Funkcje skrótu (haszujące)

Bartosz Chomiński

1 Wstęp

Rozwiążemy w tej notatce pewne klasyczne zadanie: mamy listę długich słów i chcemy umieć odpowiadać na zapytania postaci „czy słowo o numerze i jest równe słowu o numerze j?", a następnie jego uogólnienie.

Na początek rozwiązanie oczywiste – każde takie zapytanie traktujemy oddzielnie i sprawdzamy kolejne litery obu porównywanych słów aż nie dotrzemy do pozycji, na której słowa się różnią lub do końca któregoś ze słów. To rozwiązanie może okazać się zbyt wolne dla dużych danych, więc potrzebujemy szybszego sposobu.

2 Biblioteka

Przeformułujmy nieco oryginalne zadanie. Teraz rozważmy półkę książek w bibliotece i zapytania postaci „czy książka o numerze i ma taką samą treść jak książka o numerze j?". Merytorycznie treść zadania się nie zmieniła, ale stało się ono znacznie prostsze.

Oczywiście w każdym zapytaniu wystarczy porównać okładki rozpatrywanych książek i prawdopodobnie uzyskamy w ten sposób zgodną z prawdą odpowiedź. Na pewno, natomiast, nikt nie będzie czytał obu książek literka po literce do momentu wystąpienia rozbieżności.

Zastanówmy się, dlaczego w zadaniu z biblioteką było prościej. Całe ułatwienie wynika stąd, że przypisanie okładek książkom spełnia pewne dwa warunki. Dla odrobiny formalizmu oznaczmy przez h abstrakcyjną funkcję, która przyjmuje książkę i zwraca jej okładkę. Te dwa warunki, dzięki którym zadanie z biblioteką jest prostsze można zapisać następująco, przyjmując, że x i y to dowolne książki:

$$h(x) \neq h(y) \implies x \neq y \tag{1}$$

$$h(x) = h(y) \implies x = y \text{ prawie na pewno} \tag{2}$$

Pierwszy warunek pozwala nam wykluczać równość książek tylko na podstawie różnych okładek tych książek, a drugi warunek ogranicza liczbę fałszywie pozytywnych równości (czyli sytuacji, w których dwie książki są różne, ale mają takie same okładki).

Spróbujmy przenieść tę użyteczną własność funkcji h na oryginalne zadanie ze słowami. Przed chwilą funkcję h dostarczali nam wydawcy książek, a teraz, w ogólnym przypadku, musimy ją sobie wymyślić sami.

3 Odpowiednia funkcja h

Zdefiniujmy problem, przed którym stoimy. Potrzebujemy zdefiniować funkcję h, która:

- przyjmuje słowo dowolnej długości,
- zwraca obiekty, które można szybko porównywać (np. liczby typu `long long`),
- daje się szybko obliczać,
- dla różnych słów często daje różne wyniki.

3.1 Antyprzykłady

Rozpatrzmy na początek kilka antyprzykładów, żeby wyrobić sobie intuicję co do działania funkcji h.

Funkcja $h(s) =$ długość słowa s spełnia pierwsze trzy warunki, natomiast całkiem często daje te same wyniki dla różnych słów – np. $h(\texttt{a}) = 1$ i $h(\texttt{b}) = 1$.

Funkcja $h(s) = s[0]$ również spełnia pierwsze trzy warunki, natomiast wystarczy odrobinę dłuższy przykład, niż wyżej, by obalić zachodzenie czwartego warunku: $h(\texttt{aa}) = \texttt{a}$ i $h(\texttt{ab}) = \texttt{a}$.

Funkcja $h(s) = s$ spełnia pierwszy, trzeci i czwarty warunek, natomiast porównywanie jej wyników zawraca nas do problemu, który właśnie chcemy rozwiązać.

3.2 System pozycyjny

Ponownie lekko zmieńmy treść zadania, które chcemy rozwiązać. Roboczo załóżmy, że słowa, które dostaje funkcja h składają się wyłącznie z cyfr systemu

dziesiętnego i nie mają zer wiodących. Wtedy dość dobrym kandydatem byłaby funkcja

$$h(s) = (\text{liczba, której zapis dziesiętny to } s) \mod p,$$

gdzie p to duża liczba pierwsza. Oczywiście tu też bardzo łatwo można znaleźć kolizję, bo np. dla $p = 10^9 + 7$ mielibyśmy $h(\texttt{1000000008}) = 1$ i $h(\texttt{1}) = 1$, natomiast częstość kolizji dla losowych danych i możliwość omijania kolizji przez wielokrotne losowanie p sprawia, że ta funkcja h jest o wiele lepszym kandydatem, niż poprzednie antyprzykłady.

Zauważmy jeszcze, że wzór na funkcję h wygląda następująco[1]:

$$h(a_0a_1a_2\ldots a_n) = (a_0 \cdot 10^n + \cdots + a_n \cdot 10^0) \mod p.$$

3.3 Uogólnienie

Powróćmy teraz do oryginalnego zadania tego rozdziału, a więc dopuszczamy dowolne słowa, w których mogą być dowolne litery. Gdyby tych możliwych liter było 10, to każdej z nich moglibyśmy jednoznacznie przypisać cyfrę i skorzystać z funkcji h uzyskanej nieco wyżej. Jeśli natomiast byłoby ich więcej, niż 10 (i tak zwykle jest), to moglibyśmy zwiększyć podstawę systemu liczbowego z 10 do np. 31 i przyporządkować literom wartości od 1 do 26 (jako cyfry systemu o podstawie 31).

Wobec tego nasza uogólniona funkcja h mogłaby działać następująco:

$$h(a_0a_1a_2\ldots a_n) = ([a_0]\cdot 31^n + \cdots + [a_n]\cdot 31^0) \mod p,$$

gdzie [c] to numer (od 1 do 26) litery `c`.

Przykładowo, dla słowa $s = \texttt{klops}$ i $p = 1\,000\,003$ funkcję h obliczylibyśmy następująco:

$$\begin{aligned} &h(\texttt{klops}) = \\ &= ([\texttt{k}] \cdot 31^4 + [\texttt{l}] \cdot 31^3 + [\texttt{o}] \cdot 31^2 + [\texttt{p}] \cdot 31 + [\texttt{s}]) \mod p \\ &= (11 \cdot 31^4 + 12 \cdot 31^3 + 15 \cdot 31^2 + 16 \cdot 31 + 19) \mod p \\ &= 10\,531\,153 \mod 1\,000\,003 = 531\,123. \end{aligned}$$

3.4 Obliczanie

By efektywnie obliczać funkcję h, wystarczy wyznaczyć na początek tablicę reszt z dzielenia kolejnych potęg liczby 31 przez ustalone p i następnie korzystać z tych wartości w definicji funkcji h, a można też obliczać funkcję h iteracyjnie od 0, domnażając aktualną wartość przez 31 i dodając kod aktualnie rozpatrywanej litery.

4 Podsłowa

Przyszedł czas na uogólnienie oryginalnego zadania ze wstępu tej notatki. Tym razem zadanie brzmi tak: mamy dużo długich słów i chcemy umieć odpowiadać na zapytania postaci „czy podsłowo $[a, b]$ słowa o numerze i jest równe podsłowu $[c, d]$ słowa o numerze j?”.

Tym razem nie możemy sobie pozwolić na obliczenie wartości funkcji h dla wszystkich sensownych słów, bo byłoby ich za dużo – słowo długości N ma $\mathcal{O}(N^2)$ podsłów. Wobec tego oczywiście musimy się ograniczyć do obliczania funkcji h tylko na podsłowach występujących w zapytaniach, ale oczywisty sposób obliczania funkcji h kosztuje nas czas proporcjonalny do liczby liter argumentu, a więc może się okazać, że nadal kosztuje to zbyt dużo czasu.

Wyprowadzimy sposób obliczania funkcji h dla podsłów ustalonego słowa s o długości N w czasie $\mathcal{O}(N)$ (preprocessing) i $\mathcal{O}(1)$ (na zapytanie).

4.1 Sumy prefiksowe

Załóżmy na chwilę, z przyczyn, które za niedługo staną się oczywiste, że funkcja h po prostu dodaje do siebie kody liter – bez mnożenia przez potęgi ustalonej podstawy. Wtedy funkcję h dla dowolnego podsłowa moglibyśmy liczyć identycznie jak sumy prefiksowe, czyli wyznaczamy wartości $T_0 = h(a_0), T_1 = h(a_0a_1), T_2 = h(a_0a_1a_2)$ i zauważamy, że

$$h(a_ia_{i+1}\ldots a_j) = h(a_0\ldots a_j) - h(a_0\ldots a_{i-1}) = T_j - T_{i-1}.$$

[1] W tym wzorze występuje pewne nadużycie notacyjne, które Odpowiednio Świadomy Formalnie Czytelnik z pewnością zauważy – a_i po lewej stronie równości są cyframi, natomiast po prawej stronie są liczbami oznaczającymi te cyfry. Autor posiada jednak nadzieję, że ten wybryk notacyjny nie wytworzy w Czytelnikach zbyt dużych wątpliwości.

4.2 Domnażanie potęg

Teraz zajmijmy się potęgami, które są domnażane do kodów liter w funkcji h, której faktycznie chcemy używać. Podobnie jak wyżej zdefiniujmy

$$T_i = h(a_0 a_1 \dots a_i) = (a_0 \cdot 31^i + \dots + a_i \cdot 31^0) \mod p.$$

Mamy dalej

$$\begin{aligned} & h(a_i \dots a_j) = \\ & = (a_i \cdot 31^{j-i} + \dots + a_j \cdot 31^0) \mod p \\ & = ((a_0 \cdot 31^j + \dots + a_j \cdot 31^0) - \\ & \quad - (a_0 \cdot 31^j + \dots + a_{i-1} \cdot 31^{j-i+1})) \mod p \\ & = (T_j - 31^{j-i+1} \cdot (a_0 \cdot 31^{i-1} + \dots + a_{i-1})) \mod p \\ & = (T_j - 31^{j-i+1} T_{i-1}) \mod p. \end{aligned}$$

Wobec tego mamy jawny wzór na wyznaczenie funkcji h dla podsłów, mając wyłącznie wartości funkcji h dla kolejnych prefiksów całego słowa.

5 Porady praktyczne

W zastosowaniu na zawodach warto używać funkcji haszujących z p rzędu miliarda, jak również bez podanego wprost modulo – wówczas, przy zastosowaniu typu `long long`, efektywnie wszystkie wartości będą liczone modulo 2^{64}. Zagrożenia związane z takim postępowaniem zostały dobrze opisane w wymienionym dalej artykule J. Pachockiego i J. Radoszewskiego.

Warto również stosować tzw. *haszowanie podwójne*, a więc takie funkcje h, których wynikiem są pary składające się z rezultatów opisanego wyżej wielomianowego haszowania dla różnych p.

Należy uważać przy gęstym porównywaniu między sobą elementów dużego zbioru słów. Jeśli zbiór liczy N obiektów i funkcja haszująca przyjmuje co najwyżej U wartości, to gdy N będzie rzędu $\sqrt{U}$, prawdopodobnie zaczną pojawiać się kolizje. Szczegółowe informacje można uzyskać czytając o *paradoksie dnia urodzin*.

6 Zobacz też

- `https://pl.wikipedia.org/wiki/Funkcja_skr%C3%B3tu`
- `https://pl.wikipedia.org/wiki/Paradoks_dnia_urodzin`
- J. Pachocki, J. Radoszewski, Where to Use and How not to Use Polynomial String Hashing. `https://ioinformatics.org/journal/INFOL119.pdf`
- `https://cp-algorithms.com/string/string-hashing.html`