Devoir surveillé n° 3 — 3h

L'utilisation d'Internet, le partage d'informations avec autrui et la consultation de tout document sont interdits pendant toute la durée de l'épreuve.

Mise en place

- Se connecter avec le nom d'utilisateur `ds3-<login>` où `<login>` est votre nom d'utilisateur du laboratoire d'informatique (tout en minuscules avec les tirets) et le mot de passe par défaut.
- Ouvrir un terminal et entrer la commande `ls`. Vous devez voir les trois fichiers suivants :
  - `ds3-<login>.ml`
  - `expect_ds3.exp`
  - `eval_ds3.sh`

  Vous écrirez vos programmes CAML exclusivement dans le fichier de nom `ds3-<login>.ml` qui se trouve à la racine de votre répertoire, sans en changer le nom ni l'emplacement.
- Ouvrir EMACS à l'aide de la commande :

  ```
  emacs ds3-<login>.ml &
  ```
- Revenir dans le terminal et entrer la commande suivante :

  ```
  bash eval_ds3.sh ds3-<login>.ml
  ```

  Vous devez alors voir l'évaluation de votre code : aucune fonction n'est encore définie. Au fur et à mesure de l'épreuve, vous pouvez vérifier que vos fonctions sont bien prises en compte à l'aide de cette commande (n'oubliez pas que l'on peut remonter dans l'historique des commandes avec les flèches du clavier pour éviter de les retaper à chaque fois).

Consignes

Votre programme devra respecter scrupuleusement les noms et les types indiqués (ou un type plus général ou équivalent bien entendu) car il sera corrigé automatiquement. Le code fourni devra compiler sans erreurs ni aucun message d'avertissement. Vous n'utiliserez que des caractères ASCII pour les noms de variables. Il est impératif de veiller au strict respect de ces consignes.

Lorsque l'on pose des hypothèses sur les arguments (par exemple qu'un argument est un entier naturel), il n'est pas nécessaire dans vos fonctions de le vérifier ni de prévoir le comportement si l'hypothèse est violée (par exemple si l'argument est négatif). On pourra cependant lever des exceptions (`failwith "Argument non valable"` par exemple) si on le souhaite.

⚠ Le script d'évaluation affiche `vrai` si votre fonction est bien définie et possède, a priori, le bon type. Cela ne garantit pas du tout que votre programme est correct ! On prendra donc, comme d'habitude, bien soin de tester toutes les fonctions sur des cas triviaux, puis un peu plus complexes.

Si une fonction met plus d'une seconde à s'évaluer sur une entrée complexe, commentez ce test avant de rendre votre production. Ceci est particulièrement important dans la partie III. Vérifiez bien à la toute toute fin que le script d'évaluation s'exécute correctement !

Rappel EMACS

Sauvegarder	CTRL-X, CTRL-S
Interpréter la phrase courante	CTRL-X, CTRL-E
Tout interpréter	CTRL-C, CTRL-B
Interruption	CTRL-C, CTRL-K
Quitter	CTRL-X, CTRL-C

⚠ N'oubliez pas de sauvegarder ! Conseil : utiliser la combinaison de commandes CTRL-X puis CTRL-S (sauvegarder) *puis* CTRL-X puis CTRL-E (interpréter).

I Questions de cours

Question 1

Écrire une fonction `idx_max_array : 'a array -> int` qui renvoie le plus petit indice qui réalise le maximum d'un tableau ou -1 si le tableau est vide.

Question 2

Écrire une fonction `nb_occurrences : 'a -> 'a array -> int` qui renvoie le nombre d'occurrences d'un élément dans un tableau.

Question 3

Écrire une fonction `appartient : 'a -> 'a array -> bool` qui vérifie si un élément appartient à un tableau. On veillera à ce que la fonction soit efficace dans le meilleur cas.

Question 4

Écrire une fonction `rev_array : 'a array -> unit` qui inverse l'ordre des éléments d'un vecteur.

Question 5

Écrire une fonction `make_tenseur` de type `int -> int -> int -> 'a -> 'a array array array` qui créé et initialise un tableau à trois dimensions.

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.

On représente en CAML un point p_i par ses coordonnées, c'est-à-dire par un couple de flottants (x_i, y_i). On définit donc : `type point = float * float;;`. On représente un ensemble de points par un tableau de points.

Question 6

Écrire une fonction `distance`, de type `point -> point -> float` qui calcule la distance euclidienne entre deux points du plan. On rappelle que la distance euclidienne entre deux points de coordonnées respectives (x, y) et (x', y') est $\sqrt{(x-x')^2 + (y-y')^2}$.

II.1 Une méthode naïve

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.

Question 7

Implémenter cette stratégie naïve en écrivant une fonction `plus_proches_naif` de type `point array -> point * point` prenant en entrée un vecteur de n points et renvoyant un couple de points réalisant la distance minimale entre deux points (distincts) quelconques du nuage de points. Si plusieurs couples réalisent la distance minimale, on choisira le couple d'indice minimal pour l'ordre lexicographique.

Question 8

Quelle est la complexité de cette fonction dans le pire cas en fonction du nombre n de points ? Définir la constante de type `int`, `complexite_methode_naive`, dont la valeur est le numéro de la réponse correcte.

1 : $\Theta(n)$
2 : $\Theta(n \log n)$
3 : $\Theta(n \log^2 n)$
4 : $\Theta(n^2)$
5 : $\Theta(n^2 \log n)$
6 : $\Theta(n^3)$
7 : $\Theta(n!)$
8 : Aucune des réponses ci-dessus.

II.2 Méthode *diviser pour régner*

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 $\delta = \min(d_G, d_D)$. Dans l'approche *diviser pour régner*, les distances d_D et d_G sont obtenues par appel récursif (c'est l'étape « régner »). Le cas de base est atteint lorsqu'il y a moins de trois points, puisque le problème n'est défini que pour $n \geq 2$. Il suffit alors de calculer toutes les distances, ce qui se fait en $\Theta(1)$ puisqu'il y a moins de trois points.

Pour 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, on peut prendre la médiane des abscisses des points. Il existe des algorithmes permettant de faire cela en $\Theta(n)$, mais l'approche la plus simple est de trier les points par leurs abscisses et de prendre l'élément d'indice $\lfloor \frac{n}{2} \rfloor - 1$ du tableau trié. On a vu en cours, en TP, en DM et en DS que l'on peut trier une liste (ou un tableau) en $\Theta(n \log n)$ dans le pire des cas, à l'aide du « tri fusion ».

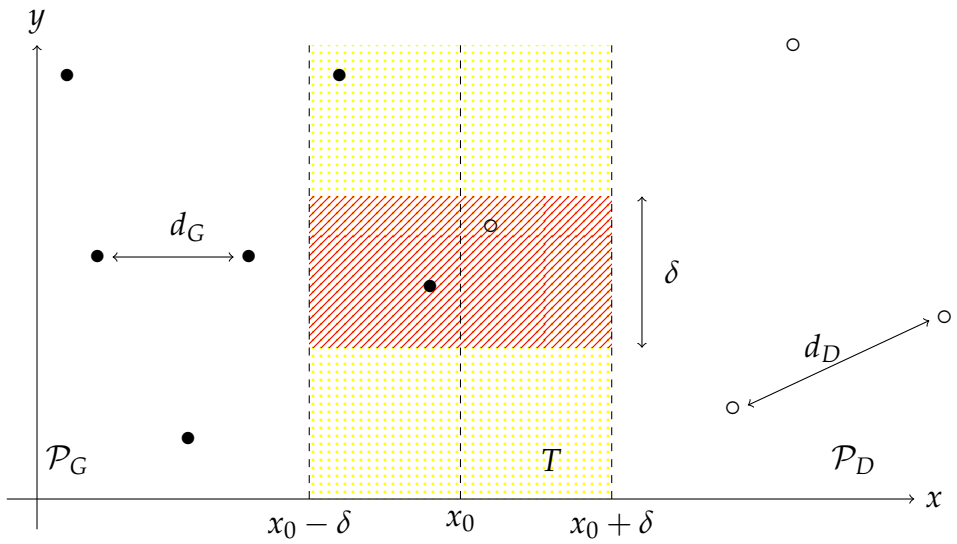

FIGURE 1 – Illustration de la stratégie *diviser pour régner* pour trouver la distance minimale dans un nuage de points.

La fonction `Array.stable_sort` implémente le tri fusion en CAML. La fonction `tri_abscisses`, que j'ai implémentée pour vous, permet de trier un tableau de points suivant l'ordre lexicographique abscisses/ordonnées.

Question 9

Écrire une fonction `tri_ordonnees` de type `('a * 'b) array -> unit` qui permet de trier un tableau par ordonnées d'abord, c'est-à-dire suivant l'ordre lexicographique ordonnées/abscisses, en adaptant évidemment la fonction `tri_abscisses`.

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.

Question 10

Écrire une fonction `selectionne_points_dans_T` de type `points array -> float -> float -> points array` qui prend pour arguments un tableau de points t ainsi que deux flottants x_0 et δ et qui sélectionne dans ce tableau les points dont l'abscisse est comprise entre $x_0 - \delta$ et $x_0 + \delta$ (inclus) sous la forme d'un tableau t'. L'ordre des points de t' devra être le même que celui dans t. La complexité de cette fonction devra être *linéaire* en la taille de t. Attention : on *ne suppose pas* que t est trié par abscisses et cette fonction ne doit pas modifier[a] t.

a. On ne peut de toutes façons pas trier t par abscisses ici puisque l'on exige une complexité linéaire.

Remarquons 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. Observons qu'un carré de côté δ ne peut contenir qu'au plus quatre points situés à une distance supérieure ou égale à δ, et que s'il en contient quatre, alors ceux-ci sont nécessairement situés aux quatre coins. Ceci ne peut pas être le cas à la fois du côté gauche et du côté droit. Ainsi, s'il y avait huit points ou plus, il y en aurait au moins deux appartenant à $\mathcal{P}_G$ ou $\mathcal{P}_D$ qui seraient à une distance strictement inférieure à δ ce qui est

impossible.

Pour chaque point de T, il suffit donc de calculer les distances entre ce point et les 6 points d'ordonnées immédiatement supérieures 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 trié. Il suffit donc de trier les points de T par ordonnées croissantes. La fonction `plus_proches_dans_T` implémente cela.

Question 11

Compléter le code de la fonction `plus_proches_diviser`, qui doit être de type `point array -> point * point` et qui implémente la méthode *diviser pour régner* décrite ci-dessus pour trouver le couple de points les plus proches.

Calculons la complexité $T(n)$ de cette approche. Pour $n \geq 4$, lors de chaque appel, on a :

- Calcul de x_0, ce qui nécessite un tri des abscisses des n points et donc une complexité en $\Theta(n \log n)$.
- Séparation de $\mathcal{P}$ en $\mathcal{P}_G$ et $\mathcal{P}_D$, ce qui se fait en $\Theta(n)$.
- Appels récursifs sur $\mathcal{P}_G$ et $\mathcal{P}_G$ dont les tailles sont [1] $\lfloor \frac{n}{2} \rfloor$ et $\lceil \frac{n}{2} \rceil$, pour un coût de $T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil)$.
- Calcul des points de T ce qui se fait en $\Theta(n)$.
- Tri des points de T par ordonnées en $\Theta(n \log n)$ dans le pire des cas.
- Calcul de la distance minimale des points de T, de complexité dans le pire des cas en $6 \times n \times \Theta(1) = \Theta(n)$
- Quelques autres opérations en $\Theta(1)$ (calcul du minimum de d_G et d_D, etc.)

En ajoutant ces complexités on trouve une équation de récurrence typique des méthodes *diviser pour régner*, avec un coût de division/combinaison en $\Theta(n \log n)$:

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

Question 12

Quelle est la complexité $T(n)$ de cette méthode dans le pire cas en fonction du nombre n de points ? Définir la constante de type `int`, `complexite_methode_diviser`, dont la valeur est le numéro de la réponse correcte.

1 : $\Theta(n)$
2 : $\Theta(n \log n)$
3 : $\Theta(n \log^2 n)$
4 : $\Theta(n^2)$
5 : $\Theta(n^2 \log n)$
6 : $\Theta(n^3)$
7 : $\Theta(n!)$
8 : Aucune des réponses ci-dessus.

On se limitera au cas où n est une puissance de 2. *Indication : poser $n = 2^k$ avec $k \geq 2$ et diviser par 2^k pour trouver une somme télescopique.*

Dans l'approche proposée, il est nécessaire de trier les points lors de chaque appel récursif. On se propose donc de trier le nuage de points lors d'un pré-traitement, une fois pour toutes. On remarque qu'il est nécessaire de trier d'une part par rapport aux abscisses et d'autre part par

1. Dans ce sens ou dans l'autre, selon la manière précise dont on définit $\mathcal{P}_G$ et $\mathcal{P}_D$, ce qui n'a pas d'importance ici.

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. On peut adapter sans problème les fonctions précédentes en tenant compte de ce pré-traitement et montrer que la complexité de la division et de la combinaison est alors en $\Theta(n)$.

Question 13

Quelle serait alors la complexité avec cette optimisation dans le pire cas en fonction du nombre n de points ? Définir la constante de type `int`, `complexite_methode_optimisee`, dont la valeur est le numéro de la réponse correcte.

1 : $\Theta(n)$
2 : $\Theta(n \log n)$
3 : $\Theta(n \log^2 n)$
4 : $\Theta(n^2)$
5 : $\Theta(n^2 \log n)$
6 : $\Theta(n^3)$
7 : $\Theta(n!)$
8 : Aucune des réponses ci-dessus.

On se limitera encore au cas où n est une puissance de 2.

III Contamination de cellules

Cette partie n'est pas évaluée par le script `eval_ds3.sh` mais en remplissant la fiche réponse « Contamination de cellules ». Il s'agit d'une partie de l'épreuve pratique d'algorithmique et de programmation du concours informatique des ÉNS.

Dans votre fichier `ds3-<login>.ml` est indiqué un numéro u_0 qui servira d'entrée à vos programmes. À la fin du sujet, se trouve aussi une fiche réponse déjà remplie avec les réponses attendues pour un $\tilde{u}_0$ particulier (précisé sur cette même fiche et que nous notons avec un tilde pour éviter toute confusion). Cette fiche est destinée à vous aider à vérifier le résultat de vos programmes en les testant avec $\tilde{u}_0$ au lieu de votre u_0. Vous indiquerez vos réponses (correspondant à votre u_0) sur la fiche réponse vierge et vous me la remettrez à la fin de l'épreuve.

On prendra garde aux éventuels dépassements de capacité de représentation des entiers en CAML.

Dans ce sujet, une *norme* est une application $||\cdot|| : \mathbb{N}^2 \to \mathbb{R}_+$, $(x,y) \mapsto ||(x,y)||$ qui possède des propriétés que vous étudierez l'année prochaine. Trois exemples de normes sont donnés dans le sujet.

Définition III.1

La distance associée à une norme $||\cdot||$ est l'application $d : \mathbb{N}^2 \times \mathbb{N}^2 \to \mathbb{R}_+$, qui a deux points (x,y) et (x',y') associe la quantité $||(x,y) - (x',y')|| = ||(x-x', y-y')||$.

Par exemple, les distances associées aux normes définies dans le sujet entre deux points $p = (x,y)$ et $p' = (x',y')$ sont :

- $d_1(p,p') = |x-x'| + |y-y'|$
- $d_2(p,p') = \sqrt{(x-x')^2 + (y-y')^2}$
- $d_\infty(p,p') = \max(|x-x'|, |y-y'|)$

Les distances ainsi définies correspondent à différentes notions possibles de « distance ».