

Rapport de TP : Alignement optimal et détection de plagiat

Annie LIM, Quentin GARRIDO

6 janvier 2020

Table des matières

1	Introduction	2
2	Exercice 1	2
3	Exercice 2	2
4	Exercice 3	2
5	Exercice 4	2
5.1	Théorie	2
5.2	Implémentation	3
6	Annexe : Code source	3

1 Introduction

Ce TP a pour but de concevoir un logiciel d'aide à la détection de plagiat. Ce logiciel d'alignement de séquences affichera simultanément le texte que l'on pense être du plagiat avec le texte original, en mettant en avant les correspondances. Moins les textes diffèrent et plus les chances de détecter un plagiat sont grandes.

2 Exercice 1

Pour calculer le score d'un alignement optimal entre x et y , nous pouvons utiliser l'algorithme de distance de Levenshtein, appelé distance d'édition (edit distance).

Nous voulons observer les différences entre deux textes. Cela revient à calculer leur score d'alignement, le coût des opérations nécessaires (deletion, insertion, substitution) pour obtenir le même texte. Plus ce score est faible et plus les textes sont similaires, et donc sujet d'être un plagiat.

Le score optimal correspond au minimum entre les trois valeurs données par les opérations deletion, insertion et substitution.

Algo...

Cet algorithme est bien de complexité $O(|x| \times |y|)$.

3 Exercice 2

Soit une matrice T telle que $T[i][j]$ est le score d'un alignement optimal entre x_i et y_j avec x_i ry y_j les préfixes de x et y de longueur i et j .

A partir de cette matrice, nous pourrions retrouver les opérations nécessaires à la solution optimale pour aligner les deux textes, afin de construire les textes 1 et 2 modifiés alignés.

Le backtracking consiste à suivre le chemin minimum de la matrice T de $T[i][j]$ jusqu'à $T[0][0]$.

Algo...

Cet algorithme est bien de complexité $O(|x| + |y|)$.

4 Exercice 3

5 Exercice 4

5.1 Théorie

Le principal changement ici est que nous voulons mettre en correspondance des lignes entre elles (séparées par des $\backslash n$).

Précédemment nous alignions un texte composé de caractères, mais maintenant nous voulons aligner un texte composé de lignes/phrases/paragraphes qui seront nos éléments de "base".

Le problème étant très similaire au précédent, la méthode que nous utilisions devrait pouvoir être adaptée à ce nouveau problème.

Pour ce faire nous allons définir une nouvelle distance de Levenshtein agissant sur des lignes entières et plus uniquement des caractères. Nous allons définir la substitution, insertion, et délétion comme suit :

$$\text{Ins}'(y) = \text{Lev}(\epsilon, y) = |y|$$

$$\text{Del}'(x) = \text{Lev}(x, \epsilon) = |x|$$

$$\text{Sub}'(x, y) = \text{Lev}(x, y)$$

Ici $Lev(x,y)$ est la distance de levenshtein définie précédemment, et x et y sont des lignes.

Il est assez facile de voir pourquoi nous avons choisi comme coût d'insertion et de délétion la longueur du paragraphe. En effet cela correspond à ajouter (resp. enlever) les caractères un par un, avec un coût de 1 à chaque fois.

Pour la substitution il est un peu moins clair au premier abord sur quelle valeur choisir. Le choix le plus simple est de supprimer puis d'insérer les paragraphes, cependant ce ne serait pas une distance car dans ce cas là $Sub(x, x) \neq 0$.

Nous avons étudié plusieurs distances entre les textes, chacune avec leur défauts et avantages, mais celle qui paraît la meilleure est la distance de Levenshtein, qui nous donnera une meilleure indication de la différence entre nos paragraphes, et nous permettra ensuite facilement de créer un alignement ayant du sens.

Puisque nous avons considéré un coût d'ajout et de suppression d'un caractère de 1 pour définir Sub' et Ins' nous devons faire pareil dans la distance de Levenshtein, et nous considérerons un coût de substitution de 1 si les caractères sont différents et 0 sinon.

Nous pouvons ensuite définir notre nouvelle distance de Levenshtein comme suit :

$$Lev'(x.a, y.b) = \min \begin{cases} Lev'(x.a, y) + Ins'(b) \\ Lev'(x, y.b) + Del'(a) \\ Lev'(x, y) + Sub'(a, b) \end{cases} = \min \begin{cases} Lev'(x.a, y) + |b| \\ Lev'(x, y.b) + |a| \\ Lev'(x, y) + Lev(a, b) \end{cases}$$

Ici a et b ne sont plus des caractères mais sont désormais des paragraphes.

Nous sommes donc en mesure d'adapter le code précédemment écrit pour cette nouvelle version, sans faire beaucoup de changements.

Nous pouvons nous demander si Lev' est toujours une distance.

Étant donné que Lev est une distance et que $Sub'(x) = Ins'(x)$ nous pouvons conclure que nous avons bien une distance.

5.2 Implémentation

6 Annexe : Code source

```

1 #include<stdio.h>
2 #include<stdlib.h>
3 #include <sys/stat.h>
4 #include <string.h>
5
6 struct alignement
7 {
8     char * x;
9     char * y;
10 };
11
12
13 /* ===== */
14 char * readtextfile(char * filename)
15 /* Retourne le contenu du fichier texte filename */
16 /* ===== */
17 {
18     struct stat monstat;
19     int N;
20     char * text = NULL;
21     FILE *fd = NULL;
22

```

```

23 N = stat(filename, &monstat);
24 if (N == -1)
25 {
26     fprintf(stderr, "error : bad file %s\n", filename);
27     exit(0);
28 }
29 N = monstat.st_size;
30 text = (char *)malloc(N+1);
31 if (text == NULL)
32 {
33     fprintf(stderr, "readtextfile() : malloc failed for text\n");
34     exit(0);
35 }
36 fd = fopen(filename, "r");
37 if (!fd)
38 {
39     fprintf(stderr, "readtextfile: can't open file %s\n", filename);
40     exit(0);
41 }
42 fread(text, sizeof(char), N, fd);
43 if ((N>0) && (text[N-1] == '\n')) text[N-1] = '\0';
44 else text[N-1] = '\0';
45 fclose(fd);
46 return text;
47 }
48
49 /* ===== */
50 int lmax(int a, int b)
51 /* Retourne le maximum de a et b */
52 /* ===== */
53 {
54     if (a < b) return b;
55     else return a;
56 }
57
58 /* ===== */
59 int lmin2(int a, int b)
60 /* Retourne le minimum de a et b */
61 /* ===== */
62 {
63     if (a < b) return a;
64     else return b;
65 }
66
67 /* ===== */
68 int lmin3(int a, int b, int c)
69 /* Retourne le minimum de a, b et c */
70 /* ===== */
71 {
72     return lmin2(lmin2(a,b),c);
73 }
74
75 /* ===== */
76 void retourne(char *c)
77 /* Retourner la chaine de caractere c */
78 /* ===== */
79 {
80     char tmp;
81     int m, j, i;
82     m = strlen(c);
83     j = m/2;
84     for(i = 0; i < j; i++){
85         tmp = c[i];
86         c[i] = c[m-i-1];
87         c[m-i-1] = tmp;
88     }

```

```

89 }
90 /* ===== */
91 void afficheSeparateurHorizontal(int nbcar)
92 /* ===== */
93 {
94     int i;
95     printf("|-");
96     for(i=0; i < nbcar; i++)
97         printf("-");
98     printf("-|");
99     for(i=0; i < nbcar; i++)
100         printf("-");
101     printf("-|\n");
102 }
103
104
105 /* ===== */
106 void affiche(char* textel, char* texte2, int nbcar)
107 /* Affiche simultanement textel et texte 2 en positionnant nbcar
108    caracteres sur chaque ligne. */
109 /* ===== */
110 {
111     int i, l1, l2, l;
112
113     char *t1,*t2;
114
115     char out[512];
116
117     l1 = strlen(textel);
118     l2 = strlen(texte2);
119
120     t1 = (char*) malloc(sizeof(char) * (nbcar + 1));
121     t2 = (char*) malloc(sizeof(char) * (nbcar + 1));
122
123     l = Imax(l1, l2);
124     afficheSeparateurHorizontal(nbcar);
125     for(i = 0; i < l; i+= nbcar){
126         if (i < l1) {
127             strncpy(t1, &(textel[i]), nbcar);
128             t1[nbcar] = '\0';
129         } else t1[0] = '\0';
130         if (i < l2) {
131             strncpy(t2, &(texte2[i]), nbcar);
132             t2[nbcar] = '\0';
133         } else t2[0] = '\0';
134
135         sprintf(out, " | %-5ds | %-5ds |\n", '%', nbcar, '%', nbcar);
136         printf(out, t1, t2);
137     }
138     afficheSeparateurHorizontal(nbcar);
139     free(t1);
140     free(t2);
141 }
142
143
144
145 /* ===== */
146 /* idem affiche , mais avec un formatage different*/
147 /* ===== */
148 void affiche2(char* textel, char* texte2, int nbcar)
149 {
150
151     int i, l1, l2, l;
152
153     char *t1,*t2;
154

```

```

155 char out[512];
156
157 l1 = strlen(texte1);
158 l2 = strlen(texte2);
159
160 t1 = (char*) malloc(sizeof(char) * (nbcars + 1));
161 t2 = (char*) malloc(sizeof(char) * (nbcars + 1));
162
163 l = lmax(l1, l2);
164
165 for(i = 0; i < l; i+= nbcars){
166     if (i < l1) {
167         strncpy(t1, &(texte1[i]), nbcars);
168         t1[nbcars] = '\0';
169     } else t1[0] = '\0';
170     if (i < l2) {
171         strncpy(t2, &(texte2[i]), nbcars);
172         t2[nbcars] = '\0';
173     } else t2[0] = '\0';
174
175     sprintf(out, "x: %c-%ds \ny: %c-%ds\n", '%', nbcars, '%', nbcars);
176     printf(out, t1, t2);
177 }
178 free(t1);
179 free(t2);
180 }
181
182
183
184 /*===== */
185 /* Exercice 3 */
186 /*===== */
187
188
189 int sub(char a, char b){
190     if(a == b)
191         return 0;
192     return 1;
193 }
194
195
196 int** compute_distance(char* texte1, char* texte2){
197     int n = strlen(texte1);
198     int m = strlen(texte2);
199
200     int** T= (int**) malloc((m+1)*sizeof(int*));
201     for(int i=0; i<=m; i++)
202         T[i]= (int*) malloc((n+1)*sizeof(int));
203
204     //T[m+1][n+1]
205     T[0][0] = 0;
206     for(int i=1; i<=n; i++)
207         T[0][i] = T[0][i-1] + 1; //Cout del
208     for(int j=1; j<=m; j++)
209         T[j][0] = T[j-1][0] + 1; //Cout ins
210     T[1][1] = lmin3(T[0][1]+1, T[1][0]+1, T[0][0]);
211     for(int i=1; i<=n; i++)
212         for(int j=1; j<=m; j++){
213             T[j][i] = lmin3(T[j-1][i]+1, \
214                 T[j][i-1]+1, \
215                 T[j-1][i-1]+sub(texte1[i-1], texte2[j-1]));
216         }
217
218     return T;
219 }
220

```

```

221 char** get_alignement(int** T, char* textel, char* texte2, int verbose){
222     char blank = ' ';
223     int n = strlen(textel);
224     int m = strlen(texte2);
225     int i = m;
226     int j = n;
227
228     int len = m+n;
229     char *textel_aligned = (char *)malloc((len+1)*sizeof(char));
230     char *texte2_aligned = (char *)malloc((len+1)*sizeof(char));
231
232     int k = 0;
233     while(i !=0 || j != 0){
234         //Si on a choisi Ins
235         if( i>0 && (T[i][j] == T[i-1][j] + 1)){
236             if(verbose) printf("Ins %c dans texte 1\n",texte2[i-1]);
237             textel_aligned[k] = blank;
238             texte2_aligned[k] = texte2[i-1];
239             i--;
240         }
241         //Si on a choisi Del
242         else if( j>0 && (T[i][j] == T[i][j-1] + 1)){
243             if(verbose) printf("Del %c dans texte 1\n",textel[j-1]);
244             textel_aligned[k] = textel[j-1];
245             texte2_aligned[k] = blank;
246             j--;
247         }
248         //Si on a choisi Sub
249         else if((T[i][j] == T[i-1][j-1] + sub(textel[j-1],texte2[i-1]))){
250             if(verbose) printf("Remplacer %c par %c dans le texte 1\n",textel[j-1],texte2[j
251             -1]);
252             textel_aligned[k] = textel[j-1];
253             texte2_aligned[k] = texte2[i-1];
254             j--;
255             i--;
256         }
257         else{
258             printf("Erreur dans le calcul de la table\n");
259             break;
260         }
261         k++;
262     }
263     textel_aligned[k] = '\0';
264     texte2_aligned[k] = '\0';
265
266     char **textes = (char**)malloc(2*sizeof(char*));
267     retourne(textel_aligned);
268     retourne(texte2_aligned);
269     textes[0] = textel_aligned;
270     textes[1] = texte2_aligned;
271     return textes;
272 }
273
274 void align_sentence(char* textel, char*texte2){
275
276     int** T = compute_distance(textel, texte2);
277     printf("Distance entre les textes: %d, longueur du textel: %ld, longueur du texte2: %
278     ld\n",\
279     T[strlen(texte2)][strlen(textel)], strlen(textel),strlen(texte2));
280
281     char** result = get_alignement(T, textel, texte2, 0);
282     affiche(result[0], result[1],40);
283
284     //Free results
285     free(result[0]);
286     free(result[1]);

```

```

285     free(result);
286
287     //Free T
288     for(int i=0;i<=strlen(texte2);++i)
289         free(T[i]);
290     free(T);
291 }
292
293
294 /*===== */
295 /*           Exercice 4           */
296 /*===== */
297
298 int count_occurences(char* texte, const char sep){
299     int count = 0;
300     for(int i = 0; i < strlen(texte);++i){
301         if(texte[i] == sep)
302             count++;
303     }
304     return count;
305 }
306
307 char** split(char* texte, int count, const char* sep){
308     if(count == 0)
309         return NULL;
310
311     char* texte_cop = strdup(texte);
312     char** liste = (char**) malloc((count+1)*sizeof(char*));
313     char* reste = NULL;
314     char* token;
315     int i = 0;
316     for(token = strtok_r(texte_cop, sep, &reste); token != NULL; token = strtok_r(NULL, sep, &reste))
317     {
318         liste[i] = strdup(token);
319         i++;
320     }
321     return liste;
322 }
323
324 int sub_strings(char* textel, char* texte2){
325     int** T = compute_distance(textel, texte2);
326     int val = T[strlen(texte2)][strlen(textel)];
327
328     for(int i = 0; i <= strlen(texte2);++i)
329         free(T[i]);
330     free(T);
331     return val;
332 }
333
334 int ins_strings(char* texte2){
335     return strlen(texte2);
336 }
337 int del_strings(char* textel){
338     return strlen(textel);
339 }
340
341 char* blank_string(char blank, int n){
342     char* string = (char*) malloc((n+1)*sizeof(char));
343     memset(string, blank, n);
344     string[n] = '\0';
345     return string;
346 }
347
348
349 int** compute_distance_strings(char** textel, int n1, char** texte2, int n2){

```



```

350     int n = n1+1;
351     int m = n2+1;
352
353     int** T= (int**) malloc((m+1)*sizeof(int*));
354     for(int i=0; i<=m; i++)
355         T[i]= (int*) malloc((n+1)*sizeof(int));
356
357     //T[m+1][n+1]
358     T[0][0] = 0;
359     for(int i=1; i<n+1;++i)
360         T[0][i] = T[0][i-1] + del_strings(textel[i-1]); //Cout del
361     for(int j=1; j<m+1;++j)
362         T[j][0] = T[j-1][0] + ins_strings(texte2[j-1]); //Cout ins
363     T[1][1] = Imin3(T[0][1]+1, T[1][0]+1, T[0][0]);
364     for(int i=1; i<n+1;++i)
365         for(int j=1; j<m+1;++j){
366             T[j][i] = Imin3(T[j-1][i]+ins_strings(texte2[j-1]),\
367                             T[j][i-1]+del_strings(textel[i-1]),\
368                             T[j-1][i-1]+sub_strings(textel[i-1],texte2[j-1]));
369         }
370
371
372     return T;
373 }
374
375 char*** get_alignement_texts(int** T, char** textel, int n1, char** texte2, int n2, int
    verbose, int* count){
376     char blank = ' ';
377     int n = n1+1;
378     int m = n2+1;
379     int i = m;
380     int j = n;
381
382     int len = m+n;
383     //int len = Imax(m,n);
384     char **textel_aligned = (char **)malloc((len)*sizeof(char*));
385     char **texte2_aligned = (char **)malloc((len)*sizeof(char*));
386
387     int k = 0;
388     while(i !=0 || j != 0){
389         //Si on a choisi Ins
390         if( i>0 && (T[i][j] == T[i-1][j] + ins_strings(texte2[i-1]))){
391             if(verbose) printf("Ins\n");
392             textel_aligned[k] = blank_string(blank, strlen(texte2[i-1]));
393             texte2_aligned[k] = strdup(texte2[i-1]);
394             i--;
395         }
396         //Si on a choisi Del
397         else if( j>0 && (T[i][j] == T[i][j-1] + del_strings(textel[j-1]))){
398             if(verbose) printf("Del\n");
399             textel_aligned[k] = strdup(textel[j-1]);
400             texte2_aligned[k] = blank_string(blank, strlen(textel[j-1]));
401             j--;
402         }
403         //Si on a choisi Sub
404         else if((T[i][j] == T[i-1][j-1] + sub_strings(textel[j-1],texte2[i-1]))){
405             if(verbose) printf("Sub\n");
406
407             int** T_temp = compute_distance(textel[j-1],texte2[i-1]);
408
409             char** alignes = get_alignement(T_temp, textel[j-1], texte2[i-1],0);
410
411             textel_aligned[k] = alignes[0];
412             texte2_aligned[k] = alignes[1];
413
414             //Free T

```

```

415     int size =strlen(texte2[i-1]);
416     for(int i=0; i<=size;++i){
417         free(T_temp[i]);
418     }
419     free(T_temp);
420     //Free alignes
421     free(aligned);
422     j--;
423     i--;
424 }
425 else{
426     printf("Erreur dans le calcul de la table\n");
427     break;
428 }
429 k++;
430 }
431
432 *count = k;
433
434
435 char ***textes = (char***)malloc(2*sizeof(char**));
436 textes[0] = texte1_aligned;
437 textes[1] = texte2_aligned;
438 return textes;
439 }
440
441 void align_texts(char* textel, char* texte2){
442     int n1 = count_occurences(textel, '\n');
443     char** listel = split(textel, n1, "\n");
444     int n2 = count_occurences(texte2, '\n');
445     char** liste2 = split(texte2, n2, "\n");
446
447     int** T = compute_distance_strings(listel, n1, liste2, n2);
448     printf("Distance entre les textes: %d, longueur du textel: %d, longueur du texte2: %d\n", \
449         T[n2+1][n1+1], strlen(textel), strlen(texte2));
450     int count;
451     char*** result = get_alignement_texts(T, listel, n1, liste2, n2, 0, &count);
452     for(int i = count-1; i>=0;--i){
453         affiche(result[0][i], result[1][i], 40);
454     }
455
456     //Free results
457     for(int i=0; i<count; ++i){
458         free(result[0][i]);
459         free(result[1][i]);
460     }
461     free(result[0]);
462     free(result[1]);
463     free(result);
464
465     //Free T
466     for(int i=0; i<=n2+1; ++i)
467         free(T[i]);
468     free(T);
469
470     //Free listel et liste2
471     for(int i=0; i<=n1; ++i){
472         free(listel[i]);
473     }
474     free(listel);
475
476     for(int i=0; i<=n2; ++i){
477         free(liste2[i]);
478     }
479     free(liste2);

```

```
480 }
481 }
482
483 /* ===== */
484 int main(int argc, char **argv)
485 /* ===== */
486 {
487     char *textel, *texte2;
488
489     printf("=====\\n");
490     printf("                Exercice 3                \\n");
491     printf("=====\\n");
492
493     textel = readtextfile("textel.txt");
494     texte2 = readtextfile("texte2.txt");
495
496     align_sentence(textel, texte2);
497
498     free(textel);
499     free(texte2);
500
501     printf("=====\\n");
502     printf("                Exercice 4                \\n");
503     printf("=====\\n");
504
505     textel = readtextfile("t1.txt");
506     texte2 = readtextfile("t2.txt");
507
508     align_texts(textel, texte2);
509
510     free(textel);
511     free(texte2);
512
513 }
```