TD 8 - Récursion et induction

Objectif : Mettre en œuvre les principes de l'induction.

Échauffement - Soit les ensembles de mots suivants :

$$E' = \{bon, jour, soir, bonjour, bonsoir, ou, on\}, \qquad E = E' \cup \{o, bo, jo, oi\}$$

Quels sont les éléments minimaux de E' ? Quel est l'ensemble des minorants de E' ? E' a-t-il un minimum, et si oui lequel ?

Exercice 1 - En considérant l'ordre sur $\mathbb{N}^*$ *est diviseur de*, montrer en utilisant le principe d'induction que tout entier supérieur ou égal à 2 est décomposable en un produit de facteurs premiers.

Exercice 2 - Soit un alphabet A. L'ensemble E est défini par :

$$A \subset E$$
$$\forall x, y \in E, (x)y \in E$$

Montrer que tout élément x de E contient autant de parenthèses ouvrantes que de parenthèses fermantes.

Exercice 3 - Soit A un ensemble fini, A^* l'ensemble des mots formés sur A, et A^k l'ensemble des mots de longueur k formés sur A. On donne les fonctions de base suivantes :

ε : la suite vide. C'est un élément de A^*
premier : $A^* \setminus \{\varepsilon\} \to A$ renvoie le premier élément d'une suite. premier(example) = e
dernier : $A^* \setminus \{\varepsilon\} \to A$ renvoie le dernier élément d'une suite. dernier(example) = e
saufpremier : $A^* \setminus \{\varepsilon\} \to A$: la suite excepté le premier élément. saufpremier(example) = xample
saufdernier : $A^* \setminus \{\varepsilon\} \to A$: la suite excepté le dernier élément. saufpremier(example) = exampl
estvide : $A^* \to \{vrai, faux\}$: vrai ssi le mot est vide. estvide(ε) = *vrai*, estvide(example) = $faux$.
longueur : $A^* \to \mathbb{N}$: le nombre d'éléments de la suite. longueur(example) = 7
concg : $A \times A^* \to A^*$: concaténation à gauche. concg(g, auche) = gauche
concd : $A \times A^* \to A^*$: concaténation à droite. concd(e, droit) = droite.

Soit l'algorithme suivant :

Algorithme 1 : Algo1(mot, a)

```
début
    /* ENTRÉES : Un mot mot ∈ A*, une lettre a ∈ A*/
    /* SORTIE : A déterminer */
    si estvide(mot) alors retourner ε
    sinon
        si premier(mot) = a alors retourner Algo1(saufpremier(mot), a)
        sinon retourner concg(premier(mot), Algo1(saufpremier(mot), a))
fin
```

1. Dérouler cet algorithme avec le mot *examen* et la lettre *e*
2. Quel est le rôle de cet algorithme ?
3. Démontrez-le.

Exercice 4 - Ecrire un algorithme **recursif** qui prend en entrée un mot $m \in A^*$, et qui renvoit $Vrai$ si ce mot est un palindrome, $Faux$ sinon.

On rappelle qu'un palindrome est un mot qui peut se lire à l'endroit comme à l'envers. Par exemple, *rever*, *kayak* ou encore $elle$ sont des palindromes.

Exercice 5 - Soient n points distincts répartis dans le plan. On cherche la distance minimale entre deux points. Pour cela, on propose un algorithme itératif, l'algorithme 2, et l'algorithme 3 fondé sur l'approche "diviser pour régner".

Pour l'algorithme 3, on constitue d'abord deux vecteurs de points : H contient les points triés par abscisse croissante ; V contient les points triés par ordonnée croissante. L'algorithme 3 effectue les opérations suivantes :

— On répartit les éléments de V en deux sous-vecteurs (fonction *select*) : V_g contient tous les points à gauche (largement) du point d'abscisse médiane $H(med)$, V_d contient tous les points à droite (strictement) du point d'abscisse médiane $H(med)$.

— On applique récursivement l'algorithme 3, d'une part sur $H(1 \to med)$ et V_g (la partie gauche des points), d'autre part sur $H(med+1 \to n)$ et V_d (la partie droite des points). On a ainsi la distance minimale dans chaque sous-ensemble. Soit δ le minimum de ces deux valeurs.

— Il reste maintenant à regarder la distance minimale entre un point du sous-ensemble de gauche et un point du sous-ensemble de droite. Il n'est pas nécessaire d'examiner tous les couples de points : il suffit de considérer uniquement les couples de points situés dans une bande verticale de largeur 2δ autour de la droite verticale passant par $V(med)$. En effet, les points hors de cette bande n'ont aucune chance d'être à une distance inférieure à δ. On fait donc de nouveau appel à la fonction *select*, sur les vecteurs V_g et V_d, pour constituer les vecteurs de points Q_g et Q_d, qui contiennent les points concernés (triés par ordonnées croissantes) respectivement de gauche et de droite. Puis la distance minimale entre un point de gauche et un point de droite est donnée une fonction *DistMinBande*, dont la complexité maximale en nombre de comparaisons est de $6n$.

a) Calculez la complexité de l'algorithme 2.

b) Proposez un algorithme pour la fonction *select*, qui a pour arguments un vecteur de points, une borne inférieure en abscisse et une borne supérieure en abscisse, et qui retourne le vecteur des points dont l'abscisse est dans l'intervalle entre la borne inférieure (strictement) et la borne supérieure (largement). Calculez sa complexité en nombre de comparaisons pour un vecteur de taille n (si vous ne trouvez pas, vous supposerez pour la suite qu'elle est de la forme λn).

c) Exprimez la complexité maximale, en nombre de comparaisons, de l'algorithme 3, $C_{max}(n)$, en fonction de $C_{max}(\lfloor n/2 \rfloor)$, $C_{max}(\lceil n/2 \rceil)$ et n.

d) Déduisez-en $C_{max}(n)$ en fonction de n.

e) Si l'on tient compte de l'étape initiale de constitution des vecteurs H et V, quel est l'ordre de grandeur asymptotique de la complexité maximale totale ?

Algorithme 2 : DistMinIt(P, n)

```
début
    /* ENTRÉES : un vecteur de points P de taille n*/
    /* SORTIE : la distance minimale entre 2 points de P */
    min ← ∞
    pour i = 1 → n faire
        pour j = i + 1 → n faire
            si distance(P((i), P(j)) < min alors
                min ← distance(P((i), P(j))
    retourner min
fin
```

Algorithme 3 : DistMinRec(H, V, n). Pour un point p, $p.x$ désigne l'abscisse de p.

```
début
    /* ENTRÉES : deux vecteurs de points de taille n : H contient des points triés par abscisse croissante ;
    V contient les mêmes points triés par ordonnée croissante */
    /* SORTIE : la distance minimale entre 2 points */
    si n<2 alors retourner ∞
    sinon si n=2 alors retourner distance(H(1), H(2))
    sinon
        med ← n ÷ 2
        V_g ← select(V, −∞, H(med).x)
        V_d ← select(V, H(med).x, +∞)
        δ_g ← DistMinRec(H(1 → med), V_g, med)
        δ_d ← DistMinRec(H(med + 1 → n), V_d, n − med)
        δ ← min(δ_g, δ_d)
        Q_g ← select(V_g, H(med).x − δ, H(med).x)
        Q_d ← select(V_d, H(med).x, H(med).x + δ)
        δ_m ← DistMinBande(Q_g, Q_d, δ)
        retourner min(δ, δ_m)
fin
```