Option informatique: Devoir maison (Noël)

Benjamin LOISON (MP*)

24 décembre 2019

1

```
let rec insere l e = match l with
| [] -> [e]
| t::q when e > t -> t::(insere q e)
| t::q when e <> t -> e::l
| t::q -> l;;
```

2

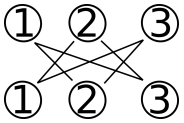
La fonction insere est linéaire en la taille de la liste l.

3

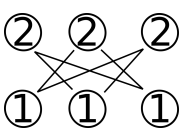
```
let voisins g s =
    let l = ref [] in
    let rec aux aMaj = match aMaj with
    | [] -> ()
    | t::q -> if t.a = s then l := insere !l t.b
              else if t.b = s then l := insere !l t.a;
              aux q;
    in aux g.aMaj;
    !l;;
```

4

Sur le graphe $Gex2$, on trouve:



Oui, il y a une bonne coloration de $Gex2$ avec moins de couleurs:



5

```
let coloration g =
    let t = Array.make g.n 0 in
    for i = 0 to (g.n - 1) do
        let v = voisins g i in
            let rec aux v b = match v with
            | [] -> b
            | a::q when t.(a) = 0 -> b
            | a::q when t.(a) < b -> aux q b
            | a::q -> aux q (t.(a) + 1)
            in t.(i) <- aux v 1;
    done;
    t;;
```

6

Pour l'existence, il existe forcément nbc(G), tel que fc(G, nbc(G)) $\neq 0$, par exemple:

on a fc(G, n(G)) $\neq 0$ d'où l'existence et on a directement $\forall p \geq$ nbc(G), fc(G, p) $\neq 0$

est != 0 car cela revient par récurrence à ne pas utiliser la nouvelle couleure

Pour l'unicité, on procède par l'absurde, supposons l'existence de nbc(G) et nbc'(G)

distincts, on a alors clairement min(nbc(G), nbc'(G)) qui

n'appartient pas à EC(G): contradiction.

7

S'il n'y a aucune arrête, alors tous les sommets peuvent avoir la même coloration donc nbc(G) = 1.

Ayant le choix du nombre pour la coloration pour chaque sommet parmi $[\![1; n]\!]$, on a donc fc(G, p) = $p^{n(G)}$

8

Tous les sommets sont reliés à tous les autres:

on a donc nbc(G) = n(G) et fc(G, p)

9

10

```
let prem_voisin g s =
    List.hd (voisins g s);;
```

11

```
let prem_ni g =
    let i = ref 0 and b = ref 0 in
    while !i < (g.n - 1) do
        if (List.length (voisins g !i)) <> 0 then (b := !i; i := g.n);
        incr i
    done;
    !b;;
```

12

```
let h g =
    let s1 = prem_ni g in
    let s2 = prem_voisin g s1 in
    let rec aux l = match l with
    | [] -> []
    | t::q when (t.a = s1 && t.b = s2) || (t.b = s1 && t.a = s2) -> aux q
    | t::q -> t::(aux q)
    in {n = g.n; aMaj = aux g.aMaj};;
```

13

```
let k g =
    let s1 = prem_ni g in
    let s2 = prem_voisin g s1 and hg = h g in
    let chg i = if i = s2 then s1 else (if i > s2 then i - 1 else i) in
    let rec aux l = match l with
    | [] -> []
    | t::q -> {a = chg t.a; b = chg t.b}::(aux q)
    in {n = g.n - 1; aMaj = aux hg.aMaj};;
```

14

15

16

17

18

```
let difference p q = (* TO KNOW: does array or list are like pointers in argument - only array are
    let r = Array.copy p in
    for i = 0 to (Array.length q) - 1 do
        r.(i) <- r.(i) - q.(i)
    done;
    r;;
```

19

20

```
let eval p x =
    let n = Array.length p in
    let res = ref p.(n - 1) in
    for i = 2 to n do
        res := x * (!res) + p.(n - i)
    done;
    !res;;
```

21