Projet de Programmation Fonctionnelle

Rendu et critères d'évaluation

Vous soumettrez sur Célène une archive au format **zip** (et uniquement ce format) ne contenant pas de fichiers compilés (exécutez `dune clean` avant d'archiver). Vous rajouterez au même niveau que `README.md` un fichier `GROUPE.md` qui contiendra la liste des membres de votre groupe (2 ou 3 étudiants). L'archive devra être nommée avec les noms des membres du projet séparés par des tirets, par exemple `NOM1-NOM2-NOM3.zip`. Il est demandé de ne changer aucun nom de fichier et d'utiliser uniquement les fichiers fournis. Ces consignes sont à respecter **strictement**. Tout rendu non conforme se verra attribuer la note **0**. Une soutenance **individuelle** aura lieu lors de la semaine des examens, selon un calendrier qui vous sera communiqué ultérieurement.

Le contexte. L'objectif de ce projet est de concevoir une structure de données représentant un tableau de n entiers sous forme d'arbre binaire et permettant de répondre *efficacement* à une requête donnée sur ce tableau, comme par exemple calculer la somme contenue dans un sous-tableau, la plus grande somme contenue dans un sous-tableau ou encore connaître le nombre de zéros dans un sous-tableau.

Dans l'arbre binaire, chaque nœud correspond à un intervalle du tableau et stocke une information permettant de répondre facilement à une requête donnée. Ces informations peuvent être de diverses formes mais auront toujours le point commun suivant : **les nœuds contiendront dans tous les cas la valeur des bornes gauche et droite des intervalles qu'ils représentent.** La racine correspond ainsi à l'intervalle $[0 \dots n-1]$ du tableau et les feuilles aux intervalles $[i \dots i]$ pour $0 \leqslant i \leqslant n-1$. L'arbre est ensuite construit récursivement en **combinant** les valeurs des deux nœuds enfants pour obtenir le nœud parent.

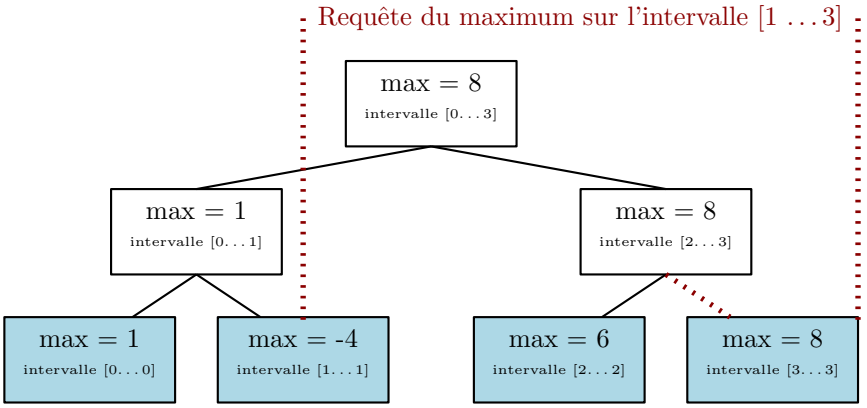
Structure et requêtes. Cette structure de données doit supporter les trois primitives suivantes :

- `create`, qui permet d'initialiser la structure par rapport à une liste d'entiers ;
- `update`, qui permet de mettre à jour une valeur dans la liste (et qui met à jour la structure en conséquence) ;
- `query`, qui permet de réaliser une **requête** dans la structure.

Les requêtes étant de natures différentes, une implémentation différente semble nécessaire pour chaque type de requête souhaité. En particulier, la manière de combiner deux nœuds devra être adaptée pour chaque type de requête. Cependant, afin de ne pas avoir à créer une structure différente pour chaque type de requête, une implémentation **générique** sera attendue et devra dépendre d'un module représentant les nœuds. **Un squelette de code est fourni dans la partie suivante (et sur Célène) pour permettre de répondre à ces besoins.**

Structure pour calculer pour le maximum de n'importe quel intervalle du tableau

Dans ce cas simple, chaque nœud de l'arbre contiendra une unique information (en plus des bornes gauche et droite de l'intervalle qu'il représente) : la **valeur maximum** contenue dans le tableau correspondant à l'intervalle qu'il représente. Les feuilles contiendront donc uniquement l'entier de la case correspondante et les nœuds internes seront eux obtenus par combinaison de leurs deux enfants, en gardant uniquement la valeur maximum (voir ci-dessous pour la liste $[1; -4; 6; 8]$).



Requête pour le maximum de n'importe quel intervalle du tableau

Le maximum d'un intervalle donné peut s'obtenir récursivement :
- si l'intervalle est de taille 1, le maximum est l'élément correspondant
- si l'intervalle est à droite *ou* à gauche du nœud courant, la requête est calculée récursivement
- si l'intervalle *traverse* les intervalles gauche et droit du nœud courant, comme illustré sur l'exemple où est demandé le maximum du sous-tableau entre les indices 1 et 3, le maximum est calculé récursivement en prenant le maximum entre la requête gauche et la requête droite.

Implémentation. Les trois primitives mentionnées sont indépendantes des informations stockées dans les nœuds et peuvent donc être implémentées en considérant un type abstrait pour les nœuds. Une signature pour ce type abstrait est disponible sur `Célène` (fichier `.mli`) et contient le code suivant :

```
module type QUERY_STRUCTURE = sig
  type data = int
  (** Le type des données sur lesquelles sont faites les requêtes *)

  type answer
  (** Le type des réponses aux requêtes *)

  type tree
  (** Le type Arbre, qui sera défini comme un arbre binaire *)

  val create : data list -> tree
  (** Création de l'arbre *)

  val update : tree -> data -> int -> tree
  (** Mise à jour d'un élément de l'arbre *)

  val query : tree -> int -> int -> answer
  (** [query t l r] est la requête pour obtenir la
      valeur pour un intervalle [l... r] *)

  val to_string : answer -> string
  (** Pour l'affichage des réponses *)
end
```

Une deuxième signature est fournie pour les nœuds de l'arbre, ces derniers devant être **créés** et **combinés**.

```
module type NODE = sig
  type data = int
  (** Le type des données sur lesquelles sont faites les requêtes *)

  type answer
  (** Le type des réponses aux requêtes *)

  type node = { answer : answer; left : int; right : int }
  (** Les nœuds contiennent des données et les bornes gauche et droite
      de l'intervalle qu'ils représentent *)

  val create : data -> answer
  (** Création d'une réponse à partir d'une unique valeur *)

  val combine : node -> node -> node
  (** Nœud obtenu par la combinaison de deux autres nœuds *)

  val to_string : answer -> string
  (** Pour l'affichage des réponses *)
end
```

Enfin, le `type tree` sera concrètement représenté par un arbre binaire, défini comme suit :

```
type 'a binary_tree =
  | Leaf of { node : 'a }
  | Node of {
      node : 'a;
      left_child : 'a binary_tree;
      right_child : 'a binary_tree;
    }
```

Les questions

Hypothèse sur le nombre d'éléments du tableau

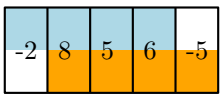
Pour simplifier la gestion de la structure on suppose que le nombre d'éléments n du tableau est toujours une puissance de 2. De plus, nous ne considérerons jamais d'arbre vide.

1. Compléter l'implémentation du module `QUERY_STRUCTURE`.
2. Compléter le module `NodeSum` respectant la signature de `Node` et permettant de réaliser des requêtes calculant la somme des éléments d'un tableau, pour un intervalle donné.
3. Compléter le module `NodeMaxOcc` respectant la signature de `Node` et permettant de réaliser des requêtes calculant la valeur maximum **et** son nombre d'occurences d'un intervalle donné du tableau.
4. Compléter le module `NodeMaxSubseg` respectant la signature de `Node` et permettant de réaliser des requêtes calculant la **plus grande somme d'éléments contigüs** pour un intervalle donné du tableau. Par exemple dans le tableau $[1; 3; -2; 8; 5; 6; -5; 1]$ la plus grande somme dans l'intervalle $[2 \dots 6]$ est 19, qui correspond à la somme de l'intervalle $[3 \dots 5]$. Une aide pour la résolution est donnée ci-dessous.

Les informations à stocker

Cette requête étant assez difficile, voici une aide sur les informations à stocker dans chaque nœud :
- `sum` : la somme de l'intervalle représenté par le nœud
- `prefix` : la valeur maximum d'un *préfixe* du tableau, qui contient la borne gauche de l'intervalle
- `suffix` : la valeur maximum d'un *suffixe* du tableau, qui contient la borne droite de l'intervalle
- `answer` : la réponse à la requête pour l'intervalle correspondant

Les informations pour un nœud donné peuvent être obtenues en **combinant** ces valeurs calculées sur les deux enfants. Dans l'exemple, les différentes valeurs obtenues sont les suivantes, l'intervalle du préfixe étant représenté en bleu et celui du suffixe en orange.



`sum` $= 12$, `prefix` $= 17$, `suffix` $= 14$, `answer` $= 19$

Gestion de projet et tests

Squelette. Il est demandé d'utiliser l'outil `dune` pour réaliser ce projet (voir le `CM7` sur les modules). Le squelette du projet est entièrement donné sur `Célène`, avec un fichier `README.md` expliquant comment utiliser l'outil `dune`.

Tests. Des tests sont disponibles pour valider (au moins partiellement) vos méthodes. **Le fichier de test ne doit évidemment pas être modifié.** Pour que ce dernier fonctionne, il est attendu que vos méthodes `to_string` respecte les tests effectués dans le fichier, à savoir :
- `NodeSum` : la valeur de la somme est affichée
- `NodeMaxOcc` : la valeur maximum `max` et le nombre d'occurences `occ` sont affichés sous la forme (`max`, `occ`).
- `NodeMaxSubseg` : la valeur maximum est affichée.