Complexité
NF93

Aziz Moukrim
Aziz.Moukrim@utc.fr
Poste 49 52

Références

- Brassard, P. Bratlez, "Algorithmique, conception et analyse", Masson, 1987
- Robert Sedgewick, “Algorithms”, Addison-Wesley, 1988
- D. Beauquier, J. Berstel, Ph. Chrétienne “Eléments d’algorithmique”, Masson, 1992
- T. Cormen, C. Leiserson, R. Rivest “Introduction à l’algorithmique”, Dunod, 1994
- Brassard, P. Bratlez, "Algorithmique, conception et analyse", Masson, 1987
- Baynat, Chrétienne, Hanen, Kedad-Sidhoum, Munier-Kordon et Picouleau (2003) Exercices et problèmes d'algorithmique (Dunod)
- Livre de Garey et Johnson, « Computers and Intractability, A guide to the theory of NP-completeness », 1979.
- J. Carlier et Ph. Chrétienne « Problèmes d’ordonnancements : algorithmes et complexité », 1988.

Analyse des algorithmes

- Exemples :
  - P1 : Multiplication de deux nombres
  - P2 : Problème des tours de Hanoï
  - P3 : Problème du voyageur de commerce
  - P4 : Problème du meilleur coup à jouer aux échecs
  - P5 : stratégie gagnante Table / pièces pour deux joueurs
- La théorie de la complexité étudie deux aspects:
  - la complexité des algorithmes : évaluer en fonction de la taille des données, le nombre des opérations des algorithmes.
  - la complexité des problèmes et leur classification.

Analyse des algorithmes

- Performances croissantes des ordinateurs en rapidité et en taille-mémoire
- « coût » de l'algorithme
  - Nombre d'opérations élémentaires
  - Quantité de mémoire requise
- Certains problèmes peuvent demander des temps prohibitifs quels que soient l'algorithme « connu » et la machine utilisés

Analyse des algorithmes

- Certains problèmes, par contre, demandent un temps prohibitif avec certains algorithmes de résolution et « raisonnable » avec d'autres.
  - Il faut pouvoir comparer les performances de différents algorithmes disponibles pour résoudre un problème particulier.
    - Comment évaluer de telles performances ?
  - Ces évaluations ne devraient dépendre ni du langage ni de la machine utilisés

Analyse des algorithmes

- Définition : Un algorithme est un ensemble de règles opératoires dont l'application permet de résoudre un problème énoncé au moyen d'un nombre fini d'opérations
- Instance : Une instance d'un problème est définie par la fourniture des diverses valeurs attendues
  - 58*36 est une instance du problème P1
- Taille du problème : il existe toujours une donnée entière positive n caractérisant le volume de données manipulées :
  - Nombre de disques de la tour de Hanoï
  - Nombre d'éléments à trier par un algorithme de tri

Analyse des algorithmes

- Complexité temporelle : on appelle complexité temporelle d'un algorithme une fonction mesurant le temps nécessaire à l'exécution de cet algorithme pour une instance de taille n
- Complexité spatiale : on appelle complexité spatiale d'un algorithme une fonction mesurant la place utilisée en mémoire par celui-ci pour une instance de taille n.

Comportement asymptotique

- Calcul de la complexité d'un algorithme : somme du
  - Nombre d'affectations,
  - Nombre d'opérations arithmétiques,
  - Nombre d'opérations logiques.
- Exemple :
  - Plus grand élément dans un tableau
  - Recherche séquentielle d'un élément dans un tableau

Complexité

- Soit D_n l'ensemble des données de taille n
- Cout(d) la complexité en temps (affectations, opérations) sur la donnée d
- Complexité dans le meilleur des cas :
  - Min(n) = min {cout(d), d ∈ D_n }
- Complexité dans le pire des cas :
  - Max(n) = max {cout(d), d ∈ D_n }

Comportement asymptotique

- Paramètres ne pouvant influencer que d'un facteur multiplicatif :
  - Machine,
  - langage,
  - compilateur,
  - génie du programmeur.

Comportement asymptotique
10^6 opérations /seconde

	10	20	30	40	50	60
n	.00001s	.00002s	.00003s	.00004s	.00005s	.00006s
n^2	.0001s	.0004s	.0009s	.0016s	.0025s	.0036s
n^3	.001s	.008s	.027s	.064s	.125s	.216s
n^5	.1s	3.2s	24.3s	1.7mn	5.2mn	13mn
2^n	.001s	1s	18mn	13j	36a	366s.
3^n	.06s	58mn	6.5a	3855s.	$2*10^8$s.	$1.3*10^{12}$s.
n!	3.6s	771.5s.	10^{17}s.	10^{32}s.	10^{49}s.	10^{66}s.

Comportement asymptotique

f(n)	Avec les ordinateurs d'aujourd'hui	100 fois plus rapide	1000 fois plus rapide
n	N1	100 N1	1000 N1
n^2	N2	10 N2	31.6 N2
n^3	N3	4.64 N3	10N3
2^n	N4	N4 +6.64	N4 + 9.97
3^n	N5	N5 + 4.19	N5 + 6.29

Tableau rapportant la taille des problèmes résolus avec une heure de calcul.

Comportement asymptotique

- Grandes valeurs du paramètre n qui mesure la taille d'une instance d'un problème
- **Complexité dans le pire cas : pour l'instance pour laquelle notre algorithme fonctionne le moins bien**

Comportement asymptotique

- La notation O (« de l'ordre de » ou « grand O ») définit une borne asymptotique supérieure.
- Pour une fonction f(n) donnée, on note O(f(n)) l'ensemble des fonctions tel que :
  - O(f(n)) = {T(n) : il existe des constantes strictement positives c et n_0 telles que $0 \leq T(n) \leq c.f(n)$ pour tout $n \geq n_0$}
- Exemples :
  - $f(n)= n^2$ et $T(n)=13n^2+7n+11$
  - $f(n)= n^2$ et $T(n)=1/2n^2-3n$

Comportement asymptotique

- Règle du maximum :
  Si f, g : N $\rightarrow R^+$ alors O(f(n)+g(n)) = O(max(f(n), g(n)))
- La relation « $\in$ O » est réflexive et transitive
- Si lim $_{n \rightarrow +\infty}$ f(n)/g(n) $\in$ R^*_+ alors f(n) $\in$ O(g(n)) et g(n) $\in$ O(f(n))
- Si lim $_{n \rightarrow +\infty}$ f(n)/g(n) =0 alors f(n) $\in$ O(g(n)) mais g(n) $\notin$ O(f(n))
- Si lim $_{n \rightarrow +\infty}$ f(n)/g(n) = + ∞ alors g(n) $\in$ O(f(n)) mais f(n) $\notin$ O(g(n))

Comportement asymptotique

- Un algorithme de l'ordre de n log(n) est dans O(n^2), dans O(n^3), etc…
- La notation Ω définit une borne asymptotique inférieure.
- Pour une fonction f(n) donnée, on note Ω(f(n)) l'ensemble des fonctions tel que :
  - Ω(f(n)) = {T(n) : il existe des constantes strictement positives c et n_0 telles que $0 \leq cf(n) \leq T(n)$ pour tout $n \geq n_0$}

Comportement asymptotique

- La notation Θ définit l'ordre exact d'une fonction.
- Pour une fonction f(n) donnée, on note Θ(f(n)) l'ensemble des fonctions tel que :
  - Θ(f(n)) = {T(n) : il existe des constantes strictement positives c_1, c_2 et n_0 telles que $0 \leq c_1 f(n) \leq T(n) \leq c_2 f(n)$ pour tout $n \geq n_0$}
- Exemples :
  - $f(n)= n^2$ et $T(n)=1/2n^2 -3n$

Comportement asymptotique

- Analyse du temps d'exécution d'un algorithme
  - Affectation ou opération élémentaire : O(1)
  - Instruction composée :
    - Si $T_1(n) = O(f_1(n))$ et $T_2(n) = O(f_2(n))$ avec $f_2(n) = O(f_1(n))$ alors $T_1(n)+T_2(n)$ est en $O(f_1(n))$.
  - Bloc d'instructions : règle de sommation
  - « Si »
    - Si $O(f_{Si})$ et $O(f_{Sinon})$ sont des bornes supérieures des blocs « Si » et « Sinon » alors $O(\max\{f_{Si}(n), f_{Sinon}(n)\})$ est une borne supérieure du temps d'exécution de l'instruction « Si »
  - « Pour, Tant_que, repeter »
    - Si O(f(n)) est une borne supérieure du corps de la boucle et O(g(n)) est une borne supérieure du nombre d'itérations alors O(f(n)g(n)) est une borne supérieure de la boucle.

Récursivité

- La définition d'un objet X est récursive si elle contient une référence à l'objet X lui même (récursivité directe) ou à un objet Y qui fait référence (directement ou indirectement) à l'objet X (récursivité croisée ou indirecte).
- Exemple : Un ascendant d'une personne est :
  - Soit son père
  - Soit sa mère
  - Soit un ascendant de son père ou de sa mère

Récursivité

- On appelle profondeur de la récursivité à un instant donné de l'exécution de l'algorithme, le nombre d'appels imbriqués.
- Tout algorithme récursif doit contenir une condition qui assure la finitude du nombre d'appels emboîtés (cas trivial).
- Exemple :
  - N!
  - Tour de Hanoi
  - Suite de Fibonacci

Récursivité

- Mécanisme de la récursivité :
  - pile des environnements
- Arbre des appels d'un programme récursif :
  - N! T(n) =
  - Tour de Hanoi T(n) =
  - Suite de Fibonacci T(n) =

Récursivité

- Calculs redondants
  - Méthode de marquage
  - Analyse du graphe des appels
- Algorithmes récursifs admettant une version itérative

Théorie de la complexité complexité des problèmes

- Au cœur de la théorie de la complexité et de son formalisme se trouve les problèmes de décision.
- Un problème de décision est une question mathématiquement définie portant sur des paramètres donnés sous forme manipulable informatiquement, et demandant une réponse par oui ou par non.

Le problème SAT

- Le problème SAT est un problème de décision d'une grande importance en théorie de la complexité.
- Le problème SAT s'énonce de la manière suivante :
  - Données : un ensemble V de variables booléennes et une famille C de clauses sur V. Une clause est un ensemble de variables booléennes sous la forme directe ou complémentée, elle est vraie dès qu'une de ses variables est vraie, *donc la somme*.
  - Question : existe-t-il une façon d'affecter les valeurs « vrai » ou « faux » aux variables afin de rendre vraie toutes les clauses ?
- Problème 3SAT : chaque clause ne contient que trois variables.

Problème du voyageur de commerce (TSP)

- Étant donné *n* points (des « villes ») et les distances séparant chaque point, trouver un chemin de longueur totale minimale qui passe exactement une fois par chaque point (et revienne au point de départ).
- On associe le problème de décision suivant.
  - Donnée : un graphe complet $G = (V, E, \omega)$ avec V un ensemble de sommets, E un ensemble d'arcs et ω une fonction de coût sur les arcs, et un entier B.
  - Question : Existe-t-il un circuit hamiltonien (c'est-à-dire passant une et une seule fois par chaque sommet) dont le coût (la somme du coût des arcs qui le composent) est inférieur ou égal à *B* ?

Problèmes de partition

- Le problème de la partition
  - Donnée : un ensemble $A=\{a_i \mid i \in I\}$ de n nombres entiers.
  - Question : Existe-t-il une partition de A en deux sous-ensembles A1 et A2 de même poids ?

Exemples de problèmes équivalents

- Problème de la couverture de sommets.
  - Donnée : un graphe $G=(V,E)$ avec V un ensemble de sommets, E un ensemble d'arcs et un entier (positif) $B \leq |V|$.
  - Question : Existe-t-il une couverture des sommets de taille inférieure ou égale à B, c'est-à-dire un sous-ensemble $V' \subseteq V$ tel que $|V'| \leq B$, et pour tout arc $(i,j) \in E$, $i \in V'$ ou $j \in V'$?
- Problème de l'ensemble indépendant (independent set).
  - Donnée : un graphe $G=(V,E)$ avec V un ensemble de sommets, E un ensemble d'arcs et un entier (positif) $B \leq |V|$.
  - Question : Existe-t-il un ensemble de sommets $V' \subseteq V$ de taille supérieure ou égale à B tel que pour tout arc $(i,j) \in E$, $i \notin V'$ ou $j \notin V'$?
- Problème de clique maximum.
  - Donnée : un graphe G = (V,E) avec V un ensemble de sommets, E un ensemble d'arcs, et un entier B.
  - Question : Existe-t-il un sous-ensemble $V' \subseteq V$ tel que le sous-graphe correspondant est complet et de cardinal supérieur ou égal à B ?

Problème de la couverture exacte

- Problème de la couverture exacte d'un ensemble par des sous-ensembles de cardinal 3, (X3C).
  - Donnée : un ensemble fini X avec $|X|=3q$ et une famille C de sous-ensembles de cardinal 3 de X.
  - Question : Existe-t-il une couverture exacte de X, c'est-à-dire une sous-famille $C' \subseteq C$ tel que $\forall x \in X$, il y a un seul $c \in C'$ avec $x \in c$?

Problèmes d'ordonnancements

- Le problème à une machine
  - Donnée : un ensemble I de n tâches indépendantes et non morcelables; pour chaque tâche $i \in I$ sa durée p_i, sa date de disponibilité r_i et sa date échue d_i.
  - Question : Existe-t-il un ordonnancement ϕ de ces n tâches sur une machine qui respecte les dates de disponibilité et les dates échues ?
- Le problème à deux processeurs
  - Donnée : un ensemble I de n tâches indépendantes et non morcelables, les durées p_i et un nombre B.
  - Question : Existe-t-il un ordonnancement des ces n tâches sur deux processeurs de durée inférieure ou égale à B ?
- Le problème du Nombre de Tâches en Retard sur une Machine (NTRM).
  - Donnée : Un ensemble I={1,2,.. n} de tâches de durées 1 et de dates échues d_1, d_2, .., d_n, un ordre partiel < sur I, et un nombre B.
  - Question : Existe-t-il un ordre de passage ϕ des tâches sur une machine respectant l'ordre partiel et tel que le nombre de tâches en retard soit $\leq$ B.

Théorie de la complexité
complexité des problèmes : notions de base

- Les problèmes de décision; pourquoi ce choix ?
- Pour introduire un formalisme et faciliter la comparaison des problèmes selon leur difficulté.
- *La théorie de la complexité est basée sur la notion de la machine de Turing…*
  - Une machine de Turing est un modèle abstrait du fonctionnement des appareils mécaniques de calcul, tel un ordinateur et sa mémoire destinée à donner une définition précise au concept d'algorithme.
  - Tout problème de calcul basé sur une procédure algorithmique peut être résolu par une machine de Turing.
  - Toute instruction du langage machine d'un ordinateur monoprocesseur peut être simulé sur une machine de Turing, par un programme de complexité O(1). Il en est de même pour toute opération élémentaire (i.e. instruction d'un langage informatique).

Machine de Turing

- Description de la machine de Turing :
  1. Un « ruban » divisé en cases consécutives. Chaque case contient un symbole parmi un alphabet fini.
  2. Une « tête de lecture/écriture » qui peut lire et écrire les symboles sur le ruban, et se déplacer vers la gauche ou vers la droite du ruban.
  3. Un « registre d'état » qui mémorise l'état courant de la machine de Turing.
  4. Une « table d'actions » qui indique à la machine quel symbole écrire, comment déplacer la tête de lecture et quel est le nouvel état, en fonction du symbole lu sur le ruban et de l'état courant de la machine.
  - *Un codage permet d'associer à une donnée d'un problème un mot x défini sur un alphabet.*
  - *A un problème de décision on associe des mots x (i.e., des données) pour lequel la réponse est oui; c'est le langage L.*

Machine de Turing non déterministe

Une machine de Turing non déterministe est constituée d'une machine de Turing déterministe et d'un *module divinatoire* (oracle) ayant un tête d'écriture, et d'un programme déterministe.

Fonctionnement :

Lors de la phase d'initialisation l'oracle écrit un mot $x \in X^*$.

Ensuite la machine de Turing marche à l'aide de son programme déterministe.

La classe NP est par définition, l'ensemble des langages reconnus polynomialement par une machine de Turing non déterministe.

Complexité des problèmes

- Alternativement :
- On distingue des classes de complexité :
  - Classe P : Un problème de décision est dans P si il peut être résolu par un algorithme *déterministe* en un temps *polynomial* par rapport à la taille de l'instance. On qualifie alors le problème de « polynomial ».
  - Classe NP : C'est la classe des problèmes de décision pour lesquels la réponse *oui* peut être décidée par un algorithme *non-déterministe* en un temps polynomial par rapport à la taille de l'instance.
    - **La vérification est polynomiale.**

Théorie de la complexité les problèmes NP-complet

Article de Stephen Cook, «The complexity of Theorem Proving Procedures», 1971.

- définit la réduction polynomiale
- définit les problèmes de décision et la classe NP
- démontre qu'il y a un problème (SAT) qui est au moins aussi difficile que tous les autres → NP-complet.

Théorie de la complexité complexité des problèmes

- La réduction polynomiale permet de comparer selon leur difficulté les problèmes NP entre eux.
- *Définition : On dit que P_1 se réduit polynomialement à P_2 ($P_1 \propto P_2$) si P_1 est polynomial ou s'il existe un algorithme polynomial A construisant à partir d'une donnée d_1 de P_1 une donnée d_2 de P_2 tel que d_1 à la réponse oui si et seulement si $d_2=A(d_1)$ a la réponse oui.*
- Problème NP-Complet : un problème est NP-Complet s'il est dans NP, et si n'importe quel problème NP peut se réduire polynomialement à ce problème.
- La réduction polynomiale est une relation de pré-ordre sur NP.
- Théorème de Cook : SAT est NP-Complet.

Complexité des problèmes, méthodes de démonstration (I)

- Comment démontrer qu'un problème Π est NP-complet
  - 1) démontrer que $\Pi \in$ NP
  - 2) démontrer qu'il existe P'$\in$NP-complet tel que P'$\propto$ Π.
- Les problèmes ci-dessous sont NP-complets :
  - le problème du voyageur de commerce,
  - les problèmes de partition,
  - les problèmes de couverture
  - le problème de la clique

Théorie de la complexité complexité des problèmes

- Démontrer les trois propositions suivantes :
  - Proposition 1. Le problème à deux processeurs est NP-complet.
    - Il suffit de démontrer que ***partition*** $\propto$ ***deux processeurs***.
  - Proposition 2. Le problème à une machine est NP-complet.
    - Il suffit de démontrer que ***partition*** $\propto$ ***le problème à une machine***.
  - Proposition 3. Le problème NTRM est NP-complet.
    - Il suffit de démontrer que ***clique*** $\propto$ ***NTRM.***

Problème NP-complet : conjecture

- Conjecture fondamentale de la théorie de la complexité : P≠NP.

Le prix de 1000000 USD sera attribué à celui qui démontre que soit P=NP soit P≠NP.