Correction du TD 13

Julien Reichert

Recherche dans un texte - Algorithme naïf

Exercice 1

```
int* recherche_naive(char* s, char* m) {
  int ns = strlen(s);
  int nm = strlen(m);
  int c_f = 0;
  int* final = malloc(ns * sizeof(int));
  for (int i = 0 ; i < ns - nm ; i++)
  {
    int count = 0;
    while (count < nm && s[i + count] == m[count]) count += 1;
    if (count == nm)
    {
      final[c_f] = i;
      c_f += 1;
    }
  }
  int* final2 = malloc((c_f+1) * sizeof(int));
  final2[0] = c_f;
  for (int i = 0 ; i < c_f ; i += 1) final2[i+1] = final[i];
  free(final);
  return final2;
}
```

Recherche dans un texte - Algorithme de Boyer-Moore

Exercice 2

	'C'	'H'	'E'	'R'	'Z'
0 ('C')	OK				
1 ('H')	+1	OK			
2 ('E')	+2	+1	OK		
3 ('R')	+3	+2	+1	OK	
4 ('C')	OK	+3	+2	+1	
5 ('H')	+1	OK	+3	+2	
6 ('E')	+2	+1	OK	+3	
7 ('Z')	+3	+2	+1	+4	OK

Exercice 3

Premier cas : Recherche de `"CHERCHEZ"` dans `"CHERCHER"` répété n fois.

Avec l'algorithme naïf : 8 comparaisons à partir de chaque "C" sauf le dernier qui est hors champ, 1 comparaison à partir de chaque autre lettre sauf les 7 dernières positions, d'où $(n-1) \times (8+1+1+1+8+1+1+1)+8$ soit $22n-14$ comparaisons.

Avec l'algorithme de Boyer-Moore : décalage de 4 crans à chaque étape jusqu'à atteindre la fin et échouer, d'où $2n-1$ comparaisons.

Deuxième cas : Recherche de `"CHERCHEZ"` dans `"CHERCHEZ"` répété n fois.

Avec l'algorithme naïf : 8 comparaisons à partir de chaque "C" dont le « numéro de l'occurrence » est pair, 4 à partir de chaque autre C sauf le dernier qui est hors champ, 1 comparaison à partir de chaque autre lettre sauf les 7 dernières positions, d'où $(n-1) \times (8+1+1+1+4+1+1+1)+8$ soit $18n-10$ comparaisons.

Avec l'algorithme de Boyer-Moore : complétion du motif et décalage d'un cran, puis décalage de 3 crans puis décalage de 4 crans pour tomber sur le motif suivant jusqu'à atteindre la fin et, d'où $(n-1) \times (8+1+1)+8$ soit $10n-2$ comparaisons.

Troisième cas : Recherche de `"CHERCHEZ"` dans `"CHEZCHEZ"` répété n fois.

Avec l'algorithme naïf : 4 comparaisons à partir de chaque "C" sauf le dernier qui est hors champ, 1 comparaison à partir de chaque autre lettre sauf les 7 dernières positions, d'où $(n-1) \times (4+1+1+1+4+1+1+1)+4$ soit $14n-10$ comparaisons.

Avec l'algorithme de Boyer-Moore : 4 comparaisons puis décalage de 4 crans à chaque étape jusqu'à atteindre la fin et échouer, d'où $10n-5$ comparaisons.

Exercice 4

Version optimisée pour `"CHERCHEZ"` : tous les décalages sauf la dernière ligne sont remplacés par des +8, car il est impossible de trouver le motif dans la zone si un `'Z'` y figure à la fin et que la reconnaissance échoue par ailleurs, puisque cette lettre est unique dans le motif.

Version optimisée pour `"CHERCHER"` :

	'C'	'H'	'E'	'R'
0 ('C')	OK	+4	+4	+4
1 ('H')	+4	OK	+4	+4
2 ('E')	+4	+4	OK	+4
3 ('R')	+4	+4	+4	OK
4 ('C')	OK	+8	+8	+8
5 ('H')	+8	OK	+8	+8
6 ('E')	+8	+8	OK	+8
7 ('R')	+3	+2	+1	OK

Exercice 5

```
let redirection motif =
  let n = String.length motif in
  let t = Array.init n (fun _ -> Hashtbl.create 8) in
  Hashtbl.add t.(0) motif.[0] 0;
  for i = 1 to n-1 do
    Hashtbl.iter (fun cle valeur -> Hashtbl.add t.(i) cle (valeur+1)) t.(i-1);
    Hashtbl.replace t.(i) motif.[i] 0;
  done;
  t;;

let boyer_moore chaine motif =
  let t_r = redirection motif in
  let nm = String.length motif in
  let nc = String.length chaine in
  let rec aux ic im =
    if ic > nc - nm then []
    else match Hashtbl.find_opt t_r.(im) chaine.[ic + im] with
      | Some 0 when im = 0 -> ic :: aux (ic + 1) (nm - 1)
      | Some 0 -> aux ic (im - 1)
      | Some a -> aux (ic + a) (nm - 1)
      | None -> aux (ic + im + 1) (nm - 1)
  in aux 0 (nm-1);;
```

Recherche dans un texte - Algorithme de Rabin-Karp

Exercice 6

Avec la convention du cours stipulant que le degré augmente de droite à gauche (plus facile pour l'algorithme final), le haché de `"mot"` est $116 + 111 \times 31 + 109 \times 31^2 = 108306$.

Exercice 7

Plus petit haché possible : 0 en utilisant le caractère de fin de chaîne trois fois (même si ce n'est pas censé se produire). Plus grand haché possible : 253215 en utilisant le caractère d'indice 255 trois fois. En raison des collisions, le nombre de hachés différents n'est pas de $256^3 = 16777216$ mais de 253216 (tous les entiers de l'intervalle, en fait).

Exercice 8

```
int hachage (char* s, int b, int p)
{
  int final = 0;
  for (int i = 0 ; s[i] != '\0' ; i += 1)
  {
    final = final * b + (int) s[i];
    if (p != 0) final = final % p;
  }
  return final;
}
```

Exercice 9

```
int hachage_suivant (int x, char* s, int b, int k, int bpkm1, int ind)
{
  return (x - s[ind] * bpkm1) * b + s[ind + k];
}
```

Exercice 10

```
int hachagebis (char* s, int b, int p, int taille)
{
  int final = 0;
  for (int i = 0 ; i < taille ; i += 1)
  {
    final = final * b + (int) s[i];
    if (p != 0) final = final % p;
  }
  return final;
}
```

La fonction ci-avant calcule le premier haché d'une tranche. Ci-après, l'algorithme proprement dit.

```
int* rabin_karp(char* s, char* m, int b, int p) {
  int ns = strlen(s);
  int nm = strlen(m);
  if (nm > ns) return NULL;
  int c_f = 0;
  int* final = malloc(ns * sizeof(int));
  int hm = hachage(m, b, p);
  int hs = hachagebis(s, b, p, nm);
  int bpkm1 = 1;
  for (int i = 1 ; i < nm ; i += 1) bpkm1 = p == 0 ? bpkm1 * b : (bpkm1 * b) % p;
  for (int i = 0 ; i < ns - nm ; i++)
  {
    if (hm == hs)
    {
      int count = 0;
      while (count < nm && s[i + count] == m[count]) count += 1;
      if (count == nm)
      {
        final[c_f] = i;
        c_f += 1;
      }
    }
    hs = hachage_suivant(hs, s, b, nm, bpkm1, i);
    if (p != 0) hs = (hs % p + p) % p; // Risque de valeurs négatives ici !
  }
  int* final2 = malloc((c_f+1) * sizeof(int));
  final2[0] = c_f;
  for (int i = 0 ; i < c_f ; i += 1) final2[i+1] = final[i];
  free(final);
  return final2;
}

int main(int argc, char** argv)
{
  int b = atoi(argv[1]);
  int p = argc > 2 ? atoi(argv[2]) : 0;
  char m[] = "ABCD";
  char s[] = "ACBDACBDABACDBBDA";
  int* rk = rabin_karp(s, m, b, p);
  printf("Nombre d'occurrences du motif : %d.\n", rk[0]);
  printf("Première (seule en fait) occurrence : en %d.\n", rk[1]);
  free(rk);
  return 0;
}
```