Quelques Éléments de correction

Attention : Les éléments de réponse ci-dessus ne sont que des éléments de réponse et ne constituent pas une réponse complète !

Quelques questions d'application immédiate du cours - *7 pts*

1. Quelle(s) affirmation(s) est(sont) vraie(s) :
   A. $(n+2)^3 = \Theta(n^3)$ *Vraie puisque $(n+2)^3/n^3$ tend vers 1 quand n tend vers $+\infty$*
   B. $3^{n+2} = O(3^n)$ *Vraie $3^{n+2} = 9 * (3^n)$*
   C. $3^{n+2} = \Theta(3^n)$ *Vraie $3^{n+2} = 9 * (3^n)$*
   D. $3^{2n} = O(3^n)$ *Faux car, $3^{2n} = (3^n) * (3^n)$, donc $3^{2n}/3^n$ tend vers $+\infty$ quand n tend vers $+\infty$*
2. Soient Meill(n), Pire(n) et Moy(n) respectivement la complexité dans le meilleur des cas, le pire des cas et en moyenne d'un algorithme A sur une donnée de taille n. Quelle(s) affirmation(s) est(sont) toujours vraies ?
   A. Meil(n) est en $O(Moy(n))$ *Vraie puisque $Meil(n) \leq Moy(n)$*
   B. Meil(n) est en $\Theta(Moy(n))$ *Faux : par exemple le tri bulle ou une recherche séquentielle*
   C. Meil(n) est en $O(Pire(n))$ *Vraie puisque $Meil(n) \leq Pire(n)$*
   D. Moy(n) est en $\Theta(Pire(n))$ *Faux : par exemple le tri rapide*
   E. Moy(n) est en $O(Pire(n))$ *Vraie puisque $Moy(n) \leq Pire(n)$*
3. Soit la récurrence $T(n) = T(n/3) + n^2, T(0) = 1$. Quelle(s) affirmation(s) est(sont) vraie(s) :
   A. $T(n) = \Theta(n)$ B. $T(n) = \Theta(n^2)$ C. $T(n) = O(n^2 \log n)$ D. $T(n) = \Theta(n^2 \log n)$
   B,C.
   Par la version du Master Theorem ($2 > \log_3 1$) vue en cours on obtient que $T(n) = O(n^2)$. Cela permet de valider $T(n) = O(n^2 \log n)$ et de contredire $T(n) = \Theta(n^2 \log n)$. La première affirmation est fausse puisque $T(n) \geq n^2$. Pour justifier que $T(n) = \Theta(n^2)$, on peut par exemple montrer facilement que $n^2 \leq T(n) \leq 2*n^2$. On peut aussi directement remarquer que $T(n) = n^2+(n/3)^2+(n/9)^2+....+1 = \Theta(n^2)$
4. Soit T(n) = 3T(n/3) + n, T(0) = T(1) = 1. Quelle affirmation est fausse ?
   A. $T(n) = O(n^2)$ B. $T(n) = \Theta(n \log n)$ C. $T(n) = O(n \log n)$ D. $T(n) = \Theta(n^2)$
   D.
   On a $T(n) = O(n \log n)$ facilement par le Master Theorem. Donc $T(n) = \Theta(n^2)$ est fausse. On peut aussi remarquer que cela correspond à un arbre ternaire de hauteur $\log_3 n$ où chaque niveau "coûte" n : $T(n) = \Theta(n \log n)$
5. Soit A, B, C trois problèmes de décision. A est connu NP-complet. B se réduit polynomialement en A, A se réduit polynomialement en C. On peut affirmer (plusieurs réponses possibles) que :
   A. B est NP *Vrai : $B \leq_P A$ et A est NP.*
   B. C est NP *On ne peut pas l'affirmer, C peut être plus difficile.*
   C. B est NP-dur *B peut être par exemple en temps constant, donc non NP-dur.*
   D. C est NP-dur *Oui, puisque A est $NP-dur$ et $A \leq_P C$.*
   E. B est NP-complet *Non, puisque B peut ne pas être $NP-dur$.*
   F. C est NP-complet *Non, puisque C peut ne pas être NP.*
6. Supposons que P soit différent de NP. Quelle affirmation est vraie :
   A. Toute propriété NP est NP-complète.
   Faux : une propriété P est NP mais pas $NP-dure$ et donc pas NP-complète- si $P \neq NP$
   B. Toute propriété NP-dure est NP. *Faux : elle peut être beaucoup plus dure.*
   C. Toute propriété NP est NP-dure. *Faux, sinon la première affirmation serait vraie !*
   D. Aucune propriété NP-complète n'est P. *Vraie. Car si une propriété NP-complète était P, on aurait P=NP.*

7. Supposons que P soit différent de NP. Pour les graphes planaires, quelle propriété est(sont) polynomiale(s) ?
   A. l'existence d'un 2-coloriage
      Oui, le problème est polynomial , car le 2-coloriage est polynomial pour les graphes quelconques (cf cours), donc a fortiori pour les graphes planaires.
   B. l'existence d'un 3-coloriage
      Non, le 3-coloriage est $NP-$dur même pour les graphes planaires.
   C. l'existence d'un 4-coloriage
      Tout graphe planaire est coloriable en 4 couleurs, donc la propriété est toujours vraie, donc polynomiale.

   Remarque : pour cette question, toutes les réponses étaient dans les transparents du cours.

Les conférences - *7 pts*

L'Université a inauguré une très belle salle de conférences, l'Odyssée. De nombreuses réservations affluent pour cette salle. Pour simplifier, on supposera ici que chaque réservation est donnée par deux entiers, un entier représentant le premier jour de la conférence et un entier représentant la durée en jours de la conférence.

Par exemple, soient 6 demandes déterminées par les couples (début, durée) suivants : (1,4), (1,1), (3,1), (2,12), (4,1), (10,2). Bien sûr, on ne peut satisfaire toutes les demandes. Par exemple la conférence (1,4) occupe la salle les jours 1,2,3,4, donc si on accepte sa demande, on ne peut accepter comme autre demande que la demande (10,2). La demande (2,12) occuperait la salle les jours 2,3,4,5,6,7,8,9,10,11,12,13 : si on l'accepte on ne peut accepter comme autre demande que la demande (1,1).

1. Dans un premier temps, on cherche à satisfaire le maximum de demandes. Le problème est donc :

   Donnée : n un nombre de demandes, et une liste de n demandes représentées par des couples d'entiers naturels (debut, duree).
   Sortie : une liste de demandes acceptées qui soit correcte -i.e. on ne peut avoir le même jour deux conférences- et de cardinal maximal.

   Pour l'exemple ci-dessus, la donnée serait donc $6, (1,4), (1,1), (3,1), (2,12), (4,1), (10,2)$; la sortie pourrait être (1,1), (3,1), (4,1), (10,12) qui est une solution optimale.

   (a) Pour 3 réservations données par $(1,6), (5,3), (7,4)$, quelle serait une solution optimale ? $(1,6), (7,4)$

   (b) Pour 5 réservations données par $(1,5), (2,2), (4,4), (5,1), (7,2)$, quelle serait une solution optimale ? $(2,2), (5,1), (7,2)$

   (c) Donner un algorithme en $\Theta(n \log n)$ pour résoudre le problème. Justifier qu'il est correct et que sa complexité est bien $\Theta(n \log n)$.

   L'algorithme sera juste un algorithme glouton où on trie les demandes par exemple par date de fin croissante -cf cours

2. Après réflexion, on décide plutôt de privilégier le taux d'occupation de la salle donc de maximiser le nombre de jours de conférence. Le problème devient donc :

   Donnée : n un nombre de demandes, et une liste de n demandes représentées par des couples d'entiers naturels (debut, duree).
   Sortie : une liste de demandes acceptées qui soit correcte -i.e. on ne peut avoir le même jour deux conférences- et qui maximise le nombre de jours d'occupation de la salle.

   Pour l'exemple ci-dessus (1,4), (1,1), (3,1), (2,12), (4,1), (10,2), une solution optimale serait donc d'accepter uniquement les deux demandes $(1,1), (2,12)$ ce qui donne une durée d'occupation de 13 jours.

   (a) Supposons qu'on ait 5 réservations données par $(1,5), (2,2), (4,4), (5,1), (7,2)$. Quelle serait une solution optimale avec le nouveau critère ? $(1,5), (7,2)$ *pour une durée de 7 jours* .

   Pour résoudre le problème, on suppose que les demandes sont triées par début croissant et qu'on a deux tables debut et duree donnant respectivement le début et la durée d'une conférence. De

plus, on suppose qu'on a construit une table *suiv* telle que $suiv[i]$ donne le plus petit $j > i$ tel que $debut[j] \geq debut[i] + duree[i]$, c.à.d. le plus petit numéro de conférence supérieur à i qui soit compatible avec la demande i. Si un tel j n'existe pas, on posera $suiv[i] = n$.
Pour l'exemple $(1,1),(1,4),(2,12),(3,1),(4,1),(10,2)$, cela donnerait :

i	0	1	2	3	4	5
debut	1	1	2	3	4	10
duree	1	4	12	1	1	2
suiv	2	5	6	4	5	6

(b) Justifiez que si on accepte la conférence i, on ne peut pas accepter les conférences de numéro compris strictement entre i et $suiv[i]$ (i.e. de numéro j tel que $i < j < suiv[i]$).
Si $i < j < suiv[i]$, par définition de $suiv[i]$, j n'est pas compatible avec i.

(c) Proposez un algorithme en $\Theta(n)$ qui calcule une solution optimale. On ne prendra pas en compte le temps de calcul des tables debut, duree et suiv qui sont supposées avoir été précalculées.
Aide : On pourra utiliser la programmation dynamique, et dans un premier temps uniquement calculer la durée optimale de réservation -pour l'exemple ce serait 13, avant de sortir également les numéros des demandes à accepter -pour l'exemple $0, 2$.
Prenons les demandes dans l'ordre de début croissant. Une solution peut être vue comme une suite de choix : Pour chaque demande i : si je l'accepte, je ne pourrai rien accepter avant la demande $suiv[i]$, sinon, je passe à la tâche $i+1$.
Donc si on appelle dureeopt[i] la durée maximale des demandes que l'on peut satisfaire à partir de la demande i, on a simplement :
dureeopt[n]=0
dureeopt[i]=max (duree[i]+dureeopt[suiv[i]],dureeopt[i+1]), i >0
Ce qu'on "cherche" est dureeopt[0].

```
//dureeopt tabbleau de n entiers
dureeopt[n]=0
pour i de n-1 à 0
    dureeopt[i]=max (duree[i]+dureeopt[suiv[i]],dureeopt[i+1]);
fin pour;
//remontée
d=dureeopt[0];
i=0;
tant que (d>0)  //ou (i<n)
    si (d>dureeopt[i+1])
       //accepter demande i
        sortir(i); d=d-duree[i];i=suivi[i];
     sinon i=i+1;
fin tant que
```

(d) Proposer un algorithme pour construire la table *suiv* et analyser sa complexité. L'efficacité de l'algorithme sera un critère d'évaluation.
Un algorithme en $O(n^2)$ était facile à obtenir. On pouvait obtenir un algorithme en $O(n \log n)$ en utilsant par exemple la dichotomie. On pouvait aussi obtenir un algorithme en $O(n + nbj)$ avec nbj la date maximale de fin d'une conférence.

Typable ? - *7 pts*

Les programmes en C peuvent être difficiles à corriger car le langage n'est que faiblement typé et permet des accès incontrôlés à la mémoire. WhatsAt est un outil de débuggage dynamique conçu par Polishchuk, Liblit, et Schulze en 2006 : il examine le *tas* d'un programme en cours d'exécution et vérifie sa cohérence avec les types de données déclarés. Pour chaque zone de mémoire allouée, l'outil détermine le type des données stockées, ou bien identifie des blocs « non-typables » qui indiquent une probable corruption de la mémoire ou une violation de la discipline de type.

Dans cet exercice, on considère une version simplifiée du problème qui est tout de même NP-complet.

Un *site* est une portion contiguë du tas qui possède un unique type. Un site peut contenir des sous-sites (par exemple, si c'est un `struct`). Un type peut être affecté à un site lorsque toutes les conditions suivantes sont réunies :

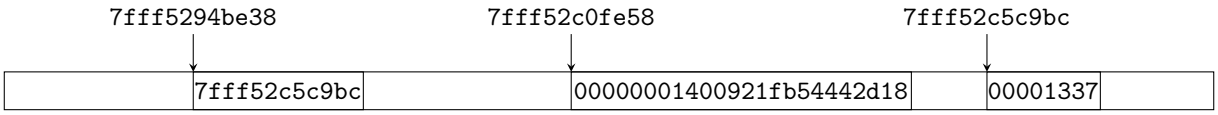
- La taille du site est égale à la taille du type (telle que `sizeof(type)` la renvoie).
- Les valeurs présentes dans le site sont cohérentes avec le type.
- Les sous-sites éventuels sont typables de façon cohérente avec le type.
- Si un site a un type « pointeur vers α », alors le site pointé a le type α.

Par exemple, voici des déclarations de types :

```
typedef enum { ZERO, FOO, BAR=0x42 } atom;
typedef struct { atom a; double b; } big;
typedef atom* remote_atom;
```

Les variables de type `atom` occupent la même taille qu'un `int` (`sizeof(atom) == 4`), mais ne peuvent prendre que les valeurs 0 (ZER0), 1 (FOO) et 42 (BAR). La structure `big` occupe donc a priori 12 octets [1]. Sur une machine moderne les pointeurs occupent 64 bits, donc la structure `atom*` occupe a priori 8 octets.

Voici par exemple une schématisation du tas d'un programme en cours d'exécution :



Le site de gauche (adresse `7fff5294be38`) ne peut pas avoir le type `remote_atom` car la valeur pointée (à l'adresse `7fff52c5c9bc`) n'est pas compatible avec le type `atom`. Le site du milieu (adresse `7fff52c0fe58`) pourrait avoir le type `big`, avec `a==FOO` et `b==3.141593`. Le site de droite (adresse `7fff52c5c9bc`) ne peut pas avoir le type `big` car il n'est pas assez gros.

Cet ensemble de sites n'est pas typable avec les trois types définis ci-dessus. Mais si on ajoutait :

```
typedef int* status_ptr;
typedef int exit_status;
```

Alors on pourrait affecter le type `status_ptr` au site de gauche et le type `exit_status` à celui de droite.

Questions

1. Justifier que HEAP TYPABILITY (cf. figure 1 ci-dessous) appartient à la classe NP.

   INPUT :
   - Deux entiers n, k
   - Une liste de n sites $(a_1, t_1, c_1), \ldots, (a_n, t_n, c_n)$ composés chacun d'une adresse de début a_i, d'une taille t_i, et d'un tableau de t_i octets.
   - Une liste de k déclarations de type de la forme `typedef (...) type_i;` pour i de 1 à n.

   QUESTION : existe-t-il une affectation d'un des types déclarés à chaque site qui respecte les contraintes ?

   FIGURE 1 – Définition du problème HEAP TYPABILITY

   Le certificat est juste une affectation de types à chaque site donc de taille $O(n \log k)$ bien polynomialement bornée par rapport à la tialle d el'netrée. La vérification consiste à vérifier la cohérence du typage.

   Pour démontrer que le problème est NP-dur, on va réduire le problème classique 3-COLORABILITY, décrit figure 2 (qui est NP-complet).

2. Expliquer ce que la réduction doit prendre en entrée et produire en sortie et quel doit être le lien entre l'entrée et la sortie pour que la réduction soit correcte. (On ne demande pas dans cette question de proposer la réduction mais uniquement d'en indiquer le cahier des charges).

1. ou 16, si le compilateur force le `double` à être aligné sur une frontière de 64 bits...

INPUT :
— Un entier n
— Un graphe non-orienté (V, E) à n sommets (décrit par sa matrice d'adjacence).
QUESTION : existe-t-il une manière de colorier les sommets avec 3 couleurs sans que deux sommets adjacents aient la même couleur ?

FIGURE 2 – Définition du problème 3-COLORABILITY

La réduction prend en entrée une instance "in" de 3-COLORABILITY et retourne une instance "out" deHEAP TYPABILITY. Pour qu'elle soit correcte il faut que "out" soit une instance positive deHEAP TYPABILITY si et seulement si "in" est une instance positive de 3-COLORABILITY.

Dans la réduction, on utilisera toujours les déclarations de type suivantes :

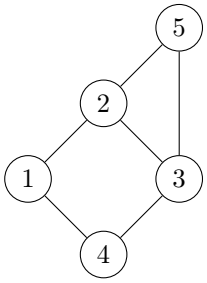
```
typedef enum { ZERO } nothing;

typedef struct { nothing t; } red_v;
typedef struct { nothing t; } green_v;
typedef struct { nothing t; } blue_v;

typedef struct { red_v    *x; blue_v  *y;} rb_e;
typedef struct { red_v    *x; green_v *y;} rg_e;
typedef struct { green_v  *x; red_v   *y;} gr_e;
typedef struct { green_v  *x; blue_v  *y;} gb_e;
typedef struct { blue_v   *x; red_v   *y;} br_e;
typedef struct { blue_v   *x; green_v *y;} bg_e;
```

L'idée consiste à créer un tas qui contient un site par sommet du graphe ainsi qu'un site par arête. Les sites associés aux sommets doivent être de type `(...)_v` tandis que ceux qui sont associés aux arêtes sont de type `(...)_e`.

3. Avec ces définitions de type, quel(s) type(s) pourrai(en)t être affecté(s) à un site de 4 octets contenant la valeur `00000000` ?
   `nothing, red_v,green_v,blue_v`
4. Déterminer un 3-coloriage valide du graphe ci-dessous, puis construire le tas associé par la réduction, en indiquant les types de chaque site (vous pouvez utiliser des flèches pour représenter les pointeurs plutôt que d'indiquer explicitement des adresses).



Par exemple : Le sommet 1 en rouge, le 2 en vert, le 3 en rouge, le 4 en vert, le 5 en bleu
Le tas associé aura 5 sites de 4 octets correspondant aux 5 sommets et 5 sites de 8 octets correspondant aux arcs. Dans un site correspondant à un sommet, la valeur sera ZERO, pour un site correspondant à un arc, la valeur correspondra aux deux adresses des sommets extrémités de l'arc.

5. Généraliser la réduction suggérée par l'exemple ci-dessus.
   La réduction construit donc un tas avec n sites de 4 octets, p sites de 8 octets correspondant aux arcs. Les valeurs des sites-sommets sont ZERO, celles des arcs correspondent aux deux adresses de leurs extrémités.
   On vérifie facilement que la réduction est polynomiale.
6. Justifier que, lorsqu'un graphe est 3-coloriable, alors le tas produit par la réduction sera typable avec les définitions de types données ci-dessus. *Si le graphe est coloriable, soit un coloriage correct ; on affecte alors à chaque "site-sommet" le type correspondant à la couleur, à chaque "site-arc" le type correspond aux couleurs de ses extrémités. De la correction du coloriage, on déduit la correction du typage* .

7. Justifier que, lorsqu'un tas produit par la réduction est typable, si un site a le type `nothing`, il correspond à un sommet isolé du graphe, i.e. qui n'est extrémité d'aucune arête. *Si un site correspond à un sommet extrémité d'un arc, pour pouvoir le typer, il faut que le site du sommet soit typé par un des trois types* `red_v,green_v,blue_v.`
.
8. Justifier que lorsqu'un tas produit par la réduction est typable, alors le graphe de départ est 3-coloriable. *On peut ne s'intéresser qu'aux sommets non isolés, les autres ne contraignant pas le coloriage. Comme le tas est typable, d'après ce qui précède, chaque sommet non isolé est typé par un des trois types* `red_v,green_v,blue_v.`
Colorions un sommet par la couleur correspondante à son type : soit alors un arc : comme son site correspondant est typable, ses deux extrémités sont typées par des couleurs différentes -par définition des types-arcs- qui correspondent aux couleurs des sommets -car l'arc contient les adresses des deux extrémités, et donc on vérifie facilement que le coloriage est correct.
9. Justifier que le problème HEAP TYPABILITY est NP-complet. *Se déduit facilement des questions précédentes*