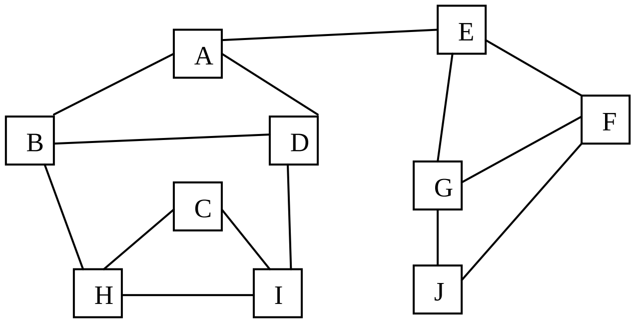
Université des Sciences et de la Technologie Houari Boumediène
Faculté d'Electronique et d'Informatique, Département d'Informatique
Master 1 IL 2013/2014, Module ''Algorithmique Avancée et Complexité''
Date : 13/01/2014 (par Mr Isli)

Corrigé de l'examen

Exercice 1 <3 points=1,5+1,5>:

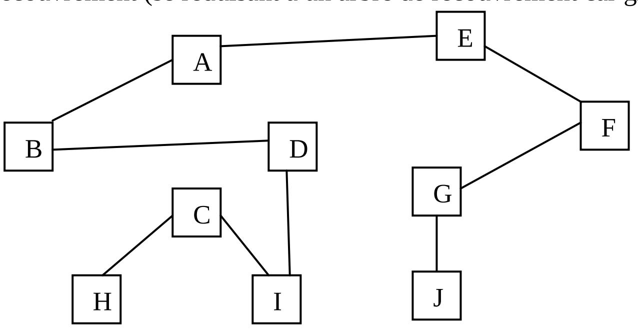
On considère le graphe simple (non orienté) suivant :



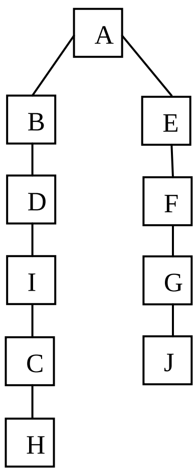
1. Donnez une forêt de recouvrement issue d'un parcours en profondeur d'abord du graphe.
2. Donnez l'ordre dans lequel le parcours considéré a visité les sommets du graphe.

Solution :

1. Forêt de recouvrement (se réduisant à un arbre de recouvrement car graphe connexe) :



2. Ordre de visite des sommets : ABDICHEFGJ
3. **Conclusion :** arbre ordonné



Exercice 2 <5points=1+1+1+2>:

1. Prouvez que n^2+log(n^2)=O(n^2).
2. Donnez la complexité de chacune des fonctions suivantes :
    1. $f_1(n)=\frac{2n^3+5n^2+3}{n+3}$
    2. $f_2(n)=\frac{n^2 log(n)+n^2+\log(n)^2}{n+1}$
3. Si f(n) est une fonction en O(n), a-t-on $3^{f(n)} = O(3^n)$?

Solution :

1. Pour montrer que n^2+log(n^2)=O(n^2), il suffit de trouver c>0 et $n_0 \geq 0$ tels que $\forall n \geq n_0$ n^2+log(n^2)≤c*n^2. On divise les deux membres par n^2 : $1+\frac{\log(n^2)}{n^2} \leq c$

   Conclusion : la limite quand n tend vers l'infini de $\frac{\log(n^2)}{n^2}$ est 0, donc si prend c=2 on peut trouver $n_0 \geq 0$ tels que $\forall n \geq n_0$ n^2+log(n^2)≤c*n^2. D'où n^2+log(n^2)=O(n^2).
2. **Complexité des fonctions f_1 et f_2 :**
    a) On montre que $f_1(n)=0(n^2)$: trouver c>0 et $n_0 \geq 0$ tels que $\forall n \geq n_0$ $\frac{2n^3+5n^2+3}{n+3} \leq c*n^2$. Division des deux membres par n^2 : $\frac{2n^3+5n^2+3}{n^3+3n^2} \leq c$, qui est équivalent à $2+\frac{3-n^2}{n^3+3n^2} \leq c$

    Conclusion : la limite quand n tend vers l'infini de $2+\frac{3-n^2}{n^3+3n^2}$ étant 2-, si prend c=2 on peut trouver $n_0 \geq 0$ tels que $\forall n \geq n_0$ $\frac{2n^3+5n^2+3}{n+3} \leq c*n^2$. D'où $\frac{2n^3+5n^2+3}{n+3}$=O(n^2).

    b) On montre que $f_2(n)=0(n\log(n))$: trouver c>0 et $n_0 \geq 0$ tels que $\forall n \geq n_0$ $\frac{n^2 log(n)+n^2+\log(n^2)}{n+1} \leq c*n\log(n)$. Division des deux membres par nlog(n) : $\frac{n}{n+1} + \frac{n}{(n+1)\log(n)} + \frac{\log(n^2)}{n(n+1)\log(n)} \leq c$.

    Conclusion : la limite quand n tend vers l'infini de $\frac{n}{n+1} + \frac{n}{(n+1)\log(n)} + \frac{\log(n^2)}{n(n+1)\log(n)}$ étant 1, si prend c=2 on peut trouver $n_0 \geq 0$ tels que $\forall n \geq n_0$ $\frac{n^2 log(n)+n^2+\log(n)^2}{n+1} \leq c*n\log(n)$. D'où $\frac{n^2 log(n)+n^2+\log(n)^2}{n+1}$=O(nlog(n)).
3. Si f(n) est une fonction en O(n), on n'a pas toujours $3^{f(n)} = O(3^n)$. En voici un contre-exemple : f(n)=2n. On a bien f(n)=O(n) mais $3^{f(n)}=9^n$ n'est pas égal à $O(3^n)$. Montrons-le par l'absurde. Supposons que $3^{f(n)}= O(3^n)$. Il existerait c_0 et $n_0 \geq 0$ tels que $\forall n \geq n_0$ $9^n \leq c * 3^n$. On obtient les deux membres par 3^n pour obtenir $3^n \leq c$. La constante c n'existe pas (la quantité 3^n ne peut pas être bornée, sa limite quand n tend vers l'infini étant l'infini).

Exercice 3 <2 points>:

Montrez que le nombre de branches vides d'un arbre binaire à n nœuds est n+1. Le nombre de branches vides est le nombre de fils droits et de fils gauches vides.

Solution :

Preuve par récurrence. Soit bv(n) le nombre de branches vides d'un arbre binaire à n nœuds. On vérifie facilement que la propriété bv(n)=n+1 est vérifiée pour n∈{1,2,3}. On suppose qu'elle reste vérifiée jusqu'à un certain k≥3 : ∀n≤k, bv(n)=n+1. Montrons que nous avons alors bv(k+1)=(k+1)+1=k+2. Soit t un arbre binaire à k+1 nœuds, et t' un arbre binaire obtenu de t par suppression d'une feuille : clairement bv(t)=bv(t')+1 (en remettant la feuille supprimée de t pour obtenir t', on réobtient l'arbre t : l'ajout d'une feuille supprime une branche vide mais en crée deux autres). Or, d'après l'hypothèse de récurrence, bv(t')=k+1. D'où bv(t)=k+1+1=k+2.

Conclusion : la propriété restant vérifiée pour n=k+1, elle est vérifiée pour tout n.

Exercice 4 <10 points=1+1+2+2+1+2+1>:

On considère le problème de décision P suivant :

- **Description :** un ensemble fini A, une fonction s associant à chaque élément x de A une taille s(x)ϵIN, et un entier positif B.
- **Question :** Existe-t-il un sous-ensemble A' de A de telle sorte que le produit des tailles des éléments de A' soit égal à B.

Le but de l'exercice est de montrer que le problème P appartient à la classe NP, et d'en donner un algorithme de résolution. Pour ce faire, il vous est demandé de procéder comme suit :

1. Donnez une structure de données permettant de représenter une instance du problème P. Expliquez.
2. Quelle information doit contenir un certificat d'une instance du problème P ? Donnez une structure de données permettant la représentation d'un tel certificat. Expliquez.
3. Donnez un algorithme de validation pour le problème P sous forme d'une fonction booléenne dont il est important que vous expliquiez les paramètres. L'algorithme, que vous appellerez **validation_P**, doit être polynômial (voir questions 4. et 5.).
4. Calculez le nombre d'opérations élémentaires de l'algorithme **validation_P** en fonction d'une taille n à préciser. Appelez ce nombre T(n).
5. Montrez que $T(n)=O(n^k)$, pour une certaine constante k à préciser.
6. Donnez un algorithme **résol_P** de résolution du problème P sous forme d'une fonction booléenne vérifiant si une instance de P est validée par **validation_P**.
7. Quelle est la complexité de l'algorithme **résol_P** ? Expliquez.

Solution :

1. Une instance I du problème P est donnée par un ensemble A de taille n, une fonction s associant à chaque élément x de A une taille s(x), et un entier B. Les éléments de A sont supposés ordonnés. Une telle instance peut être représentée par la structure de données I=(S,n,B), T étant un tableau d'entiers, n la taille de T, et B un entier (S[i] donne la taille $s(a_i)$ du $i^{ème}$ élément a_i de A).
2. Un certificat c d'une instance I donnée par un ensemble A, une fonction s et un entier B, doit dire, pour chaque élément x de A, si le sous-ensemble A' contient x. Un tel certificat peut donc être représenté par tableau de taille n de booléens : A' contient le $i^{ème}$ élément a_i de A si et seulement si c[i]=1.
3. nous donnons ci-après, sous forme d'une fonction booléenne, un algorithme de validation validation_P pour le problème P : l'algorithme aura comme arguments une une instance I=(S,n,B) de P et un certificat c de I :

Booléen validation_P(S,n,B,c)
début
- i. Produit=1 ;
- ii. pour i=1 à n faire
  - a. Produit=Produit*c[i]
- iii. Fait
- iv. si Produit=B alors retourner VRAI sinon retourner FAUX finsi

fin

4. Le nombre T(n) d'opérations élémentaires du pire cas de l'algorithme de validation est clairement un polynôme de degré 1 en n, comme l'indique le tableau ci-dessous :

Instruction	Nombre d'opérations du pire cas
i.	1
ii.	3(n+1)
ii.a.	2n
iv.	2

T(n)=5n+6

5. T(n) polynôme de degré 1 donc T(n)=O(n). En voici la preuve : trouver c>0 et $n_0 \geq 0$ tels que $\forall n \geq n_0$ $T(n) \leq c*n$.
$5n+6 \leq c*n$
On divise les deux membres par n : $5+\frac{6}{n} \leq c$ (prendre c=11 et $n_0=1$)
6. L'algorithme résol_P aura comme argument une instance I=(S,n,B) de P et est sous forme d'une fonction booléenne retournant VRAI si et seulement si l'algorithme de validation valide I, c'est-à-dire s'il existe un certificat c de I tel que validation_P(S,n,B,c) retourne VRAI. L'algorithme parcourt tous les certificats possibles de I jusqu'à éventuellement en rencontrer un validant I, ou les explorer tous sans en rencontrer un validant I.

Booléen résol_P(S,n,B)
début
- i. pour i=1 à 2^n-1 faire
  - a. c=écriture_binaire(i)
  - b. si validation(S,n,B,c) alors retourner VRAI finsi
- ii. fait
- iii. retourner FAUX

fin

7. **Complexité de résol_P :**

Instruction	Nombre d'opérations du pire cas
i.	$3*2^n$
ii.	$2*(2^n-1)$
iii.	$(5n+6)*(2^n-1)$
iv.	1

$f(n)=(3+2+5n+6)*2^n-2-5n-6+1=(5n+11)*2^n-5n-7$

Montrons que $f(n)=O(n*2^n)$: trouver $c>0$ et $n_0\geq 0$ tels que $\forall n\geq n_0$ $(5n+11)*2^n-5n-7\leq c*n*2^n$

On divise les deux membres par $n*2^n$:

$5+\frac{11}{n}-\frac{5}{2^n}-\frac{7}{n2^n}\leq c$

La quantité $\frac{11}{n}-\frac{5}{2^n}-\frac{7}{n2^n}$ tend vers 0 quand n tend vers l'infini : donc si prend c=6, on peut trouver $n_0\geq 0$ tels que $\forall n\geq n_0$ $(5n+11)*2^n-5n-7\leq c*n*2^n$

Conclusion : la complexité de résol_P est $O(n*2^n)$.