Langages de programmation 1

Projet - Vigenere

Nieto Navarrete Matias, 502920

Mai 2022

1 Déscription

Ce projet a été réalisé dans le cadre du cours INFO-F105 (Langages de programmation). Il s'agit d'un chiffrement et d'un déchiffrement du chiffre de vigenère. La partie chiffrement a été réalisé en python alors que le dechiffrement en c++.

Notre programme est composé de plusieurs fichiers dont un makefile. Celui-ci compile les fichiers decrypt.cpp et attack.cpp sans erreur et avec les flags demandés.

2 Implémentation

2.1 Encrypt

Pour le fichier encrypt.py, nous avons implementé deux fonctions :

- `def chiffre_de_vigenere(message,clef)`

  Permet de chiffrer le message reçu à l'aide de la clef.

- `def main()`

  Permet de lancer notre programme.

2.2 Vigenere

Le fichier vigenere.hpp contient les déclarations des fonction qui se trouvent dans le fichier vigenere.cpp, la structure et les includes.

- `void dechiffrement_de_vigenere(const string &message, string clef,string &plain)`

  Déchiffre un message à l'aide d'une clef.

- `void lecture_fichier(const string &nom_fichier, string &texte)`

  Ecrit le texte du fichier dans un string.

- `void ecriture_fichier(const string &nom_fichier, const string &texte)`

  Ecrit dans un fichier le texte.

2.3 Decript

Le fichier Decript.hpp contient la déclarations des méthodes de la class Decript qui se trouvent dans le fichier Decript.cpp. Cette class permet de décriptage d'un texte à l'aide d'un mot de passe.

- `string majuscule(string minuscule)`

  Transforme toutes les lettres qui sont dans le string en majuscule.

- `Decript(const string&, const string&, std`:string):

  Le constructeur de la class. Dans celui-ci, nous appelons nos fonctions qui s'occupent des fichiers et celle du déchifrement de vigenere.

2.4 Attaque

Le fichier Attaque.hpp contient les déclarations des méthodes de la class Attaque qui se trouvent dans le fichier Attaque.cpp. Cette class permet de déchiffrer un fichier cripté sans connaitre le mot de passe.

- `string supprime_caractere_non_alphabetique(const string &message)`

  Supprime tout les caractères non alphabétique d'un string.

- `string *divise_colonne(string& cypher, size_t l)`

  Divise le texte en l colonne.

- `char lettre_plus_utiliser(const string &texte)`

  Trouve la lettre la plus utilisé dans un texte.

- `string trouve_cle_non_dechiffrer(string * colonne, size_t l,size_t taille_tableau`

  Trouve la clef possible de taille l non déchiffre de la colonnne.

- `string trouve_cle_dechiffrer(const string &clef)`

  Déchiffre la clef .

- `float frequence_lettre(string texte, char lettre)`

  Calcule la fréquence d'un caractère dans un string.

- `float calcul_erreur(float *erreur, size_t l)`

  Calcule l'erreur de la clef en faisant la moyenne des distances entre la lettre la plus fréquente et la fréquence attendue de la lettre e en français.

- `struct Clef *trouve_candidat(const string &cypher, size_t l)`

  Renvoie la meilleure clef pour déchiffrer le texte sous forme de structure.

- `void decode(const string &cypher, struct Clef *clef, string &plain)`

  Déchiffre un message chiffré encodé à l'aide du chiffre de Vigenère et d'une clef connue.

- `void attack(const string &cypher, string &plain,size_t l)`

  Déchiffre un message encodé à l'aide du chiffre de Vigenère sans avoir connaissance du mot de passe.

- `Attaque(const string&, const string&, size_t)`

  Le constructeur de la class. Dans celui-ci, nous appelons nos fonctions qui s'occupent des fichiers et la methode attack.

2.5 Attack et Decrypt

Le fichier attack.cpp permet de décripter un texte sans connaissance de la clef en appelant la class Attaque alors que decrypt.cpp déchiffre un texte en connaissant la clef grâce a la class Decript.

3 Modifications

3.1 Rajout de conditions et Modification de fonction

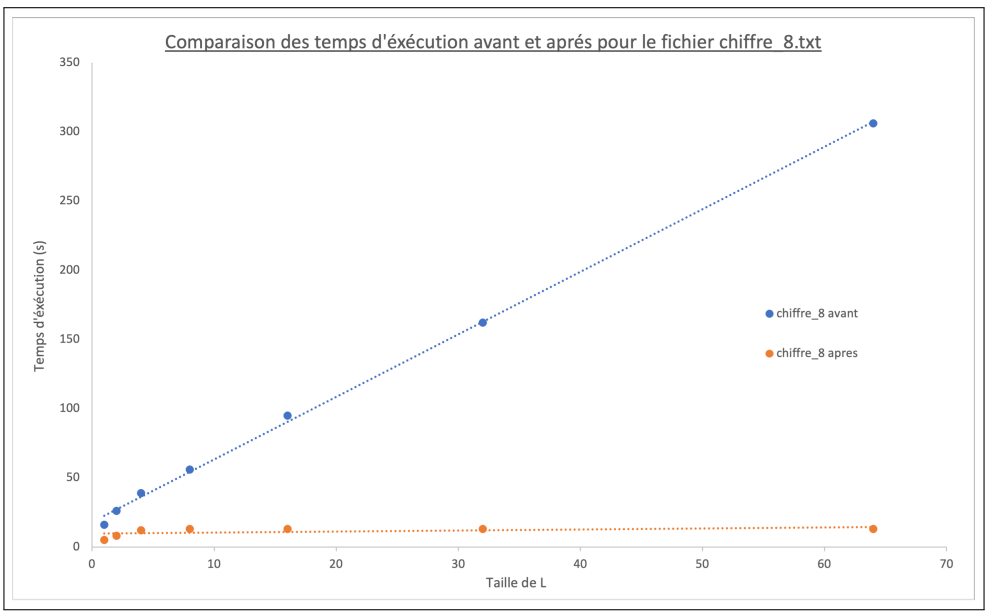
Nous avons rajouté une condition dans la constructeur de la class Attaque et une autre dans la methode trouve_candidat.
la condition dans le constructeur permet de vérifier que si la longueur L passé en argument ne soit pas plus grande que le nombre de caracteres possible à déchiffrer.
Tandis que la condition dans la methode, permet de stopper la boucle qui va de 1 jusqu'a L une fois que la bonne clef est trouvée. Nous savons que la bonne clef correspond à l'erreur la plus basse. Donc la taille de celle-ci multiplié par 2, nous sommes censé obtenir la même clef en double. Si c'est la cas nous nous arretons là.
Nous avons modifié la fonction divise_colonne de la partie 2. Celle-ci renvoyais un vecteur alors que maintenant, elle renvoi un tabelau dynamique de string.

3.2 Découpe du code

Nous avons découpée notre code en trois fichiers. le vigenere.hpp contient tout se qui est en commun avec le fichier Decript.hpp et Attaque.cpp. Dans le Decript.hpp, nous avons crée la class Decript qui est appelé dans le fichier decrypt.cpp. Et dans le fichier Attaque.hpp, nous avons créé la class Attque qui est appelé dans le ficher attack.cpp. Chaque class contient des attributs, un constructeur et un destructeur.

4 Résultats

Nous avons calculé nos temps d'éxécution pour le fichier attack.cpp avant les modifications et après les modifications (voir annexe).
Nous comparons nos résultats pour le fichier chiffre_8.txt pour des valeurs de L plus petit que 100, nous obtenons le graphique ci-dessous.



Et nous remarquons pour la droite bleu qu'à chaque fois que le L est doublé, nos temps d'exécution sont aussi doublés alors que pour la droite orange, pour n'importe quelle taille de L, nos temps d'exécution sont constants. Nous avons aussi fait compiler notre programme avec le flag O3. Nous remarquons que grâce a se flag, nos temps ont encore diminué.(Voir annexe)

5 Discussion

La partie de notre code qui nous prend le plus de temps est dans la fonction trouver_candidat. Plus exactement la divison du texte en colonne car nous devons répéter cette division pour chaque L. Ce qui cause l'augmentation de nos temps d'exécution.
Pour de ce qui est la memoire utilisée de notre code, celle-ci dépend aussi de la longueur de L car nous créons des tableaux dynamiques dont la taille dépend de L.

6 Annexe

Résultat avant modification			
fichier	L	moyenne	ecart type
chiffre_8.txt	1	26.493	
chiffre_8.txt	2	38.5838	
chiffre_8.txt	4	55.1952	
chiffre_8.txt	8	95.214	
chiffre_8.txt	16	162.186	
chiffre_8.txt	32	305.504	
chiffre_8.txt	64	585.213	
chiffre_8.txt	128	1123.42	
chiffre_8.txt	256	1930.69	
chiffre_8.txt	512		
chiffre_8.txt	1024		
chiffre_8.txt	2048		
chiffre_8.txt	4096		
chiffre_8.txt	8192		
chiffre_8.txt	16384		

Remarque : l'écart type et la moyenne n'ont pas été calculés car cela prenait énormement de temps. Nous avons decidé de représenter les valeurs du dernier fichier car entre l'avant et l'après, il y a une enorme différence.

Résultat après modification			
fichier	L	moyenne	ecart type
chiffre_8.txt	1	5.078507447242737	0.18029404082434808
chiffre_8.txt	2	7.971389055252075	0.6184107386444799
chiffre_8.txt	4	11.818895626068116	1.6478798950949043
chiffre_8.txt	8	13.072713708877563	0.3648857039121068
chiffre_8.txt	16	13.05393590927124	0.39238792479410917
chiffre_8.txt	32	13.14626877307892	0.3863312302210565
chiffre_8.txt	64	12.973607301712036	0.2054868580128098
chiffre_8.txt	128	13.113147091865539	0.41136552212285676
chiffre_8.txt	256	13.013279819488526	0.23033210045428207
chiffre_8.txt	512	13.29667363166809	0.8161301099674465
chiffre_8.txt	1024	16.269709968566893	2.5055398386462597
chiffre_8.txt	2048	13.565604639053344	1.1561895258213766
chiffre_8.txt	4096	15.065694093704224	1.6021447456281623
chiffre_8.txt	8192	13.012536334991456	0.09971966158531212
chiffre_8.txt	16384	14.705562686920166	2.107757627161699

Resultat après modification et avec le flag -O3			
fichier	L	moyenne	ecart type
chiffre_8.txt	1	3.951171803474426	0.7839145259847883
chiffre_8.txt	2	4.094890904426575	0.5374785189969707
chiffre_8.txt	4	4.8167668104171755	0.47090515591532756
chiffre_8.txt	8	4.762203788757324	0.25390095832705517
chiffre_8.txt	16	5.587743282318115	1.2528748728341415
chiffre_8.txt	32	5.504382920265198	0.46113832248105396
chiffre_8.txt	64	5.856350374221802	0.5161088900861778
chiffre_8.txt	128	5.486078047752381	0.6953142190949336
chiffre_8.txt	256	5.124530363082886	0.6684797807619679
chiffre_8.txt	512	5.241607737541199	0.42458907432315673
chiffre_8.txt	1024	5.520470333099365	0.5654136797808234
chiffre_8.txt	2048	5.3681766271591185	0.6741156045625626
chiffre_8.txt	4096	5.002711296081543	0.3870675182048024
chiffre_8.txt	8192	4.927964162826538	0.23225507638265047
chiffre_8.txt	16384	4.921916198730469	0.24793451690595558