TP : Algorithmes de recherche dans un texte

Langage C

1 Boyer-Moore-Horspool

Q1 Écrire la fonction dont le prototype est `int * derniereOccurrence(char *m);`.

```c
void bench(){
  char * m = "aabaezzAc";
  int * cd = dernieresOccurrence(m);
  int n = strlen(m);
  int i=0;
  for(i=0; i<n-1; i++)
    printf("%c:%d; " ,m[i],cd[(int)m[i]]);
  printf("%c:%d\n",m[i],cd[(int)m[i]]);
  free(cd);
}

int main(){
  bench();
  return EXIT_SUCCESS;
}
```

On obtient le rendu :

```
a:3; a:3; b:2; a:3; e:4; z:6; z:6; A:7; c:-1
```

Q2 Écrire le fonction `int bmh(char *motif, char *chaine)` qui implante l'algorithme de Boyer-Moore-Horspool.

```c
void bench2(){
  char * m = "string";
  char * t = "wikipedia";
  printf("motif=%s; texte =%s\n",m,t);
  printf("position=%d\n",bmh(m,t));
}

void bench3(){
  char * t = "stupid_spring_string_aa";
  char * m = "string";
  printf("motif=%s; texte =%s\n",m,t);
  printf("position=%d\n",bmh(m,t));
}

```

```
15  int main(){
16    bench2();bench3();
17    return EXIT_SUCCESS;
18  }
```

On obtient le rendu :

```
motif=string; texte =wikipedia
position=-1
motif=string; texte =stupid_spring_string_aa
position=14
```

2 Rabin-Karp

Pour le hachage, on prend comme paramètre $B = 256$ (nombre de carcatères en code ASCII) et un nombre premier $P = 1869461003$. Observons que P^2 tient dans le type `uint64_t` des entiers 64 bits non signés puisque $P < 2^{32}$. Ce choix de P « pas trop grand » permet de faire un calcul d'exponentiation modulo sans erreur. Des produits de la forme `(x*y) % P` ne posent alors pas de problème si x, y sont plus petits que P.

On donne :

```
1  #include <stdint.h>
2
3  uint64_t arith_power_mod(uint64_t x, uint64_t n, uint64_t m){
4    uint64_t y = 1;
5    while (n>0){
6      // invariant y_k * x_k ^{n_k} = x_0 ^{n_0} % m
7      if (n % 2 == 1) y = (y * x) % m;
8      x = (x *x) % m;
9      n= n / 2;
10   }
11   return y;
12 }
```

Listing 1 – calcul de $x^n \mod m$

Les opérations modulo se font à chaque étape, sinon les débordements arithmétiques (en C, les calculs arithmétiques se font modulo) conduiraient à des erreurs.

On rappelle à toute fin utile l'existence dans `string.h` de la fonction :

```
1  int strncmp( const char * first, const char * second, size_t length );
```

- first : la première chaîne de caractères à comparer.
- second : la seconde chaîne de caractères à comparer.
- length : le nombre maximal (un entier non signé) de caractères à comparer.

Soit les deux chaînes sont égales : dans ce cas, une valeur nulle sera retournée. Soit la première chaîne est plus petite que la seconde (dans l'ordre lexicographique) : dans ce cas, une valeur négative

sera retournée. Soit la première chaîne est plus grande que la seconde : dans ce dernier cas, une valeur positive sera renvoyées. Dans tous les cas, la valeur absolue indiquera la position du premier caractères permettant de produire le résultat (d'après Koors).

Q3 Écrire la fonction `uint64_t empreinte_rabin(char *s, int len)` qui calcule l'empreinte de Rabin du préfixe de `s` formé de ses `len` premiers caractères. Cette fonction efectue un calcul d'image par une fonction polynôme en suivant l'algorithme de Horner.
Avec :

```
1  void bench_empreinte(){
2    char * m = "string";
3    int n = strlen(m);
4    printf("empreinte_rabin(%s,%d)=%ld\n",m,n,empreinte_rabin(m,n));
5  }
6
```

on obtient :

```
empreinte_rabin(string,6)=92965031
```

Q4 Écrire la fonction `int rabin_karp(char *m, char *t)` qui implante l'algorithme du cours et renvoie le nombre de fois où le motif `m` est présent dans le texte `t`. En outre, chaque fois qu'une occurrence est trouvée, sa position s'affiche.

Contrainte À part les deux premiers calculs de hachés (obligatoirement en $O(|\texttt{m}|)$), les hachés suivants s'obtiennent en $O(1)$.
Avec :

```
1  void bench_rk(){
2    char * m = "string";
3    char * t = "strict_string_strie_string_stringer_strange";
4    printf("rabin_karp(%s,%s)=%d\n",m,t,rabin_karp(m,t));
5  }
6
```

On obtient :

```
occurrence à la position 7
occurrence à la position 20
occurrence à la position 27
rabin_karp(string,strict_string_strie_string_stringer_strange)=3
```