IS 1

03 septembre-durée 1h

1. On veut utiliser l'algorithme de Liv Zempel Welsh pour compresser le mot "CHERCHER".

   (2) Compléter la table suivante qui correspond à l'encodage construit :

C	H	E	R	CH	HE	ER	RC	CHE
0	1	2	3	4	5	6	7	8

   Donner le message compressé obtenu :

   012346

2. Ecrire une fonction naive qui cherche les motif `m` dans le texte `t`. La fonction renverra l'indice de la première lettre de la première occurence de `m` dans `t` si elle existe et -1 sinon. On complètera le code suivant en C :

```
int rech_naif(char* m, char* t ){
   int lm = strlen(m);
   int lt = strlen(t);
   int i=0;
   while (i<= lt-lm){
      int j = lm-1;
      while (j>=0 && m[j]==t[i+j]){
         j--;
      }
      if (j== (-1)){
        return i;
      }
      i++;
  }
return (-1);
}
```

   (1)

3. On s'intéresse maintenant à l'algorithme de Boyer Moore pour chercher un motif `m` de longueur `lm`. On suppose disposer d'un tableau à deux dimensions de taille $lm * 256$ noté `table` tel que `table[i][c]` contient le plus grand indice k tel $0 \leq k < i$ et `m[k]=c`.

   Reprendre le code naif et le modifier (on pourra considérer table comme une variable globale) pour qu'il effectue les décalages obtenus par cet algorithme plutôt que de tester systématiquement tous les indices.

```
int bm(char* m, char* t){
 int lm = strlen(m); int lt = strlen(t); int i=0;
 while (i <= lt - lm){
          int j = lm-1;
          while (j>=0 && m[j]==t[i+j]){
                  j--;
          }
          if (j==-1) return i;
          i += j - table[j][t[i+j]];
 }
 return (-1);
}
```

   (2)

table :

	B	O	N
1	~~-2~~ 1	~~-1~~	0
2	~~-1~~	1	0
3	2	1	0
4	2	3	0

Dérouler l'algorithme de Boyer Moore pour la recherche de "NOBO" dans "BONOBO". On donnera notamment la table de décalage du motif "NOBO".

étape 1 : BONOBO
NOBO
× ✓

$i = 0$

(1)

étape 2 : $i = 2 - 0$

BONOBO
NOBO
✓ ✓ ✓ ✓

on renvoie $i = 2$

4. Dessiner (en explicitant la démarche) un arbre de Huffman permettant de compresser le mot "ILLIMITE". En déduire, un code obtenu. Quel est le nombre de bits utilisés pour compresser le mot avec ce code?

I → 3 occ ; L → 2 occ ; M → 1 ; T → 1 ; E → 1

init : $\mathcal{F} = \{(I,3) ; (L,2) ; (M,1) ; (T,1) ; (E,1)\}$

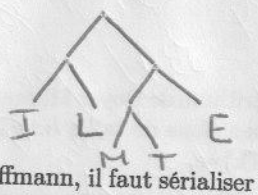
étape 1 : $\mathcal{F} = \{(I,3) ; (L,2) ; (\wedge_{M\ T}, 2) ; (E,1)\}$

(2)

étape 2 : $\mathcal{F} = \{(I,3) ; (L,2) ; (\wedge_{\wedge_{M\ T}\ E}, 3)\}$

étape 3 : $\mathcal{F} = \{(\wedge_{I\ L}, 5) ; (\wedge_{\wedge_{M\ T}\ E}, 3)\}$

étape 4 : arbre final obtenu :



taille du message compressé :
$3 \times 2 + 2 \times 2 + 3 + 3 + 2$
$= 18$

code obtenu :
I : 0 0
L : 0 1
M : 1 0 0
T : 1 0 1
E : 1 1

5. Pour compresser un texte avec l'algorithme de Huffmann, il faut sérialiser l'arbre dans le fichier de compression pour être en mesure de décompresser. On utilisera le type arbre suivant en Ocaml : `type arbre = F of string | N of arbre*arbre` (on remarque que les noeuds ne sont pas étiquetés et que loes feuilles sont étiquetées par des `string` qui de fait n'auront qu'une seule lettre mais utiliser ce type simplifie l'exercice). Ecrire une fonction Ocaml qui prend en entrée un tel arbre et un flux et qui sérialise l'arbre dans un fichier de la manière suivante : on inscrit le parcours préfixe de l'arbre avec un 0 dès lors que l'on a un noeud interne et un 1 suivant du charactère pour chaque feuille. Par exemple, l'arbre `N(N(F('a'),F('b')),F('c'))` aura pour sérialisation `001a1b1c`. On rappelle que pour écrire sur un flux `f`, on utilise la commande `output_string : out_channel->string->unit`.

(2)

```
let rec serialise a ch = match a with
  |F(c) -> output_string ch "1"; output_string ch c
  |N(g,d) -> output_string ch "0";
             serialise g ch;
             serialise d ch;;
```

6. Dans quelle famille d'algorithmes se trouve l'algorithme de Dijkstra? Sous quelle représentation est-il préférable de prendre le graphe en entrée pour une implémentation efficace de cet algorithme? Quelle est sa complexité si on utilise une file de priorité et une bonne représentation du graphe?

(2) Dijkstra → algo glouton - listes d'adjacences
Complexité pour m arêtes et n sommets :
$O((n+m)\log(n))$

7. Dans quelle famille d'algorithmes se trouve l'algorithme de Floyd Warshall? Sous quelle représentation est-il préférable de prendre le graphe en entrée pour une implémentation efficace de cet algorithme? Quelle est sa complexité si on utilise une bonne représentation du graphe?

(2) Programmat° dynamique.
Matrice d'adjacence.
$O(n^3)$ où n est le nb de sommets.

8. Donner la formule de récurrence qui constitue l'ingrédient principal de l'algorithme de Floyd Warshall.

(2) $u_k(i,j)$ représente le poids d'un plus court chemin entre i et j passant par les sommets d'étiquettes dans $\{0,\dots,k-1\}$.
$u_0(i,j) = w_{i,j}$ le poids de l'arête (i,j) avec $+\infty$ s'il n'y a pas d'arête
$\forall k \in [\![0,n]\!]$, $u_{k+1}(i,j) = \min\left(u_k(i,j) ; u_k(i,k+1) + u_k(k+1,j)\right)$

9. On considère ici des graphes non orientés connexes. Les sommets d'un graphe à n sommets ($n \in \mathbb{N}^*$) sont numérotés de 0 à $n-1$. On suppose qu'aucune arête ne boucle sur un même sommet.

Un chemin de longueur $p \in \mathbb{N}$ d'un sommet a vers un sommet b dans un graphe est la donnée de $p+1$ sommets $s_0, s_1, \dots, s_p$ tels que $s_0 = a$, $s_p = b$ et, pour tout $1 \leqslant k \leqslant p$, les sommets s_{k-1} et s_k sont reliés par une arête.

Un plus court chemin d'un sommet a vers un sommet b dans un graphe G est un chemin de longueur minimale parmi tous les chemins de a vers b. Sa longueur est notée $d_G(a,b)$.

Le diamètre d'un graphe G, noté $diam(G)$, vaut le maximum des longueurs des plus courts chemins entre deux sommets du graphe G. Autrement dit,

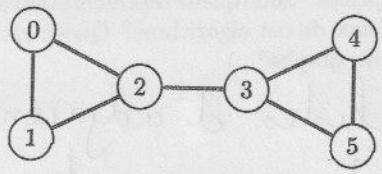
$$diam(G) = \max_{a,b \text{ sommets de } G} d_G(a,b).$$

Un chemin maximal d'un graphe G est un plus court chemin de G de longueur $diam(G)$.

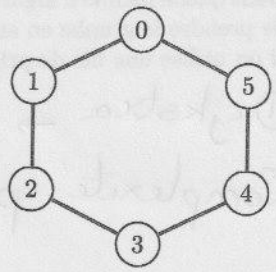
(a) Donner sans justification le diamètre et les chemins maximaux pour chacun des deux graphes G_1 et G_2 ci-dessous.

(2)

diamètre = 3



diamètre = 3



On suppose que les graphes sont représentés par listes d'adjacence.

(1) (b) **Donner l'entrée et la sortie de l'algorithme de Dijkstra. Comment cet algorithme permet-il de calculer le diamètre d'un graphe ?**

(1) (c) Quel parcours de graphe peut être utilisé pour le calcul du diamètre ?

(2) (d) Laquelle des deux méthodes précédentes est la mieux adaptée pour calculer le diamètre d'un graphe ?

(b) L'algo de Dijkstra prend en entrée un graphe G orienté pondéré à poids ≥ 0 et un sommet s; il renvoie le plus court chemin entre s et chaque s' de G. En appliquant Dijkstra à chaque sommet du graphe on pourra en déduire le diamètre comme le maximum de toutes les distances ainsi calculées.

(c) Le parcours en largeur depuis s calcule aussi ttes les distances à s et permet de manière analogue de calculer le diamètre

(d) Complexités si le graphe est sous forme de listes d'adjacence : $G=(S,A)$

Dijkstra pour chaque sommet : $O(|S| \times (|S|+|A|) \log|S|)$

Parcours en largeur pour chaq sommet : $O(|S| \times (|S|+|A|))$

Ainsi, le parcours en largeur est ici moins couteux puisque gérer une ~~liste est~~ file est ⊖ couteux que gérer une file de priorité.