ساختمان داده‌ها (۲۲۸۲۲)

مدرس: حسین بومری

[**بهار** 1400]

جلسه ۲۲: جست‌وجوی رشته

نگارنده: فاطمه سادات موسوی

در جلسات گذشته درباره‌ی پیمایش درخت‌های دودویی، پیمایش درخت‌های چندانشعابی، پیش‌ترتیب، پس‌ترتیب و میان‌ترتیب و ویژگی‌های آن‌ها صحبت کردیم. در این جلسه درباره‌ی جست‌وجوی رشته صحبت می‌کنیم و الگوریتم‌هایی را در این زمینه بیان می‌کنیم.

۱ جست‌وجوی رشته

در زمینه‌ی پردازش رشته، یکی از مسائل مهمی که مطرح می‌شود، پیدا کردن یک رشته‌ی الگو $pattern$ در یک رشته‌ی دیگر $text$ است.

الگوریتم اول:

فرض کنید دو رشته‌ی $pattern = cde$ و $text = abcdefgh$ به ما داده‌شده است. از ابتدای $text$ شروع به حرکت می‌کنیم تا حرف اول $pattern$ را پیدا کنیم. اگر حرف بعدی هم مشابه بود، $counter$ را زیاد می‌کنیم در غیر این صورت $counter$ را برابر با صفر می‌کنیم. اگر به اندازه‌ی طول $pattern$ کاراکترهای یکسانی در $text$ و $pattern$ مشاهده شد، به عبارت دیگر $counter$ به اندازه‌ی طول $pattern$ افزایش یافت، "$found$" را چاپ می‌کنیم.

شبه کد:

```
cnt= 0
for i: 0 -> len(text)
        if pat(cnt)== text(i) : cnt++
        else : cnt=0
        if (cnt== len(pat))
            print("found")
```

مرتبه‌ی زمانی: از آن‌جایی که یک حلقه داریم که هر حرف $text$ را یک بار پیمایش می‌کند، مرتبه‌ی زمانی برابر با $O(n)$ خواهد بود که n طول $text$ است.

اشکال: الگوریتمی که در بالا بیان کردیم برای همه‌ی رشته‌ها درست کار نمی‌کند. در واقع ممکن است $pattern$ در $text$ موجود باشد ولی به ما جواب درست ندهد. برای درک بهتر این موضوع مثال زیر را در نظر بگیرید:

فرض کنید $pattern = cdcda$ و $text = acdcdcda$ باشد. همان‌طور که مشاهده می‌کنید الگوی $pattern$ در $text$ وجود دارد. اما در اجرای الگوریتم بالا $counter$ از اولین حرف c شروع به افزایش می‌کند. وقتی به حرف a در $pattern$ می‌رسیم، در $text$ به حرف c می‌رسیم که این دو برابر نیستند. پس $counter$ برابر با صفر می‌شود و این باعث می‌شود هیچ تطابقی در ادامه‌ی کار پیدا نشود. در واقع

داریم تطابق‌هایی را که ممکن بوده با شروع از کاراکتر c دیگر صورت بگیرد از دست می‌دهیم. این مشکل زمانی اتفاق می‌افتد که بخشی از ابتدای $pattern$ داخل خود $pattern$ تکرار شده باشد.

سوال: چگونه مشکل بالا را حل کنیم؟

الگوریتم دوم:

یک اشاره‌گر به ابتدای $pattern$ و یک اشاره‌گر به ابتدای $text$ در نظر می‌گیریم (به ترتیب j و i) و چک می‌کنیم آیا کاراکتر $i+j$ ام از $text$ با کاراکتر i ام از $pattern$ برابر است یا نه. سپس j را یک کاراکتر به جلو حرکت می‌دهیم. یعنی به ازای هریک از کاراکترهای $text$ وجود $pattern$ را بررسی می‌کنیم.

شبه کد:

```
for j: 0 ->len(text)
        for i: 0 -> text(j+i)= pat(i)
                if i== len(pat)
                        print("found")
```

این الگوریتم مشکلی که داشتیم را رفع می‌کند. اما مرتبه‌ی زمانی آن $O(mn)$ است که m طول $pattern$ و n طول $text$ است.

در الگوریتم بعدی راهی با مرتبه‌ی زمانی بهتر معرفی می‌کنیم.

الگوریتم سوم (KMP):

ابتدا توجه کنید که روشی که در الگوریتم اول معرفی کردیم، تنها زمانی که بخشی از ابتدای $pattern$ در آن تکرار شده‌است، درست کار نمی‌کند. اما اگر تضمینی وجود داشته‌باشد که ابتدای $pattern$ در داخل آن تکرار نشده‌است، درست کار می‌کند. برای درک بهتر، مثال زیر را در نظر بگیرید:

فرض کنید $pattern = zcdcda$ و $text = acdcdcda$ باشد. می‌دانیم کاراکتر اول $pattern$ یعنی z در ادامه‌ی $pattern$ نیامده‌است. پس وقتی حرکت را از ابتدای $text$ شروع می‌کنیم و تا جایی از $text$ پیش می‌رویم اطمینان داریم که در آن فاصله حرف z دیگری وجود نداشته‌است. پس امکان ندارد که در این فاصله کاراکتری را برای شروع $match$ از دست داده باشیم. پس کافیست از ادامه‌ی رشته به دنبال $pattern$ بگردیم و مشکلی که برای الگوریتم اول بیان کردیم اتفاق نمی‌افتد.

بنابراین مشکل تنها زمانی‌است که بخشی از ابتدای $pattern$ در آن تکرار شده باشد. برای حل این مشکل الگوریتم سوم را مطرح می‌کنیم.

ایده‌ی اولیه‌ی این الگوریتم این است که به ازای هر کاراکتر $pattern$ بررسی کنیم در صورت به خطا خوردن ($match$ نشدن) چقدر از کاراکترهای قبلی با ابتدای $pattern$ یکسانند؟ در واقع تابعی را محاسبه می‌کنیم که به ما می‌گوید برای هر کاراکتر اگر در آن‌جا به خطا بخوریم چند کاراکتر عقب‌تر بوده که با شروع $match$ کردن از آن‌جا ممکن بوده یک الگو پیدا شود و به این ترتیب تطابق‌هایی که ممکن بود در الگوریتم اول از دست بدهیم، پیدا می‌شوند.

چند مثال از $pattern$ های مختلف:

$cdcda$

00120

$aaaaab$

012340

$ababcab$

0012012

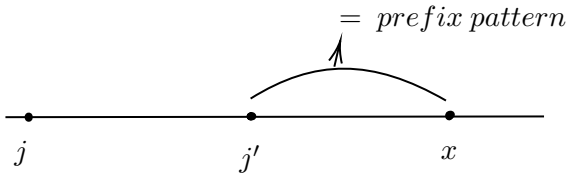
$ababab\hat{c}ab$

001234012

(عدد نوشته شده در زیر هر کاراکتر نشان می‌دهد اگر کاراکتر بعدی $match$ نشود از حرف چندم $pattern$ باید $match$ کردن را ادامه بدهیم.)

فرض کنید از کارکتر j ام $text$ ابتدای یک الگو را پیدا کرده‌ایم ولی در کاراکتر x به خطا می‌خوریم. در الگوریتم اول از ادامه‌ی کاراکتر x به دنبال الگو می‌گشتیم ولی می‌دانیم که ممکن است در فاصله‌ی بین j تا x اندیسی مانند j' وجود داشته باشد که با شروع $match$ از آن‌جا یک الگو بیابیم. ویژگی‌ای که این j' باید داشته باشد این است که ماکسیمم فاصله را از x داشته‌باشد به گونه‌ای که کاراکترهایی که در فاصله‌ی j' تا x هستند، با همان طول از کاراکترهای ابتدای $pattern$ یکسان باشند. در این‌جا تنها کافی‌ست اشاره‌گر $text$ را ثابت نگه داریم و اشاره‌گر $pattern$ را به مقداری که توسط تابع محاسبه شده‌است، کاهش دهیم. نشان می‌دهیم با این روش امکان ندارد هیچ الگویی را از دست داده باشیم.

برهان خلف: فرض کنید کاراکتری مانند j'' قبل از j' وجود دارد که با شروع از آن تا x با ابتدای الگو $match$ شده باشد. در این صورت باید مقدار محاسبه شده توسط تابع بیشتر می‌بود و تابع اشتباه محاسبه شده‌است که تناقض است. پس j' بلند ترین فاصله از x را به ما می‌دهد که با $prefix$ ای از ابتدای $pattern$ یکسان است.



مثال اجرای الگوریتم روی رشته:

رشته‌ی $text = abababababcab$ و $pattern$ زیر را در نظر بگیرید:

$ababab\hat{c}ab$

001234012

از ابتدای $text$ شروع به حرکت می‌کنیم. رشته‌ی $ababab$ از ابتدای $text$ و $pattern$ باهم $match$ می‌شوند. ولی کارکتر بعدی در $text$ برابر با a و کاراکتر بعدی در $pattern$ برابر با c است. در این‌جا اشاره‌گر $text$ ثابت می‌ماند ولی اشاره‌گر $pattern$ به عدد محاسبه شده توسط تابع که در زیر کاراکتر قبلی نوشته‌شده (یعنی ۴) کاهش می‌یابد و از آن‌جای الگو به بعد دوباره $match$ شدن را چک می‌کند. یعنی اشاره‌گر ها در این حالت به صورت زیر قرار گرفته‌اند:

$$text = abababâbabcab$$

$$pattern = ababâbcab$$

همان‌طور که مشاهده می‌کنید هر دو اشاره‌گر روی a قرار گرفته‌اند. پس در هر دو به سمت جلو حرکت می‌کنیم. تا زمانی که دوباره در $text$ به حرف a می‌رسیم در حالی که اشاره‌گر $pattern$ روی حرف c است. به صورت زیر:

$$text = abababababâbcab$$

$$pattern = ababababĉab$$

دوباره شاره‌گر $text$ ثابت می‌ماند ولی اشاره‌گر $pattern$ به عدد محاسبه شده توسط تابع که در زیر کاراکتر قبلی نوشته‌شده (یعنی ۴) کاهش می‌یابد و از آن‌جای الگو به بعد دوباره $match$ شدن را چک می‌کند. یعنی اشاره‌گرها در این حالت به صورت زیر قرار می‌گیرند:

$$text = abababab\hat{a}bcab$$

$$pattern = abab\hat{a}bcab$$

همان‌طور که مشاهده می‌کنید هر دو اشاره‌گر روی a قرار گرفته‌اند. پس در هر دو به سمت جلو حرکت می‌کنیم و به همین ترتیب تا پایان $text$ و $pattern$ حروف یکسانند و الگو پیدا می‌شود.

مثال دیگر:

رشته‌ی $text = abababababcabc$ و $pattern$ زیر را در نظر بگیرید:

$$ababab caba$$

$$0012340123$$

تنها تفاوت این مثال با مثال قبلی افزوده شده یک حرف a به انتهای $pattern$ و افزوده شده یک حرف c به انتهای $text$ است. باز هم از ابتدای $text$ شروع به حرکت می‌کنیم. رشته‌ی $ababab$ از ابتدای $text$ و $pattern$ باهم $match$ می‌شوند. ولی کاراکتر بعدی در $text$ برابر با a و کاراکتر بعدی در $pattern$ برابر با c است. در این‌جا اشاره‌گر $text$ ثابت می‌ماند ولی اشاره‌گر $pattern$ به عدد محاسبه شده توسط تابع که در زیر کاراکتر قبلی نوشته‌شده (یعنی ۴) کاهش می‌یابد و از آن‌جای الگو به بعد دوباره $match$ شدن را چک می‌کند. یعنی اشاره‌گر ها در این حالت به صورت زیر قرار گرفته‌اند:

$$text = ababab\hat{a}babcabc$$

$$pattern = abab\hat{a}bcaba$$

همان‌طور که مشاهده می‌کنید هر دو اشاره‌گر روی a قرار گرفته‌اند. پس در هر دو به سمت جلو حرکت می‌کنیم. تا زمانی که دوباره در $text$ به حرف a می‌رسیم در حالی که اشاره‌گر $pattern$ روی حرف c است. به صورت زیر:

$$text = abababab\hat{a}bcabc$$

$$pattern = ababab\hat{c}aba$$

دوباره اشاره‌گر $text$ ثابت می‌ماند ولی اشاره‌گر $pattern$ به عدد محاسبه شده توسط تابع که در زیر کاراکتر قبلی نوشته‌شده(یعنی ۴) کاهش می‌یابد و از آن‌جای الگو به بعد دوباره $match$ شدن را چک می‌کند. یعنی اشاره‌گرها در این حالت به صورت زیر قرار می‌گیرند:

$$text = abababab\hat{a}bcabc$$

$$pattern = abab\hat{a}bcaba$$

همان‌طور که مشاهده می‌کنید هر دو اشاره‌گر روی a قرار گرفته‌اند. پس در هر دو به سمت جلو حرکت می‌کنیم. تا زمانی که به حالت زیر می‌رسیم: (در $text$ به حرف a می‌رسیم در حالی که اشاره‌گر $pattern$ روی حرف c است.)

$$text = abababababcab\hat{c}$$

$$pattern = ababab cab\hat{a}$$

پس اشاره‌گر $text$ ثابت می‌ماند ولی اشاره‌گر $pattern$ به عدد محاسبه شده توسط تابع که در زیر کاراکتر قبلی نوشته‌شده (یعنی ۲) کاهش می‌یابد:

$$text = abababababcab\hat{c}$$

$$pattern = ab\hat{a}babcaba$$

در ادامه پردازش رشته تمام می‌شود و هیچ $match$ ای پیدا نمی‌شود.

۲ overview مطالب جلسه آینده

در جلسه‌ی آینده این الگوریتم را برای $text = abababababcababcabc$ و $pattern$ زیر بررسی می‌کنیم:

$$ababab$$

$$01234$$

سپس درباره‌ی مرتبه‌ی زمانی الگوریتم صحبت می‌کنیم. برای محاسبه‌ی مرتبه‌ی زمانی باید سه مورد را در نظر بگیریم:
۱ـ طول متن
۲ـ مقادیری که اشاره‌گر $text$ ثابت بوده ولی در $pattern$ به عقب برگشته‌ایم
۳ـ زمان محاسبه‌ی تابع
خواهیم دید میزان بازگشت‌ها به عقب حداکثر به اندازه‌ی مقدار جلو رفتن اشاره‌گر $text$ یعنی طول متن است و محاسبه‌ی تابع نیز در مرتبه‌ی زمانی طول $pattern$ انجام می‌شود. پس در نهایت مرتبه‌ی زمانی کل مساله برابر با $O(n+m)$ می‌شود که m طول $pattern$ و n طول $text$ است.