Algorithmique I
Année 2007-2008

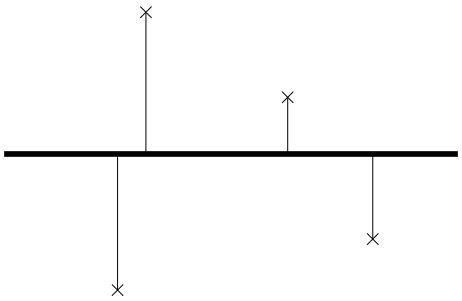
A. Benoit, B. Depardon
JF. Pineau, C. Rezvoy

TD n°7

1 Du pétrole et des idées

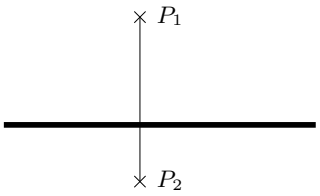
Le professeur Robert Ive est consultant pour une compagnie pétrolière qui projette de construire un grand oléoduc en ligne droite d'Est en Ouest à travers un champ pétrolier. À partir de chaque puits un raccordement doit être connecté directement sur l'oléoduc principal suivant le plus court chemin. Connaissant les coordonnées des puits, comment le professeur trouvera-t'il l'emplacement idéal de l'oléoduc principal qui minimise la longueur totale des raccordements ?

Solution :



Idée simplificatrice : projeter tous les puits sur l'axe Nord-Sud.

Avec deux points seulement, la longueur totale des raccordements est minimale si l'oléoduc passe entre les puits, ou même sur l'un des puits :



Soit y_k l'ordonnée de P_k. L'altitude optimale de l'oléoduc est le médian est y_k. La chose est supposée assez évidente pour se dispenser de la prouver.

Si le nombre n de puits est pair, on a le choix entre deux médians : inférieur ou supérieur.

Si n est impair, l'oléoduc passe par le puit médian. □

2 Médian pondéré

Soient n éléments distincts $x_1, ..., x_n$ de poids positifs $p_1, ..., p_n$ tels que

$$\sum_{i=1}^{n} p_i = 1$$

Le *médian pondéré (inférieur)* est l'élément x_k satisfaisant :

$$\sum_{x_i<x_k} p_i < \frac{1}{2} \quad \text{et} \quad \sum_{x_i>x_k} p_i \leq \frac{1}{2}$$

Question 2.1 Démontrer que le médian de $x_1, ..., x_n$ est le médian pondéré des x_i affectés des poids $p_i = 1/n$ pour i compris entre 1 et n.

Solution : *On cherche à calculer le médian des x_k.*

$\sum_{x_i<x_k} p_i < \frac{1}{2}$ *et* $\sum_{x_i>x_k} p_i \leq \frac{1}{2}$
$\Longrightarrow$ $\frac{k-1}{n} < \frac{1}{2}$ *et* $\frac{n-k}{n} \leq \frac{1}{2}$
$\Longrightarrow$ $k - 1 < \frac{n}{2}$ *et* $n - k \leq \frac{n}{2}$
$\Longrightarrow$ $\frac{n}{2} \leq k < \frac{n}{2} + 1$
$\Longrightarrow$ $k = \lfloor \frac{n}{2} \rfloor$ *c'est le médian inférieur.* □

Question 2.2 Comment calculer *simplement* le médian pondéré de n éléments. Donner la complexité en comparaisons et en additions.

Solution : *Algorithme simple de calcul du médian pondéré*

Algorithme 1 Calcul du médian pondéré

Trier les éléments : $x_{\sigma(1)} < \cdots < x_{\sigma(n)}$
Sommer les poids $p_{\sigma(k)}$ jusqu'à avoir : $p_{\sigma(1)} + \cdots + p_{\sigma(k+1)} \geq \frac{1}{2}$
k est alors le médian pondéré

Complexité :
$(n \log n)$ *comparaisons à cause du tri*
(n) *additions dans la sommation* □

Question 2.3 Comment calculer le médian pondéré en temps linéaire dans le pire des cas.

Solution : *Algorithme en temps linéaire de calcul du médian pondéré*

Algorithme 2 Calcul du médian pondéré, $O(n)$

Poser $E = \{ (x_i, p_i) \}$
si $|E| = 1$ **alors**
  retourner x_1
Calculer le médian x des x_i, noter p son poids
Construire $E_1 = \{ (x_i, p_i) \ / \ x_i < x \}$
Construire $E_2 = \{ (x_i, p_i) \ / \ x_i > x \}$
Calculer $S_1 = \sum_{E_1} p_i$
si $S_1 > \frac{1}{2}$ **alors**
  relancer l'algorithme sur E_1
sinon
  relancer l'algorithme sur $E_2 \cup \{ (x, p + S_1) \}$
k est alors le médian pondéré

Lorsque l'algorithme est relancé, il faut réajuster tous les poids pour que leur somme fasse toujours 1.

Calcul de $I(n)$, complexité d'une itération de l'algorithme, sans récursivité :
- *Le calcul du médian non pondéré s'effectue en (n)*
  (cf. cours : algorithme faisant des paquets de cinq).
- *Les autres opérations s'effectuent en (n) également.*

$\Longrightarrow$ $I(n) = (n)$

Complexité totale : $T(n) = T(\frac{n}{2}) + I(n) = T(\frac{n}{2}) + (n)$

On peut alors appliquer le Master Theorem :

Notations : $T(n) = a\,T(\frac{n}{b}) + (n^\alpha)$
Valeurs : $a = 1 \ ; \ b = 2 \ ; \ \alpha = 1$
Cas : $1 = a < b^\alpha = 2$
Conclusion : $T(n) = (n^\alpha) = (n)$

$T(n) = (n)$ et on a donc un algorithme linéaire pour calculer le médian pondéré. □

3 Le problème de l'emplacement du bureau de poste

Considérons un espace métrique (E, d). Soient n éléments distincts $x_1, ..., x_n$ de poids respectifs $p_1, ..., p_n$ tels que

$$\forall i, \ p_i \geq 0 \qquad \text{et} \qquad \sum_{i=1}^{n} p_i = 1$$

On souhaite trouver un point x (qui n'est pas nécessairement un des x_i) qui minimise $\sum_{i=1}^{n} p_i d(x, x_i)$.

Question 3.1 Cas de la dimension 1 : (E, d) est l'ensemble des réels muni de la distance usuelle. Montrer que le médian pondéré est une des solutions optimales.

Solution : *On suppose les x_i triés tels que $x_1 < \cdots < x_n$.*

Il est évident que $x \in [\, x_1 \,;\, x_n \,]$.
On note x_m *le médian pondéré des* (x_i, p_i).

Soit $x \in [\, x_i \,;\, x_{i+1} \,]$ *et* $\epsilon > 0$ *avec* $x + \epsilon \in [\, x_i \,;\, x_{i+1} \,]$.
La situation est la suivante : $\quad x_i \leq x < x + \epsilon \leq x_{i+1}$

$S(x) = \sum_{k=1}^{n} p_i \, |x - x_i|$
La définition de x *impose que* $S(x)$ *soit minimal.*

$$S(x) = \sum_{x_k < x} p_k \, (x - x_k) + \sum_{x < x_k} p_k \, (x_k - x)$$

$$\begin{aligned} S(x+\epsilon) &= \sum_{x_k \leq x_i} p_k \, (x + \epsilon - x_k) + \sum_{x_k > x_i} p_k \, (x_k - x - \epsilon) \\ &= S(x) + \epsilon \left(\sum_{x_k \leq x_i} p_k - \sum_{x_k > x_i} p_k \right) \\ &= S(x) + \epsilon \left(\sum_{x_k \leq x_i} p_k + \sum_{x_k \leq x_i} p_k - 1 \right) \qquad \textit{car :} \quad \sum_{k=1}^{n} p_k = 1 \end{aligned}$$

$$S(x+\epsilon) = S(x) + \underbrace{\epsilon}_{>0} \left(2 \sum_{x_k \leq x_i} p_k - 1 \right)$$

$$S(x+\epsilon) < S(x) \iff \sum_{x_k < x_{i+1}} p_k < \tfrac{1}{2}$$

Si $x < x_{i+1} < x_m$ *alors* $\sum_{x_k < x_{i+1}} p_k < \sum_{x_k < x_m} p_k < \frac{1}{2} \implies S(x+\epsilon) < S(x)$
$\implies$ *$S(x)$ n'est pas minimal donc x n'est pas un médian pondéré.*

On en déduit : $x \geq x_m$.

En posant $\epsilon < 0$, *on montre avec un raisonnement similaire que* $x \leq x_m$, *ce qui permet de conclure :* $x = x_m$. *Corrolaire : l'exercice 2 n'a pas servi à rien !* □

Question 3.2 Cas de la dimension 2 : (E, d) est le plan muni de la distance 1, ou *distance Manhattan*, c'est à dire :

$$d((x_1, y_1), (x_2, y_2)) = |x_1 - x_2| + |y_1 - y_2|$$

Résoudre l'énoncé qui s'appelle dans ce cas «problème du bureau de poste».

Solution : *On calcule indépendamment* x_k *médian sur les* x, *et* y_l *médian sur les* y.

La solution au problème est le point (x_k, y_l).

Note 1 : ne pas commettre l'erreur de se limiter au calcul d'un seul médian et de déclarer solution le point (x_k, y_k). *En dimension 2, le point solution n'est pas toujours confondable avec un des* P_k *comme c'était le cas dans la question précédente.*

Note 2 : cette solution simple convient car on utilise la distance de Manhattan. Elle ne fonctionnerait pas avec la distance usuelle $d(A, B) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$. □

4 Tri par tas

Le tri par tas est réalisé en deux étapes : insertion des n éléments dans le tas, puis déconstruction du tas en extrayant à chaque fois le plus petit élément.

La construction du tas se fait en $O(n \log n)$, et la suppression des éléments du tas se fait également en $O(n \log n)$.

Il est possible d'améliorer la complexité de la construction du tas, elle peut se faire en $O(n)$ (cependant la suppression des n éléments du tas reste en $O(n \log n)$). Proposez un algorithme pour construire le tas en temps linéaire.

Solution : *Dans un tas de taille n, les éléments situés aux positions de $\lfloor n/2 \rfloor$ à n sont des feuilles de l'arbre. On peut donc construire le tas en partant du niveau le plus bas et en comparant les feuilles avec leur père, et en vérifiant que la relation d'ordre partiel est bien vérifiée. Cela revient à ordonner les sous-arbres dont les racines sont aux positions $\lfloor n/2 \rfloor, \lfloor n/2 \rfloor - 1, \ldots, 1$ (exemple figure 1, pour le tableau :* 15 12 11 7 13 9*).*

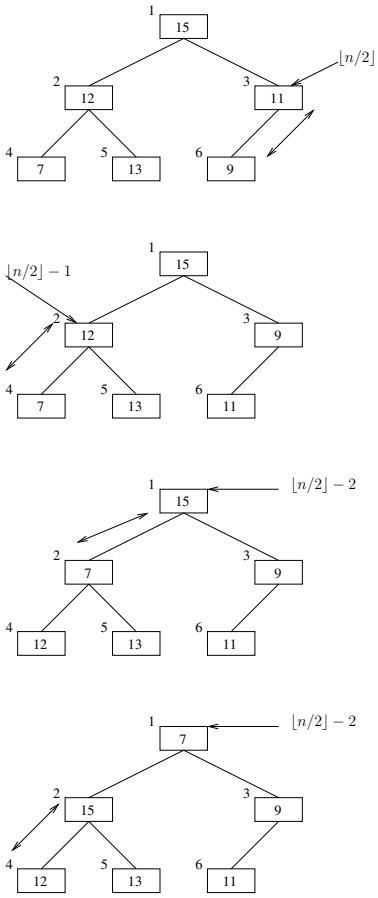


FIG. 1 – Exemple construction tas en temps linéaire

Pour ordonner un sous-arbre, il se peut que l'on doivent descendre jusqu'à une feuille pour faire des permutations. L'algorithme suivant traite le cas s'un sous-arbre dont la racine est $t[i]$.

Algorithme 3 ordonner(t, i) : ordonne le tas pour le sous-arbre sous la racine i

On suppose que les sous-arbre fils de $t[i]$ ont déjà été traités
si $2i+1 > n$ ou $t[2i] \leq t[2i+1]$ **alors** // recherche de l'indice du plus petit des deux fils
  $j \leftarrow 2i$
sinon
  $j \leftarrow 2i+1$
si $t[j] < t[i]$ **alors** // échange, si nécessaire, avec ce fils
  $t[j] \leftrightarrow t[i]$
  si $j \leq n/2$ **alors** // si t_j n'est pas une feuille, il faut réordonner le sous-arbre de t_j
    ordonner(t, j)

L'algorithme de tri par tas est donc modifié de la manière suivante :

Algorithme 4 tri par tas(t)

pour p de $\lfloor n/2 \rfloor$ à 1 **faire**
  ordonner(t, p)
pour p de 1 à n **faire**
  $t[p] \leftarrow$ suppression_min(t)

Complexité linéaire : *la construction du tas fait $\lfloor n/2 \rfloor$ appels à la procédure ordonner. À chaque appel on fait deux comparaisons d'éléments, avec potentiellement un échange, et également un appel récursif à ordonner(t, j), avec j valant $2p$ ou $2p+1$.*

Notre complexité (en nombre de comparaisons ou d'échanges) est dépendante du nombre d'appels de la procédure ordonner :

- *si $\lfloor n/4 \rfloor + 1 \leq p \leq \lfloor n/2 \rfloor$, alors on n'a pas d'appel récursif, car $j > \lfloor n/2 \rfloor$ (c'est une feuille) : 1 appel en tout*
- *si $\lfloor n/8 \rfloor + 1 \leq p \leq \lfloor n/4 \rfloor$, alors on a au plus un appel récursif : 2 appels en tout*

Plus généralement, quand $\lfloor n/2^{i+1} \rfloor + 1 \leq p \leq \lfloor n/2^i \rfloor$, on a en tout i appels au plus : un appel pour p, et pour les suivants j est multiplié par 2 à chaque fois, et on arrête quand $j > \lfloor n/2 \rfloor$. On finit avec un intervalle réduit à $\lfloor n/2^i \rfloor = 1$, où k est tel que $2^k \leq n < 2^{k+1}$.

Si on considère un intervalle $I = [\lfloor n/2^{i+1} \rfloor + 1, \lfloor n/2^i \rfloor]$, alors pour chaque élément de cet intervalle on aura i appels. On peut majorer le nombre d'éléments dans cet intervalle :

$$\lfloor n/2^i \rfloor - (\lfloor n/2^{i+1} \rfloor + 1) + 1 = \lfloor n/2^i \rfloor - \lfloor n/2^{i+1} \rfloor \leq n/2^i$$

Donc le nombre d'appels à la procédure peut être majoré par :

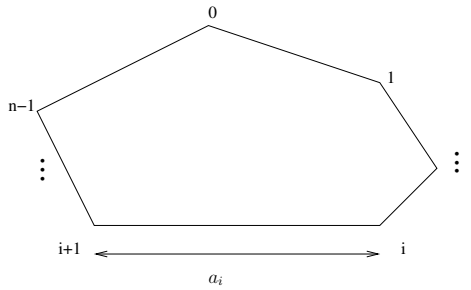
$$\sum_{i=1}^{k} i \cdot \frac{n}{2^i} \leq \frac{n}{2} \sum_{i \geq 1} \frac{i}{2^{i-1}}$$

Or la série $\sum_{i \geq 1} i \cdot x^{i-1}$ est la série dérivée de $\sum_{i \geq 0} x^i = \frac{1}{1-x}$, soit $\frac{1}{(1-x)^2}$.

On a donc le nombre d'appels de la procédure ordonner qui est majoré par $4 \cdot n/2$ ($x = 1/2$), donc une complexité linéaire. □

5 Division du périmètre d'un polygone

On considère un polygone à n sommets, numérotés dans le sens des aiguilles d'une montre de 0 à $n-1$. La suite des longueurs des côtés est $\{a_0, a_2, \ldots, a_{n-1}\}$, comme représenté dans la figure ci-dessous.



Question 5.1 On cherche tout d'abord à déterminer les deux indices i, j qui minimisent la valeur absolue de la différence entre les deux portions de périmètre qu'ils déterminent, i.e., qui minimisent (sommations modulo n) :

$$|(\sum_{l=i}^{j-1} a_l) - (\sum_{l=j}^{i-1} a_l)|$$

5.1.1 Donner un algorithme naïf et calculer sa complexité.

Solution : *Solution naïve en $O(n^3)$: on compare tous les couples (i, j) et on prend le minimum de la différence.*

Algorithme 5 Division du périmètre, $O(n^3)$

$diff_{min} \leftarrow \infty$
pour i de 0 à $n-1$ **faire**
  pour j de $i+1$ à $i-1$ **faire** // incrément modulo n
    $diff \leftarrow |(\sum_{l=i}^{j-1} a_l) - (\sum_{l=j}^{i-1} a_l)|$
    si $diff < diff_{min}$ **alors**
      $diff_{min} \leftarrow diff$
      $i_{min} \leftarrow i$; $j_{min} \leftarrow j$
retourner $(i_{min}, j_{min}, diff_{min})$

Solution un peu améliorée en $O(n^2)$:
- *calculer le périmètre total P,*
- *rechercher pour tout i le meilleur sommet j : cherche le premier sommet j tel que la distance de i à j soit supérieure ou égale au demi-périmètre $P/2$, les seuls candidats pour partager au mieux le polygone à partir de i sont alors j et $j-1$ (modulo n),*
- *prendre le minimum des différences sur tous les sommets i.*

Algorithme 6 Division du périmètre, $O(n^2)$

```
//calcul du périmètre
P ← 0
pour i de 0 à n − 1 faire
   P ← P + a_i
diff_min ← P
// recherche des indices i et j
pour i de 0 à n − 1 faire
   // recherche du "meilleur" j pour le i courant
   j ← i ; chemin ← 0
   tantque 2 · chemin < P faire
      chemin ← chemin + a_j
      j ← (j + 1)[n]
   // soit j soit j − 1
   diff ← |2 · chemin − P|
   si diff < diff_min alors
      i_min ← i, j_min ← j, diff_min ← dif
   pred ← (j − 1)[n]
   chemin ← chemin − a_pred
   diff ← |2 · chemin − P|
   si diff < diff_min alors
      i_min ← i, j_min ← j, diff_min ← dif
retourner (i_min, j_min, diff_min)
```

□

5.1.2 Proposer une solution en temps linéaire.

Solution :

Algorithme 7 Division du périmètre, $O(n)$

```
//calcul du périmètre
P ← 0
pour i de 0 à n − 1 faire
   P ← P + a_i
diff_min ← P
// recherche des indices i et j
pour i de 0 à n − 1 faire
   // recherche du "meilleur" j pour le i courant
   j ← i ; chemin ← 0
   tantque 2 · chemin < P faire
      chemin ← chemin + a_j
      j ← (j + 1)[n]
   // soit j soit j − 1
   diff ← |2 · chemin − P|
   si diff < diff_min alors
      i_min ← i, j_min ← j, diff_min ← diff
   pred ← (j − 1)[n]
   chemin ← chemin − a_pred
   diff ← |2 · chemin − P|
   si diff < diff_min alors
      i_min ← i, j_min ← j, diff_min ← diff
   chemin ← chemin − a_i
retourner (i_min, j_min, diff_min)
```

□

Question 5.2 Trouver en temps linéaire trois indices i, j, k qui minimisent la différence entre le plus grand "tiers" et le plus petit "tiers" qu'ils déterminent ($\max(\sum_{l=i}^{j-1} a_l, \sum_{l=j}^{k-1} a_l, \sum_{l=k}^{i-1} a_l) - min(\sum_{l=i}^{j-1} a_l, \sum_{l=j}^{k-1} a_l, \sum_{l=k}^{i-1} a_l)$). Pouvez vous généraliser pour la découpe en k portions ?

Solution : *On veut minimiser* $\max(\sum_{l=i}^{j-1} a_l, \sum_{l=j}^{k-1} a_l, \sum_{l=k}^{i-1} a_l) - min(\sum_{l=i}^{j-1} a_l, \sum_{l=j}^{k-1} a_l, \sum_{l=k}^{i-1} a_l)$.

□