Algorithmique II
Programmation Dynamique
(Correction)

Exercice 1 :

Distance d'édition Nous disposons de trois opérations élémentaires : SUBS, INS, DEL qui consistent respectivement à *substituer* une lettre par une autre, à *insérer* une lettre et à *supprimer* une lettre le tout dans un mot.
Par exemple, pour aller de ACGT à CGCT on peut substituer A par C pour avoir CCGT, puis supprimer le second C pour l'insérer à la 3-ième place après G. On sait alors que la distance est

$$d(\text{ACGT}, \text{CGCT}) \leq 3\,.$$

Dans ce qui suit, ε dénote le mot vide et $|m|$ est le nombre de caractères du mot m ($|\varepsilon| = 0$). On notera par $m = ua$ le mot composé par la concaténation des mots u et de a.

1. Calculer $d(\varepsilon, u)$ et $d(v, \varepsilon)$ en fonction de $|u|$ et $|v|$.

2. Calculer $d(ua, va)$ en fonction de $d(u, v)$.

3. Si a et b sont deux différentes lettres montrer que $d(ua, vb) = 1 + \min(d(u, v), d(ua, v), d(u, vb)$
▷ *soit c'est l'ancienne distance puis le SUBS*(a, b)
soit $d(ua, v)$ *puis suppression de* a
soit c'est $d(u, vb)$ *puis insertion de* b.

4. En déduire une fonction récursive simple mais exacte pour calculer la distance de deux chaînes de caractères données en entrée.
▷

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define min(a,b) (a<b?a:b)
#define min3(a,b,c) min(min(a,b),c)

char *ncpy(char *src, int n){
  int i;
  char *dest = (char*)malloc(sizeof(char)*n);
  for (i = 0; i < n && src[i] != '\0'; i++)
    dest[i] = src[i];
  for ( ; i < n; i++)
    dest[i] = '\0';
  return dest;
}

int dist(char *u, char *v) {
  char *s=NULL,*t=NULL;
  if (strlen(u)==0 || strlen(v)==0) return strlen(v)+strlen(u);
  s=ncpy(u,strlen(u)-1);
  t=ncpy(v,strlen(v)-1);
  if (u[strlen(u)-1] == v[strlen(v)-1])
    return dist(s,t);
  return 1 + min3( dist(s,t), dist(u,t), dist(s,v));
}

int main(int argc, char ** argv) {
  if (argc < 3) {
```

```
    (void) printf("Usage: %s string1 string2\n", argv[0]);
    return -1;
  }
  (void) printf("distance = %d\n",dist(argv[1], argv[2]));
  return 0;
}
```

5. Quelle est la complexité en temps de la fonction précédente ?

▷

$$t(n) = 1 + 3.t(n-1)$$

Donc $t(n) = O(3^n)$.

6. En utilisant une espace mémoire en $O(n^2)$, on peut accélérer l'algorithme précédent. En ne considérant que les "préfixes", que mettriez-vous en mémoire ?

▷ *L'idée est de ne pas refaire récursivement les mêmes calculs. On va donc stocker matriciellement la distance*

$$D[i,j] = dist(u[1]u[2]\cdots u[i], v[1]v[2]\cdots v[j])$$

Pour i *de* 1 *à* n *(idem pour* j*).*

7. En déduire un nouveau algorithme dynamique calculant une borne sur la distance d'édition entre deux chaînes. En quoi c'est une borne et pas la valeur exacte ?

▷

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define min(a,b) (a<b?a:b)
#define min3(a,b,c) min(min(a,b),c)

int distDYN(char *u, char *v) {
  int **d,i,j;
  int cout_subs=0;
  if (strlen(u)==0 || strlen(v)==0) return strlen(v)+strlen(u);
  d=(int**) malloc(sizeof(int*)*strlen(u));
  for (i=0;i<strlen(u);i++){
    d[i] = (int*)malloc(sizeof(int)*strlen(v));
    d[i][0] = i;
  }
  for (j=0;j<strlen(v);j++) d[0][j] = j;

  for (i=1;i<strlen(u);i++) {
    for (j=1;j<strlen(v);j++) {
      cout_subs= ((u[i] == v[j])? 0: 1);
      d[i][j] = (int)min3( (d[i-1][j]+1), (d[i][j-1]+1) , (d[i-1][j-1]+cout_subs) );
    }
  }
  return d[strlen(u)-1][strlen(v)-1];
}


int main(int argc, char ** argv) {
  char *u,*v;
  if (argc < 3) {
    (void) printf("Usage: %s string1 string2\n", argv[0]);
    return -1;
  }
  u = (char*)malloc(sizeof(char)*(strlen(argv[1])+1));
  v = (char*)malloc(sizeof(char)*(strlen(argv[2])+1));
  sprintf(u,"0%s",argv[1]);
  sprintf(v,"0%s",argv[2]);
```

```
  (void) printf("distance dynamique = %d\n",distDYN(u, v));
  return 0;
}
```

Exercice 2 :

Dans cet exercice, on s'intéresse au nombre de manières de partitionner un entier.

1. Montrer qu'il y a 6 manières d'écrire 5 comme somme d'entiers strictement plus petit que 5

▷

$$1+1+1+1+1,\ 1+1+3,\ 1+3+1,\ 3+1+1,\ 1+4,\ 4+1\,.$$

2. Soit D_n le nombre de manières d'écrire n comme somme de 1, 3 et 4. Trouver une formule de récurrence pour D_n

▷ $D_0 = D_1 = D_2 = 1$, $D_3 = 2$ *et* $D_n = 0$ *si* $n < 0$.
Puis $D_n = D_{n-1} + D_{n-3} + D_{n-4}$.

3. En déduire un algorithme en quelques lignes pour calculer D_n. Quelle est sa complexité ?

▷

```
int D(int n) {
  if (n < 3 ) return (n==0 || n==1 || n==2);
  else return D(n-1)+D(n-3)+D(n-4);
}
```

4. Montrer comment on peut calculer en temps logarithmique D_n.

▷ *On a une écriture matricielle de* D_n *en s'inspirant de Fibonacci.*

Exercice 3 :

Etant donné deux chaînes de caractères, x et y on veut trouver la plus longue sous-suites notée $\mathrm{LCS}(x, y)$ des deux et l'imprimer.
Ex : $x = \mathrm{ABCBDAB}$ et $y = \mathrm{BDCABC}$, $\mathrm{LCS}(x, y) = \mathrm{BCAB}$

1. Donner une solution dynamique à ce problème.

▷

– *LA récurrence : si* $x[i] = y[j]$ *alors* $d_{i,j} = d_{i-1,j-1} + 1$
sinon aucun de $x[i]$ *et de* $x[j]$ *ne va contribuer au LCS :*

$$d_{i,j} = \max d_{i-1,j}, d_{i,j-1}$$

sachant que de base on a $d_{i,0} = d_{0,j} = 0$.

```
/* n longueur de x, m longueur de y */
for(i=0;i<n;i++) d[i][0] = 0;
for(j=0;j<m;j++) d[0][j] = 0;
for (i=1; i<n;i++) {
    for (j=1; j<m;j++) {
       if (x[i] == y[j])
           d[i][j] = d[i-1][j-1] +1;
       else
           d[i][j] = max(d[i-1][j],d[i][j-1])
    }
}
```