第 5 章

Rolling Hash のおはなし

76 回生 maguro

5.1 はじめに

こんにちは。76 回生 (高 1) の maguro です。中学生活はあっという間で、すぐに高校生になってしまいました。

最近は専ら競技プログラミングをしています (Capture The Flag もちょっとだけやっています)。

部誌に書く内容に迷っていたら、ちょうど実生活でも活かせそうな Rolling Hash というアルゴリズムを勉強したのでそのことについて書きたいと思います。

前提の知識として、計算量という概念を扱います。詳しくは Wikipedia の「ランダウの記号」のページをご覧ください。また、少し数学 (数Ⅰ・A 程度) の知識があると読みやすいと思います。

5.2 導入

突然ですが、次の問題を考えてみましょう。

文字列の完全一致検索問題

N 文字の文字列 S が与えられます。

今、太郎君は長さ A の文字列 T が S の中に含まれているかを知りたいです。 T が S に含まれていないなら -1 を、含まれているならば文字列が S の先頭から何番目から始まっているかを全て、昇順に返すプログラムを作成してください。

- $1 \leq A \leq N \leq 10^6$
- S, T は英小文字からなる。

ひとつの方法としては、それぞれの検索において S の 1 文字目から A 文字目までが T と一致する、2 文字目から $A+1$ 文字目までが T と一致する、3 文字目から $A+2$ 文字目までが T と一致する……という風に愚直に調べる方法があります。

しかし、これだと計算量が $O(NA)$ となってしまい、 N や A が 100 万のような大きい値になるとかなり時間が掛かってしまいます。そこで Rolling Hash の出番です。

5.3 Rolling Hash ってなに？

Rolling Hash とは、「**文字列を 1 つの大きな数字としてみる**」というアイデアを用いて、文字列検索を高速に行うアルゴリズムです。

ここでの「1 つの大きな数字」が**ハッシュ**と言われるものです。ハッシュの値をハッシュ値と言います。

ただ、いきなり文字列を数字として扱うのは難しいのでもう少し簡単にした問題を考えてみましょう。

数字の一致問題

1 以上 9 以下の整数からなる長さ N の文字列 S が与えられます。
S の空でない連続する部分列のうち、 10 進法の整数とみなしたときに桁数が A の正の整数 T と等しくなるものは存在するでしょうか？ もし存在しないなら -1 を、存在するならばその文字列が S の先頭から何番目から始まっているかを全て、昇順に返すプログラムを作成してください。

- $1 \leq A \leq N \leq 10^6$
- S, B は 1 から 9 までの数字からなる

ここで、 S の連続する部分列 S' に対して、 S' のハッシュ値を **S' を 10 進法の整数とみなしたときの値**と定義します。ハッシュ値が等しい 2 つの文字列は当然等しいです。よって、ハッシュ値を高速に計算することが出来たらこの問題は解けそうです。

では、どうやってハッシュ値を高速に計算するのでしょうか？

5.4　ハッシュ値の計算

上の問題で $S = 123456789$ である場合を考えます。

また、 S の a 文字目から b 文字目の前までのハッシュ値を $hash[a][b]$ と書くことにします[*1]。例えば $hash[0][3] = 123$ 、 $hash[1][4] = 234$ です。

ここで、 $hash[0][6] = 123456$ 、 $hash[0][2] = 12$ から、 $hash[2][6] = hash[0][6] - hash[0][4] \times 10^4$ であることが分かります。これは一般に拡張することができ、 $hash[a][b] = hash[0][b] - hash[0][a] \times 10^{b-a}$ が成り立ちます（証明は読者に任せます)。この式に出てくる 10 のことをハッシュの法と言うことにします。

また、 $hash[0][i]$ $(0 \leq i \leq N)$ は、 S の j 文字目を S_j とおくと、 $hash[0][0] = 0, hash[0][i] = hash[0][i-1] \times 10 + S_{i-1}$ $(1 \leq i \leq N)$ を用いると $O(N)$ で求めることができます[*2]。

よって、 $hash[0][0], hash[0][1], hash[0][2], \cdots, hash[0][N]$ 、また $10^0, 10^1, 10^2, \cdots, 10^N$ を事前に求めることで任意の a, b $(0 \leq a \leq b \leq N)$ について $hash[a][b]$ を $O(1)$ で求めることができ、元の問題は全ての連続する部分列についてハッシュ値を求めることで $O(N)$ で求めることができました。

文字列の完全一致検索問題に戻ってみましょう。この問題は、数字の一致問題で出てきた S が数字から英小文字になっています。ここで、数字の一致問題の解法を 10 進法から 27 進法に変えるとどうでしょうか。仮に、$a = 1, b = 2, \cdots, z = 26$ として、ハッシュの法を 27 にすると、解法はほとんど一緒であることが分かります。

5.5　ハッシュの衝突

ここまででハッシュはそのまま扱ってきましたが、このままだとハッシュの最大値が $27^N - 1 \fallingdotseq 10^{1.4 \times 10^5}$ になり、いくらコンピューターでもこのような量の数字は扱えません。そこで、多くの場合ハッシュはある整数 m で割った余りにすることが多いです（この m のことを mod と呼ぶことにします)。

しかし、ハッシュをこのようにしたときにある問題が発生します。それが **ハッシュの衝突**です。

ハッシュの衝突とは、本来は異なっている文字列が同じハッシュを取ることです。当然ですがこのような事態は

[*1] 現実の世界では物を 1 から数えることがほとんどですが、プログラミングの世界では 0 から数えることが多いです。ここでは文字列の前から何番目を数えるときに 0 から数えることにします。

[*2] この操作が Rolling Hash の Rolling に当たる部分です。

なるべく避けられないといけません。

そこで、なるべく衝突を回避し、かつ高速に動作するために以下のような工夫をしています。

0 が割り当てられるような文字を作らない

例えば、 $a = 0$ としたときに、 ba と $baaa$ は同じハッシュを取ってしまいます。これは必ず避けられないといけません。

m をなるべく大きくする

ハッシュは全て m で割った余りになるので、必ず 1 以上 m 未満になります。よって、 m 個以上のハッシュを生成した場合、必ずどこかのハッシュ同士が衝突してしまいます。そのため、 m はなるべく大きくする必要があります。

また、ハッシュが 1 以上 m 未満のランダムな値を取るとき、ハッシュが衝突する確率は想像以上に高いです。

誕生日攻撃 - Wikipedia にあるように、 32 ビット（すなわち $2^{32} - 1 \fallingdotseq 4.3 \times 10^9$）の mod を用いて Rolling Hash をすると、 10^5 個のハッシュを生成するだけでハッシュが衝突する確率が 75% 程度あることが分かります。

mod を 64 ビット（すなわち $2^{64} - 1 \fallingdotseq 1.84 \times 10^{19}$）の mod を用いて Rolling Hash をしても、 7.2×10^9 個のハッシュを生成すると衝突する確率が 75% 程度ありますが、これ以上 mod を増やすと実装がややこしくなったり実行時間が長くなったりする可能性があります。なので後ろの方に掲載している実装は 64 ビットの mod を用いています。心配なら 128 ビットの mod を用いると良いでしょう。

ハッシュの法をランダムな値にする

Codeforces などのハック[*3]が存在する競技プログラミングのコンテストでは、いくら mod を大きくしたとしても、悪意のあるユーザーがわざとハッシュを衝突させようとする可能性があります。そこで、これを回避するために、ハッシュの法を実行時ランダムな値にするという方法があります（mod をランダムな値にする方法もありますが、割り算をするときに mod が定数だと実行時間が短くなります）。これで、いくら悪意のあるユーザーでもハッシュの衝突を意図的に起こすのは難しくなります。

mod を 2 のべき乗にしない

先ほど「mod を乱数にするといくら悪意のあるユーザーでもハッシュの衝突を意図的に起こすのは難しくなる」という話をしましたが、 mod が 2 のべき乗の時は話が別です。このとき、ハッシュの法を乱数にしていても、意図的に非常に高い確率でハッシュを衝突させることが出来ます。絶対にやめましょう。

5.6 最後に

Rolling Hash の話をしてきましたが如何でしたでしょうか。ハッシュの衝突を回避するにあたって気を付けないといけないことはまだあるのですが、ここでは割愛します。

最後に自分の書いた実装（C++）を掲載しておきます。競技プログラミング特有の書き方（using namespace std; 等）があるのは大目に見てください。

[*3] 他人のユーザーが提出しているコードで、間違っているものを見つけた時に、正しくない解答をそのコードが出力するような入力を与えることで、自分の点数を得るシステム。