Algorytmy tekstowe - Haszowanie

Wstęp

Każdy z nas umie określić, czy dwie liczby są równe. Nietrudno się domyślić, że podobna umiejętność jest przydatna także w przypadku tekstów. Na dzisiejszej lekcji poznasz metodę haszowania. Jej zaletą jest stosunkowo łatwa implementacja i efektywność działania. Posiada również pewną wadę, ale o tym później.

Nieoptymalny sposób porównywania dwóch słów

Na początku spróbujmy rozwiązać prostszy problem. Mając dane dwa słowa W i S, chcielibyśmy sprawdzić, czy podsłowa $[i;j]W$ i $[k;l]S$ są takie same.

`abacababca` i `bcabaccabc` (zaznaczone przedziały nie są takie same)
`aacbbaacab` i `cabbbaaccc` (zaznaczone przedziały są takie same)

Fakt 1: Słowa są równe, gdy literki na odpowiednich pozycjach są takie same. Innymi słowy, jeśli zachodzi:

$$W[i] = S[k], W[i+1] = S[k+1] ... W[j] = S[l]$$

to słowa są równe. W przeciwnym wypadku się różnią. Załóżmy, że S i W mają długość $n (1 \leqslant n \leqslant 10^6)$. Porównywanie kolejnych literek zajmuje $O(k-l+1)$, czyli w najgorszym wypadku $O(n)$ czasu. Co gdybyśmy chcieli porównać w ten sposób $q (1 \leqslant q \leqslant 10^6)$ różnych par podsłów? Odbyłoby się to w czasie $O(qn)$, czyli w najgorszym wypadku $O(10^6 \cdot 10^6) = O(10^{12})$. Jeśli nie chcemy czekać kilku lat na wynik i nie mamy superkomputera, musimy wymyślić lepszą metodę.

```
bool same=true;
for(int i=0;i<length;i++)
{
    if(w1[i]!=w2[i])same=false;
}
return same;
```

W poszukiwaniu szybkości

Spróbujmy wymyślić inny, szybszy sposób porównywania słów. Tak jak wcześniej zauważyłem, umiemy porównywać liczby. Wykorzystajmy to! Stwórzmy taką funkcję $F(s_i)$, gdzie s_i jest słowem, że: $F(s_1) = F(s_2)$, gdy $s_1 = s_2$ oraz $F(s_1) \neq F(s_2)$, gdy $s_1 \neq s_2$ Na początku zamieńmy pojedyncze litery na liczby; każdą na numer jej pozycji w alfabecie (a na 1, b na 2, c na 3 itd.). Od teraz pisząc „litera", będę miał na myśli odpowiadającą jej wartość. Pierwszym pomysłem, na jaki możemy wpaść jest F sumujące litery w słowie. Zauważmy, że jest to beznadziejne rozwiązanie, które bardzo często nie spełnia drugiego założenia F. Na przykład dla słów `abcbbd` i `abcdbb` mamy:

$$F(\texttt{abcbbd}) = 1+2+3+2+2+4 = 14 = 1+2+3+4+2+2 = F(\texttt{abcdbb})$$

Wykorzystajmy fakt, że tak naprawdę obchodzi nas tylko, czy litery na tych samych pozycjach są równe. Spróbujmy zatem przemnożyć każdą z liter przez numer jej pozycji w słowie i zsumujmy uzyskane wartości. Brzmi sensowniej od wcześniejszego podejścia i faktycznie działa lepiej. Choćby dla poprzedniego przykładu:

$$F(\texttt{abcbbd}) = 1 \cdot 0 + 2 \cdot 1 + 3 \cdot 2 + 4 \cdot 1 + 5 \cdot 1 + 6 \cdot 3 = 35$$
$$F(\texttt{abcdbb}) = 1 \cdot 0 + 2 \cdot 1 + 3 \cdot 2 + 4 \cdot 3 + 5 \cdot 1 + 6 \cdot 1 = 31$$

więc $F(\texttt{abcbbd}) \neq F(\texttt{abcdbb})$ Niestety takie F dalej nie działa. Na przykład dla słów `cb` i `ac` mamy:

$$F(\texttt{cb}) = 1 \cdot 3 + 2 \cdot 2 = 7 = 1 \cdot 1 + 2 \cdot 3 = F(\texttt{ac})$$

Haszowanie

Niech p będzie małą liczbą pierwszą, większą od rozmiaru alfabetu. W przypadku alfabetu angielskiego możemy wybrać $p = 29$ lub $p = 31$. Zdefiniujmy nasze F jako:

$$F(S) = p^0 \cdot S_0 + p^1 \cdot S_1 + ... + p^{n-1} \cdot S_{n-1}$$

gdzieS_i oznacza literę na i-tej pozycji. Nie ulega wątpliwości fakt, że dla dwóch takich samych słów - argumentów wartości F będą takie same. Pozostaje pytanie co z warunkiem:

$$F(s_1) \neq F(s_2), \text{ gdy } s_1 \neq s_2$$

Oczywiście może zdarzyć się przypadek, że nie zostanie on spełniony. Jednakże jest to bardzo mało prawdopodobne. Pozwolę sobie pominąć dowód tego faktu. Musicie uwierzyć mi na słowo :) Tak policzone F nazwiemy funkcją haszującą, a całą metodę porównywania tekstów haszowaniem.

Kilka spraw praktycznych

1. Na olimpiadach jak i w prawdziwym życiu haszowanie jest wystarczającą i najczęściej używaną metodą porównywania tekstów. Jednakże mogą zdarzyć się pojedyncze przypadki zadań, w których autor układa złośliwe testy. W takiej sytuacji należy napisać dwie różne funkcje haszujące różniące się jedynie liczbą p i za każdym razem wykonywać te same operacje dwa razy. W ten sposób prawdopodobieństwo błędu jest tak niskie, że wygenerowanie testów uwalających jest praktycznie niemożliwe.

2. Funkcja F bardzo szybko rośnie i już przy niedługich słowach osiąga wartości powyżej zakresów `long long`'a. Zamiast trzymać wartość F, będziemy trzymać jej resztę z dzielenia przez liczbę pierwszą M rzędu miliarda (np. $10^9 + 696969$ czy $10^9 + 7$). Z tego powodu:

$$F(S) = (p^0 \cdot S_0 + p^1 \cdot S_1 + ... + p^{n-1} \cdot S_{n-1}) \bmod M$$

Działając na resztach nie możemy wykonywać operacji dzielenia, bo:

$$(a \bmod M) \div (b \bmod M) \neq (a \div b) \bmod M$$

oraz musimy uważać przy odejmowaniu. Nawet jeśli $b > a$ to $a \bmod M$ może być większe niż $b \bmod M$. Oznacza to, że ich różnica może być ujemna, czego byśmy nie chcieli. Dlatego do różnicy dwóch liczb zawsze należy dodać M przed wykonaniem operacji modulo:

$$(b-a) \bmod M = (b \bmod M - a \bmod M + M) \bmod M$$

3. Musimy umieć obliczać potęgi liczby p. W tym celu zdefiniujmy sobie tablicę Pot tak, że $Pot[i] = p^i$. Zauważmy, że:

$$p^0 = 1$$
$$p^i = p^{i-1} \cdot p, \text{ dla } i > 0$$

Wszystkie potrzebne wartości funkcji Pot możemy obliczyć w następujący sposób:

```
Pot[0] = 1;
for(int i = 1; i <= n;++i)
{
    Pot[i] = (Pot[i - 1] * p) % M;
}
```

Kilka ciekawych sztuczek

1. Często będzie potrzebna umiejętność porównywania dwóch podsłów. Zastanówmy się najpierw, jak porównać podsłowo S z podsłowem W zaczynającymi się na tej samej pozycji odpowiednich słów. W $tab[i]$ będziemy trzymać wartość funkcji haszującej dla i-tego prefiksu S. Analogicznie zdefiniujmy $tab1[i]$ dla W. Porównajmy sobie teraz podsłowa $[a;b]$. Zauważmy, że:

$$(tab[j] - tab[i-1] + M) \bmod M =$$
$$= (p^0 \cdot S_0 + p^1 \cdot S_1 + .. + p^{j-1} \cdot S_{j-1} - (p^i \cdot S_i + p^{i+1} \cdot S_{i+1} + \ldots + p^{j-1} \cdot S_{j-1} + M)) \bmod M =$$
$$= (p^i \cdot S_i + \ldots + p^{j-1} \cdot S_{j-1}) \bmod M =$$
$$= (p^i (p^0 \cdot S_i + p^1 \cdot S_{i+1} + \ldots + p^{j-i} \cdot S_j)) \bmod M = (p^i \cdot F(S[i..j])) \bmod M$$

Analogicznie:

$$(tab1[j] - tab1[i-1] + M) \bmod M = (p^i \cdot F(W[i..j])) \bmod M$$

Wiedząc, że podsłowa będą równe wtedy i tylko wtedy, gdy zachodzi warunek:

$$F(S[i..j]) = F(W[i..j])$$

wiemy, że wystarczy sprawdzić, czy:

$$(p^i \cdot F(S[i..j])) \bmod M = (p^i \cdot F(W[i..j])) \bmod M$$

czyli:

$$(tab[j] - tab[i-1] + M) \bmod M = (tab1[j] - tab1[i-1] + M) \bmod M$$

Jak poradzić sobie, gdy podsłowa zaczynają się na różnych pozycjach? Wiemy, że:

$$(tab[j] - tab[i-1] + M) \bmod M = (p^i \cdot F(W[i;j])) \bmod M.$$

Zatem, dla podsłowa $[i;j]S$ i $[k;l]W$:

$$(tab[j] - tab[i-1] + M) \bmod M = (p^i \cdot F(S[i;j])) \bmod M$$
$$(tab1[l] - tab1[k-1] + M) \bmod M = (p^k \cdot F(W[k;l])) \bmod M$$

Pomnóżmy pierwszą wartość przez p^k a drugą przez p^i. Otrzymamy wtedy:

$$(p^{i+k} \cdot F(S[i;j])) \bmod M$$
$$(p^{i+k} \cdot F(W[k;l])) \bmod M$$

Te wartości będą równe wtedy i tylko wtedy, gdy:

$$F(S[i;j]) = F(W[k;l])$$

czyli gdy nasze podsłowa są równe.

2. Bardzo przydatną umiejętnością jest też znajdowanie indeksu pierwszej litery, na której słowa się różnią. Możemy wykorzystać do tego wyszukiwanie binarne po tablicy tab. „Wstrzelmy się" w połowę długości słów i sprawdźmy, czy ich hasze są takie same. Jeżeli tak, to wszystkie literki na tym prefiksie są takie same. Oznacza to, że pierwsza różna pozycja znajduje się na prawo od naszego „strzału". W przeciwnym wypadku jest gdzieś na tym prefiksie