Grands Moulins

INITIATION À LA PROGRAMMATION (IP2)

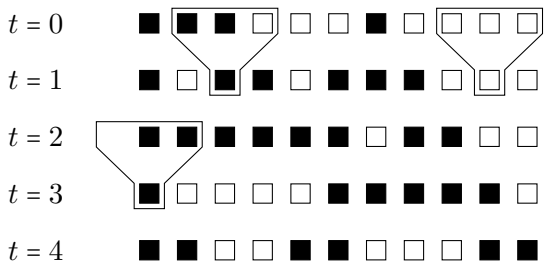
TP 6 - 7 : Liste doublement chaînée et automate cellulaire

Le but de ce TP est d'implémenter une version simplifiée d'automate cellulaire, vu alors comme une liste doublement *chaînée de booléens, qui indique si la cellule correspondante est vivante ou morte.*

Exercice 1 [Liste doublement chaînée]

1. Créez une classe `Cellule` contenant :
   (a) trois attributs privés : `precedente` et `suivante` de type `Cellule`, `noire` de type `boolean` ;
   (b) les accesseurs pour chacun des attributs ;
   (c) un constructeur `Cellule(boolean noire)` qui initialise l'attribut `noire` avec l'argument donné, et fixe les deux autres attributs à `null` ;
   (d) une méthode `void afficher()` qui imprime (sans retour à la ligne) un dièse `#` ou un tiret `-` selon que `noire` est vraie ou fausse.
2. Définissez une classe de liste autour de ces cellules, avec références vers début et fin, et écrivez deux méthodes qui prennent en argument un booléen et permettent d'ajouter en début ou en fin une cellule correspondante.
3. Ecrivez une méthode d'affichage pour les listes , et testez votre code en définissant une séquence correspondant à la première ligne de la figure ci dessous.

Exercice 2 En préambule nous décrivons le fonctionnement attendu d'un automate cellulaire.
À chaque instant t, chaque cellule est soit noire (`noire==true`) soit blanche (`noire==false`).
L'état à l'instant $(t+1)$ d'une cellule donnée dépend de l'état à l'instant t de ses voisines, ainsi que de son propre état à l'instant t. Considérons ici par exemple la règle de l'*unanimité*, c'est-à-dire que la cellule d'indice i à l'étape $(t+1)$ est blanche ssi les cellules $(i-1)$, i et $(i+1)$ portent toutes les trois le même état à l'étape t.
Par convention la cellule virtuellement à gauche de la première est toujours considérée comme blanche (de même pour la cellule virtuellement à droite de la dernière).
Sont encadrés ci-dessous des exemples d'application de la règle de l'unanimité.
À $t = 1$, la cellule 3 est noire car, à $t = 0$, les états des cellules 2, 3 et 4 ne sont pas identiques.
De même l'avant-dernière cellule est blanche à $t = 1$ car ses deux voisines ont le même état que le sien à $t = 0$.
La première cellule du temps 3 est noire : cela illustre le fait que la première cellule considère que sa voisine (virtuelle) de gauche était blanche au temps 2.



1. Renommez votre classe de liste en `Automate`.

2. Ajoutez un constructeur `Automate(String str)` qui prend une chaîne de caractères constituée de `'#'` et de `'-'`, et crée l'automate correspondant.

   On veut créer une fonction qui change la couleur des cellules en fonction du temps.
   Il n'est pas possible de le faire avec un seul parcours car la mise à jour prématurée d'une cellule peut changer le résultat de la mise à jour de sa voisine. La solution retenue consiste à enrichir le modèle en ajoutant un attribut, et faire plusieurs passages.
3. Ajoutez à la classe `Cellule` un attribut `prochainEtat` de type `boolean`. Modifier les constructeurs pour que ce nouvel attribut soit toujours initialisé à `false`.
4. Ajoutez à l'automate une méthode `prochaineEtape()` qui met à jour `prochainEtat`, pour chacune de ces cellules, en suivant la règle de l'unanimité.
5. Ajoutez ensuite une méthode `miseAJour()` qui transfère les valeurs de `prochainEtat` dans `noire` pour chacune des cellules de l'automate.
6. Créez dans la classe `Automate` la méthode `uneEtape()` qui combine les deux méthodes précédentes.
7. Ajoutez une méthode `nEtapes(int n)` qui affiche d'abord l'état courant, puis effectue n étapes successives, en affichant les étapes intermédiaires.
8. Testez avec `n` valant `4` et comparez avec la figure de l'énoncé.
9. Testez avec diverses chaînes de caractères, en particulier celle avec un seul `#` au milieu.
10. Organisez votre code pour qu'on puisse construire des automates qui utilisent soit la règle de l'unanimité, soit celle de la majorité que vous écrirez également.