Eléments de base

15 avril 2008

1 Introduction

L'objet de ce cours est d'introduire les outils et les méthodes permettant de répondre à la double question centrale suivante :

Etant donné un problème P,
(1) existe-t-il un algorithme pour le résoudre ?
et si oui
(2) avec quel coût (en espace et surtout en temps) ?

Au sens strict les deux parties de la question relèvent de deux domaines différents de l'informatique. Le premier est "la théorie de la calculabilité" (est-ce qu'un problème peut-être résolu ?) et le deuxième est "la théorie de la complexité" (sachant qu'un problème peut être résolu, quel est le coût de sa résolution ? est-ce que le coût est réaliste ?).

Ces deux domaines de l'informatique appartiennent à l'informatique théorique qui peut être définie comme étant l'étude des fondements mathématiques de l'informatique. Ce cours appartient donc à un ensemble de cours dans lequel on trouve notamment la théorie des graphes, la théorie des langages, la théorie de l'information et la logique computationnelle.

Le caractère théorique de ce cours ne signifie pas que ce qu'on y étudie est inutilisable dans la pratique, encore moins qu'il est "superflu" dans une formation d'ingénieurs. En effet, ce cours vous permettra de :

- savoir quels sont les problèmes qu'on ne peut pas résoudre sur ordinateur, autrement dit de connaître les limites de l'informatique,
- savoir évaluer la difficulté d'un problème pour savoir si on peut le résoudre de manière exacte ou s'il faut chercher une solution approchée,

– découvrir des problèmes connus pour être difficiles et que vous pouvez rencontrer dans votre professionnelle (l'ordonnancement par exemple).

Pour pouvoir répondre à la question énoncée ci-dessus, nous avons besoin d'éléments mathématiques et algorithmiques dont certains ont déjà été abordés dans d'autres cours. Dans ce chapitre nous introduisons (ou rappelons) ces éléments. Ainsi :

– Le premier paragraphe est consacré à des rappels sur la complexité des algorithmes.
– Dans le deuxième paragraphe nous rappelons les définitions principales de la théorie des graphes.
– Des éléments de la logique propositionnelle sont rappelés dans le troisième paragraphe.
– Le quatrième paragraphe contient des rappels sur la théorie des langages.
– La machine de Turing est introduite au cinquième paragraphe.

2 Algorithmes et complexité

Un algorithme est une suite d'instructions permettant de résoudre un problème. Prenons deux exemples : un algorithme de tri et un algorithme de recherche d'un élément dans un tableau trié.

– Donné : un tableau d'entiers tab de taille N.
– Résultat : le même tableau trié.

```
Pour taille=N, ..., 2 Faire
Début
   Pour i=1, ..., taille-1 Faire
   Début
      Si (tab[i]>tab[i+1] Alors
      Début
         tmp = tab[i]
         tab[i] = tab[i+1]
         tab[i+1] = tmp
      FinSi
   FinPour
FinPour
```

- Donné : un tableau d'entiers tab de taille N et un entier e
- Résultat : un booléen trouvé qui vaut VRAI si l'élément appartient au tableau et FAUX sinon

---

```
Inf = 1
Sup = N
trouvé = Faux
Tant que ((inf < sup) ET (trouvé = Faux))
Début
    Milieu = (inf + sup)/2
    Si (e = tab[milieu]) Alors Trouvé = Vrai
    Sinon
    Début
       Si (e < tab[milieu]) Alors Sup = milieu+1
       Sinon inf = milieu - 1
    FinSi
FinTantQue
```

2.1 Complexité des algorithmes

Pour être vraiment utile, un algorithme ne doit pas être trop coûteux en temps et en espace. Ceci est caractérisé par la complexité temporelle et spatiale.

La complexité (temporelle) d'un algorithme est le nombre d'opérations élémentaires dont il est composé. Une opération élémentaire est une opération dont le coût ne dépend pas du problème (addition, multiplication, permutation, comparaison, ...). La complexité d'un algorithme est exprimée en fonction de la taille du problème, c'est-à-dire la taille des données qu'on fournit à l'entrée de l'algorithme. Par exemple, dans le cas du tri, la taille du problème est celle du tableau que l'on souhaite trier.

On exprime la complexité en fonction de la taille N du problème avec la notation $\mathcal{O}(f(N))$: cela signifie que le nombre d'opérations, notons le $g(N)$, peut s'écrire sous la forme $\lambda f(N)+T(N)$ où $T(N)$ est un terme négligeable devant $f(N)$, ou ce qui revient au même, que pour les grandes valeurs de N, nous avons $g(N) \leq \lambda f(N)$.

Appliquons cela à nos deux exemples.

– Algorithme de tri :

1. La boucle principale est exécutée $(N-1)$ fois (une fois pour chaque valeur de taille).
2. Pour chaque valeur de $taille$, la deuxième boucle est exécutée $(taille - 1)$ fois.
3. Pour chaque itération de cette dernière boucle, c'est-à-dire pour chaque valeur de i, il y a au plus 4 opérations élémentaires : 1 comparaison et 3 affectations.
4. Au total, cet algorithme comporte donc $NbOp$ opérations élémentaires avec :

$$\begin{array}{rcl} NbOp & = & \sum_{taille=2}^{N} 4 * (taille - 1) \\ & = & 4 * \sum_{p=1}^{N-1} p \\ & = & 4 * \frac{(N-1)*N}{2} \\ & = & 2 * (N-1) * N \\ & = & 2 * N^2 - 2 * N \end{array}$$

5. La complexité de cet algorithme est donc en $\mathcal{O}(n^2)$.

– Algorithme de recherche :

1. Au pire, la boucle principale est exécutée tant que les valeurs de Inf et Sup vérifient $Inf \leq Sup$, c'est-à-dire pour des tableaux de taille N, $\frac{N}{2}$, $\frac{N}{4}$, ...,$\frac{N}{2^i}$,...,1.
2. Le nombre d'exécutions de cette boucle est donc de nb tque $\frac{N}{2^{nb-1}} = 1$ c'est-à-dire que $nb = logN + 1$.
3. Au pire, chaque étape est composée de 4 opérations élémentaires : 2 comparaisons et 2 affectations.
4. Le nombre total d'opérations élémentaires est donc $4 * (logN + 1)$.
5. La complexité de cet algorithme est donc en $\mathcal{O}log(N)$.

Une étude complète de la complexité des algorithmes n'est pas l'objet de ce cours. Notons juste que :

– une complexité logarithmique (comme celle de notre algorithme de recherche) est ce qu'on peut espérer de mieux. Elle correspond à des algorithmes d'une grande efficacité.
– Une complexité polynomiale est une bonne complexité, surtout si le degré du polynome est faible (ce qui est le cas de notre algorithme de tri).
– Un algoritme dont la complexité est exponentielle (par exemple $N!$) est inutilisable dès que la taille N du problème devient importante. Pour les

problèmes dont on n'a que des algorithmes de complexité exponentielle, on a souvent recours à des algorithmes de résolution approchée.

Deux dernières remarques pour clore ce paragraphe.

Remarque 2.1 *Nous avons introduit plus haut la notation $\mathcal{O}(f(N))$ et en avons donné la signification. Cette notation est la plus utilisée pour décrire la complexité d'un algorithme mais ce n'est pas la seule. Parmi les autres notations citons les deux suivantes :*

- $\Omega(f(N))$ *: pour les grandes valeurs de N, nous avons $g(N) \geq \mu f(N)$.*
- $\Theta(f(N))$ *: pour les grandes valeurs de N, nous avons $\mu f(N) \leq g(N) \leq \lambda f(N)$.*

Remarque 2.2 *Dans notre calcul de la complexité, nous avons considéré à chaque fois le nombre* ***maximal*** *d'opérations élémentaires. C'est ce qui s'appelle la* ***complexité au pire des cas****. Une autre façon de faire, est de considérer, le nombre moyen des opérations élémentaires (moyenne sur toutes les situations possibles), c'est la* ***complexité moyenne****.*

3 Graphes

Définition 3.1 *Un* ***graphe*** *G(S,A) est constitué de deux ensembles :*

- *l'ensemble S des* ***sommets*** *$S = \{s_1, ..., s_n\}$*
- *l'ensemble A des* ***arcs*** *$A = \{a_1, ..., a_m\}$. Un arc a_i est un couple de sommets (s_{i1},s_{i2}).*

De manière tout à fait naturelle, un graphe est représenté par un diagramme dans lequel un sommet est représenté par un point (un petit cercle) et un arc a_i par une flèche qui connecte les deux sommets s_{i_1} et s_{i_2}. La figure 1 montre un graphe dont les caractéristiques sont les suivantes :

- S = {A, B, C, D, E}
- A = {(A,B), (A,D), (B,C), (B,E), (C,A), ((E,C)}

Dans la définition donnée ci-dessus, l'ensemble A est constitué de couples : les deux sommets d'un arc ne jouent pas le même rôle. Ceci se traduit sur le diagramme par une orientation des courbes (flèche) représentant les arcs. Un tel graphe est dit orienté. Il est possible de faire abstraction de cette orientation et de considérer des paires de sommets. Dans ce cas, le graphe est dit non orienté et les liens entre les paires de sommets s'appellent des arêtes.

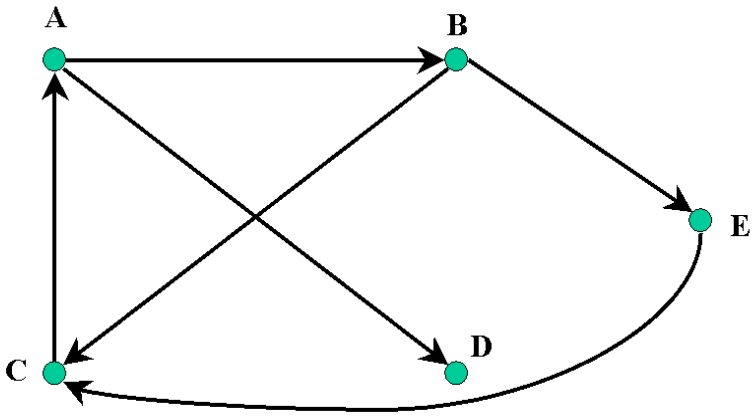

FIG. 1 – Exemple d'un graphe.

Définition 3.2 *Dans un graphe G(S,A) un* ***chemin*** *est une suite ordonnée de sommets (non nécessairement tous distincts)* $\mu = s_{i_1}, ..., s_{i_q}$ *telle que* $(s_{i_p} s_{i_{p+1}}) \in A$ *pour p=1,...,q-1.*

Définition 3.3 *Dans un graphe G(S,A) une* ***cha****îne est une suite ordonnée de sommets (non nécessairement tous distincts)* $\mu = s_{i_1}, ..., s_{i_q}$ *telle que* $(s_{i_p} s_{i_{p+1}}) \in A$ *ou* $(s_{i_{p+1}}, s_{i_p}) \in A$*pour p=1,...,q-1 et deux arêtes successives sont toujours distinctes.*

Définition 3.4 *Dans un graphe G(S,A) un* ***circuit*** *est une suite circulaire de sommets (non nécessairement tous distincts)* $\sigma = (s_{i_1}, ..., s_{i_q})$ *telle que* $(s_{i_p} s_{i_{p+1}}) \in A$ *et pour p=1,...,q-1 et* $(s_{i_q}, s_{i_1}) \in A$.

Définition 3.5 *Dans un graphe G(S,A) un* ***cycle*** *est une suite ordonnée de sommets (non nécessairement tous distincts)* $\sigma = (s_{i_1}, ..., s_{i_q})$ *telle que* $(s_{i_p} s_{i_{p+1}}) \in A$ *ou* $(s_{i_{p+1}}, s_{i_p}) \in A$ *pour p=1,...,q-1 et* $(s_{i_q}, s_{i_1}) \in A$ *ou* $(s_{i_1}, s_{i_q}) \in A$ *et deux arêtes successives sont toujours distinctes.*

Définition 3.6 *Un chemin, une chaîne, un circuit ou un cycle est dit* ***hamiltonien*** *lorsqu'il passe par tous les sommets une et une seule fois.*

Définition 3.7 *Un chemin, une chaîne, un circuit ou un cycle est dit* ***eulérien*** *lorsqu'il passe par tous les arcs une et une seule fois.*

Définition 3.8 *Un graphe orienté (resp. non orienté) est dit* ***valué*** *si on attribue une valeur (distance, coût, temps, ..) à chacun de ses arcs (resp arêtes).*

Définition 3.9 *Un arc a_i (s_{i1},s_{i2}) est dit* ***incident*** *vers l'intérieur de s_{i1} et incident vers l'extérieur de s_{i2}.*

Définition 3.10 *Dans un graphe orienté le demi degré intérieur (resp. extérieur) d'un sommet s est le nombre d'arcs incidents à s vers l'extérieur (resp. vers l'intérieur). Il sera noté $d^+(s)$ (resp. $d^-(s)$). Le* ***degré*** *du sommet noté $d(s)$ est la somme $d^+(s)+d^-(s)$ du degré intérieur et extérieur.*

Définition 3.11 *Un graphe est dit* ***complet*** *si toute paire de sommets y est reliée par une arête (ou un arc).*

Définition 3.12 *Dans un graphe $G(S, A)$ un* ***sous-graphe*** *est un graphe $G_1(S_1, A_1)$ tel que $S_1 = S$ et $A_1 \subseteq A$. Autrement dit, un sous-graphe est obtenu à partir d'un graphe en gardant tous les sommets et en supprimant des arêtes (ou des arcs).*

Définition 3.13 *Dans un graphe $G(S, A)$ un* ***graphe partiel*** *est un graphe $G_2(S_2, A_2)$ tel que $S_1 \subseteq S$ et $A_2 = A \bigcap (S_2 \times S_2)$. Autrement dit, un graphe partiel est obtenu en supprimant des sommets et toutes les arêtes (ou tous les arcs) qui ont un de ces sommets comme extrémité.*

Définition 3.14 *Dans un graphe $G(S, A)$ une* ***clique*** *est un sous-graphe complet.*

Outil important Outil important de modélisation mathématique, les graphes nous seront utiles dans ce cours pour formuler et étudier un bon nombre de problèmes illustrant les principaux concepts étudiés dans ce cours. Ainsi :

- C'est dans la théorie des graphes que nous irons chercher des problèmes connus pour être très coûteux en temps de résolution (Voyageur du commerce, ordonnancement, ...).
- C'est dans la théorie des graphes que nous irons chercher des problèmes qui nous permettront de disntinguer les classes de problèmes (Circuit hamiltonien, circuit eulérien, ...).

4 Eléments de logique

Dans ce paragraphe, nous nous intéressons exclusivement à la logique propositionnelle.

A la base de la logique propositionnelle il y a les **propositions** : des symbôles ayant une valeur de vérité. A partir de ces symbôles et à l'aide des connceteurs logiques nous pouvons construire des **formules bien formées** (fbf). Ce qui nous donne la définition suivante :

Définition 4.1

Une proposition est une formule bien formée.

Si A est une fbf, alors $\neg A$ est une fbf.

Si A et B sont des fbf, alors $A \vee B$ est une fbf.

Si A et B sont des fbf, alors $A \wedge B$ est une fbf.

Si A et B sont des fbf, alors $A \Longrightarrow B$ est une fbf.

Si A et B sont des fbf, alors $A \Longleftrightarrow B$ est une fbf.

La valeur de vérité d'une fbf dépend des valeurs de vérité des propositions qui y interviennent. Cette dépendance (ou encore le cacul des valeurs de vérité de la fbf) se déduit des tables de vérité des différentes connecteurs. Nous ne donnerons pas ici ces tabes de vérités, nous nous contentons de rappeler que :

- La table de vérité du connecteur $\Longrightarrow$ se déduit de la propriété : $A \Longrightarrow B$ est équivaente à $\neg A \vee B$.
- La table de vérité connecteur $\Longleftrightarrow$ se déduit de la propriété : $A \Longleftrightarrow B$ est équivaente à $(A \Longrightarrow B) \wedge (B \Longrightarrow A)$.

Ceci nous permet d'introduire les définitions suivantes :

Définition 4.2 *Etant donnée une fbf A. Soit Δ_A l'ensemble des propositions intervenant dans A. On appelle une* ***interprétation*** *de A une fonction de Δ_A dans $\{VRAI, FAUX\}$. Autrement dit une interprétaion de A est une valuation des propositions intervenant dans A.*

Définition 4.3 *Etant donnée une fbf A et une interprétation I de A. On dit que A est* ***satisfiable*** *par I si A a la valeur VRAI lorque les propositions intervenant dans A prennent les valeurs que leur associent I.*

Définition 4.4 *Etant donnée une fbf A et une interprétation I de A. I est un* ***modèle*** *de A si A est satisfiable par I.*

Définition 4.5 *Etant donnée une fbf A. A est dite* ***unsatisfiable*** *si elle n'a pas de modèle.*

Définition 4.6 *Etant donnée une proposition p, un* ***littéral*** *relatif à cette proposition est soit p soit $\neg p$.*

Définition 4.7 *Une* ***clause*** *C est une disjonction de littéraux $l_1 \vee l_2 \vee ... \vee l_n$.*

Définition 4.8 *Une* ***forme conjonctive normale (fcn)*** *est une conjonction de clauses $C_1 \wedge C_2 \wedge ... \wedge C_n$.*

5 Eléments de la théorie des langages

Définition 5.1 *Un **alphabet** Σ est un ensemble fini de symbôles.*

- Exemples d'alphabets :
  1. L'ensemble des lettres, $\Sigma_1 = \{a, b, c, ..., x, y, z\}$.
  2. L'ensemble des chiffres, $\Sigma_2 = \{0, 1, 2, ...,9\}$.
  3. L'alphabet binaire, $\Sigma_3 = \{0, 1\}$.
  4. L'ensemble $\Sigma_4 = \{a, b\}$.

Définition 5.2 *Un **mot** ω défini sur un alphabet Σ est une suite de symboles s_1, s_1, ..., s_n appartenant à l'alphabet Σ.*

- Exemples de mots :
  1. $\omega_1 =$ *eisti* est un mot construit sur Σ_1.
  2. $\omega_2 = 063423$ est un mot construit sur Σ_2.
  3. $\omega_3 = 01110$ est un mot construit sur Σ_3.
  4. $\omega_4 =$ *ababb* est un mot construit sur Σ_4.

Définition 5.3 *Un langage L défini sur un alphabet Σ est un ensemble (fini ou infini) de mots définis surΣ.*

- Exemples d'alphabets :
  1. $L_1 = \{$tous les mots de la langue française$\}$ est un langage défini sur Σ_1.
  2. $L_2 = \{$tous les numéros de téléphone$\}$ est un langage défini sur Σ_2.
  3. $L_4 = \{a^n b^n$ n$\geq 0\}$ est un langage défini sur Σ_4.

6 Machine de Turing

Définition 6.1 *Une machine de Turing est défini par un quadruplet (K, Σ, δ, q_0) où*

- *Σ est un **alphabet**,*
- *K est un ensemble fini d'**états**,*
- *$q_0 \in K$ est l'état **initial**,*
- *δ est une fonction, appelée la fonction de **transition**, de $\Sigma \times K$ vers $\Sigma \times K \times \{\rightarrow, \leftarrow, -\}$.*

Le fonctionnement d'une machine de Turing peut être décrit comme suit :

- Elle dispose d'une bande infinie sur laquelle il est possible de lire et d'écrire.
- Elle dispose d'une tête de lecture/écriture.
- Elle reçoit en entrée une chaîne de symbôles $s_o s_1 ... s_n$ où $s_i \in \Sigma$.
- Au départ elle est dans l'état q_o.
- Elle commence la lecture par le symbôle le plus à gauche de la chaîne.
- A chaque fois qu'un symbôle s est lu :
  - elle trouve la transition $(s, q) \rightarrow (s', q', dep)$,où q est l'état en cours.
  - elle écrit s'.
  - elle effectue le déplacement dep.
  - elle passe dans l'état q'.

L'exemple suivant permet d'illustrer le fonctionnement d'une machine de Turing. Soit donc la machine de Turing définie comme suit :

- Elle a un ensemble d'états Q contenant
  - un état initial q_d,
  - deux états de fin q_s et q_e,
  - et deux autres états q_0 et $q_1 2$.
- L'ensemble fini de valeurs que l'on peut lire ou écrire dans les cases de la bande est {0, 1, b}. Le symbole b est un symbole particulier que l'on appelle "blanc".
- Elle possède une fonction de transition qui à un état de la machine autre que q_e et q_s associe un triplet dont le premier élément indique le nouvel état de la machine, le deuxième le caractère qui sera écrit dans la case pointée et le troisième le déplacement à effectuer (ce déplacement sera donc noté $+1$, -1 ou 0). Cette fonction de transition est définie par le tableau suivant :

	0	1	b
q_d	(q_0,0,+1)	(q_1,1,+1)	(q_s,b,0)
q_0	(q_e,0,0)	(q_1,1,+1)	(q_s,b,0)
q_1	(q_0,0,+1)	(q_e,1,0)	(q_s,b,0)

- la machine s'arrête sur l'état q_e et q_s.

On vérifie aisément que cette machine sort en succès (état q_s) si la chaîne qui lui est soumise est vide ou si c'est une chaîne alternée de 0 et de 1 et en échec (état q_e) dans le cas contraire, c'est-à-dire si la chaîne contient deux 0 ou deux 1 successifs.

7 Conclusion

Dans ce chapitre nous sommes partis de la question centrale du cours ensuite nous avons introduit, ou rappelé, les éléments mathématiques et algorithmiques qui nous seront utiles pour y répondre. Nous allons à présent "rassembler les briques", autrement dit expliquer en quoi chacun de ces éléments intervient dans la réponse à la question.

1. Pour étudier la "résolubilité" des problèmes, il existe plusieurs outils dont il a été demontré qu'ils étaient équivalents. La machine de Turing est l'un de ces outils et c'est celui que nous utiliserons dans ce cours.
2. Une machine de Turing prend en entrée un **mot** et qu'elle nous retourne en réponse un autre mot et/ou une réponse binaire (échec/succès, oui/non). L'ensemble des mots pour lesquels une machine de Turing donne la même réponse (oui ou non, succès ou échec) forment un langage. Un algorithme prend en entrée une donnée et retourne une donnée et/ou une réponse binaire (échec/succès, oui/non). . Pour étudier nos problèmes à l'aide des machines de Turing nous ferons donc sans cesse des aller-retour entre (problème/algorithme) et (langage/machine de Turing).
3. Lorsqu'un problème peut être résolu, nous avons besoin pour connaître le coût de sa résolution du concept de **complexité**.
4. La complexité nous permettra de définir des **classes** de problèmes. Après avoir défini ces classes, nous étudierons des problèmes représentatifs de chacune de ces classes et c'est en théorie des graphes et dans la logique que nous trouverons ces problèmes.