TP12 : Erreurs en OCaml

Objectifs du TP :

- Lire et comprendre du code en OCaml.
- Identifier, comprendre et corriger des erreurs dans ce langage.
- Réviser quelques structures de données : arbres, graphes, files de priorité.

Commencez par copier le répertoire TP_erreurs_OCaml dans votre espace personnel.

1. Évaluer une à une les lignes de 1_arbres_types.ml. Modifier le type `'a arbre` et l'arbre `a` de sorte à remplacer le constructeur `Vide` par `V`. Réévaluer, commenter et résoudre le problème.
2. Pour chacune des fonctions dans 2_arbres.ml :
   a) Inférer sa spécification à l'aide des éléments à disposition.
   b) Tant qu'elle ne s'évalue pas sans erreur ni warning : l'évaluer, lire et comprendre le message d'erreur et corriger **uniquement** l'erreur indiquée.
3. On considère à présent les fonctions du fichier 3_graphes.ml :
   a) Leur appliquer le même traitement qu'à la question 2.
   b) Réécrire `transposer` en utilisant `List.iter`.
   c) Modifier `parcours` de sorte à effectuer un parcours en profondeur de l'ensemble du graphe.
4. Pour chacune des fonctions du fichier 4_evaluations.ml :
   a) Évaluer la fonction. Y a-t-il un problème ?
   b) Évaluer le test accompagnant la fonction. Expliquer et corriger.
5. a) Appliquer le même traitement à la fonction de 5_filtre.ml qu'à la question 2.
   b) Réécrire la fonction `filtrer` à l'aide de `List.filter`.
6. a) Appliquer le traitement de la question 2 à la fonction de 6_lecture.ml.
   b) Vérifier le bon fonctionnement de la fonction `lire_lignes` corrigée. Est-on satisfait ?
   c) Réécrire `lire_lignes` de manière récursive.
7. On observe les fonctions de 7_file_prio.ml. Une file de priorité y est implémentée à l'aide d'un tas min décomposé en deux morceaux : les clés et leurs priorités. Si une clé est en position i dans `cles`, sa priorité est en position i dans `priorites`. La partie "active" des tableaux est située entre les indices 0 et `fin` inclus. Les tableaux `cles` et `priorites` sont de même taille, fixée à la création de la file.
   a) Pour chacune des fonctions, appliquer le traitement décrit en question 2 et vérifier que le comportement des fonctions corrigées est cohérent avec les spécifications inférées sur l'exemple proposé.
   b) Écrire une fonction `extraire_min` renvoyant l'élément de priorité minimal dans une file non vide et modifiant cette dernière de sorte à le supprimer.

Bonus

On cherche à implémenter l'algorithme de Dijkstra sur un graphe $G = (S, A)$ représenté par listes d'adjacence et dont les sommets sont numérotés de 0 à $|S| - 1$. Pour ce faire, on reprend le fichier file_prio.ml et on modifie le type `file_prio` en y ajoutant un champ `emplacements` tel que :

- Si la clé (sommet ici) `i` n'est pas dans la file, `emplacements.(i) = -1`.
- Sinon, `cles.(emplacements.(i)) = i` et la priorité correspondante est dans la case `emplacements.(i)` de `priorites`.

Autrement dit, `emplacements` permet de retrouver l'emplacement d'une clé dans le tas ce qui est nécessaire pour pouvoir en modifier facilement la priorité.

8. Écrire une fonction `diminuer_prio : file_prio -> int -> float -> unit` telle que `diminuer_prio f s p` modifie la priorité de `s` dans `f` en la remplaçant par `p` si `p` est inférieure à la priorité actuelle de `s`.
9. En déduire une fonction `dijkstra` prenant en entrée un graphe G et un sommet s de G et déterminant pour tout sommet u la distance minimale de s vers u.