Nom :

Prénom :

Examen d'algorithmique

EPITA ING1 2014 S1; A. DURET-LUTZ

Durée : 1h30

janvier 2012

Consignes

- Cette épreuve se déroule **sans document** et **sans calculatrice**.
- Répondez sur le sujet dans les cadres prévus à cet effet.
- Soignez votre écriture, et ne donnez pas plus de détails que ceux nécessaires à vous justifier.
- Il y a 7 pages d'énoncé et une page d'annexe.
- Le barème, indicatif, correspond à une note sur 26.

La paire la plus proche (17 points)

On considère un ensemble de points du plan euclidien, numérotés de 0 à $n-1$. Chaque point p est représenté par une paire de nombres flottants (ses coordonnées) et ces n points sont rangés dans un tableau P de taille n. Ainsi $P[i].x$ est l'abscisse du i^{e} point, tandis que $P[i].y$ est son ordonnée.

Notre but est de trouver la plus petite distance existant entre deux points de cet ensemble c'est-à-dire le valeur de $\min_{i \neq j} \sqrt{(P[i].x - P[j].x)^2 + (P[i].y - P[j].y)^2}$.

Voici un algorithme naïf calculant cette plus petite distance en considérant toutes les paires de points :

```
1  MINDIST1(P, n)
2     d ← ∞
3     for i ← 0 to n − 1
4        for j ← 0 to n − 1
5           if i ≠ j then
6              d ← min(d, √((P[i].x − P[j].x)² + (P[i].y − P[j].y)²))
7     return d
```

1. **(1pt)** Combien de fois la ligne 6 est-elle exécutée ? Donnez une réponse précise en fonction de n.

Réponse :

2. **(1pt)** Donnez la complexité de MINDIST1. (En fonction de n toujours.)

Réponse :

3. **(1pt)** Comment pourrait-on très simplement diviser par 2 le nombre d'exécutions de la ligne 6 ?

Réponse :

4. **(1pt)** Comment pourrait-on faire pour que la fonction racine carrée ne soit appelée qu'une seule fois dans tout l'algorithme ?

Réponse :

5. **(1pt)** Quelle est l'influence des deux dernières optimisations sur la classe de complexité de l'algorithme ?

Réponse :

Considérons maintenant un algorithme "diviser pour régner". (A) Tout d'abord le plan est découpé par une ligne verticale en deux régions : la moitié des points à gauche et l'autre moitié à droite de la frontière. (B) On calcule ensuite récursivement la plus petite distance dans chacune de ces deux régions : cela nous donne deux distances d_l (*left*) et d_r (*right*), mais il se peut que nous ayons manqué la plus petite distance si elle est à cheval sur la frontière comme dans la FIG. 1. (C) On compare donc d_l et d_r avec les distances des couples à cheval sur la frontière : une étape délicate à réaliser efficacement.

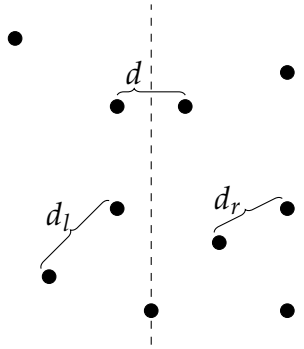


FIGURE 1 – Cas où la plus petite distance est à cheval sur la séparation.

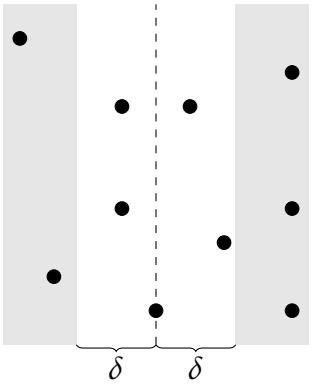


FIGURE 2 – On élimine les points qui sont à une distance supérieure à $\delta = \min(d_l, d_r)$.

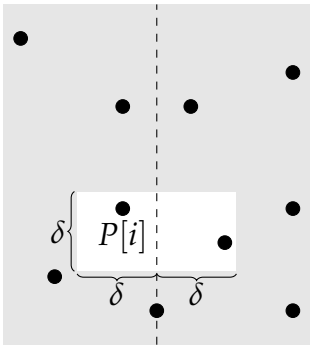


FIGURE 3 – Les points à comparer avec $P[i]$ sont dans un rectangle de taille $(2\delta, \delta)$.

Reprenons ces trois étapes en détail :

(A) Le découpage du tableau P en deux parties est simple si les points on été préalablement triés par abscisses croissantes : il suffit alors de placer la frontière à l'abscisse du point médian : tous les points $P[0]$, $P[1]$, $\ldots P[\lfloor n/2 \rfloor]$ sont à gauche, et tous les points $P[\lfloor n/2 \rfloor + 1], \ldots, P[n-1]$ sont à droite.

(B) Les appels récursifs qui calculeront des distances sur ces deux moitiés de tableaux devront les diviser à leur tour : on va donc devoir passer en paramètre les indices l et r des bornes gauche et droite du sous-tableau considéré, comme ce que nous avons fait dans QUICKSORT en cours. La récursion se termine lorsqu'on demande de calculer la plus petite distance d'un tableau ne contenant qu'un point.

(C) Une fois que l'on connaît d_l et d_r, on sait que la plus petite distance est au plus $\delta = \min(d_l, d_r)$: elle peut être plus petite si deux points de part et d'autre de la frontière sont plus proches. Pour vérifier on pourrait calculer les distances de tous les points de gauche à tous les points de droite, mais ce serait inefficace. En fait, la valeur de δ nous permet déjà d'éliminer tous les points du tableau qui sont à une distance de la frontière supérieure à δ. Nous ne devons donc considérer que les points sur fond blanc de la FIG. 2. Prenons maintenant un point $P[i]$ de cette bande blanche : s'il existe un point tel que sa distance à $P[i]$ est inférieure à δ, alors il existe forcément un rectangle de taille $(2\delta, \delta)$ qui entoure ces deux points et est centré sur la frontière (FIG. 3). Combien de points ce rectangle peut-il contenir ? Le carré de gauche ne peut pas contenir plus de 4 points (un à chaque coin) autrement la distance d_l serait plus petite que ce que nous avons calculée. De même le carré de droite ne peut contenir plus de 4 points.[1] Cela nous fait donc un maximum de 8 points dans le rectangle autour de $P[i]$. Revenons maintenant à la bande blanche de la FIG. 2 : si nous trions tous les points de cette bande blanche par rapport à leurs ordonnées, ils est maintenant clair que pour chaque point nous ne devons que calculer les distances par rapport aux 7 points qui le suivent (dans l'ordre des ordonnées).

Voici l'algorithme correspondant :

```
 1  MINDISTREC(P, l, r)
 2     if r ≤ l then
 3        return □
 4     else
 5        m ← l + ⌊(r − l)/2⌋
 6        d_l ← MINDISTREC(P, l, m)
 7        d_r ← MINDISTREC(P, m + 1, r)
 8        δ ← min(d_l, d_r)
 9        k ← 0
10        for i ← l to r
11           if |P[i].x − P[m].x| < δ then
12              Q[k] ← P[i]
13              k ← k + 1
14        R ← sort Q (array of size k) according to y-coordinates
15        for i ← 0 to k − 1
16           for j ← i + 1 to min(i + 7, k − 1)
17              δ ← min(δ, √((R[i].x − R[j].x)² + (R[i].y − R[j].y)²))
18        return δ
19
20  MINDIST2(P, n)
21     P ← sort P according to x-coordinates
22     return MINDISTREC(P, 0, n − 1)
```

1. Dans ce scénario deux des points de gauche sont superposés avec deux des points de droite, ce qui est possible si le tableau P comporte des doublons.

6. **(0.5pt)** Quel algorithme de tri proposez-vous d'utiliser aux lignes 21 et 14 ?

Réponse :

7. **(1pt)** Quelle est la complexité de la ligne 21 en fonction de n pour l'algorithme de tri que vous avez choisi.

Réponse :

8. **(1pt)** Quelle valeur faut-il retourner ligne 3 ? **Répondez directement sur l'algorithme.**
9. **(1pt)** Si on note $n = r + 1 - l$, quelle valeur maximale peut prendre k (la taille de Q) lorsque la ligne 14 est exécutée ?

Réponse :

10. **(1pt)** En déduire la complexité de la ligne 14.

Réponse :

11. **(1.5pt)** Donnez une définition récursive de la complexité de MINDISTREC en fonction de la taille $n = r + 1 - l$ du tableau manipulé. Pour simplifier, ignorez les parties entières et supposez que les deux appels récursifs lignes 6 et 7 se font sur des tableaux de même taille.

Réponse :

12. **(3pts)** Quelle est la solution de l'équation précédente. (Indice : si le théorème général ne s'applique pas, appliquez la méthode par réécritures successives sans vous décourager—simplifiez les log et cherchez à utiliser la formule des $\sum kx^k$.)

Réponse :

L'algorithme précédent n'est pas optimal à cause du tri ligne 14. Une méthode plus efficace consiste à garder dans un tableau une copie de tous les points "pré-triés" par rapport aux ordonnées, et d'utiliser ce tableau lors du filtrage des points lignes 9–13.

Cela donne l'algorithme suivant :

```
1   MinDistRec'(P, Q, l, r)
2     if r ≤ l then
3       return [ ]
4     else
5       m ← l + ⌊(r − l)/2⌋
6       d_l ← MinDistRec(P, Q, l, m)
7       d_r ← MinDistRec(P, Q, m + 1, r)
8       δ ← min(d_l, d_r)
9       k ← 0
10      for i ← 0 to n − 1
11        if |Q[i].x − P[m].x| < δ then
12          R[k] ← Q[i]
13          k ← k + 1
14      for i ← 0 to k − 1
15        for j ← i + 1 to min(i + 7, k − 1)
16          δ ← min(δ, √((R[i].x − R[j].x)² + (R[i].y − R[j].y)²))
17      return δ
18
19  MinDist3(P, n)
20    P ← sort P according to x-coordinates
21    Q ← sort P according to y-coordinates
22    return MinDistRec'(P, Q, 0, n − 1)
```

13. **(2pts)** La complexité $T(n)$ de l'algorithme MinDistRec' présenté ci-dessus satisfait l'équation $T(n) = 2T(n/2) + \Theta(n)$. Quelle est la solution de cette equation ?

Réponse :

14. **(1pt)** Déduisez-en la complexité de MinDist3.

Réponse :

La plus longue sous-séquence commune (9 points)

Le problème de la PLSSC est celui-ci :

Entrée : deux chaînes, p.ex. `"loutre"` et `"troupe"`.
Sortie : une plus longue sous-séquence commune, `"oue"` ou `"tre"`.

Une *sous-séquence* d'une chaîne S est une suite de lettres de S qui ne sont pas forcément consécutives dans S mais doivent apparaître dans le même ordre. Une sous-séquence peut être vide.

Approche bovine (4.5 points)

1. **(1pt)** Si une chaîne possède n lettres (supposées toutes différentes), combien peut-on construire de sous-séquences différentes ?

Réponse :

2. **(1pt)** Parmi ces sous-séquences combien en existe-t-il de taille k, pour $0 \leq k \leq n$.

Réponse :

3. **(1pt)** Étant donné une chaîne S de taille k, et une chaîne C de taille m, quelle est la complexité de tester si S est une sous-séquence de C ? (Donnez une formule en fonction de k et m.)

Réponse :

4. **(2.5pt)** En déduire la complexité d'un algorithme qui énumère toutes les sous-séquences de la première chaîne (de taille n) puis teste chacune pour voir si c'est une sous-séquence de la seconde chaîne (de taille m). Vous supposerez $n \leq m$.

Réponse :

Programmation Dynamique (3.5 points)

Notons $X = x_1x_2 \cdots x_n$ et $Y = y_1y_2 \cdots y_m$ les chaînes d'entrée, et $Z = z_1z_2 \cdots z_k$ une PLSSC. Notons de plus X_i le i^e préfixe de X, c.-à-d. $X_i = x_1x_2 \cdots x_i$. On a :
- $x_n = y_m \implies z_k = x_n = y_m$ et Z_{k-1} est une PLSSC de X_{n-1} et Y_{m-1}
- $x_n \neq y_m \wedge z_k \neq x_n \implies Z$ est une PLSSC de X_{n-1} et Y
- $x_n \neq y_m \wedge z_k \neq y_m \implies Z$ est une PLSSC de X et Y_{m-1}

Le problème a donc une sous-structure optimale.

Définissons récursivement $C[i,j]$, la longueur de la plus longue sous-séquence commune à X_i et Y_j :

$$C[i,j] = \begin{cases} 0 & \text{si } i = 0 \text{ ou } j = 0 \\ C[i-1,j-1]+1 & \text{si } i,j > 0 \text{ et } x_i = y_i \\ \max(C[i,j-1], C[i-1,j]) & \text{si } i,j > 0 \text{ et } x_i \neq y_i \end{cases}$$

1. **(1.5pt)** Quelle est la complexité d'un algorithme de programmation dynamique qui calculerait $C[n,m]$?

Réponse :

2. **(2pts)** Comment trouver la plus longue sous-séquence commune (et non uniquement sa taille) à partir du tableau C ?

Réponse :

C'est déjà fini...

Notations asymptotiques

$$\mathrm{O}(g(n)) = \{f(n) \mid \exists c \in \mathbb{R}^{+\star},\ \exists n_0 \in \mathbb{N},\ \forall n \geq n_0,\ 0 \leq f(n) \leq cg(n)\}$$
$$\Omega(g(n)) = \{f(n) \mid \exists c \in \mathbb{R}^{+\star},\ \exists n_0 \in \mathbb{N},\ \forall n \geq n_0,\ 0 \leq cg(n) \leq f(n)\}$$
$$\Theta(g(n)) = \{f(n) \mid \exists c_1 \in \mathbb{R}^{+\star},\ \exists c_2 \in \mathbb{R}^{+\star},\ \exists n_0 \in \mathbb{N},\ \forall n \geq n_0,\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \iff g(n) \in \mathrm{O}(f(n)) \text{ et } f(n) \notin \mathrm{O}(g(n)) \qquad f(n) \in \mathrm{O}(g(n)) \iff g(n) \in \Omega(f(n))$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \iff f(n) \in \mathrm{O}(g(n)) \text{ et } g(n) \notin \mathrm{O}(f(n)) \qquad f(n) \in \Theta(g(n)) \iff \begin{cases} f(n) \in \Omega(g(n)) \\ g(n) \in \Omega(f(n)) \end{cases}$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \in \mathbb{R}^{+\star} \iff f(n) \in \Theta(g(n))$$

Ordres de grandeurs

constante	$\Theta(1)$	
logarithmique	$\Theta(\log n)$	
polylogarith.	$\Theta((\log n)^c)$	$c > 1$
	$\Theta(\sqrt{n})$	
linéaire	$\Theta(n)$	
	$\Theta(n \log n)$	
quadratique	$\Theta(n^2)$	
	$\Theta(n^c)$	$c > 2$
exponentielle	$\Theta(c^n)$	$c > 1$
factorielle	$\Theta(n!)$	
	$\Theta(n^n)$	

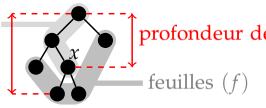
Théorème général

Soit à résoudre $T(n) = aT(n/b) + f(n)$ avec $a \geq 1$, $b > 1$

- Si $f(n) = \mathrm{O}(n^{\log_b a - \varepsilon})$ pour un $\varepsilon > 0$, alors $T(n) = \Theta(n^{\log_b a})$.
- Si $f(n) = \Theta(n^{\log_b a})$, alors $T(n) = \Theta(n^{\log_b a} \log n)$.
- Si $f(n) = \Omega(n^{\log_b a + \varepsilon})$ pour un $\varepsilon > 0$, et si $af(n/b) \leq cf(n)$ pour un $c < 1$ et tous les n suffisamment grands, alors $T(n) = \Theta(f(n))$.

(Note : il est possible de n'être dans aucun de ces trois cas.)

Arbres



Pour tout arbre binaire :

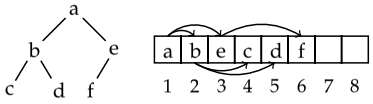
$n \leq 2^{h+1} - 1$ $\qquad$ $h \geq \lceil \log_2(n+1) - 1 \rceil = \lfloor \log_2 n \rfloor$ si $n > 0$

$f \leq 2^h$ $\qquad$ $h \geq \lceil \log_2 f \rceil$ si $f > 0$

$f = ni + 1$ (si l'arbre est *complet* = les nœud internes ont tous 2 fils)

Dans un arbre binaire équilibré une feuille est soit à la profondeur $\lfloor \log_2(n+1) - 1 \rfloor$ soit à la profondeur $\lceil \log_2(n+1) - 1 \rceil = \lfloor \log_2 n \rfloor$. Pour ces arbres $h = \lfloor \log_2 n \rfloor$.

Un arbre parfait (= complet, équilibré, avec toutes les feuilles du dernier niveau à gauche) **étiqueté** peut être représenté par un tableau.



Les indices sont reliés par :

$\text{Père}(y) = \lfloor y/2 \rfloor$

$\text{FilsG}(y) = y \times 2$

$\text{FilsD}(y) = y \times 2 + 1$

Identités utiles

$$\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$$

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1} - 1}{x - 1} \quad \text{si } x \neq 1$$

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x} \quad \text{si } |x| < 1$$

$$\sum_{k=0}^{\infty} k x^k = \frac{x}{(1-x)^2} \quad \text{si } |x| < 1$$

$$\sum_{k=1}^{n} \frac{1}{k} = \Theta(\log n)$$

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

Définitions diverses

La complexité d'un problème est celle de l'algorithme le plus efficace pour le résoudre.

Un tri stable préserve l'ordre relatif de deux éléments égaux (au sens de la relation de comparaison utilisée pour le tri).

Un tri en place utilise une mémoire temporaire de taille constante (indépendante de n).

Rappels de probabilités

Espérance d'une variable aléatoire X : C'est sa valeur attendue, ou moyenne. $\mathrm{E}[X] = \sum_x \Pr\{X = x\}$

Variance : $\mathrm{Var}[X] = \mathrm{E}[(X - \mathrm{E}[X])^2] = \mathrm{E}[X^2] - \mathrm{E}^2[X]$

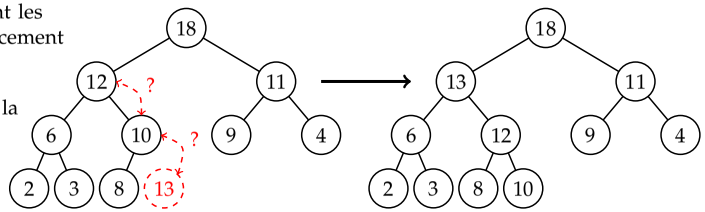
Loi binomiale : On lance n ballons dans r paniers. Les chutes dans les paniers sont équiprobables ($p = 1/r$). On note X_i le nombre de ballons dans le panier i. On a $\Pr\{X_i = k\} = C_n^k p^k (1-p)^{n-k}$. On peut montrer $E[X_i] = np$ et $\mathrm{Var}[X_i] = np(1-p)$.

Tas

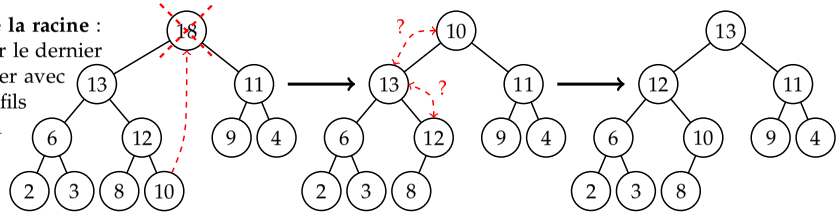
Un tas est un arbre parfait partiellement ordonné : l'étiquette d'un nœud est supérieure à celles de ses fils.

Dans les opérations qui suivent les arbres parfaits sont plus efficacement représentés par des tableaux.

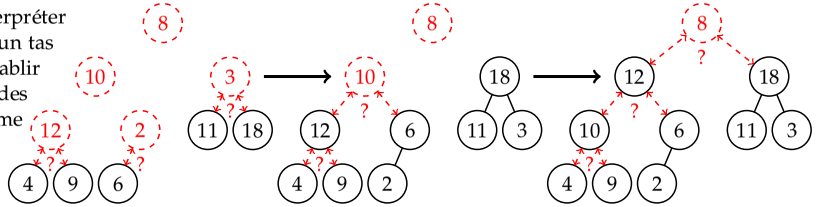
Insertion : ajouter l'élément à la fin du tas, l'échanger avec son père tant qu'il lui est inférieur (en remontant vers la racine). $T_{\text{insert}} = \mathrm{O}(\log n)$



Suppression de la racine : la remplacer par le dernier nœud ; l'échanger avec son plus grand fils tant que celui-ci est plus grand. $T_{\text{rem}} = \mathrm{O}(\log n)$



Construction : interpréter le tableau comme un tas (incorrect) puis rétablir l'ordre en partant des feuilles (vues comme des tas corrects). $T_{\text{build}} = \Theta(n)$



Arbres Rouge et Noir

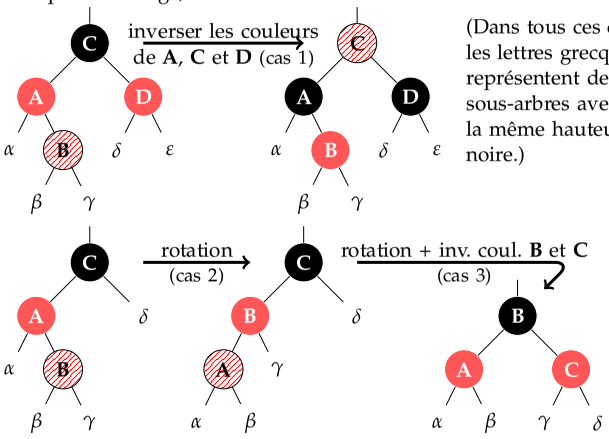
Les ARN sont des arbres binaires de recherche dans lesquels : (1) un nœud est rouge ou noir, (2) racine et feuilles (NIL) sont noires, (3) les fils d'un nœud rouge sont noirs, et (4) tous les chemins reliant un nœud à une feuille (de ses descendants) contiennent le même nombre de nœuds noirs (= la *hauteur noire*). Ces propriétés interdisent un trop fort déséquilibre de l'arbre, sa hauteur reste en $\Theta(\log n)$.

Insertion d'une valeur : insérer le nœud avec la couleur rouge à la position qu'il aurait dans un arbre binaire de recherche classique, puis, si le père est rouge, considérer les trois cas suivants dans l'ordre.

Cas 1 : Le père et l'oncle du nœud considéré sont tous les deux rouges. **Répéter** cette transformation à partir du grand-père si l'arrière grand-père est aussi rouge.

Cas 2 : Si le père est rouge, l'oncle noir, et que le nœud courant n'est pas dans l'axe père–grand-père, une rotation permet d'aligner fils, père, et grand-père.

Cas 3 : Si le père est rouge, l'oncle noir, et que le nœud courant est dans l'axe père–grand-père, un rotation et une inversion de couleurs rétablissent les propriétés des ARN.



(Dans tous ces cas, les lettres grecques représentent des sous-arbres avec la même hauteur noire.)