Rapport de Projet

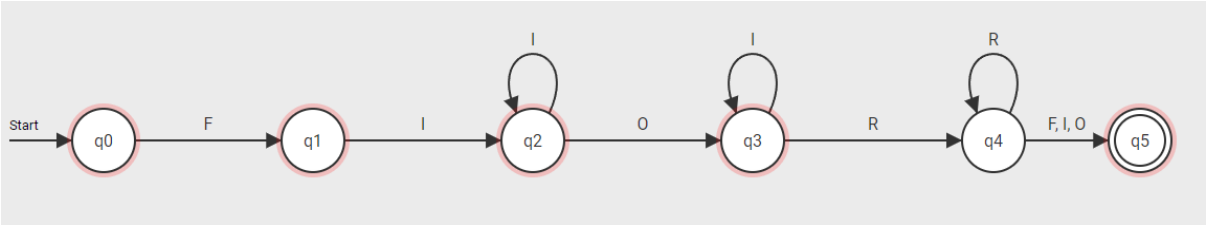
Étudiant: Romaric Chaffray 22002823

Date: 19/04/2023

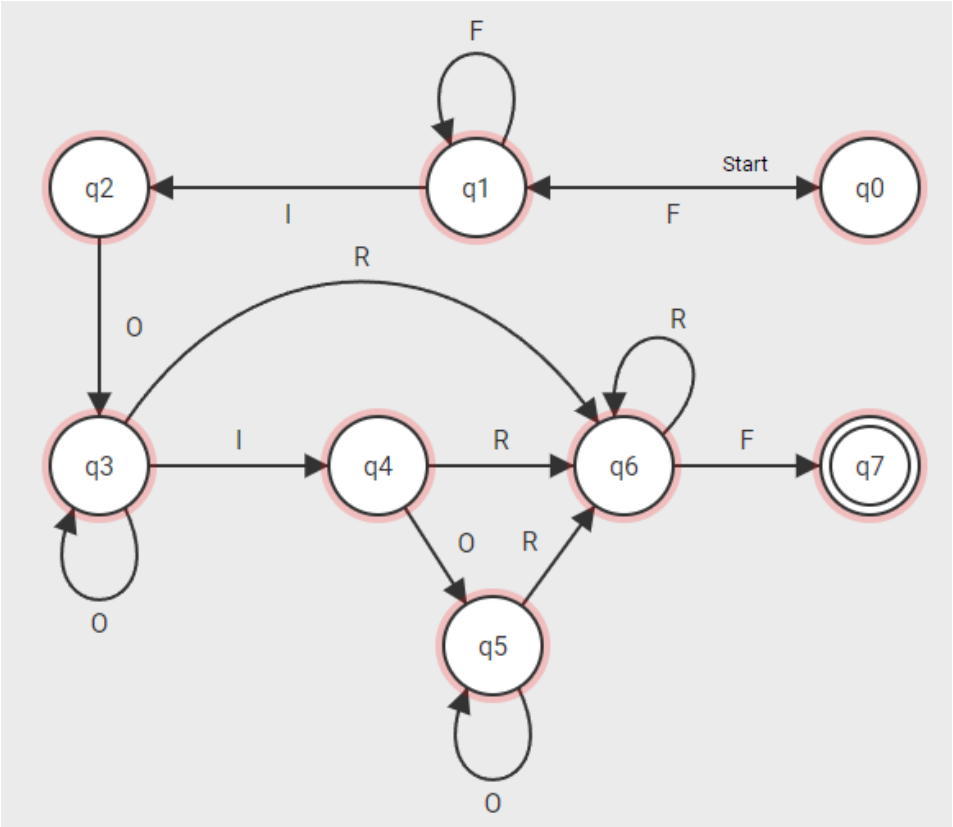
Introduction:
Le rapport décrit les différentes étapes du projet de recherche d'occurrences de motifs dans une chaîne de caractères, réalisé en C.

Étape 1: Schémas d'Automates et Tables de Transitions:

Motif : FI+OI*R+



Motif 2 : F+IO+I?O*R+



Pour chaque motif attribué, nous avons construit le schéma de l'automate correspondant ainsi que les tables de transitions associées. Ces schémas et tables sont essentiels pour comprendre le comportement de l'automate lors de la recherche d'occurrences dans la chaîne de caractères.

Motif : FI+OI*R+

0	1	-1	-1	-1
1	-1	2	-1	-1
2	-1	2	3	-1
3	-1	3	-1	4
4	7	7	7	4

Motif 2 : F+IO+I?O*R+

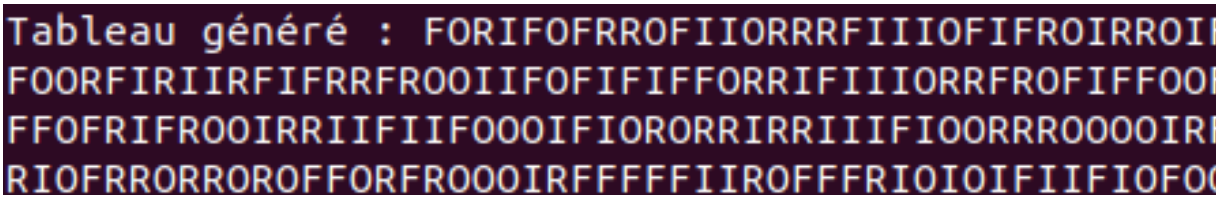
0	1	-1	-1	-1
1	1	2	-1	-1
2	-1	-1	3	-1
3	-1	4	3	6
4	-1	-1	5	6
5	-1	-1	5	6
6	7	7	7	6

Étape 2 et 3: Recherche d'Occurrences des Motifs:

Dans cette étape, nous avons rempli un tableau de 4000 caractères avec les lettres de l'alphabet personnel attribué ici “F, I, R, O” grâce à une fonction “genere(char* tableau)”.

```
void genere(char* tableau) {
    char alphabet[] = {'F', 'R', 'I', 'O'};
    int i;
    //pour avoir des valeurs aléatoires
    srand(time(NULL));
    //pour remplir le tableau avec des lettres aléatoires de l'alphabet
```

```
    for (i = 0; i < TAILLE_TABLEAU; i++) {
        tableau[i] = alphabet[rand() % TAILLE_ALPHABET];
    }}
```



Ensuite, nous avons utilisé la fonction `automate` pour rechercher toutes les occurrences du premier motif, puis du deuxième motif, en utilisant les automates construits précédemment. Cette fonction parcourt la chaîne de caractères en entrée, détectant les occurrences des motifs spécifiés. À chaque occurrence trouvée, elle collecte les informations sur la position dans le tableau (indice de début) et calcule le nombre de points rapportés en accord avec les règles spécifiées F = 10, R = 20, I = 3, O = 4 via la fonction "int calculer_points(char c)" inspirée du code des exercices.
Nous avons collecté les informations sur la position dans le tableau (indice de début) et le nombre de points rapportés par chaque occurrence, selon les règles spécifiées via une liste chaînée appelée "Occurrence".

```
struct Occurrence {
    int position; //la position de l'occurrence
    int points; //les points de l'occurrence
    char motif[1000]; //l'occurrence en question
    int numero_de_motif; //le numéro du motif correspondant à
l'occurrence
    struct Occurrence *next;
};
```

Étape 4: Validation du Programme de Recherche d'Occurrences:
Afin de vérifier la justesse du programme de recherche d'occurrences, nous avons utilisé le site regex101.com, qui identifie les occurrences dans un texte. Cela nous a permis de comparer les résultats obtenus par notre programme avec ceux fournis par le site. Ce programme de test analyse la recherche d'occurrences des deux motifs et fournit des informations permettant de confirmer la validité de notre implémentation.

Veuillez consulter l'image ci-dessous pour une illustration de ce processus pour les étapes 4 et 5.

```
Nombre total d'occurrences trouvées avant suppression des doublons : 66
Nombre total d'occurrences trouvées après suppression des doublons : 19
debug: Occurrences triées
Occurrences trouvées :
Position : 0, Points : 0, Motif : , Motif numéro : 0
Position : 428, Points : 37, Motif : FIOR, Motif numéro : 1
Position : 1817, Points : 40, Motif : FIIOR, Motif numéro : 1
Position : 40, Points : 40, Motif : FIOIR, Motif numéro : 1
Position : 878, Points : 41, Motif : FIOOR, Motif numéro : 2
Position : 510, Points : 43, Motif : FIIIOR, Motif numéro : 1
Position : 90, Points : 47, Motif : FFIOR, Motif numéro : 2
Position : 121, Points : 49, Motif : FIOOOOR, Motif numéro : 2
Position : 2634, Points : 54, Motif : FFIOIOR, Motif numéro : 2
Position : 276, Points : 57, Motif : FIORR, Motif numéro : 1
Position : 2376, Points : 60, Motif : FIIORR, Motif numéro : 1
Position : 2184, Points : 60, Motif : FIOIRR, Motif numéro : 1
Position : 227, Points : 63, Motif : FIIIORR, Motif numéro : 1
Position : 341, Points : 67, Motif : FFFFIOR, Motif numéro : 2
Position : 264, Points : 67, Motif : FFIORR, Motif numéro : 2
Position : 923, Points : 75, Motif : FFFFIOOOR, Motif numéro : 2
Position : 10, Points : 80, Motif : FIIORRR, Motif numéro : 1
Position : 441, Points : 81, Motif : FIOORRR, Motif numéro : 2
```

Étape 5, 6 et 7: Ensemble des Occurrences Différentes:

Dans un premier temps nous créerons une fonction "int max_hash_value(struct Occurrence *occurrences)" cette fonction permet de trouver le hash maximum parmi toutes les occurrences de motifs dans une liste donnée pour qu'on sache la taille de la table de hash qu'on va créer.

```c
int max_hash_value(struct Occurrence *occurrences) {
    int max_hash = -1;
    //pour trouver le hash maximum
    while (occurrences != NULL) {
        int hash = compute_hash(occurrences->motif);
        if (hash > max_hash) {
            max_hash = hash;
        }occurrences = occurrences->next;
    }
    return max_hash;}
```

En utilisant la technique du hashcode, nous avons constitué l'ensemble de toutes les occurrences différentes des deux motifs, sans doublons. Nous avons également calculé le nombre d'occurrences de chaque motif, ainsi que le nombre total d'occurrences différentes. Chaque occurrence a été insérée dans la table en fonction de sa valeur calculée par la fonction 'compute_hash', appelée par la fonction 'add_nom'.

```
void hashage(struct Occurrence *occurrences_premier, struct Occurrence
*occurrences){
   //déjà on trie les occurrences avant de les ajouter à la table de
hachage
   trier_occurrences(occurrences);
   printf("debug: Occurrences triées\n");
   struct Occurrence *current = occurrences_premier; //ça c'est un
pointeur de parcours initialisé au premier élémentrs le premier élément
   int hash = max_hash_value(current);
   init_tabhash(hash);
   current = occurrences_premier;
   afficher_occurrences(current);
   while (current != NULL) {
       add_nom(current->motif); // Ajouter l'occurrence à la table de
hachage
       current = current->next;
   }
   print_tabhash(hash+1);
```

Cette fonction permet aussi de trier les occurrences avant de les classer dans la table grâce à la fonction “trier_occurrences” ce qui nous donne :

```
383 : FIOOR,
386 : FIORR,
450 : FIIIOR, FIIOIR,
456 : FFIORR,
459 : FIIORR,
462 : FIOOOR,
465 : FIOORR,
532 : FIIOIRR,
538 : FFIORRR,
596 : FIIOIIIR,
Moyenne des points : 52.74
Ensemble de points différents trouvés :
57 63 40 60 69 37 49 43 67 45 72 100 87 41 47 61
```

Étape 8: Comparaison des algorithmes

Nous avons implémenté deux algorithmes différents pour calculer l'ensemble de tous les nombres de points différents sans doublons rapportés par les différentes occurrences. Voici un résumé des deux algorithmes et de leur performance en termes de comparaisons :

Algorithme 1 :

Méthode : L'algorithme utilise un tableau booléen pour enregistrer les points déjà rencontrés.

Avantages : Simple et effectue peu de comparaison.

Aperçu :

```
void algo1(struct Occurrence *occurrences) {
    bool *points_utilises = (bool *)calloc(100, sizeof(bool));
    if (points_utilises == NULL) {
        printf("Erreur d'allocation de mémoire\n");
        exit(0);
    }
    int nb_total_points = 0;
    int nb_comparaisons = 0; //on ajoute un compteur de comparaisons
    occurrences = occurrences->next;
    struct Occurrence *current = occurrences;
    while (current != NULL) {
        int points = 0;
        for (int i = 0; current->motif[i] != '\0'; i++) {
            points += calculer_points(current->motif[i]);
        }
        if (!points_utilises[points]) {
            points_utilises[points] = true;
            nb_total_points++;
        }
        current = current->next;
        nb_comparaisons++;//et on incrémente le compteur de comparaisons
    }
```

Algorithme 2 :

Méthode : L'algorithme utilise une liste chaînée pour stocker les points uniques rencontrés.

Inconvénients : Peut nécessiter plus de mémoire pour stocker les points uniques dans la liste chaînée et est plus complexe à implémenter.

Aperçu :

```
struct Node {
    int points;
    struct Node *next;
};
void algo2(struct Occurrence *occurrences) {
    struct Node *head = NULL;
    //on initialise : la liste chainée
    int nb_total_points = 0;
    //le compteur de nombres de points différents
    int nb_comparaisons = 0; //le compteur de comparaisons
    int somme_points = 0; //et la somme totale des points
    //ça parcours des occurrences
    occurrences = occurrences->next;
    struct Occurrence *current = occurrences;
    while (current != NULL) {
        int points = 0;
        //et calcul le nombre total de points pour chaque occurrence
        for (int i = 0; current->motif[i] != '\0'; i++) {
            points += calculer_points(current->motif[i]);
        }
        //puis on vérifie si le nombre de points est déjà dans la liste
        struct Node *temp = head;
        bool trouve = false;
        while (temp != NULL) {
            nb_comparaisons++;
            //on incrémente le compteur de comparaisons
            if (temp->points == points) {
                trouve = true;
                break;
            }
            temp = temp->next;
        }
        //si le nombre de points n'est pas dans la liste, l'ajouter
        if (!trouve) {
            struct Node *newNode = (struct Node *)malloc(sizeof(struct
Node));
            if (newNode == NULL) {
                fprintf(stderr, "Erreur d'allocation de mémoire\n");
```

```
            exit(EXIT_FAILURE);
        }
        newNode->points = points;
        newNode->next = head;
        head = newNode;
        nb_total_points++;
        somme_points += points;
    //ça joute le nombre de points à la somme totale
     }
     current = current->next; //et on passe à l'occurrence suivante
}
//fait la somme des points et calcule la moyenne
float moyenne_points = (float)somme_points / nb_total_points;
```

Comparaison des deux algorithmes:

L'algorithme 1 effectue moins de comparaisons que l'algorithme 2 (21 contre 156). Cette différence est peut-être due à la manière dont chaque algorithme gère la recherche de doublons. Dans l'algorithme 1, un tableau booléen est utilisé pour marquer les points déjà rencontrés, ce qui permet d'éviter les doublons avec une complexité en temps de O(n). En revanche, dans l'algorithme 2, une liste chaînée est utilisée pour stocker les points déjà rencontrés, ce qui nécessite une recherche linéaire à chaque insertion pour vérifier les doublons, ce qui entraîne une complexité en temps plus élevée de O(n^2) dans le pire des cas.

Étape 9: Calcul du nombre de points moyen :

Après avoir calculé l'ensemble de tous les nombres de points différents à l'étape 8, nous avons également calculé le nombre de points moyen à partir de cet ensemble. Voici le résumé :

L'algorithme 2, en plus de compter les points uniques et le nombre de comparaisons, calcule également la somme totale des points uniques rencontrés.

En utilisant cette somme totale et le nombre total de points différents trouvés, nous avons calculé le nombre de points moyen.

Le nombre de points moyen est affiché à la fin de l'exécution de l'algorithme 2. Le nombre de points moyen à partir de l'ensemble constitué à l'étape 8 est calculé en divisant la somme totale des points par le nombre total de nombres de points différents trouvés.
Résultat : Moyenne des points : 61,17

```
Algorithme 1 :
Nombre total de nombres de points différents trouvés : 18
Nombre de comparaisons effectuées : 21
Algorithme 2 :
Nombre total de nombres de points différents trouvés : 18
Nombre de comparaisons effectuées : 156
Nombre de points moyen : 61.17
```

Étape 10: Comparaison:

```
Nombre total d'occurrences trouvées après suppression des doublons : 12
debug: Occurrences triées
Occurrences trouvées :
Position : 0, Points : 0, Motif : , Motif numéro : 0
Position : 185, Points : 37, Motif : FIOR, Motif numéro : 1
Position : 2644, Points : 40, Motif : FIIOR, Motif numéro : 1
Position : 1620, Points : 40, Motif : FIOIR, Motif numéro : 1
Position : 42, Points : 41, Motif : FIOOR, Motif numéro : 2
Position : 190, Points : 43, Motif : FIIOIR, Motif numéro : 1
Position : 1126, Points : 57, Motif : FIORR, Motif numéro : 1
Position : 2650, Points : 60, Motif : FIIORR, Motif numéro : 1
Position : 3605, Points : 61, Motif : FIOORR, Motif numéro : 2
Position : 3959, Points : 68, Motif : FIOOIORR, Motif numéro : 2
Position : 1684, Points : 80, Motif : FIOIRRR, Motif numéro : 1
17 : FIOOIORR,
304 : FIOR,
377 : FIOIR, FIIOR,
383 : FIOOR,
386 : FIORR,
450 : FIIOIR,
459 : FIIORR,
465 : FIOORR,
538 : FFIORRR,
541 : FIOIRRR,
Moyenne des points : 51.17
Algorithme 1 :
Nombre total de nombres de points différents trouvés : 11
Nombre de comparaisons effectuées : 12
Algorithme 2 :
Nombre total de nombres de points différents trouvés : 11
Nombre de comparaisons effectuées : 56
Nombre de points moyen : 52.18
```

L'ensemble d'occurrences calculé à l'étape 5 comprend 12 occurrences après suppression des doublons, tandis que l'ensemble calculé à l'étape 8 comprend également 11 occurrences après suppression des doublons. Ainsi, le nombre d'éléments dans les deux ensembles n'est pas identique. Cela peut sembler surprenant, étant donné que les algorithmes utilisés pour générer ces ensembles font la même chose. Cependant, cela peut être expliqué par le fait que les données d'entrée sont des points à l'exécution du programme, ce qui signifie que les mêmes occurrences différentes avec le même score tel que : le motif "FIOIR" avec un score de 40 points et "FIIOR" qui a le même score mais qui sont détectées et que les mêmes doublons sont supprimés à chaque fois.

Étape 11: Comparaison moyenne:

On peut confirmer cette hypothèse en regardant les moyennes, par exemple :

```
Moyenne des points : 58.56
Algorithme 1 :
Nombre total de nombres de points différents trouvés : 16
Nombre de comparaisons effectuées : 18
Algorithme 2 :
Nombre total de nombres de points différents trouvés : 16
Nombre de comparaisons effectuées : 122
Nombre de points moyen : 58.50
```

Ici on a une légère différence de 0.06 points qui signifie que bien que les données soient les mêmes elles ne sont pas traitées de la même façon.

Étape 12 : Nombre de Points Minimal et Maximal par Motif

Pour chacun des deux motifs, le nombre de points minimal et maximal qu'une occurrence peut rapporter dépend des caractères présents dans le motif et de leur valeur en points.

Motif 1 (FI+OI*R+):

Nombre de Points Minimum : Le minimum de points serait obtenu si le motif était "FIOR". Ainsi, le nombre de points minimal serait 37.

Nombre de Points Maximum : Le maximum de points serait atteint si le motif était composé d'une infinité de 'R' avec 20 points chacun. Ainsi, le nombre de points maximal serait 10 (F) + 3 (I) + 4 (o) + 20 X 3 997(les 4000 lettres moins "f", "i" et "o") soit 79 957.

Motif 2 (F+IO+I?O*R+):
Nombre de Points Minimum : Le minimum de points serait obtenu si le motif était "FIOR". Ainsi, le nombre de points minimal serait 37.

Nombre de Points Maximum : Le maximum de points serait atteint si le motif était composé d'une infinité de 'R' avec 20 points chacun. Ainsi, le nombre de points maximal serait 10 (F) + 3 (I) + 4 (o) + 20 X 3 997(les 4000 lettres moins "f", "i" et "o") soit 79 957.
soit la même chose.

Étape 13 : Motif le Plus Rentable
Dans les deux motifs, le nombre maximal de points est le même, étant donné qu'ils dépendent principalement de la répétition infinie de 'R'. Ainsi, en termes de points, les deux motifs ont le même potentiel maximum.
Le motif qui rapporterait le plus de points serait celui qui inclut le plus grand nombre de lettres 'R', car chaque occurrence de 'R' contribue à 20 points. Par conséquent, entre les deux motifs, le motif avec la plus grande occurrence de 'R' serait le plus rentable par exemple "R+".

compilation :
Pour compiler le programme il faut taper "cmake CMakeLists.txt", puis "make" puis "./exo" si le programme indique :
"malloc(): unsorted double linked list corrupted
Abandon (core dumped)"
il faut retaper ./exo, ce bug peut arriver 1 ou 2 fois mais c'est rare.

Conclusion:
Ce projet nous a permis de mettre en pratique nos connaissances en langage C et en algorithmique. Nous avons réussi à concevoir et à implémenter un programme efficace pour la recherche d'occurrences de motifs dans une chaîne de caractères, ainsi qu'à analyser les résultats obtenus. Ce projet a été une expérience enrichissante pour mieux comprendre les concepts d'expressions régulières et d'automates.

Références:
Les codes et lib “hashcode.*” ainsi que “free-list.*” du cours, flaci.com et regex101.com.

Signature:
Chaffray Romaric 19/04/2023.