Université Pierre et Marie Curie

Master de Sciences et Technologies

Méthodes Informatiques en Biologie - BMC427

Année M1 - Session de Juin 2007

Durée 2h ; tous documents autorisés

Note sur 60 ; le barême est donné à titre indicatif

A. Première partie (30)

Une cellule d'algue verte possède des chloroplastes, qui produisent de l'oxygène à partir du gaz carbonique en présence de lumière. Par ailleurs, la cellule respire (qu'elle soit éclairée ou non) et consomme donc de l'oxygène et produit du gaz carbonique.
On calcule la production d'oxygène par chloroplaste, mais on évalue la respiration globalement pour la cellule.
On veut évaluer le bilan gazeux de cette cellule au cours d'un cycle de 24 heures.

A.1. (10) Quels sont les objets qui apparaissent dans l'analyse du problème ? Quelles sont leurs propriétés et leurs relations ?
Traduire cette analyse par une architecture de classes Java, dans laquelle on définira les attributs et les méthodes pertinents pour chaque objet (les méthodes ne seront présentées que par leur interface, le code étant remplacé par un commentaire).

A.2. (8) On évalue les échanges gazeux globaux d'une culture cellulaire. Sous quelle forme peut-on représenter cette culture dans une classe Java ?

A.3. (12) On mélange maintenant dans la même culture des algues vertes et des algues dépourvues de chlorophylle. Comment peut-on représenter ces deux types de cellules, et comment représente-t-on la culture dans un code Java ?

B. Deuxième partie (38)

On a écrit le code de plusieurs classes dans le but de réaliser un programme d'analyse de séquences nucléotidiques (voir page suivante).

B.1. (4) Écrire pour la classe `Nucleotide` la méthode `complem()` qui permet à un nucléotide de retourner son nucléotide complémentaire (on considérera que le complémentaire du nucléotide indéfini `*` est `*`).

B.2. (8) Utiliser cette méthode pour écrire dans la classe `ADN` une méthode `invcompl()` qui retourne la séquence inverse-complémentaire. Naturellement, il faut respecter le sens habituel de lecture, de 5' vers 3' : l'inverse-complémentaire de `GCATAGACGACTA` est `TAGTCGTCTATGC`.

B.3. (6) Enfin, écrire dans `Analyse` l'appel à `invcompl()` de façon à obtenir l'inverse-complémentaire de la séquence traitée.

B.4. (10) Écrire dans la classe `ADN` la méthode `palindr(int d, int l)`, qui retourne `true` si la séquence comporte à partir du nucléotide en position *d* un palindrome de longueur *2l*.
NB. en Biologie Moléculaire, on appelle palindrome une séquence auto-complémentaire. Ainsi `GAATTC` est un palindrome de longueur 6.

B.5. (10) Ajouter dans `Analyse` la recherche de tous les palindromes de longueur 4 ou plus. Le programme devra donner pour chaque palindrome trouvé sa position et sa longueur.

Nucleotide.java

```
class Nucleotide {

  private char symbole;

  public Nucleotide(char nuc) {
    if( ( nuc == 'A' ) || ( nuc == 'a' ) )
      symbole = 'A';
    else
      if( ( nuc == 'C' ) || ( nuc == 'c' ) )
        symbole = 'C';
      else
        if( ( nuc == 'G' ) || ( nuc == 'g' ) )
          symbole = 'G';
        else
          if( ( nuc == 'T' ) || ( nuc == 't' ) )
            symbole = 'T';
          else
            symbole = '*';
  }

  public char getSymb() {
    return symbole;
  }

}
```

ADN.java

```
class ADN {

  private Nucleotide[] sequence;

  public ADN (String seq) {
    int l = seq.length();
    sequence = new Nucleotide[l];
    for(int c=0; c<l; c++)
      sequence[c]=new Nucleotide(seq.charAt(c));
  }

}
```

Analyse.java

```
class Analyse {

  public static void main(String args[]) {
    ADN adn1 = new ADN(args[0]);
  }

}
```