Algorithmique des textes

Boyer-Moore

BONDU | DEROUET | KONATE | SARR

Description des méthodes

Description des méthodes

Méthode naïve

Exploration EXHAUSTIVE. On compare chaque charactère de la séquence avec chaque caractère du motif cherché jusqu'à trouver concordance…

Complexité de en O(L*K), avec L longueur de la séquence, K celle du motif.

Description des méthodes

Méthode naïve

Exploration EXHAUSTIVE. On compare chaque charactère de la séquence avec chaque caractère du motif cherché jusqu'à trouver concordance…

Complexité de en O(L*K), avec L longueur de la séquence, K celle du motif.

Méthode à un décalage

- On commence les comparaisons à partir de la fin du motif
- Prétraitement du motif pour créer une table de saut
- Formule du décalage : longueurMotif – indiceLettre - 1

Complexité de en O(L/K) dans le « meilleur » des cas (hmmm…)

Exemple & résultats (1)

```
--- Displaying jump table
---
* Case 0 : [ o | 3 ]
* Case 1 : [ u | 2 ]
* Case 2 : [ r | 1 ]
* Case 3 : [ s | 0 ]
--- Done ---
```

```
  ! (3)
ours
J'aime les ours polaires
^  ^ (0 | 3)
------------------------
       ! (3)
    ours
J'aime les ours polaires
    ^  ^ (4 | 7)
------------------------
          ! (3)
        ours
J'aime les ours polaires
        ^  ^ (8 | 11)
------------------------ jumped ! >>
             ! (3)
          ours
J'aime les ours polaires
          ^  ^ (11 | 14)
------------------------
```

Exemple & résultats (2)

```
--- Displaying jump table
---
* Case 0 : [ o | 3 ]
* Case 1 : [ u | 2 ]
* Case 2 : [ r | 1 ]
* Case 3 : [ s | 0 ]
--- Done ---
```

```
 ! (2)
          ours
J'aime les ours polaires
          ^  ^ (11 | 14)
------------------------
           ! (1)
          ours
J'aime les ours polaires
          ^  ^ (11 | 14)
------------------------
          ! (0)
          ours
J'aime les ours polaires
          ^  ^ (11 | 14)
------------------------
                ! (3)
             ours
J'aime les ours polaires
             ^  ^ (15 | 18)
------------------------
```

Exemple & résultats (3)

```
--- Displaying jump table
---
* Case 0 : [ o | 3 ]
* Case 1 : [ u | 2 ]
* Case 2 : [ r | 1 ]
* Case 3 : [ s | 0 ]
--- Done ---
```

```
 ! (3)
                    ours
J'aime les ours polaires
                    ^  ^ (19 | 22)
------------------------
                       ! (3)
                    ours
J'aime les ours polaires
                    ^  ^ (20 | 23)
------------------------
                      ! (2)
                    ours
J'aime les ours polaires
                    ^  ^ (20 | 23)
------------------------

Pattern found at indice(s) : {11}
```

Implémentation – Miscellaneous (1)

Signatures

```
dataP* initJumpTable(const char pattern[]);

void forward(int* ip, int* is, int* n, int* m);

int inPattern(char letter, const char pattern[], int init);

void displayJumpTable(dataP* jumpTable, int patternLen);

void displayComparison(const char pattern[], const char sequence[], int ip, int is);

int* boyer_moore(const char pattern[], const char sequence[], int display);
```

Implémentation – Miscellaneous (2)

Unique structure

```
1. typedef struct
2. {
3.   char letter; // Only useful for display
4.   int jump;
5. } dataP;
```

```
1. void forward(int* ip, int* is, int* n, int* m)
2. {
3.   *is += *n; // is forward
4.   *ip = *n-1; // ip init
5.
6.   if(*is > *m-1) //check if we are still in the s.
7.     *is = *m-1;
8. }
```

Avancer - Fonctionnement

1

2

Implémentation - Miscellaneous

Unique structure

```
1. typedef struct
2. {
3.   char letter; // Only useful for display
4.   int jump;
5. } dataP;
```

```
1. void forward(int* ip, int* is, int* n, int* m)
2. {
3.   *is += *n; // is forward
4.   *ip = *n-1; // ip init
5.
6.   if(*is > *m-1) //check if we are still in the s.
7.     *is = *m-1;
8. }
```

Avancer - Fonctionnement

1 Modif. indices

2

Implémentation - Miscellaneous

Unique structure

```
1. typedef struct
2. {
3.   char letter; // Only useful for display
4.   int jump;
5. } dataP;
```

```
1. void forward(int* ip, int* is, int* n, int* m)
2. {
3.   *is += *n; // is forward
4.   *ip = *n-1; // ip init
5.
6.   if(*is > *m-1) //check if we are still in the s.
7.     *is = *m-1;
8. }
```

Avancer - Fonctionnement

1. Modif. indices
2. Test dépassement

Implémentation

Table de saut

```
1.  // Setup the jumpTable with BOYER-MOORE specs.
2.  dataP* initJumpTable(const char pattern[])
3.  {
4.    int n = strlen(pattern);
5.    dataP* jumpTable = malloc(sizeof(dataP)*n);
6.
7.    for(int i = 0; i < n; i++)
8.    {
9.      jumpTable[i].letter = pattern[i];
10.     jumpTable[i].jump = n-i-1;
11.   }
12.
13.   return jumpTable;
14. }
```

1

2

3

Implémentation

Table de saut

```
1.  // Setup the jumpTable with BOYER-MOORE specs.
2.  dataP* initJumpTable(const char pattern[])
3.  {
4.    int n = strlen(pattern);
5.    dataP* jumpTable = malloc(sizeof(dataP)*n);
6.
7.    for(int i = 0; i < n; i++)
8.    {
9.      jumpTable[i].letter = pattern[i];
10.     jumpTable[i].jump = n-i-1;
11.   }
12.
13.   return jumpTable;
14. }
```

```
#include <string.h> !
```

1. Allocation
2.
3.

Implémentation

Table de saut

```
1.  // Setup the jumpTable with BOYER-MOORE specs.
2.  dataP* initJumpTable(const char pattern[])
3.  {
4.    int n = strlen(pattern);
5.    dataP* jumpTable = malloc(sizeof(dataP)*n);
6.
7.    for(int i = 0; i < n; i++)
8.    {
9.      jumpTable[i].letter = pattern[i];
10.     jumpTable[i].jump = n-i-1;
11.   }
12.
13.   return jumpTable;
14. }
```

1. Allocation
2. Remplissage
3.

Implémentation

Table de saut

```
1.  // Setup the jumpTable with BOYER-MOORE specs.
2.  dataP* initJumpTable(const char pattern[])
3.  {
4.    int n = strlen(pattern);
5.    dataP* jumpTable = malloc(sizeof(dataP)*n);
6.
7.    for(int i = 0; i < n; i++)
8.    {
9.      jumpTable[i].letter = pattern[i];
10.     jumpTable[i].jump = n-i-1;
11.   }
12.
13.   return jumpTable;
14. }
```

1. Allocation
2. Remplissage
3. Return

Implémentation

Table de saut

Affichage !

```
void displayJumpTable(dataP* jumpTable, int patternLen);
```

```
--- Displaying jump table ---
* Case 0 : [ m | 4 ]
* Case 1 : [ i | 3 ]
* Case 2 : [ a | 2 ]
* Case 3 : [ o | 1 ]
* Case 4 : [ u | 0 ]
--- Done ---
```

1. Allocation
2. Remplissage
3. Return

Implémentation

1

2

Présence d'une lettre dans un motif

```
1.  int inPattern(char letter, const char pattern[], int init)
2.  {
3.    for(int i = init; i >= 0; i--)
4.    {
5.      if(letter == pattern[i])
6.      {
7.        return i;
8.      }
9.    }
10.   return -1;
11. }
```

Implémentation

Présence d'une lettre dans un motif

Prototype

1

2

```
1.  int inPattern(char letter, const char pattern[], int init)
2.  {
3.    for(int i = init; i >= 0; i--)
4.    {
5.      if(letter == pattern[i])
6.      {
7.        return i;
8.      }
9.    }
10.   return -1;
11. }
```

Implémentation

Prototype 1

Recherche 2

Présence d'une lettre dans un motif

Utile plus tard

```
1.  int inPattern(char letter, const char pattern[], int init)
2.  {
3.    for(int i = init; i >= 0; i--)
4.    {
5.      if(letter == pattern[i])
6.      {
7.        return i;
8.      }
9.    }
10.   return -1;
11. }
```

Implémentation

Présence d'une lettre dans un motif

Prototype 1

Recherche 2

```
1.  int inPattern(char letter, const char pattern[], int init)
2.  {
3.    for(int i = init; i >= 0; i--)
4.    {
5.      if(letter == pattern[i])
6.      {
7.        return i;
8.      }
9.    }
10.   return -1;
11. }
```

Implémentation – Boyer-Moore (1)

```
int* boyer_moore(const char pattern[], const char sequence[], …)
{
  int* iFound = (int*)calloc(0, sizeof(int)); // Dynamic table to
store the indices found
(...)
  int n = strlen(pattern);
  int m = strlen(sequence);

  dataP* jumpTable = initJumpTable(pattern);
(...)

  int indiceP = n-1; // indice for pattern
  int indiceS = n-1; // indice for sequence
  int letterIn, tempTest;
  int patternsFound = 0;
  int end = 0;
[...]
```

1

2

Implémentation – Boyer-Moore (1)

```
int* boyer_moore(const char pattern[], const char sequence[], …)
{
  int* iFound = (int*)calloc(0, sizeof(int)); // Dynamic table to
store the indices found
(... test)
  int n = strlen(pattern);
  int m = strlen(sequence);

  dataP* jumpTable = initJumpTable(pattern);
(...)

  int indiceP = n-1; // indice for pattern
  int indiceS = n-1; // indice for sequence
  int letterIn, tempTest;
  int patternsFound = 0;
  int end = 0;
[...]
```

1 Indice des séquences trouvées

2

Implémentation – Boyer-Moore (1)

```
int* boyer_moore(const char pattern[], const char sequence[], …)
{
  int* iFound = (int*)calloc(0, sizeof(int)); // Dynamic table to
store the indices found
(...)
  int n = strlen(pattern);
  int m = strlen(sequence);

  dataP* jumpTable = initJumpTable(pattern);
(... display)

  int indiceP = n-1; // indice for pattern
  int indiceS = n-1; // indice for sequence
  int letterIn, tempTest;
  int patternsFound = 0;
  int end = 0;
[... while]
```

1

2 Set de variables

Implémentation – Boyer-Moore (2)

D'autres variables utiles (suite directe)

```
while(!end)
{
  (... display)
  letterIn = inPattern(sequence[indiceS], pattern, indiceP);
  // Checks if the pattern contains a specific letter, then
returns the indice of that letter in the pattern or -1

  /* ...doing that from the "init" indice (in order to restrain
the pattern for future comparisons) */

  tempTest = (n-1)-indiceP; /* Used to compare letters of a
sequence's window to a pattern using indiceP */
```

inPattern(…) !

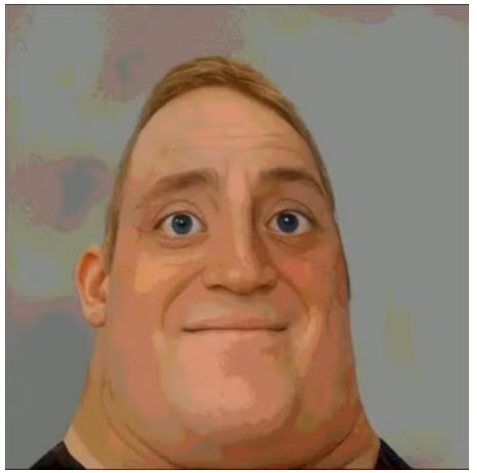
Implémentation – Boyer-Moore (2)

D'autres variables utiles (suite directe)

```
while(!end)
{
  (... display)
  letterIn = inPattern(sequence[indiceS], pattern, indiceP);
  // Checks if the pattern contains a specific letter, then
returns the indice of that letter in the pattern or -1

  // ...doing that from the "init" indice (in order to restrain
the pattern for future comparisons)

  tempTest = (n-1)-indiceP; // Used to compare letters of a
sequence's window to a pattern using indiceP
```



Oui ça fait beaucoup…

Implémentation – Boyer-Moore (3)

Première condition !

```
if(sequence[indiceS-tempTest] != pattern[indiceP]) // !(Last Letter)
{
    if(letterIn < 0) // Explain dude
    {
      if(indiceS == m-1)
        end = 1;
      else
        forward(&indiceP, &indiceS, &n, &m);
    }
    else // The letter have a chance to match the pattern
    {
      indiceS += jumpTable[letterIn].jump;
      if(indiceS > m-1)
        end = 1;
    }
```

I

La lettre pointée du motif ne coïncide pas avec la lettre pointée de la séquence

Implémentation – Boyer-Moore (3)

Première condition !

```
if(sequence[indiceS-tempTest] != pattern[indiceP]) // !(Last Letter)
{
    if(letterIn < 0)
    {
      if(indiceS == m-1)
        end = 1;
      else
        forward(&indiceP, &indiceS, &n, &m);
    }
    else // The letter have a chance to match the pattern
    {
      indiceS += jumpTable[letterIn].jump;
      if(indiceS > m-1)
        end = 1;
    }
```

I

La lettre pointée du motif ne coïncide pas avec la lettre pointée de la séquence

Implémentation – Boyer-Moore (3)

Première condition !

```
if(sequence[indiceS-tempTest] != pattern[indiceP]) // !(Last Letter)
{
    if(letterIn < 0) // Explain dude
    {
      if(indiceS == m-1) // End if we reach the sequence's end
        end = 1;
      else
        forward(&indiceP, &indiceS, &n, &m);
    }
    else // The letter have a chance to match the pattern
    {
      indiceS += jumpTable[letterIn].jump;
      if(indiceS > m-1)
        end = 1;
    }
```

I

La lettre pointée du motif ne coïncide pas avec la lettre pointée de la séquence

1 & lettre absente

2

Implémentation – Boyer-Moore (3)

Première condition !

```
if(sequence[indiceS-tempTest] != pattern[indiceP]) // !(Last Letter)
{
    if(letterIn < 0) // Explain dude
    {
      if(indiceS == m-1)
        end = 1;
      else
        forward(&indiceP, &indiceS, &n, &m);
    }
    else // The letter have a chance to match the pattern
    {
      indiceS += jumpTable[letterIn].jump; // Table de saut utilisée
      if(indiceS > m-1)
        end = 1;
    }
```

I

La lettre pointée du motif ne coïncide pas avec la lettre pointée de la séquence

1

2 & lettre présente

Implémentation – Boyer-Moore (3)

Première condition !

```
if(sequence[indiceS-tempTest] != pattern[indiceP]) // !(Last Letter)
{
    if(letterIn < 0) // Explain dude
    {
      if(indiceS == m-1)
        end = 1;
      else
        forward(&indiceP, &indiceS, &n, &m);
    }
    else // The letter have a chance to match the pattern
    {
      indiceS += jumpTable[letterIn].jump; // Table de saut utilisée
      if(indiceS > m-1) // Une condition d'arrêt
        end = 1;
    }
```

I

La lettre pointée du motif ne coïncide pas avec la lettre pointée de la séquence

1

2

End

Implémentation – Boyer-Moore (4)

Deuxième condition !

```
else if(indiceP == 0) // In that case the pattern match the sequence
  {
    patternsFound += 1;
    iFound = (int*)realloc(iFound, patternsFound*sizeof(int));
    (... Test)
    iFound[patternsFound-1] = indiceS-(n-1);

    if (indiceS == m-1) // If we reached the end
    {
      end = 1;
    }
    else
    {
      forward(&indiceP, &indiceS, &n, &m);
    }
  }
```

II

Les lettres pointées coïncident ET indiceP a déjà parcouru le motif entier

Implémentation – Boyer-Moore (4)

Deuxième condition !

```
else if(indiceP == 0) // In that case the pattern match the sequence
  {
    patternsFound += 1;
    iFound = (int*)realloc(iFound, patternsFound*sizeof(int));
    (... Test)
    iFound[patternsFound-1] = indiceS-(n-1);

    if (indiceS == m-1) // If we reached the end
    {
      end = 1;
    }
    else
    {
      forward(&indiceP, &indiceS, &n, &m);
    }
  }
```

II

Les lettres pointées coïncident ET indiceP a déjà parcouru le motif entier

Implémentation – Boyer-Moore (4)

Deuxième condition !

```
else if(indiceP == 0) // In that case the pattern match the sequence
  {
    patternsFound += 1;
    iFound = (int*)realloc(iFound, patternsFound*sizeof(int));
    (... Test)
    iFound[patternsFound-1] = indiceS-(n-1);

    if (indiceS == m-1) // If we reached the end
    {
      end = 1;
    }
    else
    {
      forward(&indiceP, &indiceS, &n, &m);
    }
  }
```

II

Les lettres pointées coïncident ET indiceP a déjà parcouru le motif entier

1 Stockage de l'indice trouvé

2

Implémentation – Boyer-Moore (4)

Deuxième condition !

```
else if(indiceP == 0) // In that case the pattern match the sequence
  {
    patternsFound += 1;
    iFound = (int*)realloc(iFound, patternsFound*sizeof(int));
    (... Test)
    iFound[patternsFound-1] = indiceS-(n-1);

    if (indiceS == m-1) // If we reached the end
    {
      end = 1;
    }
    else
    {
      forward(&indiceP, &indiceS, &n, &m);
    }
  }
```

II

Les lettres pointées coïncident ET indiceP a déjà parcouru le motif entier

1

2 On avance ou on s'arrête

Implémentation – Boyer-Moore (5)

Sinon !

```
else
{
    indiceP--; // Pour poursuivre la comparaison :)
}
```

III

Implémentation – Boyer-Moore (5)

Affichage des résultats, free, return

```
if(patternsFound > 0)
{
  printf("\n\nPattern found at indice(s) : ");
  for (int i = 0; i < patternsFound; i++)
  {
    printf("{%d} ", iFound[i]);
  }
}
else
{
  printf("\n\nPattern not found.");
}

free(jumpTable); // Hehe.
return iFound;
```

Affichage !

```
Pattern found at indice(s) :
{14} {19} {24}
```

Analyse de l'algorithme

Correction

L'algorithme est correct : une fois lancé on obtient du programme un pointeur sur un tableau (iFound) qui recence les indices des sous-séquences trouvées.

Terminaison

La terminaison de toutes les fonctions différentes de boyer_moore() est immédiate. La terminaison de boyer_moore() est vérifiée, elle dépend de la variable « end » (qui dépend de l'indice de la séquence). [Revenir]

Complexité

Hors-programme, mais anyways ! (Oral)

Version d'Iwan le subversif (1)

```c
typedef struct str_listTrouves
{
  int taille;
  int* positions;

} ListTrouves;

char* fic_to_txt(char* filepath);

int recherche_exhaustive(char* texte, char* motif);

listeChainee* recherche_exhaustive_liste(char* texte, char* motif);
```

Version d'Iwan le subversif (1)

```c
typedef struct str_listTrouves
{
  int taille;
  int* positions;

} ListTrouves;

char* fic_to_txt(char* filepath);

int recherche_exhaustive(char* texte, char* motif);

listeChainee* recherche_exhaustive_liste(char* texte, char* motif);
```

Version d'Iwan le subversif (1)

```
typedef struct str_listTrouves
{
  int taille;
  int* positions;

} ListTrouves;

char* fic_to_txt(char* filepath);
//retourne un char* contenant le texte d'un fichier entier
//lit uniquement les 10 000 premiers charactères de chaque ligne

int recherche_exhaustive(char* texte, char* motif);

listeChainee* recherche_exhaustive_liste(char* texte, char* motif);
```

Version d'Iwan le subversif (1)

```
typedef struct str_listTrouves
{
  int taille;
  int* positions;

} ListTrouves;

char* fic_to_txt(char* filepath);

int recherche_exaustive(char* texte, char* motif);
//retourne l'indice de la derniere 1ere occurence trouvée dans une chaine
//recherche très (trop) exaustive
//retourne -1 si aucune occurence du motif n'est dans le texte

listeChainee* recherche_exaustive_liste(char* texte, char* motif);
//retourne une liste comptant toutes les occurences trouvees
```

Version d'Iwan le subversif (2)

```
listeChainee* boyer_moore_simple(char* texte, char* motif);
//retourne une liste chainée comportant les occurrences trouvées dans une chaine
//utilise l'algo de Boyer_Moore simple

int* tableau_saut_suffixe(char* motif);
```

Et à propos de Boyer-Moore II…

Laissons Emile nous en parler…

Conclusion, perspectives