الگوریتم پیشرفته

بهبود ورودی در تطبیق رشته (Input Enhancement in String Matching)

دکتر حامد فهیمی

دانشگاه فردوسی مشهد

دانشکده علوم ریاضی

الگوریتم‌های Horspool و Boyer-Moore

---

۱. مقدمه

مسئله تطبیق رشته (String Matching) شامل یافتن الگوی P (به طول m) در متن T (به طول n) است.

- **روش Brute-Force:** در بدترین حالت دارای پیچیدگی $O(nm)$ است، زیرا برای هر موقعیت ممکن است m مقایسه انجام دهد.
- **ایده بهبود ورودی (Input Enhancement):** به جای پردازش مستقیم، ابتدا الگوی P را پیش‌پردازش می‌کنیم تا اطلاعاتی به دست آوریم که در حین جستجو به ما اجازه دهد "پرش‌های" بلندتری روی متن انجام دهیم.

دو الگوریتم مشهور که از این ایده استفاده می‌کنند عبارتند از:

1. الگوریتم **Knuth-Morris-Pratt (KMP)** (چپ به راست مقایسه می‌کند).
2. الگوریتم **Boyer-Moore** (راست به چپ مقایسه می‌کند).

در این جلسه روی خانواده الگوریتم‌های Boyer-Moore (شامل نسخه ساده‌شده آن، الگوریتم Horspool) تمرکز می‌کنیم.

---

۲. الگوریتم Horspool (نسخه ساده‌شده Boyer-Moore)

این الگوریتم توسط R. Horspool ارائه شد و اغلب برای متون تصادفی (مانند متن‌های زبان طبیعی) به اندازه Boyer-Moore کارآمد است، اما پیاده‌سازی ساده‌تری دارد.

نحوه عملکرد

1. **جهت مقایسه:** الگو را با متن تراز کرده و مقایسه‌ها را از **راست به چپ** (از آخرین کاراکتر الگو) انجام می‌دهیم.
2. **ایده پرش (Shift):** اگر عدم تطبیق رخ داد یا حتی اگر تطبیق کامل شد، باید الگو را به سمت راست حرکت دهیم. اندازه این پرش بر اساس کاراکتری از متن تعیین می‌شود که **هم‌راستا با آخرین کاراکتر الگو** است. بیایید این کاراکتر را c بنامیم.

محاسبه جدول پرش (Shift Table)

برای تعیین اندازه پرش، یک جدول بر اساس تمام کاراکترهای ممکن الفبا می‌سازیم. فرض کنید طول الگو m است. اندازه پرش $t(c)$ به صورت زیر محاسبه می‌شود :

$$t(c) = \begin{cases} m & \text{اگر } c \text{ در } m-1 \text{ کاراکتر اول الگو نباشد} \\ \text{فاصله آخرین رخداد } c \text{ تا انتهای الگو} & \text{اگر } c \text{ در } m-1 \text{ کاراکتر اول موجود باشد} \end{cases}$$

نکته مهم: ما فقط $m-1$ کاراکتر اول الگو را برای محاسبه فاصله بررسی می‌کنیم. آخرین کاراکتر الگو در محاسبه فاصله لحاظ نمی‌شود (مگر اینکه در جای دیگری از الگو تکرار شده باشد).

مثال: ساخت جدول برای الگوی `BARBER`

طول الگو $m = 6$.

کاراکترهای موجود در $m-1$ بخش اول (`BARBE`): B, A, R, E.

- **B:** در اندیس ۰ و ۳ است. آخرین رخداد (اندیس ۳) فاصله تا انتها ($m-1-3=2$) است. پس $t(B) = 2$.
- **A:** در اندیس ۱ است. فاصله ($6-1-1=4$). پس $t(A) = 4$.

- **R:** در اندیس ۲ است. فاصله $(6 - 1 - 2 = 3)$. پس $t(R) = 3$.
- **E:** در اندیس ۴ است. فاصله $(6 - 1 - 4 = 1)$. پس $t(E) = 1$.
- **سایر کاراکترها:** در الگو نیستند، پس $t(c) = 6$.

کاراکتر	A	B	E	R	سایر
Shift	4	2	1	3	6

شبه‌کد الگوریتم Horspool

```
ALGORITHM HorspoolMatching(P[0..m-1], T[0..n-1])
Preprocessing: Generate Shift Table .1 //
  ShiftTable(P) -> Table

Searching .2 //
  i <- m - 1
  while i <= n - 1 do
    k <- 0
    while k <= m - 1 and P[m - 1 - k] == T[i - k] do
      k <- k + 1
    if k == m
Match found //       return i - m + 1
    else
Shift based on text char aligned with pattern end //       i <- i + Table[T[i]]
  return -1
```

تحلیل پیچیدگی

- **بدترین حالت:** $O(nm)$ (مشابه Brute-force).
- **حالت میانگین:** $O(n)$ (اغلب بسیار سریع‌تر از Brute-force برای متون بزرگ و الفبای بزرگ).

---

۳. الگوریتم Boyer-Moore (نسخه کامل)

این الگوریتم پیچیده‌تر است و از **دو** قانون برای تعیین بیشترین پرش ممکن استفاده می‌کند و ماکسیمم آن دو را انتخاب می‌کند:

$$d = \max\{d_1, d_2\}$$

۱. قانون نماد بد (Bad-Symbol Shift) - d_1

این قانون مشابه الگوریتم Horspool عمل می‌کند اما تعداد کاراکترهای تطبیق‌یافته (k) را نیز در نظر می‌گیرد.
فرمول:

$$d_1 = \max\{t_1(c) - k, 1\}$$

که در آن $t_1(c)$ همان مقدار جدول Horspool برای کاراکتر ناسازگار در متن است و k تعداد کاراکترهایی است که از راست با موفقیت تطبیق داده‌ایم.

۲. قانون پسوند خوب (Good-Suffix Shift) - d_2

این قانون بر اساس کاراکترهایی که با موفقیت تطبیق یافته‌اند (پسوند به طول k) عمل می‌کند.

- الگوریتم بررسی می‌کند آیا این پسوند $(suff(k))$ جای دیگری در الگو تکرار شده است؟
- اگر تکرار شده باشد، الگو را طوری جابجا می‌کند که رخداد قبلی با متن تراز شود.
- اگر تکرار نشده باشد، کل الگو را به اندازه m جابجا می‌کند (با در نظر گرفتن استثناهایی برای پیشوندها) .

مثال اجرایی: الگوی `BAOBAB`

متن: `B E S S _ K N E W _ A B O U T _ B A O B A B S`

1. تطبیق `BAOBAB` با ابتدای متن.

2. آخرین حرف الگو `B` است، حرف متناظر در متن `_` (فاصله) است. تطبیق شکست می‌خورد ($k = 0$).
3. جدول Bad-Symbol برای `_` مقدار ۶ می‌دهد. پرش $d_1 = 6$.
4. ... (الگوریتم ادامه می‌یابد تا جایی که تطبیق بخشی صورت گیرد).
5. در یک مرحله، `BAB` (پسوند) تطبیق می‌یابد اما کاراکتر قبل از آن شکست می‌خورد. در اینجا از قانون Good-Suffix استفاده می‌شود تا الگوی `BAB` بعدی در کلمه `BAOBAB` زیر بخش تطبیق یافته قرار گیرد .

---

۴. حل تمرین‌های منتخب

تمرین ۱: جستجو با Horspool

سوال: الگوریتم Horspool را برای جستجوی الگوی `SORTING` در متن زیر اعمال کنید:
`SORTING_ALGORITHM_CAN_USE_BRUTE_FORCE_METHOD`

حل:
گام ۱: ساخت جدول پرش (Shift Table)
طول الگو $m = 7$. الفبا شامل حروف و `_` است.
بررسی حروف `SORTIN` (بدون G آخر):

- S (اندیس 0): $7 - 1 - 0 = 6$
- O (اندیس 1): $7 - 1 - 1 = 5$
- R (اندیس 2): $7 - 1 - 2 = 4$
- T (اندیس 3): $7 - 1 - 3 = 3$
- I (اندیس 4): $7 - 1 - 4 = 2$
- N (اندیس 5): $7 - 1 - 5 = 1$
- G (آخرین): پیش‌فرض $m = 7$.
- سایر (_ و ...): ۷.

جدول: `{S:6, O:5, R:4, T:3, I:2, N:1, G:7, Others:7}`

گام ۲: جستجو

1. الگو زیر `SORTING` قرار می‌گیرد.
- مقایسه آخر: `G` با `_` (در متن). عدم تطبیق.
- کاراکتر متن `_` است. مقدار پرش از جدول: ۷.
2. الگو ۷ خانه به راست می‌رود. زیر `ALGORITHM` قرار می‌گیرد.
- مقایسه `G` با `M` . عدم تطبیق.
- کاراکتر متن `M` است. در جدول نیست (Others). پرش: ۷.
3. الگو ۷ خانه جلو می‌رود... (و به همین ترتیب ادامه می‌یابد).

تمرین ۲: توالی یابی DNA

سوال: جدول پرش را برای قطعه ژنی `TCCTATTCTT` بسازید.

حل:
طول الگو $m = 10$. الفبا: $\{A, C, G, T\}$.
باید ۹ کاراکتر اول `TCCTATTCT` را بررسی کنیم:

- T (اندیس 0): 9
- C (اندیس 1): 8
- C (اندیس 2): 7
- T (اندیس 3): 6
- A (اندیس 4): 5
- T (اندیس 5): 4
- T (اندیس 6): 3
- C (اندیس 7): 2
- T (اندیس 8): 1

نکته: وقتی یک حرف تکرار می‌شود (مثل T یا C)، **آخرین** رخداد (راست‌ترین در ۹ کاراکتر اول) مقدار نهایی جدول را تعیین می‌کند.

جدول نهایی:

- **A:** مقدار ۵ (از اندیس 4).
- **C:** مقدار ۲ (از اندیس 7).
- **G:** در الگو نیست $\rightarrow 10$.
- **T:** مقدار ۱ (از اندیس 8).