

## TP 10

## Présentation

Le but du TP est de montrer une application des *tries*, à savoir la résolution efficace du problème consistant à trouver toutes les mots pouvant être formés sur une grille de Boggle.

## 1 Fonctions auxiliaires

Dans cette première partie, on implémente les fonctions auxiliaires non vues en cours.

- 1. Compléter la fonction make\_grid : int -> string array qui crée une grille aléatoire. Une grille est un tableau de chaînes de caractères, chaque chaîne représentant une ligne de la grille. On pourra utiliser les fonctions suivantes :
  - Array.init : int -> (int -> 'a) -> 'a array la fonction prend en argument un entier n (la taille du tableau à créer) et une fonction appelée successivement sur 0, 1, ..., n-1 qui doit renvoyer la valeur à mettre dans chacune des cases du tableau.
  - String.init : int -> (int -> char) -> string qui fonctionne de façon similaire, mais la fonction passée en argument doit créer un caractère
  - Char.chr: int -> char crée un caractère dont le code ASCII est donné, l'entier devant être entre 0 et 255. Le code ASCII du caractère 'A' est 65
  - Random.int : int -> int qui attend un entier n et renvoie un entier aléatoire choisit entre 0 (inclus) et n exclu.
- 2. Compléter la fonction print\_grid : grid -> unit qui affiche la grille sur la sortie standard. On pourra utiliser une boucle for i = 0 to ... do ... done ou la fonction

similaire à List.iter

- 3. Compléter la fonction load\_dico : string -> unit Trie.trie qui prend en argument un nom de fichier et renvoie un trie. Deux points importants :
  - les fonctions relatives aux *trie* sont dans le fichier trie.ml, il faut donc les appeler avec le nom de module, par exemple Trie.add ou Trie.empty. De même si on souhaite mentionner le constructeur Node d'un *trie*, il faut écrire Trie.Node
  - on se sert d'un *trie* comme d'un ensemble, donc il n'y a que les clés qui nous intéressent. On peut donc associer une valeur bidon à chaque clé. On utilise ici la valeur () (unit).
- 4. Compléter la fonction print\_word : char list -> unit pour qu'elle affiche le mot donné en paramètre dans la console. Attention le mot est formé de la liste des caractères dans l'ordre inverse. La fonction Trie.rev\_implode du TP précédent permet de reconstituer la chaîne.
- 5. Lire le code des fonctions solve\_list et solve\_node (il n'y a pas de code à écrire dans cette question). La seule différence par rapport à la version présentée en cours est que la fonction solve\_list est donnée en premier (c'est juste une question de présentation), car il est plus facile de comprendre la récursion si on « part » de cette fonction.

Étant donnée une grille **grid**, une première case **(row,col)**, une liste **1** de couples (caractère, *trie*), une liste de lettre déjà ajoutées **words** et un ensemble des positions déjà visitées, alors :

- On cherche s'il y a un *trie* dans 1 associé au caractère se trouvant dans grid.(row).[col]
- S'il n'y en a pas, on ne fait rien
- S'il y en a un, c'est qu'on peut continuer le mot courant, donc on ajoute le caractère au mot qu'on est en train de former et on appelle **solve\_node** sur le *trie*

Dans **solve\_node**, si le nœud sur lequel on est est *terminal*, alors on peut afficher le mot formé jusque là. Ensuite on ajoute la case courante à l'ensemble des cases déjà visitées, puis on peut continuer récursivement la recherche dans toutes les cases voisines de la case courante qui n'ont pas déjà été visitées.

La fonction main donnée plus bas se contente juste d'appler solve\_list sur la liste des (cararctères, trie) de la racine du trie, pour chaque lettre de la grille.

- 6. Compléter la fonction **next\_positions** pour qu'elle renvoie la liste des positions (ligne, colonne) des voisins non-visités d'une case donnée pour la grille donnée.
- 7. Exécuter le programme **boggle.exe** ce dernier crée une petite grille 5x5 et affiche tous les mots de cette grille.

## 2 Améliorations

Dans cette section on essaye d'illustrer que le mélange entre ordre-supérieur et effets de bords peut être utile.

- 1. modifier les fonctions solve\_list et solve\_node pour qu'elle prennent en premier argument une fonction f : char list -> unit et utiliser cette fonction f à la place de print\_word dans le code de solve\_node. Modifier le main pour passer print\_word comme argument à solve\_list et vérifier que le programme fonctionne toujours.
- 2. Sans modifier la fonction solve\_node, faire en sorte que votre programme n'affiche plus tous les mots mais uniquement le nombre de mots trouvés. Vous avez le droit de modifier le main comme vous le souhaitez.
- 3. Augmenter la taille de la grille (par exemple 20 par 20) et constater que votre solution fonctionne toujours dans un temps raisonnable.
- 4. Étendez la solution précédente pour faire en sorte que votre programme respecte les vraies règles du Boggle :
  - les mots doivent faire au moins trois lettre
  - les mots ne peuvent compter qu'une seule fois, même s'ils sont trouvés de deux façons différentes
- 5. modifier les fonctions solve\_node et solve\_list pour accumuler dans words non seulement les caractères mais leur positions dans la grille. Faire en sorte que votre programme affiche une animation dans la console, des mots trouvés dans la grille. On pourra utiliser les séquences AINSI suivantes :

```
let green = "\x1b[102m"
let reset = "\x1b[0m"
let clear_screen = "\x1b[2J\x1b[H"
```

La première fait en sorte que tous les caractères affichés dans la console sont sur fond vert. La seconde rétablit la couleur d'affichage normale. La troisième efface l'écran.