INF431

Premiers exercices d'algorithmique
CORRIGÉ
Version: 1475:2400M

1 Variations sur les mariages stables

1.1 Postes multiples

L'algorithme de Gale et Shapley permet de trouver une affectation de n candidats à n services d'une administration, chaque service classant les n candidats selon ses propres critères et chaque candidat classant les n services selon ses préférences. Supposons maintenant qu'il n'y ait que m services avec $m < n$, mais que chaque service dispose de plusieurs postes : plus précisément on suppose que le i-ème service a toujours une unique liste de préférence mais qu'on doive lui affecter n_i candidats (avec $\sum_{i=1}^{m} n_i = n$).

Question 1 Comment résoudre ce nouveau problème d'affectation ? ◇

Solution. La solution la plus simple est de faire une "réduction" du nouveau problème à un problème du type traité en cours : On construit un nouvel ensemble de listes de préférences des candidats en dupliquant n_i fois le $i$ème service dans la liste de chaque candidat pour que ces listes deviennent de taille n. De même on obtient un nouvel ensemble de listes de préférences des services en dupliquant n_i fois la liste du i-ème service (autrement dit on considère une copie de la liste par poste disponible). L'affectation obtenue par Gale-Shapley pour ces nouvelles listes donne immédiatement une affectation stable du problème initial. □

1.2 Préférences de caste

Supposons que notre administration soit noyautée par les anciens élèves d'une célèbre école (les Y) : chaque service désire avant tout recruter un Y pour bénéficier de son réseau d'amitiés, et classe donc systématiquement les Y devant les autres candidats. Inversement tous les candidats classent les postes de direction devant les postes d'exécutants.

Question 2 Montrer que s'il y a k postes de direction et k candidats qui sont Y, alors chaque poste de direction est occupé par un Y. ◇

Solution. Supposons qu'il existe une affectation stable telle qu'un Y, appelé A, soit affecté à un poste a d'exécutant, et montrons l'absurdité de la chose. Par le principe des tiroirs il existe aussi un poste de direction b occupé par un non-Y de nom B.

Mais alors A préfère le poste b et réciproquement le service qui propose b a classé A devant B : l'affectation n'est pas stable, on a bien une contradiction. □

1.3 Préférences incomplètes

L'expérience montre qu'il existe des individus qui préfèrent rester célibataire que d'être marié à une personne qui ne leur plaît pas suffisamment... On modélise cela en demandant à chaque homme ou femme de se classer dans sa liste de préférence (éventuellement en dernière position). Ceci amène à étendre la définition de stabilité aux couplages non nécessairement parfaits. Un couplage est instable s'il contient des éléments $h \in H$ et $f \in F$ qui se préfèrent mutuellement à leurs fiancés respectifs dans

le couplage, ou s'il contient un élément $x \in H \cup F$ qui préfère rester célibataire qu'épouser son ou sa fiancé(e) dans le couplage. De manière équivalente l'instabilité du couplage revient à l'existence d'une paire $(x, y) \in (H \times F) \cup \{(x, x) \mid x \in H \cup F\}$ telle que $y >_x C(x)$ et $x >_y C(y)$, où $C(x)$ désigne le ou la fiancée de x dans le couplage, ou x lui-même si x n'est pas fiancé.

Question 3 Montrer que dans un couplage stable personne ne peut être marié à un individu qui apparaît plus bas que lui-même dans sa liste de préférence. ◇

Solution. Il suffit de remarquer que si x est marié à un individu qui apparaît plus bas que lui-même dans sa liste de préférence, la paire (x, x) est instable. □

Cette propriété implique qu'on peut tronquer la liste de chaque individu à partir de la position où il apparaît : on parle de liste incomplète, et on dit que x apprécie y si y apparaît dans sa liste tronquée.

Question 4 Étendre l'algorithme de Gale et Shapley aux listes incomplètes, et montrer :
(a) que l'algorithme termine
(b) que, si une femme reçoit une proposition d'un homme h alors soit elle n'apprécie pas h, soit elle sera dans la suite fiancée à h ou à un homme au moins aussi bien placé que h dans sa liste.
(c) et que le couplage obtenu est stable. ◇

Solution. L'algorithme est le même : les hommes non fiancés proposent aux femmes (ou décident de rester célibataire) dans l'ordre de leur liste de préférence. Les femmes acceptent à tout moment la proposition la plus avantageuse qui leur est faite (tenant compte de la possibilité qu'elles ont de rester célibataires). Voici le pseudo-code :

tant que il existe un homme libre et n'ayant pas demandé toutes les femmes qu'il apprécie en mariage
- **soit** h un tel homme
- **soit** f la femme préférée de h parmi celles qu'il apprécie et qu'il n'a pas encore demandées en mariage
  - **si** f est libre et apprécie h
  - **alors** h et f se fiancent
  - **sinon si** f est fiancée à h' mais lui préfère h
  - **alors** f quitte h' (qui redevient libre) et se fiance avec h

(a) La terminaison vient, comme dans le cours, du fait que la même proposition n'est pas faite deux fois. (b) Les invariants de boucles sont d'une part qu'à tout moment l'ensemble des couples fiancés est un couplage, ce qui se vérifie immédiatement, et d'autre part qu'une femme fiancée une fois le reste jusqu'au bout, avec des hommes successifs de mieux en mieux placés dans sa liste de préférence, ce qui se vérifie par une analyse des cas dans lesquels une femme peut changer de fiancé. (c) La preuve de stabilité se fait par l'absurde comme dans le cours, en utilisant le second invariant. □

2 Le problème k-somme

De nombreux problèmes de décision ou d'optimisation sont exprimés comme des problèmes de sommes de sous-ensembles. Le plus classique, *somme de sous-ensembles*, demande, étant donnés un ensemble E de N entiers et un entier $S \in \mathbb{N}$, s'il existe un sous-ensemble de E dont les éléments se somment à S. Ce problème est NP-complet, ce qui signifie que l'on ne connaît pas d'algorithme permettant de le résoudre en un nombre d'opérations croissant polynomialement avec N. Il devient cependant de complexité polynomiale si l'on borne *a priori* la cardinalité du sous-ensemble. Il s'agit alors du problème suivant :

> PROBLÈME k-SOMME : Étant donnés un ensemble E de N entiers et un entier S, déterminer s'il existe $s_1, \ldots, s_k$ distincts dans E tels que $s_1 + \cdots + s_k = S$.

Pour la mesure de la complexité dans ce problème, on fait l'hypothèse que l'addition et la comparaison d'entiers ont un coût unitaire, ce qui est réaliste pour des entiers stockés sur un nombre fixe de mots machines, comme ceux du type `int` de Java ou C.

De nombreuses questions de géométrie algorithmique se ramènent au problème k-somme. Par exemple, l'inclusion d'un polygone dans un autre à translation près se réduit à 4-somme. Plus simplement, étant donnés N points du plan, décider si 3 d'entre eux sont alignés est relié à 3-somme. Par ailleurs, 2-somme est utilisé semble-t-il comme question de test dans certains entretiens d'embauche.

Dans tout cet exercice, les algorithmes demandés sont à présenter sous forme de pseudo-code.

2.1 Approche naïve

Question 5 Proposer un algorithme simple pour résoudre le problème 2-somme avec une complexité quadratique. ◇

Solution. Si $E = \{s_1, \ldots, s_N\}$, on parcourt les paires d'éléments (s_i, s_j) avec $1 \leq i < j \leq N$ et on teste si $s_i + s_j = S$. Le pseudo-code correspondant s'écrit donc :

procédure NAIVE2SUM(E, S, N)
 pour $i \leftarrow 1$ **à** N
  faire **pour** $j \leftarrow i+1$ **à** N
   faire si $E[i] + E[j] = S$ **alors renvoyer** TRUE
 renvoyer FALSE

□

Question 6 En généralisant cette méthode, proposer un premier algorithme de complexité polynomiale pour le problème k-somme, pour $k \geq 2$. ◇

Solution. En suivant la même structure, avec $k-1$ niveaux de boucles imbriqués, le problème k-somme est résolu en $O(n^k)$ opérations. Une version récursive est assez naturelle.

procédure NAIVEKSUM(E, S, N, k)
 si $k = 0$ **alors renvoyer** $(S == 0)$
 si $N = 0$ **alors renvoyer** FALSE
 pour $i \leftarrow 1$ **à** N
  faire si NAIVEKSUM$(E[i+1..N], S - E[i], N-1, k-1)$ **alors renvoyer** TRUE
 renvoyer FALSE

□

2.2 Tri

Question 7 Proposer un algorithme permettant de résoudre le problème 2-somme sur une liste triée en complexité seulement linéaire. ◇

Solution. Il suffit de faire partir un indice de chaque extrémité de la liste, d'augmenter le plus petit si la somme est inférieure à l'objectif, et de diminuer l'autre sinon.

procédure SORTED2SUM(E, S, N)
 $i \leftarrow 1; j \leftarrow N$
 tant que $i < j$
  faire **si** $E[i] + E[j] = S$ **alors renvoyer** TRUE
  sinon si $E[i] + E[j] > S$ **alors** $j \leftarrow j - 1$
  sinon $i \leftarrow i + 1$
 renvoyer FALSE

La correction de cet algorithme peut se prouver par récurrence sur N. Si $N = 2$, soit la somme des deux éléments vaut S et cela est détecté au premier test, soit il n'y a pas de solution, et alors $i = j$ dès la sortie de la première passe dans la boucle et le résultat FALSE est correct. En admettant la correction pour les listes de taille $N-1$, considérons une liste de taille N. Soit la somme S est égale à $s_1 + s_N$ et cela est détecté au premier test, soit elle est plus grande et en sortant de la première passe dans la boucle on s'est ramené au cas $N-1$, soit elle est plus petite, et on s'est ramené au cas $N-1$ avec tous les indices décalés de 1. Dans les deux cas l'hypothèse de récurrence s'applique.

Pour estimer la complexité, il suffit d'observer que la différence $j - i$ décroît strictement à chaque passage infructueux dans la boucle, dans laquelle on passe donc moins de N fois. Chaque passage a un coût borné, ce qui mène à un coût total en $O(N)$. □

Question 8 En déduire un algorithme de complexité quadratique pour 3-somme (l'entrée n'est pas supposée triée). ◇

Solution. Il suffit de commencer par trier la liste, par exemple en utilisant un tri par insertion, avant d'utiliser l'algorithme de la question précédente. Soit $(s_1, \ldots, s_N)$ l'ensemble trié. Pour chaque élément s_i, on teste s'il peut participer à une somme à S avec deux autres éléments de l'ensemble, qu'il suffit de prendre d'indices plus grands. On teste donc si le problème 2-somme a une solution avec $\{s_{i+1}, \ldots, s_N\}$ et $S - s_i$.

procédure 3SUM(E, S, N)
  SORT(E)
  pour $i \leftarrow 1$ **à** N
    faire si SORTED2SUM$(E[i+1..N], S - E[i], N - i)$ **alors renvoyer** TRUE
  renvoyer FALSE

□

Cet algorithme, bien que très simple, est pratiquement le meilleur connu actuellement. Le problème de savoir si N^2 est l'ordre de grandeur d'une borne inférieure à la complexité de 3-somme est ouvert.

2.3 Diviser pour régner

La conception d'algorithmes efficaces repose souvent sur un principe simple : diviser pour régner. Cette idée fera l'objet du Cours 4. En voici quelques conséquences.

Question 9 Déduire de la question 7 un algorithme de complexité $O(N \log N)$ pour 2-somme. (On ne demande pas le pseudo-code.) ◇

Solution. Le tri par insertion ne suffit pas, mais le tri fusion, à base de diviser-pour-régner, fournit une solution dont la complexité est en $O(N \log N)$. Une fois ce tri effectué, l'algorithme 2-somme linéaire de la question 7 conclut, et le coût total est bien en $O(N \log N)$. □

Cette complexité $O(N \log N)$ est également une borne inférieure pour ce problème. Notre algorithme est donc optimal !

Question 10 (Plus difficile). Proposer un algorithme de complexité $O(N^2 \log N)$ pour 4-somme. ◇

Solution. Le principe est le suivant :

1. on construit tous les $N(N-1)/2$ triplets $(s_i + s_j, i, j)$ avec $1 \leq i < j \leq N$ en $O(N^2)$ opérations, en mémorisant quels couples (i, j) donnent la valeur $s_i + s_j$ par un tableau auxiliaire ;
2. on trie cette liste en $O(N^2 \log N^2) = O(N^2 \log N)$ opérations ;
3. on effectue alors une variante de SORTED2SUM sur cette liste, en un total de $O(N^2)$ opérations. Dans cette phase, lorsque deux éléments se somment à S, on vérifie que les couples correspondants ont des indices disjoints.

Le pseudo-code suivant précise ces idées :

procédure 4SUM(E, S, N)
$k = 0$;
pour $i \leftarrow 1$ **à** N
faire { **pour** $j \leftarrow i+1$ **à** N **faire** { $k \leftarrow k+1$; $A[k] \leftarrow (E[i] + E[j], i, j)$; }
SORT(A);
$i \leftarrow 1; j \leftarrow k$;
tant que $i < j$
faire { **si** $A[i,1] + A[j,1] = S$ **alors** { **pour** $m \leftarrow 0$ **tant que** $A[i+m,1] = A[i,1]$ **faire si** $\{A[j,2], A[j,3]\} \cap \{A[i+m,2], A[i+m,3]\} = \emptyset$ **alors renvoyer** TRUE; $j \leftarrow j-1$ }
sinon si $A[i,1] + A[j,1] > S$ **alors** $j \leftarrow j-1$
sinon $i \leftarrow i+1$ }
renvoyer FALSE

Lorsque deux 2-sommes Σ_1 et Σ_2 se somment à S, pour vérifier qu'elles forment une 4-somme, on parcourt toutes les paires de couples se sommant à Σ_1 et Σ_2. La correction est donc claire. La complexité de cette phase reste quadratique parce que le nombre d'itérations dans la boucle intérieure (où m est incrémenté) est borné par 3 : au plus deux des couples se sommant à Σ_1 peuvent avoir pour sommant l'un de $A[j,2]$, $A[j,3]$ (les s_i initiaux sont supposés distincts, ou peuvent être rendus distincts en $O(N \log N)$ opérations). □

La même idée montre que l'on peut obtenir un algorithme résolvant k-somme lorsque k est pair en complexité $O(N^{k/2} \log N)$, et une variante en $O(N^{(k+1)/2})$ lorsque k est impair. On est loin de la méthode naïve de la question 6 !

Références

[1] J. Kleinberg et E. Tardös. *Algorithm Design*. Pearson, 2006.

[2] D. E. Knuth. *Mariages stables*. Presses de l'université de Montréal, 1976.

[3] A. E. Roth et J. H. Vande Vate. Random paths to stability in two-sided matching. *Econometrica*, 1990.

[4] Anka Gajentaan et Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational Geometry. Theory and Applications*, 5(3):165–185, 1995. ISSN 0925-7721.

[5] Nir Ailon et Bernard Chazelle. Lower bounds for linear degeneracy testing. *Journal of the ACM*, 52(2):157–171, 2005. ISSN 0004-5411.

[6] Jeff Erickson. Lower bounds for linear satisfiability problems. *Chicago Journal of Theoretical Computer Science*, pages Article 8, 28 pp. (electronic), 1999. ISSN 1073-0486.