IFT2015 automne 2016 - Devoir 5

Philippe Caron

6 décembre 2016

5.1 Structure de données pour les clades

Afin de pouvoir effectivement programmer les algorithmes, une version minimaliste d'arbre binaire a été implémentée (code joint au travail). Simplement, l'arbre est composé de `Node`s qui possède chacune soit deux enfants (nœuds internes), deux nœuds externes, ou chacun des deux. On peut accéder aux enfants gauche et droite grace à `.left()` et `.right()` dans le cas interne, et `.lex()` et `.rex()`. En prenant compte de cette structures les algorithmes suivant ont été développés et testé (fonctionnels).

a. Remplissage de `idx`

Voici l'algorithme en Java pour remplir le tableau `idx` en assumant que les clefs sont connues d'avance et que leur ordre est défini dans le tableau `key` :

```
public void fillIDX() {
   key = new ArrayList<T>();
   fillKey(root);
   try {
      Collections.sort((List) key);
   } catch (Exception e) {
      return;
   }
   idx = new int[key.size()];
   fillIDX(root, 0);
}

    public int fillIDX(Node<T> s, int n) {
   if (s.left() == null) {
      idx[key.indexOf(s.lex())] = n;
      if (s.right() == null) {
         idx[key.indexOf(s.rex())] = n + 1;
         return n + 2;
      } else {
         return fillIDX(s.right(), n + 1);
      }
   } else if (s.right() == null) {
      return (idx[key.indexOf(s.rex())] = fillIDX(s.left(), n)) + 1;
   } else {
      return fillIDX(s.right(), fillIDX(s.left(), n));
   }
}
```

b. Calcul de L, R, N

Pour calculer ces fonctions, on peut définir trois fonctions récursives très simple :

```
    public int L(Node<T> s) {
  if (s.left() == null)
    return idx[key.indexOf(s.lex())];
  return L(s.left());
}

public int R(Node<T> s) {
  if (s.right() == null)
    return idx[key.indexOf(s.rex())];
  return R(s.right());
}

public int N(Node<T> s) {
  return R(s) - L(s) + 1;
}
```

Cependant puisqu'il faudra les utiliser lors du remplissage de H, on peut les intégrer à la fonction en retournant un triple (L, R, N) ;

```
public void fillH() {
  fillIDX();
  H = new int[key.size()][2];
  fillH(root, true);
}

public int[] fillH(Node<T> s, boolean first) {
  if (s.left() == null) {
    int l = idx[key.indexOf(s.lex())];
    if (s.right() == null) {
      int r = idx[key.indexOf(s.rex())];

      if (first)
        H[l] = new int[] { l, r };
      else
        H[r] = new int[] { l, r };

      return new int[] { l, r, 2 };
    }
    int[] r = fillH(s.right(), true);

    if (first)
      H[l] = new int[] { l, r[1] };
    else
      H[r[1]] = new int[] { l, r[1] };

    return new int[] { l, r[1], r[2] + 1 };
  } else if (s.right() == null) {
    int r = idx[key.indexOf(s.rex())];
    int[] l = fillH(s.left(), false);

    if (first)
      H[l[0]] = new int[] { l[0], r };
    else
      H[r] = new int[] { l[0], r };

    return new int[] { l[0], r, l[2] + 1 };
  } else {
    int[] l = fillH(s.left(), false);
```

```
        int[] r = fillH(s.right(), true);

        if (first)
           H[l[0]] = new int[] { l[0], r[1] };
        else
           H[r[1]] = new int[] { l[0], r[1] };

        return new int[] { l[0], r[1], l[2] + r[2] };
     }
  }
```

5.2 Comparaison de deux arbres

On est certain que le remplissage de H sera correct en tout temps. L'intervale de la racine (disons r d'un arbre donné est toujours équivalent à l'indice de sa feuille la plus à gauche (g) jusqu'à l'indice de sa feuille la plus à droite (d). On dit que $H[g]$ contient l'intervale $[g, d]$, ceci est toujours vraie car si on parle de la racine, H est vide.

On peut déjà prouver qu'il est impossible qu'il y aie une collision avec n'importe quel enfant de droite de la racine. En effet, puisque la position est encodée par l'indice de gauche, il est impossible qu'un enfant de droite puisse atteindre la feuille la plus à gauche de sont parent.

Évidemment, les enfants de gauche ne peuvent pas êtres encodés par la feuille de gauche car ils la partage tous. Il sont donc encodés par la feuille de droite. Analogiquement à la preuve du paragraphe ci-haut, il est évident que cet encodage est unique pour tous les enfants de gauche puisqu'aucun ne peut partager la même feuille d'extrême droite.

Il reste à prouver qu'il n'y a aucune collision entre les indices de gauche et les indices de droite. Il y a 4 cas de collision possible, celle-ci sont droite-droite, gauche-gauche, droite-gauche, gauche-droite. Les deux premiers sont prouvés, il ne nous restes que les deux autres. Les deux cas sont similaires, autrement dit, un indice g équivaut à un indice d déjà défini dans le tableau H. Cela voudrait dire que la feuille la plus à gauche d'un sous-arbre serait la même que la feuille la plus à droite d'un arbre précédemment évalué, ou vice-versa. Or ceci est impossible puisque cela signifirait que cette feuille a deux parents, et on ne supporte pas plusieurs parent dans cette implémentation de l'arbre binaire. On est donc sur que ce système de remplissage est sécuritaire.