IF111 - Algorithmes et structures de données-TD2 : Piles, Files et Listes

Jonathan Narboni, Rohan Fossé

jonathan.narboni@labri.fr, rfosse@labri.fr

Exercice 1

On dispose d'une pile contenant des nombres entiers. Écrire un algorithme qui permet d'afficher le contenu de la pile. On veut voir s'afficher dans l'ordre le sommet, le second élément, et ainsi de suite. On prendra soin que la pile, à l'issue de l'affichage, contienne les mêmes éléments *dans le même ordre*. L'algorithme utilise une pile auxiliaire. Comment modifier cet algorithme pour afficher les éléments dans l'ordre inverse ?

AffichePile(P)

1. {
2. **entier** x;
3. $P_{aux} \leftarrow CreerPileVide()$;
4. **tant que** ! $PileVide(P)$
5. $x \leftarrow Depiler(P)$
6. $affiche r(x)$;
7. $Empiler(P_{aux}, x)$;
8. **fin tant que**;
9. **tant que** $!PileVide(P_{aux})$
10. $Empiler(P, Depiler(P_{aux}))$;
11. **fin tant que**
12. }

Pour afficher en ordre inverse il suffit de afficher à la ligne 10 quand on depiler de P_{aux} et avant qu'on empile en P.

Exercice 2

On veut écrire des algorithmes pour manipuler des files et des piles. Les algorithmes peuvent utiliser des piles et des files auxiliaires.

- On se donne une pile $P1$ contenant des entiers positifs. Ecrire un algorithme pour déplacer les entiers de $P1$ dans une pile $P2$ de façon à avoir dans $P2$ tous les nombres pairs en dessous des nombres impairs. L'algorithme prend en entrée $P1$ et $P2$ et utilise $O(1)$ mémoire auxiliaire.

```
   Deplace(P1,P2)
   {
tant que ! EstVide(P1) faire
   x ← Depiler(P1)
   si x est pair alors
tant que Sommet(P2) est impaire faire
Empiler(P1,Depiler(P2))
fin tant que
fin si
Emplier(P2, x)
fin tant que
        fin tant que
   }
```

- Ecrire un algorithme qui renverse une file. Faites de même pour une pile.

```
   RenverseFile(F)
1. {
2.    entier x;
3.    Paux ← CreerPileVide();
4.    tant que !FileVide(F)
5.    x ← Defiler(F)
6.    Empiler(Paux, x);
7.    fin tant que;
8.    tant que !PileVide(Paux)
9.    Enfiler(F, Depiler(Paux));
10.   fin tant que
11.}

   RenversePile(P)
1. {
2.    entier x;
3.    Faux ← CreerFileVide();
4.    tant que !PileVide(P)
5.    x ← Depiler(P)
6.    Enfiler(Faux, x);
7.    fin tant que;
8.    tant que !FileVide(Faux)
9.    Empiler(P, Defiler(Faux));
10.   fin }tant que 11.
```

- En utilisant seulement des files, écrire un algorithme qui prend en entrée une file F contenant les éléments triés en ordre croissant (la tête est le plus petit élément de la file) et un élément e à enfiler. L'algorithme retournera la file F contenant l'élément e tout en préservant l'ordre croissant des éléments.

```
TriMain(F,e)
{entier x;
 bool insere ← faux;
 F_a ← CreerFileVide();
 tant que (EstVide(F) = faux) et (insere = faux) faire
    x ← Defiler(F);
    si x < e alors Enfiler(F_a, x);
    sinon
     Enfiler(F_a, e);
     Enfiler(F_a, x);
     insere ← vrai;
    fin si
fin tant que
si (insere = faux) alors Enfiler(F_a, e);
sinon
    tant que (EstVide(F) = faux)
    Enfiler(F_a, Defiler(F));
    fin tant que
fin si
tant que (EstVide(F_a) = faux)
Enfiler(F, Defiler(F_a));
fin tant que
}
```

Exercice 3

On veut implémenter une file en utilisant deux piles. Écrire les opérations suivantes : `EstVide(F)`, retourne vrai si la file est vide, faux sinon; `Enfiler(F,e)` qui insère e en queue de la file F, `Defiler(F)` retourne l' élément de tête et le supprime de la file, `Tête(F)` qui retourne l' élément de tête sans le supprimer de la file. Pour toute pile P vous avez à disposition les opérations de base suivantes pour la manipuler : `EstVide(P)`, `Empiler(P,e)`, `Depiler(P)` et `Sommet(P)`.

Une solution : Nous utiliserons deux piles P_1 et P_2 pour implémenter le type abstrait file. L'objet file aura deux attributs de type pile appelés P_1 et P_2. L'idée est que la tête de la file F correspond au sommet de la pile P_1 et le dernier élément de la file est le sommet de la pile P_2. La file est vide si les deux piles P_1 et P_2 sont vides.
Enfiler(F, e): Enfile un élément e dans la file F. Ça correspond à empiler l'élément dans la pile P_2.
Defiler(F) retourne la tête de la file, c'est-à-dire le premier élément inséré dans F. Cette opération est implémentée de la manière suivante : si la file P_1 n'est pas vide on retourne l'élément dépilé de P_1. Si P_1 est vide mais P_2 ne l'est pas, on reverse tous les éléments de P_2 dans P_1 et on dépile le sommet de P_1.
`Tête(F)` est similaire à **Defiler**(F), mais on utilisera l'opération `Sommet(`P_1`)` à la place de `Depiler(`P_1`)`.

EstVide(F)
{ **retourner** `EstVide(`P_1`)` ET `EstVide(`P_2`)`}

Enfiler(F, e)
{`Empiler(`P_2, e`)` }

Defiler(F)
{ **si** `EstVide(`P_1`)`=Faux **alors retourner** `Depiler(`P_1`)`;
sinon
si `EstVide(`P_2`)`=Faux **alors**
tant que `EstVide(`P_2`)` $= Faux$ **faire**
`Empiler(`P_1,`Depiler(`P_2`))`;
fin tant que
retourner `Depiler(`P_1`)`;
fin si
retourner "ERREUR file vide"
fin si
}

Exercice 4

On veut manipuler des listes. Écrivez les fonctions suivantes de manière récursive en utilisant les primitives `tête(L)`, `queue(L)` et `estVide(L)`. (on suppose que toutes les listes contiennent des entiers)

1. paire(L) qui retourne $vrai$ si la liste L ne contient que des entiers pairs, $faux$ sinon.

   paire(L)
   { **si** `EstVide(L)` **alors retourner** $vrai$; **fin si**
   si (`tête(L)`% 2=0) **alors retourner paire(**`queue(L)`**)**
   sinon `retourner` $faux$
   fin si
   }

2. avant_dernier(L) qui renvoie l'avant-dernier élément de la liste L s'il existe, $faux$ sinon.

```
avant_dernier(L)
{ si EstVide(L) alors retourner faux; fin si
 si EstVide(queue(L)) alors retourner faux; fin si
 si EstVide(queue(queue(L))) alors retourner(tête(L))
 sinon retourner avant_dernier(queue(L))
 fin si
}
```