Algorithmique avancée

Corrigé du TP2

J.M. Dischler

Algorithmes sur les nombres premiers

Nous considérons des ensembles d'objets de type entiers positifs ou nuls. L'ensemble sera représenté à l'aide de la class *ArrayList*. Cette classe propose des opérations d'adjonction, de suppression et de recherche d'éléments.

Une documentation est disponible sous :

`http://www.tutorialspoint.com/java/java_arraylist_class.htm`

1. Ecrire un algorithme qui permette de remplir l'ensemble avec les n premiers nombres premiers. On commencera par écrire un algorithme naïf utilisant une fonction qui teste si oui ou non un entier est premier.
2. Ecrire ensuite un algorithme qui utilise l'ensemble lui même. En effet, un nombre n'est pas premier s'il est divisible par un nombre premier.

3. Ecrire une fonction récursive qui décompose un entier inférieur à n en ses diviseurs premiers. On suppose que l'ensemble des n premiers nombres premiers a préalablement été généré.

Refaire l'exercice en utilisant la class *LinkedList* à la place de *ArrayList*.

Chaînes de caractères

On considère le problème suivant : soit deux chaînes de caractères s et t. On se propose de comparer ces deux chaînes sachant que la première chaîne peut contenir des caractères spéciaux : $\sharp$, $\star$ et ?. Le $\sharp$ remplace un ou plusieurs caractères, le $\star$ zéro ou plusieurs caractères et le ? un caractère. La fonction de comparaison renvoie un booléen, indiquant si la première chaîne peut correspondre à la seconde (on parle de *pattern matching*).

1. Proposer un algorithme récursif. Montrer que la complexité au pire est exponentielle.

```
/* Version recursive, complexite au pire si que des * dans s1, d'ou en O(l1*2^l2)*/
bool jocker(char *s1, int p1, char *s2, int p2)
{
if (p1==-1 && p2==-1) return true;
if (p1==-1) return false;
if (s1[p1]=='*')
{
if (p2==-1) return jocker(s1,p1-1, s2, p2);
return jocker(s1,p1-1, s2, p2) || jocker(s1, p1, s2, p2-1);
}
if (s1[p1]=='#')
{
if (p2==-1) return false;
return jocker(s1,p1-1, s2, p2-1) || jocker(s1, p1, s2, p2-1);
}
if (p2==-1) return false;
```

```
if (s1[p1]=='?') return jocker(s1, p1-1, s2, p2-1);
if (s1[p1]==s2[p2]) return jocker(s1, p1-1, s2, p2-1);
return false;
}
```

2. Proposer un algorithme basé sur le principe de la programmation dynamique. Quelle est sa complexité.

```
/* Version dynamique, complexite en Theta(l1*l2)*/
bool dyna_jocker(char *s1, int l1, char *s2, int l2)
{
bool JOCKER[100][100];
int p1,p2;

for (p1=0; p1<=l1+1; p1++) { JOCKER[p1][0]=false; }
for (p2=0; p2<=l2+1; p2++) { JOCKER[0][p2]=false; }
JOCKER[0][0]=true;

for (p1=0; p1<=l1; p1++)
for (p2=-1; p2<=l2; p2++)
{
if (s1[p1]=='*')
{
if (p2==-1) JOCKER[p1+1][p2+1]=JOCKER[p1][p2+1];
else JOCKER[p1+1][p2+1]=JOCKER[p1][p2+1]||JOCKER[p1+1][p2];
}
else if (s1[p1]=='#')
{
if (p2==-1) JOCKER[p1+1][p2+1]=false;
else JOCKER[p1+1][p2+1]=JOCKER[p1][p2] || JOCKER[p1+1][p2];
}
else
{
if (p2==-1) JOCKER[p1+1][p2+1]=false;
else if (s1[p1]=='?') JOCKER[p1+1][p2+1]=JOCKER[p1][p2];
else if (s1[p1]==s2[p2]) JOCKER[p1+1][p2+1]=JOCKER[p1][p2];
else JOCKER[p1+1][p2+1]=false;
}
}
```