la zone de recherche $(1 \to \text{fin})$, puis :

1. si la zone de recherche est vide, alors c'est fini
2. on ouvre le dictionnaire au "milieu" m de la zone de recherche $(d \to f)$...
3. si le mot recherche y est, c'est fini !
4. si on recommence (1) sur la zone réduite $(d \to m-1)$ ou $(m+1 \to f)$.

A chaque fois, on divise la taille de l'espace de recherche par 2 !
Pour 1 millions de noms, 20 comparaisons (au max) suffisent pour résoudre le problème...

Recherche dichotomique

Rechercher(T, x, bg, bd)

si $bg > bd$ **alors retourner** *nil*
$m := (bg + bd)/2$
$(y, v) := T[m]$
si $x == y$ **alors retourner** v
sinon
    si $x < y$ **alors**
        retourner *Rechercher*$(T, x, bg, m - 1)$
    sinon
        retourner *Rechercher*$(T, x, m + 1, bd)$

T est un tableau de paires (x, v) trié selon les x...

Recherche dichotomique

Version itérative...

Rechercher(T, x)

$bg := 0$
$bd := |T| - 1$
tant que $bg \leq bd$ **faire**
    $m := (bg + bd)/2$
    $(y, v) := T[m]$
    si $x == y$ **alors retourner** v
    sinon
        si $x < y$ **alors** $bd := m - 1$
        sinon $bg := m + 1$
retourner *nil*

Plan

Stocker de l'information

Le "post-it" $\rightarrow$ une variable : une zone mémoire repérée par un *identificateur*.

Stocker de l'information

Le "post-it" $\rightarrow$ une variable : une zone mémoire repérée par un *identificateur*.

instruction d'affectation

- $X := exp$
- $X \leftarrow exp$
- $exp \rightarrow X$
- $X = exp$
- ...

Stocker de l'information

Le "post-it" $\rightarrow$ une variable : une zone mémoire repérée par un *identificateur*.

instruction d'affectation

- $X := exp$
- $X \leftarrow exp$
- $exp \rightarrow X$
- $X = exp$
- ...

X peut être remplacée par n'importe quel objet qui désigne une zone mémoire pouvant stocker une telle expression exp : $X[4]$, $z.m$, $(*p)$, ...

exp est une expression... construites à partir d'opérateurs de base, de constantes, de variables...

Instruction conditionnelle

instruction conditionnelle

`Si` C `Alors ...`

`Si` C `Alors ... Sinon ...`

C est une condition construite à partir de combinateurs Booléens ($\wedge$, $\vee$, $\neg$) et de conditions atomiques : $X == 3$, $T[i] < K$, $mot1 < mot2$, ...

Entrées/sorties

instruction d'entrées/sorties

`Afficher` *exp*

`Lire`(v)

NB : v est une variable désignant l'endroit de la mémoire **où** la donnée qui sera lue devra être stockée.

Boucles

boucles...

1. `Pour chaque` x `dans` S `Faire: ...`

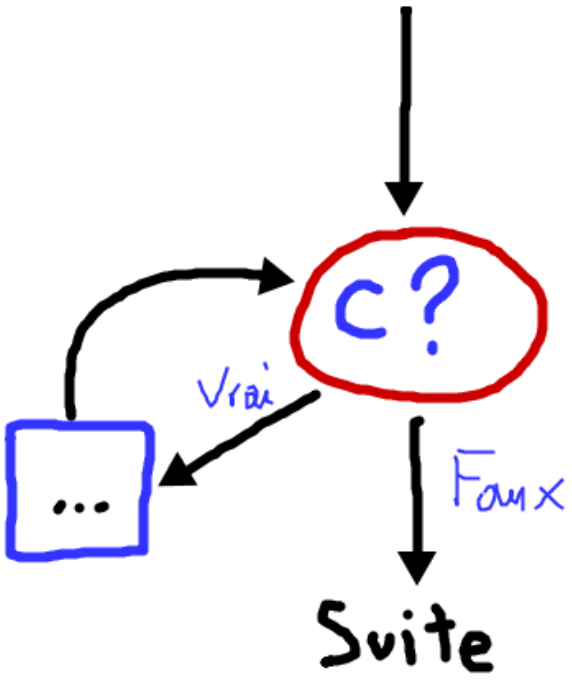
   (`Pour` $i = 1$ `à` 100 `Faire: ...`)

2. `Tant que` C `Faire: ...`

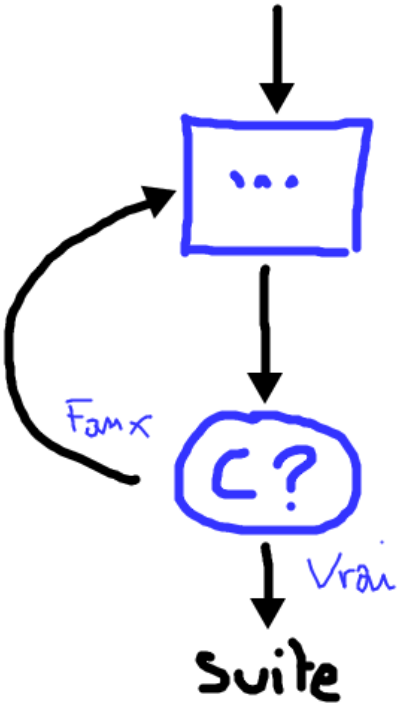
3. `Répéter ...jusqu'à` C

Boucles

`Tant que` C `Faire: ...`



`Répéter ... jusqu'à` C



Les fonctions

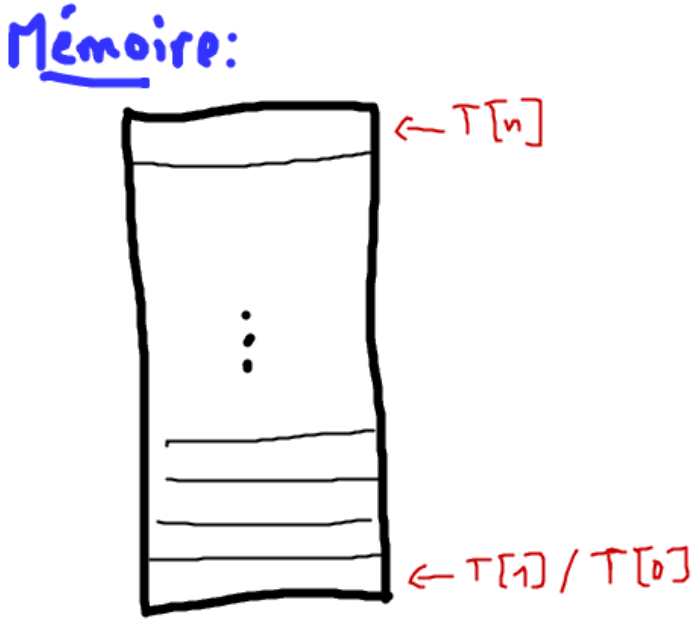
On peut isoler un algorithme dans une fonction :

un nom + des paramètres (et leurs types) + un type de résultat

Exemple : `RechercherMin(`T `: tableau d'entiers) : entier`

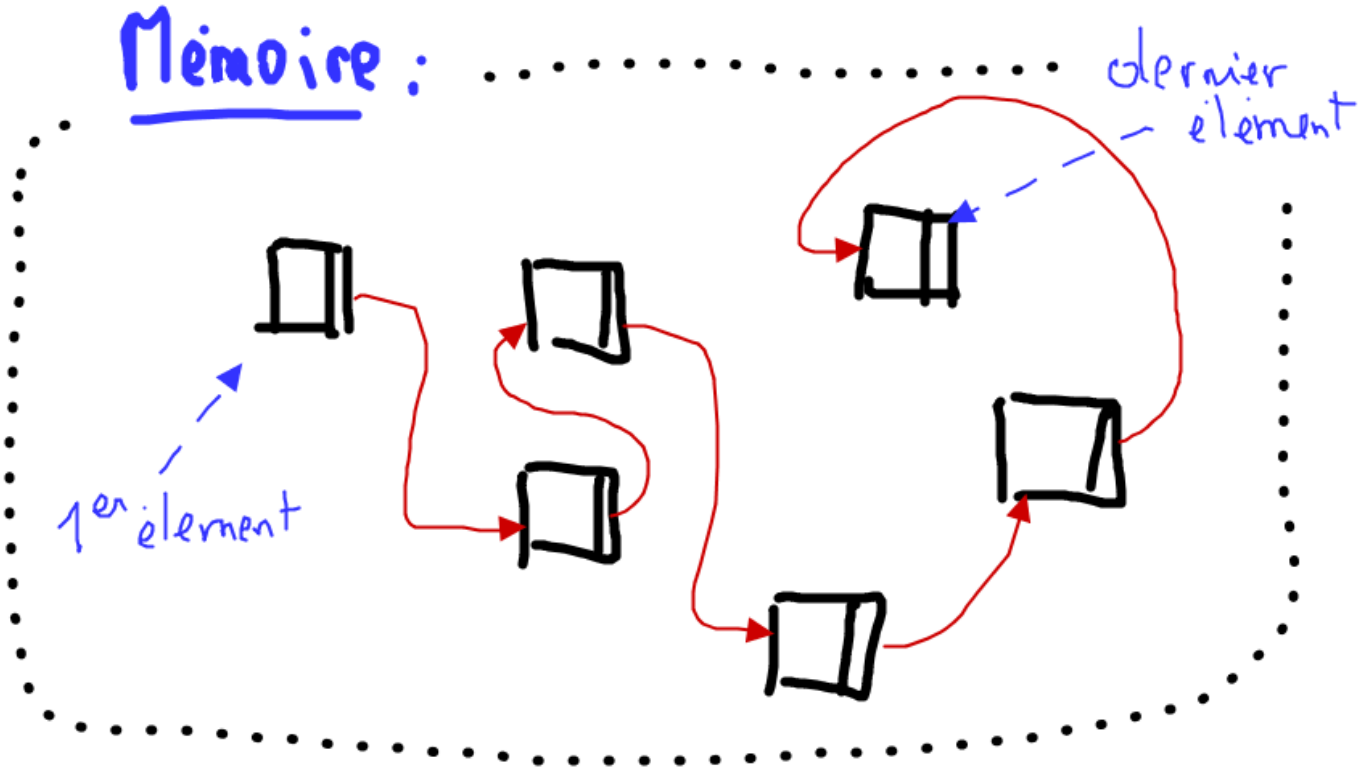
Structures de données

- Tableaux :
  - $T[1 \ldots 100]$, $M[1 \ldots 10][1 \ldots 10]$, $\ldots$
  - Taille fixe ou. . . modifiable (mais avec un coût).
  - Accès direct au i^{eme} élément.



Structures de données

- Listes :
  - Ajout en début ou en fin de liste, découpage, insertion... : faciles.
  - $L[i]$ ou longueur(L) : avec un coût.



Structure de données

La plupart des langages de programmation offrent des types de données très riches avec de nombreuses fonctions associées.

Exemple : les listes de Python.

Cela dépend des langages et attention au coût de ses opérations.

Autres. . . Les piles et les files

- Les piles : structure de stockage LIFO : “Last in, first out”
- Les files : structure de stockage FIFO : “First in, first out”