

Programmation fonctionnelle – Projet (groupe 2)

1 Informations générales

1.1 Travail à rendre

Le projet est à réaliser en OCaml **individuellement**. Il sera accompagné d'un **dossier** contenant impérativement la description des choix faits, la description des types et des fonctions. Pour chaque fonction, on donnera impérativement l'interface complète (dans le code en commentaire et dans le rapport pour les fonctions présentées).

Même si le sujet est décomposé en questions, il est possible qu'une question se résolve par l'écriture d'une ou plusieurs fonctions intermédiaires. Celles-ci doivent comporter une interface également.

Le dossier fournira également des cas de tests accompagnés des résultats attendus et retournés.

Sur le site du cours figure un petit document sur ce que l'on attend dans un rapport. Consultez-le!

Attention, le code doit être purement fonctionnel : pas de boucle, pas d'affectation, pas de tableau. La seule tolérance concerne les affichages (voir document en ligne).

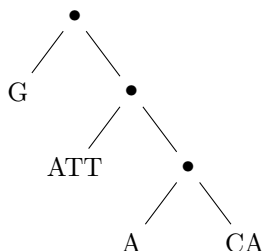
1.2 Calendrier et procédure de remise

Le projet est à rendre le **mercredi 29 mars 2017** minuit au plus tard sur le site de dépôts `exam.ensiee.fr`. Cliquez sur **IPF_S2_2017**. Vous y déposerez une archive contenant votre rapport au format pdf (impérativement) et le code de votre projet.

2 Enoncé du projet

La majorité des langages de programmation fournissent une notion primitive de chaîne de caractères. Si ces chaînes s'avèrent adaptées à la manipulation de mots ou de textes relativement courts, elles deviennent généralement inutilisables sur de très grands textes. L'une des raisons de cette inefficacité est la duplication d'un trop grand nombre de caractères lors des opérations de concaténation ou l'extraction d'une sous-chaîne.

Or il existe des domaines où la manipulation efficace de grandes chaînes de caractères est essentielle (représentation du génome en bio-informatique, éditeurs de texte, ...). Ce projet propose une alternative à la notion usuelle de chaîne de caractères que nous appelons **string_builder**. Un **string_builder** est un arbre binaire, dont les feuilles sont des chaînes de caractères usuelles et dont les noeuds internes représentent des concaténations. Ainsi le **string_builder**



représente le mot **GATTACA**, obtenu par concaténation de quatre mots **G**, **ATT**, **A** et **CA**. L'intérêt des **string_builder** est d'offrir une concaténation immédiate et un partage possible de caractères entre plusieurs chaînes, au prix d'un accès aux caractères un peu plus coûteux.

Un `string_builder` est donc soit un mot (feuille), soit une concaténation de deux autres `string_builder` (noeud). Pour des raisons d'efficacité, on conserve dans les feuilles aussi bien que dans les noeuds la longueur $\text{length}(c)$ de la chaîne de caractères c correspondante.

2.1 Échauffement

Question 1. Définir le type `string_builder`.

Définir la fonction `word` qui prend en argument une chaîne de caractères et qui renvoie le `string_builder` correspondant.

Définir la fonction `concat` qui prend en argument deux `string_builder` et qui renvoie le nouveau `string_builder` résultant de leur concaténation.

Question 2. Définir une fonction `char_at` qui prend en argument un entier i et un `string_builder` représentant le mot $[c_0 ; \dots ; c_{n-1}]$, et qui renvoie le caractère c_i .

note : On supposera $0 \leq i < n$, et on utilisera la fonction `String.get` pour obtenir le i^{e} caractère d'une chaîne de caractères.

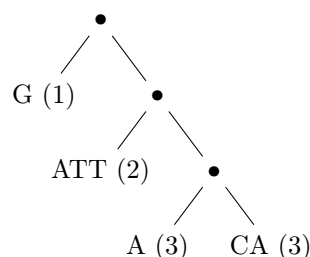
Question 3. Définir une fonction `sub_string` qui prend en arguments un entier i , un entier m et un `string_builder` sb représentant le mot $[c_0 ; \dots ; c_{n-1}]$ et qui renvoie un `string_builder` représentant le mot $[c_i ; \dots ; c_{i+m-1}]$, c'est-à-dire la sous-chaîne de c débutant au caractère i et de longueur m .

note : On supposera $0 \leq i < i + m \leq n$, et on s'attachera à réutiliser dans le `string_builder` résultant autant de sous-arbres de sb que possible.

2.2 Équilibrage

Le hasard des concaténations peut amener un `string_builder` à se retrouver déséquilibré, c'est-à-dire à avoir certaines de ses feuilles très éloignées de la racine et donc d'accès plus coûteux. Le but de cette partie est d'étudier une stratégie de rééquilibrage *a posteriori*. Considérons un `string_builder` sb composé de $k + 1$ feuilles, et donc de k noeuds internes. Notons ces $k + 1$ feuilles m_0, \dots, m_k . lorsqu'on les considère de la gauche vers la droite, si bien que sb représente le mot $m_0 m_1 \dots m_k$. La profondeur de la feuille m_i dans sb est notée $\text{depth}(m_i)$ et est définie comme la distance de m_i à la racine de sb .

Voici un exemple de `string_builder` pour $k = 3$ où la profondeur de chaque feuille est indiquée entre parenthèses.



Le coût de l'accès à un caractère de la feuille m_i est défini comme la profondeur de cette feuille dans sb , soit $\text{depth}(m_i)$ (on ne considère pas le coût de l'accès dans le mot m_i lui-même). Le coût total d'un `string_builder` est alors défini comme la somme des coûts d'accès à tous ses caractères et vaut donc

$$C(sb) = \sum_{i=0}^k \text{length}(m_i) \times \text{depth}(m_i).$$

Question 4. Définir la fonction `cost` qui prend en argument un `string_builder` et qui renvoie son coût selon la définition précédente.

Question 5. Définir une fonction `random_string` qui prend en argument un entier i et qui génère un arbre de profondeur i . Vous présenterez l'algorithme utilisé pour la génération dans votre rapport, et vous discuterez de la pertinence de vos choix.

note : Pensez à consulter la documentation en ligne des modules `Random`, `Char` et `String`.

Question 6. Définir une fonction `list_of_string` qui prend en argument un `string_builder` et qui renvoie la liste des chaînes de caractères dans le même ordre que dans l'arbre (parcours infixe).

On propose l'algorithme d'équilibrage suivant :

- Transformer l'arbre en une liste de ses feuilles respectant l'ordre.
- Tant qu'il existe au moins 2 éléments dans la liste :
 1. trouver les deux éléments successifs dont la concaténation a le coût le plus faible,
 2. les retirer, les concaténer et insérer le résultat à leur position.
- Concaténer les deux derniers éléments.

Question 7. Définir une fonction `balance` qui prend en argument un `string_builder` et qui renvoie un nouveau `string_builder` équilibré selon l'algorithme précédent.

Question 8. Proposer une fonction qui calcule les gains (ou les pertes) en coût de la fonction `balance` sur un grand nombre d'arbres générés aléatoirement.

La fonction peut, entre autre, renvoyer le min, le max, la moyenne et la valeur médiane.