Question 6 : Guillaume Coutisse

Comment faire pour implémenter le type abstrait de données Pile à l’aide de

deux files ? Décrivez en particulier le fonctionnement des méthodes push et pop

dans ce cas. A titre d’exemple, précisez l’état de chacune des deux files après avoir

empilé les entiers 1 2 3 à partir d’une pile initialement vide. Décrivez ce qu’il

se passe ensuite lorsque l’on effectue l’opération pop. Quelle est la complexité

temporelle de ces méthodes si l’on suppose que chaque opération enqueue et

dequeue s’exécute en temps constant. Cette implémentation d’une pile est-elle

efficace par rapport aux autres implémentations présentées dans DSAJ-5 ?

Pour réaliser une pile à l’aide de deux files, nous devons trouver un moyen de retirer et enlever le 1er élément de la « pile » à l’aide des fonctions « enqueue » et « dequeue ». Pour se faire nous allons donc utiliser les deux files, la première est vide et la seconde contient les éléments mis dans la pile, on considère (via un boolean) que la pile est par exemple la seconde file. Les opérations « isempty » et « size » sont dont les même et doivent juste être appelé sur la bonne file (ce qu’on sait faire facilement via le boolean). Pour la méthode « pop », nous utilisons la fonction « dequeue », celle-ci permet de retirer et de retourner le 1er élément de la liste (qui se trouve être notre sommet de la pile) et de la retourner. Pour réaliser la méthode « push » nous allons avoir besoin des deux files. En effet la fonction « enqueue » rajoute un élément à la fin de la file, or nous voulons qu’il le rajoute au début de celle-ci (donc au sommet). Nous allons donc rajouter cet élément dans la file vide et recopier le contenu de l’autre liste via les fonctions « dequeue » et « enqueue » à la suite du nouvel élément. Il ne reste qu’à changer le boolean pour dire que notre pile est l’autre file.

Exemple :

File A :

File B :   
Pile (file B) :   
  
push(1) : A.enqueue(1) ;

File A : 1

File B :   
Pile (file A) : 1

push(2) : B.enqueue(2) ; B.enqueue(A.pop()) ;

File A : 1

File B : 2  
->

File A :

File B : 2 1

Pile (file B) : 2 1

push(3) : A.enqueue(3) ; A.enqueue(B.pop()) ; A.enqueue(B.pop()) ;

File A : 3

File B : 2 1  
->

File A : 3 2

File B : 1

->

File A : 3 2 1

File B :

Pile (file A) : 3 2 1  
  
On peut donc facilement voir que la complexité temporelle de « pop » est O(1) car il ne dépend pas de la taille des files, il se contente juste de retourner et retirer le 1er élément de la liste. Pour « push », la complexité temporelle est O(n) car il dépend de la taille de la seconde file qui doit être vidée et mise dans la 1er, il va donc effectuer l’opération de retirer et mettre un élément de la seconde dans la 1er file n fois pour les n éléments.

Enfin, vu que la complexité du push est plus élevée, l'implémentation est moins efficace que celle proposée dans le bouquin.