Leçon 5 : Implémentations et applications des piles, files et files de priorité

prérequis : tableau, liste, liste chaînée, pointeur, type abstrait, arbre binaire

Niveau : MPII / Terminale

Références : Ellipse MPI/MPII

I - Piles

I.1 - Définition

Une pile est une structure de données représentant une collection de données du même type, fondée sur le principe « dernier arrivé, premier sorti » (LIFO).

Elle dispose de 3 fonctions :

- PileVide : renvoie true ssi la pile est vide
- Dépiler : dépile le sommet de la pile et le renvoie
- Empiler : ajoute un élément au sommet

Analogie 1 : Une pile d'assiettes

I.2 - Implémentations

I.2.1 - Avec une liste

Remarque 2 : Le type list OCaml réalise une pile immuable.

Programme 3 : pile en OCaml :

```
type 'a pile = 'a list ;;
let pileVide (p : 'a pile) : bool = p = [] ;;
let depile (p : 'a pile) : 'a = function
  | [] -> failwith "Liste vide"
  | x :: q -> x ;;
let empile (p : 'a pile) (x : 'a) : 'a pile = x :: p ;;
```

Complexité : O(1) pour les trois fonctions

Exercice 4 : Créer un type de pile mutable en OCaml et les fonctions pileVide, empile, depile associées.

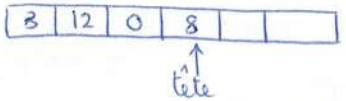
Solution : Utiliser des references

Remarque 5 : Le module Stack de la bibliothèque OCaml fournit des piles mutables.

I.2.2 - Implémentation avec un tableau

Idée 6 : Si la capacité de la pile est bornée, on peut la réaliser directement avec un tableau, en mémorisant un pointeur vers la tête de la pile.

exemple 7 : Illustration avec une pile d'entiers de capacité 6, contenant 4 éléments :



Programme 8 : pile réalisée en C avec un tableau

```
typedef struct Pile {
    int capacité ;
    int tete ;
    int * donnees ;
} pile ;
```

Exercice 9 : (TP) Implémenter pileVide, depile et empile en O(1).

Exercice 10 : Si la pile n'est pas bornée, peut-on toujours la stocker dans un tableau ? Comment ?

Solution : tableau dynamique

I.3 - Applications des piles

I.3.1 - La pile d'exécution

Def 11 : Lors de l'exécution d'un programme, les paramètres, les variables locales et les appels de fonctions sont stockés sur

la pile d'exécution. En particulier, lors des fonctions récursives il faut faire attention aux débordements possibles de la pile (Stack_Overflow)

I.3.2 Parcours en profondeur (DFS)

Algo 12 : Parcours en profondeur (G, s0) :

```
P = creer_pile()
p.empiler(s0)
Tant que p est non vide :
    s = dépiler(P)
    Si s non marqué :
        marquer s
        afficher s
        pour (s, v) ∈ A :
            Si v non marqué : p.empiler(v)
```

Exercice 13 : Complexité ? réponse : $O(|S| + |A|)$

II - Les files

II.1 - Définition

Def 14 : Une file est une séquence $x_0 \dots x_{n-1}$ de valeurs du même type où il est possible de retirer l'élément le plus ancien (premier arrivé, premier servi ou FIFO). Elle dispose de trois fonctions :

- est_vide : renvoie true ssi la file est vide
- enfile : ajout d'un élément
- defile : retire l'élément le plus vieux et le renvoie

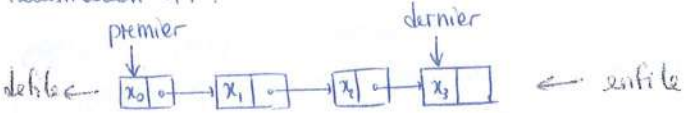
Analogie 15 : File d'attente dans un magasin

II.2 - Implémentations

II.2.1 - Avec une liste chaînée

Idée 16 : On forme la liste des éléments dans l'ordre de leur insertion, on maintient deux pointeurs vers les premier et dernier éléments de la liste.

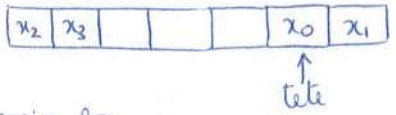
Illustration 17 :



II.2.2 - Avec un tableau circulaire

Idée 18 : Si la capacité de la file est bornée, on peut utiliser un tableau. Les éléments y sont rangés dans l'ordre d'insertion, et on garde un pointeur vers le premier élément.

Illustration 19 :



Exercice 20 :
Ecrire les fonctions est_vide, enfile et defile associées en $O(1)$.

II.2.3 - Avec deux piles

Idée 21 : Dans la première pile, les éléments qu'on insert. Dans la deuxième, on retire les éléments. Lorsque la seconde pile est vide, on y déplace tous les éléments de la première.

développement 1 : Complexité amortie de enfiler et défiler avec cette implémentation.

Nom : KANY Daphné Sujet : 5 Jury : Page : 3/4

II.3- Applications

II.3.1 - Stratégie d'ordonnancement

Dans votre CPU, un ordonnanceur décide quel processus s'éxecute sur le processeur. La stratégie du tourniquet utilise une file : les processus y sont insérés lors de leur creation puis s'éxecute à tour de rôle.

II.3.2 - Parcours en largeur d'un graphe (BFS)

Algo 13: Parcours en largeur (G, s0):

```
f = créer_file()
f.enfiler (s0)
Tant que f non vide:
    s = défiler (f)
    Si s non marqué:
        marquer s
        afficher s
        pour (s,v) ∈ A:
            Si v non marqué: f.enfiler(v)
```

Remarque 14: Similitude avec le parcours en profondeur

III. File de priorité

III.1 - Définition

Def 15: Une file de priorité est une structure dans laquelle on stocke un ensemble d'élements totalement ordonnés. Deux opérations sont fournies :

- Ajouter un nouvel élément dans la file de priorité
- retirer le plus petit élement de la file de priorité

III.2 - Implémentations

III.2.1- Implémentation naïve par liste

Exercice 16: Proposer une implémentation de file de priorité à l'aide du type list OCaml. Quelles sont les complexité de l'ajout et du retrait ? (O(1) et O(n))

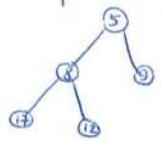
III.2.2 - Implémentation par tas

Def 17: Un arbre binaire contenant des éléments totalement ordonnés a une structure de tas si :

- il est vide
- il est de la forme N(l, x, r), les arbres l et r ont une structure de tas et x est inférieur ou égal à tous les noeuds de l et r.

Def 18: Un tas min est un arbre binaire ayant une structure de tas semi-complet à gauche.

Exemple 19:



est un tas binaire, représentable par le tableau | 5 | 8 | 9 | 17 | 12 |

Developpement 2 : Correction de l'insertion dans un tas-min

III.3 - Application : tri par tas

Principe 20 : Insérer tous les élements à trier dans une file de priorité implémentée par un tas-min, pour ensuite les ressortir du plus petit au plus grand.

Insertion dans un tas-min :
En tant que dernière feuille, puis inversion successive avec son père jusqu'à ce que la structure de tas soit respectée → $O(\log_2(n))$

Retrait du min : On met la dernière feuille à la place de la racine, puis inversion successive avec son plus petit fils ($O(\log n)$)

Ne rien écrire hors du cadre.