I21: Introduction à l'algorithmique

Cours 7: Piles et Files

Nicolas Méloni

Licence 1: 2ème semestre
(2017/2018)

Structure de données dynamiques

- Les tableaux sont des structures de données statiques représentant fidèlement la configuration des données en mémoire.
- Les structures de données dynamiques sont des abstractions permettant de gérer un ensemble d'éléments dans un ordre précis.

Structure de données dynamiques

- Dans ce cours nous nous intéresserons à deux structures particulères ne permettant que l'ajout ou la suppression d'éléments :
  - les piles, pour lesquelles on ne peut supprimer que le dernier élément ajouter (principe LIFO (Last-In-First-Out))
  - les files, pour lesquelles on ne peut supprimer que l'éléments le plus ancien (principe FIFO (First-In-First-Out)).

Les piles

Soit une pile `P`. On dipose de 4 opérations :

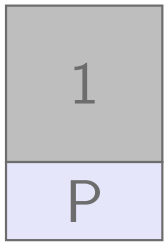
- `Lire(P)` : renvoit la valeur de l'élément au sommet de la pile ;
- `Vide(P)` : renvoit `VRAI` si la pile est vide et `FAUX` sinon ;
- `Empiler(p,x)` : ajoute l'élément `x` au sommet de la pile ;
- `Dépiler(P)` : supprime l'élément au sommet de la pile et renvoit sa valeur ;

En théorie, chacune de ses opérations est de complexité $O(1)$.

Les piles

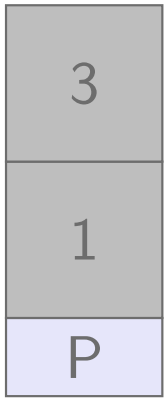
Les piles

- `Empiler(P,1)`



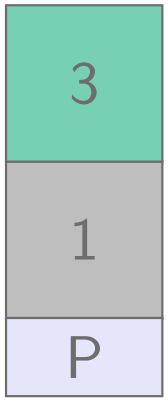
Les piles

- `Empiler(P,1)`
- `Empiler(P,3)`



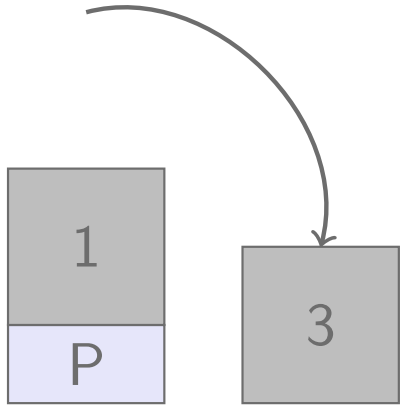
Les piles

- `Empiler(P,1)`
- `Empiler(P,3)`
- `Lire(P)`



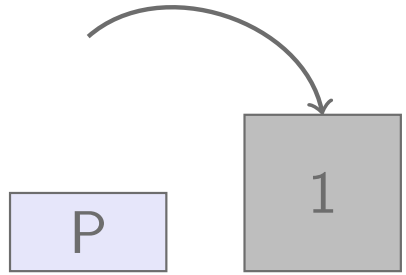
Les piles

- `Empiler(P,1)`
- `Empiler(P,3)`
- `Lire(P)`
- `Depiler(P)`



Les piles

- `Empiler(P,1)`
- `Empiler(P,3)`
- `Lire(P)`
- `Depiler(P)`
- `Depiler(P)`



Implémentation des piles

On peut implémenter une structure de pile d'au plus n élements à l'aide d'un tableau de taille n :

- le tableau `P[1:n]` permet de stocker les éléments de la pile ;
- une variable (globale) `sommet` indèxe l'élément au sommet de la pile ;
- la pile est constituée des éléments `P[1:sommet]`

Implémentation des piles

```
1  ALGORITHME Pile_Vide(P):
2  DONNEES
3     P: tableau de taille n
4  DEBUT
5     SI sommet = 0 ALORS
6        RENVOYER VRAI
7     SINON
8        RENVOYER FAUX
9     FSI
10 FIN
```

```
1  ALGORITHME Lire(P):
2  DONNEES
3     P: tableau de taille n
4  DEBUT
5     RENVOYER P[sommet]
6  FIN
```

Implémentation des piles

```
1  ALGORITHME Empiler(P,x):
2  DONNEES
3    P: tableau de taille n
4  DEBUT
5    SI sommet = n ALORS
6      AFFICHER "debordement"
7      RENVOYER ERREUR
8    SINON
9      sommet ← sommet+1
10     P[sommet] ← x
11   FSI
12 FIN
```

```
1  ALGORITHME Depiler(P):
2  DONNEES
3    P: tableau de taille n
4  DEBUT
5    SI Pile_Vide(P) = FAUX ALORS
6      sommet ← sommet−1
7      RENVOYER P[sommet+1]
8  FIN
```

Parenthésage d'une expression

Problème : Parenthésage

Entrée : une chaîne de caractères C de longueur n
Sortie : VRAI si les parenthèses et les crochets sont équilibrés, FAUX sinon

Exemple : `"(a+5*[2-7]+(3*3+x))"` est bien parenthésée mais pas `"(3+[6*x-2)]"`.

Parenthésage d'une expression

```
1  ALGORITHME Parenthesage(C):
2  DONNEES
3    C: chaine de longueur n
4  VARIABLES:
5    i: entier
6    P: pile
7  DEBUT
8    i ← 1
9    TQ i ⩽ n FAIRE
10     SI C[i]='(' OU C[i]='[' ALORS
11       Empiler(P,C[i])
12     SINON SI C[i]=')' ALORS
13       SI Depiler(P) ≠ '(' ALORS
14         RENVOYER FAUX
15       FSI
16     SINON SI C[i]=']' ALORS
17       SI Depiler(P) ≠ '[' ALORS
18         RENVOYER FAUX
19       FSI
20     FSI
21     i ← i+1
22   FTQ
23   RENVOYER Pile_Vide(P)
24 FIN
```

- Meilleur cas : $\Theta(1)$
- Pire cas : $\Theta(n)$
- Complexité : $O(n)$

Les files

Soit une file `F`. On dipose de 4 opérations :

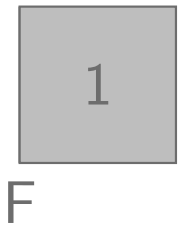
- `Lire(F)` : renvoit la valeur de l'élément en tête de la file ;
- `Vide(F)` : renvoit `VRAI` si la file est vide et `FAUX` sinon ;
- `Enfiler(F,x)` : ajoute l'élément `x` à la queue de la file ;
- `Defiler(F)` : supprime l'élément en tête de la pile et renvoit sa valeur ;

En théorie, chacune de ses opérations est de complexité $O(1)$.

Les piles

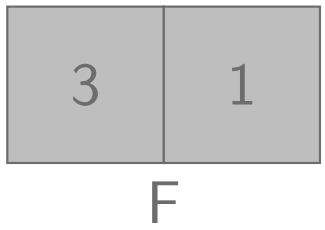
Les piles

- `Enfiler(F,1)`



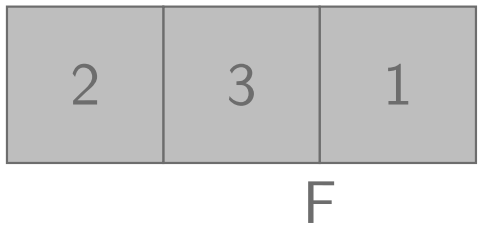
Les piles

- `Enfiler(F,1)`
- `Enfiler(F,3)`



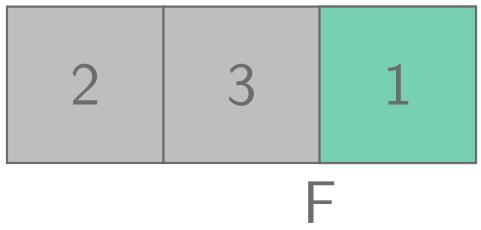
Les piles

- `Enfiler(F,1)`
- `Enfiler(F,3)`
- `Enfiler(F,2)`



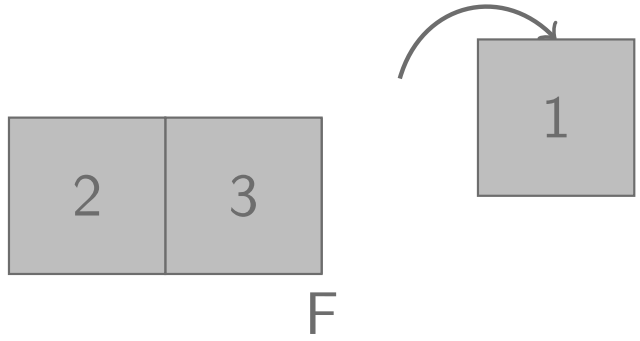
Les piles

- `Enfiler(F,1)`
- `Enfiler(F,3)`
- `Enfiler(F,2)`
- `Lire(F)`



Les piles

- `Enfiler(F,1)`
- `Enfiler(F,3)`
- `Enfiler(F,2)`
- `Lire(F)`
- `Defiler(F)`



Implémentation des piles

On peut implémenter une structure de file d'au plus n élements à l'aide d'un tableau de taille n :

- le tableau `F[1:n]` permet de stocker les éléments de la file ;
- une variable (globale) `tete` indèxe l'élément de tête de la file ;
- une variable (globale) `queue` indèxe l'élément de queue de la file ;
- la file est constituée des éléments `P[queue:tete]`

Implémentation des files

```
1  ALGORITHME File_Vide(F):
2  DONNEES
3     F: tableau de taille n
4  DEBUT
5     SI tete < queue ALORS
6        RENVOYER VRAI
7     SINON
8        RENVOYER FAUX
9     FSI
10 FIN
```

```
1  ALGORITHME Lire(F):
2  DONNEES
3     P: tableau de taille n
4  DEBUT
5     RENVOYER P[tete]
6  FIN
```

Implémentation des files

```
1  ALGORITHME Enfiler(F,x):
2  DONNEES
3    P: tableau de taille n
4  DEBUT
5    SI queue = 0 ALORS
6      AFFICHER "debordement"
7      RENVOYER ERREUR
8    SINON
9      queue ← queue-1
10     F[queue] ← x
11   FSI
12 FIN
```

```
1  ALGORITHME Defiler(F):
2  DONNEES
3    P: tableau de taille n
4  DEBUT
5    SI File_Vide(F) = FAUX ALORS
6      tete ← tete-1
7      RENVOYER F[tete+1]
8  FIN
```

Affichage des nombres binaires

Problème : Affichage des nombres binaires

Entrée : un entier n
Sortie : afficher l' écriture binaire de tous les nombres entre 1 et n.

Exemple : $n = 9$,
Affichage : `1, 10, 11, 100, 101, 110, 111,1000,1001`.

Décomposition en base 2

```
1  ALGORITHME Base2(k):
2  DONNEES
3     k: entier
4  VARIABLES:
5     i: entier
6     C: chaine de caracteres
7  DEBUT
8     i ← 1
9     TQ k > 0 FAIRE
10       SI n est impair ALORS
11          C ← Concat("1",C)
12       SINON
13          C ← Concat("0",C)
14       FSI
15       k ← ⌊k/2⌋
16    FTQ
17    RENVOYER C
18 FIN
```

- Complexité : $\Theta(\log(k))$

2Affichage des nombres binaire

```
1  ALGORITHME NombreBinaire(n):
2  DONNEES
3     n: entier
4  VARIABLES:
5     i: entier
6     C: chaine de caracteres
7  DEBUT
8     i ← 1
9     TQ i ⩽ n FAIRE
10       C ← Base2(i)
11       AFFICHER(C)
12       i ← i+1
13    FTQ
14 FIN
```

- Complexité : $\Theta(n \log(n))$

Affichage avec une file

On peut faire mieux à l'aide d'une file

- Idée : un nombre à $k+1$ chiffres est un nombre à k chiffres auquel on ajoute soit 0 soit 1 à la fin ;
- si une file contient tous les nombres à k chiffres, il suffit de les faire défiler et de leur rajouter 0 et 1 à chacun.

Affichage avec une file

```
1  ALGORITHME NombreBinaire(n):
2  DONNEES
3     n: entier
4  VARIABLES:
5     i: entier
6     C: chaine de caracteres
7     F: file
8  DEBUT
9     i ← 1
10    Enfiler(F,"1")
11    TQ i ⩽ n FAIRE
12       C ← Defiler(F)
13       AFFICHER(C)
14       Enfiler(Concat(C,"0"))
15       Enfiler(Concat(C,"1"))
16       i ← i+1
17    FTQ
18 FIN
```

- Complexité : $\Theta(n)$