

Algorithmique et Structures de Données

Akkouche Samir :

E-mail : samir.akkouche@liris.cnrs.fr

Pronost Nicolas:

E-mail : nicolas.pronost@univ-lyon1.fr

Plan du cours

1. Rappels
2. Tableaux dynamiques
3. Listes chaînées
4. Piles et Files
 1. Piles
 2. Files
5. Arbres
 1. Définitions et exemple
 2. Arbres binaires
 3. Tas binaire
 4. Arbres binaires de recherche

TDA pile

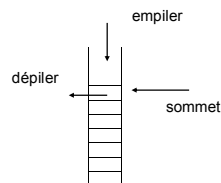
- **Caractérisation :**

Seul est accessible le sommet de la pile

- **Opérations :**

empiler
depiler
consultersommet
pilevide

➡ Dernier arrivé, premier utilisé



Module Pile

```
module Pile
importer module Element
Exporter type Pile
procedure initialiser ( s : Pile)
    Precondition s non initialisée
    Postcondition s initialisée à vide
    Paramètre en mode donnée-résultat : s
procedure testament( s : Pile)
    Precondition s est bien initialisée
    Postcondition l'espace réservé pour la pile est libéré proprement
    Paramètre en mode donnée-résultat : s
fonction consultersommet ( s : Pile) : Element
    Precondition s est bien initialisée et non vide
    Résultat : renvoie l'information contenue au sommet de la Pile sans en modifier le contenu.
    Paramètre en mode donnée : s
-----Asuivre-----
finModule
```

Module Pile (Suite)

procedure empiler(s : Pile, e : Element)

Precondition s est bien initialisée
Postcondition e est placé au sommet de la pile
Paramètre en mode donnée-résultat : s
Paramètre en mode donnée : e

procedure depiler(s : Pile)

Precondition s est bien initialisée et non vide
Postcondition supprime le sommet de la pile
Paramètre en mode donnée-résultat : s

fonction estVide(s : Pile) : boolean

Precondition s est bien initialisée
Résultat renvoie vrai si la Pile est vide faux sinon
Paramètre en mode donnée s

...

Implantation

Deux possibilités :

1. Utilisation de tableau dynamique
2. Utilisation de listes chaînées

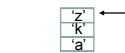
finModule

Exemple

Début

```
p : Pile
s : Element
v : booléen
initialiser(p)
empiler(p, 'a')
empiler(p, 'k')
empiler(p, 'z')
dépiler(p)
empiler(p, 'm')
s <- consultersommet(p)
v <- estVide(p)
testament(p)
```

Fin



Etat de la pile après
trois emplacements



Etat de la pile après
depiler



Etat de la pile après
empiler

Implantation de la pile

- Liste chaînée :

empiler = insertion en tête
depiler = retrait en tête

➡ Coût constant

- Tableau Dynamique :

empiler = insertion en dernière_position
depiler = retrait en dernière_position

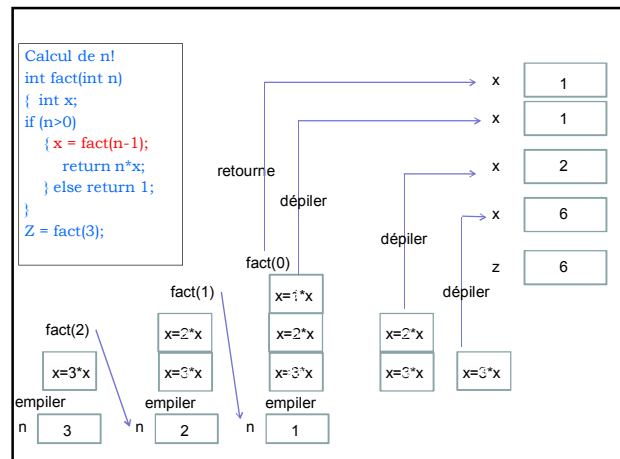
➡ Coût constant

Pile : Mise en œuvre

Structure du fichier pile.h

```
#ifndef __PILE
#define __PILE
#include "element.H"
#include "tableauDynamique.H"
struct sPile { TableauDynamique td; };
typedef struct sPile Pile;
void initialiser( Pile * );
Element consultersommet(Pile *);
void testament(Pile *);
void empiler(Pile *, Element);
void depiler(Pile *);
int estvide(Pile);
#endif
```

Exemple 1 Gestion de la récursivité



Exemple 2 Evaluation d'une expression arithmétique

Evaluation d'instructions du type :

$$\text{bidul} = ((\text{machin} - 1) + x * \text{toto} / 8) / 25 - 12$$

Identificateur = expression arithmétique

Solution : transformer la représentation infixée en postfixée

infixée : $a + 2$
 préfixée : $+ a 2$
 postfixée : $a 2 +$

bidul machin $1 - x \text{ toto} * 8 / + 25 / 12 - =$

Passage Infixée - Postfixée

● Priorité des opérateurs

* et / sont plus prioritaires que + et -, = est le moins prioritaire
bidul = ((machin -1) + x*toto/8)/25 - 12

● Méthode : Soit P une pile et Post un tableau

Si l'élément courant est :

- ⇒ un identificateur : recopier dans Post
- ⇒ un nombre : recopier dans Post
- ⇒ un opérateur + - * / : alors
 - dépiler les opérateurs de priorité supérieur ou égale
 - les recopier dans Post
 - empiler l'opérateur courant
- ⇒ une parenthèse droite) :
 - dépiler jusqu'à (
 - recopier dans Post sauf (
- ⇒ = ou (: empiler
- ⇒ Fin : tout dépiler et recopier dans Post

Evaluation d'une expression arithmétique

Infixe 2 '+' 4 '*' 3 '-' 5 Epost 2 4 3 '*' '+' 5 '-'

Méthode :

Soit P une pile et EPost un tableau d'Elements

Extraire un élément EPost[i]

Si EPost[i] est un nombre alors

empiler(P, EPost[i])

Finsi

Si EPost[i] est un opérateur alors

x <- consultersommet(P), dépiler(P),

y <- consultersommet(P), dépiler(P)

z <- Evaluer(y,x, EPost[i]),

empiler(P, z)

Finsi

//Le résultat est au sommet de la pile.

retourner consultersommet(P)

1. Rappels
2. Tableaux dynamiques
3. Listes chaînées
4. Piles et Files
 1. Piles
 2. Files
5. Arbres
 1. Définitions et exemple
 2. Arbres binaires
 3. Tas binaire
 4. Arbres binaires de recherche

TDA File

● Caractérisation :

Premier arrivé, premier servi

=> Insertion en fin et récupération en tête

● Opérations :

enfiler

defiler

consulterPremier

filevide

Module File

```

module File
importer moduleElement
exporter File
procedure initialiser(f : File)
    Precondition f non initialisée
    Postcondition f initialisée à vide
    Paramètre en mode résultat f
procedure testament(f : File)
    Precondition f est bien initialisée
    Postcondition l'espace réservé pour la file est libéré proprement
    Paramètre en mode donnée-résultat f
fonction estVide(f : File) : booléen
    Precondition f est bien initialisée
    Résultat renvoie vrai si la file est vide
    Paramètre en mode donnée f
    -----A suivre-----
finModule
    
```

Module File

```

module File(suite)
procedure enfiler(f : File, e Element)
    Precondition f est bien initialisée
    Postcondition e est placé en dernière position dans la file
    Paramètre en mode donnée-résultat f
procedure defiler(f : File)
    Precondition f est bien initialisée et non vide
    Postcondition l'élément en « tête » est supprimé de la file
    Paramètre en mode donnée-résultat f
fonction premier(f : File) : Element
    Precondition f est bien initialisée et non vide
    Résultat renvoie le premier élément de la file
    Paramètre en mode donnée-résultat f
implantation
    Deux possibilités :
    1.Utilisation de tableau dynamique
    2.Utilisation de listes chaînées
finModule
    
```

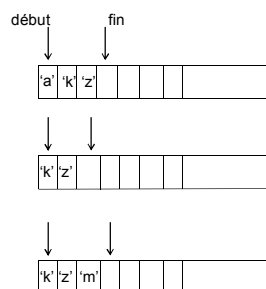
Exemple

Début

```

f : File
t : Element
v : booléen
initialiser(f)
enfiler(f, 'a')
enfiler(f, 'k')
enfiler(f, 'z')
defiler(f)
enfiler(f, 'm')
t <- premier(f)
v <- estVide(f)
testament(f)
    
```

Fin



Coûts

Liste chaînée :

enfiler = insertion en fin
defiler = retrait en tete

➔ Coût ?

Tableau Dynamique :

enfiler = insertion en derniere_position
defiler = retrait en premiere_position

➔ Coût ?