EXAMEN D'ALGORITHMIQUE ET COMPLEXITE

Master Informatique, première année, janvier 2017

Vous avez droit a une feuille A4 recto-verso.

Calculatrices, téléphones et ordinateurs portables, tablettes, lunettes connectées et montres connectées, oreillettes sont interdits.

Répondez aux questions dans l'ordre. Numérotez vos réponses.

La clarté de vos réponses est essentielle : programmer, c'est rédiger.

1 Mélanger un tableau (4 lignes, 2 points, 5 mn)

Proposez une méthode en $O(n)$ pour mélanger un tableau de n éléments. Expliquez quel peut-être l'intérêt d'une telle méthode.

2 k ième élément (6 lignes, 2 points, 10 min)

Proposez une méthode pour trouver le k ième élément dans un tableau non trié de n éléments, $e_0, e_1, \ldots e_{n-1}$. Vous supposerez que les e_i sont rangés dans un ordre aléatoire. Le tableau et k sont donnés. Par convention, le 0 ième élément est le plus petit. Faites l'analyse en complexité de votre méthode. Est-elle optimale ?

3 Paradoxe des anniversaires (5 lignes, 2 points, 10 mn)

Expliquez ce qu'est le paradoxe des anniversaires, en 3 lignes au plus. Quelle est la conséquence sur une table de hachage ayant 365 clefs possibles ?

4 Division entière (8 lignes, 2 points, 15 mn)

Arthur a programmé l'addition, la soustraction, la multiplication et la comparaison de grands entiers (sans utiliser la librairie pré-existante, nums). Il lui reste la division à programmer. Sa fonction (division a b) doit rendre le couple (q, r) avec $q = a \div b$ et $r = a \bmod b$. On suppose $a \geq 0$ et $b > 0$.

1. Décrire, en français et en 2 lignes, la méthode triviale (par soustraction).
2. Proposez en 6 lignes au plus le principe d'une méthode plus efficace.

5 Faites le programme (1 page, 7 points, 30 mn)

Une entreprise produit du X et du Y. Produire une unité de X nécessite une unité de A et 2 unités de B. Produire une unité de Y nécessite 3 unités de A et 2 unité de B. L'entreprise dispose de 12 unités de A et de 30 unités de B. Le bénéfice est de 2 UM (unités monétaires) par unité de X produite, et de 1 UM par unité de Y produite. L'entreprise souhaite maximiser son bénéfice. Des quantités fractionnaires de X et de Y peuvent être produites (ce ne sont pas des voitures et des camions), et des quantités fractionnaires de A et de B peuvent être consommées. Vous n'avez pas besoin de calculatrice : les valeurs sont choisies pour que les calculs soient triviaux.

1. Formulez le problème ; notez x la quantité de X produite, y la quantité de Y produite, a la quantité de A non utilisée, b la quantité de B non utilisée.

2. Dessinez et hachurez le polytope admissible dans le plan x, y.

3. Résolvez le problème, numériquement **et** graphiquement.

4. Ajoutez la contrainte : toutes les ressources en A et en B doivent être consommées. Y a-t-il une solution ? Si oui, laquelle ? Sinon, pourquoi ?

5. La matrice qui apparaît dans ce problème est-elle totalement unimodulaire ? (1 ligne)

6 Affectation (1/2 page, 3 pts, 20 mn)

Une entreprise doit effecteur les tâches A, B et C. Trois machines sont disponibles : U, V et W. Il faut affecter une machine (et une seule) à chaque tâche. Inversement, chaque tâche est effectuée par une seule machine, soit U, soit V, soit W. Faire A avec U coûte 10 UM (unité monétaire). Faire A avec V coûte 1 UM. Faire A avec W est impossible. Faire B avec U est impossible. Faire B avec V coûte 2 UM. Faire B avec W coûte 5 UM. Faire C avec U est impossible. Faire C avec V coûte 5 UM. Faire C avec W coûte 5 UM. L'entreprise souhaite réaliser A, B et C au coût moindre.

1. Posez ce problème, et résolvez-le. Eventuellement, dessinez-le.

2. Ce problème est réductible à plusieurs problèmes classiques ; mentionnez-en deux.

7 Permutation (2 lignes, 2 points, 10 mn)

Soit $E_n = \{0, \ldots, n-1\}$. Une permutation P est une bijection de E_n. Elle peut être représentée par le vecteur $[P(0); \ldots P(n-1)]$. Tous les éléments de E_n apparaissent une fois et une seule dans ce vecteur. Toute permutation P a une période p ; c'est le plus petit entier non nul tel que $P^{(p)}$ (la permutation P itérée p fois) est la permutation identité. Toute permutation peut être décomposée en cycles. Par exemple, $Q = \{(0, 2, 4), (1, 3), (5)\} \Rightarrow Q(0) = 2$, $Q(2) = 4$, $Q(4) = 0$, $Q(1) = 3$, $Q(3) = 1$; $Q(5) = 5$. Le vecteur de Q est $[2; 3; 4; 1; 0; 5]$. Il est possible de passer d'une représentation à une autre en $O(n)$. Décrivez en 2 lignes au plus une méthode efficace pour calculer la période d'une permutation donnée par son vecteur.

8 Correction

8.1 Mélanger un tableau

Soit n la taille du tableau. Pour i de 0 à n-1, tirer un entier $0 \leq j < n$ au hasard et échanger le i ième et le j ième élément du tableau. La méthode est en $O(n)$.

Pour mélanger une liste, la convertir en tableau, mélanger le tableau, puis convertir le tableau mélangé en liste.

Le tri rapide fonctionne lentement sur certains tableaux. Mélanger le tableau évite ce cas de figure : on peut alors prendre comme pivot l'élément en début de tableau ou de liste.

8.2 k ième élément dans un ensemble N de n éléments

On suppose $0 \leq k < n$. C'est trivial si $n = 1$ et $k = 0$. Sinon : choisir un pivôt π au hasard dans l'ensemble, qu'on partage en P (plus petits que π), E (égaux à π), G (plus grands que π). Notons $|K|$ le cardinal d'un ensemble K. Si $k < |P|$ alors chercher le k ième élément de P. Si $|P| + |E| \leq k < n$, alors chercher le $k - |P| - |E|$ élément de G. Sinon l'élément est dans E donc vaut π.

Comme le pivot a été choisi au hasard, P et G sont très probablement à peu près de même taille, donc de taille proche de $n/2$.

D'où la récurrence : $T(n) = T(n/2) + n$, qui a pour solution $T(n) = O(n)$.

Remarque : Si on n'a pas de chance, l'algorithme est en $O(kn)$, d'où l'intérêt du mélange. Cette méthode est "une moitié du tri rapide (quicksort)" : il y a un seul appel récursif, au lieu de deux pour le tri rapide. L'idée est de ne trier que la partie qui contient le k ième élément.

Les erreurs commises :

Le k ième élément n'est pas l'élément de valeur k.

Le k ième élément n'est pas l'élément à l'indice k (ça le serait si l'ensemble était représenté par un tableau trié) ; d'ailleurs, l'énoncé définit le 0 ième élément : c'est le plus petit élément du tableau.

Bien sûr, il est possible de trier le tableau en $O(n \log n)$, et ensuite d'accéder au k ième en $O(1)$.

Bien sûr, il est possible de trouver le k ième en $O(kn)$: chercher le $k - 1$ ième élément dans l'ensemble amputé de son plus petit élément. Mais si $k = n/2$ cette méthode est en temps quadratique.

Le programme suivant n'était pas demandé :

```
module L=List;;
module A= Array;;
(* cmp est la fonction de comparaison;
   l est la liste; n est la longueur de l *)
let rec kieme cmp l k n = match l with
   | [] -> failwith "bug!"
   | [t] -> if k=0 then t else failwith "bug2!"
   | p::_ -> let lower = L.filter (function x -> cmp x p < 0) l in
```

```
          let nlower = L.length lower in
          if k < nlower then kieme cmp lower k nlower
          else let greater = L.filter (function x -> cmp x p > 0) l in
               let ngreater = L.length  greater in
               let nequals = n - nlower - ngreater in
               if k >= nlower + nequals
               then kieme cmp greater (k-nlower-nequals) ngreater
               else p  ;;
let rec iaj i j = if i=j then [i] else i::(iaj (i+1) j);;
let melange_tableau tab = let n=A.length tab in
        for i=0 to n-1 do let j= Random.int n in
   let tmp=tab.(i) in tab.(i) <- tab.(j); tab.(j) <- tmp
        done; tab;;
let melange_liste l = A.to_list (melange_tableau (A.of_list l));;
let lll = melange_liste (iaj 0 100000);;
for i=0 to 100 do let k = Random.int 100000 in
        assert (kieme ( - ) lll k 100001 = k)
done;;
```

8.3 Paradoxe des anniversaires

Il suffit de 23 personnes dans une pièce pour qu'il y ait (approximativement) une chance sur deux pour que deux de ces personnes aient la même date d'anniversaire (même en tenant compte des années bissextiles).

Donc dans une table de hachage avec 365 tiroirs, il y a une chance sur 2 pour qu'il y ait une collision (deux éléments dans le même tiroir) dès que la table contient 23 éléments.

Précision : ce phénomène n'est donc dû ni à une erreur de programmation dans la gestion de la table de hachage, ni à un défaut intrinsèque de cette structure de données.

8.4 La division euclidienne entre deux entiers

La division entière de a par b rend (q, r), avec q le quotient et r le reste.

La méthode triviale. Si $a < b$ alors $(q, r) = (0, a)$. Sinon soit (q', r') la division de $a - b$ par b ; alors $(q, r) = (1 + q', r)$.

La "division russe" est plus efficace que la division triviale :

Si $a < b$ alors $(q, r) = (0, a)$; sinon, soit p la plus grande puissance de 2 telle que $p \times b \leq a$. Soit (q', r') la division de $a - p \times b$ par b ; alors $(q, r) = (p + q', r')$. Il suffit d'un nombre logarithmique ($O(\log_2 q)$) de soustractions $a - p \times b$.

Notons que $p \times b$ peut se calculer avec seulement des additions : $2b = b + b, 4b = (2b) + (2b), 8b = (4b) + (4b)$, etc. En fait, cette division n'utilise que les opérations $<, +, -$; elle n'utilise pas le produit, comme le prouve le programme suivant (qui n'était pas demandé).

```
(* (util a b p p_b) calcule (p, p_b) tels que: p est une puissance de 2,
   p_b=p*b et p*b <= a < 2*p*b; il utilise seulement + et > *)
let rec util a b p p_b =
        let deux_pb = p_b + p_b in
```

```
        if deux_pb > a then (p, p_b)
        else util a b (p+p) (p_b+p_b) ;;
(* (divise a b) calcule (q, r) avec q= a / b et r = a mod b,
   avec seulement des opérations  <, -, + *)
let rec divise a b = if a < b then (0, a)
        else let (p, pb)= util a b 1 b in
             let (q', r') = divise (a - pb) b in
             let (q, r)= (p + q', r' ) in
             (* assert( q = a / b && r= a mod b); just to check *)
             (q, r) ;;
```

La suite n'était pas demandée ; ce sont d'autres réponses possibles.

Méthode 2 : la méthode scolaire. Diviser le nombre formé par les deux premiers chiffres de poids fort de a par le premier chiffre (de poids fort) de b. L'intérêt est qu'il suffit le plus souvent d'un seul essai pour trouver le premier chiffre du quotient q. Cette méthode utilise $+, -, \times, \leq$.

Méthode 3 : trouver q par dichotomie entre p et $2p$, où $2p$ est la plus petite puissance de 2 plus grande que a. Mais diviser par 2 semble nécessaire pour la dichotomie... Cela peut se régler de deux façons : remarquer que la différence entre p et $2p$ est une puissance de deux ; ou bien programmer la division par 2.

Méthode 4, par Newton. Implanter une arithmétique sur de grands flottants au dessus de l'arithmétique sur les grands entiers. Calculer l'inverse de b par Newton : $f(x) = 1/x - b \Rightarrow f'(x) = -1/x^2$ et $N(x) = x(2 - bx)$. Finalement multiplier a par l'inverse de b.

8.5 Faites le programme

La principale difficulté de cet exercice était la traduction de l'énoncé en équations.

Rappel : a est ce qui reste de A, b est ce qui reste de B. Il faut maximiser le bénéfice : $\phi = 2x + y$.

Pour la ressource A, il en est utilisé : $1x + 3y \leq 12 \Rightarrow a = 12 - (x + 3y)$

Pour la ressource B, il en est utilisé : $2x + 2y \leq 30 \Rightarrow b = 30 - (2x + 2y)$

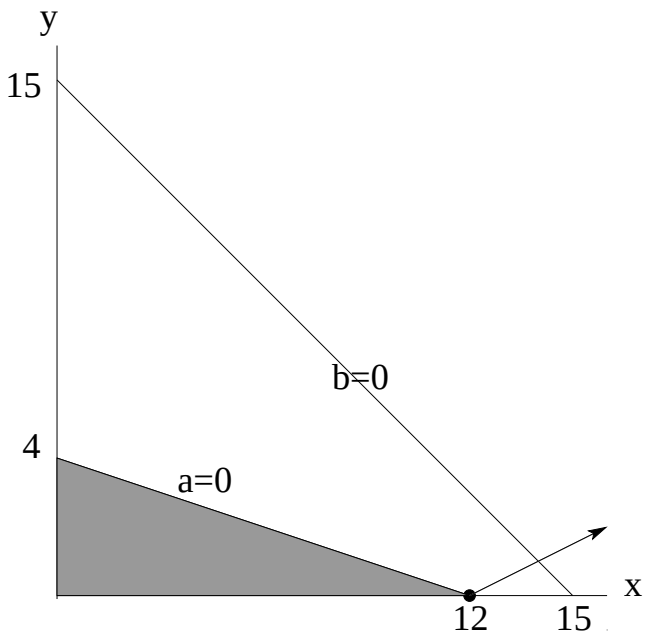
1, 2, 3. Le tableau initial est :

$$\begin{array}{ll} \max 2x + y & \max 2x + y \\ 1x + 3y \leq 12 \Rightarrow 1x + 3y + a = 12 & a = 12 - (1x + 3y) \\ 2x + 2y \leq 30 \Rightarrow 2x + 2y + b = 30 & b = 30 - (2x + 2y) \end{array}$$

Le dessin ci-dessous montre la solution et le pivot (colonne x, ligne a). Le tableau final est :

$$\begin{array}{l} \max 24 - 2a - 5y \\ x = 12 - a - 3y \\ b = 6 + 2a + 4y \end{array}$$

Donc la solution est : $x = 12, y = 0, a = 0, b = 6, \phi = 24$**.**



4. Impossible : les droites $a = 0$ et $b = 0$ se coupent en un point de coordonnée y négative, donc hors du polyèdre admissible.

5. La matrice n'est pas totalement unimodulaire : elle contient des coefficients différents de 0, 1, -1.

8.6 Affectation de ressources

1. Il faut calculer le couplage (une case par ligne, une case par colonne) de coût moindre (c'est 17) dans cette matrice carrée :

	A	B	C
U	**10**	∞	∞
V	1	**2**	5
W	∞	5	**5**

Ce qui suit n'était pas demandé. On trouve vite la solution, car le problème est de petite taille. Comment la prouver ? Enlèvons 10 à la ligne U, 2 à la colonne B, et 5 à la colonne C : cela soustrait 17 à tous les couplages parfaits ; la matrice obtenue a toutes ses entrées positives ou nulles ; un couplage de coût nul, donc forcément optimal, apparaît (en gras) :

	A	$B-2$	$C-5$
$U-10$	**0**	∞	∞
V	1	**0**	0
W	∞	3	**0**

Cette matrice finale prouve que, dans la matrice initiale, le côut moindre est bien 17 et que le couplage optimal est le même : $A \leftrightarrow U, B \leftrightarrow V, C \leftrightarrow W$.

2. Le couplage parfait de coût moindre est un problème de flot de débit donné et de coût minimal. Il peut être résolu par programma-

tion linéaire, avec une matrice sous-jacente qui est totalement unimodulaire. Cette propriété garantit qu'on trouvera une solution entière au sens suivant : par chaque arc, il transite soit un flot de 1, soit un flot de 0 (pas de flot 1/3 dans un arc et 2/3 dans l'autre, par exemple).

Remarque : si tous les coûts de la matrice sont entiers, les coûts de tous les couplages (parfaits ou non) sont bien sûr entiers. Mais cela ne prouve pas la totale unimodularité de la matrice ; il y a beaucoup de problèmes de programmation linéaire où la solution est entière sans que la matrice sous-jacente soit totalement unimodulaire. C'est le cas des problèmes 3-SAT formulés comme des problèmes de programmation linéaire.

Ce supplément n'était pas demandé. Voici une formulation du problème sous forme de programme linéaire, qui ne considère pas de flot, et qui généralise la preuve de la question précédente. Soit M la matrice initiale des coûts. Les inconnues sont les valeurs L_l et C_c : il faut soustraire L_l à tous les éléments de la ligne l, et il faut soustraire C_c à tous les éléments de la colonne c pour obtenir la matrice "finale", qui prouve le couplage optimal. Les contraintes sont : $M_{lc} - L_l - C_c \geq 0$ pour tous les l, c. Il faut maximiser $\sum_l L_l + \sum_c C_c$. Il y a donc $2n$ inconnues (sans compter les variables d'écart). L'"algorithme hongrois" résout ce problème.

8.7 Permutation

Calculer en $O(n)$ la décomposition en cycles de la permutation P. La période de P est le plus petit commun multiple des longueurs des cycles de P.