توجه کنید: ممکن است عمل **PROCESS** در الگوریتم پیمایشی از چند متغیر معین استفاده کند که باید قبل از اعمال **PROCESS** بر هر یک عناصر آرایه مقدار اولیه بگیرد. بنابراین ممکن است الگوریتم نیازمند یک مرحلهٔ مقدار اولیه گرفتن باشد.

مثال ۴ـ۴

آرایهٔ **AUTO** در مثال ۱ـ۴ **(ب)** را درنظر بگیرید که تعداد اتومبیلهای فروخته‌شده هر سال از **1932** تا **1984** را ثبت می‌کند. هر یک از قطعه برنامه‌های زیر که عمل داده شده را انجام می‌دهند شامل پیمایش **AUTO** است.

(الف) NUM تعداد سالهایی را که در طی آن اتومبیلهای فروخته‌شده بیش از **300** است پیدا کنید.

```
1. [Initialization step.] Set NUM := 0.
2. Repeat for K = 1932 to 1984:
       If AUTO[K] > 300, then: Set NUM := NUM + 1.
   [End of loop.]
3. Return.
```

(ب) هر سال و تعداد اتومبیلهای فروخته‌شدهٔ آن سال را چاپ کنید.

```
1. Repeat for K = 1932 to 1984:
       Write: K, AUTO[K].
   [End of loop.]
2. Return.
```

در قسمت (الف) ملاحظه کردید که قبل از پیمایش آرایهٔ **AUTO** نیازمند یک مرحله برای تخصیص مقدار اولیه به متغیر **NUM** هستیم.

۵ـ۴ اضافه کردن و حذف‌کردن یک عنصر

فرض کنید **A** یک مجموعه از عناصر داده‌ای در حافظهٔ کامپیوتر باشد. منظور از "اضافه کردن" عبارت است از عمل اضافه کردن یک عنصر جدید به مجموعهٔ **A** و منظور از "حذف‌کردن" عبارت است از عمل حذف یکی از عناصر **A** . این بخش، عمل اضافه‌کردن و حذف یک عنصر را در صورت خطی‌بودن **A** مورد بحث و بررسی قرار می‌دهد.

هرگاه فضای حافظهٔ اختصاص یافته برای یک آرایه خطی به اندازهٔ کافی بزرگ باشد تا بتواند یک عنصر اضافی را در خود جای دهد عمل اضافه‌کردن یک عنصر در "پایان" آرایهٔ خطی را می‌توان به‌سادگی انجام داد. ازطرف دیگر فرض کنید بخواهیم یک عنصر به وسط آرایه اضافه کنیم. در آن‌صورت، به‌طور متوسط باید نصف عناصر به پائین انتقال (شیفت!) داده شوند در مکانهای جدید، عناصر جدید

قرار گیرند و نظم عناصر دیگر آرایه نیز حفظ شود.

به‌طور مشابه، حذف عنصری که در "پایان" یک آرایه است مشکل به‌نظر نمی‌آید اما حذف یک عنصر در مکان‌هایی مانند وسط آرایه نیازمند آن است که هر عنصر بعدی به مکان جدیدی در بالای آرایه انتقال داده شوند تا فضای خالی‌شدهٔ بالای آرایه را پر کند.

توجه کنید: از آنجا که آرایه‌های خطی معمولاً به صورت قابل گسترش از پائین، نظیر شکل ۱ـ ۴ رسم می‌شوند منظور از اصطلاح "در پائین آرایه" مکانهایی با اندیس‌های بزرگتر و منظور از اصطلاح "در بالای آرایه" مکان‌هایی با اندیس‌های کوچکتر است.

مثال ۵ـ ۴

فرض کنید **TEST** به صورت یک آرایه **5** عنصری معرفی شده است اما داده‌ها تنها در خانه‌های **TEST[1]**، **TEST[2]** و **TEST[3]** ذخیره شده است. اگر **X** مقدار محتوای خانهٔ بعدی **TEST** باشد، آنگاه تنها با دستور جایگزینی

TEST[4] := X

X را به لیست اضافه می‌کنیم. به‌طور مشابه، اگر **Y** مقدار محتوای خانهٔ بعدی **TEST** باشد، آنگاه تنها با دستور جایگزینی

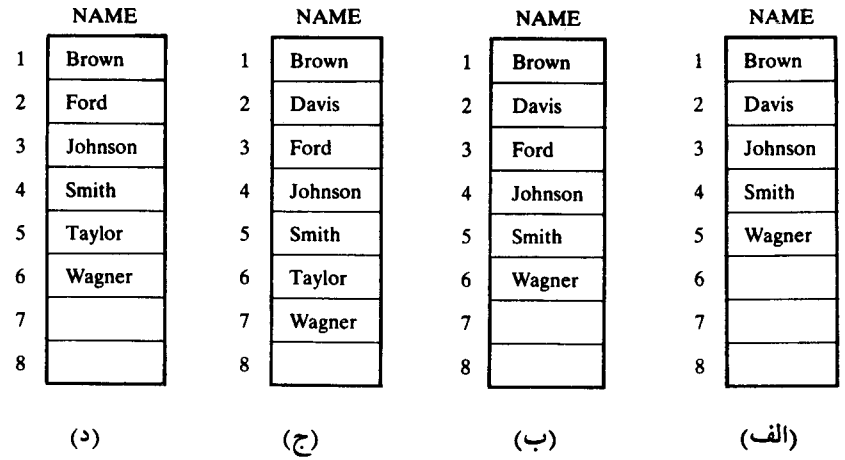
TEST[5] := Y

Y را به لیست اضافه می‌کنیم. با وجود این دیگر نمی‌توان نمرهٔ جدیدی از آزمون **TEST** را به لیست اضافه کرد.

مثال ۶ـ ۴

فرض کنید **NAME** یک آرایهٔ خطی **8** عنصری باشد و پنج نام در این آرایه، به صورت نشان داده شده در شکل ۴ـ ۴ **(الف)** ذخیره شده است. ملاحظه می‌شود که این نامها به ترتیب الفبایی در آرایه مرتب شده‌اند و فرض کنید بخواهیم ترتیب الفبایی نامهای آرایه همواره حفظ شود. فرض کنید **Ford** به آرایه اضافه شده است. آنگاه هر یک از نامهای **Johnson**، **Smith** و **Wagner** باید یک خانه به طرف پائین آرایه مانند شکل ۴ـ ۴ **(ب)** انتقال داده شوند. بعد از آن فرض کنید **Taylor** به این آرایه اضافه می‌شود. آنگاه **Wagner** باید مانند شکل ۴ـ ۴ **(ج)** یک خانه به پائین آرایه انتقال داده شود.

بالاخره فرض کنید **Davis** از آرایه حذف شده است. آنگاه هر یک از پنج نام **Ford**، **Johnson**، **Smith**، **Taylor** و **Wagner** مانند شکل ۴ـ ۴ **(ب)** باید یک خانه به بالای آرایه انتقال داده شوند. واضح است که اگر در آرایه چند هزار نام وجود داشته باشد این‌گونه انتقال داده‌ها پر هزینه خواهد بود.



شکل ۴ـ۴

الگوریتم زیر یک عنصر داده‌ای ITEM را در مکان K ام آرایهٔ خطی LA که دارای N عنصر است اضافه می‌کند. چهار مرحلهٔ اول، هر عنصر از مکان K ام به بعد را یک خانه به پائین منتقل می‌کند و در آرایهٔ LA فضای خالی ایجاد می‌کند. تأکید می‌کنیم که این عناصر به‌ترتیب عکس انتقال می‌یابند یعنی اول LA[N]، بعد LA[N − 1] و ... و بالاخره LA[K] در غیر اینصورت داده‌ها ممکن است پاک شوند. (مسأله ۳ـ۴ را ببینید.) مشروحاً این که، نخست قرار می‌دهیم J := N آنگاه با استفاده از J به عنوان یک شمارنده، با افزایش J هر بار، حلقه اجرا می‌شود تا اینکه J به K برسد. در مرحلهٔ بعدی یعنی مرحلهٔ 5، ITEM، به داخل آرایه در فضایی که اکنون ایجاد شده است اضافه می‌شود. قبل از پایان (خروج از) الگوریتم، N تعداد عناصر آرایهٔ LA یک واحد افزایش می‌یابد تا عنصر جدید را دربر بگیرد.

Algorithm 4.2: (Inserting into a Linear Array) INSERT(LA, N, K, ITEM)
Here LA is a linear array with N elements and K is a positive integer such that $K \le N$. This algorithm inserts an element ITEM into the Kth position in LA.

1. [Initialize counter.] Set J := N.
2. Repeat Steps 3 and 4 while $J \ge K$.
3. [Move Jth element downward.] Set LA[J + 1] := LA[J].
4. [Decrease counter.] Set J := J − 1.
   [End of Step 2 loop.]
5. [Insert element.] Set LA[K] := ITEM.
6. [Reset N.] Set N := N + 1.
7. Exit.

الگوریتم زیرعنصر K ام را از آرایهٔ خطی **LA** حذف می‌کند و آن را در متغیر **ITEM** جایگزین می‌کند :

Algorithm 4.3: (Deleting from a Linear Array) DELETE(LA, N, K, ITEM)
Here LA is a linear array with N elements and K is a positive integer such that $K \leq N$. This algorithm deletes the Kth element from LA.

1. Set ITEM := LA[K].
2. Repeat for J = K to N − 1:
   [Move J + 1st element upward.] Set LA[J] := LA[J + 1].
   [End of loop.]
3. [Reset the number N of elements in LA.] Set N := N − 1.
4. Exit.

توجه کنید : تأکید می‌کنیم که اگر در یک مجموعه از عناصر داده‌ای، همواره چند عمل حذف و اضافه‌کردن مورد نیاز باشد، آنگاه آرایهٔ خطی نمی‌تواند کاراترین روش ذخیرهٔ داده‌ها باشد.

۶ـ۴ مرتب‌کردن، مرتب‌کردن حبابی

فرض کنید **A** یک لیست **n** عنصری باشد، منظور از **مرتب‌کردن A** عبارت است از عمل کنار هم قراردادن عناصر **A** طوری که این عناصر به ترتیب صعودی باشند یعنی طوری که

$$A[1] < A[2] < A[3] < \ldots < A[N]$$

برای مثال فرض کنید **A** در آغاز به صورت لیست زیر باشد.

8, 4, 19, 2, 7, 13, 5, 16

پس از مرتب‌کردن، **A** به صورت لیست زیر درخواهد آمد :

2, 4, 5, 7, 8, 13, 16, 19

بنظر می‌رسد که مرتب‌کردن کار بدیهی و ساده‌ای باشد. درواقع، مرتب‌کردن با کارایی زیاد می‌تواند خیلی پیچیده باشد. واقعیت این است که الگوریتم‌های متعددی برای مرتب‌کردن وجود دارد. برخی از این الگوریتم‌ها در فصل ۹ بررسی می‌شوند. در اینجا یک الگوریتم بسیار ساده و معروف به نام **مرتب‌کردن حبابی** را ارائه داده و در مورد آن توضیح می‌دهیم.

توجه کنید : تعریف بالا از مرتب‌کردن عبارت است از قراردادن داده‌های عددی به ترتیب صعودی در کنارهم. این محدودیت فقط برای سهولت در نمادگذاری است. واضح است که به قراردادن داده‌های عددی به ترتیب نزولی در کنار هم یا قراردادن داده‌های غیرعددی به ترتیب الفبایی درکنار هم نیز مرتب کردن می‌گویند. در حقیقت **A** اغلب یک فایل از رکوردها است و منظور از مرتب‌کردن **A**، عبارت است از قراردادن رکوردهای **A** کنار هم به‌طوری که برحسب مقدارهای یک کلید داده مرتب باشند.

مرتب‌کردن حبابی

فرض کنید لیستی از اعداد A[1], A[2], ..., A[N] در حافظه است. الگوریتم مرتب‌کردن حبابی به‌صورت زیر کار می‌کند :

مرحلهٔ ۱ . A[1] و A[2] را با هم مقایسه کنید و آنها را با ترتیب خواسته شده کنار هم قرار دهید طوری که A[2] > A[1] آنگاه A[2] و A[3] را با هم مقایسه کنید و آنها را طوری کنار هم قرار دهید که A[3]>A[2] آنگاه A[3] و A[4] را با هم مقایسه کنید و آنها را طوری کنار هم قرار دهید که A[4]>A[3] این کار را تا آنجا ادامه دهید که A[N – 1] و A[N] با هم مقایسه شده باشند و آنها را طوری کنار هم قرار دهید که A[N]>A[N – 1] .

ملاحظه می‌کنید که در مرحلهٔ 1 ، n – 1 مقایسه انجام می‌شود. طی مرحلهٔ 1 ، بزرگترین عنصر در مکان n ام در وضعیت صعودی یا در وضعیت نزولی است. پس از کامل‌شدن مرحلهٔ 1 ، A[N] شامل بزرگترین عنصر خواهد بود.

مرحلهٔ ۲ . مرحلهٔ 1 را با یک مقایسهٔ کمتر تکرار کنید. یعنی اکنون پس از مقایسه و احتمالاً قرارگرفتن A[N – 2] و A[N –1] کنارهم کار را متوقف می‌کنیم. مرحلهٔ ۲ شامل N –2 مقایسه است و پس از کامل‌شدن مرحلهٔ ۲ ، دومین عنصر بزرگ در مکان A[N – 1] جای خواهد گرفت.

مرحلهٔ ۳. مرحلهٔ 1 را با دو مقایسهٔ کمتر تکرار کنید یعنی پس از مقایسه و احتمالاً قرارگرفتن A[N – 3] و A[N – 2] کنار هم کار را متوقف می‌کنیم.

. . . . . .
. . . . . .
. . . . . .

مرحلهٔ N – 1 . A[1]، A[2] را با هم مقایسه کنید و آنها را طوری کنار هم قرار دهید که A[2]>A[1] .

پس از n – 1 مرحله، لیست به صورت صعودی مرتب خواهد شد.

فرایند متوالی پیمایش تمام یا قسمتی از یک لیست غالباً یک "گذر" یا "مرحله" Pass نامیده می‌شود. از این‌رو هر یک از مراحل بالا یک گذر نامیده می‌شود. بنابراین الگوریتم مرتب‌کردن حبابی به n – 1 گذر احتیاج دارد که در آن n تعداد داده‌های ورودی است.

مثال ۷ـ ۴

فرض کنید اعداد زیر در آرایهٔ A ذخیره شده‌اند :

32, 51, 27, 85, 66, 23, 13, 57

مرتب‌کردن حبابی را بر روی آرایهٔ A بکار می‌بندیم و هر گذر را به‌طور جداگانه مورد بحث و بررسی

قرار می‌دهیم :

گذر 1. مقایسه‌های زیر را داریم.

(الف) A_1 و A_2 را مقایسه کنید. از آنجا که **51>32** ، لیست تغییر نمی‌کند.

(ب) A_2 و A_3 را مقایسه کنید. از آنجا که **51>27** ، **51** و **27** را به صورت زیر جابجا کنید :

32, (27,) (51,) 85, 66, 23, 13, 57

(ج) A_3 و A_4 را مقایسه کنید. از آنجا که **85>51** ، لیست تغییر نمی‌کند.

(د) A_4 و A_5 را مقایسه کنید. از آنجا که **85>66** ، **85** و **66** را به صورت زیر جابجا کنید.

32, 27, 51, (66,) (85,) 23, 13, 57

(ه) A_5 و A_6 را مقایسه کنید. از آنجا که **85>23** ، **85** و **23** را به‌صورت زیر جابجا کنید :

32, 27, 51, 66, (23,) (85,) 13, 57

(و) A_6 و A_7 را مقایسه کنید. از آنجا که **85>13** ، با جابجاییِ **85** و **13** نتیجه می‌شود :

32, 27, 51, 66, 23, (13,) (85,) 57

(ز) A_7 و A_8 را مقایسه کنید. از آنجا که **85 >57** ، با جابجاییِ **85** و **57** نتیجه می‌شود :

32, 27, 51, 66, 23, 13, (57,) (85)

در پایان گذر اول، بزرگترین عدد یعنی **85** به آخرین مکان آرایه انتقال می‌یابد. با وجود این، بقیهٔ اعداد مرتب‌شده نیستند، حتی اگر جای بعضی از این اعداد تغییر کرده باشد.

در مورد گذرهای باقیمانده ما تنها به نشان‌دادن جابجایی اکتفا می‌کنیم :

گذر 2.

(27,) (33,) 51, 66, 23, 13, 57, 85
27, 33, 51, (23,) (66,) 13, 57, 85
27, 33, 51, 23, (13,) (66,) 57, 85
27, 33, 51, 23, 13, (57,) (66,) 85

در پایان گذر 2، دومین عنصر بزرگ آرایه یعنی 66 به پائین و در همسایگی آخرین عنصر آرایه انتقال می‌یابد.

گذر 3.

27, 33, (23,) (51,) 13, 57, 66, 85
27, 33, 23, (13,) (51,) 57, 66, 85

گذر 4.

27, (23,) (33,) 13, 51, 57, 66, 85
27, 23, (13,) (33,) 51, 57, 66, 85

گذر 5.

(23,) (27,) 13, 33, 51, 57, 66, 85
23, (13,) (27,) 33, 51, 57, 66, 85

گذر 6.

(13,) (23,) 27, 33, 51, 57, 66, 85

گذر 6 درحقیقت دارای دو مقایسهٔ A_1 با A_2 و A_2 و A_3 است. مقایسهٔ دوم باعث جابجایی نمی‌شود.

گذر 7. بالاخره A_1 با A_2 مقایسه می‌شود. چون **13<23**. جابجایی صورت نمی‌گیرد. از آنجاکه لیست دارای **8** عنصر است، پس از گذر هفتم به صورت مرتب‌شده درخواهد آمد. ملاحظه می‌کنید که در این مثال، لیست واقعاً پس از گذر ششم مرتب شده خواهد بود. این وضعیت در پایان بخش بررسی می‌شود.

اکنون به صورت رسمی، الگوریتم مرتب‌کردن حبابی را بیان می‌کنیم.

Algorithm 4.4: (Bubble Sort) BUBBLE(DATA, N)
Here DATA is an array with N elements. This algorithm sorts the elements in DATA.

```
1. Repeat Steps 2 and 3 for K = 1 to N − 1.
2.      Set PTR := 1. [Initializes pass pointer PTR.]
3.      Repeat while PTR ≤ N − K: [Executes pass.]
            (a) If DATA[PTR] > DATA[PTR + 1], then:
                    Interchange DATA[PTR] and DATA[PTR + 1].
                [End of If structure.]
            (b) Set PTR := PTR + 1.
        [End of inner loop.]
    [End of Step 1 outer loop.]
4. Exit.
```

ملاحظه می‌کنید که در الگوریتم یک حلقهٔ داخلی وجود دارد که توسط متغیر **PTR** کنترل می‌شود و این حلقه در درون یک حلقهٔ خارجی قرار دارد که توسط اندیس **K** کنترل می‌شود. علاوه بر این مشاهده می‌کنید که **PTR** به عنوان اندیس آرایه بکار رفته است و یک شمارنده است اما از **K** به عنوان اندیس آرایه استفاده نشده است.

پیچیدگی الگوریتم مرتب‌کردن حبابی

معمولاً زمان اجرای یک الگوریتم مرتب‌کردن برحسب تعداد مقایسه‌های آن الگوریتم اندازه‌گیری می‌شود. $f(n)$ تعداد مقایسه‌های مرتب‌کردن حبابی به‌سادگی محاسبه می‌شود. به‌ویژه این که، در گذر اول $n - 1$ مقایسه انجام می‌شود که بزرگترین عنصر را در مکان آخر قرار می‌دهد. در گذر دوم $n - 2$ مقایسه انجام می‌شود که بزرگترین عنصر دوم را در همسایگی مکان آخر (یک خانه مانده به خانه آخر) قرار می‌دهد و الی آخر.

$$f(n) = (n-1) + (n-2) + \cdots + 2 + 1 = \frac{n(n-1)}{2} = \frac{n^2}{2} + O(n) = O(n^2)$$

به بیان دیگر، زمان موردنیاز برای اجرای الگوریتم مرتب‌کردن حبابی متناسب با n^2 است که در آن **n** تعداد داده‌های ورودی است.

توجه کنید : بعضی از برنامه‌نویسان برای این که اعلام کنند آیا در طی یک گذر جابجایی صورت گرفته است یا نه، از یک متغیر **1** بیتی **FLAG** یا متغیر منطقی **FLAG** در الگوریتم مرتب کردن حبابی استفاده می‌کنند. اگر پس از هر گذر **FLAG = 0** باشد آنگاه لیست از قبل مرتب شده است و هیچ نیازی به ادامهٔ کار وجود ندارد. این کار باعث می‌شود تعداد گذرها کم شود. با وجود این هنگام استفاده از چنین علامتی **FLAG** ، بایستی به آن مقدار اولیه داده شود مقدار آن تغییر کند و درطی هر گذر متغیر **FLAG** مورد آزمایش قرار گیرد که آیا مقدار آن تغییر کرده است یا خیر. از این‌رو استفاده از علامت **FLAG** تنها زمانی مفید و کارا است که لیست از ابتدا، تقریباً مرتب باشد.

۷ـ۴ جستجوکردن، جستجوی خطی

فرض کنید **DATA** مجموعه‌ای از عناصر داده‌ای در حافظه باشد و فرض کنید **ITEM** اطلاعات معلوم داده شده است. منظور از **جستجوکردن** عبارت است از عمل پیداکردن **LOC** مکان **ITEM** در **DATA** یا چاپ پیغامی نظیر **ITEM** در **DATA** وجود ندارد. جستجو را **موفق** گویند اگر **ITEM** در **DATA** وجود داشته باشد در غیر اینصورت آن را **ناموفق** گویند.

اغلب ممکن است بخواهیم پس از یک جستجوی ناموفقِ **ITEM** در **DATA** ، عنصر **ITEM** را به **DATA** اضافه کنیم. آنگاه به جای تنها یک الگوریتم **جستجو**، از الگوریتم **جستجو و اضافه‌کردن** استفاده می‌کنیم. الگوریتم‌های جستجو و اضافه‌کردن در بخش مسایل به‌طور مشروح توضیح داده می‌شود.

الگوریتمهای جستجوی بسیار متفاوتی وجود دارد. معمولاً الگوریتمی که انتخاب می‌شود بستگی به سازماندهی اطلاعات در **DATA** دارد. عمل جستجو به تفصیل در فصل ۹ شرح داده می‌شود. این بخش الگوریتم ساده‌ای را توضیح می‌دهد که **جستجوی خطی** نام دارد و در بخش بعد الگوریتم معروف **جستجوی دودویی** را مورد بحث و بررسی قرار می‌دهیم.

پیچیدگی الگوریتم‌های جستجو برحسب تعداد مقایسه‌های موردنیاز $f(n)$ برای پیداکردن **ITEM** در **DATA** اندازه‌گیری می‌شود که در آن **DATA** شامل **n** عنصر است. نشان خواهیم داد که جستجوی خطی یک الگوریتم دارای زمان خطی است اما جستجوی دودویی، الگوریتمی باکارآیی به‌مراتب بیشتر است و زمان آن متناسب‌با $\log_2 n$ است. از سوی دیگر معایب اعتماد بیش از حد به الگوریتم جستجوی دودویی را نیز بررسی می‌کنیم.

جستجوی خطی

فرض کنید **DATA** یک آرایهٔ خطی با **n** عنصر باشد. در مورد **DATA** هیچ اطلاع دیگری داده نشده است. شهودی‌ترین راه برای جستجوی یک **ITEM** داده شده در **DATA** عبارت است از مقایسهٔ **ITEM** با تک تک عناصر **DATA** . به عبارت دیگر نخست باید آزمایش کنیم که آیا **ITEM = DATA[1]** و آنگاه آزمایش می‌کنیم که آیا **ITEM = DATA[2]** و الی آخر. این روش که **DATA** را برای تعیین محل **ITEM** ، بطور متوالی پیمایش می‌کند **جستجوی خطی** یا **جستجوی متوالی** نامیده می‌شود.

برای بیان سادهٔ مطلب، نخست **ITEM** را در **DATA[N+1]** یعنی در مکان بعد از آخرین عنصر **DATA** جایگزین می‌کنیم. آنگاه نتیجهٔ

$$\mathbf{LOC = N + 1}$$

است که در آن **LOC** مکانی را نشان می‌دهد که **ITEM** برای اولین بار در **DATA** ظاهر شده است و مبین ناموفق‌بودن عمل جستجو است. هدف از این جایگزینی اولیه اجتناب از آزمایش تکراری است که آیا به انتهای آرایهٔ **DATA** رسیده‌ایم یا خیر. با این روش در نهایت باید عمل جستجو "**موفق**" باشد.

نمایش رسمی جستجوی خطی در الگوریتم ۵ـ۴ نشان داده شده است.

Algorithm 4.5: (Linear Search) LINEAR(DATA, N, ITEM, LOC)
Here DATA is a linear array with N elements, and ITEM is a given item of information. This algorithm finds the location LOC of ITEM in DATA, or sets LOC := 0 if the search is unsuccessful.

1. [Insert ITEM at the end of DATA.] Set DATA[N + 1] := ITEM.
2. [Initialize counter.] Set LOC := 1.
3. [Search for ITEM.]
   Repeat while DATA[LOC] ≠ ITEM:
      Set LOC := LOC + 1.
   [End of loop.]
4. [Successful?] If LOC = N + 1, then: Set LOC := 0.
5. Exit.

ملاحظه می‌کنید مرحله 1 تضمین می‌کند که باید حلقهٔ مرحلهٔ ۳ پایان یابد. بدون مرحلهٔ ۱ (الگوریتم ۴ـ ۲ را ببینید) دستور **Repeat** مرحلهٔ ۳ باید با دستور زیر که شامل دو مقایسه است نه یک مقایسه تعویض شود:

Repeat while LOC ≤ N و DATA[LOC] ≠ ITEM

از طرف دیگر، برای استفاده از مرحلهٔ ۱، باید تضمین کنیم که در پایان آرایهٔ **DATA** یک خانهٔ حافظه استفاده‌نشده وجود دارد، درغیر اینصورت باید از الگوریتم جستجوی خطی که در الگوریتم ۴ـ ۲ شرح داده شده، استفاده کنیم.

مثال ۸ـ ۴

آرایهٔ **NAME** شکل ۵ـ ۴ (الف) را که در آن **n = 6** است، درنظر بگیرید.

(الف) فرض کنید بخواهیم تحقیق کنیم که آیا **Paula** در آرایه وجود دارد یا خیر؟ در صورت مثبت‌بودن جواب، مکان آن را پیدا کنیم. الگوریتم به‌طور موقت **Paula** را در پایان آرایه قرارمی‌دهد. این وضعیت درشکل ۵ـ۴ (ب) باقراردادن **NAME[7] = paula** نشان داده شده است. آنگاه این الگوریتم عمل جستجو در آرایه را از بالا تا پائین انجام می‌دهد. از آنجا که **Paula** برای نخستین بار در **NAME[N+1]** طاهر شده است، **Paula** در آرایهٔ اصلی وجود ندارد.

(ب) فرض کنید بخواهیم تحقیق کنیم که آیا **Susan** در آرایه وجود دارد یا خیر؟ در صورت مثبت‌بودن جواب، مکان آن را پیدا کنیم. الگوریتم به‌طور موقت **Susan** را در پایان آرایه قرارمی‌دهد. این‌وضعیت درشکل ۵ـ۴ (ب) باقراردادن **NAME [7] = Susan** نشان داده شده است. آنگاه این الگوریتم عمل

جستجو در آرایه را از بالا تا پائین انجام می‌دهد. از آنجا که **Susan** برای نخستین بار در **NAME [4]** (که **n≥4** است) ظاهر شده است، می‌فهمیم که **Susan** در آرایهٔ اصلی وجود دارد.

	NAME
1	Mary
2	Jane
3	Diane
4	Susan
5	Karen
6	Edith
7	Susan
8	

(ج)

	NAME
1	Mary
2	Jane
3	Diane
4	Susan
5	Karen
6	Edith
7	Paula
8	

(ب)

	NAME
1	Mary
2	Jane
3	Diane
4	Susan
5	Karen
6	Edith
7	
8	

(الف)

شکل ۵ـ ۴

پیچیدگی الگوریتم جستجوی خطی

همانگونه که در بالا متذکر شدیم، پیچیدگی الگوریتم جستجو به وسیلهٔ تعداد مقایسه‌های موردنیاز $f(n)$ برای پیداکردن **ITEM** در **DATA** اندازه‌گیری می‌شود که در آن **DATA** شامل **n** عنصر است. دو حالت مهم و قابل توجه که مورد بحث و بررسی قرار می‌گیرد حالت میانگین و بدترین حالت است.

واضح است که بدترین حالت وقتی اتفاق می‌افتد که عمل جستجو در تمام آرایهٔ **DATA** انجام شود و **ITEM** در **DATA** ظاهر نشده باشد.

در این حالت، الگوریتم نیازمند

$$f(n) = n + 1$$

مقایسه است. بنابراین در بدترین حالت، زمان اجرا متناسب با n است.

زمان اجرای حالت میانگین از مفهوم امید ریاضی در احتمالات استفاده می‌کند. (بخش ۵ـ ۲ را ببینید). فرض کنید P_k احتمال آن باشد که **ITEM** در **DATA[K]** ظاهر شده باشد و **q** احتمال آن باشد که

ITEM در DATA ظاهر نشده باشد. در آن صورت ($P_1 + P_2 + ... + P_n + q = 1$). هرگاه ITEM در DATA[K] ظاهر شده باشد چون الگوریتم از K مقایسه استفاده می‌کند، میانگین تعداد مقایسه‌ها به صورت زیر محاسبه می‌شود:

$$f(n) = 1.\ p_1 + 2.\ p_2 + ... + n.\ p_n + (n + 1).q$$

به‌خصوص این که فرض کنید q خیلی کوچک و ITEM با احتمالی مساوی در هر عنصر DATA ظاهر شده باشد. آنگاه $q \approx 0$ و هر $p_i = 1/n$ بنابراین

$$f(n) = 1 \cdot \frac{1}{n} + 2 \cdot \frac{1}{n} + \cdots + n \cdot \frac{1}{n} + (n+1) \cdot 0 = (1 + 2 + \cdots + n) \cdot \frac{1}{n}$$
$$= \frac{n(n+1)}{2} \cdot \frac{1}{n} = \frac{n+1}{2}$$

یعنی در این حالت خاص، میانگین تعداد مقایسه‌های موردنیاز برای یافتن مکان ITEM تقریباً برابر نصف تعداد عناصر در آرایه است.

۸ـ۴ جستجوی دودویی

فرض کنید DATA یک آرایه است که در آن داده‌های عددی به ترتیب صعودی، یا معادل آن، داده‌های حرفی نیز به ترتیب صعودی ذخیره شده‌اند. آنگاه الگوریتم جستجوی بسیار کارآیی بنام **جستجوی دودویی** وجود دارد که می‌توان از آن برای پیداکردن LOC مکان یک ITEM داده شده از اطلاعات در DATA استفاده کرد. این الگوریتم قبلاً به صورت رسمی بیان شده است. ایدهٔ کلی این الگوریتم را به کمک یک نمونهٔ واقعی از مثال آشنایی شرح می‌دهیم که در زندگی روزمره با آن سروکار دارید.

فرض کنید بخواهید مکان یک اسم را در راهنمای تلفن پیدا کنید یا بخواهید در یک فرهنگ لغت، یک لغت را پیدا کنید. واضح است که یک جستجوی خطی را انجام نمی‌دهید. به‌عوض آن، کتابچهٔ راهنما را از وسط باز می‌کنید و دنبال آن نیمه از راهنما می‌گردید که حدس زدید اسم مورد نظر شما در آن نیمه قرار دارد. آنگاه نیمهٔ اخیر را از وسط نصف کرده و در یک چهارم از راهنما می‌گردید که حدس زدید اسم موردنظر شما در آن یک چهارم قرار دارد. بدنبال آن یک چهارم اخیر را از وسط نصف کرده و در یک هشتم از راهنما می‌گردید که حدس زدید اسم موردنظر شما در آن یک هشتم قرار دارد. کار را همینطور تا آخر ادامه می‌دهید. با توجه به این که دائماً (خیلی سریع) تعداد مکانهای ممکن، در راهنما کاهش می‌یابد، در نهایت مکان اسم و اسم موردنظر را پیدا می‌کنید.

هرگاه الگوریتم جستجوی دودویی را بر آرایهٔ DATA اعمال کنید به صورت زیر عمل می‌کند. در هر

مرحله از الگوریتم، جستجوی ITEM در یک قطعه از عناصر DATA کاهش می‌یابد.

DATA[BEG], DATA[BEG + 1], DATA[BEG + 2], ..., DATA[END]

توجه دارید که متغیرهای BEG و END به ترتیب مکانهای اول و آخر قطعهٔ مورد بررسی را نشان می‌دهند. این الگوریتم ITEM را با عنصر وسط آن قطعه یعنی DATA[MID] مقایسه می‌کند که در آن MID از رابطهٔ زیر به دست می‌آید :

MID = INT((BEG + END)/2)

ما برای مقدار صحیح A از تابع کامپیوتری INT(A) استفاده می‌کنیم. اگر DATA[MID] = ITEM ، آنگاه جستجو موفق است و قرار می‌دهیم LOC := MID در غیراینصورت قطعهٔ جدیدی از DATA را به صورت زیر به دست می‌آوریم :

(الف) اگر ITEM<DATA[MID] آنگاه ITEM می‌تواند تنها در نیمهٔ چپ قطعه ظاهر شود :

DATA[BEG], DATA[BEG + 1], ..., DATA[MID − 1]

بنابراین قرار می‌دهیم END := MID − 1 و عمل جستجو را مجدداً انجام می‌دهیم :

(ب) اگر ITEM>DATA[MID] آنگاه ITEM می‌تواند تنها در نیمهٔ راست قطعه ظاهر شود :

DATA[MID + 1], DATA[MID + 2], ..., DATA[END]

بنابراین قرار می‌دهیم BEG := MID + 1 و عمل جستجو را مجدداً ادامه می‌دهیم :

در آغاز، کار را با تمام آرایهٔ DATA یعنی با BEG = 1 و END = n یا با بیان کلی‌تر با BEG = LB و END = UB شروع می‌کنیم.

اگر ITEM در DATA ظاهر نشده باشد آنگاه نهایتاً به

END<BEG

می‌رسیم که اعلام می‌کند جستجو ناموفق است و در چنین حالتی دستور جایگزینی LOC := NULL را داریم. در اینجا NULL مقداری است که در محدودهٔ مجموعهٔ اندیسهای DATA قرار ندارد. در بیشتر موارد می‌توانیم NULL را برابر صفر اختیار کنیم. (NULL = 0)

در اینجا الگوریتم جستجو دودویی را به صورت رسمی زیر بیان می‌کنیم :

توجه کنید : هرگاه ITEM در DATA وجود نداشته باشد نهایتاً الگوریتم به مرحله‌ای می‌رسد که BEG = END = MID آنگاه مرحلهٔ بعدی END<BEG را نتیجه می‌دهد و کنترل کار به مرحلهٔ پنجم الگوریتم داده می‌شود. این وضعیت در قسمت (ب) مثال بعد اتفاق می‌افتد.

Algorithm 4.6: (Binary Search) BINARY(DATA, LB, UB, ITEM, LOC)
Here DATA is a sorted array with lower bound LB and upper bound UB, and ITEM is a given item of information. The variables BEG, END and MID denote, respectively, the beginning, end and middle locations of a segment of elements of DATA. This algorithm finds the location LOC of ITEM in DATA or sets LOC = NULL.

```
1. [Initialize segment variables.]
   Set BEG := LB, END := UB and MID = INT((BEG + END)/2).
2. Repeat Steps 3 and 4 while BEG ≤ END and DATA[MID] ≠ ITEM.
3.      If ITEM < DATA[MID], then:
            Set END := MID - 1.
        Else:
            Set BEG := MID + 1.
        [End of If structure.]
4.      Set MID := INT((BEG + END)/2).
   [End of Step 2 loop.]
5. If DATA[MID] = ITEM, then:
        Set LOC := MID.
   Else:
        Set LOC := NULL.
   [End of If structure.]
6. Exit.
```

مثال ۹ـ ۴

فرض کنید DATA آرایه‌ای متشکل از 13 عنصر به صورت زیر ذخیره شده است :

DATA: 11, 22, 30, 33, 40, 44, 55, 60, 66, 77, 80, 88, 99

برای یافتن مقادیر مختلف ITEM ، جستجوی دودویی را در DATA بکار می‌بندیم.

(الف) فرض کنید **ITEM = 40** . جستجو برای ITEM در آرایهٔ DATA در شکل ۶ـ ۴ به تصویر درآمده است که در آن مقادیر **DATA[BEG]** و **DATA[END]** در هر مرحله از الگوریتم با دایره و مقدار **DATA[MID]** با یک مربع نشان داده شده است. به ویژه این که، **END, BEG** و **MID** مقادیر متوالی زیر را دارا هستند :

(۱) در آغاز **BEG = 1** و **END = 13** از این‌رو

MID = INT[(1 + 13) / 2] = 7 و در نتیجه **DATA[MID] = 55**

(۲) چون **40<55** ، **END** با مقدار **END = MID − 1 = 6** تغییر می‌کند. از این‌رو

MID = INT[(1 + 6) / 2] = 3 و در نتیجه **DATA[MID] = 30**

(۳) چون **40>30** ، **BEG** با مقدار **BEG = MID + 1 = 4** تغییر می‌کند. از این‌رو

MID = INT[(4 + 6) / 2] = 5 و در نتیجه DATA[MID] = 40

ITEM را در مکان LOC = MID = 5 پیدا کرده‌ایم.

(1) (11,) 22, 30, 33, 40, 44, [55,] 60, 66, 77, 80, 88, (99)
(2) (11,) 22, [30,] 33, 40, (44,) 55, 60, 66, 77, 80, 88, 99
(3) 11, 22, 30, (33,) [40,] (44,) 55, 60, 66, 77, 80, 88, 99 [موفق]

شکل ۶ـ ۴ . جستجوی دودویی برای ITEM = 40

(ب) فرض کنید ITEM = 85 . جستجو برای ITEM در آرایهٔ DATA در شکل ۷ـ ۴ به تصویر درآمده است. در اینجا BEG و END مقادیر متوالی زیر را دارا هستند :

(۱) مجدداً در آغاز MID = 7, END = 13, BEG = 1 و DATA[MID] = 55 .

(۲) چون 85>55 ، BEG با مقدار BEG = MID + 1 = 8 تغییر می‌کند. از این‌رو

MID = INT[(8 + 13) / 2] = 10 و در نتیجه DATA[MID] = 77

(۳) چون 85>77 ، BEG با مقدار END = MID + 1 = 11 تغییر می‌کند. از این‌رو

MID = INT[(11 + 13) / 2] = 12 و در نتیجه DATA[MID] = 88

(۴) چون 85<88 ، END با مقدار END = MID − 1 = 11 تغییر می‌کند. از این‌رو

MID = INT[(11 + 11) / 2] = 11 و در نتیجه DATA[MID] = 80

ملاحظه می‌کنید که اکنون BEG = END = MID = 11 .

چون 85>80 ، BEG با مقدار BEG = MID + 1 = 12 تغییر می‌کند. اما اکنون BEG>END . از این‌رو ITEM در DATA وجود ندارد.

(1) (11,) 22, 30, 33, 40, 44, [55,] 60, 66, 77, 80, 88, (99)
(2) 11, 22, 30, 33, 40, 44, 55, (60,) 66, [77,] 80, 88, (99)
(3) 11, 22, 30, 33, 40, 44, 55, 60, 66, 77, (80,) [88,] (99)
(4) 11, 22, 30, 33, 40, 44, 55, 60, 66, 77, (80,) 88, 99 [ناموفق]

شکل ۷ـ ۴. جستجوی دودویی برای ITEM = 85

پیچیدگی الگوریتم جستجوی دودویی

پیچیدگی الگوریتم جستجوی دودویی به وسیلهٔ تعداد مقایسه‌های موردنیاز $f(n)$ برای تعیین مکان ITEM در DATA اندازه‌گیری می‌شد که در آن DATA شامل n عنصر است. ملاحظه می‌شود که هر مقایسه اندازهٔ نمونه را نصف می‌کند. از این‌رو حداکثر $f(n)$ مقایسه لازم است تا مکان ITEM پیدا شود که در آن

$$f(n) = [\log_2 n] + 1 \qquad \text{یا معادل آن} \qquad 2^{f(n)} > n$$

یعنی زمان اجرای بدترین حالت تقریباً برابر $\log_2 n$ است. همچنین می‌توان نشان داد که زمان اجرای حالت میانگین تقریباً برابر زمان اجرای بدترین حالت است.

مثال ۱۰ـ۴

فرض کنید DATA شامل 1 000 000 عنصر باشد. ملاحظه می‌کنید که :

$$2^{10} = 1024 > 1000 \qquad \text{و از این‌رو} \qquad 2^{20} > 1000^2 = 1\ 000\ 000$$

بنابراین با استفاده از الگوریتم جستجوی دودویی تنها به حدود 20 مقایسه احتیاج است تا مکان یک عنصر ITEM در آرایهٔ DATA با 1 000 000 عنصر پیدا شود.

محدودیتهای الگوریتم جستجوی دودویی

از آنجا که الگوریتم جستجوی دودویی از کارآیی بالایی برخوردار است مثلاً برای یک لیست اولیهٔ 1 000 000 عنصری تنها حدوداً به 20 مقایسه احتیاج است، چرا با وجود این، می‌خواهیم از الگوریتم جستجوی دیگری استفاده کنیم؟ ملاحظه کردید که این الگوریتم احتیاج به دو شرط دارد: (۱) لیست بایستی مرتب شده، باشد و (۲) بایستی به عنصر وسط هر زیرلیست، دسترسی مستقیم داشته باشیم. به این معنی که اساساً باید از یک آرایهٔ مرتب شده برای نگهداری داده‌ها استفاده کرد. اما وقتی عملیات اضافه و حذف کردن زیادی در آرایه مورد نیاز باشد، نگهداری داده‌ها در یک آرایهٔ مرتب شده معمولاً پرهزینه است. بنابراین در چنین وضعیتهایی می‌توانیم از ساختمان دادهٔ متفاوتی نظیر لیستهای پیوندی یا درخت جستجوی دودویی استفاده کرده تا داده‌ها را در آنها ذخیره کنیم.

۹ـ۴ آرایه‌های چندبعدی

آرایه‌های خطی‌ای که تا اینجا مورد بررسی قرار گرفته‌اند **آرایه‌های یک‌بعدی** نیز نامیده می‌شوند. زیرا به هر عنصر این نوع آرایه‌ها می‌توان به کمک تنها یک اندیس دسترسی پیدا کرد. اکثر زبانهای

برنامه‌نویسی اجازهٔ استفاده از آرایه‌های دوبعدی و سه‌بعدی را به برنامه‌نویس می‌دهند یعنی آرایه‌هایی که به هر عنصر آنها می‌توان به ترتیب به کمک دو یا سه اندیس دسترسی پیدا کرد. شایان ذکر است بعضی از زبانهای برنامه‌نویسی اجازه می‌دهند تعداد بعدهای یک آرایه حتی تا ۷ بُعد باشد. در این بخش آرایه‌های چندبعدی مورد بررسی قرار می‌گیرد.

آرایه‌های دوبعدی

یک آرایهٔ دوبعدی $m \times n$ مجموعه‌ای با $m \times n$ عنصر داده‌ای است به‌گونه‌ای که هر عنصر آن با یک جفت عدد صحیح (مانند K, J) بنام اندیس، مشخص شده باشد و دارای این خاصیت باشد که

$$1 \leq J \leq m \quad \text{و} \quad 1 \leq J \leq n$$

عنصر A با اندیس اول J و اندیس دوم K به صورت زیر نمایش داده می‌شود :

$$A_{J,K} \quad \text{یا} \quad A[J, K]$$

آرایه‌های دوبعدی را در ریاضیات، **ماتریس**، و در کاربردهای تجاری و بازرگانی **جدول** می‌نامند. بنابراین به آرایه‌های دوبعدی گاهی اوقات **آرایه‌های ماتریسی** نیز می‌گویند.

یک روش استاندارد برای نمایش آرایهٔ دو بعدی $m \times n$ ماتریس A وجود دارد که در آن عناصر A تشکیل یک آرایهٔ مستطیلی با m سطر و n ستون رامی‌دهد که در آن عنصر A[J, K] در سطر J ام و ستون K ام قرار دارد. به یک لیست افقی از عناصر، **سطر** و به لیست عمودی از عناصر، **ستون** می‌گویند. شکل ۸ـ ۴ حالتی را نشان می‌دهد که در آن ماتریس A دارای ۳ سطر و ۴ ستون است.

		Columns			
		1	2	3	4
Rows	1	A[1, 1]	A[1, 2]	A[1, 3]	A[1, 4]
	2	A[2, 1]	A[2, 2]	A[2, 3]	A[2, 4]
	3	A[3, 1]	A[3, 2]	A[3, 3]	A[3, 4]

شکل ۸ـ ۴ . آرایهٔ ۴ × ۳ دوبعدی A

تأکید می‌کنیم که هر سطر شامل آن دسته از عناصری است که اندیس اول آنها با هم برابر است و هر ستون شامل آن عناصری است که اندیس دوم آنها با هم برابر است.

مثال ۱۱ـ ۴

فرض کنید تمام دانشجویان یک کلاس 25 نفره در 4 آزمون امتحانی شرکت کرده‌اند. فرض می‌شود

دانشجویان از 1 تا 25 شماره‌گذاری شده‌اند. نمرات آزمون را می‌توان در یک آرایهٔ ماتریسی 4 × 25 بنام SCORE به صورتی که در شکل ۹ـ ۴ نشان داده شده است جایگزین کرد.

Student	Test 1	Test 2	Test 3	Test 4
1	84	73	88	81
2	95	100	88	96
3	72	66	77	72
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
25	78	82	70	85

شکل ۹ـ ۴ . آرایهٔ SCORE

بنابراین **SCORE[K, L]** حاوی نمرهٔ دانشجوی **K** ام در آزمون **L** ام است.

SCORE[2, 1], SCORE[2, 2], SCORE[2, 3], SCORE[2, 4]

حاوی 4 نمرهٔ آزمون دانشجوی دوم است.

فرض کنید **A** یک آرایهٔ **m × n** دوبعدی باشد. اولین بعد **A** شامل مجموعهٔ اندیسهای 1 ، ... ، **m** با **کران پائین 1 و کران بالای m** است و دومین بعد **A** شامل مجموعهٔ اندیسهای 1 ، ... ، **n** با **کران پائین 1 و کران بالای n** است. **طول** یک بعد، تعداد اعداد صحیح موجود در مجموعهٔ اندیسهای آن است. جفت طول‌های **m × n** (بخوانید **m** در **n**) **اندازهٔ آرایه** نامیده می‌شود.

برخی از زبان‌های برنامه‌نویسی به برنامه‌نویس اجازه می‌دهند که آرایه‌های چندبعدی تعریف کنند که در آن کران پائین 1 نیستند نظیر آرایه‌هایی که غالباً **بی‌نظم** یا **نامنظم** نامیده می‌شوند. با وجود این، مجموعهٔ اندیسهای هر بعد همچنان متشکل از اعداد صحیح متوالی از کران پائین تا کران بالای بعد است. طول یک بعد معین یعنی تعداد اعداد صحیح موجود در مجموعهٔ اندیسهای آن، از فرمول زیر بدست می‌آید :

$$\text{Length} = \text{upper bound} - \text{lower bound} + 1 \qquad (۴ـ۳)$$

توجه دارید که این فرمول همان فرمول (۱ـ ۴) است که برای آرایه‌های خطی مورد استفاده قرار گرفته است. درحالت کلی همواره فرض می‌کنیم که آرایه‌ها منظم هستند یعنی کران پائین هر بعد آرایه برابر 1 است مگر آن که خلاف آن بیان شود.

هر زبان برنامه‌نویسی دارای قاعده و روش خاصی برای معرفی آرایه‌های چندبعدی است. همانگونه که در مورد آرایه‌های خطی ملاحظه کردید تمام عناصر چنین آرایه‌هایی باید از یک نوع داده باشند. برای

مثال فرض کنید DATA یک آرایهٔ 8 × 4 دوبعدی با عناصری از نوع اعشاری باشد. FORTRAN، PL / I و PASCAL چنین آرایه‌ای را به صورت زیر معرفی می‌کنند :

```
FORTRAN:    REAL DATA(4, 8)
PL/1:       DECLARE DATA(4, 8) FLOAT;
Pascal:     VAR DATA: ARRAY[1 . . 4, 1 . . 8] OF REAL;
```

ملاحظه می‌کنید که در پاسکال کرانهای پائین حتی اگر این کران 1 باشد هنگام معرفی آرایه ذکر می‌شود.

توجه کنید : زبانهای برنامه‌نویسی‌ای که توانایی معرفی آرایه‌های نامنظم را دارند معمولاً برای جداکردن کران پائین از کران بالای هر بُعد از یک کولن و برای جداکردن ابعاد، ازهم از یک کاما استفاده می‌کنند. برای مثال در FORTRAN :

```
INTEGER NUMB(2:5, − 3:1)
```

NUMB را به‌صورت یک آرایهٔ دوبعدی از نوعِ صحیح معرفی می‌کند. در اینجا مجموعه اندیسهای بُعدها به ترتیب از اعداد صحیح

$$2, 3, 4, 5 \quad \text{و} \quad -3, -2, -1, 0, 1$$

تشکیل شده‌اند. بنابه فرمول (۳ـ ۴) ، طول بعد اول برابر 4=1+2 − 5 و طول بعد دوم برابر 5 = 1 + (3 −)−1 است. بنابراین NUMB دارای 40=4.5 عنصر است.

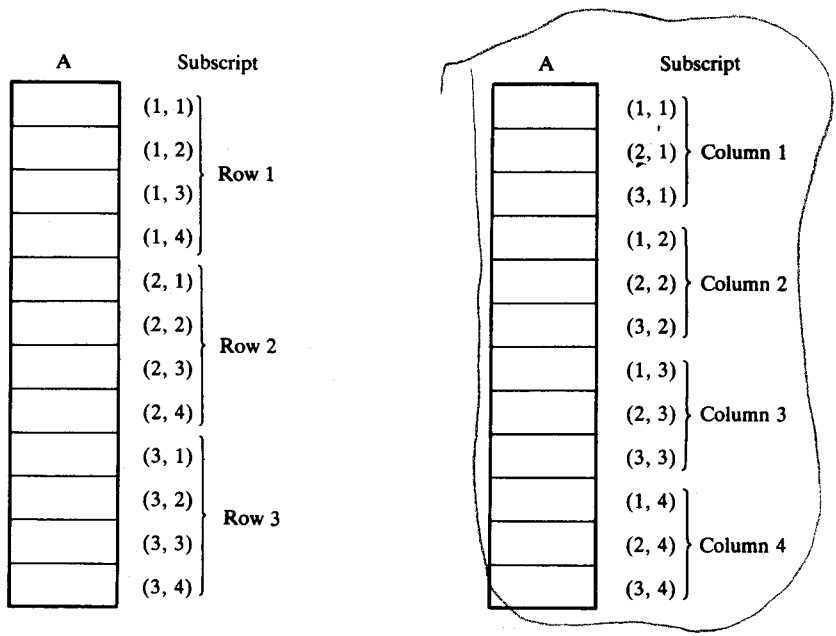
نمایش آرایه‌های دوبعدی در حافظه

فرض کنید A یک آرایهٔ m × n دوبعدی باشد. هرچند A به صورت یک آرایهٔ مستطیلی از عناصر با m سطر و n ستون به تصویر کشیده شده است، اما این آرایه در حافظهٔ کامپیوتر توسط یک بلاک با m×n خانهٔ متوالی حافظه نمایش داده می‌شود. به‌خصوص این‌که، زبان برنامه‌نویسی آرایه را به دو صورت ۱ـ **ستون به ستون** که **روش ستونی** نیز نامیده می‌شود ۲ـ **سطر به سطر** که **روش سطری** نیز نامیده می‌شود ذخیره می‌کنند. شکل ۱۰ـ ۴ وقتی که A یک آرایهٔ دوبعدی 4 × 3 است را با این دو روش نشان می‌دهد. تأکید می‌کنیم که انتخاب هر یک از این دو نمایش بستگی به زبان برنامه‌نویسی* دارد نه به کاربر یا برنامه‌نویس.

یادآور می‌شویم که برای یک آرایهٔ خطی A ، کامپیوتر آدرس (LOC(LA[K]) هر عنصر LA[K] از LA را نگه نمی‌دارد بلکه تنها آدرس Base(LA) یعنی آدرس عنصر اول LA را نگه می‌دارد. کامپیوتر از فرمول

* ـ زبان FORTRAN آرایه‌ها را به صورت ستونی و زبان‌های Algol - like مانند زبان C و PASCAL ، آرایه‌ها را به صورت سطری، در آرایهٔ یک بعدی ذخیره می‌کند. مترجم

$$LOC(LA[K]) = Base(LA) + w(K - 1)$$



(الف) روش ستونی (ب) روش سطری

شکل ۱۰ـ۴

برای پیداکردن آدرسی LA[K] به‌موقع و بدون درنظرگرفتن K استفاده می‌کند. در اینجا W تعداد کلمات در حافظه یا طول کلمه برای آرایه LA است و یک،کران پائین مجموعهٔ اندیس LA است.

از وضعیتی مشابهٔ وضعیت بالا برای نگهداری هر آرایهٔ دوبعدی A که $m \times n$ است استفاده می‌شود. به عبارت دیگر کامپیوتر آدرس پایه Base(A) آدرس اولین عنصر A یعنی A[1, 1] را نگه می‌دارد و آدرس LOC(A[J, K]) ی A[J, K] را با استفاده از فرمول

$$LOC(A[J, K]) = Base(A) + w[M(K - 1) + (J - 1)] \quad \text{(روش ستونی)} \qquad (۴ـ۴)$$

و فرمول

$$LOC(A[J, K]) = Base(A) + w[N(J - 1) + (K - 1)] \quad \text{(روش سطری)} \qquad (۴ـ۵)$$

محاسبه می‌کند. در این جا نیز W تعداد کلمات خانهٔ حافظه یا طول کلمه را برای آرایهٔ A نمایش می‌دهد.

توجه دارید که فرمولها برحسب J و K خطی هستند و می‌توان آدرس LOC(A[J, K]) را به‌موقع بدون درنظر گرفتن K به دست آورد.

مثال ۱۲ـ ۴

آرایهٔ ماتریسی SCORE در مثال ۱۱ـ ۴ را که 4 × 25 است درنظر بگیرید. فرض‌کنید Base(SCORE) = 200 و تعداد W=4 کلمه در خانهٔ حافظه وجود داشته باشد. علاوه بر این فرض می‌شود زبان برنامه‌نویسی آرایه‌های دوبعدی را با استفاده از روش سطری ذخیره می‌کند. آنگاه آدرس SCORE[12, 3] یعنی نمرهٔ آزمون سوم دانشجوی دوازدهم، به صورت زیر محاسبه می‌شود :

$$LOC(SCORE[12, 3]) = 200 + 4[4(12 - 1) + (3 - 1)] = 200 + 4[46] = 384$$

ملاحظه می‌شود که در محاسبهٔ آدرس تنها از رابطهٔ (۵ـ ۴) استفاده شده است.

بدیهی است که آرایه‌های چندبعدی تفاوت بین نمایشهای منطقی و فیزیکی داده‌ها را بهتر بیان می‌کنند. شکل ۸ـ ۴ چگونگی نمایش منطقی A آرایهٔ ماتریسی 4 × 3 را به‌صورت یک آرایهٔ مستطیلی از داده‌ها که در آن A[J, K] در سطر J و ستون K ظاهر می‌شود نشان می‌دهد. ازطرف دیگر، داده‌ها به صورت فیزیکی به وسیلهٔ یک مجموعهٔ خطی از خانه‌های حافظه یا به صورت یک بعدی در حافظهٔ کامپیوتر ذخیره می‌شوند. این وضعیت در سراسر کتاب مورد توجه ماست. مثلاً برخی از ساختمان داده‌ها نظیر درختها یا گرافها را می‌توان به صورت منطقی درنظر گرفت درحالی که از نظر فیزیکی، به صورت خطی در حافظهٔ کامپیوتر ذخیره می‌شوند.

حالت کلی آرایه‌های چندبعدی

آرایه‌های چندبعدی در حالت کلی مانند آرایهٔ دو بعدی تعریف می‌شود. به‌ویژه این که، یک آرایهٔ n بعدی $m_1 \times m_2 \times \ldots \times m_n$ بنام B مجموعه‌ای از $m_1.m_2 \ldots m_n$ عنصر داده‌ای است که در آن هر عنصر به وسیلهٔ لیستی از n عدد صحیح نظیر $k_1, k_2, \ldots, k_n$ مشخص می‌شود که اندیس نام دارند و دارای این خاصیت است که

$$1 \le K_1 \le m_1, \quad 1 \le K_2 \le m_2, \quad \ldots, \quad 1 \le K_n \le m_n$$

عنصر B با اندیسهای $k_1, k_2, \ldots, k_n$ به صورت زیر نمایش داده می‌شود :

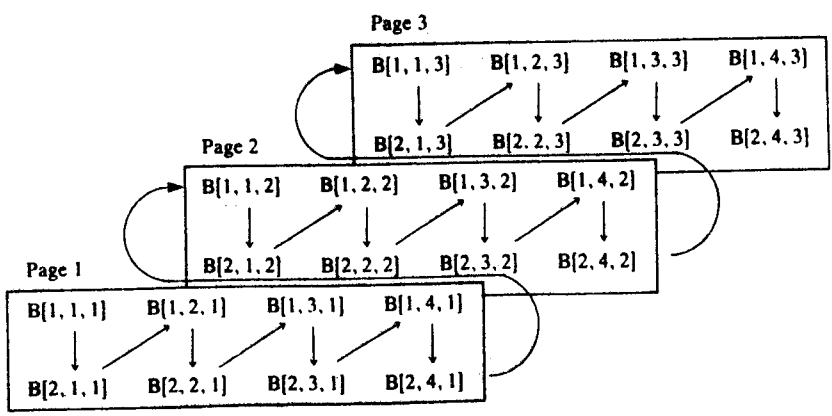
$$B[K_1, K_2, \ldots, K_N] \quad \text{یا} \quad B_{K_1,K_2,\ldots K_n}$$

آرایه در حافظه، در دنباله‌ای از خانه‌های حافظهٔ پشت سرهم ذخیره می‌شوند. به‌خصوص اینکه،

زبانهای برنامه‌نویسی آرایه **B** را به یکی از دو صورت **سطری** یا **ستونی** ذخیره می‌کنند. منظور ما از روش سطری آن است که عناصر به صورتی لیست می‌شوند که اندیسها مانند کیلومترشمار اتومبیل تغییر می‌کنند یعنی به‌گونه‌ای که آخرین اندیس اول تغییر می‌کند (با سرعت خیلی زیاد)، اندیس همسایگی اندیس آخر در مرحلهٔ دوم تغییر می‌کند (با سرعت کمتر) و الی آخر. منظور ما از روش ستونی آن است که عناصر به صورتی لیست می‌شوند که اولین اندیس، اول تغییر می‌کند (با سرعت خیلی زیاد) ، دومین اندیس در مرحله دوم تغییر می‌کند (با سرعت کمتر) و الی آخر.

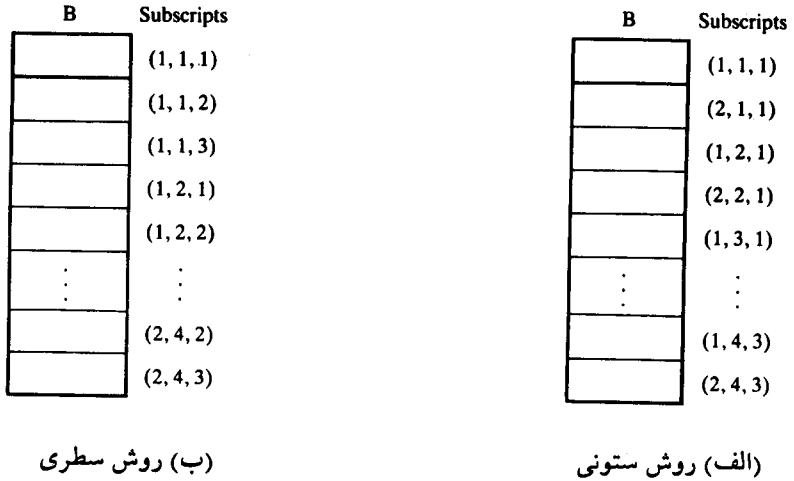
مثال ۱۳ـ ۴

فرض کنید **B** یک آرایه سه‌بعدی $3 \times 4 \times 2$ باشد. آنگاه **B** دارای $24 = 3 \times 4 \times 2$ عنصر است. این 24 عنصر **B** معمولاً به صورت شکل ۱۱ـ ۴ نشان داده می‌شود یعنی به صورت سه لایه که صفحه‌ها نامیده شده ظاهر می‌شوند و در آن هر صفحه شامل آرایهٔ مستطیلی 4×2 از عناصر با اندیس سوم مساوی است. بنابراین سه اندیس یک عنصر در آرایه سه‌بعدی به ترتیب سطر، ستون و صفحهٔ Page آن عنصر نامیده می‌شود.



شکل ۱۱ـ ۴

دو روش ذخیرهٔ **B** در آرایهٔ یک بعدی شکل ۱۲ـ ۴ نشان داده شده است. ملاحظه می‌کنید که پیکانها در شکل ۱۱ـ ۴ روش ستونی قرار گرفتن عناصر را نمایش می‌دهند.



شکل ۱۲ـ۴

در حالت کلی تعریف آرایه‌های چندبعدی اجازه می‌دهد که کرانهای پائین مخالف 1 باشد. فرض کنید C چنین آرایهٔ n بعدی‌ای باشد. همانند قبل، مجموعهٔ اندیس برای هر بعد C شامل اعداد صحیح متوالی از کران پائین تا کران بالای بعد است. طول L_i بعد i آرایهٔ C برابر تعداد عناصر مجموعهٔ اندیس‌ها است همچنین L_i را می‌توان مانند قبل از فرمول

$$L_i = \text{upper bound} - \text{lower bound} + 1 \qquad (۴\text{ـ}۶)$$

برای هر اندیس معین K_i به دست آورد، اندیس مؤثر E_i از L_i برابر تعداد اندیسهایی است که قبل از K_i در مجموعه اندیس قرار دارد و E_i را می‌توان با استفاده از فرمول

$$E_i = K_i - \text{lower bound} \qquad (۴\text{ـ}۷)$$

بدست آورد. بنابراین اگر C به روش ستونی ذخیره شده باشد آدرس $LOC(C[K_1, K_2, \ldots, K_N])$ یک عنصر دلخواه C را می‌توان با استفاده از فرمول

$$Base(C) + w[(((\ldots(E_N L_{N-1} + E_{N-1})L_{N-2}) + \ldots + E_3)L_2 + E_2)L_1 + E_1] \qquad (۴\text{ـ}۸)$$

و اگر C به روش سطری ذخیره شده باشد، با استفاده از فرمول

$$Base(C) + w[(\ldots((E_1 L_2 + E_2)L_3 + E_3)L_4 + \ldots + E_{N-1})L_N + E_N] \qquad (۴\text{ـ}۹)$$

به دست آورد. یکبار دیگر متذکر می‌شویم که Base(C) آدرس عنصر اول C و W تعداد کلمات در خانهٔ حافظه یا طول کلمه را نشان می‌دهد.

مثال ۱۴ـ ۴

فرض کنید **MAZE** یک آرایه سه‌بعدی باشد که به صورت زیر تعریف می‌شود :

MAZE(2:8, − 4:1, 6:10)

در آن‌صورت طول سه‌بعد **MAZE** به ترتیب عبارتند از :

$$L_1 = 8 - 2 + 1 = 7, \qquad L_2 = 1 - (-4) + 1 = 6, \qquad L_3 = 10 - 6 + 1 = 5$$

بنابراین **MAZE** دارای $L_1.L_2.L_3 = 7.6.5 = 210$ عنصر است.

فرض کنید زبان برنامه‌نویسی، **MAZE** را به روش سطری در حافظه ذخیره می‌کند، همچنین فرض کنید **Base(MAZE) = 200** و **W=4** کلمه در خانهٔ حافظه وجود داشته باشد. آدرس یک عنصر **MAZE** به عنوان مثال **MAZE[5, − 1, 8]** به صورت زیر محاسبه می‌شود. اندیسهای مؤثر به‌ترتیب عبارتند از :

$$E_1 = 5 - 2 = 3, \qquad E_2 = -1 - (-4) = 3, \qquad E_3 = 8 - 6 = 2$$

با استفاده از فرمول (۹ـ ۴) برای روش سطری داریم :

$$E_1L_2 = 3 \cdot 6 = 18$$
$$E_1L_2 + E_2 = 18 + 3 = 21$$
$$(E_1L_2 + E_2)L_3 = 21 \cdot 5 = 105$$
$$(E_1L_2 + E_3)L_3 + E_3 = 105 + 2 = 107$$

بنابراین :

LOC(MAZE[5, − 1, 8]) = 200 + 4(107) = 200 + 428 = 628

۱۰ـ ۴ اشاره‌گرها، آرایه‌های نوع اشاره‌گر

فرض کنید **DATA** آرایه دلخواهی باشد. متغیر **P** یک **اشاره‌گر** نامیده می‌شود هرگاه **P** به یک عنصر در **DATA** "اشاره کند" یعنی هرگاه **P** دارای آدرس یک عنصر **DATA** باشد. بطور مشابه آن را یک **آرایهٔ از نوع اشاره‌گر** گویند اگر هر عنصر آرایهٔ **PTR** یک اشاره‌گر باشد. اشاره‌گرها و آرایه‌های نوع اشاره‌گر برای آسانی پردازش اطلاعات در **DATA** مورد استفاده قرار می‌گیرند. این بخش، این ابزار مفید را در پوشش یک مثال توضیح می‌دهد.

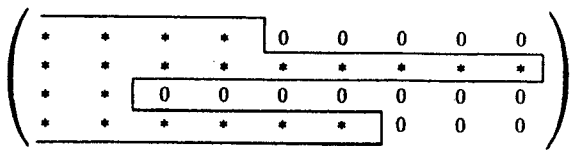
سازمانی را درنظر بگیرید که لیست اعضایش را به چهارگروه تقسیم می‌کند که در آن هر گروه شامل یک لیست الفبایی از اعضایی است که در یک منطقهٔ معین زندگی می‌کنند. شکل ۱۳ـ ۴ یک چنین لیستی را نشان می‌دهد.

Group 1	Group 2	Group 3	Group 4
Evans	Conrad	Davis	Baker
Harris	Felt	Segal	Cooper
Lewis	Glass		Ford
Shaw	Hill		Gray
	King		Jones
	Penn		Reed
	Silver		
	Troy		
	Wagner		

شکل ۱۳ـ ۴

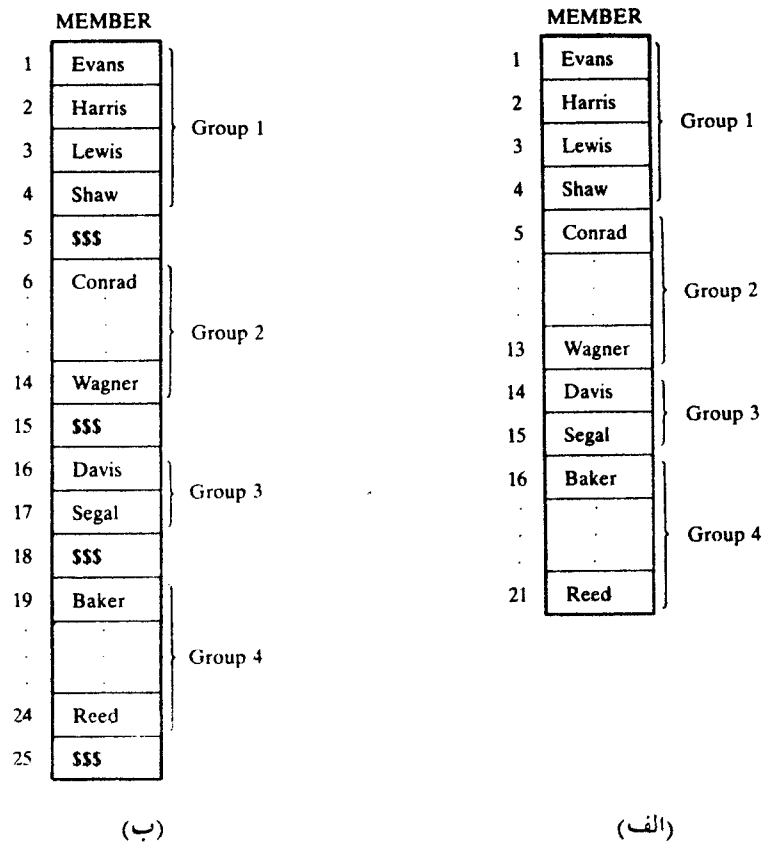
ملاحظه می‌کنید که در لیست بالا 21 نفر وجود دارد و هر گروه بترتیب شامل 4، 9، 2 و 6 نفر است.

فرض کنید لیست اعضا با حفظ گروههای مختلف در حافظه ذخیره می‌شود. یک راه برای رسیدن به این منظور استفاده از یک آرایهٔ n × 4، دوبعدی است که هر سطر شامل یک گروه است یا یک آرایهٔ n×4، دوبعدی است که هر ستون شامل یک گروه است. هرچند این ساختمان داده اجازهٔ دسترسی به تک‌تک گروهها را می‌دهد ولی هنگامی که اندازهٔ گروه خیلی بزرگ می‌شود بیهوده حافظهٔ زیادی مصرف خواهد شد. به‌ویژه اینکه داده‌های شکل ۱۳ـ ۴ احتیاج به دست‌کم یک آرایهٔ 9× 4 یا 4 ×9 عنصری برای ذخیرهٔ 21 نام خواهد داشت که تقریباً دو برابر حافظهٔ موردنیاز است. شکل ۱۴ـ ۴ نمایش آرایهٔ 9 × 4 را نشان می‌دهد. در این نمودار ستاره‌ها عناصر داده‌ای و صفرها خانه‌های حافظهٔ بلااستفاده را نشان می‌دهند. آرایه‌هایی که سطرها یا ستون‌هایش با تعداد مختلفی از عناصر داده‌ای شروع می‌شود و با خانه‌های حافظه مصرف نشده پایان می‌یابد آرایه‌های **پله‌ای** یا **دندانه‌دار** نام دارند.



شکل ۱۴ـ ۴ آرایه‌های دندانه‌دار (پله‌ای)

راه دیگری که می‌توان لیست اعضا را در حافظه ذخیره کرد در شکل ۱۵ـ ۴ **(الف)** به تصویر کشیده شده است یعنی یک گروه پس از گروه دیگرِ لیست، در یک آرایهٔ خطی قرار می‌گیرد.



شکل ۱۵ـ ۴

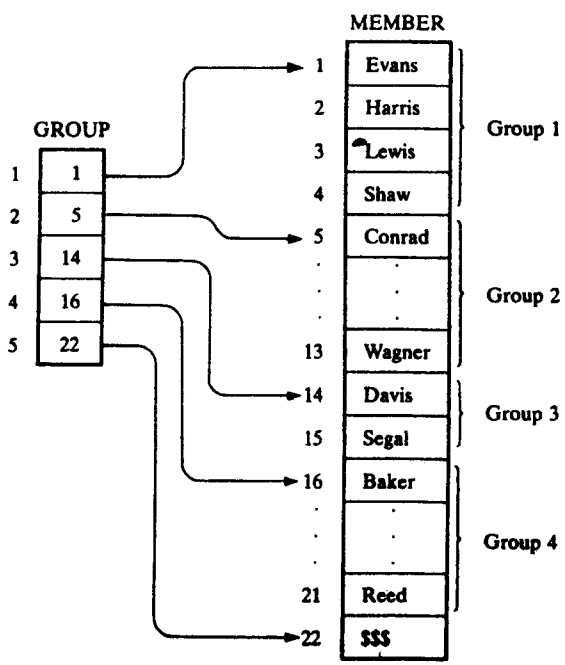
واضح است از نظر حافظه این روش کارا است. همچنین تمام لیست قابل پردازش است به عنوان مثال براحتی می‌توان تمام نام‌های داخل لیست را چاپ کرد. ازطرف دیگر، هیچ راهی برای دسترسی به یک گروه خاص وجود ندارد مثلاً هیچ راهی برای پیداکردن و چاپ نامهای داخل گروه سوم به تنهایی وجود ندارد.

یک نسخهٔ اصلاح شدهٔ روش بالا در شکل ۱۵ـ ۴ (ب) به تصویر درآمده است. به بیان دیگر، نام‌ها گروه به گروه در یک آرایهٔ خطی لیست شده‌اند بجز اینکه اکنون نگهبان یا علامتی نظیر سه علامت دلار در اینجا بکار گرفته شده است که مبین پایان یک گروه است، این روش تنها از تعداد اندکی حافظهٔ اضافی (یک حافظه برای هر گروه)، استفاده می‌کند اما اکنون به هر گروه خاص می‌توان دسترسی پیدا کرد. برای

مثال اکنون یک برنامه‌نویس می‌تواند آن دسته از نامهای گروه سوم را که پس از نگهبان دوم یا قبل از نگهبان سوم قرار دارند پیدا کرده چاپ کند. عیب اصلی این نمایش آن است که همچنان از اول لیست باید پیمایش شود تا گروه سوم را شناسایی کند. به بیان دیگر گروههای مختلف با این نمایش مشخص نمی‌شوند.

آرایه‌های نوع اشاره‌گر

دو ساختمان دادهٔ شکل ۱۵ـ ۴ که از نظر حافظه کارا می‌باشند را می‌توان به‌آسانی اصلاح کرد طوری که بتوان تک‌تک گروهها را شماره‌گذاری کرد. این کار با استفاده از یک آرایهٔ نوع اشاره‌گر، در اینجا GROUP انجام می‌شود که شامل مکان گروههای مختلف یا به‌ویژه مکان عناصر اول گروههای مختلف می‌باشد. شکل ۱۶ـ ۴ چگونگی اصلاح شکل ۱۵ـ ۴ (الف) را نشان می‌دهد.



شکل ۱۶ـ ۴

ملاحظه می‌کنید که GROUP[L] و GROUP[L + 1] – 1 به ترتیب شامل عنصرهای اول و آخر

گروه L است. دیده می‌شود که GROUP[5] به نگهبان لیست اشاره می‌کند و GROUP[5] – 1 مکان عنصر آخر گروه 4 را به دست می‌دهد.

مثال ۱۵ـ۴

فرض کنید بخواهیم تنها نامهای گروه L ام شکل ۱۶ـ۴ را چاپ کنیم که مقدار L بخشی از ورودی است. از آنجا که GROUP[L] و GROUP[L + 1] – 1 به ترتیب شامل مکانهای نام اول و آخر گروه L است. قطعهٔ برنامهٔ زیر این کار را انجام می‌دهد :

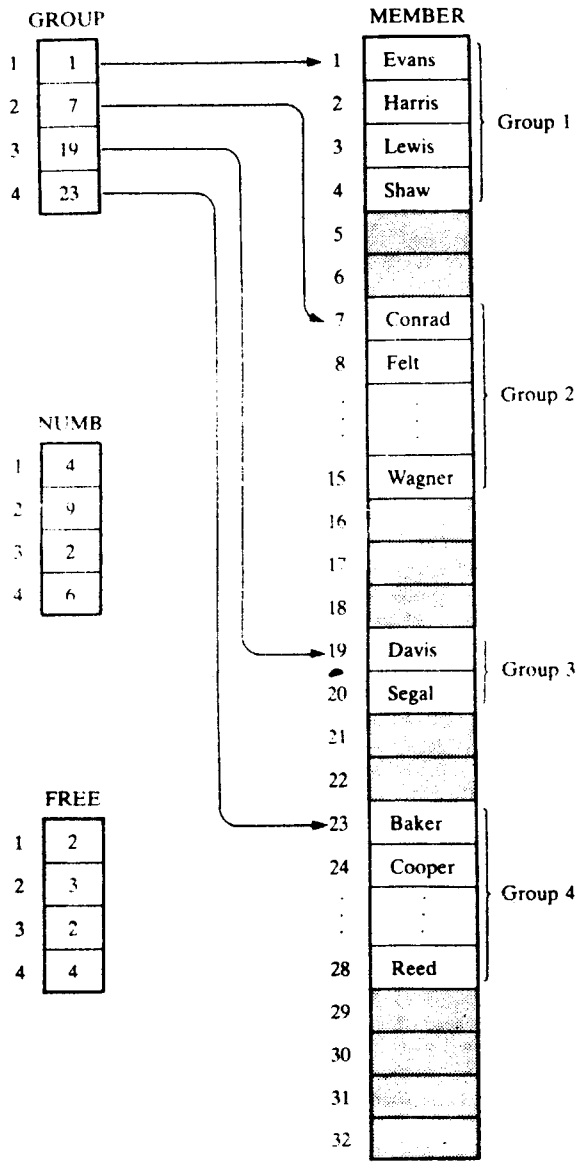
```
1. Set FIRST := GROUP[L] and LAST := GROUP[L + 1] – 1.
2. Repeat for K = FIRST to LAST:
       Write: MEMBER[K].
   [End of loop.]
3. Return.
```

سادگی این قطعه برنامه از این واقعیت ناشی می‌شود که آرایهٔ نوع اشاره‌گر GROUP گروه L ام را مشخص می‌کند. متغیرهای FIRST و LAST اساساً برای سهولت در نمادگذاری بکار گرفته شده‌اند.

ساختمان دادهٔ شکل ۱۶ـ۴ با یک تغییر کوچک در شکل ۱۷ـ۴ نشان داده شده است که در آن خانه‌های حافظهٔ بلااستفاده با سایه مشخص شده است. ملاحظه می‌کنید که اکنون بین گروه‌ها چند خانهٔ خالی وجود دارد. بنابراین یک عنصر جدید می‌تواند در یک گروه اضافه شود بدون آن که احتیاج به جابجایی عناصر گروه دیگری باشد. با استفاده از این ساختمان داده به یک آرایه NUMB احتیاج است که تعداد عناصر هر گروه را به دست می‌دهد. ملاحظه می‌شود که GROUP[K + 1] – GROUP[K] تعداد کل حافظه موجود برای گروه K است. بنابراین

$$FREE[K] = GROUP[K + 1] - GROUP[K] - NUMB[K]$$

تعداد خانه حافظهٔ خالی بعد از GROUP K است. گاهی اوقات بهتر است به صورت صریح آرایهٔ اضافی FREE را تعریف کنیم.



شکل ۱۷ـ ۴

مثال ۱۶ـ ۴

مجدداً فرض کنید بخواهیم تنها نامهای گروه L ام را چاپ کنیم که در آن L بخشی از ورودی است اما اکنون به صورت شکل ۱۷ـ ۴ ذخیره می‌شوند. ملاحظه می‌کنید که

GROUP[L] + NUMB[L] − 1 و **GROUP[L]**

به ترتیب شامل مکان نامهای اول و آخر گروه **L** است. بنابراین قطعه برنامهٔ زیر، این عمل را انجام می‌دهد.

```
1. Set FIRST := GROUP[L] and LAST := GROUP[L] + NUMB[L] − 1.
2. Repeat for K = FIRST to LAST:
       Write: MEMBER[K].
   [End of loop.]
3. Return.
```

متغیرهای **FIRST** و **LAST** اساساً برای سهولت در نمادگذاری بکار رفته‌اند.

۱۱ـ۴ رکوردها؛ ساختارهای رکوردی

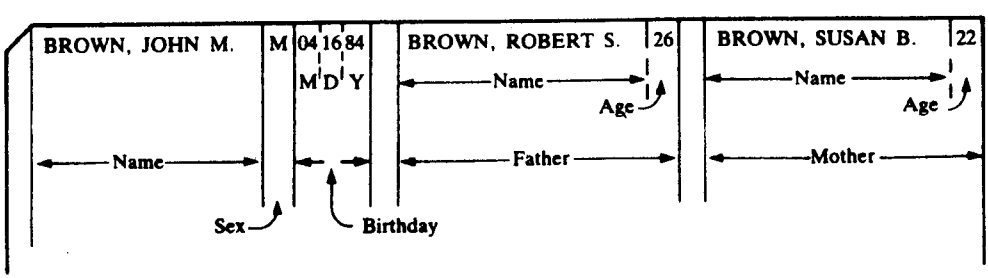
مجموعه‌هایی از داده‌ها هستند که غالباً به صورت سلسله مراتب فیلد، رکورد و فایل سازماندهی می‌شوند. به‌ویژه این که، یک **رکورد** مجموعه‌ای از اقلام داده‌ای مرتبط به هم است که هر یک از این اقلام، **فیلد** یا **خصیصه** نامیده می‌شوند و یک **فایل** مجموعه‌ای از رکوردهای مشابه است. هر فیلد می‌تواند متشکل از چند زیرفیلد باشد. آن دسته از فیلدها که غیرقابل تجزیه هستند **فیلدهای ابتدایی** یا **اتمی** یا **اسکالر** نامیده می‌شوند. **شناسه** نامی است که به اقلام داده‌ای متعدد داده می‌شود.

اگرچه یک رکورد مجموعه‌ای از اقلام داده‌ای بنام فیلد است اما بین رکورد و یک آرایه خطی تفاوتهای به شرح زیر وجود دارد:

(الف) یک رکورد می‌تواند مجموعه‌ای از داده‌های غیرهمجنس باشد یعنی فیلدهای یک رکورد می‌توانند از نوعهای مختلف باشند اما عناصر یک آرایه همجنس هستند.

(ب) فیلدهای یک رکورد به وسیله نامهای خصیصه‌شان مشخص می‌شوند از این‌رو بین عناصر آنها می‌تواند هیچ ترتیب طبیعی وجود نداشته باشد در صورتی که در آرایه‌ها عناصر برحسب اندیس مرتب هستند.

تحت رابطهٔ تقسیم فیلد به زیرفیلدها، فیلدهای یک رکورد تشکیل ساختار سلسله مراتبی را می‌دهند که می‌تواند به وسیلهٔ شماره‌های "سطح" به صورتی که درمثال ۱۷ـ۴ و ۱۸ـ۴ داده شده است بیان گردد.



شکل ۱۸ـ۴

مثال ۱۷ـ ۴

فرض کنید یک بیمارستان مشخصات هر نوزاد تازه تولد یافته را در یک رکورد نگهداری می‌کند که حاوی فیلدهای زیر است : نام **Name** ، جنس **Sex** ، روزتولد **Birthday** ،پدر **Father** ، مادر **Mother** . علاوه بر این فرض کنید که روز تولد فیلدهای مرکبی است که دارای‌فیلدهای ماه **Month** ، روز **Day** و سال **Year** است، همچنین پدر و مادر فیلدهای مرکبی هستند که دارای زیرفیلدهای نام **Name** و سن **Age** است. شکل ۱۸ـ ۴ چگونگی بیان چنین رکوردی را نشان می‌دهد.

```
1  Newborn
     2  Name
     2  Sex
     2  Birthday
          3  Month
          3  Day
          3  Year
     2  Father
          3  Name
          3  Age
     2  Mother
          3  Name
          3  Age
```

ساختار رکورد بالا معمولاً به‌صورت فوق‌بیان می‌شود. توجه دارید که نام **Name** در این رکورد سه بار و سن **Age** دو بار ظاهر شده است:

عددِ سمت چپ هر شناسه **شماره سطح** نامیده می‌شود. ملاحظه می‌کنید که پس از هر فیلد مرکب، زیرفیلدهای آن نوشته شده است و سطح زیرفیلدها، 1 واحد بیشتر از سطح فیلدهای مرکب است. علاوه بر این یک فیلد در فیلد مرکب واقع است اگر و فقط اگر بلافاصله با شمارهٔ سطح بزرگتر پس از فیلد مرکب باشد.

بعضی از شناسه‌ها در یک ساختار رکوردی می‌توانند به آرایه‌هایی از عناصر مرتبط باشند. درواقع فرض کنید به جای خط اولِ ساختار بالا

```
1  Newborn(20)
```

را داشته باشیم. این عبارت مبین آن است که فایل از **20** رکورد تشکیل شده است و نمادگذاری اندیسیِ رایج برای تمایز بین رکوردهای مختلف یک فایل بکار می‌رود. به عبارت دیگر می‌نویسیم:

$\mathbf{Newborn_1}$, $\mathbf{Newborn_2}$, $\mathbf{newborn_3}$, ...

یا **Newborn[1], Newborn[2], newborn[3], ...**

که رکوردهای مختلف یک فایل را نشان می‌دهد.

مثال ۱۸ـ ۴

رکوردهای دانشجویی یک کلاس به صورت زیر سازماندهی شده است:

```
1  Student(20)
     2  Name
          3  Last
          3  First
          3  MI (Middle Initial)
     2  Test(3)
     2  Final
     2  Grade
```

شناسهٔ **Student(20)** بیانگر آن است که تعداد دانشجویان **20** نفر است. شناسهٔ **Test(3)** بیانگر آن است که هر دانشجو در سه آزمون امتحانی شرکت کرده است. ملاحظه می‌کنید که برای هر دانشجو **8** فیلد ابتدایی وجود دارد، چون آزمون **Test** سه بار حساب می‌شود. روی هم‌رفته، برای تمام دانشجویان در این ساختار **160** فیلد ابتدایی وجود دارد.

مشخص‌کردن فیلدهای یک رکورد

فرض کنید بخواهیم به اطلاعات یک فیلد در رکورد دسترسی پیدا کنیم. در برخی از این موارد، به‌سادگی نمی‌توان نام اطلاعات فیلد را نوشت چون همین نام ممکن است در مکانهای دیگری از رکورد ظاهر شود. برای مثال سن **Age** در دو مکان رکورد مثال ۱۷ـ ۴ ظاهر شده است، بنابراین برای مشخص‌کردن یک فیلد خاص بایستی بتوان **وضعیت** نام فیلد را با استفاده از نام فیلد مرکب مربوطه در ساختار بیان کرد. این بیانِ **وضعیت فیلد** با استفاده از نقطه (مانند نقطهٔ اعشار) مشخص می‌شود، که فیلد مرکب را از زیرفیلدهایش جدا می‌کند.

مثال ۱۹ـ ۴

(الف) ساختار رکوردی **Newborn** را در مثال ۱۷ـ ۴ درنظر بگیرید. جنس **Sex** و سال **Year** لازم نیست تعیین وضعیت شوند چون هر یک از این فیلدها به صورت منحصربفرد در ساختار معرفی شده‌اند. از طرف دیگر، فرض کنید بخواهیم به سن **Age** پدر **Father** دسترسی پیدا کنیم. این کار به صورت زیر انجام می‌شود:

Newborn. Father. Age یا بطور خلاصه **Father.Age**

به رجوع اول بیان وضعیت کامل می‌گویند. گاهی اوقاع برای روشن‌شدن مطلب شناسه‌های بیان وضعیت به فیلدها اضافه می‌شود.

(ب) فرض کنید جای خط اول در ساختار رکوردی مثال ۱۰ـ ۴ را با عبارت زیر عوض کرده‌ایم :

```
1  Newborn(20)
```

به عبارت دیگر، Newborn بنابه تعریف یک فایل 20 رکوردی باشد. آنگاه هر فیلد به‌طور اتوماتیک یک آرایهٔ 20 عنصری می‌شود. برخی از زبانهای برنامه‌نویسی اجازه می‌دهند به جنس Sex تازه تولد یافتهٔ ششم به صورت زیر دسترسی پیدا کنیم :

Newborn.Sex[6] یا بطور خلاصه Sex[6]

به طور مشابه، به سن Age پدر Father تازه تولد یافتهٔ ششم به صورت زیر دسترسی پیدا کنیم:

Newborn.Father.Age[6] یا بطور خلاصه Father.Age[6]

(ج) ساختار رکوردی دانشجویان را در مثال ۱۸ـ ۴ درنظر بگیرید. از آنجا که دانشجو Student یک فایل 20 عنصری است تمام فیلدها به‌طور اتوماتیک آرایه‌های 20 عنصری می‌شوند. علاوه براین Test یک آرایهٔ دوبعدی می‌شود. به‌خصوص این که، به Test دوم دانشجوی ششم به صورت زیر دسترسی پیدا می‌کنیم:

Student.Test[6,2] یا بطور خلاصه Test[6,2]

ترتیب اندیسها متناظر با ترتیب شناسه‌های بیان وضعیت است. برای مثال

```
Test[3, 1]
```

به آزمون سوم دانشجوی اول دسترسی پیدا نمی‌کنیم بلکه به آزمون اول دانشجوی سوم دسترسی داریم.

توجه کنید : در بعضی از کتاب‌ها برای بیان وضعیت شناسه‌های رکوردها، به جای نماد نقطه‌گذاری بین فیلدها، از نماد تابعی زیر استفاده می‌شود. برای مثال

Age(Father (Newborn)) را به جای Newborn.Father.Age

First(Name (Student[8])) را به جای Student.Name.First[8]

می‌نویسند.

ملاحظه می‌کنید که ترتیب بیان وضعیت شناسه‌ها در نماد تابعی عکس ترتیب آن در نماد نقطه‌گذاری است.

۱۲ـ۴ نمایش رکوردها در حافظه، آرایه‌های موازی

از آنجا که رکوردها می‌توانند شامل داده‌های غیرهمجنس باشند، از این‌رو عناصر یک رکورد را

نمی‌توان در یک آرایه ذخیره کرد. برخی از زبانهای برنامه‌نویسی نظیر PL / I ، PASCAL و COBOL دارای ساختار رکوردی تعبیه شده در زبان هستند.

مثال ۲۰ـ ۴

ساختار رکوردی Newborn مثال ۱۷ـ ۴ را درنظر بگیرید. این رکورد را در زبان PL / I با معرفی به صورت زیر، می‌توان ذخیره کرد که مجموعه‌ای از داده‌ها بنام ساختار Structure تعریف می‌کند :

```
DECLARE  1  NEWBORN,
            2  NAME CHAR(20),
            2  SEX CHAR(1),
            2  BIRTHDAY,
                  3  MONTH FIXED,
                  3  DAY FIXED,
                  3  YEAR FIXED,
            2  FATHER,
                  3  NAME CHAR(20),
                  3  AGE FIXED,
            2  MOTHER
                  3  NAME CHAR(20),
                  3  AGE FIXED;
```

ملاحظه می‌کنید که متغیرهای جنس SEX و سال YEAR منحصر بفرد هستند، از این‌رو برای مراجعه به آنها احتیاج به بیان وضعیت نیست. ازطرف دیگر AGE منحصر بفرد نیست. بنابراین از نماد

FATHER.AGE یا MOTHER.AGE

بسته به این که بخواهیم به سن پدر مراجعه کنیم یا سن مادر استفاده می‌کنیم.

فرض کنید در یک زبان برنامه‌نویسی ساختارهای سلسله مراتبی که در PL / I ، PASCAL و COBOL موجود است وجود نداشته باشد. با این فرض که رکورد شامل داده‌های غیرهمجنس است رکورد را می‌توان در متغیرهای منفردی ذخیره کرد که برای هر فیلد ابتدایی از یک متغیر استفاده می‌کند. از طرف دیگر، فرض کنید خواسته باشیم تمام رکوردهای فایل را ذخیره کنیم. توجه دارید که تمام عناصر داده‌ای که به یک شناسه تعلق دارند از یک نوع هستند. چنین فایلی را می‌توان به صورت مجموعه‌ای از **آرایه‌های موازی** در حافظه ذخیره کرد، به عبارت دیگر عناصر آرایه‌های مختلف که دارای یک اندیس هستند به یک رکورد تعلق دارند. این وضعیت در دو مثال بعدی نشان داده شده است.

مثال ۲۱ـ ۴

فرض کنید لیست اعضا شامل نام NAME، سن AGE، جنس SEX و شماره تلفن PHONE هر عضو

است. می‌توان این فایل را در چهار آرایهٔ موازی **NAME** ، **AGE** ، **SEX** و **PHONE** به صورت نشان داده شده در شکل ۱۹ـ ۴ ذخیره کرد. به عبارت دیگر برای یک اندیس معین **K** ، عنصرهای **NAME[K]** ، **AGE[K]**, **SEX[K]** و **PHONE[K]** به یک رکورد تعلق دارند.

	NAME	AGE	SEX	PHONE
1	John Brown	28	Male	234-5186
2	Paul Cohen	33	Male	456-7272
3	Mary Davis	24	Female	777-1212
4	Linda Evans	27	Female	876-4478
5	Mark Green	31	Male	255-7654
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

شکل ۱۹ـ ۴

مثال ۲۲ـ ۴

بار دیگر رکورد **Newborn** مثال ۱۷ـ ۴ را درنظر بگیرید. می‌توان فایلی متشکل از این‌گونه رکوردها در نُه آرایهٔ خطی به صورت زیر ذخیره کرد :

NAME, SEX, MONTH, DAY, YEAR, FATHERNAME, FATHERAGE, MOTHERNAME, MOTHERAGE

که در آن از هر آرایه، برای یک فیلد ابتدایی استفاده می‌کنیم. در اینجا برای نام **NAME** و سن **AGE** پدر و مادر باید از نامهای متغیر متفاوتی استفاده شود که در مثال قبل لزومی به این کار نبود. مجدداً فرض می‌کنیم که آرایه‌ها موازی هستند یعنی برای اندیس‌ثابت **K**، عناصر زیر به یک رکورد تعلق دارند.

NAME[K], SEX[K], MONTH[K], ..., MOTHERAGE[K]

رکودهایی با طول متغیر

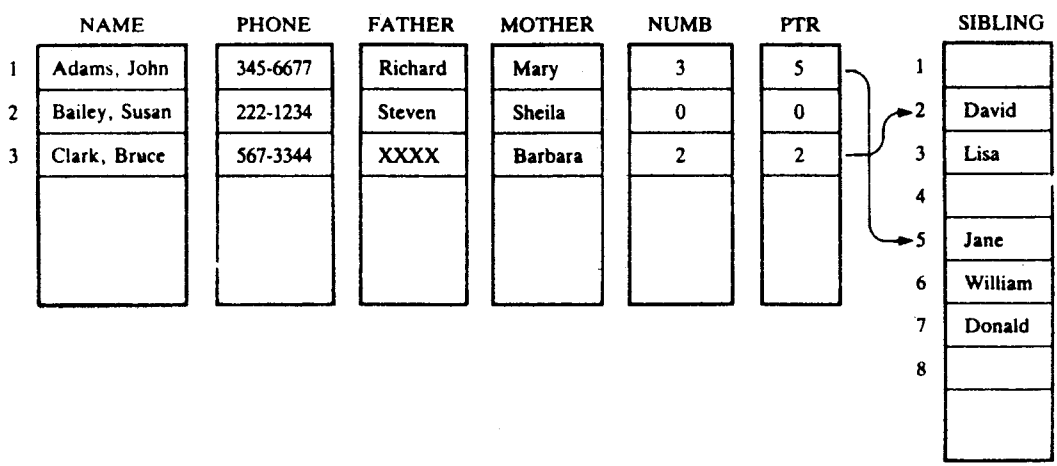
فرض کنید یک مدرسهٔ ابتدایی مشخصات هر دانش‌آموز را که شامل فیلدهای زیراست نگهداری می‌کند. نام **Name** ، شماره تلفن **Telephone Number** ، پدر **Father** ، مادر **Mother** ، دیگر افراد خانواده **Siblings**. در اینجا پدر، مادر و دیگر افراد خانواده به ترتیب شامل نام پدر و مادر دانش‌آموز و یا برادر یا خواهرهای وی است که در همان مدرسه درس می‌خوانند. سه نمونه از این رکوردها می‌تواند به صورت

زیر باشد :

Adams, John;	345-6677;	Richard;	Mary;	Jane, William, Donald
Bailey, Susan;	222-1234;	Steven;	Sheila;	XXXX
Clark, Bruce;	567-3344;	XXXX;	Barbara;	David, Lisa

در اینجا XXXX مبین فوت‌شدن پدر است یا این که با دانش‌آموز زندگی نمی‌کند یا دانش‌آموز برادر یا خواهری در مدرسه ندارد.

رکوردهای بالا مثالی از یک رکورد با طول متغیر است، چون عنصر داده‌ای دیگر افراد خانواده می‌تواند شامل هیچ یا چند اسم باشد. یک راه ذخیرهٔ فایل در آرایه‌ها در شکل ۲۰ـ۴ به تصویر درآمده است که در آن



شکل ۲۰ـ۴

FATHER, PHONE, NAME و MOTHER چهار فیلد اول رکورد را نگه می‌دارند آرایه‌های خطی هستند و آرایه‌های NUMB و PTR به‌ترتیب تعداد و مکان دیگر افراد خانواده را در آرایهٔ SIBLING به دست می‌دهند.

۱۳ـ۴ ماتریسها

بردارها و ماتریسها اصطلاحات ریاضی هستند که به مجموعه‌هایی از اعداد اشاره می‌کنند و به‌ترتیب مشابه آرایه‌های خطی و دو بعدی در کامپیوتر هستند. به بیان دیگر:

(الف) یک بردار **n** عنصری **V** لیستی از **n** عدد است که معمولاً به صورت زیر نشان داده می‌شود :

$$V = (V_1, V_2, ..., V_n)$$

(ب) یک ماتریس **m×n** مانند **A** آرایه‌ای از **m×n** عدد است که در **m** سطر و **n** ستون به صورت زیر چیده شده باشد :

$$A = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ A_{m1} & A_{m2} & \cdots & A_{mn} \end{pmatrix}$$

در مباحث مربوط به بردارها و ماتریسها برای اشاره بـه تک تک اعـداد از اصـطلاح اسکـالر اسـتفاده می‌شود.

یک ماتریس با یک سطر (ستون) را می‌توان به صورت یک بردار درنظر گرفت و به‌طور مشابه، یک بردار را می‌توان به صورت یک ماتریس که تنها، یک سطر (ستون) دارد درنظر گرفت.

یک ماتریس که **n** تعداد سطر یا ستون آن برابر باشد یک **ماتریس مربعی** یا یک **ماتریس n مربعی** نام دارد. **قطر** یا **قطر اصلی** یک ماتریس **n** مربعی **A** شامل عنصرهای A_{nn} ,...., A_{22} , A_{11} است.

در بخش بعد تعدادی از عملیات جبری مربوط به بردارها و ماتریسها مورد بررسی قرار خـواهـد گرفت. آنگاه بدنبال این بخش راههای کارا و موثر ذخیرهٔ انواع معین از ماتریسها، موسوم به ماتریسهای **تُنُک** یا **خلوت Sparse** بررسی خواهند شد.

جبر ماتریسها

فرض کنید **A** و **B** دو ماتریس **m×n** هستند. مجموع **A** و **B** که به صورت **A + B** نوشته می‌شود یک ماتریس **m×n** است که با جمع عناصر متناظر در **A** و **B** به دست می‌آید و حاصلضرب یک اسکالر **K** و ماتریس **A** که به صورت **K . A** نوشته می‌شود یک ماتریس **m × n** است که از ضرب هر عنصر **A** در **K** به دست می‌آید.برای بردارهای **n** عنصری این عملیات بطور مشابه تعریف می‌شود.

بهتر است ضرب ماتریس رانخست با تعریف ضرب اسکالر دو بردار شرح دهیم.

فرض کنید **U** و **V** دو بردار **n** عنصری باشند. آنگاه حاصلضرب اسکالر **U** و **V** که به صورت **U.V** نوشته می‌شود عددی اسکالر است که از ضرب عنصرهای **U** در عنصرهای متناظرش در **V** و آنگاه جمع آنها به دست می‌آید:

$$U \cdot V = U_1V_1 + U_2V_2 + \cdots + U_nV_n = \sum_{k=1}^{n} U_kV_k$$

تأکید می‌کنیم که **U.V** یک عدد یا یک اسکالر است نه یک بردار.

اکنون فرض کنید **A** یک ماتریس **M × P** و **B** یک ماتریس **P × n** باشد. **حاصلضرب A و B** را به صورت **AB** می‌نویسیم که یک ماتریس **mxn** بنام **C** است که عنصر **ij** ام آن به صورت زیر بـه دست می‌آید :

$$C_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j} + \cdots + A_{ip}B_{pj} = \sum_{k=1}^{p} A_{ik}B_{kj}$$

به عبارت دیگر، C_{ij} برابر است با حاصلضرب اسکالر سطر **i** ام **A** و ستون **j** ام **B**.

مثال ۲۳ـ ۴

(الف) فرض کنید

$$A = \begin{pmatrix} 1 & -2 & 3 \\ 0 & 4 & 5 \end{pmatrix} \qquad \text{و} \qquad B = \begin{pmatrix} 3 & 0 & -6 \\ 2 & -3 & 1 \end{pmatrix}$$

آنگاه

$$A + B = \begin{pmatrix} 1+3 & -2+0 & 3+(-6) \\ 0+2 & 4+(-3) & 5+1 \end{pmatrix} = \begin{pmatrix} 4 & -2 & -3 \\ 2 & 1 & 6 \end{pmatrix}$$

$$3A = \begin{pmatrix} 3\cdot 1 & 3\cdot(-2) & 3\cdot 3 \\ 3\cdot 0 & 3\cdot 4 & 3\cdot 5 \end{pmatrix} = \begin{pmatrix} 3 & -6 & 9 \\ 0 & 12 & 15 \end{pmatrix}$$

(ب) فرض کنید **U = (1, − 3, 4, 5), V = (2, − 3, − 6, 0)** و **W = (3, −5, 2, − 1)** آنگاه :

$$U \cdot V = 1\cdot 2 + (-3)\cdot(-3) + 4\cdot(-6) + 5\cdot 0 = 2 + 9 - 24 + 0 = -13$$
$$U \cdot W = 1\cdot 3 + (-3)\cdot(-5) + 4\cdot 2 + 5\cdot(-1) = 3 + 15 + 8 - 5 = 21$$

(ج) فرض کنید

$$A = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} \qquad \text{و} \qquad B = \begin{pmatrix} 2 & 0 & -4 \\ 3 & 2 & 6 \end{pmatrix}$$

ماتریس حاصلضرب **AB** تعریف می‌شود و یک ماتریس **3 × 2** است. عناصر سطر اول **AB** به ترتیب با ضرب سطر اول **A** در هر یک از ستون‌های **B** به دست می‌آید:

$$\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}\begin{pmatrix} 2 & 0 & -4 \\ 3 & 2 & 6 \end{pmatrix} = \begin{pmatrix} 1\cdot 2 + 3\cdot 3 & 1\cdot 0 + 3\cdot 2 & 1\cdot(-4) + 3\cdot 6 \end{pmatrix} = \begin{pmatrix} 11 & 6 & 14 \end{pmatrix}$$

به‌طور مشابه، عناصر سطر دوم **AB** به ترتیب با ضرب سطر دوم **A** در هر یک از ستونهای **B** بدست می‌آید :

$$\begin{pmatrix}1 & 3\\ 2 & 4\end{pmatrix}\begin{pmatrix}2 & 0 & -4\\ 3 & 2 & 6\end{pmatrix}=\begin{pmatrix}11 & 6 & 14\\ 2\cdot2+4\cdot3 & 2\cdot0+4\cdot2 & 2\cdot(-4)+4\cdot6\end{pmatrix}=\begin{pmatrix}11 & 6 & 14\\ 16 & 8 & 16\end{pmatrix}$$

به عبارت دیگر :

$$AB=\begin{pmatrix}11 & 6 & 14\\ 16 & 8 & 16\end{pmatrix}$$

الگوریتم زیر، حاصلضرب **AB** دو ماتریس **A** و **B** را پیدا می‌کند که به صورت آرایه‌های دو بعدی ذخیره می‌شوند. الگوریتم‌های جمع و ضرب اسکالرِ ماتریسها که کاملاً مشابهٔ الگوریتم‌های جمع و ضرب اسکالر بردارها است به عنوان تمرین به دانشجو واگذار می‌شود.

Algorithm 4.7: (Matrix Multiplication) MATMUL(A, B, C, M, P, N)
Let A be an M × P matrix array, and let B be a P × N matrix array. This algorithm stores the product of A and B in an M × N matrix array C.

```
1.  Repeat Steps 2 to 4 for I = 1 to M:
2.      Repeat Steps 3 and 4 for J = 1 to N:
3.          Set C[I, J] := 0. [Initializes C[I, J].]
4.          Repeat for K = 1 to P:
                C[I, J] := C[I, J] + A[I, K] * B[K, J]
            [End of inner loop.]
        [End of Step 2 middle loop.]
    [End of Step 1 outer loop.]
5.  Exit.
```

پیچیدگی الگوریتم ضرب ماتریسها با شمارش **C** تعداد عمل ضرب اندازه‌گیری می‌شود. دلیل این که پیچیدگی جمع در چنین الگوریتمهایی محاسبه نمی‌شود آن است که عمل ضرب در کامپیوتر زمان به‌مراتب بیشتری نسبت به جمع به خود اختصاص می‌دهد. پیچیدگی الگوریتم **4.7** بالا برابر است با :

$$C = m.n.p$$

نتیجهٔ فوق از این واقعیت ناشی می‌شود که چون مرحلهٔ **4** فقط شامل یک عمل ضرب است **m.n.p** بار اجرا می‌شود. تحقیقات گسترده‌ای صورت گرفته است تا الگوریتم‌هایی برای ضرب ماتریسها پیدا کنند تا بتوانند تعداد عملیات ضرب را به حداقل برسانند. مثال بعدی یک نتیجهٔ جالب توجه و شگفت‌آوری را در این زمینه به‌دست می‌دهد.

مثال ۲۴ـ۴

فرض کنید **A** و **B** دو ماتریس **2×2** باشند. داریم :

$$A=\begin{pmatrix}a & b\\ c & d\end{pmatrix},\quad B=\begin{pmatrix}e & f\\ g & h\end{pmatrix}\quad \text{و}\quad AB=\begin{pmatrix}ae+bg & af+bh\\ ce+dg & cf+dh\end{pmatrix}$$

در الگوریتم 4.7 حاصلضرب ماتریسی **AB** با استفاده از **C = 2.2.2 = 8** عمل ضرب بدست می‌آید. از طرف دیگر **AB** را می‌توان به صورت زیر به دست آورد که در آن تنها از **7** عمل ضرب استفاده می‌شود.

$$AB = \begin{pmatrix} (1+4-5+7) & (3+5) \\ (2+4) & (1+3-2+6) \end{pmatrix}$$

(1) $(a+d)(e+h)$
(2) $(c+d)e$
(3) $a(f-h)$
(4) $d(g-e)$
(5) $(a+b)h$
(6) $(c-a)(e+f)$
(7) $(b-d)(g+h)$

بعضی از نسخه‌های زبان برنامه‌نویسیِ **BASIC** دارای عملیات ماتریسی تعبیه شده در زبان هستند. به‌ویژه اینکه، در زیر چند دستور مجاز زبان **BASIC** ارائه شده است که در آن **A** و **B** آرایه‌های دوبعدی هستند که دارای ابعاد متناسب با هم می‌باشند و **K** یک اسکالر است :

```
MAT C = A + B
MAT D = (K) * A
MAT E = A * B
```

هر دستور با کلمهٔ کلیدی **MAT** که مبین انجام عملیات ماتریسی است، شروع می‌شود. بدین ترتیب در دستورهای بالا **C** جمع ماتریسی **A** و **B** است و **D** ضرب اسکالر ماتریس **A** در اسکالر **K** و **E** ضرب ماتریسی **A** و **B** خواهد بود.

۱۴ـ۴ ماتریسهای خلوت

ماتریسهایی که دارای تعداد نسبتاً زیادی درایهٔ صفر باشد **ماتریس خلوت** یا **ماتریس تُنُک** نام دارد. دو نمونهٔ کلی از ماتریسهای خلوتِ **n** مربعی که در اکثر کاربردها ظاهر می‌شوند در شکل ۲۱ـ۴ نشان داده شده است. گاهی اوقات رسم بر این است که بلاک‌های دارای صفر این‌گونه ماتریسها را مانند شکل ۲۱ـ۴ حذف می‌کنند.