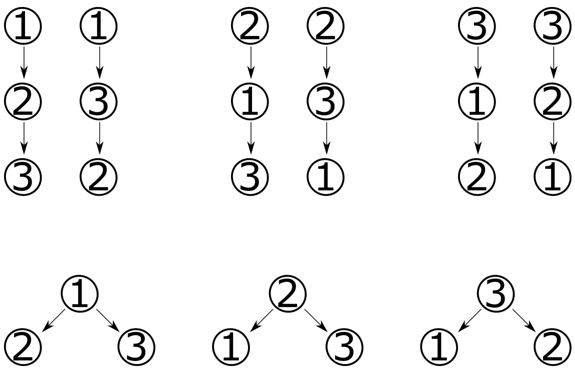
Option informatique: Devoir maison (Toussaint)

Benjamin LOISON (MP*)

22 octobre 2019

1 Première partie

1.17



1.18

```
let calculer_peres racine fils freres =
    let peres = Array.make (Array.length fils) (-1) in
        let rec aux indice father =
            let child = fils.(indice) and brother = freres.(indice) in
            if (child <> -1) then (peres.(child) <- indice; aux child indice);
            if (brother <> -1) then (peres.(brother) <- father; aux brother father);
        in aux racine 0;
    peres;;
```

1.19

La fonction calculer_peres est linéaire en le nombre de noeuds de l'arbre considéré.

1.20

```
let calculer_arites fils freres =
    let rec aux indice = match freres.(indice) with
    | (-1) -> 1
    | f -> 1 + (aux f); in
    let n = Array.length fils in
        let arites = Array.make n 0 in
            for i = 0 to (n - 1) do
                if (fils.(i) <> (-1)) then
                    arites.(i) <- (aux fils.(i));
```

```
            done;
    arites;;
```

1.21

La fonction calculer_arites est linéaire en le nombre de noeuds de l'arbre considéré.

1.22

```
let inserer table nb d =
    let rec aux i j = match (i + j + 1) / 2 with
    | k when j < i ->
        for i = (max 0 (nb - 1)) to k do
            table.(i + 1) <- table.(i);
        done;
        table.(k) <- d;
    | k when table.(k) > d -> aux (k + 1) j
    | k -> aux i (k - 1)
    in aux 0 (nb - 1);
    nb + 1;;
```

1.23

La fonction inserer est linéaire en le nombre d'entiers contenus dans le tableau trié considéré.

1.24

Le codage de Prüfer de l'abre A_3 est: 9, 1, 6, 7, 1, 3, 7, 9, 9, 3.

1.25

```
let calculer_Prufer racine fils freres =
    let n = Array.length fils and
        peres = calculer_peres racine fils freres and
        arites = calculer_arites fils freres and
        feuillesIndex = ref 0 in
    let feuilles = Array.make n 0 and prufer = Array.make (n - 1) 0 in
    for i = n - 1 downto 0 do
        if arites.(i) = 0 then
        (   feuilles.(!feuillesIndex) <- i;
            incr feuillesIndex)
    done;
    for i = 0 to n - 2 do
        let noeud = feuilles.(!feuillesIndex - 1) in
        let pere = peres.(noeud) in
            decr feuillesIndex;
            prufer.(i) <- pere;
            arites.(pere) <- arites.(pere) - 1;
            if arites.(pere) = 0 then
                feuillesIndex := inserer feuilles !feuillesIndex pere;
    done;
    prufer;;
```

1.26

La fonction calculer_Prufer est quadratique en le nombre de noeuds de l'arbre considéré.

2 Seconde partie

2.27

Chaque occurence d'une étiquette dans le codage de Prüfer est synonyme de l'existence d'un fils de cette étiquette, d'où:

```
let calculer_arites_par_Prufer prufer =
    let n = Array.length prufer in
    let arites = Array.make (n + 1) 0 in
    for i = 0 to n - 1 do
        print_int i;
        let j = prufer.(i) in
        arites.(j) <- arites.(j) + 1
    done;
    arites;;
```

2.28

2 est le dernier élément du codage de Prüfer c'est donc la racine.

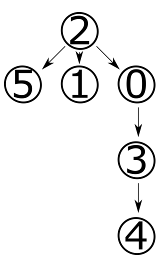
L'arbre étant consécutivement étiqueté et 1 n'apparaît pas dans le codage de Prüfer donc 1 est une feuille et est fils de 2.

0 est le plus petit élément de $[\![0;5]\!]$, l'unique occurence de 0 dans le codage de Prüfer et la présence de la séquence 0, 2 dans le codage de Prüfer montre que 0 est fils de 2.

5 et 4 sont des feuilles car ils n'apparaissent pas dans le codage de Prüfer.

3 est le père de 4 car 4 est la plus petite feuille parmi 4 et 5.

D'où l'arbre suivant:



2.29

On part donc de la liste des feuilles: 1, 6, 9, 12, 13.

Donc dans l'ordre:

1 est fils de 3.

6 est fils de 10.

9 est fils de 3 alors la liste des feuilles devient: 3, 10, 12, 13

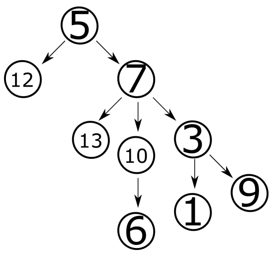
3 et 10 sont fils de 7.

12 est fils de 5.

13 est fils de 7.

7 est fils de 5.

D'où l'abre suivant:



2.30

2.31

2.32

Par bijection on a: card $A(E) =$ card $S(E) = n^{n-1}$ car cela revient à un tirage successifs avec remise de $n-1$ éléments dans un ensemble à n éléments.