Feuille d'exercices n°14 - Révisions de MP2I

Notions abordées

- recherche de motif : algorithme de Boyer-Moore, de Rabin-Karp, automate des motifs
- compression de texte : algorithme de Huffman, algorithme LZW
- programmation dynamique (plusieurs exemples)
- diviser pour régner et calcul de complexité
- logique propositionnelle : mise sous FNC, mise sous FND
- graphe de flot de contrôle

Exercice 1 : Algorithme de Boyer-Moore (simplifié)

Q. 1 Quel problème résout l'algorithme de Boyer-Moore ?

Q. 2 Donner le pseudo-code d'un algorithme naïf qui résout ce problème. Préciser sa complexité.

Q. 3 Rappeler l'idée de l'algorithme de Boyer-Moore.

Q. 4 On suppose que la fenêtre de texte $t[\![i, i+|x|[\![$ ne coïncide pas avec le motif x du fait d'un mauvais caractère à l'indice j, *i.e.* parce que $t_{i+j} \neq x_j$. En se concentrant sur ce caractère, quel est l'indice i' qui définit la prochaine fenêtre du texte qui est susceptible de coïncider avec le motif ?

Q. 5 Appliquer l'algorithme découlant de la remarque précédente sur le texte `tatatututoto` et pour le motif `tuto`.

Q. 6 Identifier quelles données peuvent être précalculées à partir du motif x pour rendre plus efficace le calcul du décalage proposé à la **Q.4**. Donner alors le pseudo-code de ce pré-calcul et préciser sa complexité.

Q. 7 On suppose que la fenêtre de texte $t[\![i, i+|x|[\![$ ne coïncide pas avec le motif x du fait d'un mauvais caractère à l'indice j, *i.e.* parce que $t_{i+j} \neq x_j$. On suppose que ce caractère est le premier en partant de la droite où la fenêtre du texte et le motif ne coïncident plus, ainsi $t[\![j+1, i+|x|[\![= x[\![j+1, |x|[\![$. En se concentrant sur ce suffixe, quel est l'indice i' qui définit la prochaine fenêtre du texte qui est susceptible de coïncider avec le motif ?

Q. 8 Appliquer l'algorithme découlant de la remarque précédente♣ sur le texte `babbababbbaab` et pour le motif `babaab`.

Q. 9 Identifier quelles données peuvent être précalculées à partir du motif x pour rendre plus efficace le calcul du décalage proposé à la **Q.7**. Donner alors le pseudo-code de ce pré-calcul et préciser sa complexité.

Q. 10 Appliquer l'algorithme de Boyer-Moore combinant les deux remarques précédentes sur le texte `taratatauturlututu` et pour le motif `turlututu`.

Q. 11 Proposer le pseudo-code complet de l'algorithme de Boyer-Moore. Quelle est la complexité de cet algorithme en fonction de la taille du motif et de celle du texte ?

♣. et de la remarque précédente uniquement

Exercice 2 : Recherche de motif avec un automate

On fixe Σ un alphabet, et un mot $x = x_1x_2\dots x_m \in \Sigma^+$. Pour un mot $t \in \Sigma^\star$ de longeur $n \in \mathbb{N}$ et $j \in [\![1, n]\!]$, on dira qu'une **occurence** de x se termine à l'indice j dans t si et seulement si $x_1x_2\dots x_m = t_{j-m+1}\dots t_{j-1}t_j$. Dans ce cadre là, les mots x et t ont des rôles très différents, c'est pourquoi on appellera x le **motif**, c'est-à-dire le mot dont on cherche des occurences, et on appellera **texte** un mot $y \in \Sigma^\star$ dans lequel on cherche des occurences du motif. Le but de cet exercice est de construire, à partir du motif seulement, un automate déterministe qui permette de calculer efficacement toutes les occurences du motif dans un texte.

Pour tout mot $w \in \Sigma^\star$, on note $\mathcal{P}(w)$, resp. $\mathcal{S}(w)$, l'ensemble des préfixes, resp. suffixes, de w.

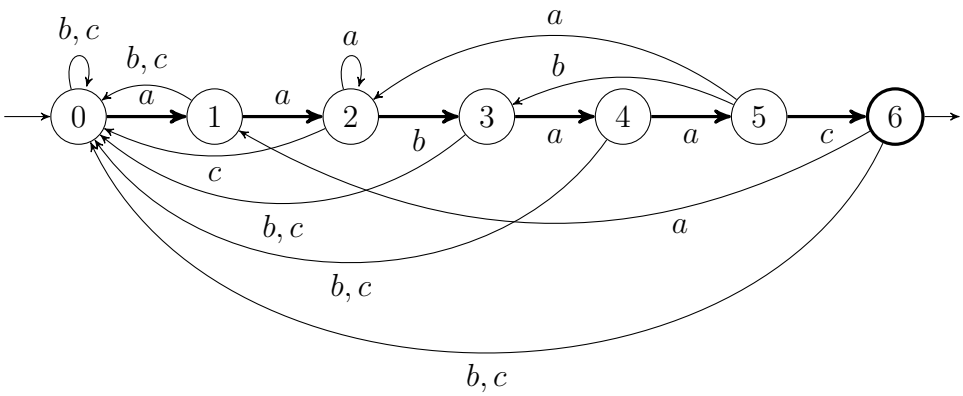
$$\forall w \in \Sigma^\star,\ \mathcal{P}(w) = \{p \in \Sigma^\star \mid \exists \overline{w} \in \Sigma^\star, w = p\cdot\overline{w}\} \quad \text{et} \quad \mathcal{S}(w) = \{s \in \Sigma^\star \mid \exists \underline{w} \in \Sigma^\star, w = \underline{w}\cdot s\}$$

Q. 1 Soit $u \in \Sigma^\star$. Pour $s \in \mathcal{S}(u)$ que vaut $\mathcal{S}(s)$ en fonction de $\mathcal{S}(u)$? *Une brève justification suffit.*

Q. 2 Soit $u \in \Sigma^\star$. $\mathcal{P}(u) \cap \mathcal{S}(u)$ peut-il être vide ?

Q. 3 Soit $u \in \Sigma^\star$. Donner un majorant de $\{|w| \mid w \in \mathcal{P}(u)\}$ et montrer qu'il est atteint (c'est donc un maximum).

L'automate du motif x est un automate déterministe à $m+1$ états qui servent à représenter la progression dans le motif. Par exemple l'automate du motif $aabaac$ est le suivant.



Afin de définir formellement un tel automate pour un motif x quelconque, on introduit la fonction f qui à une longueur $i \in [\![0, m]\!]$ et une lettre $a \in \Sigma$ associe la longueur du plus long suffixe de $x_1\dots x_i\cdot a$ qui soit aussi un préfixe de x.

$$f = \begin{pmatrix} [\![0,m]\!]\times\Sigma & \to & [\![0,m]\!] \\ (i,a) & \mapsto & \max\limits_{w \in \mathcal{P}(x)\cap\mathcal{S}(x_1\dots x_i a)} |w| \end{pmatrix}$$

Q. 4 Pour $i \in [\![0, m-1]\!]$, que vaut $f(i, x_{i+1})$?

On définit alors $\mathcal{A}_x = (\Sigma, Q = [\![0,m]\!], I = \{0\}, F = \{m\}, \delta)$ où $\delta = \{(i, a, f(i,a)) \mid i \in [\![0,m]\!], a \in \Sigma\}$

Q. 5 Pour le motif $x = babba$ sur l'alphabet $\Sigma = \{a, b\}$ (on a donc $m = 5$), dresser la table des $f(i,z)$ pour $(i,z) \in [\![0,m]\!]\times\Sigma$ puis dessiner $\mathcal{A}_x$.

Q. 6 Donner le pseudo-code d'un algorithme naïf construisant l'automate $\mathcal{A}_x$ à partir du motif x. On attend un algorithme naïf en $\mathcal{O}(|\Sigma|m^3)$.

Puisque l'automate $\mathscr{A}_x$ est complet et déterministe, pout tout état $q \in Q$ et toute lettre $a \in \Sigma$, il existe un unique état $q' \in Q$ tel que $(q, a, q') \in \delta$. On note cet état $\delta^1(q, a)$, ce qui revient à noter δ^1 la fonction de transition de $\mathscr{A}_x$. De plus on note $\delta^\star$ la fonction de transition étendue de $\mathscr{A}_x$.

$$\delta^\star = \left(\begin{array}{lll} Q \times \Sigma^\star & \longrightarrow & Q \\ (q, \varepsilon) & \mapsto & q \\ (q, z \cdot u) & \mapsto & \delta^\star\Big(\delta^1(q, z), u\Big) \\ \text{où } z \in \Sigma & & \end{array} \right) = \left(\begin{array}{lll} Q \times \Sigma^\star & \longrightarrow & Q \\ (q, \varepsilon) & \mapsto & q \\ (q, u \cdot z) & \mapsto & \delta^1\Big(\delta^\star(q, u), z\Big) \\ \text{où } z \in \Sigma & & \end{array} \right)$$

On a affirmé en introduction que les états de l'automate $\mathscr{A}_x$ servent à représenter la progression dans le motif x. Cette affirmation vague peut maintenant être reformulée de manière plus précise. Pour tout mot $u \in \Sigma^\star$, sur l'automate $\mathscr{A}_x$, la lecture de u mène à un état qui indique la longueur du plus long préfixe de x qui est un suffixe de u. Avec les notations introduites plus haut, cet invariant peut être reformulé comme suit.

$$\forall u \in \Sigma^\star,\ \delta^\star(0, u) = \max_{w \in \mathscr{P}(x) \cap \mathscr{S}(u)} |w|$$

Q. 7 Démontrer la propriété précédente par récurrence sur la taille de u.

Q. 8 Expliquer comment utiliser l'automate $\mathscr{A}_x$ pour repérer toutes les occurences de x dans un texte t de taille n ? Quelle serait la complexité temporelle de cette procédure (sans compter le pré-traitement consistant à créer $\mathscr{A}_x$) ? Quelle serait la complexité spatiale d'une telle méthode ?

Exercice 3 : Algorithme de Huffman (Révisions)

Q. 1 Donner l'arbre de codage produit par l'algorithme de Huffman pour le texte `abracadabrab`.
On placera les lettres par ordre alphabétique de gauche à droite.

Q. 2 À l'aide de l'arbre de codage précédent, compresser le texte `babadada`. Combien de bits compte le code obtenu ?

Exercice 4 : Algorithme de Rabin-Karp (Révisions)

Dans cet exercice on considère un alphabet $\Sigma = [\![0, B-1]\!]$ où $B \in \mathbb{N} \setminus \{0, 1\}$, un entier $N \in \mathbb{N}^\star$ et une fonction de hachage $h : \Sigma^\star \to [\![0, N-1]\!]$.
Pour les exemples de mots de $\Sigma^\star$, on utilisera les lettres a, b, c, … à la place des entiers 0, 1, 2, …

Q. 1 Existe-t-il une telle fonction h qui soit injective ?

On se place dans le cas où $\forall w \in \Sigma^\star$,

$$h(w) = \left(\sum_{i=1}^{|w|} w_i \right) \mod N.$$

Q. 2 Donner un algorithme permettant, pour tout $a_{k-1}a_{k-2}\ldots a_1a_0 \in \Sigma^*$ et $b \in \Sigma$, de calculer en $\mathcal{O}(1)$ la valeur de $h(a_{k-2}\ldots a_1a_0 \cdot b)$ connaissant celle de $h(a_{k-1}\ldots a_1a_0)$.

Q. 3 Appliquer l'algorithme de Rabin-Karp pour la fonction de hachage ci-dessus et la valeur $N = 10$ sur le texte $b\,a\,c\,j$ et le motif $a\,b$.

On se place dans le cas où $\forall w \in \Sigma^\star$,

$$h(w) = \left(\sum_{i=1}^{|w|} w_i B^{|w|-i} \right) \mod N.$$

Q. 4 Donner un algorithme permettant, pour tout $a_{k-1}a_{k-2}\dots a_1a_0 \in \Sigma^*$ et $b \in \Sigma$, de calculer en $\mathcal{O}(1)$ la valeur de $h(a_{k-2}\dots a_1a_0{\cdot}b)$ connaissant celle de $h(a_{k-1}\dots a_1a_0)$.
Indication : On s'autorisera, au besoin, un pré-calcul.

Q. 5 Appliquer l'algorithme de Rabin-Karp pour la fonction de hachage ci-dessus avec $B = 4$ et pour $N = 10$, sur le texte $c\,d\,a\,b$ et le motif $a\,b$.

Exercice 5 : Algorithme de Lempel-Ziv-Welsh (Révisions)

Q. 1 Rappeler quel est le principe de la compression selon LZW. Que peut-on dire des longueurs des motifs ajoutés au dictionnaire au cours de la compression ? Que peut-on dire des motifs ajoutés au dictionnaire lors de la décompression par rapport à ceux ajoutés lors de la compression ?

Q. 2 Exécuter à la main la compression de LZW sur le mot `abaaab`. *On pourra prendre comme dictionnaire de base celui qui numérote les lettres de l'alphabet minuscule de* 0 *à* 25.

Q. 3 Exécuter à la main la décompression de LZW sur la chaîne obtenue à la question précédente, sans utiliser le dictionnaire construit précédemment.

On cherche maintenant à implémenter la compression et la décompression selon LZW en OCaml. On utilise pour implémenter les dictionnaires (et les ensembles si besoin) le module `Hashtbl`. On fournit dans le fichier `compagnon_lzw.ml` plusieurs fonctions utiles, notamment de manipulation élémentaire des dictionnaires mis en jeu lors de la compression ou la décompression.

Q. 4 💻 Coder en OCaml la compression et la décompression selon LZW. Tester ces procédures sur des textes et calculer les taux de compression observés.

Q. 5 Citer un autre algorithme de compression (au programme). En quoi ces deux méthodes de compression diffèrent-elles ?

Éléments pour établir un algorithme de programmation dynamique

→ identifier une famille de "sous-problèmes"[a] qu'il est intéressant de résoudre ;
→ identifier quels paramètres permettent de caractériser ces sous-problèmes relativement à l'instance de départ ;
→ définir une quantité paramétrée qui représente la valeur optimale du sous-problème défini par ces paramètres ;
→ justifier que cette famille de quantités permet de résoudre le problème initial ;
→ dire comment calculer les cas de base, *i.e.* les quantités pour les paramètres minimaux ;
→ dire comment calculer l'une de ces quantités à partir des valeurs pour des paramètres plus petits ;
→ si on opte pour de l'impératif, fournir le pseudo-code qui explique quelle taille de tableau alouer pour stocker ces quantités, dans quel ordre on va remplir le tableau, comment on va ensuite récupérer la valeur optimale ;
→ dans le cas où seule la valeur optimale nous intéresse, il est souvent possible de modifier l'algorithme précédent en vue de réduire sa consommation mémoire ;

→ dans le cas où une solution optimale doit aussi être fournie, on complète l'algorithme précédent en ajoutant le plus souvent l'enregistrement d'information pertinente lors du remplissage du tableau, et en ajoutant une phase de reconstruction de la solution qui remonte dans le tableau à partir d'une case de valeur optimale.

a. On appelle ici, comme c'est l'usage, famille de sous-problèmes une famille d'instances, instances du problème initial ou parfois d'une variante de celui-ci.

Exercice 6 : Découpe de période

On suppose qu'on cherche à découper période de L jours en sessions de manière à maximiser l'utilité du découpage. La durée de chaque session est un nombre de jours entier choisi parmi un ensemble de durées possibles. À chaque durée de session possible est associée une utilité. L'utilité du découpage est la somme des utilités des sessions (peu importe l'ordre).

Q. 1 Formaliser ce problème concret comme un problème d'optimisation nommé DÉCOUPE.

Q. 2 Donner une relation de récurence permettant de résoudre le problème DÉCOUPE.

Q. 3 Donner le pseudo-code d'un algorithme de programmation dynamique permettant de calculer la valeur optimale d'une instance de DÉCOUPE.

Q. 4 Donner le pseudo-code d'un algorithme de programmation dynamique permettant de calculer une solution optimale d'une instance de DÉCOUPE.

Q. 5 Quelles sont les complexités spatiales et temporelles des deux algorithmes précédents ?

Exercice 7 : Plus long sous-mot/facteur commun

Q. 1 Rappeler les définitions de **facteur** d'un mot et **sous-mot** d'un mot.
Donner un exemple illustrant la différence entre les deux.

Q. 2 Définir le problème du plus long sous-mot commun (PLSMC).

Q. 3 Définir le problème du plus long facteur commun (PLFC).

Q. 4 Expliquer comment résoudre le problème PLSMC par programmation dynamique.

Q. 5 Expliquer comment résoudre le problème PLFC par programmation dynamique.

Exercice 8 : Accessibilité dans un graphe non orienté

Soit $G = (S, A)$ un graphe non orienté. On note $n = |S|$ et on suppose que $S = [\![1, n]\!]$. De plus on note $m = |A|$. Pour $(u, v) \in S^2$, on dit que u et v sont **reliés** dans G, ce qu'on note $u \sim v$ s'il existe une chaîne entre u et v dans G. On cherche à calculer la relation $\sim$, c'est-à-dire à déterminer pour chaque paire de sommets s'ils sont reliés par une chaîne du graphe. On cherche à calculer cette relation sous la forme d'une matrice de booléens $(T[u][v])_{(u,v)\in S^2}$ indiquant si $u \sim v$.

Sommets intérieurs Si $(\gamma_k)_{k\in[\![0,l]\!]}$ est une chaîne élémentaire♣ de G, on note $I(\gamma)$ l'ensemble des sommets intérieurs de γ, *i.e.* les sommets de γ qui n'en sont pas les extrémités.

$$I(\gamma) \overset{\text{déf}}{=} \{\gamma_k \mid k \in [\![1, l-1]\!]\}$$

On remarque qu'une chaîne ayant seulement 0 ou 1 arête ne possède aucun sommet intérieur, autrement dit si $l \leqslant 1$, alors $I(\gamma) = \emptyset$ et la réciproque est vraie.

Chaînes élémentaires dans les sous-graphes. Pour tout $(u, v) \in S^2$, on note $\mathcal{C}_{u,v}$ l'ensemble des chaînes élémentaires de u à v. De plus, pour tout $k \in [\![0, n]\!]$, on définit $\mathcal{C}^{(k)}_{u,v}$ l'ensemble des chaînes élémentaires, de u à v, dont tous les sommets intérieurs sont dans $[\![1, k]\!]$.

$$\mathcal{C}^{(k)}_{u,v} \overset{\text{déf}}{=} \{\gamma \in \mathcal{C}_{u,v} \mid I(\gamma) \subseteq [\![1, k]\!]\}.$$

♣. *i.e.* ne passant pas deux fois par un même sommet

Famille de relations $\sim_k$. On définit finalement, pour $k \in [\![0, n]\!]$, la relation binaire $\sim_k$ sur S comme suit.

$$\forall (u, v) \in S^2, u \sim_k v \overset{\text{déf}}{\Leftrightarrow} C_{u,v}^{(k)} \neq \emptyset$$

Autrement dit, $u \sim_k v$ dès lors qu'il existe une chaîne de u à v n'utilisant, pour sommet intérieurs, que des sommets de $[\![1, k]\!]$.

Q. 1 Démontrer ou réfuter la proposition suivante : $\forall k \in [\![1, n]\!]$, la relation $\sim_k$ est une relation d'équivalence.

Q. 2 Justifier en quoi le calcul des relations $(\sim_k)_{k \in [\![0,n]\!]}$ permet de répondre au problème du calcul de $\sim$.

On cherche à établir des relations de récurrence permettant le calcul des $(\sim_k)_{k \in [\![0,n]\!]}$.

Q. 3 Que vaut $\sim_0$?

Q. 4 Soient $u \in S$, $v \in S$, $k \in [\![1, n]\!]$. Proposer et démontrer une relation entre $u \sim_k v$ et $u \sim_{k-1} k$, $k \sim_{k-1} v$ et $u \sim_{k-1} v$.

Q. 5 Quel problème pose l'implémentation d'une telle relation de récurrence ?

Dans les algorithmes qui suivent, la famille de relations $(\sim_k)_{k \in [\![0,n]\!]}$ sera représentée par un tableau tri-dimensionnel T indicé par $[\![0, n]\!] \times S^2$ contenant des booléens :

$$\forall k \in [\![0, n]\!], \forall (u, v) \in S^2, T[k][u][v] = \text{V si et seulement si } u \sim_k v$$

Q. 6 Proposer un algorithme permettant de calculer $\sim$, tout en évitant le problème soulevé en **Q. 5**.

Q. 7 Dans le cas où la complexité spatiale de l'algorithme proposé n'est pas en $\mathcal{O}(n^2)$, proposer une amélioration permettant d'atteindre une telle complexité spatiale.

Q. 8 De quel algorithme au programme de MP2I l'algorithme 6 est-il proche ?

Q. 9 Au moyen d'un parcours de graphe, proposer une solution algorithmique alternative au problème du calcul de $\sim$. Préciser la complexité temporelle de cette solution.

Exercice 9 : Suites récurrentes de complexité

Dans cet exercice il vous est demandé de donner, pour chacune des suites suivantes :
- un Θ décrivant son comportement asymptotique ;
- un programme OCaml **dont cette suite est la complexité**, pour des constantes a et b au choix♣ ;
- un schéma de l'arbre des appels récursifs d'un appel au programme proposé.

a) $\begin{cases} u_0 &= 1 \\ u_n &= u_{n-1} + a \quad a \in \mathbb{R}^{+\star} \end{cases}$

b) $\begin{cases} u_0 &= 1 \\ u_n &= u_{n-1} + an \quad a \in \mathbb{R}^{+\star} \end{cases}$

c) $\begin{cases} u_0 &= 1 \\ u_n &= au_{n-1} + b \quad a \in [2, +\infty[, b \in \mathbb{R}^{+\star} \end{cases}$

d) $\begin{cases} u_0 &= 1 \\ u_n &= u_{n/2} + b \quad b \in \mathbb{R}^{+\star} \end{cases}$

e) $\begin{cases} u_0 &= 1 \\ u_n &= u_{n/2} + bn \quad b \in \mathbb{R}^{+\star} \end{cases}$

f) $\begin{cases} u_0 &= 1 \\ u_1 &= 1 \\ u_n &= u_{\lceil \frac{n}{2} \rceil} + u_{\lfloor \frac{n}{2} \rfloor} + b \quad b \in \mathbb{R}^{+\star} \end{cases}$

♣. Insistons : il s'agit écrire un programme OCaml ayant cette suite comme complexité et non un programme OCaml calculant cette suite.

$g)$ $\begin{cases} u_0 &= 1 \\ u_n &= au_{n/2} + b \quad a \in [2, +\infty[, b \in \mathbb{R}^{+\star} \end{cases}$ $h)$ $\begin{cases} u_0 &= 1 \\ u_1 &= 1 \\ u_n &= u_{\lceil \frac{n}{2} \rceil} + u_{\lfloor \frac{n}{2} \rfloor} + bn \quad b \in \mathbb{R}^{+\star} \end{cases}$

Exercice 10 : Quelques équivalents classiques

Q. 1 Donner le comportement asymptotique des suites suivantes♣ au moyen de la notation Θ.

$a)$ $\lfloor \log_2(n) \rfloor$ $b)$ $2^{\lfloor \log_2(n) \rfloor}$

Q. 2 Donner le comportement asymptotique des sommes suivantes♡ au moyen de la notation Θ.

$a)$ $\sum_{k=1}^{n} k$ $c)$ $\sum_{k=0}^{n} 2^k$ $e)$ $\sum_{k=0}^{n} k2^k$ $g)$ $\sum_{k=1}^{n} \log k$

$b)$ $\sum_{k=1}^{n} k^2$ $d)$ $\sum_{k=0}^{n} \frac{1}{2^k}$ $f)$ $\sum_{k=1}^{n} \frac{1}{k}$ $h)$ $\sum_{k=0}^{n} k \log k$

Exercice 11 : Éléments majoritaires

On dit qu'un élément x est majoritaire dans un multi-ensemble E de cardinal n lorsque le nombre d'occurrences de x dans E est strictement supérieur à $\frac{n}{2}$. On suppose donc un multi-ensemble E représenté par un tableau de ses éléments et on cherche si oui ou non E admet un élément majoritaire (s'il existe, il est unique). Dans le cas d'une réponse affirmative on renverra l'élément en question. On s'intéresse dans cet exercice au nombre de tests d'égalité (ou inégalité évidemment) effectués entre des éléments du multi-ensemble E. On suppose définie une fonction `nbOccur` prenant en arguments un tableau T, un élément x, et deux indices i et j du tableau et retournant le nombre d'occurrences de x dans le tableau T entre les indices i et j (au sens large). On suppose de plus que l'appel `nbOccur`(T, x, i, j) fait $j - i + 1$ comparaisons.♠

Q. 1 Rappeler la définition d'un algorithme naïf `majoritaire`(T) retournant s'il existe un élément majoritaire du multi-ensemble E représenté par le tableau T. On précisera la complexité pire cas de l'algorithme.

Q. 2 Supposant connu l'élément majoritaire du sous-tableau $T[0..n/2-1]$♢ et l'élément majoritaire du sous-tableau $T[n/2..n-1]$ où T est un tableau de taille n, donner l'élément majoritaire de T en faisant de l'ordre de n comparaisons.

Q. 3 Donner alors un algorithme du paradigme diviser pour régner permettant la résolution du problème de la recherche d'un élément majoritaire.

Q. 4 Donner la complexité pire cas dans le cas où la taille du tableau donné en entrée est une puissance de 2.

Exercice 12 : Sous-tableau de somme maximale

Étant donné un tableau d'entiers, le problème du sous-tableau maximum consiste à trouver un ensemble d'indices consécutifs (*i.e.* un intervalle d'indices) non vide qui maximise la somme des éléments du tableau. Par exemple, le sous-tableau maximum du tableau `[-3, 4, 5, -1, 2, 3, -6, 4]` est le tableau`[4, 5, -1, 2, 3]` de valeur 13. Étant donné un tableau T on note donc `stm`(T) la valeur du sous-tableau maximum.

♣. *i.e.* implicitement indexées par n

♡. *i.e.* des suites, implicitement indexées par n, des sommes jusqu'au rang n

♠. Les suppositions des deux phrases précédentes sont là pour vous faire gagner du temps, il **faut** que vous sachiez implémenter un tel algorithme `nbOccur`.

♢. comprendre : on sait s'il existe un élément majoritaire et s'il existe, on sait qui il est

Q. 1 Formaliser ce problème comme un problème d'optimisation.

Q. 2 Dans un premier temps, on considère l'algorithme de résolution ci-dessous. Quelle est la complexité de cet algorithme ?

Algorithme 1 : Sous tableau maximum

Entrée : A un tableau de taille n
Sortie : $\mathsf{stm}(A)$

```
1 A_max ← max A[1], ..., A[n] ;
2 pour i = 1 à n faire
3     pour j = i à n faire
4         sum ← 0 ;
5         pour k = i à j faire
6             sum ← sum + A[j] ;
7             si sum > A_max alors
8                 A_max ← sum
9 retourner A_max
```

Q. 3 Proposer une variante en $\Theta(n^2)$ de cet algorithme.

Dans la suite de l'exercice, on s'intéresse à des algorithmes de type diviser pour régner. Pour un tableau A indicé par $[1..n]$ avec $n > 1$, on note $A_1 = A[1..m]$ et $A_2 = A[m+1..n]$ les deux sous-tableaux définis par $m = \left\lfloor \frac{n+1}{2} \right\rfloor$. On note alors $\mathsf{stm}_1(A) = \mathsf{stm}(A_1)$ et $\mathsf{stm}_2(A) = \mathsf{stm}(A_2)$. On introduit aussi $\mathsf{stm}_3(A)$ la valeur maximum d'un sous-tableau de A qui couvre à la fois les indices m et $m+1$.

Q. 4 Comment calculer $\mathsf{stm}_3(A)$ efficacement ? Quelle est la complexité en fonction de n la taille de A ?

Q. 5 Connaissant $\mathsf{stm}_1(A)$, $\mathsf{stm}_2(A)$ et $\mathsf{stm}_3(A)$, quelle est la valeur de $\mathsf{stm}(A)$? Dans le cadre d'une méthode diviser pour régner qui calcule $\mathsf{stm}(A)$ en calculant récursivement et $\mathsf{stm}_1(A)$ et $\mathsf{stm}_2(A)$ en quoi consisterait l'opération de fusion ? Quelle serait la complexité de cette opération ?

Q. 6 Soit T_n le nombre d'opérations élémentaires (additions et comparaisons d'éléments du tableau) que réalise cette méthode pour un tableau de taille n. Donner l'équation de récurrence satisfaite par T_n. En déduire la complexité de la méthode.

On s'intéresse maintenant à une autre approche diviser pour régner afin de réduire encore la complexité. Pour ce faire, on définit, pour un tableau A indicé par $[1..n]$ les valeurs suivantes :

- $\mathsf{pref}(A) = \max\{\sum_{k=1}^{i} A[k] | i \in [1..n]\}$
- $\mathsf{suff}(A) = \max\{\sum_{k=i}^{n} A[k] | i \in [1..n]\}$
- $\mathsf{tota}(A) = \sum_{k=1}^{n} A[k]$

Autrement dit, $\mathsf{pref}(A)$ (resp. $\mathsf{suff}(A)$) est la valeur maximum d'un sous-tableau préfixe (resp. suffixe) de A, et $\mathsf{tota}(A)$ est la somme des valeurs de A.

Q. 7 Expliquer comment calculer ces valeurs pour un tableau A.

Q. 8 Quelle est la complexité de l'algorithme diviser pour régner associé ? Justifier.

Exercice 13 : Multiplication d'entiers selon Karatsuba

Dans cet exercice on s'intéresse au problème de la multiplication de grands entiers machines. On suppose les entiers représentés par la séquence indicée des bits de leur décomposition en base 2. Par exemple, l'entier 14 est représenté par la séquence $(1, 1, 1, 0)$. On appellera alors taille d'un entier la longueur d'une telle séquence. On suppose de telles séquences stockées dans des tableaux permettant l'indexation en temps constant. On remarque que les multiplications et divisions entières par des puissances de 2 correspondent à des décalages de bits. En effet, si $(u_{n-1}, u_{n-2}, \ldots, u_0)$ est la représentation d'un entier u et $p \in \mathbb{N}$ alors $u/2^p$ est représenté par $(u_{n-p-1}, u_{n-p-2}, \ldots, u_p)$ et $u \times 2^p$ par $(u_{n-1}, u_{n-2}, \ldots, u_0, \underbrace{0, 0, \ldots 0}_{p})$. Dans tout l'exercice on s'intéresse uniquement au problème de la multiplication de deux entiers de même taille et on mesure la complexité au regard du nombre de multiplications de nombres à 1 bit. Ainsi l'opération 1×0 est considérée comme une opération atomique de coût 1 alors que le coût de calcul de 1110110×101110 dépend de l'algorithme choisi pour faire le calcul de $\times$.

Q. 1 Expliquer en quoi l'hypothèse d'une taille commune des deux opérandes n'est pas trop simplificatrice.

Q. 2 Rappeler l'algorithme de multiplications naïf d'entiers (celui appris en primaire), décliné pour des nombres écrits en base 2. Quelle est sa complexité ?

On remarque que si $u = u_a + 2^p u_b$ avec $u_a < 2^p$ et $v = v_a + 2^p v_b$ avec $v_a < 2^p$ alors $u \times v = u_a \times v_a + 2^p(u_b \times v_a + v_b \times u_a) + 2^{2p} u_b \times v_b$.

Q. 3 Proposer un algorithme de multiplication d'entiers utilisant le paradigme diviser-pour-régner et utilisant la remarque précédente. Faire une rapide étude de complexité dans le cas où les entrées sont de taille 2^k. Cet algorithme diviser-pour-régner présente-t-il un avantage par rapport à l'algorithme naïf.

Q. 4 En s'intéressant à la valeur de $u_a v_a + u_b v_b - (u_a - u_b)(v_a - v_b)$ proposer un algorithme faisant trois appels récursifs sur des entrées dont la taille est divisée par 2.

Q. 5 Faire une étude rapide de complexité dans le cas où les entiers en arguments sont de taille $n = 2^p$.

Q. 6 Donner la complexité pire cas de l'algorithme de la question 4 au moyen d'un Θ. Au vu de la question précédente, on pourra intuiter puis démontrer par récurrence un encadrement sur le nombre de multiplications élémentaires effectuées par l'algorithme de la question 4 sur deux entrées de taille respectivement n et m.

Q. 7 Dans les questions précédentes nous avons ignoré le coût des additions et soustractions sur les grands entiers. On ne suppose plus que c'est le cas : une addition, soustraction sur deux entiers de tailles n et m coûte dorénavant $\max(n, m)$. Faire une rapide étude de complexité de l'algorithme de la question 4 (on se contentera des entiers de la forme 2^p). Conclure.

Exercice 14 : Algorithmes en logique propositionnelle (révisions)

On fixe dans cet exercice la formule $H = \Big((y \vee x) \wedge (\neg y \vee z)\Big) \leftrightarrow \Big(z \rightarrow (x \wedge y)\Big)$

Q. 1 Appliquer l'algorithme du cours pour fournir H' une FND équivalente à H.

Q. 2 Appliquer l'algorithme du cours pour fournir H'' une FNC équivalente à H.

Q. 3 Appliquer à H' un algorithme polynomial permettant de résoudre le problème FND-SAT. Que peut-on en déduire pour H ?

Q. 4 Appliquer l'algorithme de Quine pour déterminer si H'' est satisfiable, et dans ce cas donner un environnement propositionnel qui la satisfait.
On traitera les variables dans l'ordre alphabétique et on commencera par appliquer une substitution par $\top$.

Exercice 15 : Graphe de flot de contrôle

Q. 1 Donner le graphe de flot de contrôle de la fonction `expo_rapide` dont le code C est donné ci-dessous.

```c
int expo_rapide(int n, int p) {
  /* hyp : p ⩾ 0
     retourne n^p
   */
  int res;
  int m;
  int aux;

  res = 1;
  m = p;
  aux = n;
  while (m != 0) {
    if (m % 2 != 0) {
      res = res * aux ;
    }
    aux = aux * aux;
    m = m / 2;
  }
  return res;
}
```

Q. 2 Proposer un couple (n,p) de valeurs assurant que chaque arc du graphe de flot de contrôle de la question précédente est emprunté lors de l'exécution de la fonction `expo_rapide` sur ces valeurs `n` et `p`.

Q. 3 *Question subsidiaire* Proposer un invariant de boucle liant les variables `res`, `aux`, `n` et `m` qui permettrait de montrer la correction de la fonction `expo_rapide`.