Devoir surveillé n° 2 — 1h15

La calculatrice et l'aide-mémoire CAML ne sont pas autorisés.

I Exercices de cours

I.1) Une séquence `(expr1; expr2; ...; exprn)` où `expr1`, `expr2`, ..., `exprn` sont des expressions est une expression en CAML. Vrai ou faux ? Si oui, alors quel est son type et sa valeur. Si non, pourquoi ?

I.2) En CAML, quel est le type de l'expression `(let r = ref [] in r := 0 :: !r)` ? La syntaxe de cette expression est correcte mais avez-*vous* le droit de l'écrire ?

I.3) Combien allons-nous voir d'étoiles ?

```
for j = 0 to 20 do
  for i = 10 to j do
    print_string "*"
  done
done;;
```

I.4) Le programme suivant a été écrit par un élève [1]. Il comporte 4 erreurs.

```
let myst a=  let b=1 in
let c=ref 0;
while !b<a do
b:=2*b;incr c
done
!c;;
```

- Comprendre et expliquer ce que veut faire ce programme ;
- Signaler et corriger les erreurs ;
- Réécrire ce programme de manière lisible en respectant toutes les conventions que nous nous sommes imposées (indentation, espaces, nom des variables, etc).

I.5) On considère une équation de récurrence de la forme

$$T(n) = aT(\left\lfloor \frac{n}{2} \right\rfloor) + bT(\left\lceil \frac{n}{2} \right\rceil) + f(n)$$

avec une fonction de complexité $T : \mathbb{N}^* \to \mathbb{N}$, $a, b \in \mathbb{N}$, $a + b \geq 1$ et $f : \mathbb{N}^* \to \mathbb{N}$ que l'on suppose croissante. Que peut-on dire de la complexité $T(n)$, en ordre de grandeur asymptotique, en fonction de la manière dont $f(n)$ se compare, en ordre de grandeur, à des fonctions polynomiales ? On demande de citer directement, mais précisément, le résultat, pas de le redémontrer.

II Deux points les plus proches

On s'intéresse au problème suivant : étant donné un ensemble $\mathcal{P} = \{p_1, \ldots, p_n\}$ de $n \geq 2$ points *distincts* du plan, trouver deux points réalisant la distance minimale parmi ce nuage de points. Dans tout le problème on suppose que $n \geq 2$ et que tous les points sont distincts.

En PYTHON comme en CAML, on représente un point p_i par ses coordonnées, c'est-à-dire par un couple de flottants (x_i, y_i).

1. Qui n'étudie pas à Carnot, évidemment !

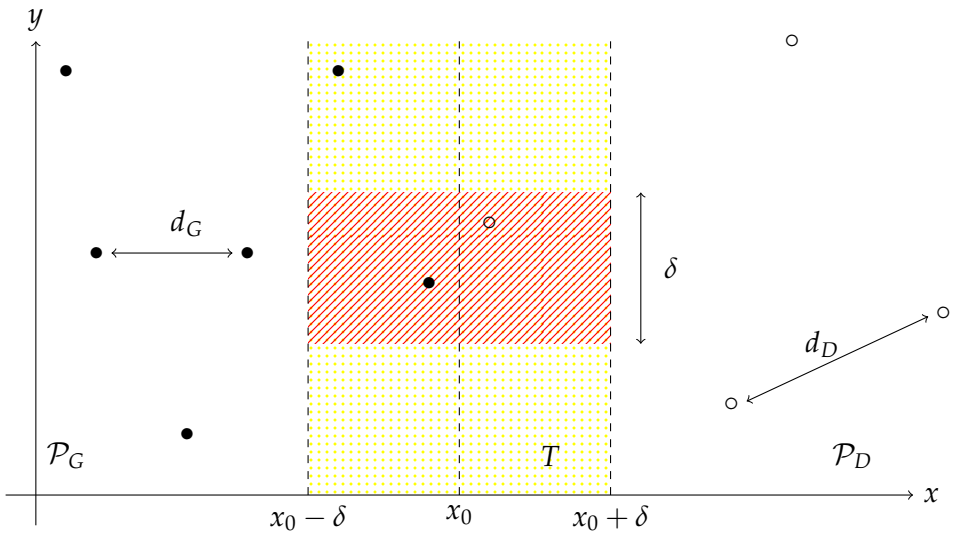


FIGURE 1 – Illustration de la stratégie diviser pour régner pour trouver la distance minimale dans un nuage de points.

II.1) En CAML, écrire une fonction `distance` qui calcule la distance euclidienne entre deux points du plan, donner son type et sa complexité. On s'attachera à rendre cette fonction aussi lisible que possible pour en faciliter la correction.

II.2) Écrire maintenant cette fonction `distance` en PYTHON. On supposera que la fonction `sqrt` est déjà importée, c'est-à-dire que l'on a déjà effectué `from math import sqrt`.

Une solution naïve au problème consiste à calculer toutes les distances entre tous les couples de points et à prendre celle qui réalise le minimum.

II.3) Écrire en PYTHON une fonction `distance_plus_proche` prenant en entrée une liste de n points et renvoyant la distance minimale entre deux points quelconques, en implémentant cette stratégie naïve. Comme toujours, une ou deux phrases d'explications sont attendues.

II.4) Quelle est la complexité de cette approche ?

II.5) Que faudrait-il[2] modifier dans cette fonction pour renvoyer également un couple de points du nuage de points réalisant ce minimum et comment faudrait-il s'y prendre ?

On va adopter une stratégie de type *diviser pour régner*. On sépare les points de $\mathcal{P}$ en deux parties $\mathcal{P}_G$ et $\mathcal{P}_D$ séparées par une droite verticale d'abscisse x_0 (voir figure 1). Notons d_G, respectivement d_D, la distance minimale entre deux points de $\mathcal{P}_G$, respectivement $\mathcal{P}_D$, et $\delta = \min(d_G, d_D)$.

II.6) Dans l'approche diviser pour régner, comment sont obtenues les distances d_D et d_G ?

II.7) Quel est le(s) cas de base (pour le(s)quel(s) il ne faut plus faire d'appels récursifs) ? Comment résoudre le problème pour le(s) cas de base et quelle en est la complexité ?

II.8) Expliquer comment choisir x_0 de telle manière à ce que la différence de cardinal entre $\mathcal{P}_G$ et $\mathcal{P}_D$ soit au plus un. Cette méthode devra avoir une complexité au pire en $\Theta(n \log n)$.

La distance minimale est soit δ, soit une distance entre un point de $\mathcal{P}_G$ et un point de $\mathcal{P}_D$. Dans ce dernier cas, elle est atteinte pour des points situés dans la bande délimitée par les droites verticales d'abscisses $x_0 - \delta$ et $x_0 + \delta$, que nous appelons T, représentée par des pointillés sur la figure 1. On note que dans le pire des cas il y a n points dans T.

II.9) Expliquer comment il est possible de sélectionner les points de $\mathcal{P}$ qui appartiennent à T, en temps linéaire.

2. On ne demande donc pas de le faire.

II.10) Montrer que dans une tranche de T de hauteur δ, représentée par la zone hachurée sur la figure 1, il y a au plus sept points.

Pour chaque point de T, il suffit donc de calculer les distances entre ce point et 12 autres points[3] de T, ce qui conduira à un coût linéaire en le nombre de points de T. Si les points de T sont triés par ordre croissant des ordonnées, les six points qui peuvent être dans la tranche de hauteur δ au dessus d'un point de T, sont parmi les six points qui suivent ce point dans le tableau (ou la liste) triée. Il suffit donc de trier les points de T par ordonnées croissantes.

II.11) En déduire, *très soigneusement*, que la complexité de la recherche de la distance des plus proches voisins d'un nuage de point vérifie une équation de récurrence de la forme

$$T(n) = \begin{cases} \Theta(1) \text{ si } n \in \{2,3\} \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + \Theta(n \log n) \text{ si } n \geq 4 \end{cases}$$

II.12) En déduire que $T(n) = \Theta(n \log^2 n)$. On remarque que le résultat du cours ne s'applique pas ici, mais que l'on peut utiliser la même méthode. On se limitera au cas où n est une puissance de 2.

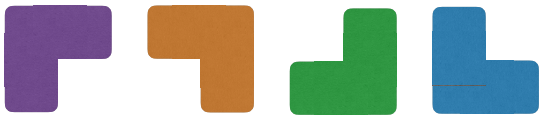
Dans l'approche proposée, il est nécessaire de trier les points lors de chaque appel récursif, et on s'aperçoit que la complexité de la division et de la combinaison est dominée par celle du tri. On se propose donc de trier le nuage de points lors d'un pré-traitement, une fois pour toute. On remarque qu'il est nécessaire de trier d'une part par rapport aux abscisses et d'autre part par rapport aux ordonnées. On utilise donc deux tableaux (ou listes) redondants, l'un trié par rapport aux abscisses et l'autre par rapport aux ordonnées.

II.13) Expliquer comment adapter les questions précédentes en tenant compte de ce pré-traitement et montrer que la complexité de la division et de la combinaison est alors en $\Theta(n)$.

II.14) En déduire la complexité de la méthode *diviser pour régner* pour répondre au problème des plus proches voisins dans un nuage de point.

III Pavage d'un échiquier

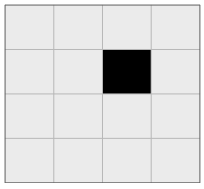
On considère un échiquier carré de côté 2^k avec $k \geq 1$ que l'on souhaite paver (recouvrir) avec les quatre motifs suivants :



III.1) Pourquoi est-ce impossible ?

On suppose que l'on accepte de laisser une case vide, choisie arbitrairement, que l'on colorie en noir. On peut alors évidemment recouvrir un échiquier 2×2 à l'aide l'une des quatre pièces, en fonction de l'emplacement de la case noire à laisser libre.

III.1) Donner un exemple de recouvrement de l'échiquier suivant :



III.2) Montrer, en raisonnant par récurrence sur k et en utilisant une approche *diviser pour régner*, qu'un tel recouvrement — laissant une case libre arbitraire — est toujours possible.

3. En fait, il suffit même de se restreindre aux 6 points d'ordonnées supérieures.