Fiche TD9 : Piles et files

Exercice 1 *Manipulation élémentaire de piles et files*

1. On considère une pile P initialement vide. Donner l'évolution de cette pile et décrire les éventuels résultats renvoyés au cours de la séquence d'opérations suivante : empiler(P, 2), empiler(P, 7), sommet(P), empiler(P, 4), est_vide(P), depiler(P), sommet(P).

2. On considère une file F contenant initialement un unique élément : a. Donner l'évolution de cette file et décrire les éventuels résultats renvoyés au cours de la séquence d'opérations suivante : enfiler(F, b), defiler(F), tete(F), defiler(F), est_vide(F), enfiler(F, a), enfiler(F, b).

Exercice 2 *Petits algorithmes sur les piles*

1. a) Rappeler les opérations de base que l'on doit pouvoir effectuer sur une pile.
   b) Quelle est la complexité de ces opérations si on implémente les piles avec un tableau auquel on adjoint un indice de sommet de pile ?

2. Utiliser ces opérations pour implémenter les algorithmes suivants. Les analyses de complexité partent du principe que les piles impliquées seront implémentées en utilisant un tableau.

   a) Ecrire une fonction qui intervertit lorsque c'est possible les deux éléments situés au sommet d'une pile.
   b) Ecrire une fonction prenant en argument une pile et renvoyant une pile constituée des mêmes éléments mais dans l'ordre inverse. Déterminer sa complexité temporelle et spatiale.
   c) Ecrire une fonction renvoyant une copie de la pile donnée en entrée. Attention, la pile en entrée doit être conservée. Déterminer sa complexité temporelle et spatiale.
   d) Ecrire une fonction recevant en arguments une pile P et un entier n et modifiant la pile P en effectuant dessus n permutations circulaires successives. Par exemple,

si $P =$ | 7 | 8 | 4 | 1 | 5 | et $n = 2$, P doit être modifiée de sorte à ce quelle vaille : | 4 | 1 | 5 | 7 | 8 |

   Déterminer la complexité temporelle et spatiale de votre algorithme.

Pour continuer à vous entraîner, vous pouvez reprendre la construction des algorithmes ci-dessus avec des files.

Exercice 3 *Implémentation de piles et files*

Les implémentations classiques de piles et de files vues en cours ne sont bien sûr pas les seules qu'on puisse imaginer ou vouloir utiliser :

1. Implémenter deux piles avec un unique tableau de taille n fixée. Les opérations empiler et dépiler devront s'effectuer en temps constant pour chacun des deux piles.

2. On souhaite implémenter une file à l'aide de deux piles P_1 et P_2 en utilisant l'idée suivante : le sommet de P_1 correspond à la tête de la file et le sommet de P_2 à son dernier élément.

   a) Quelle file est représentée par le couple $P_1 =$ | 1 | 8 | , $P_2 =$ | 10 | -4 | ?
   b) Implémenter chacune des opérations définissant une file en supposant connues celles pour une pile.
   c) A supposer qu'on a implémenté les piles de sorte à ce que toutes les opérations sur cette structure se fassent en temps constant, quelle est la complexité de vos précédents algorithmes ?

Exercice 4 *Automates à pile déguisés*

On considère des chaînes de caractères (qu'on appelera "mots") composées uniquement des caractères (et). Parmi ces mots on souhaite détecter les mots de Dyck c'est-à-dire ceux qui sont bien parenthésés. Par exemple, le mot ()(()()(())) est bien parenthésé alors que ni (() ni ())(ne le sont. Dans toute la suite, on pourra au besoin représenter toute suite de caractères par une liste contenant lesdits caractères.

1. Utiliser une pile pour construire un algorithme renvoyant vrai si son entrée est un mot bien parenthésé et faux sinon.

2. Ecrire un algorithme qui étant donné un mot bien parenthésé numérote les paires de parenthèses correspondantes avec le même entier en commençant la numérotation à 0. Par exemple, pour le mot ()(()()(())), on pourra avoir en sortie la liste $[0, 0, 1, 2, 2, 3, 4, 4, 3, 1]$.

On considère à présent qu'il est possible d'utiliser plusieurs parenthèses différentes qu'on identifie à l'aide d'un entier non nul. Plus précisément, une parenthèse ouvrante est représentée par un entier et la parenthèse fermante correspondante par l'opposé de cet entier ; par exemple si { est associée à 2, la représentation de } est -2. On considère une liste de caractères pouvant être ou non des parenthèses et on considère qu'on dispose d'une fonction `est_parenthese` qui renvoit vrai si et seulement si le caractère considéré est une parenthèse et d'une fonction `id_parenthese` qui renvoie l'entier représentant la parenthèse prise en entrée.

3. Ecrire une fonction prenant en entrée une liste de caractères et renvoyant vrai si et seulement si cette liste correspond à une expression bien parenthesée. On pourra utiliser une pile. Par exemple, cet algorithme devra renvoyer faux sur l'entrée $(x + 2) \times \{x - (3 \times y\}$ et vrai sur l'entrée $(x + [1 - y]) \times \{3 + 7\}$.

Exercice 5 *Automates à pile déguisés, bis*

On considère des chaînes de caractères n'impliquant que les lettres a et b. On s'obligera à utiliser une pile pour répondre aux questions suivantes (même si en l'occurrence, ne pas utiliser de pile permettrait de conclure tout de même).

1. Ecrire une fonction renvoyant vrai si la chaîne (qu'on pourra implémenter par une liste de caractères au besoin) en entrée appartient à l'ensemble $\{a^n b^n \mid n \in \mathbb{N}\}$ où a^n dénote la chaîne $\underbrace{aaa....aaa}_{n \text{ fois}}$.

2. Ecrire une fonction renvoyant vrai si la chaîne en entrée contient exactement le même nombre de a que de b. Par exemple, elle devra renvoyer vrai sur l'entrée *abba* et faux sur l'entrée *abbab*.

Remarque : Vous verrez en deuxième année que les ensembles de mots considérés dans les exercices 4 *et* 5 *sont ce qu'on appelle des "langages algébriques". Ces langages sont exactement ceux pouvant être reconnus — c'est-à-dire détectés — par un automate à pile, structure que vous étudierez également en deuxième année.*

Exercice 6 *Evaluation d'expressions arithmétiques*

Dans cet exercice, on appelle expression arithmétique une liste dont les éléments ne peuvent être que : un entier ou un opérateur binaire de $\{+, -, \times\}$. Ces expressions arithmétiques seront écrites en notation postfixes (cette notation est aussi nommée "notation polonaise inverse", ce en raison de la nationalité d'un de ses inventeurs, Jan Łukasiewicz). Certaines vieilles calculatrices l'utilisent toujours. Cette notation consiste à placer les opérandes avant les opérateurs et a l'avantage de pouvoir se passer de parenthésage contrairement à la notation infixe. Par exemple, l'expression $(5 + 7) \times (1 - 8)$ s'écrit en notation postfixe : $5\ 7\ +\ 1\ 8\ -\ \times$.

1. Donner une notation postfixe à l'expression dont l'écriture infixe est $(3 + 8 - 2) \times (0 - 7)$.

On cherche à construire un algorithme permettant d'évaluer une expression arithmétique (cet algorithme renverra par exemple -84 sur la première expression de l'énoncé). Pour ce faire, on utilise une pile P de la façon suivante : on lit l'expression de gauche à droite, tant qu'on rencontre un nombre, on l'empile sur P. Si on rencontre un opérateur, on dépile deux éléments de P, on applique l'opérateur aux éléments dépilés et on empile le résultat du calcul.

2. Donner l'évolution de l'état de la pile lorsqu'on applique cet algorithme à l'écriture postfixe trouvée à la première question. Que constatez vous ?

3. En déduire un algorithme récursif permettant d'évaluer une expression en notation postfixe.

4. Quelle est la complexité de cet algorithme ?