IRESI : Algorithme Misra-Gries

François Godi Xavier Montillet

Pour le 3 décembre 2013

1 Principe de l'algorithme

1.1 Objectif

Cet algorithme a pour but de déterminer quels sont les éléments les plus fréquents d'un flux de données en temps réel, c'est à dire en temps linéaire par rapport à m, le nombre d'éléments du flux. Pour ce faire il n'utilise que k compteurs et, puisque dans les faits $k \ll m$, l'algorithme est économe en mémoire.

1.2 Fonctionnement

Le fonctionnement de l'algorithme est le suivant :

1. On commence par initialiser k compteurs vides.
2. Pour chaque élément x du flux :
   - Si il y a déjà, parmi les k compteurs, un compteur associé à x alors on incrémente ce compteur.
   - Sinon, si au moins un des k compteurs est vide, on associe x à l'un des compteurs vides et on positionne ce dernier à 1 pour compter l'élément courant du flux.
   - Enfin, si les k compteurs sont déjà associés à des éléments du flux tous différents de x, on décrémente tous les compteurs. Un compteur devenant nul n'est plus associé à aucun élément du flux, il redevient un compteur vide.
3. À la fin de la mesure, les compteurs contiennent nécessairement les valeurs les plus fréquentes du flux, sous certaines conditions.

1.3 Conditions de validité

Tout élément j du flux dont la fréquence a été strictement plus grande que m/k, où m est le nombre d'éléments du flux et k le nombre de compteurs utilisés, sera nécessairement présent dans l'un des compteurs à la fin de la mesure. Pour les éléments du flux dont la fréquence a été inférieure à m/k, on ne peut rien dire avec certitude mais, si le flux n'est pas pathologique, plus leur fréquence

était grande plus ils ont de chances de se trouver dans l'un des compteurs à la fin .

2 Implémentation OCaml

2.1 Choix du langage

Nous avons choisi d'implémenter cet algorithme en OCaml car nous connaissons bien ce langage et nous savions qu'il se prête bien à l'implémentation d'un algorithme comme celui ci. Notre code est robuste et peut toujours fonctionner si la nature des éléments du flux change, par exemple.

2.2 Choix d'implémentation

Un flux d'éléments de type α est modélisé par un type α enum qui, outre une fonction d'initialisation, ne possède qu'une fonction *next* qui renvoie l'élément suivant du flux :

```
type 'a enum =
  {
    reset: unit -> unit;
    next: unit -> 'a option
  }
```

Notre itérateur permet d'effectuer un traitement quelconque pour chaque élément du flux :

```
let iter f enum =
  let rec aux () =
    match enum.next () with
      | Some x -> begin
        f x;
        aux ()
      end
      | None -> ()
  in
  aux ()
;;
```

En particulier, incrémenter et décrémenter les compteurs selon les éléments du flux. Pour plus de détails sur l'implémentation, se reporter au code source.

3 Résultats