NOM : PECATTE **Prénom :** Timothée **Jury :**

Algèbre ← Entourez l'épreuve → **Analyse**

Sujet choisi : [926] Analyse d'algorithme : Complexité. Exemples.

Autre sujet : Réfs :
- Kozen, The design and analysis of algorithms
- Herbert, Algorithms and complexity
- Niedermeier, Invitation to Fixed-Parameter Algorithms

⊕ Froidevaux, Cormen

I. De la calculabilité à la complexité :

I.1. Introduction :

Rmq 1 : D'un point de vue calculabilité, deux modèles de calcul qui calculent les mêmes fonctions sont équivalents.

Ex 2 : Les machines de Turing à 1 ruban / 2 rubans de travail sont équivalentes et donc calculent toutes les deux l'addition de deux entiers.

Rmq 3 : On aimerait dire que l'algo à 2 rubans est plus "rapide/efficace" que celui à 1 ruban.

But Définir une mesure de complexité des algorithmes.

Rmq 4 : Elle dépend fortement du modèle de calcul choisi.

Rmq 5 : Il existe plusieurs "types" de complexité : temps, espace, communications, ... Dans la suite, on considèrera surtout le temps.

I.2. La complexité en temps : une approche mathématique

Déf 6 : (Opération élémentaire) Une opération élémentaire est une opération dont le nombre total d'exécution de celle-ci est proportionel au temps de calcul de l'algorithme.

Ex 7 : Comparaison, lecture en mémoire, addition, multiplication, ...

Déf 8 : Etant donné un algorithme, et une entrée x de cet algorithme, on défini $T(x)$ comme le nombre d'opérations élémentaires effectuées par l'algorithme lancé sur l'entrée x.

Déf 9 : (Pire cas). Etant donné un algorithme, on définit aussi sa complexité pire cas $T(n) = \max_{|x|=n} T(x)$, où $|\cdot|$ est une mesure de taille.

Ex 10 : Additionner n chiffres peut être fait en n additions dans le pire cas de manière naïve (donc $T(n)=n$), ou en $\lceil \log n \rceil$ additions avec un algorithme diviser pour régner (donc $T'(n) = \lceil \log n \rceil$)

Déf 11 : Ordres de grandeurs, o, O, Θ

Rmq 12 : Comme c'est la complexité asymptotique qui nous intéresse, on se contentera de donner les ordres de grandeur de $T(n)$

Ex 13 : $T(n) = \Theta(n)$ pour l'algo naïf
$T(n) = \Theta(\log n)$ pour l'algo diviser pour régner (ex 10).

I.3. La complexité en temps : une approche informatique

Prop 14 : Si "$A \perp B$", alors $T_{A \text{ puis } B}(x) = T_A(x) + T_B(x)$

Cor 15 : Complexité d'une boucle : $\sum_{i \in I} P(i)$

Cor 16 : Complexité d'un appel de fonction f, avec x en arg : $T_f(x)$

Rmq 17 : On peut même faire des appels récursifs, mais il faut d'abord montrer que l'algorithme s'arrête avant de considérer sa complexité.

Ex 18 : Fact écrit de manière récursive s'arrête et sa complexité vérifie la relation $T(n) = T(n-1) + 1$, soit $T(n) = n$.

Rmq 19 : On n'a pas de manière simple d'étudier la complexité des boucles TANT QUE, à part la mise en place d'invariant

Rmq 20 : Une fois les relations de récurrence mis en place, on peut alors s'aider d'outils mathématiques pour les résoudre, comme le "Master théorème". Idem pour simplifier des sommes.

II. Premiers exemples simples d'analyses de complexité :

II.1. Programmes itératifs :

Rmq 21 : Plutôt que de calculer précisément $P(i)$ pour une somme, si on a un majorant $Q(i)$ de $P(i)$, on a $\sum P(i) = O(\sum Q(i))$, et les $Q(i)$ peuvent être plus simples à calculer.

<u>Ex 22</u>: Dans le cas d'une double boucle imbriquée :

```
POUR i = 0 À n FAIRE
| POUR j=0 À i FAIRE
| | *une opération élémentaire*
| FIN
FIN
```

On peut majorer $P(i)$ par n pour tout i, ce qui nous donne $T(n) = \Theta(n^2)$

<u>Rmq 23</u>: Une telle sur-analyse peut parfois suffire à capturer l'ordre de grandeur

<u>Ex 24</u>: Une analyse plus fine de l'exemple 22 donne $P(i) = i$, soit $T(n) = \sum_{i=0}^{n} i$

D'où $T(n) = \frac{n(n-1)}{2} = \Theta(n^2)$, ce qui était déjà trouvé au-dessus.

<u>Ex 25</u>: (Algorithme glouton) Les algorithmes gloutons se prêtent bien à ce type d'analyse car il s'agit d'algorithme itératif où les opérations à une étape ne dépendent que de l'état actuel, pas de la suite d'étapes précédentes (comme les chaînes de Markov)

<u>Ex 26</u>: Algorithme de Kruskal : - pré-traitement : $O(m \log m)$
- chaque étape : $O(1)$ $(O(\alpha(n))$ en vrai$)$
↳ $\underline{x}$ m étapes

La complexité de la boucle est donc en $O(m)$, soit une complexité finale en $O(m \log m)$

<u>II. 2. Programmes récursifs :</u>

<u>Th27</u>: Si on a $x_{n+1} \leq b_0 x_n + \dots + b_p x_{n-p} + G(n)$, avec $b_i \geq 0$ et $\sum b_i > 1$.
Si on note c la racine réelle positive de $x^{p+1} = b_0 x^p + \dots + b_p$, et que $G(n) = o(c^n)$, alors
Alors, pour tout $\varepsilon > 0$, on a $x_n = O((c+\varepsilon)^n)$

<u>Rmq 28</u>: Si $G(n)$ est polynomial, on aura bien $G(n) = o(c^n)$, ce qui nous permet de résoudre des relations de récurrence d'algo exponentiel aisément.

<u>DEV 1</u> Algorithme pour résoudre INDEPENDANT en $O(1{,}39^n)$ ← DEV !

<u>Th29</u>: Master theorème

<u>App 30</u>: Analyse des algorithmes suivant le paradigme diviser pour régner

<u>Ex 31</u>: Tri-Fusion tri en $O(n \log n)$

<u>Ex 32</u>: Médiane peut être trouvée en $O(n)$

<u>III. Analyses plus fines de complexité :</u>

<u>III. 1. Complexité moyenne :</u>

<u>Rem 33</u>: Dans certains cas concrets, le pire cas est potentiellement peu fréquent et donc la complexité de l'algorithme n'est pas très bien capturée par la complexité pire cas.

<u>Déf 34</u>: (Complexité moyenne) $\tilde{T}(n) = \sum_{|x|=n} q(x) T(x)$, où q est une distribution de probas

<u>Rmq 35</u>: Cette définition de $\tilde{T}$ dépend de la distribution sous-jacente q

<u>Ex 36</u>: TRI_RAPIDE a une complexité pire cas en $\Theta(n^2)$ et une complexité moyenne sur la distribution uniforme en $\Theta(n \log n)$

<u>Ex 37</u>: La complexité moyenne de la recherche séquentielle d'un élément dépend de sa chance d'apparaître dans la liste : • $q = 1 \rightsquigarrow \tilde{T}(n) = \frac{n+1}{2}$
• $q = 1/2 \rightsquigarrow \tilde{T}(n) = \frac{(3n+1)}{4}$

<u>Ex 37bis</u>: Table de Hachage.

<u>III. 2. Algorithmes itératifs et complexité amortie</u>

<u>Rmq 38</u>: Dans certains cas, les différents passages dans une boucle (resp. la suite d'instructions identiques) ne sont pas indépendantes entre eux. On peut donc analyser plus finement la boucle (resp. la suite d'instructions) dans son ensemble plutôt que de majorer chaque P(i)

Déf 39 : Si une suite de m instructions (resp une boucle de taille m) a comme complexité $T(n)$, on définit la complexité amortie d'une instruction (resp d'un passage dans la boucle) par : $\hat{T}(n) = \frac{T(n)}{m}$

Ex 40 : Tableaux dynamiques : ajout et suppression en $O(1)$ en amorti

Rem 41 : La complexité amortie ne permet pas de différencier deux types d'opérations avec cette méthode de l'agrégat.

Déf 42 : Méthode comptable : attribue à chaque opération un coût amorti

Ex 43 : • tableaux dynamique : payer 3 pour l'ajout, 0 pour la suppression redonne le $O(1)$

• compteur binaire : incrémenter en $O(1)$

Déf 44 : Méthode du potentiel : attribue à un état de la structure un nombre

Rmq 45 : La méthode comptable et la méthode du potentiel sont équivalentes.

DEV2 → $\boxed{\text{DEV 2}}$ Arbres splay permettent l'insertion, la suppression, l'union, l'appartenance et la séparation d'un dictionnaire en temps amortie $O(\log n)$.

IV. Vers la théorie de la complexité :

IV. 1. Algorithmes FPT :

Rmq 46 : Pour certains problèmes où l'on cherche "le plus grand ensemble tel que…", on peut se demander comment la taille de cette ensemble influe sur la complexité

Ex 47 : Si on teste l'existence d'un vertex-cover de taille k dans un graphe de taille n, on peut répondre en temps $O(2^k . n)$

Déf 48 : Un algorithme à paramètre est FPT si sa complexité se met sous la forme $O(f(k). n^c)$, où f est une fonction croissante quelconque

Déf 49 : (Kernelisation) (I, k) se réduit à (I', k') si $k' \leq k$, $|I'| \leq g(k)$ et $(I, k) \in L$ si $(I', k') \in L$

Ex 50 : Vertex-Cover se réduit à un problème de taille linéaire en k

IV. 2. Optimalité :

Rmq 51 : Pour le tri fusion qui trie en $O(n \log n)$, on aimerait dire que on ne peut pas faire mieux sous certaines conditions.

Th 52 : Tout algorithme qui trie une séquence en utilisant des comparaisons a une complexité en temps en $\Omega(n \log n)$.

Cor 53 : Tri-Fusion est optimal

Rmq 54 : Pour prouver une optimalité, il faut donc prouver une borne sup et une borne inf, et c'est souvent les bornes inf qui sont difficiles à obtenir.

Rmq 55 : L'optimalité dépend fortement du modèle de calcul choisi

Ex 56 : Dans le modèle de calcul quantique, on peut trouver un élément dans une séquence non triée de taille N en temps $O(\sqrt{N})$ avec l'algorithme de Grover.

Question 57 : Est-ce que deux algorithmes qui ont la même complexité sont équivalents ?

Déf 58 : Réduction

Déf 59 : Classes de complexité

Ex 60 : P vs NP