4I505 CPA – Examen Mai 2016

Les documents distribués dans le cadre du cours et les notes personnelles sont autorisés.

1 Question de cours (4 points)

1.1 Question 1

Pour chaque question suivante donnez une preuve formelle justifiant votre réponse.

- Quelle est la complexité de l'algorithme QuickHull calculant l'enveloppe convexe d'un ensemble de points?
- Peut on concevoir un algorithme calculant le contour positif de l'enveloppe convexe d'un ensemble de n points ayant une complexité temporelle en $O(n)$?

1.2 Question 2

Rappeler les étapes principales d'une approche d'heuristique à base de *local searching* vue en cours. Il faut notamment donner le pseudo-code correspondant.

2 Exercice (4 points)

Pour tout graphe G, on dénote $V(G)$ l'ensemble des sommets et $E(G)$ l'ensemble des arêtes de G. Par abus de notation, on dénote uv $(= vu)$ l'arête entre une paire de sommets distincts $u \in V(G)$, $v \in V(G)$, et $u \neq v$. Une coupe $S \subseteq V(G)$ dans un graphe G est un sous-ensemble de sommets de G vérifiant $S \neq \emptyset$ et $S \neq V(G)$. La valeur de la coupe S, notée $\texttt{score}(S)$, est le nombre d'arêtes ayant une extrémité à l'intérieur de S et une extrémité à l'extérieur de S:

$$\texttt{score}(S) = |\{uv \in E(G) : u \in S \wedge v \notin S\}|.$$

Dans la Figure 1, la valeur de la coupe $\{\texttt{b}, \texttt{a}, \texttt{d}\}$ du graphe est 9. On appelle coupe maximum toute coupe du graphe ayant une valeur maximum.

2.1 Echauffement

- Peut on enlever un (et un seul) sommet de la coupe $\{\texttt{a}, \texttt{g}, \texttt{i}, \texttt{l}, \texttt{e}\}$ afin d'augmenter la valeur de la coupe après l'enlèvement?
- Peut on ajouter un (et un seul) sommet dans la coupe $\{\texttt{a}, \texttt{g}, \texttt{i}, \texttt{l}, \texttt{e}\}$ afin d'augmenter la valeur de la coupe après l'ajout?
- Mêmes questions avec la coupe $\{\texttt{f}, \texttt{i}, \texttt{c}, \texttt{h}, \texttt{e}\}$.

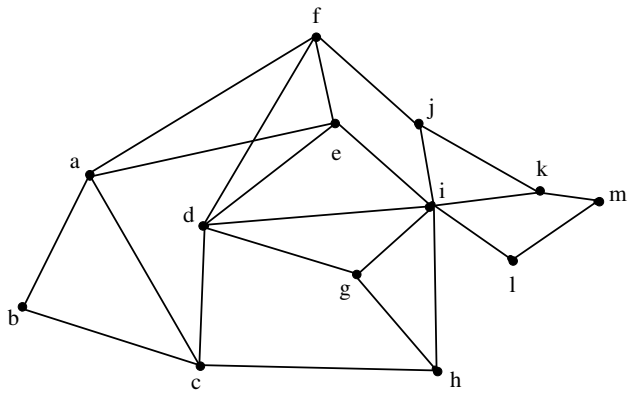

Figure 1: Un exemple de graphe.

2.2 Coupe maximum

Soient maintenant un graphe G, et une coupe $S \subseteq V(G)$ telle que $\texttt{score}(S \setminus \{v\}) \leq \texttt{score}(S)$ pour tout $v \in S$ et telle que $\texttt{score}(S \cup \{v\}) \leq \texttt{score}(S)$ pour tout $v \notin S$. On appelle degré d'un sommet le nombre de ses voisins:

$$\texttt{degree}(v) = |\{uv \in E(G) : u \in V(G)\}| \text{ pour tout } v \in V(G).$$

- Soit $s \in S$ et $t \notin S$ deux sommets (distincts) de G. Montrer que $|\{sv \in E(G) : v \notin S\}| \geq \frac{1}{2}\texttt{degree}(s)$ et que $|\{vt \in E(G) : v \in S\}| \geq \frac{1}{2}\texttt{degree}(t)$.
- Utiliser les résultats précédents pour montrer que $\texttt{score}(S) \geq \frac{m}{2}$, où $m = |E(G)|$ et S est la coupe définie dans la question précédente.
- On appelle *maxCut* la valeur maximum d'une coupe de G. Montrer que

  $$maxCut \geq \texttt{score}(S) \geq \frac{1}{2} \times maxCut,$$

  où S est la coupe définie dans les deux questions précédentes.

3 Exercice bonus (1 point)

A tout problème d'optimisation on appelle 2−approximation un algorithme calculant une solution ayant une valeur entre la valeur macimum et la moitié de cette valeur. On appelle CoupeMax le problème d'optimisation suivant:

Problème CoupeMax:

Input: un graphe G.
Output: une coupe de maximum valeur dans G.

Utiliser les résultats précédents pour donner une 2−approximation de CoupeMax, basée sur une approche de *local searching*.

4 Problème (12 points)

4.1 Famille de fonctions de hashage 2-universelle

Soit $\mathcal{H}_k$ une famille de fonctions de hashage de $[M]$ vers $[M]$, où M est 2^k pour un entier k, et $[M]$ représente l'ensemble $\{0, 1, \ldots, M-1\}$. $\mathcal{H}_k$ est constituée de toutes les fonctions h de la forme

$$h(x) = x \oplus r,$$

où r est un entier de $[M]$ est $\oplus$ est le où exclusif bit par bit. Par exemple pour $k = 2$, $\mathcal{H}_k$ est composé des 2^k fonctions suivantes

$$x \mapsto x$$
$$x \mapsto x \oplus 1$$
$$x \mapsto x \oplus 2$$
$$x \mapsto x \oplus 3.$$

La première fonction est l'identité, la deuxième change le bit de poids faible, la troisième le bit de poids fort, et la dernière inverse tous les bits.

Est-ce que cette famille est 2-universelle ?

4.2 Échantillonner dans k-means++

Considérons un flux σ de n points distincts dans $\mathbb{R}^2$. La distance euclidienne est notée par d et définie comme

$$d((x_1, y_2), (x_2, y_2)) := \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

Par extension $d(a, S)$ pour un point a et ensemble de points S est définie comme $d(a, S) := \min_{b \in S} d(a, b)$.

On veut constituer un ensemble de k points du flux de la manière suivante.

Le premier point p_1 est choisi uniformément au hasard de σ. Puis pour chaque $i = 2, \ldots, k$ on choisit comme p_i un point $a \in \sigma$ avec probabilité

$$\frac{d(a, \{p_1, \ldots, p_{i-1}\})^2}{\sum_{b \in \sigma} d(b, \{p_1, \ldots, p_{i-1}\})^2}.$$

Décrivez un algorithme qui réalise cette tache en k lectures du flux et prouvez qu'il produit la distribution probabiliste demandée.

4.3 Deux éléments manquants

On vous donne un flot de $n-2$ entiers distincts tous entre 0 et $n-1$. Il y a donc exactement deux entiers $p, q \in [n]$ qui ne sont pas présents dans le flot. Concevez un algorithme de flux qui détermine les deux entiers manquants. Quel est sa complexité en mémoire ? Si vous n'avez pas d'idée, commencez par trouver une solution pour le cas $n = 4$ puis $n = 8$. En général l'entier n peut ne pas être une puissance de 2.

A Corrections

A.1 Question de cours

- La complexité en temps de QuickHull est $\Theta(n^2)$ où n est le nombre de points donnés en entrée de l'algorithme. L'algorithme est en $O(n^2)$ parce que l'enlèvement des points appartenant à un triangle est borné par n, le nombre total des points; en même temps, il y a au plus n triangles à considérer. Plus important, l'algorithme est en $\Theta(n^2)$ quand les points donnés en entrée à l'algorithme sont distribués sur un cercle: à chaque test d'appartenance à un nouveau triangle il faut reparcourir tous les points en enlevant aucun.

- On ne peut pas avoir un algorithme donnant le contour positif de l'enveloppe convexe d'un ensemble de point en $O(n)$ car un tel algorithme résolverait Tri en la même complexité: pour trier n nombres a_1, a_2, ... a_n il suffirait alors de les distribuer sur le cercle unitaire avec les angles proportionnels aux valeurs des a_i's, puis, en suivant le contour positif de l'enveloppe convexe de ces points on peut retourner la liste triée des a_i's.

A.2 Exercice

Echauffement:

- $\{a, g, i, l, e\}$: enlever: non; ajouter: oui, exemple c.

- $\{f, i, c, h, e\}$: enlever: oui, exemple h; ajouter: oui, exemple m.

Coupe max:
Soit $s \in S$. On sait que $score(S \setminus \{s\}) \leq score(S)$. Donc par définition:

$$|\{uv \in E(G) : u \in S \setminus \{s\} \wedge v \notin S \setminus \{s\}\}| \leq |\{uv \in E(G) : u \in S \wedge v \notin S\}|$$

$$|\{uv \in E(G) : u \in S \setminus \{s\} \wedge (v \notin S \vee v = s)\}| \leq |\{uv \in E(G) : (u \in S \setminus \{s\} \vee u = s) \wedge v \notin S\}|$$

$$|\{uv \in E(G) : u \in S \setminus \{s\} \wedge v \notin S\}| + |\{us \in E(G) : u \in S \setminus \{s\}\}| \leq$$

$$|\{uv \in E(G) : u \in S \setminus \{s\} \wedge v \notin S\}| + |\{sv \in E(G) : v \notin S\}|$$

$$|\{us \in E(G) : u \in S \setminus \{s\}\}| \leq |\{sv \in E(G) : v \notin S\}|$$

$$|\{us \in E(G) : u \in S \setminus \{s\}\}| + |\{sv \in E(G) : v \notin S\}| \leq 2 \times |\{sv \in E(G) : v \notin S\}|$$

$$\frac{1}{2} degree(s) \leq |\{sv \in E(G) : v \notin S\}|$$

Pareil pour $t \notin S$.
Maintenant en sommant pour tout $s \in S$ et pour tout $t \notin S$:

$$\Sigma_{s \in S} \frac{1}{2} degree(s) + \Sigma_{t \notin S} \frac{1}{2} degree(t) \leq \Sigma_{s \in S} |\{sv \in E(G) : v \notin S\}| + \Sigma_{t \notin S} |\{vt \in E(G) : v \in S\}|$$

$$\Sigma_{x\in V(G)}\frac{1}{2}degree(x) \leq score(S) + score(S)$$

$$\frac{1}{2}\Sigma_{x\in V(G)}degree(x) \leq 2 \times score(S)$$

$$\frac{1}{2}(2 \times m) \leq 2 \times score(S)$$

$$\frac{1}{2}m \leq score(S)$$

Finalement, S étant une coupe de G, donc $score(S) \leq maxCut$. D'autre part, on remarque que $maxCut \leq m$, on déduit donc $\frac{1}{2}maxCut \leq \frac{1}{2}m$, et par l'inégalité ci-dessus, $\frac{1}{2}maxCut \leq score(S)$.

A.3 Exercice bonus

- Initialiser $S \leftarrow \{v\}$ où v est un sommet quelconque de G.
- Répéter tant que une des règles ci-dessous s'applique.
  - Règle 1: si ajouter un sommet $t \notin S$ à S augmente strictement $score(S)$, ajouter t à S.
  - Règle 2: si enlever un sommet $s \in S$ de S augmente strictement $score(S)$, enlever s de S.

Par le résultat de l'exercice précédent, cet algorithme (de *local searching*) retourne une 2-approximation au problème de CoupeMax. Le fait que l'algorithme termine est une trivialité (le *score* augemente strictement à chaque tour de boucle; sa valeur est bornée par m).

A.4 Famille de fonctions de hashage 2-universelle

Non. La famille n'est pas assez large. En effet pour une fonction $h \in \mathcal{H}_k$ on a $h(x) = y$ et $h(x') = y'$ si et seulement si $x \oplus x' = y \oplus y'$. Donc pour $x' \neq x$ la probabilité $\mathbb{P}_h[h(x) = y \wedge h(x') = y']$ est 0 si $x \oplus x' \neq y \oplus y'$ et 1 sinon. Mais pour être 2-universelle cette probabilité devrait être $1/M^2$ dans tous les cas.

Par contre la famille est 1-universelle, chaque valeur y a la même probabilité $1/2^k$ d'être l'image $h(x)$ pour $h \in \mathcal{H}_k$. Pour s'en convaincre il suffit de le montrer pour $k = 1$, et d'observer que h agit sur chaque bit de manière indépendante.

A.5 Échantillonner dans k-means++

Pour le premier point on maintient un compteur r et on garde le r-ème point avec probabilité $1/r$. La distribution du choix est uniforme ce qu'on peut prouver par induction sur r.

Pour le i-ème point, on utilise la même technique, mais pondérée. On maintient un compteur D de toutes les distances entre $\mathcal{P} = \{p_1, \ldots, p_{i-1}\}$ et les points observés. Alors

lors du traitement du r-ème point a on ajoute $d(a, \mathcal{P})$ à D et on sélectionne $p_i = a$ avec probabilité $d(a, \mathcal{P})/D$.

Comme les points sont distincts, D ne sera pas zéro, et le premier point est sélectionné avec probabilité 1. Puis il faut montrer qu'à tout moment la probabilité qu'un point b soit p_i est $d(b, \mathcal{P})/D$. C'est le cas par l'algorithme au moment du traitement de b. Puis plus tard quand un point a est traité, D sera augmenté à $D' = D + d(a, \mathcal{P})$, et a sera choisi avec probabilité $d(a, \mathcal{P})/D'$ tandis que b restera le choix de p_i avec probabilité $d(b, \mathcal{P})/D \cdot D/D' = d(b, \mathcal{P})/D'$.

A.6 Deux éléments manquants

Essayons d'abord d'appliquer la méthode pour trouver un unique élément manquant. On fait la somme S de tous les éléments du flux et on pose $x = n(n-1)/2 - S$. On a alors $x = p + q$, où p, q sont les deux éléments manquants. Il nous manque donc une information, car plusieurs couples d'éléments manquants ont la même somme. En d'autres mots il n'est pas possible de résoudre un système d'équation à une seule équation et deux inconnus.

Algorithme à deux passes et mémoire $O(\log n)$ Certains étudiants ont alors eu l'idée suivante. Si on suppose $p < q$, alors on sait que $p < x/2 < q$. Il suffit alors de restreindre le flux aux éléments inférieur à $x/2$ et dans une deuxième passe de déterminer l'unique élément p manquant dans ce flux restreint.

Algorithme à une passe et mémoire $O(\log^2 n)$ Pour trouver un algorithme à une seule passe, il faudrait donc extraire plusieurs informations du flux. La complexité en mémoire de cet algorithme est $O(\log^2 n)$ bits, correspondant à $2 \log n$ compteurs. Dans x_i^b on cumulera tous les entiers y du flux dont le i-ème bit dans l'expansion binaire est b. Soit z_i^b la somme de tous les entiers $y \in [n]$ dont le i-ème bit dans l'expansion binaire est b. Alors $x_i^b \leq z_i^b$ et il y a égalité si à la fois p et q ont dont le i-ème bit dans l'expansion binaire est $1 - b$. Comme $p \neq q$ il existe forcément une position i où les expansions binaires de p et q diffèrent. Pour cette position nous avons les inégalités strictes $x_i^0 < z_i^0$ et $x_i^1 < z_i^1$. Et parmi les différences $z_i^0 - x_i^0$ et $z_i^1 - x_i^1$ une est forcément p et l'autre q.

Algorithme à une passe et mémoire $O(\log n)$ On calcule la somme de toutes les valeurs du flux et la somme de tous les carrées des valeurs du flux. Ainsi on a un système d'équation à résoudre de la forme $x = p + q, y = p^2 + q^2$, pour deux valeurs x, y ainsi

extraite du flux. En posant $q = x - p$ on trouve

$$\begin{aligned} p^2 + (x-p)^2 &= y \\ p^2 + x^2 - 2xp + p^2 &= y \\ 2p^2 - 2xp + x^2 - y &= 0 \\ p &= \frac{2x \pm \sqrt{4x^2 - 8(x^2 - y)}}{4} \\ p &= \frac{x - \sqrt{2y - x^2}}{2} \\ q &= x - p. \end{aligned}$$