Automate

AHN Haram

19 avril 2024

Schémas et tables de transitions de l'étape 1

Les motifs que j'ai reçus, motif1 et motif2, sont les suivants : NA+UA?UR et NA+UAR. J'ai implémenté ces expressions régulières dans un automate.

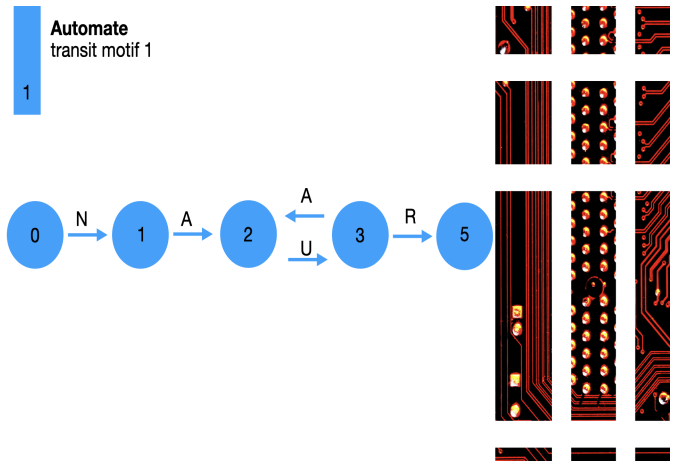

Figure 1: motif1

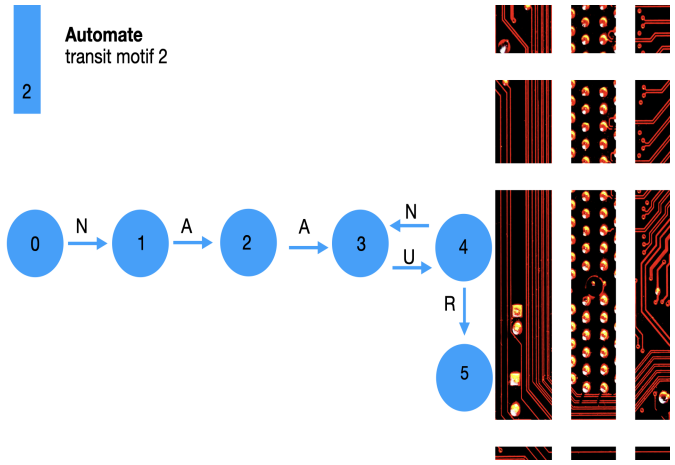

Figure 2: motif2

```
int ttransit[NBETATS][NBTYPES] = {
    {1, 0, 0, 0, 0}, {0, 2, 0, 0, 0}, {0, 2, 3,
    0, 0}, {0, 3, 0, 5, 0},
};
```

Listing 1: transit motif1

```
int ttransit[NBETATS][NBTYPES] = {
   {1, 0, 0, 0, 0}, {1, 2, 0, 0, 0}, {1, 2, 3, 0,
     0}, {1, 4, 4, 5, 0}, {1, 0, 4, 5, 0}
};
```

Listing 2: transit motif2

Traces d'exécution des étapes 2 à 9

t

```
 tape  2
---------------------------------------

Alphabet : NARU
Motif 1 : NA+UA*R
Motif 2 : NA+UA?U*R

Tableau de 4000 caract res :

NNRRUUNNURRARAUUUAUNANUANUNRRAUAUAAUAUNURAAURRNA
Affichage table des occurrences du motif 1
occ                                    [pos pts]
-----------------------------------------
NAAAUR          [46 39]
NAUAR         [183 38]
NAUR          [432 37]
NAAAUAR         [463 40]
NAUAR         [521 38]
NAUAR         [715 38]
NAUR          [789 37]
NAUR          [826 37]
NAUR          [1008 37]
NAUR          [1129 37]
NAUR          [1176 37]
NAAUR         [1465 38]
NAUR          [1932 37]
NAAUAR          [2099 39]
NAAUR         [2511 38]
NAAAAUR         [2555 40]
NAAUAR          [2580 39]
NAUR          [2619 37]
NAUR          [2895 37]
NAUR          [2976 37]
NAAUAR          [3097 39]
NAUAAR          [3391 39]
NAUAAR          [3615 39]
NAUR          [3778 37]
NAAUR         [3948 38]
25 occ du 1er motif trouvees
 tape  3
---------------------------------------
Tableau de 4000 caract res :
NNRRUUNNURRARAUUUAUNANUANUNRRAUAUAAUAUNU
Affichage table des occurrences du motif 2
occ                                    [pos pts]
-----------------------------------------
NAAAUR          [46 39]
NAUAR         [183 38]
NAAUAR          [216 39]
NAUR          [432 37]
NAUUR         [450 42]
NAAUUR          [455 43]
NAAAUAR         [463 40]
NAAAUR          [497 39]
NAUAR         [521 38]
NAUAR         [715 38]
NAUR          [789 37]
NAUR          [826 37]
NAUR          [1008 37]
NAUAR         [1115 38]
NAUR          [1129 37]
NAUR          [1176 37]
NAUR          [1354 37]
NAUR          [1367 37]
NAAUR         [1465 38]
NAUUR         [1552 42]
NAAUUR          [1630 43]
```

Listing 3: trace étap2

```
NAUR          [1711 37]
NAUR          [1932 37]
NAUR          [1937 37]
NAUAUR          [2061 43]
NAAUAR          [2099 39]
NAUAUR          [2346 43]
NAUR          [2386 37]
NAAUR         [2511 38]
NAAAAUR         [2555 40]
NAAUAR          [2580 39]
NAUR          [2619 37]
NAUR          [2895 37]
NAUR          [2976 37]
NAUAR         [3036 38]
NAAUAR          [3097 39]
NAUUR         [3381 42]
NAUAUR          [3488 43]
NAAUUR          [3630 43]
NAUR          [3677 37]
NAUR          [3778 37]
NAAUR         [3948 38]
42 occ du 2er motif trouvees

----------------------------------------
 tape  4 - Test:
Pattern 1

Occurrences trouv e par hashtable: 25
Occurrences trouv e par regex function: 37

Occurrences trouv e par hashtable: 25
Occurrences trouv e par regex function: 37

les r sultat trouv  par le hastable

-----------------------------------
NAAAUR    [46 39]
NAUAR   [183 38]
NAUR    [432 37]
NAAAUAR   [463 40]
NAUAR   [521 38]
NAUAR   [715 38]
NAUR    [789 37]
NAUR    [826 37]
NAUR    [1008 37]
NAUR    [1129 37]
NAUR    [1176 37]
NAAUR   [1465 38]
NAUR    [1932 37]
NAAUAR    [2099 39]
NAAUR   [2511 38]
NAAAAUR   [2555 40]
NAAUAR    [2580 39]
NAUR    [2619 37]
NAUR    [2895 37]
NAUR    [2976 37]
NAAUAR    [3097 39]
NAUAAR    [3391 39]
NAUAAR    [3615 39]
NAUR    [3778 37]
NAAUR   [3948 38]

-----------------------------------

 les r sultate apr s avoir v rifi  par la
    fonction regex function

-----------------------------------

 NAAAUR -> correct

 NAUAR -> correct

 NAUR ->non correct
```

Listing 4: trace étap4

```

 NAAAUAR ->non correct

 NAUAR ->non correct

 NAUAR ->non correct

 NAUR ->non correct

 NAUR ->non correct

 NAUR -> correct

 NAUR -> correct

 NAUR -> correct

 NAAUR ->non correct

 NAUR -> correct

 NAAUAR ->non correct

 NAAUR ->non correct

 NAAAAUR ->non correct

 NAAUAR ->non correct

 NAUR -> correct

 NAUR -> correct

 NAUR -> correct

 NAAUAR -> correct

 NAUAAR ->non correct

 NAUAAR ->non correct

 NAUR ->non correct

 NAAUR ->non correct

-------------------------------------

les r sultat de la regex function

NAAAUR    [46 39]
NAUAR   [183 38]
NAAUAR    [216 39]
NAUR    [432 37]
NAAAUAR   [463 40]
NAAAUR    [497 39]
NAUAR   [521 38]
NAUAR   [715 38]
NAUR    [789 37]
```

Listing 5: trace étap4

```
 tape  5
Ensemble des occurrences differentes (sans
    doublons) Occ Nb d'occurrences
---------------------------------
--> 11 occurrences differentes.
```

Listing 6: trace étap5

```

 tape  6

Occurences differentes triees par ordre croissant
     du nb d'occurences Occ Nb d'occurrences
--------------------------------
```

```
NAUAAR        2
NAAAAUR       2
NAAAUAR       2
NAUUR       3
NAUAUR        3
NAAUUR        3
NAAAUR        3
NAAUR       6
NAAUAR        7
NAUAR       8
NAUR        28
------------------------------
total occurence : 67
 tape  7

le nombre d'occurence : 11
le moyen : 6.090909

----------------------------
39       3
38       4
37       6
40       12
42       14
43       28


le total : 239.0
le nombre d'occurence: 6
le moyen: 39.8
```

Listing 8: trace étap7

```
 tape  8


-----------------------------------

Ensemble des nb de points differents :

NAAUAR NAAUR NAUAUR NAAAAUR NAUR NAAUUR NAUAR
    NAAAUR NAAAUAR NAUAAR NAUUR

--> soit 11 nb de points differents


-----------------------------------

Ensemble des nb de points differents :

39 38 37 40 42 43

--> soit 6 nb de points differents
```

Listing 9: trace étap8

Étape 10 réponse

```
 tape  8


-----------------------------------

Ensemble des nb de points differents :

NAAUR NAAAUAUR NAUR NAAUUR NAUAR NAAAUR NAAAUAR
    NAUAAR NAUUR

--> soit 9 nb de points differents


-----------------------------------

Ensemble des nb de points differents :
```

```
 tape  5
Ensemble des occurrences differentes (sans
    doublons) Occ Nb d'occurrences
---------------------------------
--> 9 occurrences differentes.
```

Listing 11: trace étap9

La fonction de hachage joue un rôle crucial dans la performance de la table de hachage en raison des caractéristiques suivantes
Consistance : la fonction de hachage doit toujours retourner la même valeur de hachage pour la même clé.
Efficacité : la fonction de hachage doit être rapide à calculer et ne nécessite pas de calculs complexes.
Distribution uniforme : la fonction de hachage doit répartir les valeurs de hachage aussi uniformément que possible, minimisant ainsi les collisions tout en utilisant de manière équitable tous les seaux de hachage possibles.
Influence de l'ordre de traitement des données
L'emplacement de stockage des données déterminé par la fonction de hachage peut influencer l'ordre dans lequel les données sont traitées. Par exemple, pour deux éléments de données A et B, la fonction de hachage peut traiter A avant B, ou vice versa. Cet ordre de traitement peut affecter le calcul des scores de la manière suivante :
Dépendance : s'il existe une dépendance entre les éléments de données (par exemple, si le score d'un élément est calculé sur la base d'un autre), l'ordre de traitement peut changer le score final.
Fonctions d'agrégation : lors de l'exécution d'opérations d'agrégation telles que la sommation ou la moyenne des valeurs de plusieurs éléments de données, l'ordre dans lequel les données sont traitées peut modifier les valeurs intermédiaires, influençant ainsi le score final.

Solutions possibles

Amélioration de la fonction de hachage : choisir ou développer une fonction de hachage qui distribue les données plus uniformément.
Indépendance de l'ordre de traitement : concevoir la logique de calcul des scores pour qu'elle soit aussi indépendante que possible de l'ordre de traitement des données.
Stratégies de résolution des collisions : appliquer des stratégies efficaces de gestion des collisions (telles que le chaînage ou l'adressage ouvert) pour standardiser l'ordre de traitement des données.
Le choix de la fonction de hachage dans l'implémentation d'une table de hachage est un élément crucial pour la gestion et le traitement efficaces des données. Une fonction de hachage bien conçue peut améliorer l'efficacité du traitement des données et contribuer à l'amélioration des performances de l'application.

réponse 11

```
 tape   7

le nombre d'occurence : 9
le moyen : 7.222222

----------------------------
37       1
38       1
44       2
43       3
42       3
48       20
39       35


le total : 291.0
le nombre d'occurence: 7
le moyen: 41.6

 tape   8


-----------------------------------

Ensemble des nb de points differents :

NAAUR NAUAUR NAUAUUR NAUR NAAUUR NAUAR NAAAUR
    NAAUAUR NAUUR

--> soit 9 nb de points differents


-----------------------------------

Ensemble des nb de points differents :

37 38 44 43 42 48 39

--> soit 7 nb de points differents
 tape   7

le nombre d'occurence : 12
le moyen : 4.916667

----------------------------
37       1
38       2
39       2
40       3
48       3
44       4
42       13
43       31


le total : 331.0
le nombre d'occurence: 8
le moyen: 41.4

 tape   8
```

Listing 12: trace étap7 et étap8

```
-----------------------------------

Ensemble des nb de points differents :

NAAUAR NAAAUUR NAAUR NAUAUR NAUAUUR NAAAAUR NAUR
    NAUAR NAAAUR NAAUUUR NAAUAUR NAUUR

--> soit 12 nb de points differents
-----------------------------------

Ensemble des nb de points differents :

37 38 39 40 48 44 42 43

--> soit 8 nb de points differents
```

Listing 13: trace étap7 et étap8

```
 tape   7

le nombre d'occurence : 9
le moyen : 7.222222

----------------------------
37       1
38       1
44       2
43       3
42       3
48       20
39       35


le total : 291.0
le nombre d'occurence: 7
le moyen: 41.6

 tape   8


-----------------------------------

Ensemble des nb de points differents :

NAAUR NAUAUR NAUAUUR NAUR NAAUUR NAUAR NAAAUR
    NAAUAUR NAUUR

--> soit 9 nb de points differents


-----------------------------------

Ensemble des nb de points differents :

37 38 44 43 42 48 39

--> soit 7 nb de points differents
```

Listing 14: trace étap7 et étap8

Analyse Comparative des Étapes 7 et 9(réponse 11)

Différences de Structure de Données et de Méthodes de Calcul Étape 7 : Le calcul de la moyenne des points basée sur la fréquence de chaque occurrence. Ce processus prend en compte la fréquence de chaque score, où les scores qui apparaissent plus fréquemment ont un impact plus significatif sur la moyenne.

Étape 8 : Identification simplement du nombre de scores uniques, sans calculer la moyenne basée sur le type de chaque score. Chaque score est considéré une seule fois.
Calcul de la Moyenne des Scores
Moyenne à l'Étape 7 : Le calcul inclut toutes les occurrences, donc un score qui se produit fréquemment influencera davantage la moyenne.
Moyenne à l'Étape 8 : Seuls les types de scores sont considérés, et la fréquence des scores n'est pas prise en compte. Par conséquent, tous les scores contribuent de manière égale au calcul de la moyenne.
Causes des Différences de Résultats
Différences dans le Traitement des Données : À l'étape 7, la fréquence de chaque score a un grand impact sur le calcul de la moyenne, donc certains scores fréquents peuvent augmenter ou diminuer la moyenne de manière significative. Impact des Scores Uniques et des Fréquences : À l'étape 8, seul le type de score est considéré, ce qui signifie que la fréquence est ignorée. Cela conduit à une contribution égale de chaque score à la moyenne, sans égard à la fréquence de leur occurrence. Conclusion et Recommandations
Lors de la comparaison des résultats des deux étapes, il est important de comprendre comment la fréquence des scores affecte la moyenne. À l'Étape 7, un score qui se produit fréquemment peut modifier considérablement la moyenne, tandis qu'à l'Étape 8, tous les scores contribuent de manière égale, résultant en des résultats plus stables. Comprendre ces différences et interpréter les résultats de chaque étape nécessite une considération claire de la structure des données et des méthodes de traitement.

Étape 12

NA suivi d'un ou plusieurs U, puis zéro ou un A, et enfin zéro ou plusieurs R. 2. Définition des règles de calcul des points Utilisez la fonction countnb pour attribuer des points à chaque caractère. Cette fonction calcule les points comme suit :
'A' : 1 point 'U' : 5 points 'N' : 13 points 'R' : 18 points
3. Calcul des points minimum et maximum
Pour chaque motif, calculez les scores minimum et maximum en appliquant la fonction countnb à toutes les variations possibles du motif.
Exemple : 'NA+UA?U*R' Score minimum : La forme la plus simple pourrait être 'NAUU' (A une fois, U deux fois, N une fois). Calcul des points : 13(N) + 1(A) + 5(U) + 5(U) = 24 points Score maximum : Maximisez le nombre de 'N' et 'U', et ajoutez 'R'. Par exemple, 'NAUUUUUUUUUR' (A une fois, U neuf fois, N une fois, R une fois). Calcul des points : 13(N) + 1(A) + 5×9(U) + 18(R) = 13 + 1 + 45 + 18 = 77 points

4. Synthèse et interprétation des résultats Documentez les scores minimum et maximum calculés pour chaque motif et analysez comment le motif affecte le score. Cela permet de comprendre la plage de scores possibles pour chaque motif et d'établir des stratégies pour maximiser les points sous certaines conditions.

Étape 13

1. Compréhension des points par caractère Basez la conception de votre motif sur les points attribués à chaque caractère, selon la fonction countnb. Par exemple :
'A' : 1 point 'U' : 5 points 'N' : 13 points 'R' : 18 points
2. Conception du motif Pour maximiser les points, vous pouvez adopter les stratégies suivantes :
Inclure autant que possible les 'R' et 'N' : Ces deux caractères offrant le plus de points, leur présence en grande quantité est avantageuse. Considérer la possibilité de répétition du motif : Concevez le motif de manière à ce qu'il puisse se répéter, permettant ainsi de gagner des points plusieurs fois à partir d'une seule occurrence du motif. Allonger le motif autant que possible : Créez un motif long pour maximiser les points obtenus lors d'une seule correspondance.

Explications sur les programmes des étapes 2 à 9

```c
typedef struct {
    int count;
    int* pos;
    char** matches;
} MatchResult;
```

Listing 15: description

Pour augmenter la stabilité du code source, les fichiers de code source ont été divisés. Pour faciliter la compilation, un Makefile a été utilisé.
Pour éviter la répétition de la même fonction et pour retourner diverses données en une seule fois, la structure MatchResult est utilisée. Cette structure est employée pour stocker les résultats de la correspondance de motifs. count représente le nombre de motifs trouvés, pos stocke les positions de début de chaque motif, et matches contient les chaînes de caractères effectivement correspondantes.

```c

MatchResult affiresetape2() {
    MatchResult res;
    int i;
    char str[MAXstr + 1];
    printf(" tape  2\n");
    printf("
    ---------------------------------------\n");
    printf("\nAlphabet : NRAU\nMotif 1 : NA+UA*R\
    nMotif 2 : NA+UA?U*R\n");
    strcpy(str, generate_random_string());
```

Listing 16: description

```
 printf("Tableau de 4000 caract res :\n");
    printf("%s\n", str);


    res = matchPattern1(str);
    printf("Affichage table des occurrences du
    motif 1\n");
    printf("occ                                  [
    pos pts]\n");
    printf("
    ----------------------------------------\n"
    );
    int points[res.count];
    for(i = 0; i < res.count; i++){
    points[i] = countnb(res.matches[i]);
  }

    for (i = 0; i < res.count; i++) {
        printf("%s\t\t\t\t[%d %d]\n", res.matches
    [i], res.pos[i], points[i]);
        insertOrUpdate(res.matches[i]);
        recordScore(points[i]);

    }
    printf("%d occ du 1er motif trouvees\n", res.
    count);
    freeMatchResult(res);


    return res;
}
char* generate_random_string() {
    static char str[MAXstr + 1];
    char chars[] = "NAUR";
    int i;
    srand(time(NULL));

    for (i = 0; i < MAXstr; i++) {
        str[i] = chars[rand() % (sizeof(chars) -
    1)];
    }


    return str;
}
int countnb(char* str) {
    int i, count = 0;
    for (i = 0; str[i] != '\0'; i++) {
        switch (str[i]) {
            case 'A': count += 1; break;
            case 'U': count += 5; break;
            case 'N': count += 13; break;
            case 'R': count += 18; break;
        }
    }
    return count;
}
void insertOrUpdate(const char *key) {
    unsigned int index = hash(key);
    HashNode *node = hashTable[index];
    while (node != NULL && strcmp(node->key, key)
     != 0) {
        node = node->next;
    }
    if (node == NULL) {
        node = (HashNode*) malloc(sizeof(HashNode
    ));
        if (node == NULL) {
            fprintf(stderr, "Failed to allocate
    memory for HashNode\n");
            return;
        }
```

Listing 17: description

```
        strcpy(node->key, key);
        node->count = 1;
        node->next = hashTable[index];
        hashTable[index] = node;
        hashTablecompara++;
    } else {
        node->count++;
    }
}
#include"header.h"

enum types get_type(char l) {
  switch (l) {
    case 'N':
      return N;
    case 'A':
      return A;
    case 'U':
      return U;
    case 'R':
      return R;
    default:
      return AUTRE;
  }
}


MatchResults find_matches(const char *text, const
     char *pattern) {
    regex_t regex;
    regmatch_t pmatch;
    MatchResults results;
    int status, offset = 0;

    results.matches = NULL;
    results.count = 0;

    if (regcomp(&regex, pattern, REG_EXTENDED) !=
     0) {
        return results;
    }

    while ((status = regexec(&regex, text +
    offset, 1, &pmatch, 0)) == 0) {
        if (results.matches == NULL) {
            results.matches = malloc(sizeof(
    MatchInfo));

        }
        else {
            results.matches = realloc(results.
    matches, sizeof(MatchInfo) * (results.count +
     1));
        }

        int match_start = offset + pmatch.rm_so;
        int match_end = offset + pmatch.rm_eo;

        results.matches[results.count].start =
    match_start;
        results.matches[results.count].end =
    match_end;
        results.matches[results.count].
    matched_str = strndup(text + match_start,
    match_end - match_start);

        results.count++;
        offset += pmatch.rm_eo; //

    }

    regfree(&regex);
    return results;
}
```

Listing 18: description

```c
void free_match_results(MatchResults results) {
    for (int i = 0; i < results.count; i++) {
        free(results.matches[i].matched_str);
    }
    free(results.matches);
}
void freeMatchResult(MatchResult result) {
  int i;
  for (i = 0; i < result.count; i++) {

    free(result.matches[i]);
  }
  free(result.pos);
  free(result.matches);
}
```

Listing 19: description

2. Fonction affiresetape2 Cette fonction est exécutée à une étape spécifique (étape 2) et effectue les opérations suivantes :
Initialisation des variables et création de la chaîne d'entrée : Une chaîne aléatoire est générée et stockée dans le tableau str. Correspondance de motifs : La fonction matchPattern1 est appelée pour tenter de trouver le premier motif défini dans la chaîne str, et les résultats sont stockés dans res. Calcul et affichage des scores : Les scores pour chaque motif correspondant sont calculés en utilisant la fonction countnb, et les résultats sont affichés. La chaîne du motif, sa position et le score calculé sont affichés. Enregistrement dans la base de données et des scores : Les fonctions insertOrUpdate et recordScore sont utilisées pour enregistrer les résultats des correspondances et les scores dans la base de données. Libération de la mémoire : La fonction freeMatchResult est appelée pour libérer la mémoire allouée à la structure MatchResult. 3. Fonctions utilisées

```
generate_random_string() : Génère une chaîne
aléatoire et la retourne.
matchPattern1(char str)* : Recherche le premier
motif dans la chaîne str et
retourne les résultats sous forme
de la structure MatchResult.
countnb(char str)* : Calcule les scores pour
la chaîne str.
insertOrUpdate(char match)* :
Insère ou met à jour le motif correspondant
dans la base de données.
recordScore(int score) : Enregistre le
score calculé.
freeMatchResult(MatchResult res) : Libère
la mémoire allouée à
la structure MatchResult.
Ce code illustre le processus complet
de recherche de motifs spécifiques
dans une chaîne et
de traitement des résultats,
faisant partie d'un programme
qui effectue diverses opérations
liées à la correspondance de motifs.
```

```c
void affiresetape4(const char* pattern,
    MatchResult (*matchFunction)(const char*),
    const char* input, MatchResults res) {
    int i, score, score2,j;
    printf("\n
    ----------------------------------------\n");
    printf(" tape  4 - Test: %s\n", pattern);

    MatchResult result = matchFunction(input);


    printf("Occurrences trouv e par hashtable: %
    d\n", result.count);
    printf("Occurrences trouv e par regex
    function: %d\n", res.count);

    for (i = 0; i < result.count; i++) {
        score = countnb(result.matches[i]);
        printf("%s\t\t[%d %d]\n", result.matches[
    i], result.pos[i], score);
    }
        int minCount = (result.count < res.count)
     ? result.count : res.count;

    if(result.count != res.count){
        printf("\n ->non correct\n");

    }
    else{
        printf("\n -> correct\n");
    }
    printf("\n
    ------------------------------------ \n");

    for (i = 0; i < minCount; i++) {
        score2 = countnb(res.matches[i].
    matched_str);
        printf("%s\t\t[%d %d]\n", res.matches[i].
    matched_str, res.matches[i].start, score2);
    }

    freeMatchResult(result);
    free_match_results(res);
}
typedef struct {
    int start;
    int end;
    char *matched_str;
} MatchInfo;

typedef struct {
    MatchInfo *matches;
    int count;
} MatchResults;
```

Listing 20: description

Le code C fourni définit la fonction affiresetape4, qui compare et affiche les résultats de deux différentes fonctions de correspondance de motifs. De plus, ce code utilise deux structures, MatchResult et MatchResults, pour stocker les résultats des correspondances de motifs. Voici une explication détaillée de chaque partie et de leur fonction. Fonction affiresetape4
Cette fonction prend les paramètres suivants :
pattern : Le motif textuel à examiner.
matchFunction : Un pointeur de fonction qui exécute la

input : La chaîne de caractères sur laquelle effectuer la correspondance de motifs. res : Une structure MatchResults contenant les résultats d'une seconde fonction de correspondance de motifs. Opérations principales de la fonction : Exécution de la correspondance de motifs : Appelle matchFunction pour effectuer la correspondance de motifs sur la chaîne input et stocke les résultats dans result. Comparaison et affichage des résultats : Compare et affiche le nombre de motifs trouvés par les deux fonctions (result.count et res.count). Affiche des détails sur chaque résultat de correspondance, incluant la chaîne correspondante, la position et le score calculé. Vérifie et affiche si les résultats des deux fonctions sont identiques ou non. Libération de mémoire :

`Appelle freeMatchResult et free_match_results pour libérer la mémoire allouée.`

```c
void affichageresexo6(){
    int nodeCount;
    double n;
    HashNode **nodes;
    printf("\n  tape  6\n");
    printf("\nOccurences differentes triees par ordre croissant du nb d'occurences Occ Nb d' occurrences\n");
    printf("---------------------------------\n");
    nodeCount = collectNodes(&nodes);
    quickSort(nodes, 0, nodeCount - 1);
    printSortedHashResults(nodes, nodeCount);
    free(nodes);
}

void affichageresexo7(){
    int nodeCount;
    double moyen;
    HashNode **nodes;
    nodeCount = collectNodes(&nodes);
    moyen = moyenoccurence(nodes);
    printf("\n tape 7\n");
    printf("\nle nombre d'occurence : %d\nle moyen : %f\n ",nodeCount,moyen);
    free(nodes);
}
```

Listing 21: description

Fonction affichageresexo6 Cette fonction est conçue pour afficher les résultats triés des occurrences uniques collectées, ordonnées par nombre croissant d'occurrences. Voici les étapes principales :
Initialisation et collecte des données : nodeCount : Initialise et récupère le nombre total de nœuds (occurrences uniques) à partir d'une structure de données de type table de hachage. nodes : Un tableau de pointeurs sur HashNode, qui sont collectés via la fonction collectNodes. Tri des données : Les données sont triées en utilisant l'algorithme de tri rapide (quickSort), ce qui permet de les ordonner selon le nombre d'occurrences de chaque nœud. Affichage des résultats : Après le tri, les résultats sont affichés en utilisant la fonction printSortedHashResults, qui parcourt le tableau de nœuds et imprime chaque occurrence avec son nombre d'occurrences. Libération de la mémoire : La mémoire allouée pour le tableau nodes est libérée à la fin de la fonction pour éviter les fuites de mémoire. Fonction affichageresexo7 Cette fonction calcule et affiche la moyenne du nombre d'occurrences parmi les nœuds uniques collectés. Les étapes sont les suivantes :
Collecte des nœuds : Similaire à affichageresexo6, cette fonction commence par collecter les nœuds à l'aide de collectNodes et stocke le nombre de nœuds dans nodeCount. Calcul de la moyenne : La moyenne des occurrences (moyen) est calculée par la fonction moyenoccurence, qui prend le tableau de nœuds et retourne la moyenne calculée sur la base du nombre d'occurrences de chaque nœud. Affichage des résultats : Les résultats, y compris le nombre total d'occurrences et la moyenne calculée, sont affichés à l'écran. Libération de la mémoire : Comme dans la fonction précédente, la mémoire allouée pour le tableau de nœuds est libérée pour éviter les fuites de mémoire.

Étape 4 fonction

```c
MatchResults find_matches(const char *text, const char *pattern) {
    regex_t regex;
    regmatch_t pmatch;
    MatchResults results;
    int status, offset = 0;

    results.matches = NULL;
    results.count = 0;

    if (regcomp(&regex, pattern, REG_EXTENDED) != 0) {
        return results;
    }
    while ((status = regexec(&regex, text + offset, 1, &pmatch, 0)) == 0) {
        if (results.matches == NULL) {
            results.matches = malloc(sizeof(MatchInfo));

        }
        else {
            results.matches = realloc(results.matches, sizeof(MatchInfo) * (results.count + 1));
        }

        int match_start = offset + pmatch.rm_so;
        int match_end = offset + pmatch.rm_eo;

        results.matches[results.count].start = match_start;
        results.matches[results.count].end = match_end;
        results.matches[results.count].matched_str = strndup(text + match_start, match_end - match_start);

        results.count++;
        offset += pmatch.rm_eo; //

    }

    regfree(&regex);
    return results;
}
```

Listing 22: description

Cette fonction prend une chaîne de texte et un motif sous forme de chaîne de caractères, puis trouve toutes les correspondances du motif dans le texte.
Paramètres : text : La chaîne de caractères où effectuer la recherche. pattern : L'expression régulière définissant le motif à rechercher.
Retour :
MatchResults : Une structure contenant les résultats des correspondances trouvées. Déroulement de la fonction : Initialisation des variables : regex : Une structure utilisée pour compiler l'expression régulière.
pmatch : Structure contenant les informations sur la correspondance trouvée.
results : Structure initialisée pour stocker les résultats. Compilation de l'expression régulière : La fonction regcomp() compile le motif en une expression régulière. En cas d'échec, elle retourne immédiatement les résultats vides.
Recherche des correspondances : La boucle continue tant que regexec() trouve des correspondances dans la chaîne de texte. Les résultats sont ajustés en fonction de l'offset pour gérer les correspondances successives dans la chaîne. Si c'est la première correspondance trouvée, de la mémoire est allouée pour un seul MatchInfo. Si des correspondances supplémentaires sont trouvées, la mémoire est réallouée pour accommoder le nouveau résultat.
Pour chaque correspondance, les détails (début, fin, et la sous-chaîne correspondante) sont stockés dans results. Mise à jour de l'offset :
Après chaque correspondance, l'offset est mis à jour pour pointer après la dernière correspondance trouvée, ce qui permet de continuer la recherche dans le reste de la chaîne. Libération de la mémoire : La mémoire allouée à l'expression régulière est libérée avec regfree(). Retour des résultats : La structure MatchResults, contenant toutes les correspondances trouvées et leurs détails, est retournée.