Lycée Chateaubriand
Informatique MP 2002/03
Mathieu Giraud

TP Caml n° 3 – Corrigé
Dessin d'arbres n-aires

1 Le partage aveugle

a. On commence par écrire `place_aux`. Le premier fils se place à la position `position`, les suivants à partir de `position +. delta`.

```
let rec place_aux position delta fils = match fils with
  | [] -> []
  | NoeudPlace(n,_,petitfils)::q ->
       NoeudPlace(n, position, petitfils)
::(place_aux (position +. delta) delta q) ;;
```

Pour réaliser `place`, il faut maintenant déterminer la position du fils le plus à gauche à donner à la fonction `place_aux`. Si les n fils sont disposés autour de 0, séparés deux à deux par une distance δ, alors le fils le plus à gauche est à la position $-\delta(n-1)/2$.

```
let place delta l =
  let pp = -. (float_of_int ((list_length l) - 1)) *. delta /. 2.
  in  place_aux pp delta l ;;
```

La fonction `place` effectue $\mathcal{O}(p)$ opérations, où p est le nombre de sous-arbres dans la liste.

b.

```
let rec x_ifie_aux delta = function
  | Noeud (n,[]) -> NoeudPlace (n, 0., [])
  | Noeud (n,fils) ->
        let fils2 = map (x_ifie_aux  (delta /. (float_of_int (list_length fils)))) fils
        in NoeudPlace(n, 0.,  (place delta fils2)) ;;

let x_ifie = x_ifie_aux 8. ;;
```

Si l'arbre contient n noeuds, la complexitié totale de `x_ifie` est en $\mathcal{O}(n)$.

2 Les franges

c.

```
let rec distance_franges g d = match (g,d) with
  | [], _ -> 0.
  | _, [] -> 0.
  | gt::gq, dt::dq -> max (1. +. gt -. dt) (distance_franges gq dq) ;;
```

d. La fonction `trouve_delta` doit simplement appeler `distance_franges` sur chaque paire de franges. La difficulté est d'exhiber ces paires de franges qui sont à chaque fois composées de "la frange droite d'un sous-arbre" et de "la frange gauche de l'arbre suivant". J'ai ici utilisé une petite fonction auxiliaire `renvoie_fg` qui renvoie cette dernière, mais il y a bien sûr d'autres solutions.

```
let renvoie_fg = function
  | [] -> failwith "renvoie_fg"
  | Franges(fg,_,_)::_ -> fg ;;

let rec trouve_delta = function
  | [] -> failwith "distance_liste_franges"
  | [_] -> 0.
  | Franges(_,_,fd)::q -> max (distance_franges fd (renvoie_fg q))
                              (trouve_delta q) ;;
```

e. Le code de `place_aux` doit maintenant "décaler" les franges pour qu'elles soient relatives à la nouvelle racine. On décale ainsi chaque valeur de chaque frange `f` par l'appel `map (fun x -> x +. position) f`.

```
let rec place_aux_2 position delta fils = match fils with
  | [] -> []
  | Franges(fg,NoeudPlace(n,_,petitfils),fd)::q -> Franges(
                     (map (fun x -> x +. position) fg),
                     (NoeudPlace(n, position,petitfils)),
                     (map (fun x -> x +. position) fd)
                  )::(place_aux_2 (position +. delta) delta q) ;;

let place_2 delta l =
  let pp = -. (float_of_int ((list_length l) - 1)) *. delta /. 2.
  in  place_aux_2 pp delta l ;;
```

f.

```
let rec complete_frange g d =  match (g,d) with
  | [], _ -> d
  | _, [] -> g
  | gt::gq, _::dq -> gt::(complete_frange gq dq) ;;
```

Pour fusionner les franges gauches, on part de celle là plus à gauche (et donc la première dans la liste des sous-arbre) puis on la "complète" progressivement par les franges suivantes.

```
let rec ffg_aux fgmax = function
  | [] -> fgmax
  | Franges(fg,_,fd)::q -> ffg_aux (complete_frange fgmax fg) q ;;

let fusionne_franges_gauches = function
  | [] -> failwith "fusionne_franges_gauches"
  | Franges(fg,NoeudPlace(_,position,_),_)::q -> position::(ffg_aux fg q);;
```

La fusion des franges droites se fait similairement... mais en partant de la frange là plus à droite. Pour simplifier, nous supposons ici qu'on dispose de la fonction `miroir` du TP 1.

```
let fusionne_franges_droites l =
  fusionne_franges_gauches (map (fun (Franges(a,b,c)) -> Franges(c,b,a))
                           (miroir l)) ;;
```

g. La nouvelle fonction `x_ifie` s'écrit alors facilement :

```
let noeuds = map (fun (Franges(_,n,_)) -> n ) ;;

let rec x_ifie_aux_2 = function
  | Noeud(n,[]) -> Franges([0.], NoeudPlace(n,0.,[]) ,[0.])
  | Noeud(n,fils) ->
      let fils2 = map x_ifie_aux_2 fils
      in let delta = trouve_delta fils2
      in let fils3 = place_2 delta fils2
      in Franges(
                  (fusionne_franges_gauches fils3),
                  (NoeudPlace(n,0., (noeuds fils3))),
                  (fusionne_franges_droites fils3)
                 );;

let x_ifie_2 a =
  let (Franges(_,ap,_)) = x_ifie_aux_2 a
  in ap ;;
```

h.* La complexité totale de l'algortihme par franges est plus importante qu'en première partie. En particulier, la fusion des franges à chaque étape de la récursion implique une complexité au pire au moins quadratique.

Remarque : on peut aussi ne pas utiliser `trouve_delta` et de serrer tous les sous-arbres au maximum permi par `distance_franges` pour que l'arbre final soit le plus compact possible... mais on ne respecte alors plus l'équidistance des fils.

La suite...

- Au prochain TP, parcours de graphes.
- Après Noël, expressions régulières et automates.