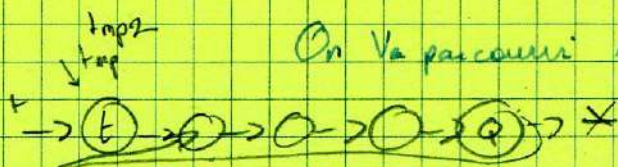


BLOQUET Romain
PL2 - 2014

Question 5 a' 8:

Question 5: Cet algorithme a pour but de faire que la queue de liste devienne la tête et la tête la queue de liste.

On va parcourir la liste jusqu'à l'avant dernier maillon.



Inverser ($l: \text{Lsc}$): l .

Donnée: l , la liste dont on veut permuter la tête et la queue.
modifiée

Variables locales: i , compteurs de maillons.
 tmp , qui permet le chaînage.

Debut:

Si $l = \emptyset$ alors retourner.
- $tmp \leftarrow tmp \leftarrow$ reserve maillon.
- $tmp2 \leftarrow tmp \leftarrow l$

tant que $l \rightarrow succ \rightarrow succ \neq \emptyset$ faire

$l \leftarrow l \rightarrow succ$

fin tant que

$l \rightarrow succ \leftarrow tmp \rightarrow succ$

$l \rightarrow succ \rightarrow succ \leftarrow tmp2$

fin.

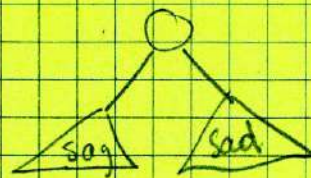
A revoir



Question 6: algorithme qui renvoi un entier (compteur du nombre d'éléments).

$nb_elt(a: AB): \text{entier}$.

Donnée: a , l'arbre dont on veut compter les éléments.



début:

Si $a = \emptyset$ retourner 0

retourner $1 + (nb_elt(a \rightarrow sag) + nb_elt(a \rightarrow sad))$

fin.

mais encore ?

Question 7: Même principe que l'algorithme précédent.

$nb(a: AB, b: \text{entier}, c: \text{entier}): \text{entier}$.

Donnée: a , l'arbre dont on veut compter le nombre d'éléments compris entre b etc.

b , et c deux entiers formant les bornes. b (borne inf) c (borne sup)

Résultat: un entier (le nombre d'éléments).

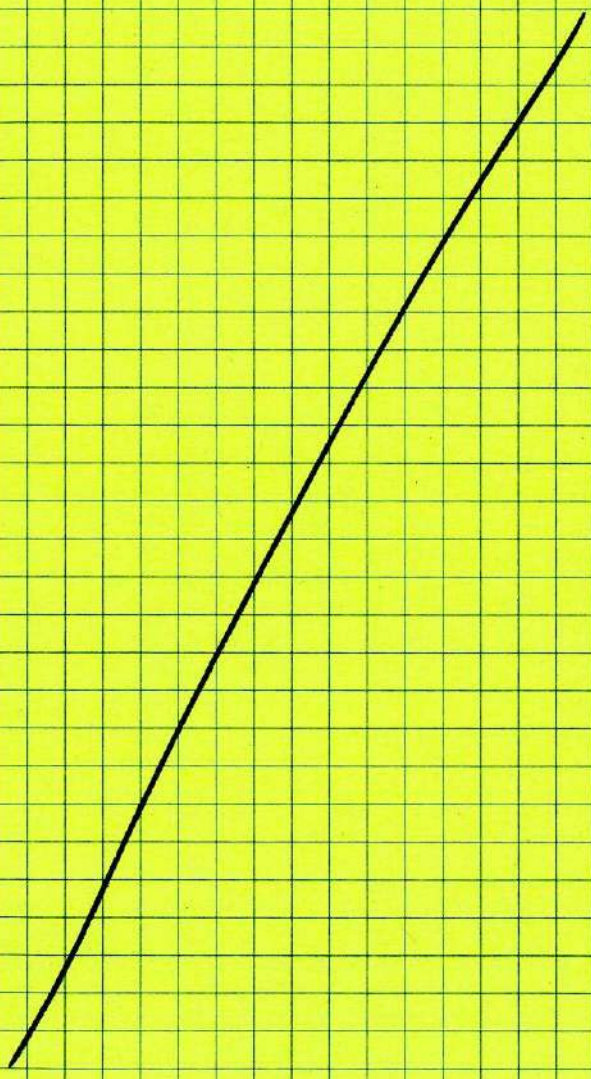
Début:

Si $a = \emptyset$ alors retourner 0.

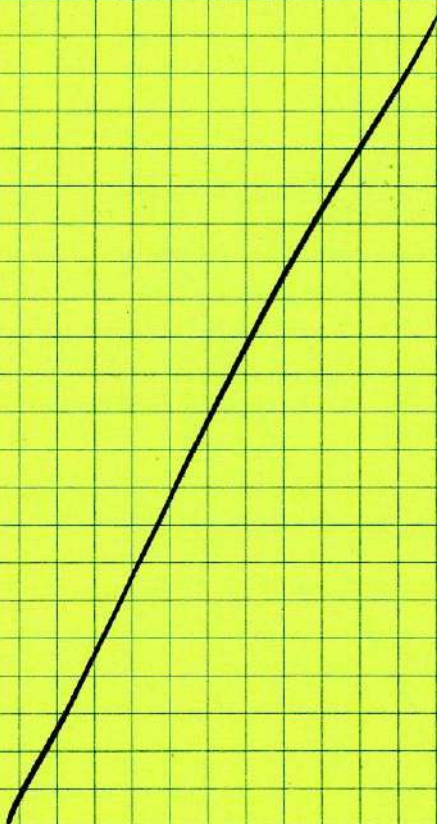
Si $a \geq b$ et $a \leq c$ alors

retourner $1 + (nb(a \rightarrow sag) + nb(a \rightarrow sad))$

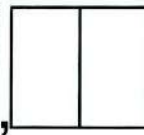
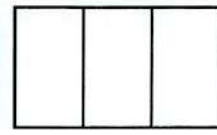
fin.



un p...
un p...
un p...



NOM Bloquet
Prénom Romain
Promo 2018
Date 19/12/2014.



BLOQUET Romain
PL2 - 2014

MATIÈRE Structure des données.

Question de 1 à 4:

Question 1:

nb-éléments (l : lsc): entier.

Donnée: l , la liste dont on veut compter le nombre d'éléments pairs.

Variable locale: i , le compteur d'élément pairs. ✓

Résultat: Un entier, nombre d'éléments pairs de la liste l .

Début

$i \leftarrow 0$

Si $l = \emptyset$ alors retourner 0) ~~supprimer (-)~~

tant que $l \neq \emptyset$ faire

si $(l \rightarrow \text{info}) \bmod 2 \neq 0$ alors

$i \leftarrow i + 1$

fin de si

$l \leftarrow l \rightarrow \text{succ}$

fin tant que.

retourner i

fin.

A

Le principe de cet algorithme est de parcourir la liste en entier, quand l'élément du nœud est pair, alors le compteur prend sa valeur plus un sinon il ne prend rien. Pour vérifier si le nombre est pair il suffit d'utiliser le modulo 2 car ce sont des entiers et si ce n'est pas pair ça ne donnera pas un entier. Enfin on retourne la valeur du compteur.

Question 2: Le principe de cet algorithme est de parcourir la liste en ayant une variable précédente qui prend la liste puis de déplacer la tête de liste vers le maillon suivant pour enfin libérer le maillon précédent, et on appelle récursivement cette fonction pour que la liste soit supprimée intégralement.

Supprimer_LSC (l: LSC)

Donnée modifiée: l, la liste chaînée linéaire à supprimer

Variable locale: précédent, le maillon qui prend l.
 — input car possibilité de post-traitement

Début:

Si $l = \emptyset$ alors retourner.

~~précédent ← remove(maillon).~~ (--) Il du par

précédent ← l

$l \leftarrow l \rightarrow \text{suivant}$

libérer (précédent)

Supprimer_LSC (l)

$l \leftarrow \emptyset$ (→)

fin.

Question 3: Pour cet algorithme nous allons utiliser longueur_liste défini par: longueur_liste (l: LSC)

Début:

Si $l = \emptyset$ alors retourner 0.

Tant que $l \neq \emptyset$ faire

$l \leftarrow l \rightarrow \text{succ}$

$i \leftarrow i + 1$

fin tant que
retourner i

fin

Affiche_Inverse (l: LSC ; lon: entier ; j: entier)

Données: l, la liste à afficher à l'envers

long, la longueur de la liste

Variables locales: i, le compteur de déplacement dans la liste.
j, compteur de déplacement.

Début: Si l = 0 alors retourner.

i = 1

j = 0

lon ← longueur_liste(l) - j

tant que i < lon

l ← l → suivant

fin tant que.

afficher l → info.

j = 1 + j

Affiche_Inverse (l, lon, j)

lourd + $O(n^2)$

B

entier (-)

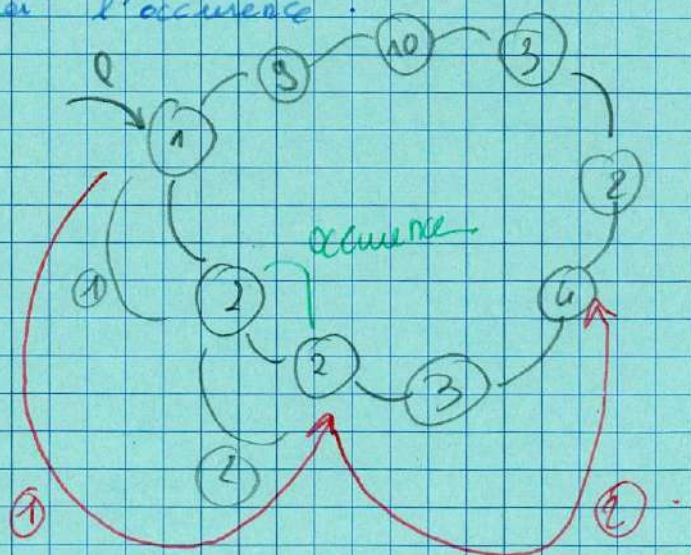
d'exit
suppr
de créer

fin.

Exercice 4.5

Le principe de cet algorithme est d'avoir deux Variables qui parcourent la liste de deux façons différentes, V1 de un maillon en un maillon, V2 de deux en deux. Elles finiront par se rencontrer. Quand elle se rencontrent, il faut refaire le chaînage et libérer l'occurrence.

je ne vois
pas en quoi
cela répond,
au pb posé



→ H.S.

Suppr_occ (lislsc): c: lsc.

Donnée modifiée: l_p la liste chaînée circulaire dont on veut conserver que la première occurrence.

Variable: V_1 , qui sert à parcourir la liste de 1 en 1

V_2 , qui sert à parcourir la liste de 2 en 1.

Résultat: $Temp$, pour pouvoir effectuer le chaînage.
 C_p , la liste chaînée l modifiée.

Début:

Si $l = \emptyset$ alors retourner $\emptyset$.
 $V_1 \leftarrow$ récupérer maillon
 $V_1 \leftarrow l$
 $V_2 \leftarrow$ récupérer maillon
 $V_2 \leftarrow l$

~~tant que~~

Si $V_1 \rightarrow info = V_2 \rightarrow info$ alors $P \rightarrow succ \leftarrow P \rightarrow succ \rightarrow succ$

tant que $V_1 \rightarrow info \neq V_2 \rightarrow info$

$Temp \leftarrow V_2$
 $V_1 \leftarrow V_1 \rightarrow succ$

$V_2 \leftarrow V_2 \rightarrow succ \rightarrow succ$

~~fin tant que~~

~~libérer (Temp)~~

~~suppr_occ (l)~~

Fin.