LE BIHAN
Erwan MP

Concours blanc
IPT

26,5/30

1

Réseau A [5, [[0,1],[1,2],[2,3],[3,0],
[0,3]]]

Réseau B [5, [[0,1],[1,2],[1,3],
[2,3],[3,4],[2,4]]] 1

2

```
def creerReseauVide(n):
    return [n, []]
```

9,5

3

```
def estUnLienEntre(paire, i, j):
    #return {*paire} == {i, j}
    return(paire[1] == i and paire[0] == j)
        or (paire[1] == j and paire[0] == i))
```

9,5

4

```
def sontAmis(reseau, i, j):
    for paire in reseau:
        if estUnLienEntre(paire, i, j): return True
    return False
```

1

La fonction effectue m appels à estUnAmiDans, qui effectue des opérations élémentaires à coût constant, la complexité est donc en $\mathcal{O}(m)$

99

5

```
def declareAmis(reseau, i, j):
    if not sontAmis(reseau, i, j):
        reseau[1].append([i, j])
```

99

6

declareAmis effectue un appel à sontAmis qui est en $\mathcal{O}(m)$ et une opération à coût constant, donc declareAmis est en $\mathcal{O}(m)$

95

6

```
def listeDesAmisDe(reseau, i):
    amis = []
    for paire in reseau[1]:
        if paire[1] == i: ~~or paire[0] == i:~~
            amis.append(paire[0])
        ~~else~~
        elif paire[0] == i:
            amis.append(paire[1])
    return amis
```

1

listeDesAmisDe effectue m traitements à coût constant (car on traite chaque paire) et a donc une complexité en $O(m)$.

/1

7

Représentation filiale A

i	0	1	2	3	4	5	6	7	8	9
parent[i]	5	1	1	3	4	5	1	5	5	7

Représentation filiale B

/1

i	0	1	2	3	4	5	6	7	8	9
parent[i]	3	9	0	3	9	4	4	7	1	9

8

```
def creerPartitionEnSingletons(n):
    return list(range(n))
```

/1

9

On remonte au parent récursivement jusqu'à tomber sur un représentant, c'est-à-dire un élément i tel que parent[i] = i.

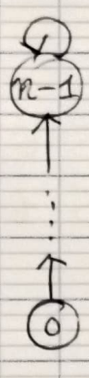
```
def representant(parent, i):
    if i == parent[i]: return i

    return representant(parent, parent[i])
```

/1

Dans le pire des cas, la partition est constituée d'un seul groupe dont i est l'élément le plus profond relativement à la racine (le représentant)

Par exemple, representant(parent, 0) où parent code la représentation fidèle suivante :

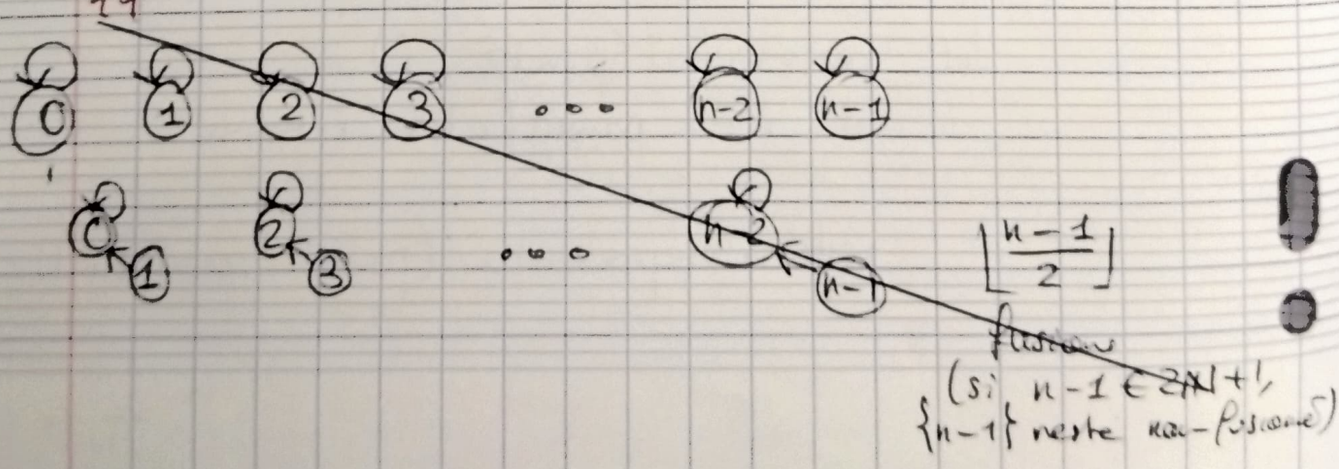


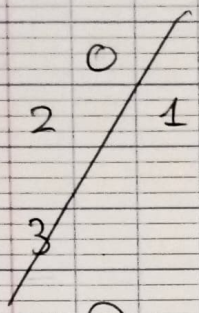
Dans ce cas, la complexité est en $\mathcal{O}(n)$: il faut $n-1$ appels récursifs pour remonter au représentant

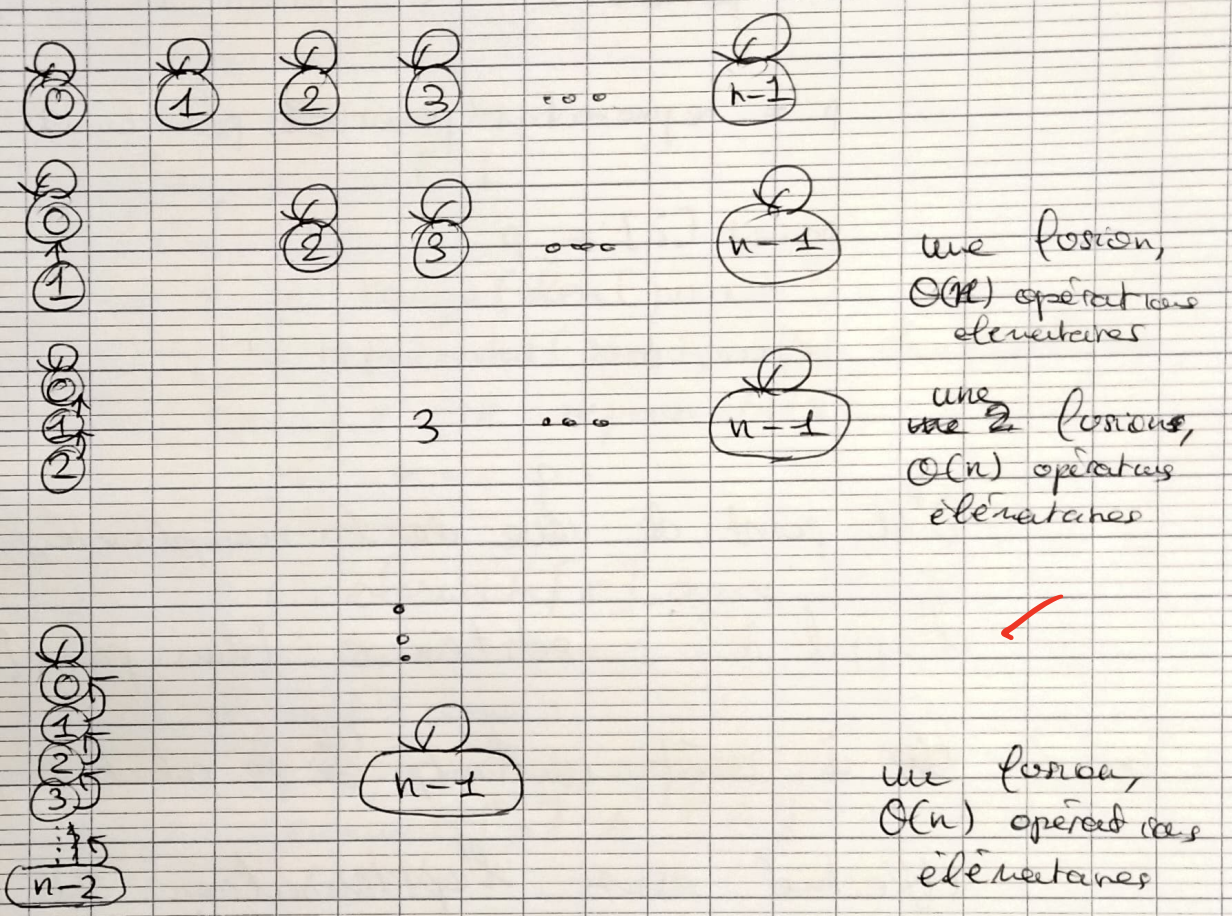
10

```
def fusion(parent, i, j):
    p, q = representant(parent, i), representant(parent, j)
    ~~p, q = representant(i), representant(j)~~
    parent[p] = q
```

11







une fusion, $O(n)$ opérations élémentaires

une ~~une~~ 2 fusions, $O(n)$ opérations élémentaires

une fusion, $O(n)$ opérations élémentaires

Au total :
$O(n \times n) = O(n^2)$
opérations élémentaires,
n fusions.

1

1

2

12

```
def representant(parent, i):
    if parent[i] == i : return i
    p = representant(parent, parent[i])
    parent[i] = p
    return p
```

3

1

D'un point de vue de la complexité, l'appel à __setitem__ (dans parent[i]=p) est à coût constant, car le calcul de p était effectué avant l'optimisation. Il ne modifie donc pas la complexité, qui reste linéaire.

1

/3

13 On se sert de la liste representants comme dictionnaire associant la position dans elements d'un groupe à son representant

```
def listeDesGroupes(parent):
    ~~parents = []~~
    representants = []
    ~~parents~~
    elements = []

    for i in range(len(parent)):
        p = representant(parent, i); done = False
        for r in representants:
            if r == p:
            ~~if 2r~~
                elements[r].append(i)
                done = True
        if not done:
            representants.append(p)
            elements[p] = [i]

    return elements
```

3

/

```
def coupeMinimumRandom.see(reseau)
    P = creerPartitionEtSyndicats(reseau[0])
    links = reseau[1][:]
    marked = 0
    unmerged = lambda: links[:len(P)-marked]
    reps = (lambda l:(representant(P, l[0]),
                      representant(P, l[1])))
    while len(P)>=3 and not unmerged():
        i = random.randint(0, len(unmerged()))
        p,q = reps(links[i])
        if p != q:
            fusion(P, p, q)
            links = links[:i]+links[:i]+[links[i]
            marked += 1
    if len(P) >= 3:
```

15 On compte tout les liens appartenant à différents groupes de P

```
def tailleCoupe(reseau, parent):
    taille = 0
    for link in reseau[1]:
        if re
    return len([link for link in reseau[1]
                if (representant(parent, link[0])
                    != representant(parent, link[1]))])
```

3

1

16

```
SELECT id2
FROM LIENS
  JOIN INDIVIDUS ON id =
WHERE id1 = x
```

9 8

17

```
SELECT nom, prenom
FROM INDIVIDUS
  JOIN LIENS ON id1 = id
WHERE id2 = x
```

1

18

```
SELECT l1.id1
FROM LIENS l1
  JOIN LIENS l2 ON l1.id2 = l2.id1
WHERE l2.id2 = x
```

1 2