POUPA
Adrien
L'3 2018
Efrei Groupe C

1/3

Structure de données

20/11/2015

18,5

POUPA Adrien
L3PRIME - 2015

Questions de cours

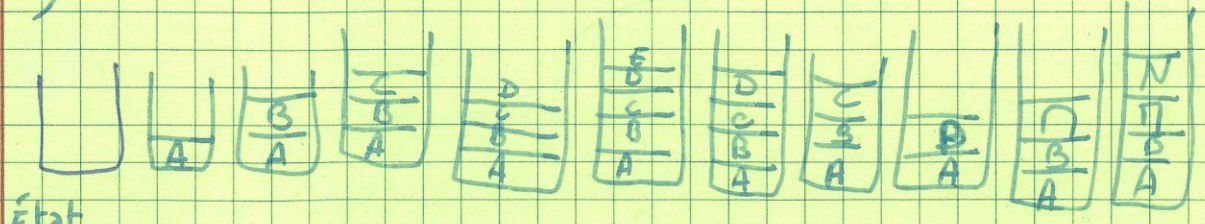
(5,75) ✓ 1) a) Une LSC contient les champs info (valeur) et succ (pointeur vers le maillon suivant). Une LDC contient en plus un champ prec, pointeur vers le maillon précédent.

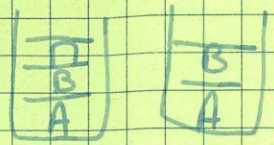
✓ b) On s'arrête de parcourir une liste simplement chaînée circulaire lorsqu'on revient au maillon à partir duquel on a commencé le parcours.

✓ c) Une file fonctionne selon le principe FIFO (First In, First Out = premier rentré, premier sorti). Une pile fonctionne de manière opposée, en LIFO (Last In, First Out = dernier rentré, premier sorti).

✓ d) Pile ABCDE ***DN**:

Chaque pile représente une étape

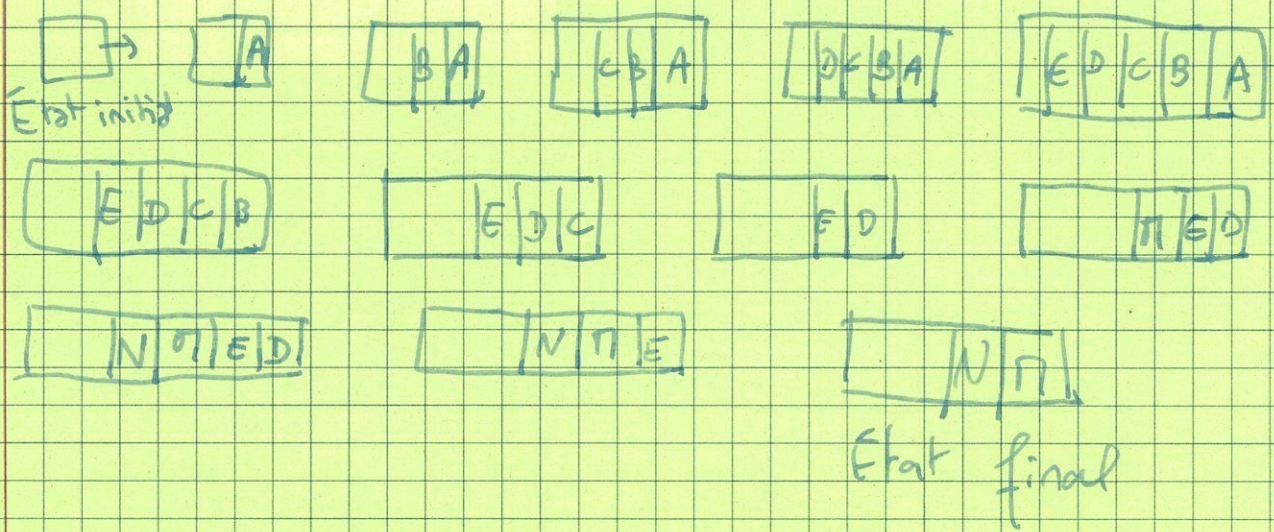




Etat final.

File ABCDE***ΠN**:

Chaque file représente une étape



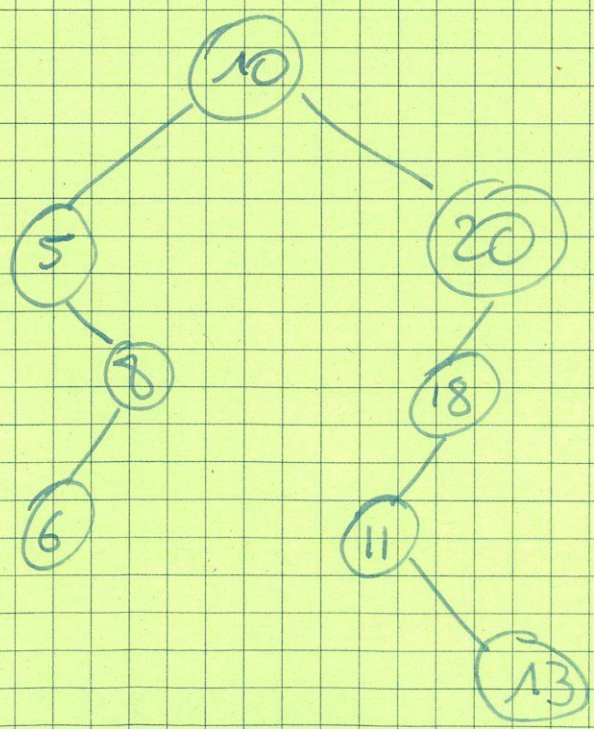
E □ → Sortie

~ e) Parcours préordre: 5, 9, 3, 8, 3, 10, 7, 6, 4

ordre: 3, 9, 3, 8, 10, 7, 5, 6, 4

postordre: 3, 10, 3, 7, 8, 9, 4, 6, 5

✓ f) Construction de l'ABR:



Listes simplement chaînées

(2) 2) Recherche Maximum (l : LSC) : entier positif ≠ énoncé

Donnée : l LSC
Variable locale : max entier positif

Début
  max ← 0
  Tant que l ≠ NULL
    Si l → info > max
      max ← l → info
    Fin Si
    l ← l → succ
  Fin tq
  Retourner max
Fin.

(2) 3) ProduitVclars (l : LSC) : entier

Donnée : l LSC
Variable locale : resultat entier

Début
  resultat ← 1
  Tant que l ≠ NULL
    resultat ← resultat * l → info
    l ← l → succ
  Fin tq
  Retourner resultat
Fin

4) InsertionDécroissante (l: LSC, e: entier) : LSC

Donnée: l LSC triée décroissante

Donnée: e entier à insérer

Variables locales: lo: LSC à retourner

pnv: pointeur sur maillon

p: pointeur sur maillon

ps: pointeur sur maillon

inséré: booléen

Résultat:

Début

lo ← l // Ne perdons pas la tête !

Si l = NULL // Liste vide

l ← réserver maillon

l → info ← e

l → succ ← null

lo ← l // Pour le return lo final

Sinon Si e > l → info // Insertion en tête

lo ← réserver maillon

lo → info ← e

lo → succ ← l

~~l ← lo~~

- Sinon // Insertion au milieu ou en fin

p ← l

ps ← l → succ

inséré ← faux

Tant que ps ≠ null et inséré = faux

Si ps → info ≤ e

pnv ← réserver maillon

pnv → info ← e

pnv → succ ← ps

p → succ ← pnv

inséré ← vrai.

Fin Si

```
        p ← p → succ
        ps ← ps → succ
    Fin tq
    Si inséré = faux // Fin de liste, on
        pnv ← réserver maillon // n'a toujours rien inséré
        pnv → succ ← nul
        p → succ ← pnv
    Fin si
  Fin sinon
  Retourner lo
Fin.
```

(0,5) 5) Suppression Croissante (l: LSC, e: entier) : LSC

Donnée: l LSC triée croissante

Donnée: e entier à partir duquel supprimer.

Variables locales : lo : LSC à retourner

p : pointeur sur maillon

ps : pointeur sur maillon

Résultat : —

```
Début
  lo ← l
  Si l=NULL
    Retourner lo
  Sinon
    p ← l
    ps ← l → succ
    Tant que ps ≠ NULL
      Si e > p → info
        Tant que ps ≠ NULL
          ps ← p → succ
          libérer ps
      Fin tq
```

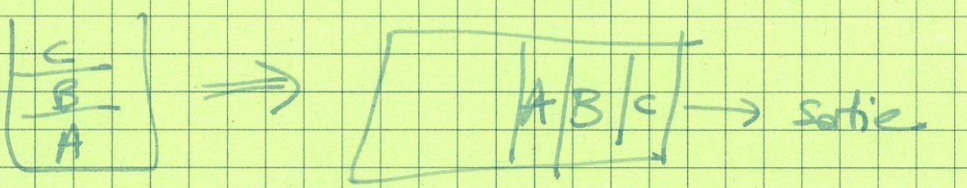
cas toute la liste l (donc l = elem') à supprimer ?

/ non

```
            Retourner lo
            p ← p → suiv
            ps ← ps → suiv
        Fin tq
        Retourner lo
    Fin Sinon
Fin
```

Piles et Files

Pour cet algorithme, il n'est pas précisé dans quel ordre on filera les éléments de la pile - On fait donc le plus simplement possible :



(2.75) Transférer Pile Dans File (p : pile, f : file)
Donnée modifiée : p pile de départ
Donnée modifiée : f file d'arrivée

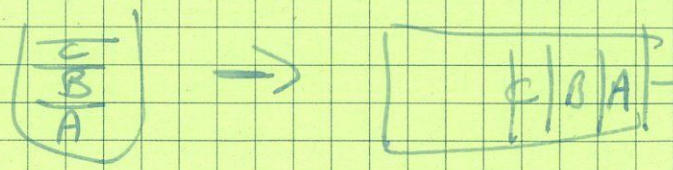
Début

```
    // Pile vide : rien à faire
    Si Est_Pile_Vide(p) = VRAI
        Retourner
    Sinon
        Tant que Est_Pile_Vide(p) = FAUX
            Enfiler(f, Dépiler(p))
        Fin tq
```

Fin sinon

Fin

Note : si on avait voulu faire



il aurait fallu créer la pile intermédiaire

et procéder comme on l'a fait.

Arbres

7)

(2,5) 8) Il n'est pas précisé qu'il est interdit de faire appel à des sous-programmes définis. Pas de pb !

```
Maximum(a : ABR) : entier
Donnée : a ABR
Résultat : —
Debut
    Si a=nul   //On ne devrait pas rentrer ici
        Retourner 0   //ABR positifs
    Si a→sad = nul
        Retourner a→info
retourner maximum(a→sad)
Fin

Minimum (a : ABR) : entier
Donnée : a ABR
Résultat : —
Debut
    Si a = nul //On ne devrait pas rentrer ici
        Retourner 0    //ABR positifs
    Si a→sag = nul
        Retourner a→info
retourner Minimum(a→sag)
Fin

Difference(a : ABR) : entier
Donnée : a ABR
Résultat : —
Debut
    Retourner Maximum(a) - Minimum(a)
Fin
```

POUR4
Adrien
(0,5)
3/3

9) SupprimerPlusPetit (a : ABR)
Donnée modifiée : a ABR
Variables locales : p et ps pointeurs sur maillon

Début
  Si a = NULL
    Retourner
  // Arbre vide

cas a vi a pas de sag ?

  Sinon
    p ← a
    ps ← a → sag
    Tant que ps ≠ NULL
      p ← p → sag // ABR : minimum à gauche
      ps ← ps → sag
    Fin tq
    libérer p^
    p ← nulle p→sag ← ?
    Et si ps→sad ≠ nul ?
  Fin sinon
Fin