Introduction et notions fondamentales

1 Exemples de domaines informatiques

L'informatique ne se résume pas à l'utilisation des ordinateurs, comme le dit Edsger Dijkstra dans sa célèbre citation : "l'informatique n'est pas plus la science des ordinateurs que l'astronomie n'est celle des télescopes". La discipline est au contraire très vaste, et compte un grand nombre de domaines parmi lesquels :

- les bases de données
- les réseaux
- les systèmes embarqués
- la robotique
- la cybersécurité
- le block chain
- la cryptographie
- le calcul scientifique
- l'intelligence artificielle
- l'interface homme-machine, humain-machine ou utilisateur
- le développement web
- la recherche opérationnelle et l'optimisation (plannings, projets...)
- la vérification de programmes (empirique et méthodique)
- la preuve de programmes (formel, s'appuie sur la logique)
- la logique

Ces domaines se divisent souvent eux-mêmes en sous-domaines : ainsi, on trouve dans l'intelligence artificielle les algorithmes génétiques (qui donnent des solutions approchées à des problèmes), le machine learning, l'inférence, l'aide à la décision (exemple : partage équitable...) ou encore le choix social computationnel (exemples : modèles de vote, découpage des territoires, cartes électorales...).

2 Informatique sans ordinateur : les algorithmes

Définition (*algorithme*) :

> Un algorithme est une méthode systématique à base d'opérations qui permet de résoudre toutes les instances d'un problème.

Les deux sections de cette partie sont consacrées à l'étude rapide de deux exemples d'algorithme.

2.1 Compter en binaire

Lorsque l'on compte sur les doigts en employant la méthode naturelle, on compte en unaire : chaque doigt compte pour 1 et on peut donc compter au maximum jusqu'à 10.

Cependant, si l'on attribue à chaque doigt de la main un poids deux fois plus grand que celui du doigt précédent, en commençant par 1 pour le pouce, on peut compter jusqu'à $2^0+2^1+2^2+2^3+2^4 = 2^5-1 = 31$: on compte alors en binaire.

Propriété : De facon générale, pour tout $n \in \mathbb{N}$ on a $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$.

Preuve :
Pour $n \in \mathbb{N}$, notons $S = \sum_{i=0}^{n} 2^i$. En l'écrivant différemment, on a une somme télescopique :

$$S = 2S - S = \sum_{i=0}^{n} 2^{i+1} - \sum_{i=0}^{n} 2^i = \sum_{j=1}^{n+1} 2^j - \sum_{j=0}^{n} 2^j = 2^{n+1} - 1$$

(on pouvait également procéder par récurrence sur n).

On propose à présent l'algorithme suivant, qui permet d'obtenir la "représentation" de tout entier de l'intervalle [0..31] dans le comptage en binaire sur les doigts.

Algorithme – Comptage en binaire

entrée : $n \in \mathbb{N}^*$
hypothèses : C

ommencer tous les doigts baissés
$i = 0$
Tant que $i < n$
    Si le pouce est baissé
        Lever le pouce
        $i \leftarrow i + 1$
    Sinon
        Baisser tous les doigts levés consécutifs au pouce
        Lever le doigt suivant
        Baisser le pouce
        $i \leftarrow i + 1$

2.2 Tours de Hanoï

On s'intéresse dans cet exemple au jeu des tours de Hanoï dont on rappelle brièvement les règles.

On dispose de n disques de diamètres allant de 1 à n (notés $(d_k)_{k\in[1..n]}$) et d'un plateau à trois piquets sur lesquels entre lesquels on peut les déplacer, sous les contraintes suivantes :

- on ne peut déplacer qu'un disque à la fois
- on n'a accès qu'aux disques situés au sommet d'un piquet lorsque l'on effectue un déplacement
- aucun disque ne peut être posé sur un disque de diamètre plus petit

Initialement, tous les disques se trouvent sur l'un des piquets (usuellement celui de gauche). Le but du jeu est alors de les déplacer tous vers un autre piquet (celui de droite souvent) en respectant les règles précédentes.

On propose une résolution du jeu de manière récursive (*cf.* chapitre 6 – "premiers pas en OCaml").

Notations :

Soit $k, n \in (\mathbb{N}^*)^2$ et x, y deux piquets quelconques parmi les trois du jeu. On note :
- $d_k : x \to y$ l'instruction consistant à déplacer d_k de x vers y (lorsque cela est possible)
- autre(x, y) le troisième piquet

Sous ces notations, l'algorithme suivant permet de résoudre n'importe quelle instance du jeu.

Algorithme – Hanoï

entrée : $n \in \mathbb{N}^*$ et x, y deux piquets
hypothèses : S

i $n = 1$ alors $d_1 : x \to y$
Sinon
Hanoï$(n-1, x, \text{autre}(x, y))$
$d_n : x \to y$
Hanoï$(n-1, \text{autre}(x, y), y)$

Propriété :

On définit la suite $(h_n)_{n\in\mathbb{N}^*} \in \mathbb{N}^{\mathbb{N}^*}$ de la manière suivante :

$$\forall\, n \in \mathbb{N}^*,\ h_n = \begin{cases} 1 \text{ si } n = 1 \\ 2h_{n-1} + 1 \text{ sinon} \end{cases}$$

Compte tenu du pseudo-code de l'algorithme précédent, h_n donne le nombre d'opérations nécessaires à Hanoï pour déplacer n disques entre deux piquets.

Il s'ensuit alors : $\forall\, n \in \mathbb{N}^*,\ h_n = 2^n - 1$.

Preuve :
Par une récurrence immédiate sur $n \in \mathbb{N}^*$.

Exercice : Développer Hanoï$(4, \text{A}, \text{C})$ de facon à ce qu'il ne reste à la fin que des déplacements.

3 Langage de programmation

Définition (*langage de programmation*) :

Un langage de programmation est une manière conventionnelle et formelle (*i.e.* avec un format bien précis) en vue de formuler des algorithmes intelligibles par les humains et exécutables par une machine.

Il est possible d'établir une analogie entre les langages de programmation et les langues parlées par les populations humaines. Les similitudes sont résumées dans le tableau de comparaison suivant.

	Langue naturelle	Langage de programmation
Alphabet	majuscules, minuscules, chiffres, ponctiuation (dont l'espace(*))	la même chose, avec en plus les symboles `@ # & / { } ( ) [ ]`, la tabulation(*), etc. (en fait, toute la table ASCII)
Mots	mots du dictionnaire, noms propres	mots-clé (`def`, `return`...), identificateurs (noms de variables, de fonctions)
Grammaire	phrases formées de mots selon les règles de la grammaire	bloc de code qui respecte la syntaxe
Sémantique	sens des phrases	interprétation du code

4 Problèmes et fonctions

4.1 Problèmes

Définition (*problème*) :

Un problème est la donnée :
- d'un format de paramètres d'entrée (décrivant quelles données on a en entrée)
- de la sortie attendue en fonction de ces paramètres.

Si la sortie attendue est un booléen (vrai V ou faux F), il s'agit d'un problème de décision.
Si la sortie attendue est une valeur maximisant ou minimisant une fonction donnée sur un ensemble donné, on

Exemple : RECTANGLE
entrée : $(a, b) \in \mathbb{N}^2$
sortie : la surface d'un rectangle de côtés a et b

Définition (*problème de décision*) :

Un problème de décision est un problème dont la sortie est un booléen (Vrai (V) ou Faux (F)).

Exemple : PARITÉ
entrée : $n \in \mathbb{N}$
sortie : Vrai si n est pair, Faux sinon

Définition (*problème d'optimisation*) :

Un problème d'optimisation est un problème dont la sortie attendue maximise ou minimise une fonction donnée sur un ensemble donné.

Exemples : SAC Á DOS
entrée : $P_{\max} \in \mathbb{R}^{+*}$
$(p_i)_{i\in[1..n]} \in (\mathbb{R}^+)^n$
$(v_i)_{i\in[1..n]} \in (\mathbb{R}^+)^n$
sortie : $(x_i^*)_{i\in[1..n]} \in \{0,1\}^n$ tel que $\sum_{i=1}^n x_i^* p_i \leq P_{\max}$ qui maximise

$$f = \begin{pmatrix} \{0,1\}^n \to \mathbb{R} \\ x \mapsto \sum_{i=1}^n x_i v_i \end{pmatrix}$$

PLUS COURT CHEMIN
entrée : un graphe orienté $\mathcal{G} = (V, E)$
un point de départ A $\in V$
un point d'arrivée B $\in V$
sortie : un chemin de longueur minimale de A à B dans $\mathcal{G}$

Définition (*instance d'un problème*) :

Une instance d'un problème est une famille de paramètres répondant au type de ses entrées.

Dans le cas particulier d'un problème de décision, on dira que l'instance est :
- positive si la sortie correspondante est Vrai
- négative si la sortie correspondante est Faux.

Exemples : 18 est une instance de Parité, de solution Vrai (c'est donc une instance positive).
17 est une autre instance de Parité, de solution Faux (c'est une instance négative).

$(7, 8)$ est une instance de Rectangle de solution 56.

4.2 Fonctions

Définition (*fonction en programmation*) :

> En programmation, une fonction peut être vue comme un rassemblement d'instructions qui s'appliquent sur des données que l'on peut faire varier, appelées arguments de la fonction.

Remarque : Si on attend généralement d'une fonction qu'elle donne ou retourne un résultat en sortie, elle peut aussi avoir des effets de bord comme des affichages ou des modifications de la mémoire (*cf.* fonctions de type `void` en C).

La résolution de problèmes tels que définis dans la section précédente passe en général par des fonctions. Quitte à décomposer des les problèmes difficiles en sous-problèmes, on essaiera d'avoir une fonction par problème.

Pour que ce découpage en fonctions ou en sous-problèmes soit lisible dans le code, on munit le code de chaque fonction d'une spécification qui contient :

- le nombre et le type des arguments
- le type de la sortie
- les hypothèses sur les arguments
- la valeur de la sortie pour les arguments (et éventuellement une description des effets de bord)

Remarque : Les deux premiers éléments énoncés précédemment constituent dans la spécification ce que l'on appelle la signature de la fonction, ou encore prototype.

Exemple : La fonction en C suivante permet de calculer l'aire d'un rectangle dont on connaît les deux dimensions, répondant ainsi au problème Rectangle de la section précédente :

```c
int surface_rectangle (int L1,int L2){
    // hyp : L1>=0 && L2 >= 0
    // retourne la surface d'un rectangle
    // de longueur L1et de largeur L2
    return (L1*L2);
}
```