Université de Lille

FST- Département Informatique

ACT
Durée 3h
Tous documents autorisés

2019–2020
Master1 Informatique
11 janvier 2019

L'équipe pédagogique ACT vous souhaite une excellente année 2019.

* * * * * * * * * * *

Les algorithmes seront écrits en pseudo-langage ou dans un langage de votre choix. La clarté de vos réponses et de votre code sera prise en compte. Le barème est un barème "minimal".

Quelques éléments de correction

Exercice 1 : Un tour d'horizon en quelques questions - *9 pts* -

Q 1. Soient Pire(n) et Meil(n), la complexité dans le pire -respectivement dans le meilleur, des cas d'un algorithme A sur une donnée de taille n. Parmi les affirmations suivantes, quelle affirmation est **toujours** vraie ? Justifier.

1. Pire(n) est en $O(Meil(n))$
2. Pire(n) est en $\Theta(Meil(n))$
3. Meil(n) est en $\Theta(Pire(n))$
4. Meil(n) est en $O(Pire(n))$

Meil(n) est en $O(Pire(n))$

Q 2. Soit $T(n) = 3T(n/3) + n$, $T(0) = T(1) = 1$. Parmi les affirmations suivantes, quelles affirmations sont vraies ? Justifier.

A) $T(n) = O(n^2)$ B) $T(n) = \Theta(n \log n)$ C) $T(n) = O(n \log n)$ D) $T(n) = \Theta(n^2)$

$T(n) = O(n^2)$, $T(n) = \Theta(n \log n)$, $T(n) = O(n \log n)$

Q 3. Soit X, Y, Z trois problèmes de décision. Y est connu NP-complet. X se réduit polynomialement en Y, Y se réduit polynomialement en Z. Parmi les affirmations suivantes, quelles affirmations sont vraies ? Justifier.

i) X est NP ii) Z est NP iii) X est NP-dur iv) Z est NP-dur v) Z est NP-complet

X est NP, Z est NP-dur

Q 4. Supposons que P soit différent de NP. Parmi les affirmations suivantes, quelles affirmations sont vraies ? Justifier.

1. Toute propriété NP est NP-dure.
2. Toute propriété NP est NP-complète.
3. Toute propriété NP-complète est NP.
4. Aucune propriété NP-complète n'est P.

Toute propriété NP-complète est NP, Aucune propriété NP-complète n'est P.

Q 5. Soient deux tableaux d'entiers ; l'un contient n éléments, l'autre $n+1$ éléments : ceux du premier tableau plus un élément "excédentaire". Chaque tableau contient des éléments tous distincts, et est trié dans l'ordre croissant.

Proposer un algorithme en $O(\log n)$ qui sort l'élément excédentaire. Justifier sa correction et sa complexité.

Exemples :
pour T1 : 2, 5, 9, 10, 12 et T2 : 2, 5, 9, 10, 11, 12, la sortie attendue est 11.
pour T1 : 2, 5, 9, 10, 12 et T2 : 1, 2, 5, 9, 10, 12, la sortie attendue est 1.
pour T1 : 2, 5, 9, 10, 12 et T2 : 2, 5, 9, 10, 12, 15 la sortie attendue est 15.

```
g=0;d=n-1;
while (g<=d){
  //invariant {T2[g..d+1] contient T1[g..d] plus l'élt recherché.
  // chaque tranche est triée en ordre croissant et contient des éléments tous distincts
```

```
  //g <=d+1; }
  m=(g+d)/2;
  if (T1[m]==T2[m]) g=m+1; else d=m-1;}
 return T2[g];
```

Q 6. Brutus cherche à énumérer toutes les suites croissantes de k entiers parmi 1..n.

Par exemple, pour k=3 et n=5 :

1,2,3 1,2,4 1,2,5 1,3,4 1,3,5 1,4,5 2,3,4 2,4,5 3,4,5

Q 6.1. Combien y a-t-il de telles suites en fonction de n et k ?

$\binom{n}{k}$

Q 6.2. Proposer un algorithme qui, pour n et k en entrée, sort toutes les suites croissantes de k entiers parmi 1..n.

Proche de l'énumération vue en TP, mais plus simple. par exemple récursivement ;

```
void sortrec(int j){
    if (j==k) sortir();
    else {
        int v=(j==0)?1:cert[j-1]+1;
        while (v<=n-k+j+1) {cert[j]=v;sortrec(j+1);v++;}}}

sortrec(0);
```

ou en utilisant "suivant" :

```
    void next() {
    int i=k-1;
    while ((i >= 0) && (cert[i]+k-1-i>=n)) i--;
    cert[i]++;
    for (j=i+1;j<k;j++) cert[j]=cert[j-1]+1; }
```

et isLast

```
 boolean isLast() {
        for (int i = k-1; i >=0; i--) {
            if (cert[i] +k-1-i < n) return false;}
        return true; }
```

Exercice 2 : Au détour des chemins - *7 pts* -

Rappel : un chemin dans un graphe orienté est une suite finie d'arcs consécutifs, le nombre d'arcs du chemin est sa longueur. Un chemin est **élémentaire** si il ne passe pas deux fois par le même sommet.

On s'intéresse à différentes variantes de problèmes de décision dans un graphe orienté.

Pour tous ces problèmes, la donnée est :

— $G = (S, A)$, un graphe orienté,

— dep, fin deux sommets de G (c.à.d. deux éléments de S), non nécessairement distincts.

— l un entier positif ou nul, tel que $l \leq |S|$

Les quatre problèmes considérés sont :

A : existe-t-un chemin de dep à fin de longueur au plus l ?

B : existe-t-un chemin élémentaire de dep à fin de longueur au plus l ?

C : existe-t-un chemin élémentaire de dep à fin de longueur l ?

D : existe-t-un chemin de dep à fin de longueur l ?

Q 1. Pour chacun des **trois** problèmes A, B, C dire si il est P ou si il est NP-dur (on suppose ici que $P \neq NP$).

Aide : Vous pouvez faire référence à des algorithmes connus en les citant. Vous pouvez utiliser le fait qu'Hamilton Path (vu en TP) est NP-dur :

Hamilton Path

Donnée : $G = (S, A)$, un graphe orienté

Sortie : **Oui**, si il existe un chemin hamiltonien dans G, i.e. une **permutation** des sommets $ham : [1..|S|] \to S$ telle que $(ham(i), ham(i+1)) \in A$, pour tout i, $1 \leq i \leq |S| - 1$.

Non, sinon.

A est P - par exemple Dijkstra ou Bellman-Ford .

B est P aussi puisqu'il existe un chemin élémentaire de dep à fin de longueur au plus l Ssi il existe un chemin de dep à fin de longueur au plus l.

C est NP-dur : on peut utiliser Hamilton Path. Par exemple on réduit Hamilton Path donné par G=(S,A) dans C- donné par G4=(SD',A'), l défini par :

S'=S + un noeud dep + un noeud fin

A'= A + des arcs de dep vers tout noeud de S et de tout noeud de fin

l= |S|+1

Alors Hamilton Path (S,A) a une solution si et seulement C (S',A',l) a une solution.

Q 2. Quels sont les problèmes *NP* parmi A, B, C, D ?

Tous. Il suffit de donner un chemin et de vérifier. Remarque, comme l est inférieur à |S| le certificat et la verif sont bien polynomiaux.

Q 3. Proposer un algorithme en $O(|S|^2 * l)$ pour le problème D. Est-il polynomial ?

Aide : vous pourrez utiliser la programmation dynamique et, par exemple, considérer les sous-problèmes de la forme $Ch(s, k)$: existe-t-il un chemin de s à fin de longueur k ? Vous pourrez choisir la représentation du graphe qui vous convient. On supposera qu'on peut tester en temps constant, étant donné i et j, si il existe un arc du sommet i vers un sommet j.

Ch(s,0)= True Ssi s=fin.

Ch(s,i+1)= $\vee_{(s,x)\in A} Ch(x, i)$.

Donc on définit une table de dimension |S|*(l+1). et le calcul de chaque valeur se fait en O(S). On retourne $Ch(deb, l)$

Q 4. Compléter l'algorithme de la question précédente pour retourner également, si il en existe un, un chemin de dep à fin de longueur l.

remontée classique à faire uniquement si Ch(deb,l) =True

```
x=deb; sortir x;
for j in l..1{
  for s in S {if ((x,s) in A && Ch(s,j-1)) cour=s;}
  x=cour; sortir x;}
  sortir fin;
```

Q 5. Pourquoi cette approche proposée pour le problème D ne peut-elle s'appliquer pour le problème C ?

difficile à étendre au cas des chemins élémentaires car le composé d'un chemin élémentaire et d'un arc n'est pas nécessairement élémentaire.

Exercice 3 : Une valise bien remplie - *4 pts* -

Le problème de la valise est défini par :

Donnée :

cap, un entier positif– la capacité maximale de la valise

n un nombre d'objets,

$p_1, \ldots, p_n$ les poids des objets –on supposera que $p_i \leq cap$, pour tout i

$v_1, \ldots, v_n$ les valeurs respectives des objets

Sortie : un chargement de la valeur qui respecte la capacité maximale, et qui maximise la valeur, i.e. $J \subset [1..n]$ tel que $\sum_{i\in J} p_i$ soit inférieur ou égal à cap et $\sum_{i\in J} v_i$ soit maximal.

Par exemple pour $cap = 4$, $n = 3$, $p_1 = 3, p_2 = 2, p_3 = 2, v_1 = 5, v_2 = 3, v_3 = 3$, le chargement optimal sera de prendre des objets 2 et 3 pour une valeur totale de 6.

Q 1. Montrer que l'heuristique suivante proposée pour ce problème n'a pas de ratio de garantie :

```
Trier les objets par v_i/p_i décroissant
Charge=0; val=0; i=1;
while ((i<=n) && (charge+p_i <=cap)){
   "prendre l'objet i":  charge= charge +p_i; val=val+v_i
    i++;}
```

exemple paramétré par n : $p_1 = 1, v_1 = 1, p_2 = n+1, v_2 = n, cap = n+1$
L'heuristique donne 1 comme valeur alors que l'optimal est n.

Q 2. On transforme maintenant l'heuristique comme suit :

```
Trier les objets par v_i/p_i décroissant
Charge=0; val=0;  i=1;
while ((i<=n) && (charge+p_i <=cap)){
     "prendre l'objet i":  charge= charge+p_i; val=val+v_i
      i++;}
si ((i<=n) && (val  <=v_i )) {
      "vider la valise et prendre juste l'objet i": val =v_i;}
```

Q 2.1. Montrer par un exemple que cette nouvelle heuristique ne produit pas toujours un chargement optimal.

L'exemple donné dans l'énoncé

Q 2.2. Montrer que cette nouvelle heuristique a un ratio de garantie inférieur ou égal à 2, i.e. que la valeur du chargement produit par l'heuristique est toujours supérieure ou égal à la moitié de la valeur optimale.

Soit val la valeur de la solution fournie par l'heuristique, opt la valeur optimale. Soit i_o l'indice correspondant à l'arrêt de la boucle dans l'heuristique ; Donc val= $max(sum_{i=1}^{i_o-1} v_i, v_{i_0})$. Soit charge=$sum_{i=1}^{i_o-1} p_i$ opt ne peut être supérieure à $sum_{i=1}^{i_o-1} + (v_{i_0}/p_{i_0}) * (cap - charge)$. Comme $charge + p_{i_0} >= cap$, $(v_{i_0}/p_{i_0}) * (cap - charge) \leq (v_{i_0}/p_{i_0}) * p_{i_0} = v_{i_0} \leq val$ donc $opt \leq 2 * val$.

Q 3. Que pensez-vous de la complexité du problème ?

NP-dur

<u>Exercice 4</u> : Ordonnancement préemptif *-Exercice Bonus*

Soit le problème suivant d'ordonnancement. On a n tâches à effectuer sur une machine (un processeur par exemple), chacune est donnée par sa date d'arrivée s_i et sa durée l_i. Les dates et durées sont des entiers correspondant aux nombres d'unités de temps. Une tâche i peut être lancée à toute date $\geq s_i$ et sa durée d'exécution est l_i. On se place dans le cas **préemptif :** à chaque nouvelle date, une tâche active peut être suspendue et une autre peut être (re-)lancée.

On cherche ici à minimiser la somme des temps de complétion des tâches, c.à.d. $\sum_{i=1}^{n} C_i$ où C_i est la date à laquelle la tâche i se termine dans l'ordonnancement.

Par exemple, si on a 2 tâches avec $s_1 = 2, l_1 = 2, s_2 = 0, l_2 = 5$, on peut choisir d'exécuter la tâche 2 au départ, puis la tâche 1 dès qu'elle arrive et enfin de terminer la 2. Ainsi la 1 sera terminée au temps 4, la 2 au temps 7 : la fonction objectif vaut 11. On aurait pu aussi exécuter entièrement la tâche 2 qui serait donc terminée au temps 5 puis la tâche 1 terminée au temps 7, obtenant 12 comme fonction objectif. La première solution est donc préférable et c'est d'ailleurs l'optimale (pourquoi ?).

Proposer un algorithme polynomial qui construit un ordonnancement préemptif optimal selon le critère ci-dessus. Justifier sa correction et analyser sa complexité.

le schéma est le suivant :

```
A chaque arrivée de tâche
 si elle se peut se terminer avant celle en cours,
       arrêter celle en cours,
       la placer dans la file avec sa durée restante,
       lancer l'exécution de  la nouvelle tâche
 sinon placer la nouvelle tâche dans la file avec sa durée.
```

```
A chaque fin de tâche
   prendre dans la file la tâche de durée minimale
```

Il faut faire un peu attention à l'implémentation pour garantir que c'est polynomial ...

pour la correction, supposons qu'une optimale soit différente de la gloutonne. Donc à un temps t la gloutonne et l'optimale diffèrent, alors qu'elles sont identiques avant. Soit x exécuté par la gloutonne, y par l'optimale. Vu l'algo le temps de complétion de x est inférieur ou égal à celui de y. soit tc ce temps Donc si on remplace dans l'optimale , les premiers tc temps d'exécution de y par une exécution de x, et les tc exécutions de x par une de y, on a encore une solution correcte. x se terminera au plus tard au temps où se terminait y. y se terminera au plus tard au temps où se terminait x. donc le temps de complétion de cette nouvelle solution est inférieur ou égal à celui de l'optimale.cette nouvelle solution est donc optimale est plus proche de la gloutonne : de proche en proche, la gloutonne est optimale.