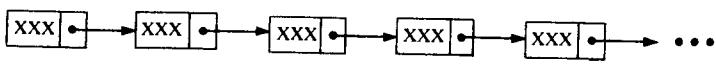
می‌شود. در اینجا ما فقط نگاهی اجمالی به چگونگی ظاهرشدن رشته‌ها در این ساختمان داده خواهیم داشت.

منظور از یک **لیست پیوندی** (یک طرفه)، یک دنباله از خانه‌های **حافظه به نام گره** است که به صورت خطی مرتب شده‌اند و هر گره شامل یک **فیلد موسوم به پیوند یا اتصال** است که به گرهٔ بعدیِ لیست اشاره می‌کند (یعنی دارای آدرس گرهٔ بعدی است).

شکل ۶ـ ۳ نمودار فرضی یک لیست پیوندی را نشان می‌دهد.



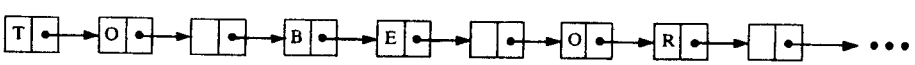
شکل ۶ـ ۳ . لیست پیوندی

رشته‌ها را می‌توان به صورت زیر در لیستهای پیوندی ذخیره کرد. در هر خانهٔ حافظه یک کاراکتر یا تعداد معینی کاراکتر جایگزین می‌شود و یک پیوند که مقداری از حافظه را اشغال می‌کند آدرس خانهٔ حافظه‌ای را که شامل کاراکتر بعدی یا تعدادی از کاراکترهای درون رشته است به دست می‌دهد. برای مثال، جملهٔ مشهور زیر را درنظر بگیرید!

بودن یا نبودن، مسأله این است.

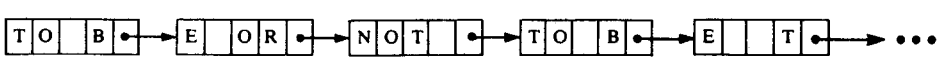
To be or not to be, that is the question.

شکل ۷ـ ۳ (الف) چگونگی ذخیره‌شدن این رشته را در حافظه نشان می‌دهد که هر گره دارای یک کاراکتر است.



شکل ۷ـ ۳ (الف). هر کاراکتر در یک گره

و شکل ۷ـ ۳ (ب) چگونگی ذخیره‌شدن این حافظه را نشان می‌دهد که هر گره دارای چهار کاراکتر است.



شکل ۷ـ ۳ (ب). چهار کاراکتر در یک گره

۴ـ۳ نوع دادهٔ کاراکتری

در این بخش مرور کلی بر روشهایی داریم که زبانهای برنامه‌نویسی مختلف دادهٔ نوع کاراکتری را مورد استفاده قرار می‌دهند. همانگونه که در فصل قبل (در بخش ۷ـ ۲) متذکر شدیم، هر نوعِ داده‌ای دارای فرمول خاص خود جهت از کد در آوردن یک دنباله از بیتها، در حافظه است.

ثابت‌ها

اکثر زبانهای برنامه‌نویسی ثابتهای رشته‌ای را با قراردادن آن، در داخل یک یا دو کوتیشن نمایش می‌دهند. برای مثال :

'THE END' و 'TO BE OR NOT TO BE'

به ترتیب ثابت‌های رشته‌ای با طول 7 و 18 کاراکتر هستند. الگوریتم‌های این کتاب نیز ثابت‌های کاراکتری را به همین صورت تعریف می‌کند.

متغیرها

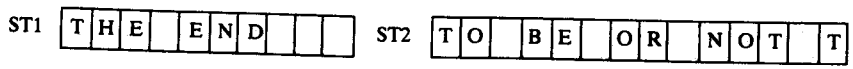
هر زبان برنامه‌نویسی دارای قانون و قاعده خاص خود برای ساختن متغیرهای کاراکتری است. با وجود این، متغیرهای کاراکتری به یکی از سه دستهٔ زیر تقسیم می‌شوند : **ایستا، نیمه‌ایستا و پویا**. منظور از یک متغیر کاراکتری ایستا، متغیری است که طول آن قبل از اجرای برنامه تعریف شده و در سراسر برنامه ثابت است. منظور از یک متغیر کاراکتری نیمه‌ایستا، متغیری است که طول آن ممکن است در خلال اجرای برنامه تغییر کند طوری که طول آن نمی‌تواند از یک مقدار ماگزیممی که قبل از اجرای برنامه توسط برنامه معین می‌شود بزرگتر شود. منظور از یک متغیر کاراکتری پویا، متغیری است که طول آن می‌تواند در خلال اجرای برنامه تغییر کند. این سه دسته متغیر کاراکتری، به ترتیب متناظر با روشهایی هستند که رشته‌ها در حافظهٔ کامپیوتر، به صورتی که در بخش قبل توضیح داده شد، ذخیره می‌شوند.

مثال ۲ـ۳

(الف) بسیاری از نسخه‌های زبان **FORTRAN** از متغیرهای کاراکتری ایستا **CHARACTER** استفاده می‌کنند. برای مثال قطعه برنامهٔ **FORTRAN** زیر را درنظر بگیرید :

```
CHARACTER ST1*10, ST2*14
ST1 = 'THE END'
ST2 = 'TO BE OR NOT TO BE'
```

دستور اول اعلام می‌کند که **ST1** و **ST2** به ترتیب متغیرهای کاراکتری **CHARACTER** به طول **10** و **14** هستند. پس از اجرای هر یک از دو دستورجایگزینی‌بالا، **ST1** و **ST2** در حافظه به صورت زیر ذخیره می‌شوند :



یعنی یک رشته در حافظه به صورت چیده شده از چپ ذخیره می‌شود. هرگاه طول رشته بزرگتر از طول خانه‌های حافظه باشد فضاهای خالی در سمت راست رشته جمع می‌شوند و اگر طول رشته کوچکتر از طول خانه‌های حافظه باشد، رشته از سمت راست بریده می‌شود.

(ب) زبان **BASIC** متغیرهای کاراکتری را به صورتی تعریف می‌کند که در انتهای نام متغیر یک علامت **$** قرار می‌گیرد. به بیان کلی‌تر، متغیرها، از نوع متغیرهای نیمه‌ایستا هستند که طول آنها از یک حد معینی نمی‌تواند بزرگتر باشد. برای مثال، قطعه برنامهٔ **BASIC** زیر :

```
A$ = ''THE BEGINNING''
B$ = ''THE END''
```

متغیرهای **A$** و **B$** را به صورت کاراکتری تعریف کرده است. هنگامی که این قطعه برنامه اجرا می‌شود، طول **A$** و **B$** به ترتیب **13** و **7** خواهد بود.

علاوه بر این در زبان **BASIC** برای نمایش ثابت‌های رشته‌ای، از دو کوتیشن استفاده می‌شود.

(ج) زبان **SNOBOL** از متغیرهای کاراکتری پویا استفاده می‌کند. برای مثال، قطعه برنامهٔ **SNOBOL** زیر :

```
WORD = 'COMPUTER'
TEXT = 'IN THE BEGINNING'
```

متغیرهای **WORD** و **TEXT** را به صورت کاراکتری تعریف می‌کند. هنگامی که این قطعه برنامه اجرا می‌شود طول **WORD** و **TEXT** به ترتیب **18** و **16** خواهد بود. با وجود این، طول این متغیرها می‌تواند دیرتر در برنامه تغییر کند.

(د) زبان **PL / I** هم از متغیرهای کاراکتری ایستا و هم از متغیرهای کاراکتری نیمه‌ایستا استفاده

می‌کند. برای مثال، دستور **PL / I** زیر :

```
DECLARE NAME CHARACTER(20),
        WORD CHARACTER(15) VARYING;
```

متغیر **NAME** را به صورت کاراکتری ایستا به طول **20** و متغیر **WORD** را به صورت کاراکتری نیمه‌ایستا تعریف می‌کند که طول آن می‌تواند تغییر کند ولی نمی‌تواند بیشتر از **15** باشد.

(ه) در زبان **PASCAL** ، یک متغیر کاراکتری (که به اختصار به صورت **CHAR** نوشته می‌شود) می‌تواند تنها یک کاراکتر را نمایش دهد و از این‌رو است که یک رشته به صورت آرایهٔ خطی از کاراکترها تعریف می‌شود. برای مثال :

```
VAR WORD: ARRAY[1 . . 20] OF CHAR
```

WORD را به صورت یک رشتهٔ **20** کاراکتری تعریف می‌کند. علاوه براین، **WORD[1]** اولین کاراکتر این رشته است و **WORD[2]** دومین کاراکتر رشته و غیره می‌باشد. به‌ویژه این که، آرایه‌های کاراکتری در پاسکال طول ثابت دارند و از این‌رو متغیرهای ایستا هستند.

۵ـ۳ عملیات بر روی رشته‌ها

هرچند یک رشته را می‌توان به صورت ساده، یک آرایهٔ خطی از کاراکترها در نظر گرفت. با وجود این در کاربردها بین رشته‌ها و انواع دیگر آرایه‌ها یک تفاوت اساسی وجود دارد. به ویژه این که، گروههایی از عناصر متوالی، در یک رشته (نظیر کلمه‌ها، عبارتها و جمله‌ها) موسوم به **زیررشته‌ها**، می‌توانند واحدهایی مستقل روی این رشته‌ها باشند. علاوه بر این، واحدهای اصلی قابل دسترس در یک رشته معمولاً، زیررشته‌ها هستند نه تک‌تک کاراکترها .

برای مثال رشته زیر را درنظر بگیرید :

```
'TO BE OR NOT TO BE'
```

می‌توانیم این رشته را به صورت دنباله‌ای از **18** کاراکتر **T , O , □ , B ,..., E** درنظر بگیریم. با وجود این، زیر رشته‌های **TO, BE, OR, ...** دارای معنی خاص خود هستند.

از سوی دیگر، یک آرایهٔ خطی **18** عنصری با **18** عدد صحیح زیر را درنظر بگیرید :

```
4, 8, 6, 15, 9, 5, 4, 13, 8, 5, 11, 9, 9, 13, 7, 10. 6, 11
```

واحد اصلیِ قابل دسترس در چنین آرایه‌ای، معمولاً تک‌تک عناصر آن آرایه است. گروههایی متوالی

از عناصر این آرایه، معمولاً معنی خاصی ندارند.

بنابه دلایل بالا، عملیات متعدّدی روی رشته‌ها تعریف شده است که معمولاً با انواع دیگرِ آرایه‌ها بکار برده نمی‌شود. در این بخش، این عملیات روی رشته‌ها توضیح داده می‌شود.

بخش بعد، چگونگی استفاده از این عملیات را در پردازش کلمات یا رشته‌ها نشان می‌دهد. بجز در حالتی که به صورت صریح بیان می‌شود یا به‌طور ضمنی توضیح داده می‌شود، فرض می‌کنیم متغیرهای نوع کاراکتری پویا هستند و طولِ متغیر دارند و طول آن، در متنی که از متغیر استفاده می‌کند، تعیین می‌شود.

زیر رشته

دسترسی به یک زیررشته از یک رشتهٔ داده شده، مستلزم داشتن اطلاعات زیر است : (۱) ***نام رشته یا خود رشته*** (۲) ***مکان اولین کاراکتر زیررشته در رشتهٔ داده شده*** و (۳) ***طول زیر رشته یا مکان آخرین کاراکتر زیر رشته***. ما این عمل را **SUBSTRING** می‌نامیم. به ویژه این که، می‌نویسیم :

SUBSTRING(string, initial, length)

که زیر رشتهٔ، یک رشتهٔ **S (String)** را نشان می‌دهد، اولین کاراکتر آن از مکان **K(Initial)** شروع شده و طول **(Length)** آن **L** است.

مثال ۳ـ ۳

(الف) با استفاده از تابع بالا داریم :

```
SUBSTRING('TO BE OR NOT TO BE', 4, 7) = 'BE OR N'
SUBSTRING('THE END', 4, 4) = '□END'
```

(ب) تابع **SUBSTRING(S, 4, 7)** در برخی از زبانهای برنامه‌نویسی به صورت زیر نمایش داده می‌شود :

PL/1:	SUBSTR(S, 4, 7)
FORTRAN 77:	S(4:10)
UCSD Pascal:	COPY(S, 4, 7)
BASIC:	MID$(S, 4, 7)

شاخص گذاری

شاخص گذاری که **تطبیق الگو** نیز نامیده می‌شود، عبارت است از تعیین مکان الگوی رشته‌ای **P** که برای نخستین بار، در رشتهٔ داده شدهٔ **T** ظاهر شده است. ما این عمل را شاخص‌گذاری **INDEX** می‌نامیم

و می‌نویسیم :

INDEX(text, pattern)

اگر الگوی **P** در متن **T** ظاهر نشود، آنگاه در **INDEX** مقدار **0** جایگزین می‌شود. آرگومان‌های **text** و **pattern** در عمل بالا، می‌تواند ثابتهای رشته‌ای یا متغیرهای رشته‌ای باشند.

مثال ۴ـ۳

(الف) فرض کنید **T** شامل متن

'HIS FATHER IS THE PROFESSOR'

باشد آنگاه : INDEX(T, 'THEN'), INDEX(T, 'THE') و INDEX(T, '□THE□')

به ترتیب مقادیر **7** ، **0** و **14** دارند.

(ب) تابع **INDEX (text , pattern)** در برخی از زبانهای برنامه‌نویسی به صورت زیر نمایش داده می‌شود.

PL/1:	INDEX(text, pattern)
UCSD Pascal:	POS(pattern, text)

توجه دارید که در پاسکال **UCSD** آرگومانها به ترتیب عکس قرار گرفته‌اند.

اتصال دو رشته

فرض کنید S_1 و S_2 دو رشته باشند. از بخش ۲ـ۳ یادآور می‌شویم که **اتصال** S_1 و S_2 که ما آن را با S_1 // S_2 نمایش می‌دهیم، رشته‌ای است که شامل کاراکترهای S_1 و بدنبال آن کاراکترهای S_2 است.

مثال ۵ـ۳

(الف) فرض کنید S_1 = 'MARK' و S_2 = 'TWAIN' باشند. آنگاه

$S_1 // S_2$ = 'MARKTWAIN' اما $S_1 // '□' // S_2$ = 'MARK TWAIN'

(ب) اتصال در برخی از زبانهای برنامه‌نویسی به صورت زیر نمایش داده می‌شود :

PL/1:	$S_1 \| \| S_2$
FORTRAN 77:	$S_1 // S_2$
BASIC:	$S_1 + S_2$
SNOBOL:	$S_1\ S_2$

(بین S_1 و S_2 یک فضای خالی وجود دارد)

طول رشته

تعداد کاراکترهای داخل یک رشته، **طول** آن رشته نامیده می‌شود. برای تعیین طول یک رشتهٔ معلوم می‌نویسیم :

LENGTH(string)

بدین ترتیب :

LENGTH('COMPUTER') = 8 LENGTH('□') = 0 LENGTH('') = 0

برخی از زبانهای برنامه‌نویسی این تابع را به صورت زیر نمایش می‌دهند :

PL/1:	LENGTH(string)
BASIC:	LEN(string)
UCSD Pascal:	LENGTH(string)
SNOBOL:	SIZE(string)

FORTRAN و **PASCAL** استاندارد که از متغیرهای رشته‌ای با طول ثابت استفاده می‌کنند و هیچ تابع کتابخانه‌ای **LENGTH** برای تعیین طول رشته‌ها ندارند. با وجود این، هرگاه از تمام فضای خالی انتهایی رشته صرفنظر کنیم، می‌توان چنین متغیرهایی را متغیرهایی با طول متغیر درنظر گرفت. بنابراین می‌توان در این زبانها، یک زیربرنامهٔ **LENGTH** نوشت طوری که : LENGTH('MARC ') = 4

در واقع، زبان **SNOBOL** دارای تابع کتابخانه‌ای برای رشته‌ها به نام **TRIM** است که فضاهای خالی انتهای رشته را حذف می‌کند :

TRIM('ERIK ') = 'ERIK'

ما در الگوریتم‌های خود، هراز چندگاهی از این تابع **TRIM** استفاده خواهیم کرد.

۶ـ۳ پردازش کلمه یا رشته

در گذشته، داده‌های کاراکتری پردازش شده توسط کامپیوتر، اساساً از عناصر داده‌ای نظیر نام و آدرس تشکیل می‌شد. امروزه کامپیوتر، کارهای تایپی و نشریاتی از قبیل نامه‌ها، مقالات و گزارشها را پردازش می‌کند. بخاطر متنها و مطالب بعدی است که از اصطلاح "پردازش کلمه" یا رشته استفاده می‌کنیم.

متن تایپ شده‌ای داده شده است. معمولاً عملیات مرتبط با پردازش کلمه یا رشته به قرار زیر است :

(الف) جانشین سازی : یک رشته را جانشین رشتهٔ دیگری در داخل متن کنیم.

(ب) اضافه کردن : یک رشته را در وسط متن اضافه کنیم.

(ج) حذف‌کردن : یک رشته را از متن حذف کنیم.

عملیات بالا را می‌توان با استفاده از عملیات روی رشته‌ها، که در بخش قبل توضیح دادیم، اجرا نمود. در زیر، این عملیات را همراه با بررسیِ آنها به طور مستقل، شرح می‌دهیم. بسیاری از این عملیات، در هر یک از زبانهای برنامه‌نویسی شرح داده شده در بالا، یا به صورت کتابخانه‌ای وجود دارند یا می‌توان آنها را به‌سادگی تعریف کرد.

اضافه کردن

متن داده شدهٔ T را در نظر بگیرید، می‌خواهیم رشتهٔ S را طوری به آن اضافه کنیم که S از مکان K شروع شود. ما این عمل را به صورت

INSERT(text, position, string)

نمایش می‌دهیم . برای مثال :

INSERT('ABCDEFG', 3, 'XYZ') = 'ABXYZCDEFG'
INSERT('ABCDEFG', 6, 'XYZ') = 'ABCDEXYZFG'

این تابع INSERT را می‌توان با استفاده از عملیاتِ روی رشته‌ها، که در بخش قبل تعریف شده است پیاده‌سازی کرد :

INSERT(T, K, S) = SUBSTRING(T, 1, K − 1)//S//SUBSTRING(T, K, LENGTH(T) − K + 1)

یعنی زیررشتهٔ ابتدایی T که قبل از مکان K است و طول K − 1 دارد، به رشتهٔ S متصل شده است و نتیجه به بقیهٔ قسمت T متصل می‌شود که از مکان K شروع می‌شود و طول LENGTH(T) − (K − 1) = LENGTH(T) K + 1 دارد. ما به صورت ضمنی فرض کرده‌ایم که T یک متغیر پویا است و اندازهٔ T خیلی بزرگ نمی‌شود.

حذف کردن

متن داده‌شدهٔ T را درنظر بگیرید، می‌خواهیم زیررشته‌ای را از آن حذف کنیم که از مکان K شروع می‌شود و طول L دارد. این عمل را به صورت زیر نمایش می‌دهیم :

DELETE(text, position, length)

برای مثال

DELETE('ABCDEFG', 4, 2) = 'ABCFG'
DELETE('ABCDEFG', 2, 4) = 'AFG'

فرض می‌کنیم که عمل حذف زمانی که مکان **K = 0** است انجام نمی‌شود. بنابراین اهمیت این "حالت صفر" دیرتر مشاهده می‌شود.

تابع **DELETE** را می‌توان با استفاده از عملیات بر روی رشته‌ها، که در بخش قبل به صورت زیر داده شده است پیاده‌سازی کرد :

```
DELETE(T, K, L) =
    SUBSTRING(T, 1, K – 1)//SUBSTRING(T, K + L, LENGTH(T) – K – L + 1)
```

یعنی زیررشتهٔ ابتدایی **T** که قبل از مکان **K** است به زیررشتهٔ انتهایی **T** که از مکان **K + L** شروع می‌شود متصل شده است. طول زیررشتهٔ ابتدایی **K – 1** است و طول زیررشتهٔ انتهایی برابر است با :

$$\text{LENGTH}(T) - (K + L - 1) = \text{LENGTH}(T) - K - L + 1$$

علاوه بر این فرض می‌کنیم به‌ازای **K = 0** ، **DELETE(T, K, L) = T** .

حال فرض کنید متن **T** و الگوی **P** داده شده است و بخواهیم از متن **T** اولین وقوع الگوی **P** را حذف کنیم. این عمل با استفاده از تابع **DELETE** بالا به صورت زیر انجام می‌شود :

```
DELETE(T, INDEX(T, P), LENGTH(P))
```

یعنی در متن **T** ، نخست **INDEX(T, P)** را محاسبه می‌کنیم یعنی مکانی را که الگوی **P** برای اولین بار در **T** ظاهر شده است پیدا می‌کنیم و بدنبال آن **LENGTH(P)** یعنی تعداد کاراکترهای **P** را محاسبه می‌کنیم. یادآور می‌شویم که وقتی **INDEX(T, P) = 0** (یعنی هنگامی که **P** در **T** ظاهر نشده باشد) متن **T** تغییر نمی‌کند.

مثال ۶ـ ۳

(الف) فرض کنید **T = 'ABCDEFG'** و **P = 'CD'** آنگاه **INDEX(T, P) = 3** و **LENGTH(P) = 2** از این رو

```
DELETE('ABCDEFG', 3, 2) = 'ABEFG'
```

(ب) فرض کنید **T = 'ABCDEFG'** و **P = 'CD'** آنگاه **INDEX(T, P) = 0** و **LENGTH(P) = 2** از این رو، بنا به "حالت صفر":

```
DELETE('ABCDEFG', 0, 2) = 'ABCDEFG'
```

که همان نتیجهٔ موردانتظار است.

فرض کنید پس از خواندن متن **T** و الگوی **P** در کامپیوتر، بخواهیم هر وقوع الگوی **P** در متن **T** را حذف کنیم. این عمل را می‌توان با تکرار چند بار دستور

DELETE(T, INDEX(T, P), LENGTH(P))

انجام داد تا این که **INDEX(T, P) = 0** (یعنی به‌محض این که **P** در **T** ظاهر نشده است).

الگوریتمی که این عمل را انجام می‌دهد به صورت زیر است :

Algorithm 3.1: A text T and a pattern P are in memory. This algorithm deletes every occurrence of P in T.

1. [Find index of P.] Set K := INDEX(T, P).
2. Repeat while K ≠ 0:
   (*a*) [Delete P from T.]
   Set T := DELETE(T, INDEX(T, P), LENGTH(P))
   (*b*) [Update index.] Set K := INDEX(T, P).
   [End of loop.]
3. Write: T.
4. Exit.

تأکید می‌کنیم که پس از هر عمل حذف طول **T** کاهش می‌یابد و از این‌رو الگوریتم باید متوقف شود. باوجود این، تعداد دفعاتی که حلقه اجرا می‌شود می‌تواند تعداد دفعات ظاهرشدن **P** در متن اولیهٔ **T** را افزایش دهد، که در مثال زیر نشان داده شده است.

مثال ۷ـ ۳

(الف) فرض کنید الگوریتم 3.1 با داده‌های زیر اجرا شده است :

T = XABYABZ, P = AB

آنگاه حلقهٔ داخلی الگوریتم دوبار اجرا خواهد شد. در خلال اجرای اول الگوریتم، اولین وقوع **AB** از **T** حذف خواهد شد که درنتیجهٔ آن **T** به صورت **T=XYABZ** درخواهد آمد. درخلال اجرای دوم، وقوع باقیماندهٔ **AB** از **T** حذف خواهد شد. از این‌رو **T = XYZ**. بنابراین **XYZ** خروجی الگوریتم است.

(ب) فرض کنید الگوریتم 3.1 با داده‌های زیر اجرا شده است :

T = XAAABBBY, P = AB

ملاحظه می‌کنید که الگوی **AB** تنها یکبار در **T** ظاهر شده است اما حلقهٔ الگوریتم سه بار اجرا می‌شود. به‌ویژه این‌که، پس از حذف **AB** در مرحلهٔ اول از **T** داریم. **T=XAABBY** و از این‌رو **AB** مجدداً در **T** ظاهر شده است. پس از حذف **AB** در مرحلهٔ دوم از **T** ملاحظه می‌کنید که **T=XABY** و **AB** همچنان در **T** وجود دارد. بالاخره پس از حذف **AB** در مرحلهٔ سوم از **T** داریم **T=XY** و **AB** در **T** ظاهر نشده است و بدین‌ترتیب **INDEX(T,P) = 0** . از این‌رو **XY** خروجی الگوریتم است.

مثال بالا نشان می‌دهد که با تغییر متن **T** در اثر حذف الگو، الگو که در ابتدا در متن وجود داشت درانتهای این عمل در متن ظاهر نخواهد شد.

جانشین سازی

فرض کنید متن **T** داده شده است، می‌خواهیم الگوی P_2 را جانشین اولین وقوع الگوی P_1 کنیم. این عمل را به صورت زیر نمایش می‌دهیم :

REPLACE(text, $pattern_1$, $pattern_2$)

برای مثال :

REPLACE('XABYABZ', 'AB', 'C') = 'XCYABZ'
REPLACE('XABYABZ', 'BA', 'C') = 'XABYABZ'

در حالت دوم، الگو **BA** در متن ظاهر نشده است و از این‌رو هیچ جانشانی صورت نمی‌گیرد. توجه داشته باشید که تابع **REPLACE** را می‌توان به صورت ترکیب یک عمل حذف و یک عمل اضافه‌کردن بیان کرد، مشروط بر این که از همان تابعهای قبلی **DELETE** و **INSERT** استفاده کنید. به‌ویژه این که، تابع **REPLACE** را می‌توان با استفاده از سه مرحلهٔ زیر اجرا کرد:

K := INDEX(T, P_1)
T := DELETE(T, K, LENGTH(P_1))
INSERT(T, K, P_2)

دو مرحلهٔ اول P_1 را از **T** حذف می‌کند و مرحلهٔ سوم P_2 را در مکان **X** ای که P_1 حذف شد به **T** اضافه می‌کند.

فرض کنید متن **T** و الگوهای **P** و **Q** در حافظهٔ یک کامپیوتر قرار دارند و بخواهیم هر وقوع **P** در متن **T** توسط الگوی **Q** جانشین شود. این عمل را با استفادهٔ مکرر از دستور

REPLACE(T, P, Q)

تا وقتی که **INDEX(T, P) = 0** می‌توان انجام داد (یعنی تا وقتی که الگوی **P** در **T** وجود نداشته باشد).

الگوریتمی که این کار را انجام می‌دهد به شرح زیر است :

Algorithm 3.2: A text T and patterns P and Q are in memory. This algorithm replaces every occurrence of P in T by Q.

1. [Find index of P.] Set K := INDEX(T, P).
2. Repeat while K ≠ 0:
   (*a*) [Replace P by Q.] Set T := REPLACE(T, P, Q).
   (*b*) [Update index.] Set K := INDEX(T, P).
   [End of loop.]
3. Write: T.
4. Exit.

توجه کنید : هرچند این الگوریتم شباهت زیادی با الگوریتم 3.1 دارد اما پایان یافتن الگوریتم را تضمین نمی‌کند. این واقعیت در مثال ۸ـ ۳ (ب) توضیح داده شده است. از طرف دیگر فرض کنید که طول Q کوچکتر از طول P باشد. در این صورت طول T پس از جانشانی کاهش می‌یابد. این مطلب تضمین می‌کند که در این حالت خاص Q کوچکتر از P است و باید الگوریتم پایان یابد.

مثال ۸ـ ۳

(الف) فرض کنید الگوریتم 3.2 با داده‌های زیر اجرا شده است :

$$T = XABYABZ, \quad P = AB, \quad Q = C$$

آنگاه حلقهٔ داخل الگوریتم دوبار اجرا خواهد شد. در خلال اجرای اول الگوریتم، C جانشین اولین وقوع AB در T می‌شود و نتیجه می‌دهد T = XCYABZ. در خلال اجرای دوم، C جانشین وقوع باقیمانده AB در T می‌شود و نتیجه می‌دهد T = XCYCZ. از این‌رو XCYCZ خروجی الگوریتم است.

(ب) فرض کنید الگوریتم 3.2 با داده‌های زیر اجرا شده است :

$$T = XAY, \quad P = A, \quad Q = AB$$

در آن صورت این الگوریتم هیچوقت به پایان نمی‌رسد. به این دلیل که صرفنظر از تعداد دفعاتی که حلقه اجرا می‌شود P همواره در متن T ظاهر خواهد شد. به‌ویژه این که:

T = XABY در پایان اجرای اول حلقه است.

$T = XAB^2Y$ در پایان اجرای دوم حلقه است.

.....

$T = XAB^2Y$ در پایان اجرای n ام حلقه است.

چون P یک زیررشتهٔ Q است از این‌رو در اینجا حلقهٔ بی‌پایان Infinite Loop یا LOOP خواهیم داشت.

۷ـ۳ الگوریتم‌های تطبیق الگو

تطبیق الگو، مسأله‌ای است که تعیین می‌کند یک الگوی رشته‌ای داده شدهٔ P در متن رشته‌ای T وجود دارد یا خیر. فرض می‌کنیم که طول P کوچکتر از طول T است. این بخش دو الگوریتم در مورد تطبیق الگوها را موردبحث و بررسی کامل قرار می‌دهد. علاوه بر آن، پیچیدگی این الگوریتم‌ها مورد بررسی قرار می‌گیرد تا بتوان کارآیی آنها را مورد مقایسه قرار داد.

توجه کنید : درخلال بررسی الگوریتم‌های تطبیق الگو، کاراکترها گاهی اوقات با حروف کوچک a، b، c، ... نمایش داده می‌شوند و از توانها برای نمایش تعداد دفعات تکرار آنها استفاده می‌شود، مانند:

$a^2b^3ab^2$ به جای $aabbbabb$ و $(cd)^3$ به جای $cdcdcd$

علاوه بر این، رشتهٔ تهی با حرف یونانی لاندا Λ نمایش داده می‌شود و اتصال رشته‌های X و Y به صورت X.Y یا فقط XY نمایش داده می‌شود.

الگوریتم اول تطبیق الگو

الگوریتم اول تطبیق الگو، الگوریتم روشنی است که در آن الگوی داده شدهٔ P با هر یک از زیررشته‌های T مقایسه می‌شود. عمل مقایسه با حرکت از چپ به راست متن T انجام می‌شود تا به یک تطبیق با P برسیم. به‌طور مشروح این که، فرض کنید :

$$W_K = \text{SUBSTRING}(T, K, \text{LENGTH}(P))$$

باشد. به‌عبارت دیگر فرض کنید W_k زیررشته‌ای از T با همان طول P و با شروع از K امین کاراکتر T باشد. نخست P را کاراکتر به کاراکتر با اولین زیررشته، W_1 مقایسه می‌کنیم. اگر تمام کاراکترها مساوی باشند در آن‌صورت $P = W_1$ و درنتیجه P در T ظاهرشده است و INDEX(T , P) = 1 . از سوی دیگر، فرض کنید به این نتیجه رسیدیم که یک کاراکتر P همان کاراکتر متناظر در W_1 نیست. در آن صورت $P \neq W_1$ و بلافاصله می‌توانیم به زیررشتهٔ بعدی W_2 برویم به عبارت دیگر، در مرحلهٔ بعد P را با W_2 مقایسه می‌کنیم. اگر $P = W_2$ ، در آن صورت P را با W_3 مقایسه می‌کنیم و الی‌آخر. عمل مقایسه متوقف می‌شود :

(الف) هرگاه به یک تطبیق P با زیررشتهٔ W_k برسیم و از این رو P در متن T وجود دارد و INDEX (T,P) = K یا (ب) هرگاه به هیچ تطبیق P با زیررشتهٔ W_k نرسیم و از این‌رو P در متن T وجود نخواهد داشت.

MAX مقدار ماگزیمم اندیس K برابر است با LENGTH(T) − LENGTH(P) + 1 .

حال به عنوان مثال فرض کنید که P یک رشتهٔ 4کاراکتری و T یک رشتهٔ 20کاراکتری باشد و فرض کنید P و T در حافظه به صورت آرایه‌های خطی نمایش داده شده‌اند که در هر عنصر آرایه یک کاراکتر ذخیره شده است. یعنی

$$P = P[1]P[2]P[3]P[4] \quad \text{و} \quad T = T[1]T[2]T[3]...T[19]T[20]$$

آنگاه P با هر یک از زیررشته‌های 4 کاراکتری بعدی T مقایسه می‌شود:

$$W_1 = T[1]T[2]T[3]T[4], \quad W_2 = T[2]T[3]T[4]T[5], \quad \ldots, \quad W_{17} = T[17]T[18]T[19]T[20]$$

توجه داشته باشید که تعداد $MAX = 20 - 4 + 1 = 17$ زیررشتهٔ 4 کاراکتری در T وجود دارد.

نمایش رسمی این الگوریتم که در آن P یک رشتهٔ R کاراکتری و T یک رشتهٔ S کاراکتری است در الگوریتم 3.3 نشان داده شده است.

ملاحظه می‌کنید الگوریتم 3.3 شامل دو حلقه است، یکی از حلقه‌ها داخل حلقهٔ دیگر قرار دارد. حلقهٔ خارجی برای هر زیررشته R کاراکتری متوالی

$$W_K = T[K]T[K+1]\cdots T[K+R-1]$$

رشتهٔ T اجرا می‌شود. حلقهٔ داخلی P را با W_k، کاراکتر به کاراکتر مقایسه می‌کند. اگر هیچ کاراکتری تطبیق نکند، در آن‌صورت کنترل به مرحلهٔ S داده می‌شود که K را افزایش می‌دهد و پس از آن منتهی به زیررشتهٔ بعدی T می‌شود. اگر تمام R کاراکتر P با کاراکترهای W_k تطابق داشته باشد، آنگاه P در T وجود دارد و K اندیس INDEX الگوی P در T است. از سوی دیگر، اگر حلقهٔ خارجی به پایان تمام مراحل خود برسد ولی در T هیچ P ظاهر نشود در آن صورت INDEX = 0.

Algorithm 3.3: (Pattern Matching) P and T are strings with lengths R and S, respectively, and are stored as arrays with one character per element. This algorithm finds the INDEX of P in T.

1. [Initialize.] Set K := 1 and MAX := S − R + 1.
2. Repeat Steps 3 to 5 while K ≤ MAX:
3. Repeat for L = 1 to R: [Tests each character of P.]
   If P[L] ≠ T[K + L − 1], then: Go to Step 5.
   [End of inner loop.]
4. [Success.] Set INDEX = K, and Exit.
5. Set K := K + 1.
   [End of Step 2 outer loop.]
6. [Failure.] Set INDEX = 0.
7. Exit.

پیچیدگی این الگوریتم تطبیق متن به‌وسیلهٔ C تعداد مقایسه بین کاراکترهای الگوی P و کاراکترهای متن T اندازه گرفته می‌شود. برای پیداکردن C ، فرض می‌کنیم N_k نمایش تعداد مقایسه‌هایی باشد که هنگام مقایسهٔ P با W_k درحلقهٔ داخلی اتفاق می‌افتد. درآن صورت

$$C = N_1 + N_2 + \cdots + N_L$$

که در آن L ، مکان L در متن T است که در آن P برای اولین بار در T ظاهر می‌شود یا L = MAX است اگر P در متن T ظاهر نشده باشد.

مثال بعدی C را برای یک P ی مشخص و T محاسبه می‌کند که در آن LENGTH(P) = 4 و LENGTH(T) = 20 و درنتیجه MAX = 20 − 4 + 1 = 17 .

مثال ۹ـ۳

(الف) فرض کنید $T = cdcd...cd = (cd)^{10}$. واضح است که P در T ظاهر نشده است. علاوه بر این، برای هر 17 بار اجرای حلقه، $N_k = 1$ ، چون اولین کاراکتر P با W_k مطابقت نمی‌کند. از این رو

$$C = 1 + 1 + 1 + ... + 1 = 17$$

(ب) فرض کنید P = aaba و T = ababaaba... ملاحظه می‌کنید که P یک زیررشتهٔ T است. در واقع $P = W_5$ و درنتیجه $N_5 = 4$. علاوه بر این از مقایسهٔ P با $W_1 = abab$ نتیجه می‌گیریم که $N_1 = 2$ ، چون اولین حرف این دو رشته با هم مطابقت دارد اما از مقایسهٔ P با $W_2 = baba$ ، ملاحظه می‌کنید که $N_2 = 1$ چون اولین حرفهای آنها با هم مطابقت ندارد. به‌طور مشابه، $N_3 = 2$ و $N_4 = 1$. بنابراین

$$C = 2 + 1 + 2 + 1 + 4 = 10$$

(ج) فرض کنید P = aaab و $T = aa..a = a^{20}$. در اینجا P در T ظاهر نشده است. همچنین هر $W_k = aaaa$ ، از این‌رو هر $N_k = 4$ ، چون سه حرف اول P با سه حرف اول T مطابقت می‌کند. بنابراین

$$C = 4 + 4 + ... + 4 = 17.4 = 68$$

در حالت کلی، هنگامی که P یک رشتهٔ r کاراکتری و T یک رشتهٔ s کاراکتر است، اندازهٔ داده‌های این الگوریتم برابر است با :

$$n = r + s$$

بدترین حالت وقتی اتفاق می‌افتد که همانند مثال ۹ـ۳ (ج) تمام کاراکترهای P بجز آخرین کاراکتر با هر زیررشتهٔ W_k مطابقت داشته باشد. در این حالت، $C(n) = r(s - r + 1)$ برای مقدار ثابت n داریم $s = n - r$ طوری که

$$C(n) = r(n - 2r + 1)$$

مقدار ماگزیمم C(n) وقتی اتفاق می‌افتد که r = (n + 1)/4 (مسأله ۱۹-۳ را ببینید.) باشد. بنابراین با قراردادن این مقدار به جای r در فرمول مربوط به C(n) نتیجه می‌گیریم:

$$C(n) = \frac{(n+1)^2}{8} = O(n^2)$$

پیچیدگی حالت میانگین در هر وضعیت واقعی، بستگی به حالتهای محتملی دارد که معمولاً ناشناخته هستند. هرگاه کاراکترهای P و T به تصادف از روی یکی از حروف الفبا انتخاب شوند، تجزیه و تحلیل پیچیدگی حالت میانگین همچنان مشکل است، اما پیچیدگی حالت میانگین همچنان به صورت ضریبی از بدترین حالت است. بنابراین، پیچیدگی این الگوریتم را چنین بیان می‌کنیم: *پیچیدگی الگوریتم تطبیق الگو برابر* $O(n^2)$ *است*. به بیان دیگر، زمان موردنیاز برای اجرای این الگوریتم متناسب با n^2 است. (این نتیجه را با نتیجهٔ صفحه ۸۹ مقایسه کنید.)

الگوریتم دوم تطبیق الگو

الگوریتم دوم تطبیق الگو، از جدولی استفاده می‌کند که از یک الگوی خاص P مشتق شده است اما مستقل از متن T است. برای روشن‌شدن مطلب فرض کنید :

P = aaba

نخست دلیلی برای درایه‌های جدول اقامه می‌کنیم و چگونگی استفاده از آنها را شرح می‌دهیم. فرض کنید $T = T_1T_2T_3...$ که در آن T_i کاراکتر i ام T را نمایش می‌دهد و فرض کنید کاراکتر اول T با دو کاراکتر اول P مطابقت می‌کند یعنی فرض کنید T = aa... آنگاه T دارای یکی از سه صورت زیر است :

(i) $T = aab\ldots$, (ii) $T = aaa\ldots$, (iii) $T = aax$

که در آن x می‌تواند هر کاراکتر دلخواهی به غیر از a یا b باشد. فرض کنید T_3 را خواندیم و دریافتیم که $T_3 = b$. در آن صورت، بدنبال آن T_4 را می‌خوانیم تا ببینیم آیا $T_4 = a$ است یا خیر. P با W_1 تطابق خواهد داشت. ازطرف دیگر، فرض کنید $T_3 = a$. در آن صورت، می‌دانیم که $P \neq W_1$، اما همچنین می‌دانیم که $W_2 = aa...$ یعنی دو کاراکتر اول زیررشتهٔ W_2 با دو کاراکتر اول P مطابقت می‌کند. از این‌رو بعد از آن T_4 را می‌خوانیم تا ببینیم آیا $T_4 = b$ است. بالاخره، فرض کنید $T_3 = x$. درآن صورت می‌دانیم $P \neq W_1$، اما همچنین می‌دانیم که $P \neq W_2$ و $P \neq W_3$، چون x در P ظاهر نمی‌شود. از این‌رو بعد از آن، T_4 را می‌خوانیم تا ببینیم آیا $T_4 = a$، یا خیر. یعنی ببینیم که آیا کاراکتر اول W_4 با کاراکتر اول P مطابقت می‌کند یا خیر.

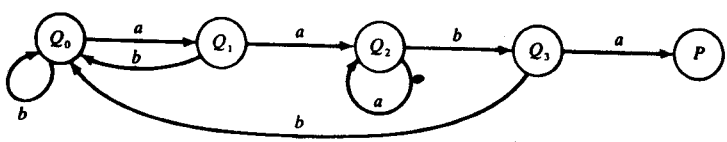
دو نکتهٔ قابل توجه و مهم در روش بالا وجود دارد. نخست، هنگام خواندن T_3، تنها لازم است T_3 با

آن تعداد از کاراکترها، که در P وجود دارد مقایسه شود. اگر هیچ یک از این کاراکترها مطابقت نداشت، در آن‌صورت ما در حالت آخری هستیم که کاراکتری مانند x در P وجود ندارد. دوم، پس از خواندن و بررسی T_3 ، T_4 را می‌خوانیم. لازم نیست مجدداً به متن T برگردیم.

شکل ۸ـ ۳ (الف) شامل جدولی است که برای الگوی P = aaba از الگوریتم دوم تطبیق الگو استفاده می‌کند. در هر دو مورد جدول و گراف همراه آن، یعنی الگوی P و زیررشته‌های Q با حروف کج لاتین نمایش داده شده است.

	a	b	x
Q_0	Q_1	Q_0	Q_0
Q_1	Q_2	Q_0	Q_0
Q_2	Q_2	Q_3	Q_0
Q_3	P	Q_0	Q_0

(الف) جدول تطبیق الگو



(ب) گراف تطبیق الگو

شکل ۸ـ۳

طرز تشکیل جدول به صورت زیر است. قبل از هر کاری، فرض می‌کنیم Q_i نمایش زیررشتهٔ اولیهٔ P با طول i باشد. از این‌رو

$$Q_0 = \Lambda, \quad Q_1 = a, \quad Q_2 = a^2, \quad Q_3 = a^2b, \quad Q_4 = a^2ba = P$$

در اینجا $Q_0 = \Lambda$ رشتهٔ تهی است. سطرهای این جدول را با زیررشته‌های اولیهٔ P به جزء خود P ، برچسب‌گذاری می‌کنیم. ستونهای این جدول با a ، b ، و x برچسب‌گذاری می‌شوند که در آن x هر کاراکتر دلخواهی می‌تواند باشد که در الگوی P ظاهر نشده است. فرض کنید f تابع‌ای باشد که این جدول معین می‌کند، یعنی فرض کنید تابع

$$f(Q_i , t)$$

درایهٔ سطر Q_i و ستون t را نمایش دهد که در آن t می‌تواند هر کاراکتر دلخواهی باشد. درایهٔ $f(Q_i , t)$ بنابه تعریف بزرگترین Q ای است که در رشتهٔ Q_i t ، یعنی درنتیجهٔ اتصال Q_i و t به صورت زیررشتهٔ نهایی ظاهر می‌شود. برای مثال :

a^2 بزرگترین Q ای است که یک زیررشتهٔ پایانی $Q_2a = a^3$ است از این‌رو $f(Q_2, a) = Q_2$

۸ بزرگترین Q ای است که یک زیررشتهٔ پایانی $Q_1b = ab$ است از این‌رو $f(Q_1, b) = Q_0$

a بزرگترین Q ای است که یک زیررشتهٔ پایانی $Q_0a = a$ است از این‌رو $f(Q_0, a) = Q_1$

۸ بزرگترین Q ای است که یک زیررشتهٔ پایانی $Q_3x = a^2bx$ است از این‌رو $f(Q_3, x) = Q_0$

و الی آخر، اگرچه $Q_1 = a$ زیررشتهٔ پایانی $Q_2a = a^3$ است، داریم $f(Q_2, a) = Q_2$ زیرا Q_2 نیز یک زیررشتهٔ پایانی $Q_2a = a^3$ است و Q_2 بزرگتر از Q_1 است. توجه دارید که به ازای هر Q ، $f(Q_i, x) = Q_0$ چون x در الگوی P ظاهر نشده است. بنابراین، ستون متناظر با x معمولاً از جدول حذف می‌شود.

جدول را می‌توان با استفاده از گراف جهت‌دار برچسب‌گذاری شدهٔ شکل ۸ـ ۳ (ب) رسم کرد. گراف به صورت زیر به دست می‌آید. نخست، متناظر با هر زیررشتهٔ اولیهٔ Q_i از P یک گره در گراف وجود دارد. Q ها **حالتهای گراف** نامیده می‌شوند و Q_0 حالت **اولیه** نام دارد. دوم، متناظر با هر درایه در جدول، یک پیکان (یک یال جهت‌دار) در گراف وجود دارد. به‌ویژه این که، اگر

$$f(Q_i , t) = Q_1$$

آنگاه یک پیکان با برچسب t از Q_i به Q_j وجود دارد. برای مثال، $f(Q_2, b) = Q_3$ از این‌رو یک پیکان با برچسب b از Q_2 به Q_3 وجود دارد. جهت سهولت در نمادگذاری، تمام پیکان‌های با برچسب x را که اجباراً منتهی به حالت اولیه Q_0 می‌شوند حذف کرده‌ایم.اکنون آماده‌ایم که برای الگوی $P = aaba$ الگوریتم دوم تطبیق الگو را ارائه دهیم. توجه دارید که در بحث زیر، برای نام تمام متغیرهای یک حرفی داخل الگوریتم، از حروف بزرگ استفاده می‌کنیم. فرض کنید $T = T_1T_2T_3...T_N$ نمایش متن رشتهٔ n کاراکتری باشد که در آن برای یافتن الگوی P عمل جستجو انجام می‌شود. با شروع از حالت اولیهٔ Q_0 و استفاده از متن T به یک دنباله از حالتهای S_1 ، S_2 ، S_3 ، ... به صورت زیر دست می‌یابیم. فرض کنید $S_1 = Q_0$ و اولین کاراکتر T_1 را می‌خوانیم. چه ازطریق جدول عمل کنیم و چه از گراف شکل ۸ـ ۳ استفاده کنیم، زوج (S_1, T_1) حالت دوم S_2 را نتیجه می‌دهد یعنی $F(S_1, T_1) = S_2$ ، کاراکتر بعدی T_2 را می‌خوانیم، زوج (S_2 , T_2) حالت S_3 را نتیجه می‌دهد و الی آخر. دو حالت ممکن وجود دارد :

(۱) حالت $S_k = P$ ، الگوی موردنظر باشد. در این حالت، P در T وجود دارد و اندیس آن

$K - LENGTH(P)$ است.

(۲) هیچ حالتی از S_1 ، S_2 ... ، S_{N+1}، برابر **P** نیست. در این حالت، **P** در **T** وجود ندارد.

الگوریتم را با استفاده از الگوی **P = aaba** و دو متن مختلف توضیح می‌دهیم.

مثال ۱۰ـ۳

(الف) فرض کنید **T = aabcaba** . با شروع از Q_0 و استفاده از کاراکترهای **T** و گراف (یا جدول) شکل ۸ـ۳ ، دنباله حالتهای زیر به دست می‌آید :

$$Q_0 \xrightarrow{Ca} Q_1 \xrightarrow{Ca} Q_2 \xrightarrow{Cb} Q_3 \xrightarrow{Cc} Q_0 \xrightarrow{Ca} Q_1 \xrightarrow{Cb} Q_0 \xrightarrow{Ca} Q_1$$

به حالت **P** دست نمی‌یابیم،از این‌رو **P** در **T** وجود ندارد.

(ب) فرض کنید **T = abcaabaca** . آنگاه به دنباله حالتهای زیر دست می‌یابیم :

$$Q_0 \xrightarrow{Ca} Q_1 \xrightarrow{Cb} Q_0 \xrightarrow{Cc} Q_0 \xrightarrow{Ca} Q_1 \xrightarrow{Ca} Q_2 \xrightarrow{Cb} Q_3 \xrightarrow{Ca} P$$

در اینجا، در حالت S_8 به الگوی **P** دست می‌یابیم. از این‌رو **P** در **T** ظاهر شده است و اندیس آن $8 - LENGTH\ (P) = 4$ است.

بیان رسمی الگوریتم دوم تطبیق الگو به صورت زیر است :

Algorithm 3.4: (Pattern Matching). The pattern matching table $F(Q_1, T)$ of a pattern P is in memory, and the input is an N-character string $T = T_1T_2 \cdots T_N$. This algorithm finds the INDEX of P in T.

1. [Initialize.] Set $K := 1$ and $S_1 = Q_0$.
2. Repeat Steps 3 to 5 while $S_K \neq P$ and $K \leq N$.
3. Read T_K.
4. Set $S_{K+1} := F(S_K, T_K)$. [Finds next state.]
5. Set $K := K + 1$. [Updates counter.]

[End of Step 2 loop.]

6. [Successful?]
   If $S_K = P$, then:
   INDEX = K − LENGTH(P).
   Else:
   INDEX = 0.
   [End of If structure.]
7. Exit.

زمان اجرای الگوریتم بالا با تعداد دفعات اجرای حلقهٔ مرحله **2** متناسب است. بدترین حالت وقتی

اتفاق می‌افتد که تمام متن T خوانده شود یعنی هنگامی که حلقه **n = LENGTH(T)** بار اجرا شود. بنابراین پیچیدگی الگوریتم بالا را می‌توان به صورت زیر بیان کرد :

پیچیدگی این الگوریتم تطبیق متن برابر **O(n)** *است.*

توجه کنید : یک مسأله ترکیباتی رابا **زمان چندجمله‌ای، حل‌پذیر گویند اگر به ازای** بعضی ازمقادیر **m** یک جواب الگوریتمی با پیچیدگی‌ای برابر $O(n^m)$ ، وجود داشته باشد و با **زمان خطی، حل‌پذیر گویند** اگر یک جواب الگوریتمی با پیچیدگی‌ای برابر **O(n)** وجود داشته باشد که در آن **n** تعداد داده‌ها است. بدین‌ترتیب حالت دوم دو الگوریتم تطبیق الگو که در این بخش تشریح شده است با زمان خطی حل‌پذیر هستند. الگوریتم اول تطبیق الگو با زمان چندجمله‌ای حل‌پذیر است.

مسأله‌های حل‌شده

اصطلاحات، ذخیرهٔ رشته‌ها

مسأله ۱ـ ۳ : فرض کنید **W** رشتهٔ **ABCD** است. **(الف)** طول **W** را پیداکنید. **(ب)** تمام زیررشته‌های **W** را بنویسید. **(ج)** تمام زیررشته‌های اولیهٔ **W** را بنویسید.

حل : **(الف)** تعداد کاراکترهای رشتهٔ **W** طول آن است، بنابراین طول **W** برابر **4** است.

(ب) هر زیردنباله‌ای از کاراکترهای **W** یک زیررشتهٔ **W** است. در **W** ، **11** زیررشته وجود دارد:

زیررشته‌ها : Λ A, B, C, D, AB, BC, CD, ABC, BCD, ABCD,

طول آنها : 0 1 2 3 4

در اینجا Λ معرف رشتهٔ تهی است.

(ج) زیررشته‌های اولیهٔ عبارتند از **ABCD** ، **ABC** ، **AB** ، **A** ،Λیعنی هم رشتهٔ تهی و هم زیررشته‌های اولیه با**A** شروع می‌شوند.

مسأله ۲ـ ۳ : فرض کنید یک زبان برنامه‌نویسی حداقل از **48**کاراکتر شامل **26** حرف، **10** رقم و حداقل **12**کاراکتر مخصوص استفاده می‌کند. حداقل تعداد بیتها و تعداد بیتهای متداول برای نمایش یک کاراکتر در حافظهٔ کامپیوتر را به‌دست آورید.

حل : از آنجا که $2^5<48<2^6$ ، حداقل یک کد **6** بیتی برای نمایش این **48**کاراکتر موردنیاز است. معمولاً یک کامپیوتر از یک کُد **7** بیتی نظیر کُد **ASCII** یا یک کد **8** بیتی نظیر کد **EBCDIC** برای نمایش کاراکترها استفاده می‌کند. این کدگذاری‌ها به کامپیوتر اجازه می‌دهند کاراکترهای مخصوص بسیار زیادتری را در

حافظه ایجاد کرده به نمایش و پردازش آنها بپردازند.

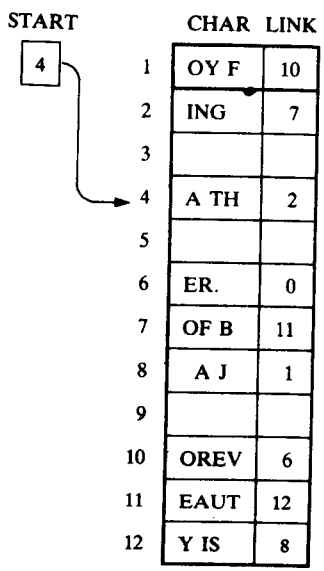
مسأله ۳ـ ۳ : به اختصار سه نوع ساختار مختلف را که برای ذخیرهٔ رشته‌ها بکار می‌رود توضیح دهید.

حل : (الف) *ساختارهای حافظه با طول ثابت.* در اینحالت رشته‌هایی که در خانه‌های حافظه ذخیره می‌شود همگی دارای طول مساوی هستند، معمولاً طول حافظه برای رشته‌ها **80** کاراکتر است.

(ب) *حافظه با طول متغیر که ماگزیمم طول آن ثابت است.* در اینحالت، رشته‌ها نیز در خانه‌های حافظه ذخیره می‌شوند همگی دارای طول مساوی هستند، بنابراین باید طول واقعی رشتهٔ داخل حافظه معلوم باشد.

(ج) *حافظه به صورت لیست پیوندی.* در اینجا هر خانهٔ حافظه به دو قسمت تقسیم می‌شود، در قسمت اول یک کاراکتر (یا تعداد ثابت کوچکی از کاراکتر) ذخیره می‌شود و قسمت دوم شامل آدرس حافظهٔ کاراکتر بعدی است.

مسأله ۴ـ ۳ : رشته ذخیره شده در شکل ۹ـ ۳ را تعیین کنید. فرض می‌شود مقدار پیوند **0** علامت پایان لیست پیوندی است.



شکل ۹ـ ۳

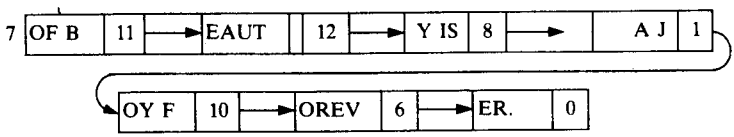
حل : در اینجا رشته در یک ساختمان لیست پیوندی ذخیره می‌شود که هر گرهٔ آن شامل **4** کاراکتر است. مقدار **START**، مکان گرهٔ اول لیست را به دست می‌دهد :

4	A TH	2

مقدار پیوند در این گره، مکان گرهٔ بعدی لیست را به دست می‌دهد :

2	ING	7

اگر این روش را ادامه بدهیم، دنباله گره‌های زیر به دست می‌آید :



بنابراین رشته ذخیره‌شده عبارت است از :

A THING OF BEAUTY IS A JOY FOREVER.

مسأله ۵ـ ۳ : برخی از (الف) مزایا (ب) معایب استفاده از حافظه پیوندی برای ذخیرهٔ رشته‌ها را بیان کنید.

حل : (الف) هنگام استفاده از حافظهٔ پیوندی به راحتی می‌توان عملیات اضافه‌کردن، حذف‌کردن، اتصال و جابجایی زیررشته‌ها را انجام داد.

(ب) برای ذخیرهٔ پیوندها به حافظه اضافی نیاز است. علاوه براین به کاراکتر وسط لیست مستقیماً نمی‌توان دسترسی پیدا کرد.

مسأله ۶ـ ۳ : به اختصار دربارهٔ معنی (الف) متغیرهای کاراکتری ایستا (ب) نیمه‌ایستا (ج) پویا توضیح دهید.

حل : (الف) طول این نوع متغیرها قبل از اجرای برنامه تعریف می‌شود و نمی‌توان طی اجرای برنامه طول آنها را تغییر داد.

(ب) طول این نوع متغیرها می‌تواند طی اجرای برنامه تغییر کند، اما طول متغیر نمی‌تواند از یک مقدار ماگزیممی که قبل از اجرای برنامه تعریف می‌شود بیشتر شود.

(ج) طول این نوع متغیرها می‌تواند طی اجرا برنامه، تغییر کند.

مسأله ۷ـ ۳ : فرض کنید **MEMBER** یک متغیر کاراکتری با طول ثابت **20** است. فرض کنید یک رشته به صورت چیده شده از چپ در خانهٔ حافظه ذخیره می‌شود که فضاهای خالی، طرف راست حافظه را پر

می‌کنند یا سمت راست‌ترین کاراکترها بریده می‌شوند. MEMBER را تحت هر یک از شرایط زیر شرح دهید (الف) اگر 'JOHN PAUL JONES' در MEMBER جایگزین شده باشد (ب) اگر 'ROBERT ANDREW WASHINGTON' در MEMBER جایگزین شده باشد.

حل : داده‌ها به صورت زیر در MEMBER نمایش داده می‌شوند :

(الف)

J	O	H	N		P	A	U	L		J	O	N	E	S					

(ب)

R	O	B	E	R	T		A	N	D	R	E	W		W	A	S	H	I	N

عملیات بر روی رشته‌ها

در مسأله‌های ۸ ـ ۳ تا ۱۱ ـ ۳ و ۱۳ ـ ۳ فرض کنید S و T متغیرهای کاراکتری باشند، به‌گونه‌ای که :

```
S = 'JOHN PAUL JONES'
T = 'A THING OF BEAUTY IS A JOY FOREVER.'
```

مسأله ۸ـ ۳ : یادآور می‌شویم که برای طول یک رشته از LENGTH(string) استفاده می‌کنیم.

(الف) این تابع در زبانهای (i) PL/ I ، (ii) BASIC ، (iii) UCSD PASCALL ، (iv) SNOBOL و (v) FORTRAN چگونه نمایش داده می‌شود؟

(ب) مطلوب است تعیین LENGTH(S) و LENGTH(T) .

حل : (الف) (i) LENGTH(string) . (ii) LEN(string) . (iii) LENGTH(string) . (iv) SIZE(string).

(v) FORTRAN هیچ تابعِ طول برای رشته‌ها ندارد. چون این زبان تنها از متغیرهایی با طول ثابت استفاده می‌کند.

(ب) فرض می‌کنیم بین کلمات تنها یک کاراکتر فضای خالی وجود دارد. از این‌رو

LENGTH(S) = 15 و LENGTH(T) = 35

مسأله ۹ـ ۳ : یادآور می‌شویم که برای نمایش زیررشتهٔ یک رشته که از مکان معینی شروع شده و دارای طول مشخص است از SUBSTRING(string, position, length) استفاده می‌کنیم.

مطلوب است تعیین (الف) SUBSTRINGS(S, 4, 8) و (ب) SUBSTRING(T, 10, 5) .

حل : (الف) با شروع از کاراکتر چهارم و ثبت 8 کاراکتر، داریم :

SUBSTRING(S, 4, 8) = 'N□PAUL□J'

(ب) به طور مشابه : SUBSTRING(T, 10, 5) = 'F□BEAU'

مسأله ۱۰ـ ۳ : یادآور می‌شویم که برای نمایش مکانی که یک الگو برای نخستین بار در متن T ظاهر می‌شود. از INDEX(TEXT, PATTERN) استفاده می‌کنیم. در این تابع مقدار 0 جایگزین می‌شود اگر الگو در متن ظاهر نشده باشد. مطلوب است تعیین

(الف) INDEX(S, 'JO'), (ب) INDEX(S, 'JOY'), (ج) INDEX(S, 'JO□'),

(د) INDEX(T, 'A'), (ه) INDEX(T, 'A□') و (و) INDEX(T, 'THE')

حل: (الف) INDEX(S, 'JO') = 1, (ب) INDEX(S, 'JOY') = 0, (ج) INDEX(S, 'JO□') = 10,

(د) INDEX(T, 'A') = 1, (ه) INDEX(T, 'A□') = 21 و (و) INDEX(T, 'THE') = 0

یادآوری می‌کنیم که □ برای نمایش یک فضای خالی مورد استفاده قرار می‌گیرد.

مسأله ۱۱ـ ۳ : یادآور می‌شویم که برای نمایش اتصال دو رشتهٔ S_1 و S_2 از S_2 // S_1 استفاده می‌کنیم.

(الف) این تابع در زبانهای (i) PL/I (ii) FORTRAN (iii) BASIC (iv) SNOBOL و (v) UCSD PASCAL چگونه نمایش داده می‌شود؟

(ب) مطلوب است تعیین (i) 'END' // 'THE' و (ii) 'END' // ' □ ' // 'THE' .

(ج) مطلوب است تعیین (i) SUBSTRING(S, 1, 9) // '□,' // SUBSTRING(S, 11, 5) و

(ii) 'GIVEN' // SUBSTRING(T, 28, 3) .

حل : (الف) (i) $S_1 \parallel S_2$, (ii) S_1 // S_2, (iii) $S_1 + S_2$, (iv) S_1 S_2 (بین S_1 و S_2 یک فضای خالی وجود دارد) (v) $CONCAT(S_1, S_2)$

(ب) منظور از S_2 // S_1 رشته‌ای است که از کاراکترهای S_1 و بدنبال آن کاراکترهای S_2 تشکیل شده باشد. از این‌رو (i) THEEND (ii) THE END .

(ج) (i) JONES, JOHN PAUL (ii) FORGIVEN .

مسألهٔ ۱۲ـ ۳ : یادآور می‌شویم که برای نمایش اضافه‌کردن یک رشتهٔ S در یک متن داده شدهٔ T با شروع از مکان K از INSERT(text, position, string) استفاده می‌کنیم.

(الف) مطلوب است تعیین (i) INSERT('AAAAA', 1, 'BBB'),

(ii) و INSERT('AAAAA', 3, 'BBB') (iii) INSERT('AAAAA', 6, 'BBB')

(ب) فرض کنید T متن 'THE STUDENT IS ILL' باشد. با استفاده از INSERT ، T را طوری تغییر دهید که (i) The student is very ill (ii) The student is ill today

و (iii) The student is very ill today را بخوانند.

حل : (الف) (i) BBBAAAAA, (ii) AABBBAAA و (iii) AAAAABBB .

(ب) دقت کنید در جاهای لازم فضاهای خالی بگنجانید. (i) INSERT(T,15, '□ VERY')

(ii) INSERT(T, 19, '□ TODAY') یا INSERT(INSERT(T,19, '□ TODAY'),15, '□ VERY') (iii)

. INSERT(INSERT(T, 15, '□ VERY'), 24, '□ TODAY')

مسأله ۱۳ـ ۳ : مطلوب است تعیین

(الف) DELETE('AAABBB', 2, 2) و DELETE('JOHN PAUL JONES', 6, 5)

(ب) REPLACE('AAABBB', 'AA', 'BB') و

REPLACE('JOHN PAUL JONES', 'PAUL', 'DAVID')

حل : (الف) DELETE(T, K, L) از متن T زیررشته‌ای را حذف می‌کند که از مکان K شروع شده طول L دارد. از این‌رو جوابها عبارتند از ABBB و JOHN JONES .

(ب) REPLACE(T, P_1, P_2) در متن T اولین وقوع الگوی P_1 توسط الگوی P_2 جانشین می‌شود. از این‌رو جوابها عبارتند از BBABBB و JOHN DAVID JONES .

پردازش کلمه

در مسأله‌های ۱۴ـ ۳ تا ۱۷ـ ۳ ، S یک داستان کوتاه است که در یک آرایهٔ خطی LINE با n عنصر به‌گونه‌ای ذخیره شده است که در آن هر LINE[K] یک متغیر کاراکتری ایستا با قدرت ذخیره‌سازی 80 کاراکتر است و یک خط از داستان را نمایش می‌دهد. همچنین LINE[1] یعنی اولین خط داستان تنها شامل عنوان داستان است و LINE[N] آخرین خط داستان تنها شامل نام نویسنده است. علاوه براین، هر پاراگراف با 5 فضای خالی شروع می‌شود و در هیچ کجای داستان بجز احتمالاً در عنوان LINE[1] یا نام نویسنده LINE[N] هیچ تورفتگی مانند ابتدای پاراگراف وجود ندارد.

مسأله ۱۴ـ ۳ : یک زیربرنامهٔ Procedure بنویسید تا NUM تعداد پاراگرافهای داستان کوتاه S را بشمارد.

حل : از خط LINE[2] شروع می‌کنیم و با خط LINE [N –1] شمارش تعداد خطوط را که با 5 فضای خالی شروع می‌شود به پایان می‌بریم. زیربرنامه به صورت زیر است :

```
Procedure P3.14:  PAR(LINE, N, NUM)
    1.  Set NUM := 0 and BLANK := '□□□□□'.
    2.  [Initialize counter.] Set K := 2.
    3.  Repeat Steps 4 and 5 while K ≤ N – 1.
    4.      [Compare first 5 characters of each line with BLANK.]
            If SUBSTRING(LINE[K], 1, 5) = BLANK, then:
                Set NUM := NUM + 1.
            [End of If structure.]
    5.      Set K := K + 1. [Increments counter.]
        [End of Step 3 loop.]
    6.  Return.
```

مسأله ۱۵ـ ۳ : یک زیربرنامهٔ **procedure** بنویسید تا **NUM** تعداد دفعاتی را که کلمهٔ **the** در این داستان کوتاه **S** ظاهر می‌شود بشمارد. **the** در کلمهٔ **mother** را در شمارش شرکت ندهید و فرض می‌شود که هیچ جمله‌ای با کلمهٔ **the** پایان نمی‌یابد.

حل : توجه دارید که کلمهٔ **the** می‌تواند به صورت □ **THE** در آغاز یک خط، به صورت **THE** □ در پایان یک خط یا به صورت □ **THE** □ در هر جای دیگری از یک خط می‌تواند ظاهر شود. از این‌رو برای هر خط باید این سه حالت را مورد توجه قرار دهیم. زیربرنامه به صورت زیر است :

Procedure P3.15: COUNT(LINE, N, NUM)

```
1. Set WORD := 'THE' and NUM := 0.
2. [Prepare for the three cases.]
   Set BEG := WORD// '□', END := '□' // WORD and
   MID := '□' // WORD// '□'.
3. Repeat Steps 4 through 6 for K = 1 to N:
4.      [First case.] If SUBSTRING(LINE[K], 1, 4) = BEG, then:
            Set NUM := NUM + 1.
5.      [Second case.] If SUBSTRING(LINE[K], 77, 4) = END, then:
            Set NUM := NUM + 1.
6.      [General case.] Repeat for J = 2 to 76.
            If SUBSTRING(LINE[K], J, 5) = MID, then:
                Set NUM := NUM + 1.
            [End of If structure.]
        [End of Step 6 loop.]
    [End of Step 3 loop.]
7. Return.
```

مسأله ۱۶ـ ۳ : چنانچه بخواهیم تعداد دفعات وقوع کلمهٔ دلخواهی مانند **W** را با طول **r** بشماریم توضیح دهید چه تغییراتی باید در **Procedure P3.15** بدهیم.

حل : لازم است سه نوع تغییر اصلی در این زیربرنامه داده شود.

(الف) واضح است که در مرحلهٔ **1** باید **THE** به **W** تغییر داده شود.

(ب) از آنجا که طول **W** برابر **r** است و نه **3** از این‌رو باید تغییرات مناسب در مراحل **3** تا **6** داده شود.

(ج) علاوه براین باید این امکان درنظر گرفته شود که بعد از **W** می‌تواند علامت نقطه‌گذاری نظیر

W, W; W. W?

باشد. از این‌رو بیشتر از سه حالت باید مورد بررسی قرار گیرد.

مسأله ۱۷ـ ۳ : الگوریتمی را شرح دهید که پاراگرافهای **K** ام و **L** ام داستان کوتاه **S** را جابجا می‌کند.

حل : الگوریتم را به دو زیربرنامهٔ **Procedure** تجزیه می‌کنیم.

زیربرنامهٔ **Procedure A** ، **A** : مقادیر آرایه‌های **BEG** و **END** را پیدا می‌کند که در آن

LINE[BEG[K]] و **LINE[END[K]]**

به ترتیب شامل اولین و آخرین خط پاراگراف K ی داستان S است.

زیربرنامهٔ B ، Procedure B : با استفاده از مقادیر BEG[K] و END[K] و مقادیر BEG[L] و END[L] بلاک خطوط پاراگراف K با بلاک خطوط پاراگراف L جابجا می‌شود.

تطبیق الگو

مسأله ۱۸ـ ۳ : برای هر یک از الگوهای P و متنهای T زیر، C تعداد مقایسه‌هایی را پیدا کنید که با آن INDEX اندیس P در T با استفاده از الگوریتم "آرام" ، الگوریتم ۳ـ ۳ به دست می‌آید.

(الف) $P = abc,\ T = (ab)^5 = ababababab$ (ج) $P = aaa,\ T = (aabb)^3 = aabbaabbaabb$

(ب) $P = abc,\ T = (ab)^{2n}$ (د) $P = aaa,\ T = abaabbaaabbbaaaabbbb$

حل : خاطرنشان می‌کنیم که $C = N_1 + N_2 + \ldots + \ldots + N_L$ که در آن N_k تعداد مقایسه‌هایی را نشان می‌دهد که هنگام مقایسهٔ P با W_k در حلقهٔ داخلی انجام می‌شود.

(الف) نخست توجه داشته باشید که تعداد

$$\text{LENGTH(T)} - \text{LENGTH(P)} + 1 = 10 - 3 + 1 = 8$$

زیررشتهٔ W_k وجود دارد. داریم :

$$C = 2 + 1 + 2 + 1 + 2 + 1 + 2 + 1 = 4(3) = 12$$

و INDEX(T, P) = 0 ، چون P در T ظاهر نشده است.

(ب) تعداد $2n - 3 + 1 = 2(n - 1)$ زیرکلمهٔ W_k وجود دارد. داریم :

$$C = 2 + 1 + 2 + 1 + \ldots + 2 + 1 = (n + 1)(3) = 3n + 3$$

و INDEX(T, P) = 0

(ج) تعداد $12 - 3 + 1 = 10$ زیرکلمهٔ W_k وجود دارد. داریم :

$$C = 3 + 2 + 1 + 1 + 3 + 2 + 1 + 1 + 3 + 2 = 19$$

و INDEX(T, P) = 0

(د) داریم :

$$C = 2 + 1 + 3 + 2 + 1 + 1 + 3 = 13$$

و INDEX(T, P) = 7

مسأله ۱۹ـ ۳ : فرض کنید P یک رشتهٔ r کاراکتری و T یک رشتهٔ s کاراکتری باشد و فرض کنید هنگامی که الگوریتم ۳ـ۳ بر P و T اعمال می‌شود C(n) نمایش تعداد مقایسه‌های صورت گرفته باشد. در اینجا $n = r + s$.

(الف) $C(n)$ پیچیدگی بهترین حالت را پیدا کنید.

(ب) ثابت کنید مقدار ماگزیمم $C(n)$ وقتی اتفاق می‌افتد که $r = (n+1) / 4$.

حل : (الف) بهترین حالت وقتی اتفاق می‌افتد که P یک زیررشتهٔ اولیهٔ T باشد یا به عبارت دیگر وقتی که $INDEX(T, P) = 1$ در این حالت $C(n) = r$. (فرض می‌کنیم $r \leq s$.)

(ب) بنا به بحث بخش ۷ـ۳،

$$C = C(n) = r(n - 2r + 1) = nr - 2r^2 + r$$

در اینجا n ثابت است از این‌رو $C = C(n)$ را می‌توان به صورت تابعی از r درنظر گرفت. حساب دیفرانسیل و انتگرال می‌گوید ماگزیمم مقدار C وقتی اتفاق می‌افتد که $C' = dC/dr = 0$ در اینجا C' مشتق C نسبت به r است. با استفاده از حساب دیفرانسیل و انتگرال به دست می‌آید :

$$C' = n - 4r + 1$$

با قراردادن $C' = 0$ و حل آن نسبت به r نتیجهٔ موردنظر به‌دست می‌آید.

مسألهٔ ۲۰ـ۳ : الگوی $P = aaabb$ را درنظر بگیرید. جدول و گراف جهت‌دار برچسب‌گذاری شدهٔ متناظر با آن را که در الگوریتم "سریع" یا الگوریتم دوم تطبیق الگو مورد استفاده قرار گرفت، رسم کنید.

حل : نخست لیست قطعه‌های اولیهٔ P را می‌نویسیم :

$$Q_0 = \Lambda, \quad Q_1 = a, \quad Q_2 = a^2, \quad Q_3 = a^3, \quad Q_4 = a^3b, \quad Q_5 = a^3b^2$$

برای هر کاراکتر t ، درایهٔ $f(Q_i, t)$ جدول بزرگترین Q ای است که به صورت یک زیررشتهٔ انتهایی در رشتهٔ $Q_i t$ ظاهر می‌شود. محاسبه می‌کنیم :

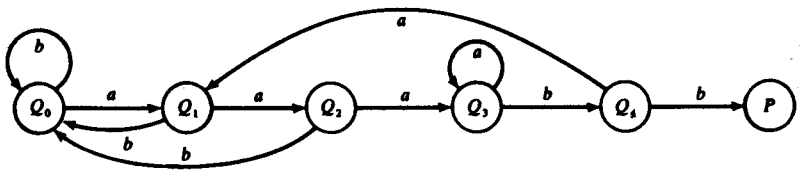
$$f(\Lambda, a) = a, \quad f(a, a) = a^2, \quad f(a^2, a) = a^3, \quad f(a^3, a) = a^3, \quad f(a^3b, a) = a$$
$$f(\Lambda, b) = \Lambda, \quad f(a, b) = \Lambda, \quad f(a^2, b) = \Lambda, \quad f(a^3, b) = a^3b, \quad f(a^3b, b) = P$$

از این‌رو جدول موردنظر در شکل ۱۰ـ۳ (الف) ارائه شده است.

	a	b
Q_0	Q_1	Q_0
Q_1	Q_2	Q_0
Q_2	Q_3	Q_0
Q_3	Q_3	Q_4
Q_4	Q_1	P

شکل ۱۰ـ۳ (الف)

گراف متناظر با آن در شکل ۱۰ـ۳ (ب) رسم شده است که در آن برای هر **Q** یک گره وجود دارد و پیکان از Q_i و Q_j برای هر درایهٔ $f(Q_i, t) = Q_j$ داخل جدول با کاراکتر **t** برچسب‌گذاری شده است.



شکل ۱۰ـ۳ (ب)

مسأله ۲۱ـ۳: جدول و گراف متناظر با آن را برای الگوریتم دوم تطبیق الگو که در آن **P = ababab** الگو است رسم کنید.

حل: زیررشته‌های اولیهٔ **P** عبارتند از:

$$Q_0 = \Lambda, \quad Q_1 = a, \quad Q_2 = ab, \quad Q_3 = aba, \quad Q_4 = abab, \quad Q_5 = ababa, \quad Q_6 = ababab = P$$

تابع f درایه‌های جدول را به صورت زیر به دست می‌دهد:

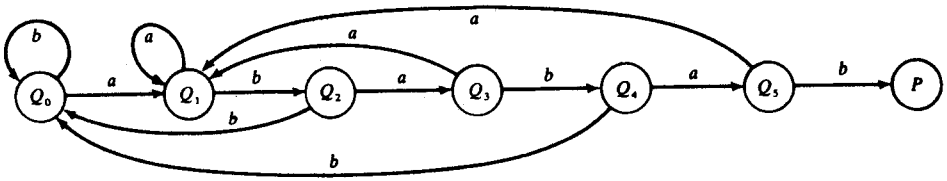
$f(\Lambda, a) = a$
$f(a, a) = a$
$f(ab, a) = aba$
$f(aba, a) = a$
$f(abab, a) = ababa$
$f(ababa, a) = a$

$f(\Lambda, b) = \Lambda$
$f(a, b) = ab$
$f(ab, b) = \Lambda$
$f(aba, b) = abab$
$f(abab, b) = \Lambda$
$f(ababa, b) = P$

جدول در شکل ۱۱ـ۳ (الف) و گراف متناظر با آن در شکل ۱۱ـ۳ (ب) آمده است.

	a	b
Q_0	Q_1	Q_0
Q_1	Q_1	Q_2
Q_2	Q_3	Q_0
Q_3	Q_1	Q_4
Q_4	Q_5	Q_0
Q_5	Q_1	P

(الف)



(ب)

شکل ۱۱ـ۳

مسأله‌های تکمیلی

رشته‌ها

مسأله ۲۲ـ۳ : رشتهٔ ذخیره شده در شکل ۱۲ـ۳ را به دست آورید.

START: 5

	CHAR	LINK
1	UNIT	11
2	HE P	8
3		
4	S	0
5	WE T	2
6	THE	1
7		
8	EQPL	12
9	TATE	4
10		
11	ED S	9
12	E OF	6

شکل ۱۲ـ۳

مسأله ۲۳ـ۳ : رشتهٔ 'XYZST' = W را درنظر بگیرید. **(الف)** تمام زیررشته‌های **W** و

(ب) تمام زیررشته‌های ابتدایی W را بنویسید.

مسأله ۲۴ـ۳ : فرض کنید W یک رشته به طول n است. مطلوب است تعیین تعداد (الف) زیررشته‌های W و (ب) زیررشته‌های ابتدایی W.

مسأله ۲۵ـ۳ : فرض کنید STATE یک متغیر کاراکتری با طول ثابت 12 است. محتوای STATE را پس از جایگزینی‌های زیر توضیح دهید. (الف) STATE := 'NEW YORK',

ب) STATE := 'SOUTH CAROLINA' و (ج) STATE := 'PENNSYLVANIA'

عملیات بر روی رشته‌ها

در مسأله‌های ۲۶ـ۳ تا ۳۱ـ۳ فرض کنید S و T متغیرهای کاراکتری هستند به‌طوری که

S = 'WE THE PEOPLE' و T = 'OF THE UNITED STATES'.

مسأله ۲۶ـ۳ : طول S و T را تعیین کنید.

مسأله ۲۷ـ۳ : مطلوب است تعیین (الف) SUBSTRING(S, 4, 8) و (ب) SUBSTRING(T, 10, 5).

مسأله ۲۸ـ۳ : مطلوب است تعیین (الف) INDEX(S, 'P') (ب) INDEX(S, 'E')

(ج) INDEX(S, 'THE') (د) INDEX(T, 'THE') (ه) INDEX(T, 'THEN') و

(و) INDEX(T, 'TE')

مسأله ۲۹ـ۳ : با استفاده از S_2 // S_1 به عنوان اتصال دو رشتهٔ S_1 و S_2، مطلوب است تعیین

(الف) 'EXIT' // 'NO' (ب) 'EXIT' // ' □ ' // 'NO' و

(ج) SUBSTRING(T, 8, 6) // ' □ ARE □ ' // SUBSTRING(S, 4, 10).

مسأله ۳۰ـ۳ : مطلوب است تعیین (الف) DELETE('AAABBB', 3, 3),

(ب) DELETE('AAABBB', 1, 4), (ج) DELETE(S, 1, 3), و (د) DELETE(T, 1, 7).

مسأله ۳۱ـ۳ : مطلوب است تعیین (الف) REPLACE('ABABAB', 'B', 'BAB'),

(ب) REPLACE(S, 'WE', 'ALL') و (ج) REPLACE(T, 'THE', 'THESE').

مسأله ۳۲ـ۳ : مطلوب است تعیین (الف) INSERT('AAA', 2, 'BBB'),

(ب) INSERT('ABCDE', 3, 'XYZ') و (ج) INSERT('THE BOY', 5, 'BIG □ ').

مسأله ۳۳ـ۳ : فرض کنید U متن 'MARC STUDIES MATHEMATICS' باشد. با استفاده از INSERT، متن U را طوری تغییر دهید که (الف) MARC STUDIES ONLY MATHEMATICS

(ب) MARC STUDIES MATHEMATICS AND PHYSICS (ج) MARC STUDIES APPLIED MATHEMATICS را بخواند.

تطبیق الگو

مسأله ۳۴ـ ۳ : الگوی $P = abc$ را درنظر بگیرید. با استفاده از الگوریتم تطبیق الگو "آرام" یعنی الگوریتم ۳ـ ۳، C تعداد مقایسه‌هایی را پیدا کنید که با آنها INDEX اندیس P در هر یک از متنهای T زیر به دست می‌آید :

(الف) a^{10}، (ب) $(aba)^{10}$، (ج) $(cbab)^{10}$ (د) d^{10} و (هـ) d^n که در آن $n > 3$ است.

مسأله ۳۵ـ ۳ : الگوی $P = a^5b$ را درنظر بگیرید. مسأله ۳۴ـ ۳ را با هر یک از متنهای T زیر مجدداً حل کنید :

(الف) a^{20} (ب) a^n که در آن $n > 6$ (ج) d^{20} و (د) d^n که در آن $n > 6$ است.

مسأله ۳۶ـ ۳ : الگوی $P = a^3ba$ را درنظر بگیرید. جدول و گراف جهت‌دار برچسب‌گذاری شدهٔ متناظر با آن را که در الگوریتم تطبیق الگو "سریع" مورد استفاده قرار گرفت رسم کنید.

مسأله ۳۷ـ ۳ : مسأله ۳۶ـ ۳ را برای الگوی $P = aba^2b$ مجدداً حل کنید.

برای مسأله‌های زیر برنامه بنویسید

در مسأله‌های ۳۸ـ ۳ تا ۴۰ـ ۳، فرض می‌شود که پیشگفتار این متن در یک آرایهٔ خطی LINE ذخیره شده است به‌طوری که LINE[K] یک متغیر کاراکتری ایستا با قدرت ذخیرهٔ 80 کاراکتر است و یک خط از پیشگفتار را نمایش می‌دهد. فرض کنید هر پاراگراف با 5 فضای خالی شروع می‌شود و در هیچ جای دیگر پیشگفتار تورفتگی وجود ندارد. علاوه بر این فرض می‌شود یک متغیر NUM وجود دارد که تعداد خطوط پیشگفتار را به دست می‌دهد.

مسأله ۳۸ـ ۳ : یک برنامه بنویسید که یک آرایهٔ خطی PAR را معین می‌کند به گونه‌ای که PAR[K] حاوی مکان K ام پاراگراف است و علاوه بر این یک متغیر NPAR را معین می‌کند که حاوی تعداد پاراگرافها است.

مسأله ۳۹ـ ۳ : یک برنامه بنویسید که یک کلمه WORD داده شده را بخواند و C تعداد دفعاتی را که WORD در LINE ظاهر شده است را بشمارد. برنامه را با استفاده از (الف) 'THE' = WORD و (ب) 'HENCE' = WORD آزمایش کنید.

مسأله ۴۰ـ ۳ : یک برنامه بنویسید که پاراگرافهای J ام و K ام را جابجا کند. برنامه را با استفاده از $J = 2$ و $K = 2$ آزمایش کنید.

در مسأله‌های ۴۱ـ ۳ تا ۴۶ـ ۳ فرض می‌شود که پیشگفتار این متن در یک متغیر کاراکتری TEXT ذخیره شده است. فرض کنید 5 فضای خالی مبین یک پاراگراف جدید است.

مسأله ۴۱ـ ۳: یک برنامه بنویسید که یک آرایهٔ خطی **PAR** را به‌گونه‌ای ایجاد کند که **PAR[K]** حاوی مکان پاراگراف **K** ام در **TEXT** باشد همچنین مقدار یک متغیر **NPAR** را پیدا کند که حاوی تعداد پاراگرافها است. این مسأله را با مسأله ۳۸ـ ۳ مقایسه کنید.

مسأله ۴۲ـ ۳: یک برنامه بنویسید که یک کلمه **WORD** داده شده را بخواند و آنگاه **C** تعداد دفعاتی را که **WORD** در **TEXT** ظاهر شده است را بشمارد. برنامه را با استفاده از (الف) **WORD = 'THE'** و (ب) **WORD = 'HENCE'** آزمایش کنید. این مسأله را با مسأله ۳۹ـ ۳ مقایسه کنید.

مسأله ۴۳ـ ۳: یک برنامه بنویسید که پاراگرافهای **J** ام و **K** ام متن **TEXT** را جابجا کند. این برنامه را با استفاده از **J = 2** و **K = 4** آزمایش کنید. این مسأله را با مسأله ۴۰ـ ۳ مقایسه کنید.

مسأله ۴۴ـ ۳: یک برنامه بنویسید که کلمه‌های **WORD1** و **WORD2** را بخوانید و آنگاه هر وقوع **WORD1** در **TEXT** توسط **WORD2** جایگزین گردد. برنامه را با استفاده از **WORD1 = 'HENCE'** و **WORD2 = 'THUS'** آزمایش کنید.

مسأله ۴۵ـ ۳: یک زیربرنامهٔ **INST(TEXT, NEW, K)** بنویسید که رشتهٔ **NEW** را در داخل متن **TEXT** در آغاز **TEXT [K]** اضافه کند.

مسأله ۴۶ـ ۳: یک زیربرنامه **PRINT(TEXT, K)** بنویسید که رشتهٔ کاراکتری **TEXT** را در خطوطی با حداکثر **K** کاراکتر چاپ کند. هیچ کلمه‌ای به دو قسمت تقسیم نشده و در دو خط ظاهر نمی‌شود، از اینرو برخی از خطوط می‌توانند شامل چند فضای خالی در انتهایشان باشند. هر پاراگراف با خط مربوطه‌اش شروع شده و به کمک 5 فضای خالی تورفتگی دارد. برنامه را با استفاده از (الف) **K = 80** (ب) **K = 70** و (ج) **K = 60** آزمایش کنید.

فصل ۴

آرایه‌ها، رکوردها و اشاره‌گرها

۱ـ۴ مقدمه

ساختمان داده‌ها به دو دستهٔ خطی و غیرخطی تقسیم می‌شود. یک ساختمان داده را **خطی** گویند، هرگاه عناصر آن تشکیل یک دنباله دهند، به بیان دیگر یک لیست خطی باشد. برای نمایش ساختمان دادهٔ خطی در حافظه، دو روش *اساسی* وجود دارد. یکی از این روشها، عبارت است از داشتن رابطهٔ خطی بین عناصری که به وسیلهٔ خانه‌های متوالی حافظه نمایش داده می‌شود. این ساختارهای خطی **آرایه‌ها** نام دارند که موضوع اصلی این فصل را تشکیل می‌دهد. روش دیگر عبارت است از داشتن رابطهٔ خطی بین عناصری که به‌وسیلهٔ **اشاره‌گرها** یا **پیوندها** نمایش داده می‌شود. این ساختارهای خطی **لیستهای پیوندی** نام دارند که موضوع اصلی مطالب فصل ۵ را تشکیل می‌دهد. *ساختارهای غیرخطی* نظیر درختها و گرافها در فصل‌های بعد مورد مطالعه قرار می‌گیرد.

عملیاتی که معمولاً بر روی یک ساختار خطی انجام می‌شود خواه این ساختار آرایه باشد یا یک لیست پیوندی، شامل عملیات زیر است :

(الف) پیمایش. پردازش هر عنصر داخل لیست را **پیمایش** گویند.

(ب) جستجوکردن. پیداکردن مکان یک عنصر با یک مقدار داده‌شده یا رکورد با یک کلید معین را جستجو کردن گویند.

(ج) اضافه‌کردن. افزودن یک عنصر جدید به لیست را *اضافه* کردن گویند.

(د) حذف‌کردن. حذف یک عنصر از لیست را *حذف* کردن گویند.

(ه) مرتب‌کردن. تجدید آرایش عناصر با یک نظم خاص را مرتب کردن گویند.

(و) ادغام‌کردن. ترکیب دو لیست در یک لیست را *ادغام* کردن گویند.

ساختار خطی خاصی که برای یک وضعیت معین انتخاب می‌شود بستگی به تعداد دفعاتی دارد که عملیات مختلف بالا روی ساختار اجرا می‌شود.

این فصل یک ساختار خطی کاملاً متداولی را مورد بررسی قرار می‌دهد که **آرایه** نام دارد. از آنجا که پیمایش، جستجو و مرتب‌کردن آرایه معمولاً ساده است از آنها اغلب برای ذخیرهٔ مجموعه‌هایی از داده‌های نسبتاً دائمی استفاده می‌شود. از سوی دیگر، اگر اندازهٔ ساختار و داده‌های آن پیوسته درحال تغییر باشد آنگاه آرایه نمی‌تواند به خوبی ساختاری نظیر لیست پیوندی باشد که در فصل ۵ بررسی می‌شود.

۲ـ۴ آرایه‌های خطی

یک **آرایهٔ خطی** لیستی از **n** یا تعداد متناهی عنصر داده‌ای **همجنس** است (یعنی عناصر داده‌ای از یک نوع هستند) بطوری که :

(الف) به عناصر آرایه به‌ترتیب به کمک یک مجموعه از اندیسها که شامل **n** عدد متوالی است رجوع می‌شود.

(ب) عناصر آرایه به ترتیب در خانه‌های متوالی حافظهٔ ذخیره می‌شوند.

n یا تعداد عناصر آرایه **طول** یا **اندازهٔ** آرایه نام دارد. اگر به‌صراحت بیان نشود فرض می‌کنیم که مجموعهٔ اندیس‌ها شامل اعداد صحیح 1 ، 2 ، ... **n** است. در حالت کلی، طول یا تعداد عناصر داده‌ای آرایه را می‌توان از مجموعهٔ اندیسها به کمک فرمول زیر به دست آورد:

$$\text{طول} = UB - LB + 1 \qquad (۴-۱)$$

که در آن **UB** **بزرگترین اندیس** یا **کران بالای آرایه** و **LB** **کوچکترین اندیس** یا **کران پائین آرایه** است.

توجه دارید که وقتی $LB = 1$ باشد طول آرایه برابر **UB** است.

عناصر یک آرایهٔ **A** را می‌توان با نماد اندیس‌گذاری

$$A_1, A_2, A_3, \ldots, A_n$$

یا با نماد پرانتزگذاری (که در **FORTRAN** ، **PL / I** و **BASIC** بکار می‌رود)

A(1), A(2),..., A(N)

یا با نماد کروشه‌ای (که در **PASCAL** بکار می‌رود)

A[1], A[2], A[3],..., A[N]

نمایش داد. ما معمولاً از نماد اندیس‌گذاری یا نماد کروشه‌ای استفاده می‌کنیم. عدد **K** در **A[K]** **زیرنویس** یا **اندیس** و **A[K]** یک **متغیر اندیس‌دار** نام دارد. توجه دارید که اندیس‌ها اجازه می‌دهند به مکان نسبی هر عنصر **A** دسترسی پیدا کنیم.

مثال ۱ـ ۴

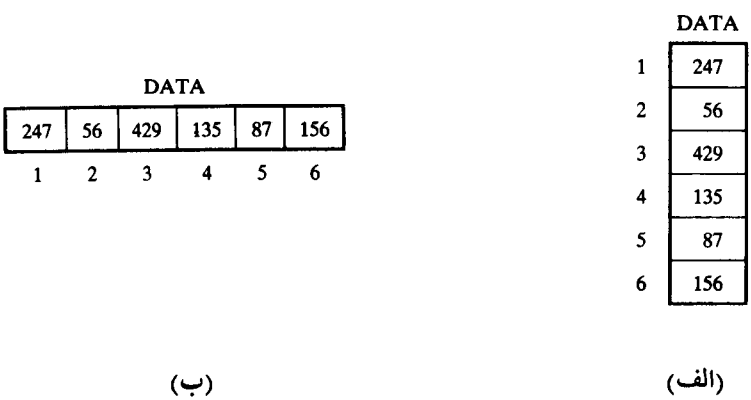
(الف) فرض کنید **DATA** یک آرایهٔ خطی **6** عنصری از اعداد صحیح باشد به‌طوری که

DATA[1] = 247 DATA[2] = 56 DATA[3] = 429 DATA[4] = 135 DATA[5] = 87 DATA[6] = 156

گاهی اوقات ما این آرایه را فقط با نوشتن

DATA: 247, 56, 429, 135, 87, 156

نمایش می‌دهیم. آرایهٔ **DATA** غالباً به صورت شکل ۱ـ ۴ (**الف**) یا شکل ۱ـ ۴ (**ب**) نمایش داده می‌شود.



شکل ۱ـ ۴

(**ب**) یک شرکت فروشنده اتومبیل از یک آرایهٔ **AUTO** برای ثبت تعداد اتومبیل‌های فروخته شدهٔ هرسال از **1932** تا **1984** استفاده می‌کند. بجای اینکه مجموعهٔ اندیسها از **1** شروع شود، بهتر است از **1932** شروع شود، طوری که :

AUTO[K] = تعداد اتومبیل‌های فروخته‌شده در سال K

آنگاه LB = 1932 کران پائین و UB = 1984 کران بالای آرایهٔ AUTO است.

با توجه به فرمول (۱ـ ۴)

$$\text{طول} = UB - LB + 1 = 1984 - 1930 + 1 = 55$$

یعنی AUTO شامل 55 عنصر است و مجموعهٔ اندیسی آن شامل تمام اعداد صحیح از 1932 تا 1984 است.

هر زبان برنامه‌نویسی قاعده و روش خاصی برای معرفی آرایه‌ها دارد. هر یک از این روشها، به صورت صریح یا ضمنی، سه نوع اطلاعات را در مورد آرایه‌ها به دست می‌دهند که عبارتند از : (۱) *نام آرایه* (۲) *نوع دادهٔ آرایه* و (۳) *مجموعهٔ اندیسهای آرایه*.

مثال ۲ـ ۴

(الف) فرض کنید DATA یک آرایهٔ خطی 6 عنصری از اعداد اعشاری باشد. زبانهای برنامه‌نویسی مختلف زیر این آرایه را به صورت زیر معرفی می‌کنند :

```
FORTRAN:   REAL DATA(6)
PL/1:      DECLARE DATA(6) FLOAT;
Pascal:    VAR DATA: ARRAY[1 . . 6] OF REAL
```

ما این آرایه را در صورت نیاز به صورت DATA(6) معرفی خواهیم کرد. متن مورد بررسی نوع داده را مشخص می‌کند از این‌رو نوع آرایه به صراحت بیان نمی‌شود.

(ب) آرایهٔ صحیح AUTO را با کران پائین LB = 1932 و کران بالای UB = 1984 درنظر بگیرید. زبانهای برنامه‌نویسی مختلف زیر این آرایه را به صورت زیر معرفی می‌کنند :

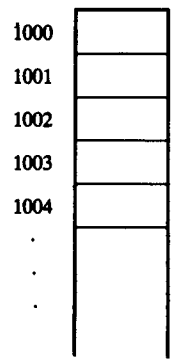
```
FORTRAN 77   INTEGER AUTO(1932:1984)
PL/1:        DECLARE AUTO(1932:1984) FIXED;
Pascal       VAR AUTO: ARRAY[1932 . . 1984] of INTEGER
```

ما این آرایه را به صورت AUTO(1932 : 1984) معرفی خواهیم کرد.

برخی از زبانهای برنامه‌نویسی مانند FORTRAN و PASCAL برای آرایه‌ها به صورت ایستا یعنی در زمان کامپایلِ برنامه حافظه اختصاص می‌دهند. از این‌رو اندازه آرایه در طول اجرای برنامه ثابت است. ازطرف دیگر، برخی از زبانهای برنامه‌نویسی اجازه می‌دهند یک عدد صحیح n خوانده شود و آنگاه آرایهٔ n عنصری را معرفی می‌کنند. در این‌گونه زبانهای برنامه‌نویسی گفته می‌شود حافظه به صورت پویا به برنامه اختصاص می‌یابد.

۳ـ۴ نمایش آرایه‌های خطی در حافظه

فرض کنید LA یک آرایهٔ خطی در حافظهٔ کامپیوتر باشد. یادآوری می‌کنیم که حافظهٔ کامپیوتر فقط یک دنباله از مکانهای دارای آدرس است که در شکل ۲ـ۴ به تصویر کشیده شده است.



شکل ۲ـ۴ـ حافظهٔ کامپیوتر

اجازه دهید از نماد زیر استفاده کنیم:

LOC (LA[K]) = آدرس عنصر LA[K] ی آرایهٔ LA

همانگونه که قبلاً متذکر شدیم، عناصر LA در خانه‌های متوالی ذخیره می‌شوند. بنابراین لازم نیست کامپیوتر آدرس هر عنصر LA را داشته باشد بلکه فقط لازم است آدرس اولین عنصر LA را بداند که ما آن را به صورت زیر نمایش می‌دهیم :

Base(LA)

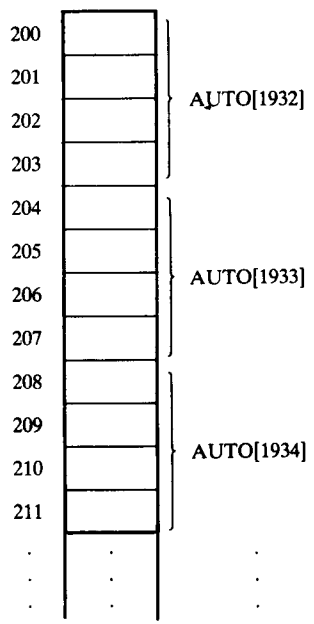
و آن را **آدرس پایه** یا **مبنای LA** می‌نامیم. با استفاده از آدرس پایه Base(LA) ، کامپیوتر آدرس تمام عناصر دیگر آرایهٔ LA را به کمک فرمول زیر محاسبه می‌کند :

$$LOC(LA[K]) = Base(LA) + w(K - \text{lower bound}) \qquad (۴-۲)$$

که در آن w تعداد کلمات موجود در خانهٔ حافظه برای آرایهٔ LA است. ملاحظه می‌کنید که زمان موردنیاز برای محاسبهٔ LOC (LA [K]) اساساً برابر زمان موردنیاز برای هر مقدار دیگر K است. علاوه بر این با معلوم‌بودن اندیس K ، می‌توان بدون جستجوی هر عنصر دیگر LA مکان LA [K] را پیدا کرد و به محتوای آن دسترسی پیدا کرد.

مثال ۳ـ ۴

آرایهٔ **AUTO** مثال ۱ـ ۴ (ب) را درنظر بگیرید که تعداد اتومبیلهای فروخته‌شدهٔ هر سال از **1932** تا **1984** را ثبت می‌کند. فرض کنید **AUTO** در حافظهٔ به صورت شکل ۳ـ ۴ به نمایش در آمده است.



شکل ۳ـ ۴

یعنی **Base(AUTO) = 200** و **W=4** کلمه در خانهٔ حافظه برای **AUTO** باشد، آنگاه

LOC(AUTO[1932]) = 200, LOC(AUTO[1933]) = 204, LOC(AUTO[1934]) = 208, ...

آدرس هر عنصر آرایه برای سال **K=1965** را می‌توان با استفاده از رابطهٔ (۲ـ ۴) به‌دست آورد.

LOC(AUTO[1965]) = Base(AUTO) + w(1965 − lower bound) = 200 + 4(1965 − 1932) = 332

مجدداً تأکید می‌کنیم که محتوای این عنصر را می‌توان بدون جستجوی هر عنصر دیگر آرایهٔ **AUTO** پیدا کرد.

توجه کنید: مجموعه‌ای از عناصر داده‌ای **A** را اندیس‌دار گویند هرگاه بتوان هر عنصر دلخواه **A** را که آن را A_k می‌نامیم در زمانی مستقل از **K** مکان‌یابی کرد یا مورد پردازش قرار داد. بحث بالا بیان می‌کند که

آرایه‌های خطی را می‌توان اندیس‌دار کرد. این یک خاصیت بسیار مهم آرایه‌های خطی است. درواقع لیستهای پیوندی که موضوع فصل بعد کتاب را تشکیل می‌دهد دارای چنین خاصیتی نیست.

۴-۴ پیمایش آرایه‌های خطی

فرض کنید A یک مجموعه از عناصر داده‌ای ذخیره شده در حافظهٔ کامپیوتر است. هرگاه بخواهیم محتوای هر عنصر A را چاپ کنیم یا تعداد عناصر A را که دارای یک خاصیت داده‌شده هستند بشماریم این عمل را می‌توان با پیمایش A یعنی با دسترسی و پردازش هر عنصر A دقیقاً یکبار (که اغلب ملاقات نامیده می‌شود) انجام داد.

الگوریتم زیر یک آرایهٔ خطی LA را پیمایش می‌کند. سادگی این الگوریتم از این واقعیت ناشی می‌شود که LA ساختار خطی دارد. ساختارهای خطی دیگری نظیر لیستهای پیوندی را می‌توان به‌سادگی پیمایش کرد. ازطرف دیگر، پیمایش ساختارهای غیرخطی نظیر درختها و گرافها به میزان قابل توجهی پیچیده و مشکل است.

Algorithm 4.1: (Traversing a Linear Array) Here LA is a linear array with lower bound LB and upper bound UB. This algorithm traverses LA applying an operation PROCESS to each element of LA.

1. [Initialize counter.] Set K := LB.
2. Repeat Steps 3 and 4 while K ≤ UB.
3. [Visit element.] Apply PROCESS to LA[K].
4. [Increase counter.] Set K := K + 1.
   [End of Step 2 loop.]
5. Exit.

علاوه بر این در شکل دیگری از این الگوریتم می‌توان از یک حلقهٔ تکرار **Repeat– For** بجای حلقهٔ **Repeat – While** استفاده نمود.

Algorithm 4.1': (Traversing a Linear Array) This algorithm traverses a linear array LA with lower bound LB and upper bound UB.

1. Repeat for K = LB to UB:
   Apply PROCESS to LA[K].
   [End of loop.]
2. Exit.