Complexité

Sujet Examen terminal

2010-2011

Durée 2H

-Documents autorisés-

Exercice 1 : machine de Türing

Soit la machine de Turing M = (Σ, Q, q0, Q', Γ) avec : Σ={0, 1, #}, Q={q0, q1, q2, qF, qV}, q0 = q0, Q' = { qF, qV }, Γ :

Q	0	1	#
q0	q1, 0, D	qF, 1, D	qV, #, D
q1	qF, 0, D	q2, 1, D	qF, #, D
q2	qF, 0, D	q0, 1, D	qF, #, D

1. Donnez la suite des configurations de M obtenues pour les mots d'entrée 0110111, 011011
2. Quel langage M décide-t-elle ?

Exercice 2 : machine de Türing

Décrire une machine de Türing qui reconnait le langage suivant : L = {a^{n1} b^{n2} c^{n3} | 0<n1<n2<n3 } à partir de l'alphabet Σ = {a, b ,c}

Exercice 3 : le problème du sac à dos

On cherche à remplir un sac-à-dos avec un certain nombre d'objets de façon à le remplir exactement. On peut modéliser ce problème de la façon suivante: on se donne *n* entiers strictement positifs a_i et un entier *S*. Existe-t-il des nombres $x_i \in \{0, 1\}$ tels que

$$S = x_0\, a_0 + x_1\, a_1 + \cdots + x_{n-1}\, a_{n-1}?$$

Si x_i vaut 1, c'est que l'on doit prendre l'objet a_i, et on ne le prend pas si x_i=0.

1. **Version 1 : donner la complexité de l'algorithme suivant**

```
   // a[0..n[ : existe-t-il x[] tel que
   // somme(a[i]*x[i], i=0..n-1) = S ?
 static void sacADosn(int[] a, int S){
     int n = a.length, N;
     int[] x = new int[n];

     for(int i = 0; i < 2^n; i++){
         // reconstruction de N = somme x[i]*a[i]
         N = 0;
         for(int j = 0; j < n; j++)
             if(x[j] == 1)
                 N += a[j];
         if(N == S){
             System.out.print("S="+S+"=");
             for(int j = 0; j < n; j++)
                 if(x[j] == 1)
                     System.out.print("+"+a[j]);
             System.out.println("");
         }
```

```
            // simulation de l'addition
            for(int j = 0; j < n; j++){
                if(x[j] == 1)
                    x[j] = 0;
                else{
                    x[j] = 1;
                    break; // on a fini
                }
            }
        }
    }
```

2. Version recursive : donner la complexité de l'algorithme suivant

```
// on a déjà calculé Si = sum(a[j]*x[j], j=0..i-1)
    static void sacADosrec(int[] a, int S, int[] x,
                           int Si, int i){
        nbrec++;
        if(Si == S)
            afficherSolution(a, S, x, i);
        else if((i < a.length) && (Si < S)){
            x[i] = 0;
            sacADosrec(a, S, x, Si, i+1);
            x[i] = 1;
            sacADosrec(a, S, x, Si+a[i], i+1);
        }
    }
static void sacADos(int[] a, int S){
        int[] x = new int[a.length];

        nbrec = 0;
        sacADosrec(a, S, x, 0, 0);
        System.out.print("# appels=" + nbrec);
        System.out.println(" // " + (1 << (a.length + 1)));
    }
```

Exercice 4 : rechercher dans du texte

1. Algorithme naïf

```
static boolean occurrence(char[] T, char[] M, int i){
    for(int j = 0; j < M.length; j++)
        if(T[i+j] != M[j]) return false;
    return true;
}

static void naif(char[] T, char[] M){
    System.out.print("Occurrences en position :");
    for(int i = 0; i < T.length-M.length; i++)
        if(occurrence(T, M, i))
            System.out.print(" "+i+",");
    System.out.println("");
}
```

   Donner la complexité de cet algorithme en notant n la taille de T et m la taille de M.

2. Algorithme linéaire de Karp-Rabin : 1ère version

```
static long signature(char[] X, int m, int i){
        long s = 0;

        for(int j = i; j < i+m; j++)
            s += (long)X[j];
        return s;
    }
```

```
static void KR(char[] T, char[] M){
    int n, m;
    long hT, hM;

    n = T.length;
    m = M.length;
    System.out.print("Occurrences en position :");
    hM = signature(M, m, 0);
    for(int i = 0; i < n-m; i++){
        hT = signature(T, m, i);
        if(hT == hM){
            if(occurrence(T, M, i))
                System.out.print(" "+i+",");
            else
                System.out.print(" ["+i+"],");
        }
    }
    System.out.println("");
}
```

Donner la complexité de cet algorithme

3. Algorithme linéraire de Karp-Rabin : 2ème version

```
static long B = ((long)1) << 16, p = 2147483647;

 // calcul de S(X[i..i+m-1])
 static long signature2(char[] X, int i, int m){
     long s = 0;

     for(int j = i; j < i+m; j++)
         s = (s * B + (int)X[j]) % p;
     return s;
 }

 // S(X[i+1..i+m]) = B S(X[i..i+m-1])-X[i] B^m + X[i+m]
 static long signatureIncr(char[] X, int m, int i,
                           long s, long Bm){
     long ss;

     ss = ((int)X[i+m]) - ((((int)X[i]) * Bm)) % p;
     if(ss < 0) ss += p;
     ss = (ss + B * s) % p;
     return ss;
 }

 static void KR2(char[] T, char[] M){
     int n, m;
     long Bm, hT, hM;

     n = T.length;
     m = M.length;
     System.out.print("Occurrences en position :");
     hM = signature2(M, 0, m);
     // calcul de Bm = B^m mod p
     Bm = B;
     for(int i = 2; i <= m; i++)
         Bm = (Bm * B) % p;
     hT = signature2(T, 0, m);
     for(int i = 0; i < n-m; i++){
         if(i > 0)
             hT = signatureIncr(T, m, i-1, hT, Bm);
         if(hT == hM){
```

```
                if(occurrence(T, M, i))
                    System.out.print(" "+i+",");
                else
                    System.out.print(" ["+i+"],");
            }
        }
        System.out.println("");
    }
```

Donner la complexité de cet algorithme

Barème indicatif : Exercice 1 : 4 points, Exercice 2 : 6 points, Exercice 3 : 4 points, Exercice 4 : 6 points