Proiectarea algoritmilor: Algoritmul Rabin-Karp

March 21, 2020

Descriere - noțiunea de simbol

Acest algoritm utilizează tehnica tabelelor de dispersie (hash).

Descriere - noțiunea de simbol

Acest algoritm utilizează tehnica tabelelor de dispersie (hash).

Un simbol este o secvență de *m* caractere.

Descriere - noțiunea de simbol

Acest algoritm utilizează tehnica tabelelor de dispersie (hash).

Un simbol este o secvență de m caractere.

Exemplu: dacă $m = 3$, `bla` este un simbol.

Descriere - noțiunea de simbol

Acest algoritm utilizează tehnica tabelelor de dispersie (hash).

Un simbol este o secvență de m caractere.

Exemplu: dacă $m = 3$, `bla` este un simbol.

Descriere - așezarea simbolurilor în tabele hash

$h(s)$	s
0	`aaa`
1	`aab`
...	
962	`bla`
...	
17575	`zzz`

Descriere - așezarea simbolurilor în tabele hash

$h(s)$	s
0	`aaa`
1	`aab`
...	
962	`bla`
...	
17575	`zzz`

Să ne imaginăm că toate simbolurile posibile sunt memorate într-o tabelă de dispersie foarte mare, astfel încât nu există coliziune.

Descriere - potrivirea pe o poziție

Cum testăm dacă patternul p apare la poziția i în textul s?

Descriere - potrivirea pe o poziție

Cum testăm dacă patternul p apare la poziția i în textul s?
Este suficient să verificăm dacă $h(p) = h(s[i..i + m - 1])$.

Descriere - calculul valorii funcției hash

Este suficient să calculăm $h(p)$ o singură dată:

Descriere - calculul valorii funcției hash

Este suficient să calculăm $h(p)$ o singură dată:

```
val_hash_pattern = h(p)
for (i = 0; i < n - m; ++i) {
  if (val_hash_pattern == h(s[i .. i + m - 1])) {
    return i;
  }
}
return -1;
```

Descriere - calculul valorii funcției hash

Este suficient să calculăm $h(p)$ o singură dată:

```
val_hash_pattern = h(p)
for (i = 0; i < n - m; ++i) {
  if (val_hash_pattern == h(s[i .. i + m - 1])) {
    return i;
  }
}
return -1;
```

Totuși, calculul `h(s[i .. i + m - 1])` se efectuează la fiecare iterație a buclei.

Descriere - calculul valorii funcției hash

Este suficient să calculăm $h(p)$ o singură dată:

```
val_hash_pattern = h(p)
for (i = 0; i < n - m; ++i) {
  if (val_hash_pattern == h(s[i .. i + m - 1])) {
    return i;
  }
}
return -1;
```

Totuși, calculul `h(s[i .. i + m - 1])` se efectuează la fiecare iterație a buclei.

Cum putem evita să refacem calculul `h(s[i .. i + m - 1])` la fiecare iterație?

Descriere - calculul valorii funcției hash

Cum putem evita să refacem calculul `h(s[i .. i + m - 1])` **la fiecare iterație?**

Idee:

Descriere - calculul valorii funcției hash

Cum putem evita să refacem calculul `h(s[i .. i + m - 1])` la fiecare iterație?

Idee:

Definim funcția *h* astfel încât să putem calcula foarte rapid:

`h(s[i .. i + m - 1])` în funcție de `h(s[i - 1 .. i + m - 2])`.

Descriere - calculul valorii funcției hash

Cum putem evita să refacem calculul `h(s[i .. i + m - 1])` la fiecare iterație?

Idee:

Definim funcția *h* astfel încât să putem calcula foarte rapid:
`h(s[i .. i + m - 1])` în funcție de `h(s[i - 1 .. i + m - 2])`.

```
val_hash_pattern = h(p)
val_hash_subsir = h(s[0 .. m - 1])
for (i = 0; i < n - m; ++i) {
  if (val_hash_pattern == val_hash_subsir) {
    return i;
  }
  val_hash_subsir = update(val_hash_subsir);
}
return -1;
```

Funcția de dispersie (hash) - idee

Un mod convenabil de a defini funcția de dispersie este următorul:

Funcția de dispersie (hash) - idee

Un mod convenabil de a defini funcția de dispersie este următorul:

Se consideră fiecare șir de m caractere ca fiind reprezentarea unui număr întreg în baza d, unde d este numărul de caractere din alfabet.

Funcția de dispersie (hash) - idee

Un mod convenabil de a defini funcția de dispersie este următorul:

Se consideră fiecare șir de m caractere ca fiind reprezentarea unui număr întreg în baza d, unde d este numărul de caractere din alfabet.

Exemplu: Dacă alfabetul este $\{a, b, \ldots, z\}$, atunci patternul `bla` are asociat numărul $1 * 26^2 + 11 * 26^1 + 0 * 26^0 = 962$.

Funcția de dispersie (hash) - definiție

În general, numărul corespunzător unui șir t de lungime l este:

$$x = t[0]d^{l-1} + t[1]d^{l-2} + \cdots + t[l-1]d^0$$

Funcția de dispersie (hash) - definiție

În general, numărul corespunzător unui șir t de lungime l este:

$$x = t[0]d^{l-1} + t[1]d^{l-2} + \cdots + t[l-1]d^0$$

Funcția de dispersie h va fi definită prin

$$h(t) = x \bmod q,$$

unde q este un număr prim foarte mare.

Funcția de dispersie (hash) - definiție

În general, numărul corespunzător unui șir t de lungime l este:

$$x = t[0]d^{l-1} + t[1]d^{l-2} + \cdots + t[l-1]d^{0}$$

Funcția de dispersie h va fi definită prin

$$h(t) = x \bmod q,$$

unde q este un număr prim foarte mare.
În formula de mai sus, am făcut un abuz de notație: prin $t[i]$ înțelegem atât caracterul $t[i]$, cât și indexul acestui caracter în alfabet.

Algoritmul Rabin-Karp - calculul funcției hash

Cum calculăm eficient formula
$h(t, l) = t[0]d^{l-1} + t[1]d^{l-2} + \cdots + t[l-1]d^0$.

Algoritmul Rabin-Karp - calculul funcției hash

Cum calculăm eficient formula
$h(t, l) = t[0]d^{l-1} + t[1]d^{l-2} + \cdots + t[l-1]d^0$.

```
h(t, l)
{
  s = 0;
  p = 1;
  for (i = l - 1; i >= 0; --i) {
    // invariant: p = pow(d, l - 1 - i)
    s = s + t[i] * p;
    p = p * d;
    s = s % q;
  }
  return s;
}
```

Calculul funcției hash pentru fiecare subșir

Cum putem găsi rapid valoarea `y = h(s[i..i + m - 1])` știind valoarea `x = h(s[i - 1..i + m - 2])`?

Calculul funcției hash pentru fiecare subșir

Cum putem găsi rapid valoarea `y = h(s[i..i + m - 1])` știind valoarea `x = h(s[i - 1..i + m - 2])`?

$$y = (s_i d^{m-1} + s_{i+1} d^{m-2} + \cdots + s_{i+m-2} d^1 + s_{i+m-1} d^0) \bmod q$$

Calculul funcției hash pentru fiecare subșir

Cum putem găsi rapid valoarea `y = h(s[i..i + m - 1])` știind valoarea `x = h(s[i - 1..i + m - 2])`?

$$\begin{aligned} y = & \quad (s_i d^{m-1} + s_{i+1} d^{m-2} + \cdots + s_{i+m-2} d^1 + s_{i+m-1} d^0) \bmod q \\ = & \quad (s_{i-1} d^m + s_i d^{m-1} + s_{i+1} d^{m-2} + \cdots + s_{i+m-2} d^1 + s_{i+m-1} d^0 - s_{i-1} d^m) \bmod q \end{aligned}$$

Calculul funcției hash pentru fiecare subșir

Cum putem găsi rapid valoarea `y = h(s[i..i + m - 1])` știind valoarea `x = h(s[i - 1..i + m - 2])`?

$$
\begin{aligned}
y = \quad & (s_i d^{m-1} + s_{i+1} d^{m-2} + \cdots + s_{i+m-2} d^1 + s_{i+m-1} d^0) \bmod q \\
= \quad & (s_{i-1} d^m + s_i d^{m-1} + s_{i+1} d^{m-2} + \cdots + s_{i+m-2} d^1 + s_{i+m-1} d^0 - s_{i-1} d^m) \bmod q \\
= \quad & (d(s_{i-1} d^{m-1} + s_i d^{m-2} + s_{i+1} d^{m-2} + \cdots + s_{i+m-2} d^0) + s_{i+m-1} d^0 - s_{i-1} d^m) \bmod q
\end{aligned}
$$

Calculul funcției hash pentru fiecare subșir

Cum putem găsi rapid valoarea `y = h(s[i..i + m - 1])` știind valoarea `x = h(s[i - 1..i + m - 2])`?

$$
\begin{aligned}
y = &\quad (s_i d^{m-1} + s_{i+1} d^{m-2} + \cdots + s_{i+m-2} d^1 + s_{i+m-1} d^0) \bmod q \\
= &\quad (s_{i-1} d^m + s_i d^{m-1} + s_{i+1} d^{m-2} + \cdots + s_{i+m-2} d^1 + s_{i+m-1} d^0 - s_{i-1} d^m) \bmod q \\
= &\quad (d(s_{i-1} d^{m-1} + s_i d^{m-2} + s_{i+1} d^{m-2} + \cdots + s_{i+m-2} d^0) + s_{i+m-1} d^0 - s_{i-1} d^m) \bmod q \\
= &\quad (dx + s_{i+m-1} d^0 - s_{i-1} d^m) \bmod q
\end{aligned}
$$

Algoritmul Rabin-Karp - update-ul funcției hash

Știind $x = h(s[i-1..i+m-2])$, cum calculăm eficient $y = h(s[i..i+m-1])$?

Algoritmul Rabin-Karp - update-ul funcției hash

Știind $x = h(s[i-1..i+m-2])$, cum calculăm eficient $y = h(s[i..i+m-1])$?
Trebuie să știm, bineînțeles, și valorile $c_{old} = s[i-1], c_{new} = s[i+m-1]$.

Algoritmul Rabin-Karp - update-ul funcției hash

Știind $x = h(s[i-1..i+m-2])$, cum calculăm eficient $y = h(s[i..i+m-1])$?
Trebuie să știm, bineînțeles, și valorile $c_{old} = s[i-1], c_{new} = s[i+m-1]$.

```
update(x, cnew, cold) {
  return (d * x + cnew + q - (cold * pow(d, m)) % q) % q;
}
```

Algoritmul Rabin-Karp - update-ul funcției hash

Știind $x = h(s[i-1..i+m-2])$, cum calculăm eficient
$y = h(s[i..i+m-1])$?
Trebuie să știm, bineînțeles, și valorile $c_{old} = s[i-1], c_{new} = s[i+m-1]$.

```
update(x, cnew, cold) {
  return (d * x + cnew + q - (cold * pow(d, m)) % q) % q;
}
```

Codul corespunde formulei obținute pe slide-ul precedent:
$y = (dx + s[i+m-1]d^0 - s[i-1]d^m) \bmod q$.

Algoritmul Rabin-Karp

```
val_hash_pattern = h(p)
val_hash_subsir = h(s[0 .. m - 1])
for (i = 0; i < n - m; ++i) {
  if (val_hash_pattern == val_hash_subsir) {
    if (apare_la_pozitia(s, n, p, m, i)) {
      return i;
    }
  }
  val_hash_subsir = update(val_hash_subsir, s[i], s[i + m]);
}
return -1;
```

Algoritmul Rabin-Karp

```
val_hash_pattern = h(p)
val_hash_subsir = h(s[0 .. m - 1])
for (i = 0; i < n - m; ++i) {
  if (val_hash_pattern == val_hash_subsir) {
    if (apare_la_pozitia(s, n, p, m, i)) {
      return i;
    }
  }
  val_hash_subsir = update(val_hash_subsir, s[i], s[i + m]);
}
return -1;
```

Observație: din cauza coliziunilor posibile, dacă valorile funcțiilor hash corespund pe două subșiruri, trebuie să ne asigurăm, folosind de exemplu funcția `apare_la_pozitia`, ca cele două șiruri sunt într-adevăr egale.

Observații

- Pentru a evita lucrul cu numere mari, operațiile se execută modulo un număr natural q.

Observații

- Pentru a evita lucrul cu numere mari, operațiile se execută modulo un număr natural q.
- Ne putem imagina ca q este dimensiunea "tabelei" hash care memorează "simbolurile" (= secvențele de m caractere) din s.

Observații

- Pentru a evita lucrul cu numere mari, operațiile se execută modulo un număr natural q.
- Ne putem imagina ca q este dimensiunea "tabelei" hash care memorează "simbolurile" (= secvențele de m caractere) din s.
- Pentru o dispersie bună, q trebuie să fie un număr prim mare.

Observații

- Pentru a evita lucrul cu numere mari, operațiile se execută modulo un număr natural q.
- Ne putem imagina ca q este dimensiunea "tabelei" hash care memorează "simbolurile" (= secvențele de m caractere) din s.
- Pentru o dispersie bună, q trebuie să fie un număr prim mare.
- În practică, dacă q este bine ales, numărul de coliziuni este mic și se poate considera că algoritmul rulează în timp $O(n+m)$.

Exercițiu: incercați să găsiți s, p și q astfel încât timpul de rulare al algoritmului să fie $O(n \cdot m)$.

Algoritmul Rabin-Karp: o posibilă implementare C

```
#define REHASH(a, b, h) ((((h)-(a)*dM) << 1) (b))
int RK(char *p, int m, char *s, int n) {
  long dM, hs, hp, i, j;
  /* Preprocesare */
  for (dM = i = 1; i < m; ++i) dM = (dM << 1); // d = 2
  for (hp = hs = i = 0; i < m; ++i) {
    hp = ((hp << 1) + p[i]);
    hs = ((hs << 1) + s[i]);
  }
  /* Cautare */
  i = 0;
  while (i <= n-m) {
    if (hp == hs && memcmp(p, s + i, m) == 0) return i;
    hs = REHASH(s[i], s[i + m], hs);
    ++i;
  }
  return -1;
}
```

sursa: http://www-igm.univ-mlv.fr/ lecroq/string/node5.html

Analiza implementării C

- S-a presupus că $d = 2$. Dacă $d = 2^k$, atunci se poate utiliza relația $d^{m-1} = (2^k)^{m-1} = (2^{m-1})^k$.

Analiza implementării C

- S-a presupus că $d = 2$. Dacă $d = 2^k$, atunci se poate utiliza relația $d^{m-1} = (2^k)^{m-1} = (2^{m-1})^k$.
- Dacă `long` este reprezentat pe 64 biți, atunci `dM` este 0 pentru $m = 65$. Deci trebuie să avem $m \leq \frac{65}{k}$.

Analiza implementării C

- S-a presupus că $d = 2$. Dacă $d = 2^k$, atunci se poate utiliza relația $d^{m-1} = (2^k)^{m-1} = (2^{m-1})^k$.
- Dacă `long` este reprezentat pe 64 biți, atunci `dM` este 0 pentru $m = 65$. Deci trebuie să avem $m \leq \frac{65}{k}$.
- Nu mai este utilizat numărul prim q deoarece se utilizează aritmetica modulară peste long.

Analiza implementării C

- S-a presupus că $d = 2$. Dacă $d = 2^k$, atunci se poate utiliza relația $d^{m-1} = (2^k)^{m-1} = (2^{m-1})^k$.
- Dacă `long` este reprezentat pe 64 biți, atunci `dM` este 0 pentru $m = 65$. Deci trebuie să avem $m \leq \frac{65}{k}$.
- Nu mai este utilizat numărul prim q deoarece se utilizează aritmetica modulară peste long.

O implementare Java cu q determinat aleatoriu (Robert Sedgewick and Kevin Wayne) poate fi gasita la adresa http://algs4.cs.princeton.edu/53substring/RabinKarp.java.html.