TD1 : structures de données linéaires

Exercice 1 : Suppression dans des listes chaînées

Nous représentons une liste simplement chaînée avec une classe `List` donnant accès à un attribut `head` de type `Cell`. Cette classe `Cell` possède deux attributs, un attribut `next` de type `Cell` et un attribue `value` stockant la valeur associée à cette cellule de la liste chaînée.

Nous souhaitons pouvoir supprimer une cellule de la liste chaînée.

Q 1.1 Commencer par dessiner une liste chaînée contenant, dans l'ordre, les valeurs 5, 3, 4. Dessiner les chaînages à modifier lors de la suppression de la cellule contenant la valeur 3. De même, en partant de la liste initiale, dessiner les chaînages à modifier lors de la suppression de la cellule contenant la valeur 5.

Nous souhaitons implémenter une méthode `delete` procédant à la suppression d'une cellule de la liste chaînée.

Q 1.2 Dans quelle classe devrait être cette méthode ?

Q 1.3 Pourquoi seulement passer la cellule à supprimer en paramètre de cette méthode n'est pas suffisant pour assurer une suppression efficace ?

Q 1.4 Écrire le (pseudo-)code de la méthode `delete`, qui prend donc en paramètre la cellule courante, à supprimer, et la cellule qui la précède, si elle existe.

Intéressons-nous maintenant au cas où la liste est doublement chaînée. Nous avons une classe `DoubleList` ayant deux attributs `head` et `tail` faisant respectivement référence à la `DoubleCell` au début et à la fin de la liste. La classe `DoubleCell` possède trois attributs : `previous` et `next`, respectivement la `DoubleCell` qui précède et qui suit la cellule courante, et `value`, correspondant à la valeur associée à cette cellule de la liste chaînée.

Q 1.5 Commencer par dessiner une liste doublement chaînée contenant, dans l'ordre, les valeurs 5, 3, 4. Dessiner les chaînages à modifier lors de la suppression de la cellule contenant la valeur 3. De même, en partant de la liste initiale, dessiner les chaînages à modifier lors de la suppression de la cellule contenant la valeur 5. De même, en partant de la liste initiale, dessiner les chaînages à modifier lors de la suppression de la cellule contenant la valeur 4.

Q 1.6 Donner le code de la méthode `delete` pour des listes doublement chaînées

Exercice 2 : Liste circulaire

On souhaite implanter une liste circulaire, c'est-à-dire que le suivant du dernier élément est le premier élément de la liste. On accédera à cette structure de donnée grâce à une primitive `valeur` qui prend en entrée la liste, retourne la valeur de l'élément courant et a pour effet de bord que le prochain appel à `valeur` donnera la valeur de l'élément suivant.

Par exemple, si la liste contient les éléments 6,9,0,3,1 alors si un appel à `valeur` retourne 3, l'appel suivant retournera 1, l'appel suivant retournera 6, etc.

Q 2.1 Proposer une définition d'une structure de données à base de tableau. On supposera que la fonction d'ajout est telle que les cases vides (sans élément) dans le tableau sont contiguës (modulo).

Q 2.2 Proposer avec cette structure un pseudocode de la fonction `valeur`.

Q 2.3 Proposer une définition d'une structure de données à base de liste simplement chaînée.

Q 2.4 Proposer avec cette structure un pseudocode de la fonction `ajouter` qui permet d'ajouter un élément dans la liste circulaire. Discuter de l'endroit où ajouter cet élément.

Q 2.5 Proposer un pseudocode de la fonction `valeur`.

Exercice 3 : Implantation de files

On souhaite implanter une file avec une liste simplement chaînée. On suppose avoir accès aux primitives et aussi à la structure de donnée elle-même, ce qui signifie qu'on peut modifier la liste sans passer par les primitives.

Q 3.1 Comment réaliser l'opération `enfiler` sans utiliser de boucle implicitement ou explicitement ?

On change maintenant notre représentation de file et on s'autorise à encapsuler la liste dans une autre structure de données afin de pouvoir réaliser l'opération `defiler` sans utiliser de boucle implicitement ou explicitement.

Q 3.2 Quelle information faut-il conserver pour garantir cela ?

Q 3.3 Énumérer les cas limites et expliquer comment les traiter.

Q 3.4 Écrire le code (ou pseudo-code) de `defiler`.

Exercice 4 : Files et piles

En utilisant uniquement les opérations primitives sur chacune des deux structures de données.

Q 4.1 Montrer que l'on peut implanter une pile avec deux files.

Q 4.2 Quel est le nombre d'itérations effectuées pour les opérations empiler et dépiler dans ce cas, en supposant que les opérations primitives sur les files ne font aucune itération ?

Q 4.3 Montrer que l'on peut implanter une file avec deux piles.

Q 4.4 Quel est le nombre d'itérations effectuées pour les opérations enfiler et défiler, en supposant que les opérations primitives sur les piles ne font aucune itération ?

Exercice 5 : Tri par sélection avec des piles

On rappelle le principe du tri par sélection du minimum :

```
soit E une suite d'elements a trier
soit F une suite vide
tant qu'il reste des elements dans E
  soit x l'element minimum de E
  ranger x dans F (de maniere ordonnee)
fin tant que
retourner F
```

On souhaite utiliser cet algorithme pour trier les éléments d'une pile (le plus petit élément sera au fond de la pile et le plus grand au sommet). La pile est la seule structure de données autorisée dans cet exercice (on n'aura droit ni aux tableaux, ni aux files, ni aux listes). Ainsi votre algorithme pourra utiliser d'autres piles et bien sûr des variables.

Q 5.1 Décrire un algorithme en français réalisant le tri des éléments d'une pile utilisant le principe du tri par sélection du minimum.

Q 5.2 Quelle est le nombre total d'itérations effectuées ?

Exercice 6 : Liste avec tableaux

On se propose dans cet exercice d'implanter une structure de liste grâce à des tableaux. On sait que cette implantation possède l'inconvénient majeur de ne pas être extensible sans recopie d'éléments mais l'avantage de pouvoir accéder au i-ème élément en temps constant. Nous allons proposer une structure de données pour tenter de remédier au premier problème tout en conservant de bonnes propriétés pour l'accès au i-ème élément.

L'idée proposée est la suivante : lorsque le tableau d'éléments sera rempli, nous allons créer un second tableau d'éléments, puis lorsque le second tableau d'éléments sera rempli nous créerons un troisème tableau, etc. Nous préservons ainsi l'extensibilité de la structure de données. Pour accéder au i-ème élément, il suffira de trouver le bon tableau.

On a donc au final une liste de tableaux d'éléments au lieu d'une liste d'éléments.

On supposera que la taille des tableaux est fixée à une constante M.

Q 6.1 Dessiner la structure de données si $M = 5$ et la liste est composée des éléments 1 à 12 insérés en tête dans l'ordre croissant.

Q 6.2 Dans quel tableau et à quel indice se trouve le $i-$ème élément de la liste ?

Q 6.3 En supposant que chacun des n éléments de la liste possède la même probabilité d'être accédé, quel sera le coût moyen d'accès à un élément de la liste ? (indication : il suffit de faire la moyenne du coût d'accès au 1^{er} élément, au 2^{e}, etc).