Devoir surveillé n° 2 — corrigé

I Exercices de cours

I.1) Vrai. Son type et sa valeur sont ceux de `exprn`.

I.2) L'expression `let r = ref [] in r := 0 :: !r` est de type `unit` mais il vous est interdit d'utiliser des références de liste cette année, sauf mention explicite du contraire.

I.3) $\sum_{j=0}^{20} \sum_{i=10}^{j} = \sum_{j=10}^{20} j - 9 = 66$.

I.4) Les quatre erreurs sont :

- `b` doit être une référence ;
- On utilise un `;` au lieu du `in` ;
- Il faut déréférencer `b` ;
- Il manque un point virgule après le `done` pour séparer l'expression conditionnelle de l'expression `!c`.

On réécrit lisiblement le programme :

```
let log2 n =
  let p = ref 1 in   (* 2^k *)
  let k = ref 0 in
  while !p < n do
    p := 2 * !p;
    incr k
  done;
  !k
;;
```

Ce programme calcule la partie entière supérieure du logarithme en base 2 de manière itérative.

I.5) Cf. cours.

II Deux points les plus proches

II.1) On traduit directement la formule mathématique, en se souvenant que l'on manipule des flottants.

```
let distance (x1, y1) (x2, y2) =
  sqrt ((x1 -. x2) ** 2.0 +. (y1 -. y2) ** 2.0)
;;
```

Le type de cette fonction est bien sûr `(float * float) -> (float * float) -> float` et sa complexité est en $\Theta(1)$.

II.2) On fait de même ; sans problèmes de type liés aux opérateurs.

```
def distance(point1, point2):
    x1, y1 = point1
    x2, y2 = point2
    return sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)
```

II.3) Il suffit de tester toutes les paires de points possibles et de rechercher le minimum, ce que l'on obtient par deux boucles imbriquées. On a supposé que $n \geq 2$, comme indiqué.

```
def distance_plus_proche(points):
    n = len(points)
    d_min = distance(points[0], points[1])
    for i in range(n):
        for j in range(i + 1, n):
            d = distance(points[i], points[j])
            if d < d_min:
                d_min = d
    return d_min
```

II.4) Le corps des boucles imbriquées est de complexité constante — y compris l'appel à `distance` — et est exécuté $\binom{n}{2}$ fois, ce qui donne une complexité en $\Theta(n^2)$.

II.5) Il suffit de conserver un pointeur vers le couple de points recherchés, mis à jour lorsque la distance l'est.

II.6) Les distances d_D et d_G sont obtenues par appel récursif (régner).

II.7) 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.

II.8) On prend pour x_0 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 triée. On a vu en cours, en TD et en TP 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 ».

II.9) Il suffit de parcourir tous les points en ne conservant que ceux dont l'abscisse est entre $x_0 - \delta$ et $x_0 + \delta$, test qui se fait en $\Theta(1)$. Comme on parcourt n points dans le pire des cas, la complexité de cette approche est $\Theta(n)$.

II.10) 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.

II.11) Nous avons déjà étudié le cas $n < 4$. Lors de chaque appel, pour $n \geq 4$, 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 par simple parcours 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 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 bien l'équation de récurrence typique des méthodes diviser pour régner, avec un coût de division/combinaison en $\Theta(n \log n)$.

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.

II.12) On suppose que $n = 2^k$ avec $k \geq 2$. Pour $2 \leq j \leq k$, on a $T(2^j) = 2T(2^{j-1}) + \Theta(j2^j)$. En divisant par 2^j on obtient

$$\frac{T(2^j)}{2^j} - \frac{T(2^{j-1})}{2^{j-1}} = \Theta(j)$$

et en sommant entre 3 et k on obtient $\frac{T(2^k)}{2^k} = \Theta(k^2)$. Ainsi $T(n) = \Theta(n \log^2 n)$.

II.13) Si les points sont triées par abscisses d'une part et par ordonnées d'autre part, il n'est plus nécessaire de les trier lors des appels récursifs. En effet, lors de la constitution de $\mathcal{P}_G$, $\mathcal{P}_D$ et T, on peut sélectionner les points correspondants par simple parcours des deux tableaux redondants, en *conservant* le caractère trié des sous-tableaux obtenus. Le coût des étapes de sélection est inchangé, le coût des étapes de tri disparaît et les autres étapes n'ont pas à être modifiées. Ainsi le coût total des étapes de division et combinaison devient linéaire.

II.14) On obtient alors une relation de récurrence de la forme :

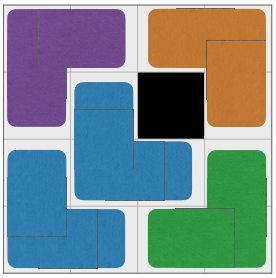
$$T(n) = \begin{cases} \Theta(1) \text{ si } n \in \{2,3\} \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + \Theta(n) \text{ si } n \geq 4 \end{cases}$$

D'après le cours, comme ici $\alpha = \log_2(1+1) = 1$, et que le coût de division/combinaison est en $\Theta(n^\alpha)$, on est dans le cas comparable pour lequel il y a résonance et la complexité est en $\Theta(n \log n)$, ce qui est mieux que précédemment et bien mieux que l'approche naïve.

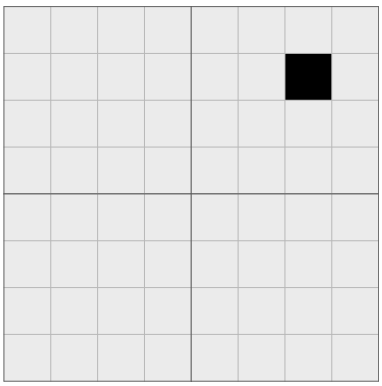
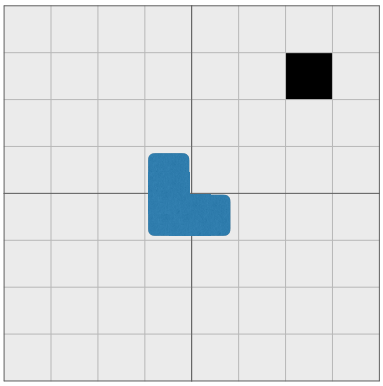
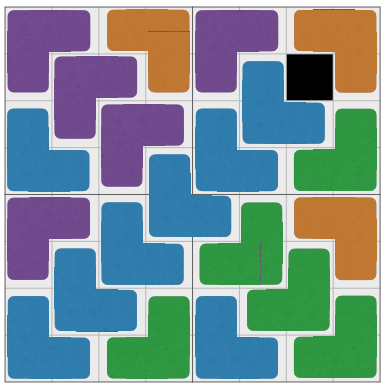
III Pavage d'un échiquier

III.1) Car 4^k n'est jamais divisible par 3.

III.2) On peut propose le pavage suivant :



III.3) L'initialisation, pour $k = 1$, est établie dans la remarque préliminaire. Soit $k \geq 1$ tel que l'on sache recouvrir un échiquier de côté 2^k sauf une case et soit un échiquier de côté 2^{k+1} dont une case arbitraire est noire. On peut diviser cet échiquier en quatre échiquiers de taille $2^k \times 2^k$. La case noire se situe sur l'un de ces quatre échiquiers (dans l'exemple celui situé au nord-est).

Parmi les quatre motifs, un seul peut-être disposé de manière à recouvrir les trois autres échiquier. On constate alors qu'il reste à recouvrir quatre échiquiers $2^k \times 2^k$ sauf une case, ce qui est possible par hypothèse de récurrence et qui montre que la propriété se transmet au rang $k+1$. Le raisonnement par récurrence permet de conclure.