Correction du devoir maison

AP3 Algorithmique et programmation — L2 mathématiques

30 novembre 2017

1. L'algorithme le plus évident consiste à calculer la distance entre toutes les paires de points possible et à sélectionner le minimum.

```
NAIF(T):
  n ← length of T # assert n>1
  min ← DISTANCE(T[1],T[2])
  foreach i in 1,...,n do
    foreach j in i+1,...,n do
      d ← DISTANCE(T[i],T[j])
      if d < min
      then min ← d
      end if
    end foreach
  end foreach
  return min
```

   Par définition de DISTANCE dans l'énoncé, l'intérieure de la double boucle s'opère en temps constant K. Ainsi l'algorithme admet une complexité de

$$\gamma(n) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} K = K \sum_{i=1}^{n} (n-i) = K \frac{n(n-1)}{2}$$

2. a) L'algorithme du cours a utilisé est ici l'algorithme de la médiane.

   b) Grâce à l'algorithme de la mediane, on peut trouver z en temps linéaire. Pour obtenir la fonction REPARTIR, il suffit alors de créer E_1 et E_2 en temps linéaire, ce qui se fait avec l'algorithme suivant :

```
input: E, z
E₁ ← []
E₂ ← []
foreach x in E do
  if x ≤ z
  then append x at E₁
  else append x at E₂
output: E₁, E₂
```

3. a) Si p et q vérifient $d(p,q) < d_m$, alors ils ne peuvent être tous les deux dans E_1 ni tous les deux dans E_2. Donc l'un est dans E_1 et l'autre est dans E_2. Pour fixer les idées, disons que $p \in E_1$ et $q \in E_2$. Notons x_p et x_q l'abscisse de p et de q respectivement. Ainsi : $x_p \leq z < x_q$.

Comme de plus $d(p,q) < d_m$, on a $x_q - x_p < d_m$. On trouve ainsi :

$$0 \leq z - x_p < d_m$$
$$0 < x_q - z < d_m$$

Au final, les points p et q se trouvent dans la zone :

$$Z(z, d_m) = \{(x, y) \in \mathbb{R}^2 : z - d_m < x < z + d_m\}$$

C'est-à-dire géométriquement dans la bande ouverte du plan délimitée par la droite $x = z - d_m$ à gauche et $x = z + d_m$ à droite. (De plus, le point p se trouve à gauche de l'axe d'équation $x = z$ et q se trouve strictement à droite de celui-ci.)

b) L'idée de l'algorithme est d'appeler `REPARTIR` afin d'obtenir E_1, E_2 et z à partir de E. On peut alors appeler récursivement l'algorithme sur E_1 et E_2 afin d'obtenir d_1 et d_2. Il ne reste plus qu'à appeler `DISTZONE` sur E_1, E_2, z et $\min(d_1, d_2)$ pour obtenir une valeur d. Le minimum cherché est alors $\min(d_1, d_2, d)$. On n'oublie pas le cas de base : si le nuage de point ne contient que deux points, le minimum cherché est la distance entre ces deux points.

```
DISTREC(E):
  if lenght(E) ≤ 1
  then return +∞
  else if length(E) == 2
  then return DISTANCE(E[0],E[1])
  else
    (E1,E2,d) ← REPARTIR(E)
    d1 ← DISTREC(E1)
    d2 ← DISTREC(E2)
    d ← DISTZONE(E1,E2,z,min(d1,d2))
    return min(d1,d2,d)
```

c) On prouve que `DISTREC` est correct par récurrence forte[1] sur la taille de E.

Initialisation. Si E est de taille 0 ou 1, on renvoie l'infini qui est le résultat attendu (le minimum d'un ensemble vide est toujours $+\infty$). Si E est de taille 2, alors l'algorithme renvoie la distance entre les seuls deux points du nuage de points E. Cette distance est minimale trivialement. L'algorithme est donc correct dans ce cas.

Hérédité. Supposons que l'agorithme soit correct sur toute entrée de taille strictement inférrieure à la taille de E. En particuler, `DISTREC` est correct sur E_1 et sur E_2, i.e. d_1 e d_2 sont les minimums des distances entre deux points distincts de E_1 et E_2 respectivement. Notons m la distance minimale entre deux points distincts de E et montrons que $m = \min(d_1, d_2, d)$. Commençons par remarquer que d_1, d_2 et d sont des distances entre points distincts de E (car `DISTREC` est supposée

[1] Au lieu de montrer qu'une propriété au rang n implique celle au rang $n+1$, on montre que la propriété à tous les rangs $k \leq n$ implique la propriété au rang $n+1$. Le principe de récurrence reste alors valide : le rang 0 implique le rang 1, le rang 0 et 1 impliue le rang 2, le rang 0 et 1 et 2 implique le rang 3, etc.

correcte sur E_1 et E_2 et DISTZONE est également supposée correcte) : par définition de m on a donc

$$m \leq d_1, \quad m \leq d_2 \quad m \leq d$$

Ce qui implique $m \leq \min(d_1, d_2, d)$. Il ne nous reste donc qu'à montrer l'inégalité inverse pour conclure. Etant donné $p, q \in E$ distincts, montrons que $\min(d_1, d_2, d) \leq d(p, q)$. Trois cas s'offrent à nous :

- si $p, q \in E_1$, alors d_1 étant la distance minimale entre deux points distincts de E_1 (DISTREC étant correcte sur E_1), on obtient $d_1 \leq d(p, q)$ et donc en particulier $\min(d_1, d_2, d) \leq d(p, q)$,
- si $p, q \in E_2$, alors d_2 étant la distance minimale entre deux points distincts de E_2 (DISTREC étant correcte sur E_2), on obtient $d_2 \leq d(p, q)$ et donc en particulier $\min(d_1, d_2, d) \leq d(p, q)$,
- enfin si $p \in E_1$ et $q \in E_2$ (ou inversement, le raisonnement reste le même), alors on sait par la question précédente que p et q sont tous deux dans la zone $Z(z, \min(d_1, d_2))$ pour laquelle d est la distance minimale entre deux points (DISTZONE étant correcte). On obtient donc $d \leq d(p, q)$ et en particulier $\min(d_1, d_2, d) \leq d(p, q)$.

Ainsi, dans tous les cas $\min(d_1, d_2, d)$ minore tous les $d(p, q)$ pour $p, q \in E$ distincts. Or m est défini comme le plus grand de ces minorants : ainsi $\min(d_1, d_2, d) \leq d$.

En regroupant les deux inégalités, on obtient le résultat voulu : $m = \min(d_1, d_2, d)$, autrement dit, DISTREC est correcte sur E.

d) *Très informellement.*[2] En supposant notre tableau de taille 2^k, on voit que la complexité $\gamma(2^k)$ de DISTREC(E) est donnée par la complexité de DISTREC sur deux entrées de taille 2^{k-1} ajoutée à la complexité de DISTZONE qui est supposée linéaire, et celle de REPARTIR également linéraire. On obtient donc

$$\gamma(2^k) = 2\gamma(2^{k-1}) + \mathrm{O}(2^{k-1}) + \mathrm{O}(2^k) = 2\gamma(2^{k-1}) + \mathrm{O}(2^k)$$

En considérant qu'on peut l'étendre à tous les n, on a

$$\gamma(n) = 2\gamma(n/2) + \mathrm{O}(n)$$

On reconnait une formule de complexité de la même forme que pour le tri fusion et on s'attend donc à obtenir un $\mathrm{O}(n \ln n)$.

Formellement. On va montrer que DISTREC a un complexité en $\mathrm{O}(n \ln n)$. Plus précisément, si l'on note $\gamma(n)$ la complexité de DISTREC sur une entrée de taille n, on va montrer qu'il existe une borne $K \in \mathbb{R}$ et un rang $n_0 \in \mathbb{N}$ tel que $\gamma(n) \leq Kn \log_2(n)$ pour tout $n \geq n_0$. Nos hypothèses sont que DISTZONE et REPARTIR sont linéaires, c'est-à-dire qu'en notant δ et κ leurs complexités respectives, il existe $K', K'' \in \mathbb{R}$ et $n_0', n_0'' \in \mathbb{N}$ tels que

$$\forall n \geq n_0',\ \delta(n) \leq K'n, \quad \forall n \geq n_0'',\ \kappa(n) \leq K''n$$

[2] Ce paragraphe explique l'intuition, mais a plutôt sa place au brouillon.

Quitte à prendre le maximum de K' et K'' et celui de n'_0 et n''_0, on peut supposer $K' = K''$ et $n'_0 = n''_0$. (Ceci permet surtout d'alléger les calculs par la suite.)
En lisant l'algorithme DISTREC, on déduit la formule :

$$\forall n > 2,\ \gamma(n) \leq \kappa(n) + 2\gamma(\lceil n/2 \rceil) + \delta(\lceil n/2 \rceil) + c$$

où c est une constante tenant compte des opérations d'assignations de variables et de la prise de minimum. Ce qui peut alors se réécrire pour les $n \geq 2n'_0$ comme :

$$\gamma(n) \leq K''n + 2\gamma(\lceil n/2 \rceil) + K'(\lceil n \rceil) + c$$

Posons $n_0 = 2n'_0$. On suit l'indication de l'énoncé et on ne considère que les n qui s'crivent comme puissances de 2. On montre alors par récurrence que

$$\forall k \geq \log_2(n_0),\ \gamma(2^k) \leq 2^k\gamma(n_0) + 2^k k(K' + K'' + c)$$

L'initialisation est claire. Supposons alors que l'inégalité soit vraie au rang $k-1$ pour un certain $k > \log_2(n_0)$. Alors :

$$\begin{aligned}
\gamma(2^k) &\leq K'2^k + 2\left(2^{k-1}\gamma(n_0) + 2^{k-1}(k-1)(K' + K'' + c)\right) + K''2^{k-1} + c \\
&\leq 2^k\gamma(n_0) + 2^k(k-1)(K' + K'') + 2^k(K' + K'') + 2^{k-1}(k-1)c + c \\
&\leq 2^k\gamma(n_0) + 2^k k(K' + K'' + c)
\end{aligned}$$

Le principe de récurrence conclut donc :

$$\forall k \geq \log_2(n_0),\ \gamma(2^k) \leq 2^k\gamma(n_0) + 2^k k(K' + K'' + c)$$

Si on pose $K = \gamma(n_0) + K' + K'' + c$, on a alors montré en particulier que pour tous les $n > n_0$ qui sont des puissances de 2 :

$$\gamma(n) \leq Kn\log_2(n)$$

4. Il suffit d'utiliser un tri qui a une complexité $\mathrm{O}(n \ln n)$. On peut utiliser le tri fusion ou le tri par tas.

5. a) Soit $p \in E$ de coordonnée (x, y) se trouvant dans la zone $Z(z, d_m)$. Supposons que $p \in E_1$ (le cas $p \in E_2$ se traite de manière similaire). On sait alors que $z - d_m < x \leq z$. Par définition de $d_m = \min(d_1, d_2)$ les autres points de E_1 qui se trouve dans $Z(z, d_m)$ avec ordonnée dans $[y - d_m, y + d_m]$ doivent se trouver à distance d_m au moins de p. Ils doivent également se trouver à distance au moins d_m les uns des autres : il y en a au plus 3 (dans le cas où $x = z$). Bien sûr il peut y avoir des points de E_2 dans la zone $Z(z, d_m)$ avec ordonnée dans $[y - d_m, y + d_m]$ et qui sont à distance moindre que d_m de p. Mais ceux-ci étant tous dans E_2, ils se doivent d'être à distance au moins d_m les un des autres. On peut certainement majorer leur nombre par 6, qui correspond au cas "limite" des points

$(z, y+d_m), (z, y), (z, y-d_m), (z+d_m, y+d_m), (z+d_m, y), (z+d_m, y-d_m)$

(Cette configuration est constituée de points qui ne sont pas dans $Z(z, d_m) \cap E_2$ mais ils délimitent clairement la zone où les points de E_2 considérés se trouvent.)

Au final, comme notre but est de majorer le nombre de points de $E \cap Z(z, d_m)$ dont les ordonnées sont dans $[y - d_m, y + d_m]$, on ne s'embête pas à saovir si nos cas son mutuellement exclusif ou non et on majre brutalement par $6 + 3 = 9$.

b) On peut maintenant concevoir un algorithme linéaire pour DISTZONE. Il suffit de prendre chacun des points $p \in E$ de coordonnées (x, y) de $Z(z, d_m)$ et de calculer sa distance à tous les autres points de $Z(z, d_m) \cap E$ dont l'ordonnée est comprise entre $y - d_m$ et $y + d_m$. Ainsi, d'après la question précédente, pour chaque tel p, on effectue au plus 9 calcul de distance. L'algorithme reste donc linéaire en la taille de $E \cap Z(z, d_m)$.

L'erreur à ne pas faire ici est de dire que pour chaque $p = (x, y)$ de E_1 dans la zone, on va parcourir tous les points de $E_2 \cap Z(z, d_m)$ et calculer la distance à p seulement si l'ordonnée se situe dans $[y - d_m, y + d_m]$. En effet, le fait même de parcourir $E_2 \cap Z(z, d_m)$ pour chaque point p de $E_1 \cap Z(z, d_m)$ rend l'algorithme quadratique ! Il faut profiter ici de ce que E_1 et E_2 sont triées par ordonnées croissantes. On peut donc dans un premier temps utiliser une fusion modifiée pour obtenir une liste de tous les points de la zone $E \cap Z(z, d_m)$ triés par ordonnées croissantes. On peut alors parcourir cette liste, et pour chaque point tester seulement ses voisins dont l'ordonnée se situe entre $y - d_m$ et $y + d_m$ dont on sait qu'ils ne seront pas plus de 9. (Et comme en question 1, on peut en fait se limiter aux successeurs de chacun des points puisque les prédécesseurs auront déjà été traités en amont.)

```
DISTZONE(E1,E2,z,dm):
  n1 ← length of E1
  n2 ← length of E2
  Z ← []

  # first we compute the zone Z by a custom merge of E1 and E2,
  # only retaining the points that are in the zone
  i ← 0
  j ← 0
  while i < n1 and j < n2 do
    (x1,y1) ← E1[i]
    (x2,y2) ← E2[j]
    if y1 ≤ y2
    then
      i ← i+1
      if x1 > z-dm
      then append (x1,y1) at Z
    else
      j ← j+1
      if x2 < z+dm
      then append (x2,y2) at Z
    end if
  end while
  while i < n1 do
```

```
  (x_1,y_1) ← E_1[i]
  append (x_1,y_1) at Z if x_1 > z-d_m
end while
while j < n_2 do
  (x_2,y_2) ← E_2[j]
  append (x_2,y_2) if x_2 < z+d_m
end while

# now we can walk through Z and compute distances when needed
min ← +∞
i ← 0
while i < (length of Z) - 1 do
  (x,y) ← Z[i]
  j ← i+1
  (x',y') ← Z[j]
  # we know this while loop will at most do 9 turns
  while y' ≤ y+d_m do
    d ← DISTANCE(Z[i],Z[j])
    if d < min
    then min ← d
    end if
    j ← j+1
  end while
  i ← i+1
end while

return min
```