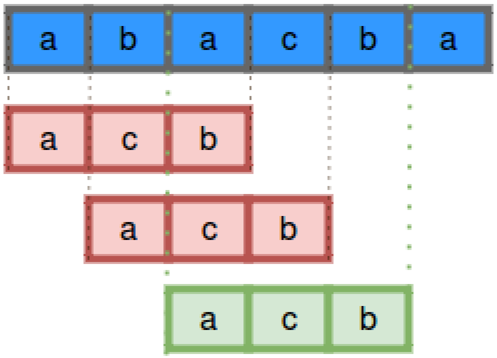
রবিন-কার্প স্ট্রিং ম্যাচিং

ডিসেম্বর ১, ২০১৬ *by* *শাফায়েত*

রবিন-কার্প (Rabin-carp) একটি স্ট্রিং ম্যাচিং অ্যালগোরিদম। দুটি স্ট্রিং দেয়া থাকলে এই অ্যালগোরিদমটি বলে দিতে পারে যে ২য় স্ট্রিংটি প্রথম স্ট্রিং এর সাবস্ট্রিং কিনা। রবিন-কার্প রোলিং হ্যাশ টেকনিক ব্যবহার করে স্ট্রিং ম্যাচিং করে। যদিও স্ট্রিং ম্যাচিং এর জন্য কেএমপি অ্যালগোরিদম ব্যবহার করাই ভালো, কিন্তু রবিন-কার্প শেখা গুরুত্বপূর্ণ মূলত রোলিং হ্যাশ কিভাবে কাজ করে সেটা শেখার জন্য।

এই লেখাটা পড়ার আগে মডুলার অ্যারিথমেটিক সম্পর্কে জেনে আসতে হবে।
স্ট্রিং ম্যাচিং করার সময় প্রথম স্ট্রিং টাকে আমরা বলবো টেক্সট (Text) এবং দ্বিতীয়টিকে প্যাটার্ন (Pattern)। আমাদের কাজ হলো টেক্সট এর মধ্যে প্যাটার্ন খুজে বের করা।

প্রথমে আমরা একটা ব্রুটফোর্স অ্যালগোরিদমের কথা ভাবি। আমরা টেক্সট এর প্রতিটা সাবস্ট্রিং বের করে প্যাটার্নের সাথে মিলিয়ে দেখতে পারি:



ছবিতে abacbaabacba টেক্সট এর ভিতর acbacb প্যাটার্নটা খোজা হচ্ছে।

```
function naive_matching(text, pattern){
  n = text.size()
```

```
  m = pattern.size()
  for(i = 0; i < n; i++) {
    for(j = 0; j < m && i + j < n; j++) {
      if(text[i + j] != pattern[j]) {
        break; // mismatch found, break the inner loop
      }
    }
    if(j == m) {
     // match found
    }
  }
}
```

নেইভ স্ট্রিং ম্যাচিং অ্যালগোরিদমের কমপ্লেক্সিটি $O(n*m)O(n*m)$, যেখানে nn হলো টেক্সট এর দৈর্ঘ্য এবং mm হলো প্যাটার্ন এর দৈর্ঘ্য।

আমাদের যদি দুটি স্ট্রিং তুলনা করার সময় একটা একটা ক্যারেক্টার না দেখে ইন্টিজারের মতো $O(1)O(1)$ এ তুলনা করতে পারতাম তাহলে আমরা খুব দ্রুত স্ট্রিং ম্যাচিং করতে পারতাম। হ্যাশিং টেকনিক ব্যবহার করে আমরা স্ট্রিং কে ইন্টিজারে পরিণত করতে পারি। রবিন-কার্প অ্যালগোরিদম এ সেটারই সুবিধা নেয়া হয়েছে।

আমরা যেকোন স্ট্রিংকে একটা $Base\text{-}BBase\text{-}B$ সংখ্যা হিসাবে কল্পনা করতে পারি যেখানে BB এর মান অ্যাসকিতে যতগুলো ক্যারেক্টার আছে তার সমান বা বড়। তাহলে আমরা একটা ফাংশন লিখতে পারি যেটা ss কে $Base\text{-}BBase\text{-}B$ সংখ্যা থেকে $Base\text{-}10Base\text{-}10$ সংখ্যায় রূপান্তর করবে। এটাই হতে পারে আমাদের হ্যাশ ফাংশন:

$$Hash(s)=s_0\cdot B^{m-1}+s_1\cdot B^{m-2}+\ldots+s_{m-1}\cdot B^1+s_{m-1}\cdot B^0 Hash(s)=s0\cdot Bm-1+s1\cdot Bm-2+\ldots+sm-1\cdot B1+sm-1\cdot B0$$

এই ফাংশনে প্যারামিটার হিসাবে একটা স্ট্রিং পাঠানো হয়েছে। $s_i si$ দিয়ে বুঝানো হয়েছে ii তম ক্যারেক্টারের অ্যাসকি ভ্যালু। এই হ্যাশ ফাংশন দিয়ে যেকোনো স্ট্রিং এর জন্য ভিন্ন ভিন্ন হ্যাশভ্যালু পাওয়া যাবে। কিন্তু সমস্যা হলো ওভারফ্লো, হ্যাশভ্যালুর মান সহজেই ৬৪-বিট এর বড় হয়ে যাবে। এই জন্য আমাদেরকে হ্যাশ ভ্যালুটাকে MM দিয়ে ভাগ করে ভাগশেষ (modulo) নিতে হবে। তাহলেই সংখ্যাটা MM এর থেকে ছোটো হয়ে যাবে:

$$Hash(s)=(s_0\cdot B^{m-1}+s_1\cdot B^{m-2}+\ldots+s_{m-1}\cdot B^1+s_{m-1}\cdot B^0) \bmod M Hash(s)=(s0\cdot Bm-1+s1\cdot Bm-2+\ldots+sm-1\cdot B1+sm-1\cdot B0) \bmod M$$

কিন্তু এইক্ষেত্রে সমস্যা হলো একাধিক স্ট্রিং এর হ্যাশভ্যালু একই হয়ে যেতে পারে। এই সমস্যাটাকে বলা হয় হ্যাশ কলিশন (hash collision)। তবে BB এবং MM যদি প্রাইম সংখ্যা হয় এবং MM এর মান অনেক বড় হয় তাহলে কলিশন করার সম্ভাবনা খুব কমে যায়। (তবে সম্ভাবনা কমে গেলেও একদম শুণ্য হয়ে যায় না, যে জন্য রবিন কার্পেরও worse case complexity $O(n*m)O(n*m)$, সে কথায় পরে আসছি)

এখন প্রথমেই আমাদের কাজ হবে প্যাটার্নের হ্যাশ ভ্যালু বের করা। এরপর টেক্সট এর প্রতিটা mm দৈর্ঘ্যের সাবস্ট্রিং এর জন্য হ্যাশভ্যালু বের করে $Hash(pattern)Hash(pattern)$ এর সাথে মিলিয়ে দেখতে হবে। এখন প্রশ্ন হলো প্রতিটা mm দৈর্ঘ্যের সাবস্ট্রিং এর জন্য হ্যাশভ্যালু কিভাবে বের করবো? যদি প্রতিটা সাবস্ট্রিং কে তুমি উপরের হ্যাশ ফাংশনে পাঠাও তাহলে কমপ্লেক্সিটি হয়ে যাবে $O(n*m)O(n*m)$। আমাদেরকে একটা পদ্ধতি বের করতে হবে যেন স্ট্রিং এর উপর শুধু একটা লুপ চালিয়েই $O(n)O(n)$ এ প্রতিটা mm দৈর্ঘ্যের সাবস্ট্রিং এর জন্য হ্যাশভ্যালু বের করা যায়। এখানেই রোলিং হ্যাশ পদ্ধতি কাজে লাগবে।

মনে করো $H_i Hi$ হলো ss এর ii তম ইনডেক্সের শুরু হয়েছে এমন mm দৈর্ঘ্যের স্ট্রিং এর হ্যাশ ভ্যালু। তাহলে আমরা লিখতে পারি:

$H_i=s_i\cdot B_{m-1}+s_{i+1}\cdot B_{m-2}+\ldots+s_{i+m-1}\cdot B_0$Hi=si·Bm−1+si+1·Bm−2+…+si+m−1·B0

এখন যদি $m=3$m=3 হয় তাহলে H_0H0 এবং H_1H1 কে লিখতে পারি:

$H_o=s_o\cdot B_2+s_1\cdot B_1+s_2$Ho=so·B2+s1·B1+s2

$H_1=s_1\cdot B_2+s_2\cdot B_1+s_3$H1=s1·B2+s2·B1+s3

এখন দেখো H_1H1 কে কিভাবে H_0H0 এর মাধ্যমে প্রকাশ করা যায়:

$H_1=((s_o\cdot B_2+s_1\cdot B_1+s_2)–(s_o\cdot B_2))\times B+s_3$H1=((so·B2+s1·B1+s2)–(so·B2))×B+s3

$H_1=(H_0–S_o\ast B_2)\cdot B+s_3$H1=(H0–So∗B2)·B+s3

সাধারণভাবে বলা যায়:

$H_i=(H_{i-1}–S_{i-1}\cdot B_{m-1})\cdot B+s_{i+m-1}$Hi=(Hi−1–Si−1·Bm−1)·B+si+m−1

এখন এই সূত্র ব্যবহার করে খুব সহজেই $O(n)$O(n) এ প্রতিটা mm দৈর্ঘ্যের সাবস্ট্রিং এর হ্যাশ ভ্যালু বের করা যাবে। শুরুতে প্রথম mm ক্যারেক্টারের জন্য হ্যাশভ্যালু বের করে নিয়ে এরপর রোলিং হ্যাশ পদ্ধতিতে বাকিগুলো বের যাবে।

রবিন-কার্পের একটা সি++ কোড দেখি:

```
//Implementation of Rabin Carp String Matching Algorithm
//https://github.com/Shafaet/Programming-Contest-Algorithms/blob/master/Useful%20C%2B%2B%20Libraries/rabin-
carp.cpp
#include <bits/stdc++.h>
using namespace std;

typedef long long i64;

//Returns the index of the first match
//Complexity O(n+m), this is unsafe because it doesn't check for collisons

i64 Hash(const string &s, int m, i64 B, i64 M){
    i64 h = 0 , power = 1;
    for(int i = m-1;i>=0;i--){
        h = h + (s[i] * power) % M;
        h = h % M;
        power = (power * B)%M;
    }
    return h;
}
int match(const string &text, const string &pattern) {
    int n = text.size();
    int m = pattern.size();
    if(n < m)return -1;
    if(m == 0 or n == 0)
        return -1;

    i64 B = 347, M = 1000000000+7;

    //Calculate B^(m-1)
    i64 power = 1;
    for(int i=1;i<=m-1;i++)
        power = (power * B) % M;

```

```
    //Find hash value of first m characters of text
    //Find hash value of pattern
    i64 hash_text = Hash(text, m, B, M);
    i64 hash_pattern = Hash(pattern, m, B, M);

    if(hash_text == hash_pattern){  //returns the index of the match
        return 0;
        //We should've checked the substrings character by character here as hash collision might happen
    }

    for(int i=m;i<n;i++){

        //Update Rolling Hash
        hash_text = (hash_text - (power * text[i-m]) % M) % M;
        hash_text = (hash_text + M) % M; //take care of M of negative value
        hash_text = (hash_text * B) % M;
        hash_text = (hash_text + text[i]) % M;

        if(hash_text==hash_pattern){
            return i - m + 1; //returns the index of the match
             //We should've checked the substrings character by character here as hash collision might happen
        }
    }
    return -1;
}


int main() {
    cout<<match("HelloWorld", "ello")<<endl;
    return 0;
}
```

এই কোডের কমপ্লেক্সিটি O(n+m)O(n+m)। কিন্তু এখানে একটা বড় সমস্যা আছে। যখন hash_text আর hash_pattern মিলে যাচ্ছে তখন আমরা ধরে নিচ্ছি যে স্ট্রিং দুটি মিলে গিয়েছে। কিন্তু আগে বলেছিলাম যে ভাগশেষ(mod) নেয়ার কারণে একাধিক স্ট্রিং এর একই হ্যাশভ্যালু আসতে পারে (কলিশন হতে পারে)। সে ক্ষেত্রে এই কোড ভুল আউটপুট দিবে।

কলিশনের কারণে ভুল এড়ানোর একটা উপায় হলো, যখনই hash_text = hash_pattern হবে তখন আবার লুপ চালিয়ে ক্যারেক্টার বাই ক্যারেক্টার মিলিয়ে দেখা। কিন্তু সেক্ষেত্রে worse case কমপ্লেক্সিটি O(n∗m)O(n∗m) হয়ে যাচ্ছে যেটা ব্রুট ফোর্সের কমপ্লেক্সিটির সমান।

আরেকটা উপায় হলো ডাবল হ্যাশিং করা। ডাবল হ্যাশিং মানে হলো ভিন্ন ভিন্ন B এবং M ব্যবহার করে প্রতিটা স্ট্রিং এর জন্য দুটি করে হ্যাশ ভ্যালু বের করবো। সেক্ষেত্রে দুই জায়গাতেই কলিশনের সম্ভাবনা খুবই কমে যাবে। চাইলে ২বারের বেশিও হ্যাশিং করা যায়। ডাবল হ্যাশিং করলে প্রোগ্রামিং কনটেস্টে বেশিভাগ সময় পার পেয়ে যাওয়া যাবে, কিন্তু বাস্তব ক্ষেত্রে এটা খুব একটা ভালো উপায় না, কারণ কলিশনের সম্ভাবনা কমে গেলেও একদম শূন্য হয়ে যাচ্ছে না।

এসব কারণে রবিন-কার্প স্ট্রিং ম্যাচিং এর জন্য তেমন একটা ব্যবহার করা হয় না, এর থেকে কেএমপি ব্যবহার করা সুবিধাজনক। কিন্তু রোলিং হ্যাশ টেকনিক অনেক ধরণের সমস্যা সমাধান করতে কাজে লাগে যে কারণে আমরা রবিন-কার্প শিখি।

চিন্তা করার জন্য সমস্যা:

তোমাকে দুটি স্ট্রিং s1,s2s1,s2 দেয়া আছে। তোমাকে এদের লংগেস্ট কমন সাবস্ট্রিং এর দৈর্ঘ্য বের করতে হবে। অর্থাৎ সবথেকে বড় স্ট্রিং বের করতে হবে যেটা দুটি স্ট্রিং এরই সাবস্ট্রিং।

সমাধান:

প্রথমে চিন্তা করো যেকোনো ইন্টিজার kk এর জন্য কি তুমি বের করতে পারবে যে kk দৈর্ঘ্যের কোনো লংগেস্ট কমন সাবস্ট্রিং আছে নাকি। এটা সহজেই O(n)O(n) কমপ্লেক্সিটিতে বের করা যাবে রোলিং হ্যাশ ব্যবহার করে। s1s1 এর প্রতিটা kk দৈর্ঘ্যের সাবস্ট্রিং এর হ্যাশভ্যালু বের করো, এরপর s2s2 এর প্রতিটা kk দৈর্ঘ্যের সাবস্ট্রিং এর হ্যাশভ্যালু বের করো। যদি কোনো হ্যাশভ্যালু কমন থাকে তারমানে kk দৈর্ঘ্যের কোনো লংগেস্ট কমন সাবস্ট্রিং আছে।

এখন লংগেস্ট কমন সাবস্ট্রিং এর সর্বোচ্চ দৈর্ঘ্য হতে পারে $maxlen=min(s_1.length,s_2.length)$maxlen=min(s1.length,s2.length)। এখন তুমি 00 থেকে maxlenmaxlen এর উপর বাইনারি সার্চ চালিয়ে সহজেই সমস্যাটা সমাধান করতে পারবে। কমপ্লেক্সিটি হবে O(n·logn)O(n·logn)।

হ্যাপি কোডিং!

রেফারেন্স: টপকোডার