TD 2. Structures de données linéaires

Exercice 1. *Échauffement*

1. Décrire comment implanter une pile et une file à l'aide d'une file de priorité.
2. Soit T un tas à n éléments. Les affirmations suivantes sont-elles correctes ?
   - **i.** $T_{[0]}$ est l'élément maximal.
   - **ii.** $T_{[n-1]}$ est l'élément minimal.
   - **iii.** L'élément minimal est dans une case d'indice $\geq (n-1)/2$.
3. **i.** Parmi les tableaux suivants, identifier les tas : $[25, 18, 13, 7, 12, 10, 6, 9, 7]$, $[17, 11, 7, 13, 10, 4]$, $[21, 6, 12, 13, 7, 2, 4, 5, 8]$. *Dessiner l'arbre quasi-complet représenté par le tableau, puis indiquer si cet arbre binaire est un tas.*
   ii. Pour ceux qui ne le sont pas, identifier le ou les nœuds mal placés, et appliquer soit REMONTER soit TASSER pour transformer le tableau en tas.
4. Montrer qu'un tableau trié par ordre décroissant est un tas.
5. Monter que les feuilles d'un tas à n éléments sont exactement les nœuds de numéro $\lfloor n/2 \rfloor$ à $n-1$.

Exercice 2. *File et tableaux circulaires*

On rappelle que dans l'implantation d'un file à l'aide d'un tableau T de taille N, on stocke deux indices (d, f) pour indiquer que les éléments de la file sont dans le sous-tableau $T_{[d,f[}$. Après un certain nombre d'appels à ENFILER et DÉFILER, on peut avoir d et f proches de N : cela signifie que les premières cases du tableau sont inutilisées. Pour éviter ce gâchis, on peut utiliser T de manière *circulaire* : si on atteint $f = N$, on *boucle* en utilisant le début du tableau.

1. On utilise un tableau de taille $N = 4$, et on effectue la suite d'opérations suivantes : ENFILER(a, F), ENFILER(b, F), ENFILER(c, F), DÉFILER(F), ENFILER(d, F), DÉFILER(), DÉFILER(), ENFILER(e, F), ENFILER(f, F).
   - **i.** Si on utilise le tableau de manière *non-circulaire*, quelle opération renvoie une erreur ?
   - **ii.** Décrire l'état du tableau après chaque opération s'il est utilisé de manière circulaire ?
   - **iii.** Exprimer, en français, la condition pour qu'il n'y ait pas d'erreur dans les deux cas : tableau circulaire et non-circulaire.
2. Écrire l'implantation des quatre opérations de File, en utilisant un tableau de manière circulaire.

Exercice 3. *Expressions bien parenthésées*

On considère des chaînes de caractères avec des parenthèses de plusieurs sortes : `()`, `[]`, `{}`, ... On souhaite déterminer si la chaîne est *bien parenthésée* : une parenthèse ouvrante doit correspondre à une parenthèse fermante dans le bon ordre : ") (" n'est pas valide ; et les parenthèses doivent être correctement imbriquées : "`([])`" est valide mais pas "`([)]`".

1. Déterminer les chaînes bien parenthésées parmi les chaînes suivantes : "`()([])[]{}`", "`{{{}[{}]}()}`", "`(){}[]({[]}))`".

On suppose avoir une fonction TYPE(c) qui prend en entrée un caractère c et renvoie le type de parenthèse dont il s'agit : le type est un entier, positif si la parenthèse est ouvrante et négatif sinon ; si c est un parenthèse ouvrante de type $t > 0$, la parenthèse fermante correspondante est de type $-t$.

2. Écrire une fonction qui teste si une expression est bien parenthésée, en utilisant une pile.

Exercice 4. *Parcours d'arbre binaire*

1. Ré-écrire les algorithmes de parcours en profondeur d'un arbre binaire sous forme itérative. *Utiliser une pile pour stocker les nœuds à visiter.*
2. i. Écrire l'algorithme de parcours en largeur d'un arbre binaire. *Utiliser une file pour stocker les nœuds à visiter.*
   ii. Justifier que si l'arbre est quasi-complet, l'ordre de traitement des sommets (en commençant à 0) fournit à chaque sommet un numéro NUM(x) tel que NUM(PARENT(x)) $= \lfloor (\text{NUM}(x) - 1)/2 \rfloor$.

Exercice 5. *File avec deux piles*

On souhaite implanter le TAD File à l'aide de deux piles. On définit donc $F = (E, S)$ où E est la *pile d'entrée* et S la *pile de sortie*. Quand on enfile un élément dans F, on l'empile sur E. Pour défiler, si S est vide, on transfère tout le contenu de E dans S, puis ensuite on dépile un élément de S.

1. Écrire les opérations FILEVIDE, ESTVIDE et ENFILER à partir des opérations sur les piles. Quelles sont leurs complexités ?
2. i. Écrire l'opération DÉFILER à partir des opérations sur les files.
   ii. Justifier que la file implantée suit bien la politique FIFO.
   iii. Quelle est la complexité dans le *pire des cas* de l'opération DÉFILER ? Et dans le *meilleur des cas* ?
3. On souhaite montrer que les opérations ENFILER et DÉFILER ont une complexité *amortie* $O(1)$, c'est-à-dire que partant d'une file vide, toute suite de N opérations ENFILER/DÉFILER coûte *au total* $O(N)$ (d'où $O(1)$ par opération).
   i. Supposons qu'il y a m opérations ENFILER et n opérations DÉFILER ($m + n = N$). Quelle relation doivent satisfaire m et n ?

On change de point de vue : au lieu de se concentrer sur les opérations, on se concentre sur les éléments qui sont dans la file. Soit e un élément qui passe dans F pendant la suite d'opérations.

ii. Combien de fois, au plus, l'opération EMPILER est appelée sur e ?
iii. Si e est extrait de F, combien de fois e a-t-il été renvoyé par un appel à DÉPILER ? Et si e reste dans F à la fin des opérations ?
iv. En déduire qu'une suite de N opérations ENFILER/DÉFILER effectue au plus $3N$ appels à EMPILER/DÉPILER.

Exercice 6. *Évaluation d'expressions arithmétiques*

On considère des expressions arithmétiques de la forme $(1 + ((2 + 3) \times (4 \times 5)))$, complètement parenthésées et ne contenant que des opérateurs binaires. On souhaite les évaluer, c'est-à-dire trouver 101 par exemple ici. Il existe un algorithme très simple dû à E. W. Dijkstra pour cela. L'idée est la suivante. Premièrement, on suppose (ou vérifie) que l'expression est bien parenthésée. Ensuite, on utilise deux piles : celles des opérandes et celle des opérateurs ($+$, $\times$, ...). On ignore les parenthèses ouvrantes. On parcourt l'expression et on applique les règles suivantes :

1. si on lit un entier, on l'empile sur la pile des opérandes ;
2. si on lit un opérateur, on l'empile sur la pile des opérateurs ;
3. si on lit une parenthèse fermante, on dépile un opérateur et deux opérandes ; on effectue le calcul et on empile le résultat sur la pile des opérandes.

1. Appliquer l'algorithme sur l'expression ci-dessus.
2. Écrire l'algorithme. *On pourra utiliser une boucle de la forme « Pour chaque caractère de l'expression : » et on supposera avoir des tests « est une parenthèse ouvrante », etc.*
3. On veut justifier que l'algorithme fonctionne.

i. Justifier qu'une expression complètement parenthésée décrit de manière unique un arbre binaire.
ii. Dessiner l'arbre correspondant à l'expression ci-dessus. Appliquer les trois parcours en profondeur sur l'arbre obtenu. À quel parcours correspond l'expression parenthésée ?
iii. Interpréter l'algorithme en terme de modification d'arbre binaire. En déduire sa correction.

Exercice 7. *Créer un tas*

1. En se basant sur l'algorithme TRITAS, écrire un algorithme CRÉERTAS qui prend en entrée un tableau T et réordonne ses éléments pour que T représente un tas.
2. Montrer que la complexité de CRÉERTAS est $O(n \log n)$.

3. Exécuter l'algorithme CRÉERTAS sur le tableau $[1, 2, 3, 4, 5, 6, 7]$.
4. *(bonus)* On veut montrer que CRÉERTAS a une complexité $O(n)$.
   i. Pourquoi cela ne contredit-il pas la question~2 ?
   ii. Montrer que pour $h < \log n$, CRÉERTAS appelle TASSER $\lceil n/2^{h+1} \rceil$ fois sur des tas de hauteur h.
   iii. En déduire que la complexité de CRÉERTAS est $O\left(\sum_{h=0}^{\lfloor \log n \rfloor} nh/2^h\right)$.
   iv. Montrer que $\sum_{h=0}^{+\infty} h/2^h = 2$.
   v. En déduire le résultat.

Exercice 8. *Files de priorité et tri*

1. Supposons qu'on dispose d'une implantation du TAD file de priorité.
   i. Écrire un algorithme de tri basé sur la file de priorité.
   ii. Analyser la complexité de l'algorithme en fonction des complexités des opérations de la file de priorité.
2. On s'intéresse aux algorithmes obtenus en fonction de l'implantation utilisée. Pour chacune des implantations ci-dessous, donner la complexité obtenue pour l'algorithme et identifier à quel algorithme classique correspond cette méthode.
   i. Tableau non trié.
   ii. Tableau trié.
   iii. Tas.