# 1 Programme de l'année

Le programme de l'année s'articule autour de 4 points :
- les arbres
- les graphes
- les expressions régulières et les automates finis
- le calcul propositionnel du premier ordre

# 2 Les bases : révisions

## 2.1 Les tableaux et les boucles

Tri par insertion (dans un nouveau tableau ou en place) : écriture du code, calcul de complexité et preuve de correction.

## 2.2 Les listes et la récursivité

Tri rapide, tri fusion : même travail.

## 2.3 Les principaux types abstraits

Un type abstrait est la description d'un conteneur d'objets qui suivent certaines règles de fonctionnement : pour un même type abstrait, il peut y avoir plusieurs réalisations concrètes différentes pourvu qu'elles respectent les règles.

Un type abstrait est dit mutable si une instance du type peut être modifiée en place (comme les tableaux), persistant (ou immuable, comme les listes) si pour modifier une instance du type, il faut en créer une nouvelle.

En général, on essaye de faire en sorte que les complexités des principales fonctions sur un type abstrait soient les plus faibles possibles. Quand il y a plusieurs réalisations concrètes, souvent les qualités de l'une sont les défauts des autres, alors on doit arbitrer les choix.

### a) Les piles

Une pile (Stack) est un type d'objet initialement vide et qui permet d'ajouter des éléments et de les retirer dans l'ordre inverse de leur insertion (Last In First Out).

On peut faire en sorte que ces opérations soient de complexité constante.

Principales fonctions :

type mutable

```
new : unit -> 'a pile ;;
push : 'a -> 'a pile -> unit ;;
pop : 'a pile -> 'a ;;
empty : 'a pile -> bool ;;
```

type persistant

```
new : unit -> 'a pile ;;
push : 'a -> 'a pile -> 'a pile ;;
pop : 'a pile -> 'a * 'a pile ;;
empty : 'a pile -> bool ;;
```

### b) Les files

Une file (Queue) est un type d'objet initialement vide et qui permet d'ajouter des éléments et de les retirer dans l'ordre d'insertion (First In First Out).

On peut faire en sorte que ces opérations soient de complexité constante ou constante en moyenne.

Principales fonctions :

type mutable

```
new : unit -> 'a file;;
add : 'a -> 'a file -> unit;;
take : 'a file -> 'a;;
empty : 'a file -> bool;;
```

type persistant

```
new : unit -> 'a file;;
add : 'a -> 'a file -> 'a file;;
take : 'a file -> 'a * 'a file;;
empty : 'a file -> bool;;
```

### c) Les files de priorité

Une file de priorité (Heap) est un type d'objet initialement vide et qui permet d'ajouter des éléments ordonnables selon leur importance et de les retirer dans l'ordre de leur importance.

On peut faire en sorte que ces opérations soient de complexité $O(\log n)$ où $n$ est le nombre d'éléments dans la file de priorité.

Principales fonctions : les mêmes que celles des files.

### d) Les dictionnaires

Un dictionnaire (Map ou Hashtbl) est un type d'objet initialement vide et qui permet d'ajouter ou de supprimer des valeurs repérés par des clefs.

Si l'ensemble des valeurs est muni d'un ordre, alors on peut faire en sorte que ces opérations soient de complexité $O(\log n)$ où $n$ est le nombre d'éléments dans le dictionnaire.

Si l'ensemble des valeurs n'a pas d'ordre, alors on se contente de complexité en $O(n)$.

Principales fonctions :

type mutable

```
new : unit -> ('a,'b) dict;;
add : 'a -> 'b -> ('a,'b) dict -> unit;;
find : 'a -> ('a,'b) dict -> 'b;;
remove : 'a -> ('a,'b) dict -> unit;;
```

type persistant

```
new : unit -> ('a,'b) dict;;
add : 'a -> 'b -> ('a,'b) dict -> ('a,'b) dict;;
find : 'a -> ('a,'b) dict -> 'b;;
remove : 'a -> ('a,'b) dict -> ('a,'b) dict;;
```