# ساختمان داده‌ها (۲۲۸۲۲)

**مدرس: حسین بومری**

**[زمستان ۹۹]**

جلسه : بیست و سوم

نگارنده: غزل فراهانی

---

در جلسه گذشته درمورد جست‌وجوی رشته‌ها صحبت کردیم و الگوریتم‌هایی در این زمینه از جلمه الگوریتم $KMP$ مطرح شد. در این جلسه الگوریتم $KMP$ و محاسبه مرتبه زمانی آن را ادامه می‌دهیم و همچنین مثال‌هایی از تحلیل سرشکن می‌بینیم.

## ۱ $KMP$

از جلسه قبل می‌دانیم که می‌توان با اشاره‌گر، یکی برای $pattern$ و دیگری برای $text$ با توجه به $pattern\ function$ تشکیل شده، جست‌وجوی رشته را انجام دهیم.

**محاسبه مرتبه زمانی:** اگر اشاره‌گر روی $text$را $j$ و اشاره‌گر دیگر را $i$ درنظر بگیریم، می‌دانیم در هر مرحله یا $j$ جلو می‌رفت و یا باتوجه به $pattern\ function$، $i$ به عقب برمی‌گشت.در نتیجه هزینه کلی الگوریتم برابر با تعداد جلورفتن‌های $j$ و بازگشت‌های $i$ است. با توجه به اینکه $j$ درنهایت $n$ بار جلو می‌رود، اردر کلی الگوریتم برابر است با:

$$O(n) + O(m).n + O(pattern\ function)$$

که $O(pattern\ function)$ همان هزینه ساخت آرایه اندیس‌های گفته شده است.

**ساختن** $pattern\ function$**:** می‌توانیم با دو اشاره‌گر که یکی اول الگو را چک می‌کند و دیگری با هر $node$ جلو می‌رود این کار را انجام داد به این صورت که اولین حرف صفر است و بعد از آن هربار مقایسه می‌کنیم و اگر مساوی بود اندیس خانه قبلی را بعلاوه یک کرده و در خانه جدید قرار می‌هیم و هر دو اشاره‌گر را یک واحد به جلو می‌بریم.

الگوریتم بالا برای همه رشته‌ها کار نمی‌کند:

$$abcab\hat{a}b$$

در رشته بالا با الگوریتم گفته شده مقدار اندیس $\hat{a}$ برابر با صفر است درحالی که این اندیس باید برابر با ۳ باشد.

یک راه حل مطرح‌شده برای مشکل قبلی این است که هربار که به مشکل خوردیم دوباره از اول رشته شروع به $match$کردن کنیم اما این الگوریتم هم برای بعضی از رشته‌ها درست کار نمی‌کند:

$$ababcabab\hat{a}b$$

در رشته بالا اندیس $\hat{a}$ برابر ۱ می‌شود درحالی که باید برابر با ۳ باشد.

راه حل: می‌دانیم تا زمانی که به خطا بخوریم $pattern$ را حساب کرده‌ایم و درواقع کاری که می‌کنیم این است در $pattern$ دنبال $pattern$ می‌گردیم. در نتیجه:

$$ababcababab$$

001201234*

$pattern:\ ababcababab$

$text:\ babcababab$

$pattern\ function:\ 00120123434$

در واقع الگوریتم نهایی به این صورت است: با استفاده از دو اشاره‌گر هربار مقایسه می‌کنیم، اگر برابر بودند مقدار اندیس خانه قبلی را بعلاوه یک کرده و در خانه جدید قرار می دهیم درغیر این صورت الگوریتم قبلی را تکرار می کنیم.
(در الگوریتم قبلی به ازای هر حرف می‌دانستیم حداکثر چقدر از $pattern$ با آن $match$ شده است، در نتیجه بیشترین طولی را که با اول آرایه یکی است می‌دانیم. درنتیجه کافیست هربار که به مشکل خوردیم، خانه قبلی آن را نگاه کنیم و ببینیم چه مقدار از اول آرایه با آن یکی است و به خانه آن اندیس می‌رویم. توجه کنید که ممکن است که نیاز باشد چند بار به عقب برگردیم. همچنین اندیس‌گذاری‌ها از صفر است.)
**تحلیل مرتبه زمانی:** با استفاده از تحلیل قبلی می‌دانیم اردر برابر است با:

$$O(n)\ +\ O(m).n\ +\ O(pattern\ function)$$

$$O(pattern\ function)\ =\ O(m)\ +\ O(m).m$$

با استفاده از تحلیل سرشکن: می‌دانیم با هربار جلو رفتن $j$، $i$ هم یک واحد جلو می‌رود. همچنین هربار $i$ به عقب برمی‌گردد، معادل آن $i$ قبلا جلو رفته است. در نتیجه جلو رفتن $i$ با جلو رفتن $j$ سرشکن می‌شود و همچنین عقب رفتن $i$هم از جلو رفتن آن کمتر مساوی‌ست و کل هزینه الگوریتم برابر با $j$۳خواهد بود که درنهایت اردر کلی برابر با $O(n)\ +\ O(m)$ است.
با الگوریتم گفته شده، هزینه پیدا کردن $pattern$ به اندازه ورودی گرفتن است و نیاز به حافظه هم ندارد و هرچه کاربر تا قبل از آن تایپ کرده را در حافظه نگه نمی‌داریم.

## ۲ مثال‌هایی از تحلیل سرشکن

اعداد ۶ بیتی را درنظر بگیرید. هربار یک واحد به عدد اضافه می‌کنیم. میخواهیم بدانیم اردر عمل $increment$کردن چقدر خواهد بود.

000000

000001

000010

000011

000100

...

می‌دانیم تعداد اعداد صفری که به یک تبدیل می‌کنیم در هر مرحله برابر با ۱ خواهد بود. همچنین هر عدد یکی که آن را به صفر تبدیل می‌کنیم قبلا صفری بوده که یک شده است. درنتیجه می‌توان تعداد صفرسازی‌ها را با تعداد یک‌سازی‌ها سرشکن کرد. درنتیجه اردر

$increment$کردن تا مرحله $n$ام برابر با $O(n)$ خواهد بود.

در اضافه کردن عضو جدید به لیست ممکن است لازم باشد حافظه جدید بگیریم. حال هربار به جای یک حافظه اضافه گرفتن، دوبرابر حافظه می‌گیریم تا عملیات $pushback$ سریع‌تر انجام شود. اگر $m$را تعداد عملیات گرفتن حافظه، $t$ را تعداد عملیات انتقال آرایه قبلی و $w$ را تعداد عملیات نشوتن عضو جدید در نظر بگیریم، می‌دانیم هر بار که حافظه می‌گیریم به اندازه $۲t$ زمان استفاده می‌کنیم و هر انتقال معادل یک نوشتن است در نتیجه اردر برابر با $O(n)$ خواهد بود:

$$O(m) + O(t) + O(w) = ۲n + n + n = ۴n$$