$$\begin{pmatrix} 4 & & & & \\ 3 & -5 & & & \\ 1 & 0 & 6 & & \\ -7 & 8 & -1 & 3 & \\ 5 & -2 & 0 & 2 & -8 \end{pmatrix}$$

(الف) ماتریس مثلثی

$$\begin{pmatrix} 5 & -3 & & & & & \\ 1 & 4 & 3 & & & & \\ & 9 & -3 & 6 & & & \\ & & 2 & 4 & -7 & & \\ & & & 3 & -1 & 0 & \\ & & & & 6 & -5 & 8 \\ & & & & & 3 & -1 \end{pmatrix}$$

(ب) ماتریس سه قطری

شکل ۲۱ـ۴

ماتریس اول که تمام درایه‌های بالای قطر اصلی آن صفر است یا به بیان دیگر، درایه‌های غیرصفر آن تنها می‌توانند روی، یا پائین قطر اصلی ظاهر شوند یک **ماتریس (پائین) مثلثی** نام دارد. ماتریس دوم که درایه‌های غیرصفر آن تنها می‌توانند روی قطر یا عناصر بلافاصله بالا و پائین قطر ظاهر شوند یک ماتریس **سه قطری** نام دارد.

ممکن است روش طبیعی نمایش ماتریسها در حافظه به صورت آرایه‌های دوبعدی، برای ماتریسهای خلوت مناسب نباشد. یعنی، تنها با ذخیرهٔ درایه‌هایی که غیرصفر هستند می‌توان در مصرف حافظه صرفه‌جویی کرد. برای ماتریسهای مثلثی این عمل در مثال زیر نشان داده شده است. حالتهای دیگر در قسمت مسأله‌های حل شده توضیح داده خواهد شد.

مثال ۲۵ـ۴

فرض کنید خواسته باشیم آرایهٔ مثلثی **A** شکل ۲۲ـ۴ را در حافظهٔ کامپیوتر ذخیره کنیم. واضح است که ذخیرهٔ درایه‌های بالای قطر اصلی **A** کاری بیهوده است چون می‌دانیم تمام این عناصر صفر هستند. از این‌رو تنها درایه‌های دیگر **A** را که در شکل ۲۲ـ۴ با پیکان مشخص شده است در آرایهٔ خطی **B** ذخیره می‌کنیم، یعنی قرار می‌دهیم:

$$B[1] = a_{11} \quad B[2] = a_{21}, \quad B[3] = a_{22}, \quad B[3] = a_{31}, \quad \ldots$$

نخست ملاحظه می‌کنید که **B** شامل

$$1 + 2 + 3 + 4 + \ldots + n = \frac{1}{2}\, n\,(n+1)$$

عنصر است که تقریباً نصف عنصر یک آرایهٔ $n \times n$ دوبعدی است. از آنجا که در برنامه‌های خود به مقدار

a_{JK} احتیاج داریم، از این‌رو فرمولی را به دست می‌آوریم که عدد صحیح L را برحسب J، K معین می‌کند که در آن

$$B(L) = a_{JK}$$

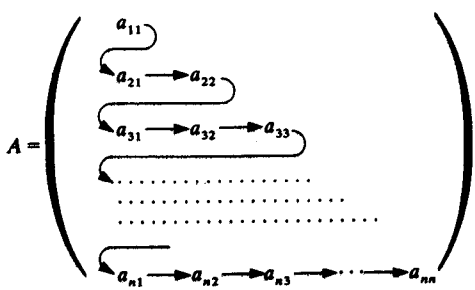
ملاحظه می‌شود که L تعداد عناصر داخل لیست حداکثر تا a_{JK} و خود آن را نمایش می‌دهد.
اکنون تعداد

$$1 + 2 + 3 + \ldots + (J - 1) = \frac{J(J-1)}{2}$$

عنصر در سطرهای بالای a_{JK} وجود دارد و K عنصر در سطر j وجود دارد که حداکثر تا a_{JK} و خود a_{JK} را شامل است. بنابراین

$$L = \frac{J(J-1)}{2} + K$$

اندیسی را می‌دهد که به کمک آن می‌توان به مقدار a_{JK} از آرایهٔ خطی B دسترسی پیدا کرد.



شکل ۲۲ـ ۴

مسأله‌های حل شده

آرایه‌های خطی

مسأله ۱ـ ۴ : آرایه‌های خطی AAA(5:50) , BBB(− 5:10) و CCC(18) را درنظر بگیرید :

(الف) تعداد عناصر هر آرایه را پیدا کنید.

(ب) فرض کنید Base(AAA) = 300 و W = 4 کلمه در خانه‌های حافظه برای AAA باشد. آدرس AAA[35], AAA[15] و AAA[55] را پیدا کنید.

حل : (الف) تعداد عناصر آرایه برابر طول آرایه است. از این‌رو از فرمول زیر استفاده می‌کنیم :

$$Length = UB - LB + 1$$

بنابراین :

$$\text{Length(AAA)} = 50 - 5 + 1 = 46$$
$$\text{Length(BBB)} = 10 - (-5) + 1 = 16$$
$$\text{Length(CCC)} = 18 - 1 + 1 = 18$$

توجه دارید که **Length(CCC) = UB** چون **LB = 1**.

(ب) از فرمول زیر استفاده می‌کنیم :

$$\text{LOC(AAA[K])} = \text{Base(AAA)} + w(K - LB)$$

از این‌رو

$$\text{LOC(AAA[15])} = 300 + 4(15 - 5) = 340$$

$$\text{LOC(AAA[35])} = 300 + 4(35 - 5) = 420$$

AAA[55] عنصر **AAA** نیست چون **55** بزرگتر از **UB = 50** است.

مسأله ۲ـ ۴ : فرض کنید یک شرکت یک آرایهٔ خطی **YEAR(1920:1970)** برای مشخصات کارمندان دارد که در آن **YER[K]** شامل تعداد کارمندانی است که در سال **K** متولد شده‌اند. برای هر یک از کارهای زیر یک قطعه برنامه بنویسید:

(الف) هر سالی را که در آن سال کارمندی متولد نشده، چاپ کند.

(ب) **NNN** تعداد سالهایی را که در آنها هیچ کارمندی متولد نشده، پیدا کند.

(ج) **N50** تعداد کارمندانی را که تا پایان سال حداقل **50** سال سن دارند پیدا کند. فرض می‌شود سال جاری سال **1984** است.

(د) **NL** تعداد کارمندانی را پیدا کنید که تا پایان سال، حداقل **L** سال سن دارند فرض می‌شود سال جاری سال **1984** است.

حل : هر قطعه برنامه، آرایه را به صورت زیر پیمایش می‌کند.

(الف)

```
1. Repeat for K = 1920 to 1970:
       If YEAR[K] = 0, then: Write: K.
   [End of loop.]
2. Return.
```

(ب)

```
1. Set NNN := 0.
2. Repeat for K = 1920 to 1970:
       If YEAR[K] = 0, then: Set NNN := NNN + 1.
   [End of loop.]
3. Return.
```

(ج) می‌خواهیم تعداد کارمندانی را که در سال **1934** یا قبل از آن متولد شده‌اند به دست آوریم.

We want the number of employees born in 1934 or earlier.

```
1. Set N50 := 0.
2. Repeat for K = 1920 to 1934:
       Set N50 := N50 + YEAR[K].
   [End of loop.]
3. Return.
```

(د) می‌خواهیم تعداد کارمندانی را که در سال **1984 − L** یا قبل از آن متولد شده‌اند به دست آوریم.

We want the number of employees born in year 1984 − L or earlier.

```
1. Set NL := 0 and LLL := 1984 − L.
2. Repeat for K = 1920 to LLL:
       Set NL := NL + YEAR[K].
   [End of loop.]
3. Return.
```

مسأله ۳ـ ۴ : فرض کنید **A** یک آرایهٔ 10 عنصری با مقادیر a_1 , a_2 ,..., a_{10} باشد. مقادیر **A** را پس از هر حلقه تعیین کنید.

(الف)

```
Repeat for K = 1 to 9:
    Set A[K + 1] := A[K].
[End of loop.]
```

(ب)

```
Repeat for K = 9 to 1 by −1:
    Set A[K + 1] := A[9].
[End of loop.]
```

حل: توجه دارید که در قسمت (الف) اندیس **K** از ۱ تا ۹ تغییر می‌کند اما در قسمت (ب) به‌ترتیبِ عکس از ۹ تا ۱ تغییر می‌کند.

(الف) نخست **A[1] := A[2]** که **A[2]** = a_1 می‌شود که a_1 مقدار **A[1]** است.

آنگاه **A[2] := A[3]** که **A[3]** = a_1 می‌شود که a_1 مقدار فعلی **A[2]** است.

آنگاه **A[3] := A[4]** که **A[4]** = a_1 می‌شود که a_1 مقدار فعلی **A[3]** است.

بدین ترتیب هر عنصر آرایهٔ **A** دارای مقدار a_1 است که همان مقدار اولیهٔ **A[1]** است.

(ب) نخست **A[9] := A[10]** که **A[10]** = a_9 قرار داده می‌شود.

آنگاه **A[8] := A[9]** که **A[9]** = a_8 قرار داده می‌شود.

آنگاه **A[7] := A[8]** که **A[8]** = a_7 قرار داده می‌شود، همینطور تا آخر.

بدین ترتیب هر مقدار در **A** به خانهٔ بعدی منتقل می‌شود. در پایان حلقه همچنان داریم **A[1]** = a_1.

توجه کنید : این مثال توضیحی بر این واقعیت است که در الگوریتم اضافه‌کردن یعنی الگوریتم ۴ـ ۴ ، درست مانند حلقهٔ **(ب)** در بالا، عنصرها به ترتیبِ عکس به طرف پائین منتقل می‌شوند.

مسأله ۴ـ ۴ : آرایهٔ خطی **NAME** شکل ۲۳ـ ۴ را که به صورت الفبایی مرتب شده درنظر بگیرید.

(الف) هرگاه بخواهیم **Johnson , Brown** و **Peters** را در سه زمان مختلف به **NAME** اضافه کنیم تعداد عناصری را

که باید جابجا شوند تعیین کنید.

(ب) هرگاه بخواهیم سه نام به یک باره اضافه شود چند عنصر باید جابجا شوند؟

(ج) یک شرکت تلفن عمل اضافه‌کردن را در دفترچهٔ راهنمای تلفن چگونه انجام می‌دهد؟

حل : (الف) اضافه‌کردن **Brown** مستلزم جابجاشدن **13** عنصر است و عمل اضافه‌کردن **Johnson** مستلزم جابجاشدن **1** عنصر و عمل اضافه‌کردن **Peters** مستلزم جابجایی **4** عنصر است. مجموعاً **24** عنصر باید جابجا شوند.

(ب) هرگاه بخواهیم عناصر را به یک باره اضافه کنیم، آنگاه لازم است **13** عنصر جابجا شوند. با الگوریتم قبلی هر عنصر تنها یکبار جابجا می‌شود.

(ج) شرکت تلفن یک لیست در گردش برای شماره‌های جدید درست می‌کند و آنگاه یکبار در سال دفترچه راهنمای تلفن را بروز درمی‌آورد و تازه! می‌کند.

	NAME
1	Allen
2	Clark
3	Dickens
4	Edwards
5	Goodman
6	Hobbs
7	Irwin
8	Klein
9	Lewis
10	Morgan
11	Richards
12	Scott
13	Tucker
14	Walton

شکل ۲۳ـ ۴

جستجوکردن، مرتب‌کردن

مسأله ۵ـ ۴ : آرایهٔ خطی مرتب‌شدهٔ الفبایی **NAME** شکل ۲۳ـ ۴ را در نظر بگیرید.

(الف) با استفاده از الگوریتم جستجوی خطی، الگوریتم ۵ـ ۴ ، برای تعیین مکان **Hobbs , Morgan** و **Fisher** چند عمل مقایسه **C** مورد استفاده قرار می‌گیرد؟

(ب) تعیین کنید برای چنین آرایهٔ ذخیره شده‌ای، در الگوریتم چه تغییری ایجاد کنیم تا جستجوی ناموفق از کارایی بیشتری برخوردار شود؟ این تغییر بر روی قسمت (الف) چه تأثیری خواهد داشت؟

حل : (الف) C(Hobbs) = 6 چون **Hobbs** با تمام نامها مقایسه می‌شود این مقایسه از نام **Allen** شروع شده تا این که **Hobbs** در **NAME[6]** پیدا می‌شود.

C(Morgan) = 10 چون **Morgan** در **NAME[10]** ظاهر می‌شود.

C(Fisher) = 15 چون **Fisher** در آغاز در **NAME[15]** قرار گرفته است. و آنگاه **Fisher** با هر نام مقایسه می‌شود تا این که در **NAME[15]** پیدا می‌شود. از این‌رو جستجو ناموفق است.

(ب) ملاحظه کنید که **NAME** به صورت الفبایی مرتب است. بنابراین جستجوی خطی می‌تواند پس از مقایسهٔ یک نام معین **XXX** با نام **YYY** متوقف می‌شود طوری که **XXX<YYY** یعنی به‌گونه‌ای که از نظر الفبایی **XXX** قبل از **YYY** قرار گرفته باشد. به کمک این الگوریتم **C(Fisher) = 5** ، چون جستجو می‌تواند پس از مقایسهٔ **Fisher** با **Goodman** در **NAME[5]** متوقف شود.

مسأله ۶ـ ۴ : فرض کنید الگوریتم جستجوی دودویی، الگوریتم **4.6** ، روی آرایهٔ **NAME** در شکل ۲۳ـ ۴ به کار بسته شده است تا مکان **Goodman** پیدا شود. **BEG** و **END** انتهایی و **MID** وسط را برای قطعهٔ آزمایشی در هر مرحله از الگوریتم پیدا کنید.

حل : یادآوری می‌کنیم که **MID = INT((BEG+END)/2)** که در آن **INT** مبین مقدار صحیح است.

مرحلهٔ ۱ : در اینجا **BEG = 1[Allen]** و **END = 14[Walton]** و درنتیجه **MID = 7[Irwin]** .

مرحله ۲ : چون **Goodman<Irwin** قرار دهید **END = 6** . از این‌رو **MID = 3[Dickens]** .

مرحله ۳ : چون **Goodman>Dickens** قرار دهید **BEG= 4** .از این‌رو **MID = 5[Goodman]** .

در آرایه مکان **Goodman** ، **LOC = 5** است. ملاحظه می‌کنید که تعداد **C = 3** مقایسه انجام شد.

مسأله ۷ـ ۴ : الگوریتم جستجوی دودویی، الگوریتم **4.6** را به‌گونه‌ای اصلاح کنید تا تبدیل به الگوریتم جستجو و اضافه‌کردن شود.

حل : در چهار مرحلهٔ اول الگوریتم به هیچ تغییری نیاز نیست. تنها وقتی **ITEM** در **DATA** ظاهر نشود الگوریتم کنترل کار را به مرحلهٔ **5** منتقل می‌کند. در چنین حالتی، بسته به این که **ITEM** قبل یا بعد از **DATA[MID]** ، اضافه می‌شود. الگوریتم به صورت زیر است :

Algorithm P4.7: (Binary Search and Insertion) DATA is a sorted array with N elements, and ITEM is a given item of information. This algorithm finds the location LOC of ITEM in DATA or inserts ITEM in its proper place in DATA.
Steps 1 through 4. Same as in Algorithm 4.6.
5. If ITEM < DATA[MID], then:
   Set LOC := MID.
   Else:
   Set LOC := MID + 1.
   [End of If structure.]
6. Insert ITEM into DATA[LOC] using Algorithm 4.2.
7. Exit.

مسأله ۸ـ ۴ : فرض کنید **A** یک آرایهٔ مرتب‌شدهٔ **200** عنصری باشد، علاوه بر این فرض کنید که عنصر معلوم **x** با احتمال یکسان در هر مکانی از **A** وجود دارد. زمان اجرای بدترین حالت $f(n)$ و زمان اجرای حالت میانگین $g(n)$ را برای پیداکردن **x** در **A** با استفاده از الگوریتم جستجوی دودویی به دست آورید.

حل : برای هر مقدار **K** فرض کنید n_k نمایشگر تعداد عناصری در **A** باشد که برای تعیین مکان آن به **K** مقایسه نیاز است. آنگاه :

k:	1	2	3	4	5	6	7	8
n_k:	1	2	4	8	16	32	64	73

73 از این واقعیت ناشی می‌شود که $1 + 2 + 4 + \ldots + 64 = 127$ بنابراین تنها $200 - 127 = 73$ باقی می‌ماند. زمان اجرای بدترین حالت $f(n) = 8$ است. زمان اجرای حالت میانگین به صورت زیر محاسبه می‌شود :

$$g(n) = \frac{1}{n}\sum_{k=1}^{8} k \cdot n_k$$
$$= \frac{1\cdot 1 + 2\cdot 2 + 3\cdot 4 + 4\cdot 8 + 5\cdot 16 + 6\cdot 32 + 7\cdot 64 + 8\cdot 73}{200}$$
$$= \frac{1353}{200} = 6.765$$

برای جستجوی دودویی ملاحظه می‌کنید که زمانهای اجرای حالت میانگین و بدترین حالت تقریباً برابر است.

مسأله ۹ـ ۴ : با استفاده از الگوریتم مرتب‌کردن حبابی، الگوریتم **4.4** ، **C** تعداد مقایسه‌ها و **D** تعداد جابجایی‌ها را پیدا کنید که $n = 6$ حرف کلمهٔ **PEOPLE** را به صورت حرفی مرتب می‌کند.

حل : جفت حرفهایی که در هر یک از $n - 1 = 5$ گذر به ترتیب با هم مقایسه می‌شوند به صورت زیر است: جفت حرفهای مورد مقایسه و جابجا شونده با مربع مشخص شده‌اند و جفت حرفهایی که با هم مقایسه ولی جابجا نمی‌شوند با دایره مشخص شده است.

Pass 1. [P E] O P L E, E [P O] P L E, E O (P P) L E
E O P [P L] E, E O P L [P E] E O P L E P
Pass 2. (E O) P L E P, E (O P) L E P, E O [P L] E P
E O L [P E] P, E O L E P P
Pass 3. (E O) L E P P, E [O L] E P P, E L [O E] P P
E L E O P P
Pass 4. (E L) E O P P, E [L E] O P P, E E L O P P
Pass 5. (E E) L O P P, E E L O P P

از آنجاکه $n = 6$، تعداد مقایسه‌ها $C = 5 + 4 + 3 + 2 + 1 = 15$ خواهد بود. تعداد جابجایی‌ها D نیز بستگی به نوع داده و همچنین بستگی به تعداد عنصر n دارد.

مسأله ۱۰ـ ۴ : اتحاد زیر راکه در تجزیه و تحلیل انواع الگوریتم‌های مرتب‌سازی و جستجو مورد استفاده قرار می‌گیرد ثابت کنید.

$$1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2}$$

حل : یک بار عمل جمع را از عدد کوچک به بزرگ و بار دیگر از عدد بزرگ به کوچک می‌نویسیم، به دست می‌آید.

$$S = 1 + 2 + 3 + \ldots + (n-1) + n$$

$$S = n + (n-1) + (n-2) + \ldots + 2 + 1$$

مجموع دو مقدار S جمع بالا به صورت زیر است:

$$2S = (n+1) + (n+1) + (n+1) + \ldots + (n+1) + (n+1)$$

چون تعداد جملات جمع طرف دوم تساوی n است از این‌رو $2S = n(n+1)$ با تقسیم نتیجهٔ اخیر بر 2 جواب مطلوب به دست می‌آید.

آرایه‌های چندبعدی، ماتریسها

مسأله ۱۱ـ ۴ : فرض کنید آرایه‌های چندبعدی A و B با استفاده از نمادهای

A(− 2:2, 2:22) و B(1:8, − 5:5, − 10:5)

تعریف شده‌اند.

(الف) طول هر بعد و تعداد عناصر **A** و **B** را پیدا کنید.

(ب) عنصر **B[3, 3, 3]** در **B** را درنظر بگیرید. اندیسهای مؤثر E_1, E_2, E_3 و آدرس این عنصر را با فرض **Base (B) = 400** و **W = 4** کلمه در خانهٔ حافظه باشد به دست آورید.

حل : (الف) طول هر بعد با فرمول زیر محاسبه می‌شود :

$$\text{Length} = \text{upper bound} - \text{lower bound} + 1$$

از این‌رو طول L_i ابعاد آرایهٔ **A** عبارت است از

$$L_1 = 2 - (-2) + 1 = 5 \quad \text{و} \quad L_2 = 22 - 2 + 1 = 21$$

بنابراین **A** دارای **105 = 5.21** عنصر است. طول L_i ابعاد **B** عبارت است از

$$L_1 = 8 - 1 + 1 = 8 \qquad L_2 = 5 - (-5) + 1 = 11 \qquad L_3 = 5 - (-10) + 1 = 16$$

بنابراین **B** دارای **1408 = 8.11.16** عنصر است.

(ب) اندیس مؤثر **E**، از $E_i = K_i - LB$ بدست می‌آید که در آن K_i اندیس معلوم و **LB** کران پائین است. درنتیجه

$$E_1 = 3 - 1 = 2 \qquad E_2 = 3 - (-5) = 8 \qquad E_3 = 3 - (-10) = 13$$

آدرس بستگی به آن دارد که آیا زبان برنامه‌نویسی، آرایهٔ **B** را به روش سطری ذخیره می‌کند یا روش ستونی. فرض می‌کنیم آرایهٔ **B** به روش ستونی ذخیره شده است. با استفاده از فرمول (۸ـ ۴) داریم :

$$E_3L_2 = 13 \cdot 11 = 143 \qquad E_3L_2 + E_2 = 143 + 8 = 151$$
$$(E_3L_2 + E_2)L_1 = 151 \cdot 8 = 1208 \qquad (E_3L_2 + E_2)L_1 + E_1 = 1208 + 2 = 1210$$

بنابراین

$$\text{LOC(B[3, 3, 3])} = 400 + 4(1210) = 400 + 4840 = 5240$$

مسأله ۱۲ـ ۴ : فرض کنید **A** یک آرایه ماتریسی مربعی **n×n** باشد. قطعه برنامه‌ای بنویسید که

(الف) NUM تعداد عناصر غیرصفر **A** را پیدا کنید.

(ب) SUM مجموع عناصر بالای قطر یعنی عناصر **A[I, J]** که **I<J**، را پیدا کنید.

(ج) PROD حاصلضرب عناصر روی قطر یعنی $(a_{11}, a_{22}, ..., a_{nn})$ را پیدا کنید.

حل : (الف)

```
1. Set NUM := 0.
2. Repeat for I = 1 to N:
3.       Repeat for J = 1 to N:
             If A[I, J] ≠ 0, then: Set NUM := NUM + 1.
         [End of inner loop.]
   [End of outer loop.]
4. Return.
```

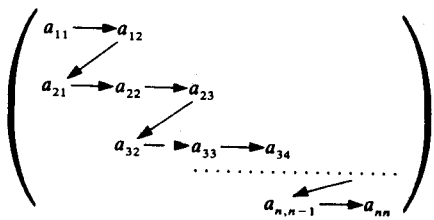
(ب)

```
1. Set SUM := 0.
2. Repeat for J = 2 to N:
3.     Repeat for I = 1 to J − 1:
           Set SUM := SUM + A[I, J].
       [End of inner Step 3 loop.]
4. Return.
```

(ج)

```
1. Set PROD := 1. [This is analogous to setting SUM = 0.]
2. Repeat for K = 1 to N:
       Set PROD := PROD * A[K, K].
   [End of loop.]
3. Return.
```

مسأله ۱۳ـ ۴ : فرض کنید **A** یک آرایهٔ سه قطری **n** مربعی به صورتی باشد که در شکل ۲۴ـ ۴ نشان داده شده است.



شکل ۲۴ـ ۴ آرایهٔ سه‌قطری

توجه دارید که روی قطر **A**، **n** عنصر و در بالا و پائین قطر **n − 1** عنصر می‌باشد. از این‌رو **A** حداکثر **3n − 2** عنصر غیرصفر دارد. فرض کنید خواسته باشیم **A** را در یک آرایهٔ خطی **B** به صورتی که بوسیلهٔ پیکانها در شکل ۲۴ـ ۴ مشخص شده است ذخیره کنیم یعنی فرمولی پیدا کنید که **L** را برحسب **J** و **K** به صورتی به دست دهد که **B[L] = A[J, K]**

یعنی **B[1] = a_{11}, B[2] = a_{12}, B[3] = a_{21}, B[4] = a_{22}, ...**

طوری که بتوان به مقدار **A[J, K]** ازطریق آرایهٔ **B** دسترسی پیدا کرد.

حل : توجه دارید که بالای **A[J, K]** تعداد **3(J − 2) + 2** عنصر و در سمت چپ **A[J, K]** تعداد **K − J + 1** عنصر وجود دارد. از این‌رو

$$L = [3(J - 2) + 2] + [K - J + 1] + 1 = 2J + K - 2$$

مسأله ۱۴ـ ۴ : یک آرایهٔ ماتریسی **n** مربعی **A** را **متقارن** گویند اگر به‌ازای تمام **J** و **K**،

$$A[J, K] = A[K, J]$$

(الف) کدامیک از ماتریسهای زیر متقارن هستند؟

$$\begin{pmatrix} 2 & -3 & 5 \\ -3 & -2 & 4 \\ 5 & 6 & 8 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 3 & -7 \\ 3 & 6 & -1 \\ -7 & -1 & 2 \end{pmatrix}$$

(ب) یک راه مؤثر و کارا برای ذخیرهٔ ماتریس متقارن **A** در حافظه بیان کنید.

(ج) فرض کنید **A** و **B** دو ماتریس متقارن **n** مربعی هستند، یک راه مؤثر و کارا برای ذخیرهٔ **A** و **B** در حافظه بیان کنید.

حل : ماتریس اول متقارن نیست چون $a_{23} = 4$ ولی $a_{32} = 6$. ماتریس دوم یک ماتریس مربعی نیست از این‌رو بنابه تعریف نمی‌تواند متقارن باشد. ماتریس سوم متقارن است.

(ب) چون **A[J, K] = A[K, J]**، تنها ذخیرهٔ آن دسته از عناصر **A** موردنیاز است که بر روی یا پائین قطر قرار دارند. این کار را می‌توان با همان روشی که برای ماتریسهای مثلثی در مثال ۲۵ـ ۴ بیان شده است انجام داد.

(ج) نخست توجه دارید که برای یک ماتریس متقارن تنها لازم است عناصر روی قطر و پائین قطر یا عناصر روی قطر یا بالای قطر را ذخیره کنیم. بنابر این **A** و **B** را می‌توان در یک آرایهٔ **n×(n+1)** بنام **C** به صورتی که در شکل ۲۵ـ ۴ نشان داده شده است ذخیره کرد که در آن **C[J, K] = A[J, K]** وقتی **J≥K** اما **C[J, K] = B[J, K − 1]** وقتی **J<K**.

$$\begin{pmatrix} a_{11} & b_{11} & b_{12} & b_{13} & \cdots & b_{1,n-1} & b_{1n} \\ a_{21} & a_{22} & b_{22} & b_{23} & \cdots & b_{2,n-1} & b_{2n} \\ a_{31} & a_{32} & a_{33} & b_{33} & \cdots & b_{3,n-1} & b_{3n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \cdots & a_{nn} & b_{nn} \end{pmatrix}$$

شکل ۲۵ـ ۴

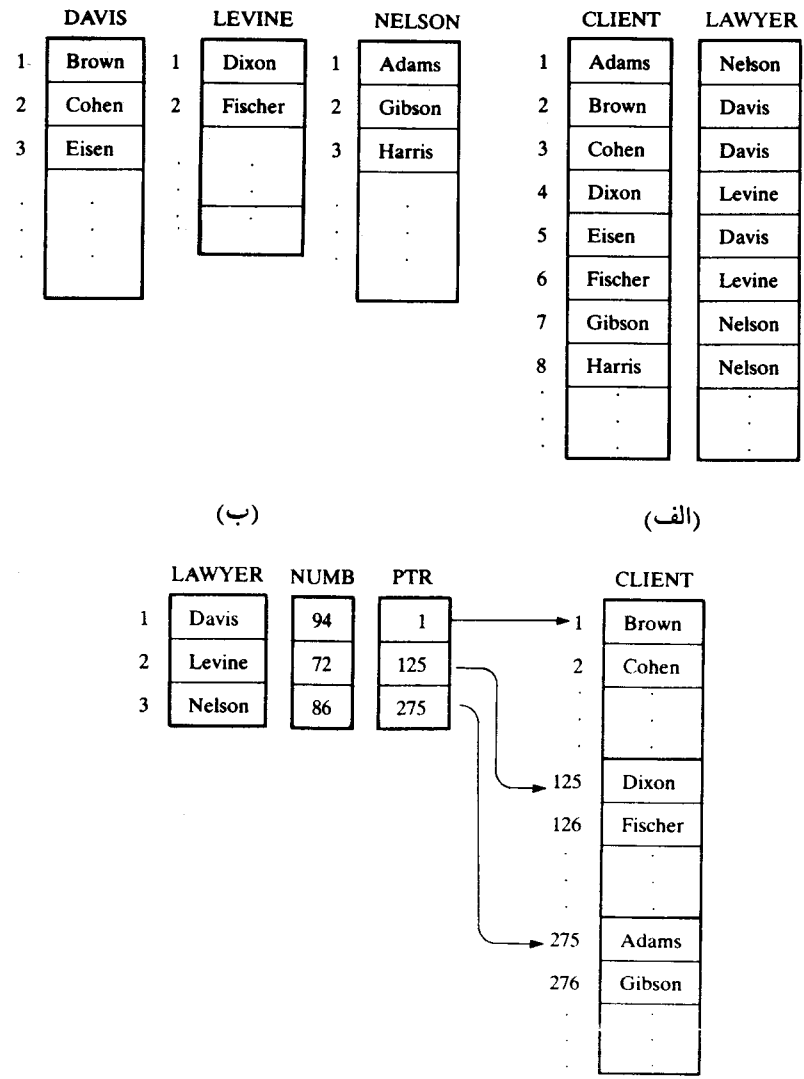
آرایه‌های نوع اشاره‌گر، ساختارهای رکوردی

مسأله ۱۵ـ ۴ : سه وکیل **Levine, Davis** و **Nelson** در یک دفتر کار می‌کنند. هر وکیل موکّل **Client** خاص خود را دارد. شکل ۲۶ـ ۴ سه راه سازماندهی داده‌ها را نشان می‌دهد.

(الف) در اینجا آرایهٔ مرتب‌شدهٔ الفبایی **CLIENT** و آرایهٔ **LAWYER** به‌گونه‌ای وجود دارد که **LAWYER[K]** وکیل مربوط به **CLIENT[K]** است.

(ب) در اینجا سه آرایهٔ مجزا **LEVINE, DAVIS** و **NELSON** وجود دارد که هر آرایه شامل لیست موکلین است.

(ج) در اینجا یک آرایهٔ **LAWYER** وجود دارد و آرایه‌های **NUMB** و **PTR** به ترتیب شماره و مکان هر لیست الفبایی موکلین وکیل را در یک آرایه **CLIENT** به دست می‌دهد. برای این منظور چه ساختمان داده‌ای مفیدتر است؟ چرا؟



شکل ۲۶ـ۴

حل : بهترین ساختمان داده بستگی به چگونگی سازماندهی دفتر کار و چگونگی رسیدگی به کار موکلین دارد.

فرض کنید تنها یک منشی و یک شماره تلفن وجو د دارد و در هر ماه تنها یک لایحه برای هر یک از موکلین تنظیم می‌گردد. علاوه براین فرض کنید تعداد موکلین غالباً از یک وکیل تا وکیل دیگر در تغییر است. آنگاه شکل ۲۶ـ ۴ (الف) احتمالاً مفیدترین ساختمان داده خواهد بود.

فرض کنید وکلا کاملاً مستقل عمل می‌کنند: هر وکیل یک منشی و یک شماره تلفن مخصوص به خود دارد و سیاههٔ موکلین (دفتر ثبت مشخصات موکلین نزد منشی) مختلف است. آنگاه شکل ۲۶ـ ۴ (ب) احتمالاً مفیدترین ساختمان داده خواهد بود.

فرض کنید که دارالوکاله، کلیهٔ امور موکلین را به‌طور متناوب رسیدگی کرده و هر یک از وکلا موظف است که در این برههٔ زمانی به پروندهٔ موکلین خود رسیدگی کند. آنگاه شکل ۲۶ـ ۴ (ج) احتمالاً مفیدترین ساختمان داده خواهد بود.

مسأله ۱۶ـ ۴ : در زیر لیست فیلدهای یک رکورد دانشجویی با شماره سطح آنها ارائه شده است :

1 Student 2 Number 2 Name 3 Last 3 First 3 MI (Middle Initial) 2 Sex 2 Birthday 3 Day 3 Month 3 Year 2 SAT 3 Math 3 Verbal

(الف) ساختار سلسله مراتبی متناظر با آن را رسم کنید.

(ب) کدامیک از فیلدها، فیلدهای ابتدایی هستند.

حل : (الف) هر چند فیلدها به صورت خطی ارائه شده‌اند با این وجود شماره سطح‌ها رابطهٔ سلسله مراتبی بین فیلدها را نشان می‌دهند. ساختار سلسله مراتبی متناظر با آن به صورت زیر است :

1 Student
  2 Number
  2 Name
    3 Last
    3 First
    3 MI
  2 Sex
  2 Birthday
    3 Day
    3 Month
    3 Year
  2 SAT
    3 Math
    3 Verbal

(ب) فیلدهای ابتدایی به آن دسته از فیلدها، گفته می‌شود که زیرفیلد ندارند :
Number, last, First, MI, Sex, Day, Month, Year, Math و **Verbal**. ملاحظه می‌کنید که یک فیلد ابتدایی است فقط اگر بعد از آن فیلد با شماره سطح بالاتر وجود نداشته باشد.

مسأله ۱۷ـ ۴ : یک استاد دانشگاه مشخصات داده‌ای هر دانشجو در یک کلاس **20** نفره را به صورت زیر نگه می‌دارد.

Name(Last, First, MI), Three Tests, Final, Grade

در اینجا **Grade** یک درایهٔ **2**کاراکتری مثلاً **B+** یا **C** یا **A** – است. یک ساختار در زبان **PL / I** برای ذخیرهٔ داده‌ها بیان کنید.

حل : یک عنصر در یک ساختار رکوردی، خود می‌تواند یک آرایه باشد. به جای ذخیرهٔ جداگانهٔ سه آزمون، آنها را در یک آرایه ذخیره می‌کنیم. چنین ساختاری به صورت زیر است:

```
DECLARE    1  STUDENT(20),
              2  NAME,
                    3  LAST   CHARACTER(10),
                    3  FIRST  CHARACTER(10),
                    3  MI     CHARACTER(1),
              2  TEST(3)  FIXED,
              2  FINAL    FIXED,
              2  GRADE    CHARACTER(2);
```

مسأله ۱۸ـ ۴ : یک دانشکده از ساختار زیر برای یک کلاس فارغ‌التحصیل استفاده می‌کنند :

```
1  Student(200)
     2  Name
          3  Last
          3  First
          3  Middle Initial
     2  Major
     2  SAT
          3  Verbal
          3  Math
     2  GPA(4)
     2  CUM
```

در اینجا **GPA[K]** معرّف معدل نمرات سال **K** ام و **CUM** معرف معدل کل است.

(الف) در فایل چند فیلد ابتدایی وجود دارد؟

(ب) چگونه می‌توان به **(i)** رشتهٔ **major** دانشجوی هجدهم و **(ii)** معدل نمرات **GPA** ی چهل و پنجمین دانشجوی سال دوم دسترسی پیدا کرد.

(ج) خروجی زیر را پیدا کنید.

(i) Write: Name[15]
(ii) Write: CUM
(iii) Write: GPA[2].
(iv) Write: GPA[1, 3].

حل : (الف) چون **GPA** برای هر دانشجو **4** بار محاسبه می‌شود، برای هر دانشجو ۱۱ فیلد مقدماتی وجود دارد از این‌رو مجموعاً **2200** فیلد مقدماتی وجود دارد.

(ب) (i) **Student. Major[8]** یا به اختصار **MAJOR[8]** . **(ii)** **GPA[45, 2]**

(ج) **(i)** در اینجا **Name[15]** معرف نام دانشجوی پانزدهم است. اما **NAME** یک فیلد مرکب یا چند قسمتی است. از این‌رو **LAST[15]** ، **First[15]** و **MI[15]** چاپ می‌شوند.

(ii) در اینجا **CUM** معرف تمام مقادیر **CUM** است. یعنی

CUM[1], CUM[2], CUM[3], ... , CUM[200]

چاپ می‌شوند.

(iii) **GPA[2]** معرف آرایهٔ **GPA** دومین دانشجو است. از این‌رو

GPA[2, 1], GPA[2, 2], GPA[2, 3], GPA[2, 4]

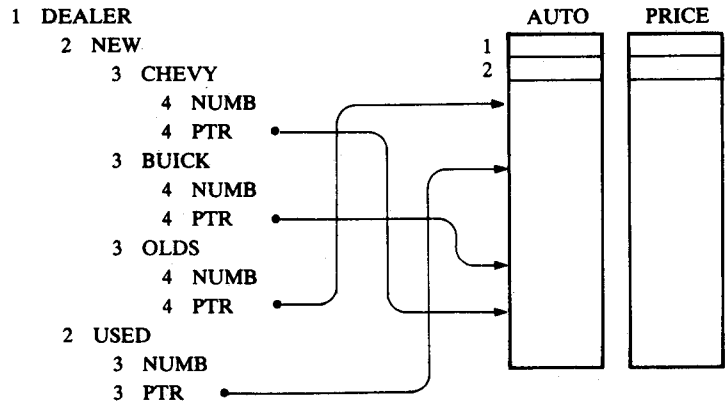
چاپ می‌شوند.

(iv) **GPA[1, 3]** یک فیلد یعنی **GPA** سال سوم اولین دانشجو است. یعنی تنها **GPA[1, 3]** چاپ می‌شود.

مسأله ۱۹ـ ۴ : یک بنگاه فروش اتومبیل شماره سریال و قیمت هر یک از اتومبیل‌ها را به ترتیب در آرایه‌های **AUTO** و **PRICE** نگهداری می‌کند. علاوه بر این از ساختمان دادهٔ شکل ۲۷ـ ۴ نیز استفاده می‌کند که ترکیبی از یک ساختار رکوردی با متغیرهای اشاره‌گر است. **Chevy** نو، **Buick** نو و **Oldsmobile** نو، ماشینهای دست دوم **used car** با هم در **AUTO** لیست می‌شوند. متغیرهای **NUMB** و **PTR** تحت **USED** به ترتیب تعداد و مکان لیست اتومبیلهای دسته دوم را به دست می‌دهد.

(الف) مکان لیست **Buick** نو در **AUTO** را چگونه مشخص می‌کنید؟

(ب) یک زیربرنامهٔ **procedure** برای چاپ شماره‌های سریال تمام **Buick** های نو که کمتر از **$10000** قیمت دارند بنویسید.



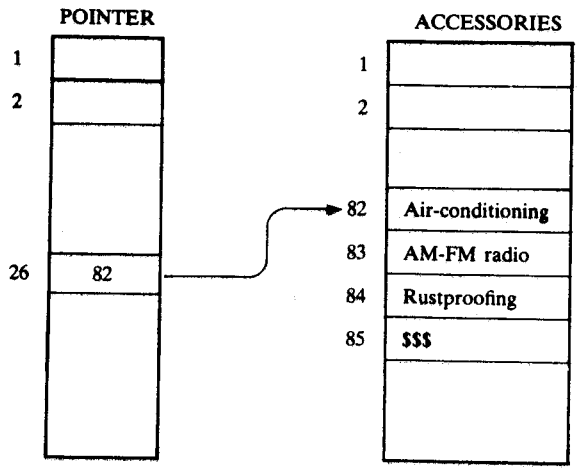
شکل ۲۷ـ ۴

حل : (الف) چون **PTR** بیش از یکبار در ساختار رکوردی ظاهر می‌شود از این‌رو باید از **BUICK.PTR** برای مراجعه به مکان لیست **Buick** نو در **AUTO** استفاده کنید.

(ب) باید لیست را **traverse** پیمایش کنید اما تنها آن دسته از **Buick** ها را چاپ کنید که قیمت آن کمتر از **$10000** است. زیربرنامهٔ **Procedure** به صورت زیر است:

Procedure P4.19: The data are stored in the structure in Fig. 4-27. This procedure outputs those new Buicks whose price is less than $10 000.

```
1. Set FIRST := BUICK.PTR. [Location of first element in Buick list.]
2. Set LAST := FIRST + BUICK.NUMB − 1. [Location of last element in
   list.]
3. Repeat for K = FIRST to LAST.
       If PRICE[K] < 10 000, then:
           Write: AUTO[K], PRICE[K].
       [End of If structure.]
   [End of loop.]
4. Exit.
```



شکل ۲۸ـ ۴

مسأله ۲۰ـ ۴ : در مسأله ۱۹ـ ۴ فرض کنید بنگاه اتومبیل همچنین می‌خواهد لوازم همراه هر اتومبیل ازقبیل کولر، رادیو و کپسول ضدآتش را نگهداری کند. از آنجا که رکورد مشخصات همراه اتومبیل طول متغیر دارد. این کار چگونه انجام می‌شود؟

حل : این کار را می‌توان همانند شکل ۲۸ـ ۴ انجام داد. یعنی علاوه بر **AUTO** و **PRICE** یک آرایهٔ **POINTER** وجود دارد به‌گونه‌ای که **POINTER[K]** مکان لیست لوازم همراه در آرایه **ACCESSORIES** را (با نگهبان **$$$**) در **AUTO[K]** به دست می‌دهد.

مسأله‌های تکمیلی

آرایه‌ها

مسأله ۲۱ـ ۴ : آرایه‌های خطی **XXX(– 10:10), YYY(1935:1985), ZZZ(35)** را درنظر بگیرید. (الف) تعداد عناصر هر آرایه را پیدا کنید. (ب) فرض کنید **Base (YYY) = 400** و **W = 4** کلمه در خانهٔ حافظه برای **YYY** باشد. آدرس **YYY[1942]** ، **YYY[1977]** و **YYY(1988)** را پیدا کنید.

مسأله ۲۲ـ ۴ : آرایه‌های چندبعدی زیر را درنظر بگیرید :

X(– 5:5, 3:33) **Y(3:10, 1:15, 10:20)**

(الف) طول هر بعد و تعداد عناصر **X** و **Y** را پیدا کنید.

(ب) فرض کنید Base(Y) = 400 و W = 4 کلمه در خانهٔ حافظه وجود داشته باشد. اندیسهای مؤثر E_1, E_2, E_3 و آدرس Y[5, 10, 15] را پیدا کنید. با این فرض که (i) Y با روش سطری و (ii) Y با روش ستونی ذخیره شده باشد.

مسألهٔ ۲۳ـ ۴ : آرایهٔ A شامل 25 عدد صحیح مثبت است. یک قطعه برنامه بنویسید که :

(الف) تمام جفت عناصری را که مجموع آنها برابر 25 است پیدا کند.

(ب) تعداد EVNUM عناصر A را که زوج هستند و تعداد ODNUM عناصر A را که فرد هستند پیدا کند.

مسأله ۲۴ـ ۴ : فرض کنید A یک آرایهٔ خطی با n مقدار عددی باشد. یک زیربرنامهٔ Procedure بنام

MEAN(A, N, AVE)

بنویسید که میانگین AVE مقادیر A را پیدا کند. **میانگین حسابی** یا **معدل** $\bar{x}$ مقادیر $x_1, x_2, ..., x_n$ با توجه به تعریف، به صورت زیر محاسبه می‌شود :

$$\bar{x} = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

مسأله ۲۵ـ ۴ : هر دانشجوی یک کلاس 30 نفره در 6 آزمون شرکت می‌کند که در آن دامنهٔ نمرات بین 0 و 100 است. فرض کنید نمرات آزمونها در یک آرایهٔ 30×6 بنام TEST ذخیره شده است، یک قطعه برنامه بنویسید که :

(الف) معدل نمرات هر آزمون را پیدا کند.

(ب) نمرهٔ نهایی هر دانشجو را پیدا کنید که در آن نمرهٔ نهایی، میانگین پنج تا از بالاترین نمرهٔ آزمون است.

(ج) تعداد NUM دانشجویانی که نمرهٔ قبولی نگرفته‌اند یعنی دانشجویانی که نمرهٔ نهایی‌شان کمتر از 60 است را پیدا کنید.

(د) میانگین نمرات نهایی را پیدا کنید.

آرایه‌های نوع اشاره‌گر، ساختارهای رکوردی

مسأله ۲۶ـ ۴ : داده‌های شکل ۲۶ـ ۴ (ج) را درنظر بگیرید. یک زیربرنامهٔ Procedure بنویسید که لیست موکلین مربوط به LAWYER[K] را پیدا کند. (ب) فرض کنید CLIENT جا برای 400 موکل دارد. آرایه‌ای بنام FREE تعریف کنید به‌گونه‌ای که FREE[K] شامل تعداد حافظه‌های خالی بعد از لیست موکلین مربوط به LAWYER[K] باشد.

مسأله ۲۷ـ ۴ : در زیرلیست فیلد رکوردهای یک فایل از کارمندان با شمارهٔ سطح آنها ارائه شده

است:

```
1 Employee(200),  2 SSN(Social Security Number),  2 Name,
3 Last,  3 First,  3 MI (Middle Initial),  2 Address,  3 Street,
3 Area,  4 City,  4 State,  4 ZIP,  2 Age,  2 Salary,  2 Dependents
```

(الف) ساختار سلسله مراتبی متناظر با آن را رسم کنید.

(ب) کدامیک از فیلدها، فیلدهای ابتدایی هستند؟

(ج) یک ساختار رکوردی برای مثال یک ساختار **PL / I** یا یک رکورد **PASCAL** برای ذخیره داده‌ها بیان کنید.

مسأله ۲۸ـ ۴ : ساختمان دادهٔ شکل ۲۷ـ ۴ را درنظر بگیرید. یک زیربرنامهٔ **Procedure** برای انجام هر یک از عملیات زیر بنویسید :

(الف) تعداد **Oldsmbiles** نو راکه با قیمت کمتر از **000 10$** به فروش می‌رسند پیداکند.

(ب) تعداد **automobiles** نو راکه با قیمت کمتر از **000 10$** به فروش می‌رسند پیداکند.

(ج) تعداد **automobiles** راکه با قیمت کمتر از **000 10$** به فروش می‌رسند پیداکند.

(د) تمام **automobiles** راکه زیر **000 10$** به فروش می‌رسند پیداکند.

توجه کنید : قسمتهای (ج) و (د) تنها نیازمند **AUTO** و **PRICE** همراه با شماره **automobiles** هستند.

مسأله ۲۹ـ ۴ : رکورد دانشجویان یک کلاس به صورت زیر سازماندهی شده است :

```
1 Student(35),  2 Name,  3 Last,  3 First,  3 MI (Middle Initial),  2 Major
2 Test(4),  2 Final,  2 Grade
```

(الف) چند فیلد ابتدایی در آن وجود دارد؟

(ب) یک ساختار رکوردی برای مثال یک ساختار **PL/ I** یا یک رکورد **PASCAL** برای ذخیره داده‌ها بیان کنید.

(ج) خروجی هر یک از دستورهای **Write** زیر را شرح دهید.

. Write: Test[4] (iii) Write:Name[15] (ii) Write:Final[15] (i)

مسأله ۳۰ـ ۴ : ساختمان دادهٔ مسأله ۱۸ـ ۴ را درنظر بگیرید. یک زیربرنامهٔ **Procedure** بنویسید که :

(الف) معدل نمرات **GPA** دانشجوی سال دوم را پیداکند.

(ب) تعداد دانشجویان رشتهٔ زیست‌شناسی را پیداکند.

(ج) تعداد نمرات **CUM** بزرگتر از **K** را پیداکند.

برای مسأله‌های زیربرنامه بنویسید

آرایه‌ها

فرض کنید داده‌های جدول ۱ـ ۴ در آرایه‌های خطی **SSN, LAST, GIVEN, CUM** و **YEAR** (با فضایی برای 25 دانشجو) ذخیره شده‌اند و متغیر **NUM** به صورتی تعریف می‌شود که حاوی تعداد واقعی دانشجویان است.

مسأله ۳۱ـ ۴ : برای هر یک از موارد زیر یک برنامه بنویسید :

(الف) لیست تمام دانشجویانی را که **CUM** آنها برابر **K** یا بیشتر از **K** است چاپ کند. برنامه را با استفاده از **K = 3.00** آزمایش کنید.

(ب) لیست تمام دانشجویانی را که سال **L** ام هستند چاپ کند. برنامه را با استفاده از **L =2** یا سال اول آزمایش کنید.

مسأله ۳۲ـ ۴ : الگوریتم جستجوی خطی را به صورت یک زیربرنامهٔ **LINEAR(ARRAY, LB, UB, ITEM, LOC)** بنویسید که یا مکان **LOC** را پیدا کند که در آن **ITEM** در **ARRAY** ظاهر می‌شود یا **LOC = 0** را برگرداند.

مسأله ۳۳ـ ۴ : الگوریتم جستجوی دودویی و اضافه‌کردن را به صورت یک زیربرنامهٔ **BINARY(ARRAY, LB, UB, ITEM, LOC)** بنویسید که یا مکان **LOC** را پیدا کند که در آن **ITEM** در **ARRAY** ظاهر می‌شود یا مکان **LOC** را پیدا کند که **ITEM** باید به **ARRAY** اضافه شود.

مسأله ۳۴ـ ۴ : یک برنامه بنویسید که شمارهٔ تأمین اجتماعی **SOC** یک دانشجو را بخواند و با استفاده از **LINEAR** رکورد دانشجو را پیدا کرده چاپ کند. برنامه را با استفاده از (الف) **0732 – 58 – 174** (ب) **5554 – 55 – 172** و (ج) **6382 – 63 – 126** آزمایش کنید.

مسأله ۳۵ـ ۴ : یک برنامه بنویسید که **NAME** (آخر) یک دانشجو را بخواند و با استفاده از **BINARY** رکورد دانشجو را پیدا کرده چاپ کند. برنامه را با استفاده از (الف) **Rogers** (ب) **Johnson** و (ج) **Bailey** آزمایش کنید.

مسأله ۳۶ـ ۴ : یک برنامه بنویسید که رکورد یک دانشجو را به صورت زیر بخواند :

SSNST, LASTST, GVNST, YEARST

و با استفاده از **BINARY** رکورد را در لیست اضافه کند. برنامه را با استفاده از:

(الف) **168-48-2255, Quinn, Michael, 2.15, 3**

(ب) **177-58-0772, Jones, Amy, 2.75, 2**

آزمایش کنید.

جدول ۱ـ ۴

Social Security Number	Last Name	Given Name	CUM	Year
211-58-1329	Adams	Bruce	2.55	2
169-38-4248	Bailey	Irene L.	3.25	4
166-48-5842	Cheng	Kim	3.40	1
187-52-4076	Davis	John C.	2.85	2
126-63-6382	Edwards	Steven	1.75	3
135-58-9565	Fox	Kenneth	2.80	2
172-48-1849	Green	Gerald S.	2.35	2
192-60-3157	Hopkins	Gary	2.70	2
160-60-1826	Klein	Deborah M.	3.05	1
166-52-4147	Lee	John	2.60	3
186-58-0480	Murphy	William	2.30	2
187-58-1123	Newman	Ronald P.	3.90	4
174-58-0732	Osborn	Paul	2.05	3
183-52-3865	Parker	David	1.55	2
135-48-1397	Rogers	Mary J.	1.85	1
182-52-6712	Schwab	Joanna	2.95	2
184-48-8539	Thompson	David E.	3.15	3
187-48-2377	White	Adam	2.50	2

مسأله ۳۷ـ ۴ : یک برنامه بنویسید که **NAME** (آخر) یک دانشجو را بخواند و با استفاده از **BINARY** رکورد دانشجو را از لیست حذف کند. برنامه را با استفاده از (الف) **Parker** و (ب) **Fox** آزمایش کنید.

مسأله ۳۸ـ ۴ : برای هر یک از حالتهای زیر یک برنامه بنویسید:

(الف) با استفاده از آرایهٔ **SSN** آرایه‌های **NUMBER** و **PTR** را به‌گونه‌ای تعریف کنید که **NUMBER** آرایهٔ مرتب‌شده‌ای از عناصر **SSN** باشد و **PTR[K]** شامل مکان **NUMBER[K]** در **SSN** باشد.

(ب) شمارهٔ تامین اجتماعی **SOC** یک دانشجو را بخواند و با استفاده از **BINARY** و آرایهٔ **NUMBER** رکورد دانشجو را پیدا کرده چاپ کند. برنامه را با استفاده از **(i) 0732 – 58 –174 (ii) 5554 - 55 - 172** و **(iii) 6382 – 63 – 126** آزمایش کنید. این مسأله را با مسأله ۳۴ـ ۴ مقایسه کنید.

آرایه‌های نوع اشاره‌گر

فرض کنید داده‌های داخل جدول ۲ـ۴ در یک آرایهٔ خطی **CLASS** (با فضایی برای **50** نام) ذخیره شده‌اند. علاوه براین فرض کنید بین بخش **Section** ها، **2** فضای خالی وجود دارد و آرایهٔ خطی **NUMB, PTR** و **FREE** طوری تعریف شده‌اند که **NUMB[K]** حاوی تعداد عناصر در بخش **K** است و

K, PTR[K] مکان اولین نام بخش **K** در **CLASS** را بدست می‌دهد و **FREE[K]** تعداد فضاهای خالی داخل **CLASS** بعد از بخش **K** است.

جدول ۲ـ ۴

Section 1	Section 2	Section 3	Section 4
Brown	Abrams	Allen	Burns
Davis	Collins	Conroy	Cohen
Jones	Forman	Damario	Evans
Samuels	Hughes	Harris	Gilbert
	Klein	Rich	Harlan
	Lee	Sweeney	Lopez
	Moore		Meth
	Quinn		Ryan
	Rosen		Williams
	Scott		
	Taylor		
	Weaver		

مسأله ۳۹ـ ۴ : یک برنامه بنویسید که یک عدد صحیح **K** را بخواند و نامهای بخش **K** را چاپ کند. برنامه را با استفاده از (الف) **K = 2** و (ب) **K = 3** آزمایش کنید.

مسأله ۴۰ـ ۴ : یک برنامه بنویسید که نام **NAME** یک دانشجو را بخواند و مکان و شماره بخش **Section** دانشجو را پیدا کرده چاپ کند. برنامه را با استفاده از (الف) **Harris** (ب) **Rivers** و (ج) **Lopez** آزمایش کنید.

مسأله ۴۱ـ ۴ : یک برنامه بنویسید که نام‌ها را به صورت ستونی، به‌صورتی که در جدول ۲ ـ ۴ داده شده است چاپ کند.

مسأله ۴۲ـ ۴ : یک برنامه بنویسید که نام **NAME** و شمارهٔ بخش **SECN** یک دانشجو را بخواند و دانشجو را در **CLASS** اضافه کند. برنامه را با استفاده از (الف) **Eden, 3;** (ب) **Novak, 4;** (ج) **Parker, 2;** (د) **Vaughn, 3** و (ه) **Bennett, 3** آزمایش کند. برنامه باید حالتهای سرریز **OverFlow** را مورد توجه قرار دهد.

مسأله ۴۳ـ ۴ : یک برنامه بنویسید که نام **NAME** یک دانشجو را بخواند و دانشجو را از **CLASS** حذف کند. برنامه را با استفاده از (الف) **Klein** (ب) **Daniels** (ج) **Meth** و (د) **Harris** آزمایش کنید.

مسأله‌های گوناگون

مسأله ۴۴ـ ۴ : فرض کنید **A** و **B** دو آرایهٔ برداری **n** عنصری در حافظه هستند و **X** و **Y** اسکالر هستند. یک برنامه بنویسید که (الف) **XA + YB** و (ب) **A.B** را پیدا کند. برنامه را با استفاده از **A = (16,– 6 , 7)** و **B = (4, 2, - 3)** و **X = 2** و **Y = – 5** آزمایش کنید.

مسأله ۴۵ـ ۴ : الگوریتم ضرب دو ماتریس، یعنی الگوریتم **4.7** را به صورت یک زیربرنامهٔ

MATMUL(A, B, C, M, P, N)

بنویسید که **C** حاصلضرب ماتریس **A** ، **m×p** و ماتریس **B** ، **P × n** را پیدا کند. برنامه را با استفاده از آزمایش کنید.

$$A = \begin{pmatrix} 4 & -3 & 5 \\ 6 & 1 & -2 \end{pmatrix} \qquad B = \begin{pmatrix} 2 & 3 & -7 & -3 \\ 5 & -1 & 6 & 2 \\ 0 & 3 & -2 & 1 \end{pmatrix}$$

مسأله ۴۶ـ ۴ : چند جمله‌ای

$$f(x) = a_1x^n + a_2x^{n-1} + \cdots + a_nx + a_{n+1}$$

را درنظر بگیرید. محاسبهٔ مقدار این چندجمله‌ای با روش معمولی مستلزم

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

عمل ضرب و **n** عمل جمع است. با وجود این، می‌توان این چندجمله‌ای را با فاکتورگیری متوالی از **x** به صورت زیر نوشت :

$$f(x) = ((\cdots((a_1x + a_2)x + a_3)x + \cdots)x + a_n)x + a_{n+1}$$

در این روش تنها از **n** عمل ضرب و **n** عمل جمع استفاده می‌شود. روش دوم اخیر برای محاسبهٔ مقدار یک چند جمله‌ای روش **هورنر** نام دارد.

(الف) چندجمله‌ای $f(x) = 5x^4 - 6x^3 + 7x^2 + 8x - 9$ را به‌گونه‌ای بنویسید تا با استفاده از روش هورنر قابل ارزیابی باشد.

(ب) فرض کنید ضریبهای یک چندجمله‌ای در یک آرایهٔ خطی **A(N + 1)** در حافظه هستند، یعنی **A[1]** ضرب x^n ، **A[2]** ضریب x^{n-1} و ... , $x^n - 1$, و **A[N + 1]** مقدار ثابت چندجمله‌ای است. یک زیربرنامهٔ **Procedure** بنام **HORNER(A, N + 1, X, Y)** بنویسید که مقدار چند جمله‌ای **Y = F(X)** را به‌ازای یک مقدار معلوم **X** با استفاده از روش هورنر محاسبه کند. برنامه را با استفاده از **X = 2** و $f(X)$ از قسمت **(الف)** آزمایش کنید.

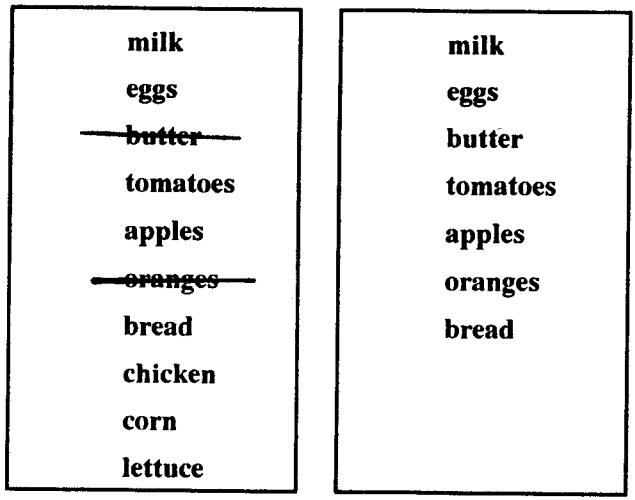
فصل ۵

لیستهای پیوندی

۱ـ۵ مقدمه

استفاده از اصطلاح "لیست" در زندگی روزمره به یک مجموعهٔ خطی از اقلام داده‌ای، مربوط می‌شود. شکل ۱ـ۵ (الف) لیست خرید از یک فروشگاه را نشان می‌دهد. این لیست دارای عنصر اول، عنصر دوم، ...،و عنصر آخر است. اغلب از ما خواسته می‌شود یک عنصر را به لیست اضافه کنیم یا آن را از لیست حذف کنیم. شکل ۱ـ۵ (ب) لیست خرید از فروشگاه را پس از اضافه کردن سه عنصر در آخر لیست و حذف دو عنصر از لیست نشان می‌دهد که روی آنها خط کشیده‌ایم.

داده‌پردازی اغلب شامل ذخیره و پردازش داده‌هایی است که در لیستها سازماندهی می‌شوند. استفاده از آرایه‌ها یک روش ذخیرهٔ چنین داده‌هایی است که در فصل ۴ مورد بحث و بررسی کامل قرار گرفت. یادآوری می‌کنیم که رابطهٔ خطی بین عناصر داده‌ای یک آرایه به‌وسیلهٔ رابطهٔ فیزیکی داده‌ها در حافظه منعکس می‌شود نه به وسیلهٔ اطلاعات خودِ عناصر داده‌ای. از طرف دیگر، آرایه‌ها نیز دارای معایبی هستند به عنوان مثال اضافه کردن و حذف عناصر در آرایه‌ها نسبتاً پرهزینه است. علاوه بر این، از آنجاکه هر آرایه معمولاً یک بلاک از فضای حافظه را اشغال می‌کند از این‌رو هنگام نیاز به حافظهٔ اضافی به راحتی نمی‌توان اندازهٔ یک آرایه را دوبرابر یا سه برابر کرد. به همین خاطر به آرایه‌ها، لیستهای **فشرده** یا **متراکم** نیز می‌گویند. علاوه بر این به آرایه‌ها، ساختمان دادهٔ **ایستا** نیز گفته می‌شود.



(ب) (الف)

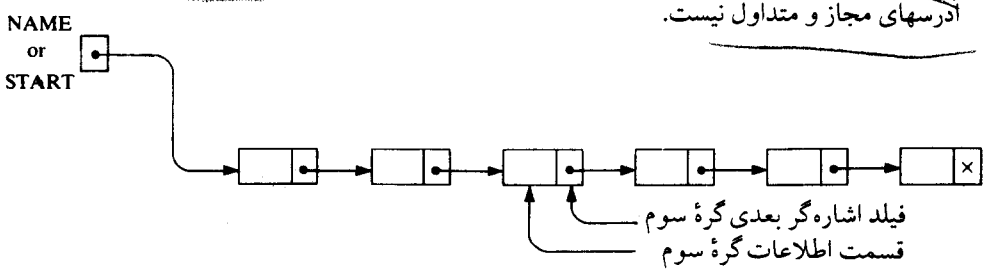
شکل ۱ـ ۵

راه دیگر ذخیرهٔ یک لیست در حافظه آن است که هر عنصر را در یک لیست، که شامل یک فیلد و آدرس عنصر بعدی در لیست است و **پیوند** یا **اشاره‌گر** نامیده می‌شود قرار می‌دهیم. بدین‌ترتیب لازم نیست عناصر متوالی داخل لیست فضای مجاور در حافظه را اشغال کنند. این کار باعث می‌شود اضافه کردن و حذف عناصر لیست براحتی انجام شود. بنابراین، اگر اساساً علاقمند باشیم جستجویی را در داده‌ها برای اضافه و حذف عنصر مثلاً در پردازش کلمه انجام دهیم نباید داده‌ها را در یک آرایه ذخیره کنیم بلکه بهتر است آنها را در یک آرایه به کمک اشاره‌گر ذخیره کنیم. ساختمان دادهٔ نوع اخیر یک **لیست پیوندی** نام دارد که موضوع اصلی مطالب این فصل را تشکیل می‌دهد. علاوه بر این لیستهای چرخشی یا حلقوی و لیستهای **دو طرفه** را که تعمیم طبیعی لیستهای پیوندی است بررسی می‌کنیم مزایا و معایب آنها شرح داده می‌شود.

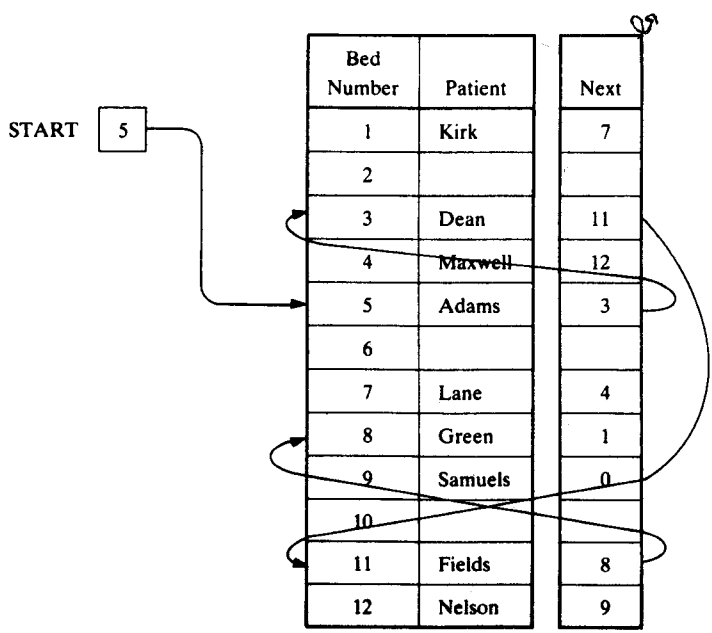
۲ـ۵ لیستهای پیوندی

یک لیست پیوندی یا یک لیست یکطرفه مجموعه‌ای خطی از عناصر داده‌ای بنام **گره‌ها** است که در آن ترتیب خطی توسط **اشاره‌گرها** داده می‌شود. به عبارت دیگر هر گره به دو قسمت تقسیم می‌شود. قسمت اول شامل اطلاعات عنصر است و قسمت دوم که **فیلد پیوند** یا **فیلد اشاره‌گر بعدی** نام دارد شامل آدرس گرهٔ بعدی در لیست است.

شکل ۲ـ ۵ یک نمودار از یک لیست پیوندی با **6** گره را نشان می‌دهد که هر گره با دو قسمت به تصویر کشیده شده است. قسمت چپ، قسمت اطلاعات گره را نشان می‌دهد که می‌تواند حاوی تمام رکورد عنصر داده‌ای مانند (**NAME, ADDRESS,...**) باشد. قسمت راست، فیلد اشاره‌گر بعدی گره را نمایش می‌دهد و یک پیکان از آن تا گرهٔ بعدی لیست رسم شده است. این کار به پیروی از روش متداول رسم پیکان از یک فیلد به یک گره صورت گرفته است هنگامی که آدرس گره در فیلد داده شده قرار دارد. اشاره‌گر آخرین گره شامل یک مقدار خاص است که **اشاره‌گر پوچ** یا **NULL** نام دارد که هیچیک از آدرسهای مجاز و متداول نیست.



شکل ۲ـ ۵ لیست پیوندی با 6 گره



شکل ۳ـ ۵

در عمل 0 یا یک عدد منفی برای اشاره‌گر پوچ یا NULL مورد استفاده قرار می‌گیرد. اشاره‌گر پوچ که در نمودار با X نمایش داده شده است علامت پایان لیست است. علاوه بر این لیست پیوندی دارای یک متغیر اشاره‌گر لیست بنام START یا NAME است که محتوای آن آدرس گرهٔ اول لیست است. از این‌رو یک پیکان از START به گرهٔ اول رسم شده است. واضح است که برای پیمودن طول لیست تنها نیازمند این آدرس در START هستیم. حالت خاصی وجود دارد که لیست، هیچ گره‌ای ندارد. به چنین لیستی لیست پوچ NULL یا لیست تهی یا خالی می‌گویند و با اشاره‌گر NULL در متغیر START نمایش داده می‌شود.

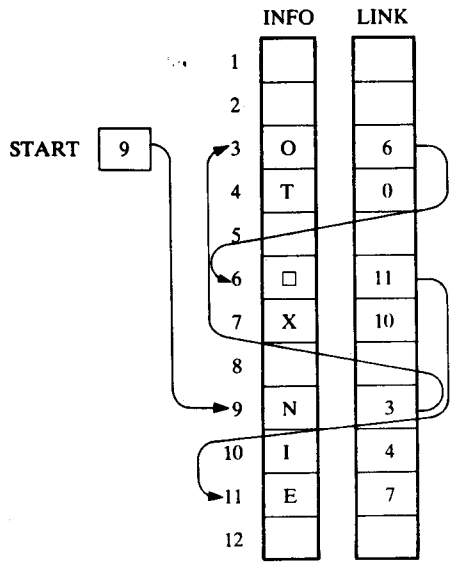
مثال ۱ـ ۵

اطاق یک بیمارستان دارای ۱۲ تخت است که از میان آنها ۹ تخت به صورتی که در شکل ۳ـ ۵ نشان داده شده است اشغال شده است. فرض کنید می‌خواهیم یک لیست الفبایی از بیماران در اختیار داشته باشیم. این لیست ممکن است به وسیلهٔ فیلد اشاره‌گر که در نمودار Next نام‌گذاری شده است داده شود. از متغیر START برای اشاره به بیمار اول استفاده می‌کنیم. از این‌رو START شامل 5 است چون بیمار اول Adams تخت شماره 5 را اشغال کرده است. اشاره‌گر Adams برابر 3 است چون Dean، بیمار بعدی تخت شمارهٔ 3 را اشغال کرده است. اشاره‌گر Dean برابر 11 است چون Fields بیمار بعدی تخت شمارهٔ 11 را اشغال کرده است و الی آخر. ورودی مربوط به بیمار آخر (Samuels) شامل اشاره‌گر پوچ است که آن را با 0 نمایش می‌دهد. در نمودار فقط برای بیان لیست چند بیمار اول پیکان رسم شده است.

۳ـ ۵ نمایش لیستهای پیوندی در حافظه

فرض کنید LIST یک لیست پیوندی باشد. آنگاه LIST به صورت زیر در حافظه ذخیره می‌شود مگر آن که خلاف آن به صورت صریح یا ضمنی بیان گردد. قبل از هرچیز، LIST مستلزم دو آرایهٔ خطی است که ما آنها را در اینجا INFO و LINK می‌نامیم. نظیر INFO[K] و LINK[K] که به ترتیب شامل قسمت اطلاعات و فیلد اشاره‌گر بعدی یک گره از LIST است. همانگونه که در بالا متذکر شدیم LIST نیز نیازمند یک نام متغیر نظیر START است که شامل مکان ابتدای لیست است و نگهبان اشاره‌گر بعد که با NULL نمایش داده می‌شود مبین انتهای لیست است. از آنجا که اندیسهای آرایه‌های INFO و LINK معمولاً مثبت هستند ما NULL = 0 را اختیار می‌کنیم، مگر آن که خلاف آن را بیان کنیم.

مثالهای زیر از لیستهای پیوندی بیانگر آن است که لزومی ندارد گره‌های یک لیست عناصر مجاور در آرایه‌های INFO و LINK را اشغال کنند و بیش از یک لیست می‌تواند در همین آرایه‌های خطی INFO و LINK نگهداری شوند. با وجود این، هر لیست باید متغیر اشاره‌گر مربوط به خود را دارا باشد که مکان گرهٔ اولش را به دست می‌دهد.



شکل ۴ـ ۵

مثال ۲ـ ۵

شکل ۴ـ ۵ تصویر یک لیست پیوندی را در حافظه نشان می‌دهد که در آن هر گرهٔ لیست شامل یک کاراکتر است. می‌توانیم لیست واقعی کاراکترها، یا به بیان دیگر رشته را به صورت زیر به دست آوریم.

START = 9 از این‌رو **INFO[9] = N** اولین کاراکتر است.

LINK[9] = 3 از این‌رو **INFO[3] = O** دومین کاراکتر است.

LINK[3] = 6 از این‌رو **INFO[6] = □** فضای خالی سومین کاراکتر است.

LINK[6] = 11 از این‌رو **INFO[11] = E** چهارمین کاراکتر است.

LINK[11] = 7 از این‌رو **INFO[7] = X** پنجمین کاراکتر است.

LINK[7] = 10 از این‌رو **INFO[10] = I** ششمین کاراکتر است.

LINK[10] = 4 از این‌رو **INFO[4] = T** هفتمین کاراکتر است.

LINK[4] = 0 یعنی مقدار آن NULL است، از این رو لیست پایان یافته است.

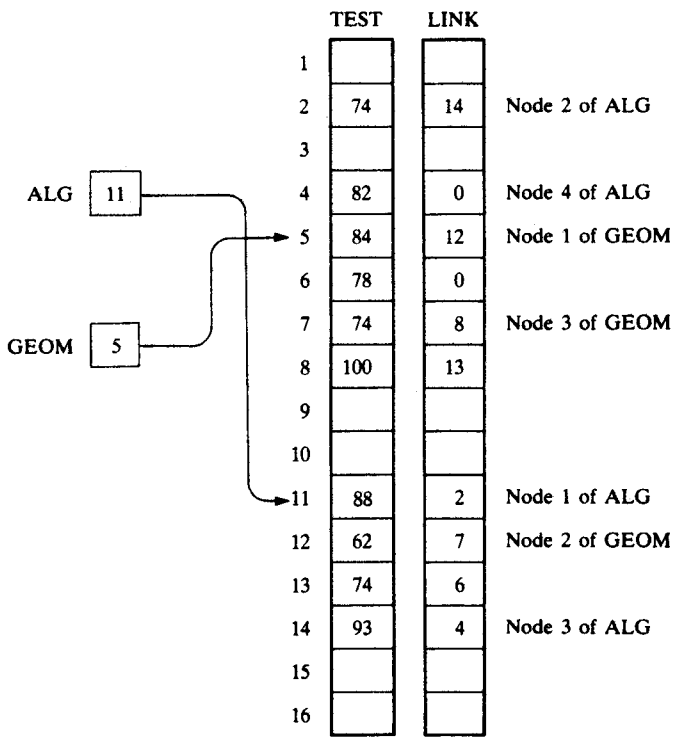
به عبارت دیگر، **NO EXIT** رشتهٔ کاراکتری موردنظر است.

مثال ۳ـ ۵

شکل ۵ـ ۵ چگونگی نگهداری دو لیست از نمرات آزمون، یعنی **ALG** و **GEOM** را در حافظه نشان می‌دهد که در آن گره‌های هر دو لیست در آرایه‌های خطی **TEST** و **LINK** ذخیره می‌شوند. ملاحظه

می‌کنید که از نام این لیستها به عنوان متغیرهای اشاره‌گر لیست نیز استفاده می‌کنیم. در اینجا **ALG حاوی 11 ، مکان اولین گره آن و GEOM حاوی 5 ، مکان اولین گرهٔ آن است. با تعقیب اشاره‌گرها** ملاحظه می‌کنید که **ALG** شامل نمرات آزمون زیر است:

88, 74, 93, 82



شکل ۵ـ ۵

و **GEOM** شامل نمرات ازمون

84, 62, 74, 100, 74, 78

است. گره‌های **ALG** و برخی از گره‌های **GEOM** در نمودار به صورت صریح با شماره مشخص شده‌اند.

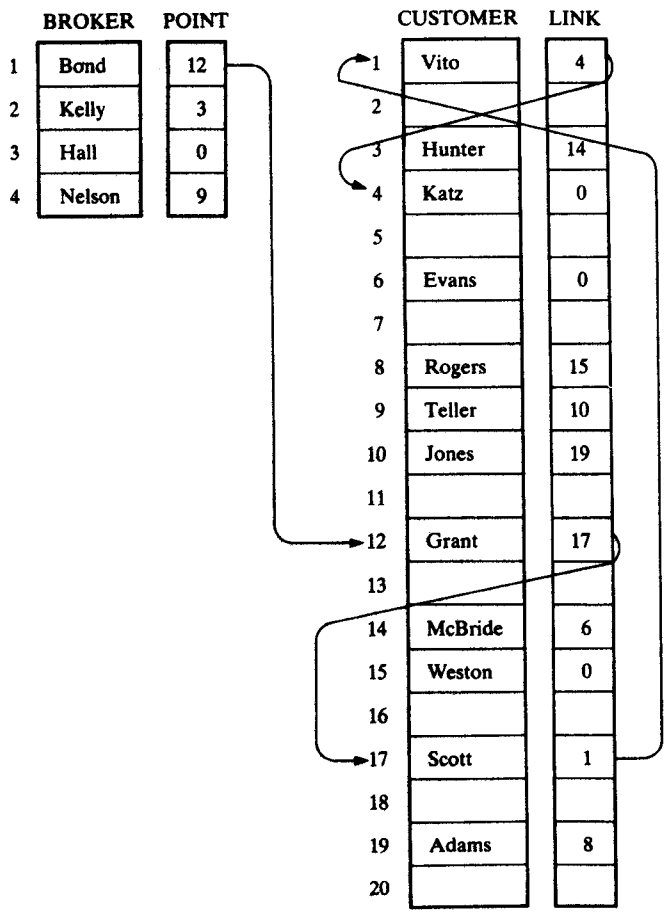
مثال ۴ـ ۵

یک شرکت فروش کامپیوتر را درنظر بگیرید که دارای چهار مأمور فروش است و هر مأمور فروش لیستی از مشتری‌های خود را در اختیار دارد. چنین داده‌ای را می‌توان مانند شکل ۶ـ ۵ سازماندهی کرد،

یعنی تمام مشتری‌های چهار لیست، در یک آرایهٔ **CUSTOMER** قرار داده می‌شوند و آرایهٔ **LINK** شامل فیلدهای اشاره‌گر بعدیِ گره‌های لیست است. همچنین آرایهٔ مأمور فروش **BROKER** وجود دارد که شامل لیست مأموران فروش است و آرایهٔ نوع اشاره‌گر **POINT** به‌گونه‌ای است که **POINT[K]** به ابتدای لیست مشتری‌های **BROKER[K]** اشاره می‌کند.

بنابراین لیست مشتری‌های Bond که با پیکان مشخص شده است شامل

Grant,Scott, Vito, Katz



شکل ۶ـ۵

به‌طور مشابه، لیست **Kelly** شامل

Hunter, McBride, Evans

و لیست **Nelson** شامل

Teller, Jones, Adams, Rogers, Weston

و لیست **Hall** یک لیست خالی است، چون اشاره‌گر **NULL** یا **0** در **POINT[3]** ظاهر شده است.

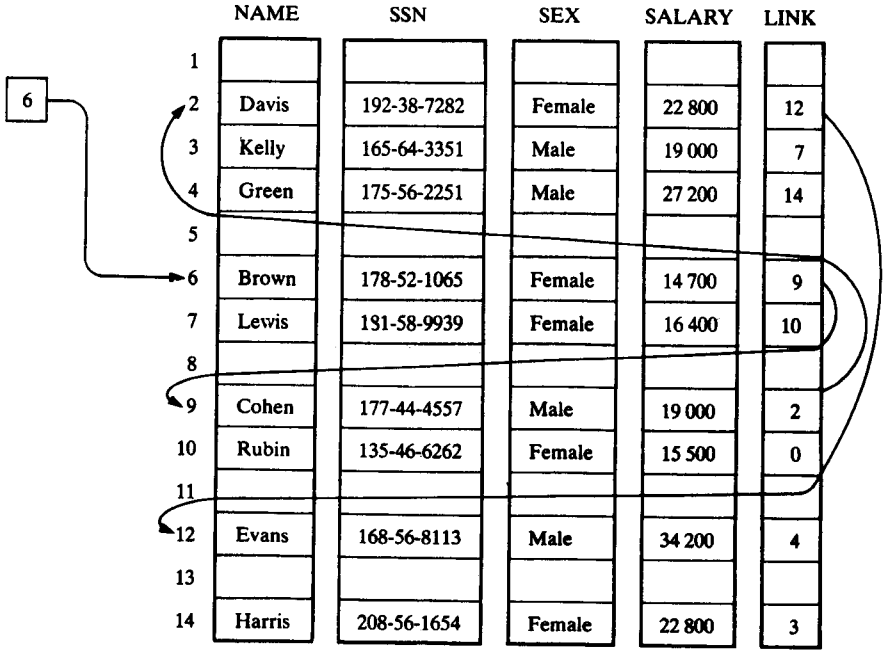
به‌بیان کلی، قسمت اطلاعاتِ یک گره می‌تواند یک رکورد بابیش از یک عنصر داده‌ای باشد. در چنین مواردی، داده‌ها بایستی در نوعی از ساختار رکوردی یا در یک مجموعه از آرایه‌های موازی مانند آنچه که در مثال زیر بیان شده است ذخیره شوند.

مثال ۵ـ ۵

فایل پرسنلی یک شرکت کوچک را درنظر بگیرید که دارای داده‌های زیر برای نُه کارمند است:

Name, Social Security Number, Sex, Monthly Salary

معمولاً چهار آرایهٔ موازی مانند **NAME, SSN, SEX, SALARY** برای ذخیرهٔ داده‌ها به صورتی که در بخش ۱۲ـ ۴ مورد بررسی قرار گرفت موردنیاز است. شکل ۷ـ ۵ چگونگی ذخیرهٔ داده‌ها را به صورت یک لیست پیوندی مرتب شدهٔ (الفبایی) نشان می‌دهد که در آن تنها از یک آرایهٔ اضافی **LINK** برای فیلد اشاره‌گر بعدی لیست استفاده شده است و متغیر **START** به اولین رکورد لیست اشاره می‌کند. ملاحظه می‌کنید که **0** به عنوان اشاره‌گر پوچ **NULL** مورد استفاده قرار گرفته است.



شکل ۵ـ ۷

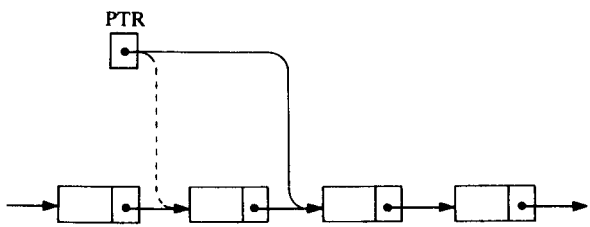
۵ـ۴ پیمایش یک لیست پیوندی

فرض کنید **LIST** یک لیست پیوندی در حافظه باشد و در آرایه‌های خطی **INFO** و **LINK** ذخیره شده است که **START** به عنصر اول اشاره می‌کند و **NULL** پایان لیست را نشان می‌دهد. فرض کنید خواسته باشیم برای پردازش هر گره دقیقاً یکبار **LIST** را پیمایش کنیم. این بخش الگوریتمی را ارائه می‌دهد که این عمل را انجام می‌دهد و آنگاه از این الگوریتم در برخی از کاربردها استفاده می‌کنیم.

الگوریتم پیمایش از متغیر اشاره‌گر **PTR** که به گرهٔ در حال پردازش اشاره دارد استفاده می‌کند. بنابراین **LINK[PTR]** به گرهٔ بعدی اشاره می‌کند که قرار است پردازش شود. بنابراین دستور جایگزینی

PTR := LINK[PTR]

اشاره‌گر را به گرهٔ بعدی در لیست، به صورتی که در نمودار ۸ـ۵ نشان داده شده است، انتقال می‌دهد.



PTR := LINK[PTR] شکل ۸ـ۵

جزئیات این الگوریتم به صورت زیر است: به **PTR** یا **START** مقدار اولیه می‌دهیم. آنگاه **INFO[PTR]** یا اطلاعات گرهٔ اول را پردازش می‌کنیم. **PTR** را با دستور جایگزینی، **PTR := LINK[PTR]** تازه می‌کنیم. از این‌رو **PTR** به گرهٔ دوم اشاره می‌کند. آنگاه **INFO[PTR]** یعنی اطلاعات گرهٔ دوم را پردازش می‌کنیم. مجدداً **PTR** را با دستور جایگزینی **PTR := LINK[PTR]** تازه می‌کنیم. آنگاه **INFO[PTR]** یعنی اطلاعات گرهٔ سوم را پردازش می‌کنیم. همینطور تا آخر، تا زمانی که **PTR = NULL** نشده است کار را ادامه می‌دهیم که خود علامت پایان لیست است.

نمایش رسمی این الگوریتم به صورت زیر است:

Algorithm 5.1: (Traversing a Linked List) Let LIST be a linked list in memory. This algorithm traverses LIST, applying an operation PROCESS to each element of LIST. The variable PTR points to the node currently being processed.

1. Set PTR := START. [Initializes pointer PTR.]
2. Repeat Steps 3 and 4 while PTR ≠ NULL.
3. Apply PROCESS to INFO[PTR].
4. Set PTR := LINK[PTR]. [PTR now points to the next node.]
   [End of Step 2 loop.]
5. Exit.

به شباهت بین الگوریتم **5.1** و الگوریتم **4.1** توجه کنید که یک آرایهٔ خطی را پیمایش می‌کند. **شباهت این دو الگوریتم متأثر از این واقعیت است که هر دو الگوریتم دارای ساختارهای خطی هستند که** شامل یک ترتیب خطی طبیعی از عناصر است.

مواظب باشید : همانند آرایه‌های خطی، عمل **PROCESS** در الگوریتم **5.1** ممکن است از چند متغیر استفاده کند که باید قبل از اِعمال **PROCESS** بر هر یک از عناصر داخل **LIST**، مقدار اولیه بگیرد. درنتیجه، ممکن است الگوریتم با مرحلهٔ مقدار اولیه‌دادن دنبال شود.

مثال ۶ـ ۵

زیربرنامه **PROCEDURE** زیر،اطلاعات هر گرهٔ لیست پیوندی را چاپ می‌کند. از آنجا که زیربرنامه باید لیست را پیمایش کند، از این‌رو شباهت زیادی با الگوریتم **5.1** خواهد داشت.

```
Procedure: PRINT(INFO, LINK, START)
           This procedure prints the information at each node of the list.

        1. Set PTR := START.
        2. Repeat Steps 3 and 4 while PTR ≠ NULL:
        3.     Write: INFO[PTR].
        4.     Set PTR := LINK[PTR]. [Updates pointer.]
           [End of Step 2 loop.]
        5. Return.
```

به بیان دیگر، زیربرنامهٔ **PROCEDURE** می‌تواند تنها با جانشانی دستور

Write: INFO[PTR]

برای مرحلهٔ پردازش در الگوریتم **5.1** به دست آید.

مثال ۷ـ ۵

زیربرنامهٔ **PROCEDURE** زیر **NUM** تعداد عناصر یک لیست پیوندی را پیدا می‌کند.

```
Procedure: COUNT(INFO, LINK, START, NUM)

        1. Set NUM := 0. [Initializes counter.]
        2. Set PTR := START. [Initializes pointer.]
        3. Repeat Steps 4 and 5 while PTR ≠ NULL.
        4.     Set NUM := NUM + 1. [Increases NUM by 1.]
        5.     Set PTR := LINK[PTR]. [Updates pointer.]
           [End of Step 3 loop.]
        6. Return.
```

ملاحظه می‌کنید که این زیربرنامه لیست پیوندی را برای شمارش تعداد عناصر آن پیمایش می‌کند. از

این‌رو، این زیربرنامه خیلی به الگوریتم پیمایش بالا یعنی الگوریتم 5.1 شبیه است. باوجود این در اینجا قبل از پیمایش لیست لازم است به متغیر NUM در یک مرحله مقدار اولیه داده شود. به‌بیان دیگر، زیربرنامه را می‌توان به صورت زیر نوشت :

Procedure: COUNT(INFO, LINK, START, NUM)

1. Set NUM := 0. [Initializes counter.]
2. Call Algorithm 5.1, replacing the processing step by:
   Set NUM := NUM + 1.
3. Return.

اکثر برنامه‌هایی که پردازش لیستها را انجام می‌دهد به این صورت هستند. مسأله ۳ـ ۵ را ببینید.

۵ـ ۵ جستجو در یک لیست پیوندی

فرض کنید یک لیست پیوندی **LIST** به صورت نشان داده شده در بخشهای ۳ـ ۵ و ۴ـ ۵، در حافظه ذخیره شده است. فرض کنید عنصر مشخص **ITEM** داده شده است.این بخش دو الگوریتم جستجو برای تعیین مکان **LOC** گره‌ای راکه در آن **ITEM** برای اولین بار در **LIST** ظاهر شده است مورد بحث و بررسی قرار می‌دهد. در الگوریتم اول فرض می‌شود که داده‌ها در **LIST** مرتب شده، نیستند در حالی که در الگوریتم دوم فرض می‌شود **LIST** مرتب شده است.

اگر **ITEM** درواقع یک مقدار کلیدی باشد و بخواهیم در یک فایل رکوردی را جستجوی کنیم که شامل **ITEM** است، آنگاه **ITEM** می‌تواند تنها یک بار در **LIST** ظاهر شود.

لیست LIST مرتب‌شده نیست

فرض کنید داده‌ها در لیست **LIST** لزوماً مرتب شده نباشد. آنگاه عمل جستجوی **ITEM** در لیست **LIST** را با پیمایش لیست با استفاده از متغیر اشاره‌گر **PTR** و مقایسهٔ **ITEM** با محتوای **INFO[PTR]** هر گرهٔ لیست **LIST** یک گره به یک گره انجام می‌دهیم. قبل از آن‌که اشاره‌گر **PTR** را با دستور

PTR := LINK[PTR]

تازه کنیم، لازم است دو آزمایش زیر را انجام دهیم. نخست باید تحقیق کنیم که آیا به انتهای لیست رسیده‌ایم یا خیر؟ یعنی نخست باید تعیین کنیم که آیا

PTR = NULL

یا خیر. در صورت منفی‌بودن جواب، باید تحقیق کنیم که آیا

INFO[PTR] = ITEM

یا خیر. این دو آزمایش را نمی‌توان در یک زمان انجام داد چون وقتی PTR = NULL باشد INFO[PTR] معنی ندارد و تعریف نشده است. بنابراین برای کنترل اجرای حلقه از آزمایش اول استفاده می‌کنیم و آزمایش دوم را در درون حلقه انجام می‌دهیم. الگوریتم به صورت زیر است :

Algorithm 5.2 SEARCH(INFO, LINK, START, ITEM, LOC)
LIST is a linked list in memory. This algorithm finds the location LOC of the node where ITEM first appears in LIST, or sets LOC = NULL.

```
1. Set PTR := START.
2. Repeat Step 3 while PTR ≠ NULL:
3.     If ITEM = INFO[PTR], then:
           Set LOC := PTR, and Exit.
       Else:
           Set PTR := LINK[PTR]. [PTR now points to the next node.]
       [End of If structure.]
   [End of Step 2 loop.]
4. [Search is unsuccessful.] Set LOC := NULL.
5. Exit.
```

پیچیدگی این الگوریتم همان پیچیدگی الگوریتم جستجوی خطی برای آرایه‌های خطی است که در بخش ۷ـ ۴ بررسی شد. به بیان دیگر، زمان اجرای بدترین حالت، متناسب با n تعداد عناصر لیست LIST است و زمان اجرای حالت میانگین تقریباً با $n / 2$ متناسب است. با این شرط که ITEM در لیست LIST یکبار ظاهر شده است که در هر گرهٔ لیست LIST احتمال مساوی دارد.

مثال ۸ـ ۵

فایل پرسنلی شکل ۷ـ ۵ را درنظر بگیرید. قطعهٔ برنامهٔ زیر شمارهٔ تأمین اجتماعی NNN یک کارمند را می‌خواند و آنگاه حقوق کارمند را 5 درصد افزایش می‌دهد.

```
1. Read: NNN.
2. Call SEARCH(SSN, LINK, START, NNN, LOC).
3. If LOC ≠ NULL, then:
       Set SALARY[LOC] := SALARY[LOC] + 0.05 * SALARY[LOC],
   Else:
       Write: NNN is not in file.
   [End of If structure.]
4. Return.
```

این قطعه برنامه حالت بروز خطا در واردکردن شمارهٔ تأمین اجتماعی NNN را درنظر می‌گیرد.

لیست LIST مرتب شده است

فرض کنید داده‌ها در لیست **LIST** مرتب شده باشند. مجدداً عمل جستجوی **ITEM** در لیست **LIST** را با پیمایش لیست با استفاده از متغیر اشاره‌گر **PTR** و مقایسهٔ **ITEM** با محتوای **INFO[PTR]** هر گرهٔ لیست **LIST** یک گره به یک گره انجام می‌دهیم. با وجود این اکنون هرگاه **ITEM** بزرگتر از **INFO[PTR]** شود کار را متوقف می‌کنیم. الگوریتم به صورت زیر است:

Algorithm 5.3: SRCHSL(INFO, LINK, START, ITEM, LOC)
LIST is a sorted list in memory. This algorithm finds the location LOC of the node where ITEM first appears in LIST, or sets LOC = NULL.

```
1. Set PTR := START.
2. Repeat Step 3 while PTR ≠ NULL:
3.      If ITEM < INFO[PTR], then:
            Set PTR := LINK[PTR]. [PTR now points to next node.]
        Else if ITEM = INFO[PTR], then:
            Set LOC := PTR, and Exit. [Search is successful.]
        Else:
            Set LOC := NULL, and Exit. [ITEM now exceeds INFO[PTR].]
        [End of If structure.]
    [End of Step 2 loop.]
4. Set LOC := NULL.
5. Exit.
```

پیچیدگی این الگوریتم نیز برابر دیگر الگوریتم‌های جستجوی خطی است یعنی زمان اجرای بدترین حالت متناسب با n تعداد عناصر لیست **LIST** است و زمان اجرای حالت میانگین تقریباً با $n / 2$ متناسب است.

یادآور می‌شویم که با یک آرایهٔ خطی مرتب شده می‌توان یک جستجوی خطی را که زمان اجرای آن متناسب با $\log_2 n$ است داشته باشیم. از طرف دیگر، از آنجا که هیچ راهی برای مشخص‌کردن عنصر وسط لیست پیوندی وجود ندارد یک الگوریتم جستجوی دودویی را نمی‌توان بر یک لیست پیوندی مرتب شده بکار بست. این خاصیت یکی از عیب‌های اصلی استفاده از لیست پیوندی به عنوان یک ساختمان داده است.

مثال ۹ـ ۵

بار دیگر فایل پرسنلی شکل ۷ـ ۵ را درنظر بگیرید. قطعه برنامه زیر نام **EMP** یک کارمند را می‌خواند و حقوق کارمند را 5 درصد افزایش می‌دهد. این مثال را با مثال ۸ـ ۵ مقایسه کنید.

```
1. Read: EMPNAME.
2. Call SRCHSL(NAME, LINK, START, EMPNAME, LOC).
3. If LOC ≠ NULL, then:
     Set SALARY[LOC] := SALARY[LOC] + 0.05 * SALARY[LOC].
   Else:
       Write: EMPNAME is not in list.
   [End of If structure.]
4. Return.
```

ملاحظه می‌کنید که اکنون می‌توانیم الگوریتم دوم جستجو، یعنی الگوریتم **5.3** را مورد استفاده قرار دهیم چون لیست به صورت الفبایی مرتب شده است.

۶ـ۵ تخصیص حافظه، جمع‌آوری حافظهٔ بلااستفاده

از جمله مواردِ تعمیر و نگهداری لیستهای پیوندی در حافظه امکان اضافه‌کردن گره‌های جدید در لیست است و این امر مستلزم مکانیسمی است که حافظهٔ بلااستفاده را به گره‌های جدید اختصاص دهد. به‌طور مشابه، به مکانیسمی موردنیاز است که به‌وسیلهٔ آن حافظهٔ گره‌های حذف‌شده برای استفادهٔ آتی در دسترس باشد. این مباحث در این بخش بررسی می‌شود، حال آن که بحث کلی اضافه‌کردن و حذف گره‌ها تا بخشهای بعد به تعویق می‌افتد.

به همراه لیستهای پیوندی، لیست خاصی در حافظه نگهداری می‌شود که از خانه‌های حافظه بلااستفاده تشکیل می شود. خانه‌های حافظه بلااستفاده دارای اشاره‌گر مخصوص به‌خود است که **لیست حافظهٔ موجود** یا **لیست حافظهٔ آزاد** یا **مخزن حافظهٔ آزاد** نامیده می‌شود.

فرض کنید لیستهای پیوندی به‌وسیلهٔ آرایه‌های موازی به صورتی که در بخشهای قبل بیان شد پیاده‌سازی شده است و بخواهیم عملیات اضافه‌کردن و حذف را بر روی لیستهای پیوندی انجام دهیم. آنگاه حافظه‌های بلااستفاده در آرایه‌ها نیز با هم پیوند داده می‌شوند و با استفاده از **AVAIL** به عنوان متغیر اشاره‌گر لیست، تشکیل یک لیست پیوندی را می‌دهد. از این‌رو به لیست حافظهٔ آزاد لیست **AVAIL** نیز می‌گویند. چنین ساختمان داده‌ای را اغلب به صورت

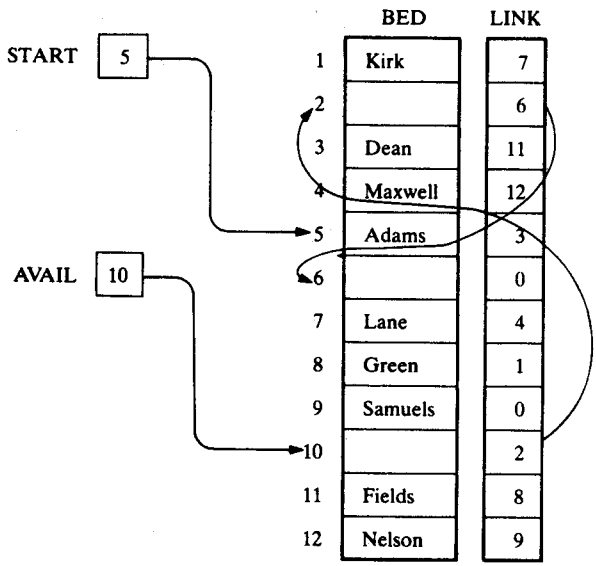
LIST(INFO, LINK, START, AVAIL)

نمایش می‌دهند.

مثال ۱۰ـ۵

فرض کنید لیست بیماران مثال ۱ـ۵ در آرایه‌های خطی **BED** و **LINK** ذخیره شده است طوری که بیمار تخت **K** ام در **BED[K]** جایگزین می‌شود. در آن‌صورت فضای موجود در آرایهٔ خطی **BED** را می‌توان مانند شکل ۹ـ۵ پیوند داد. ملاحظه می‌کنید که **BED[10]** اولین تخت در دسترس، و **BED[2]**

تخت در دسترس بعدی و **BED[6]** آخرین تخت در دسترس است، بنابراین **BED[6]** اشاره‌گر پوچ فیلد اشاره‌گر بعدی‌اش را داراست یعنی **LINK[6] = 0** .



شکل ۹ـ ۵

مثال ۱۱ـ ۵

(الف) فضای دردسترس آرایهٔ خطی **TEST** در شکل ۵ـ ۵ را می‌توان مانند شکل ۱۰ـ ۵ به هم پیوند داد. ملاحظه می‌کنید که هر یک از لیستهای **ALG** و **GEOM** می‌توانند از لیست **AVAIL** استفاده کنند. توجه دارید که **AVAIL = 9** از این‌رو **TEST[9]** اولین گرهٔ آزاد در لیست **AVAIL** است چون **TEST[10], LINK[AVAIL] = LINK[9] = 10** دومین گرهٔ آزاد در لیست **AVAIL** است و الی آخر.

(ب) فایل پرسنلی در شکل ۷ـ ۵ را درنظر بگیرید. فضای در دسترس در آرایهٔ خطی **NAME** را می‌توان مانند شکل ۱۱ـ ۵ به هم پیوند داد. ملاحظه می‌کنید که لیست حافظهٔ آزاد در **NAME** از **NAME[5], NAME[13], NAME[11], NAME[8]** و **NAME[1]** تشکیل شده است. علاوه بر این ملاحظه می‌کنید که مقدارها در **LINK** به صورت همزمان فضای حافظهٔ آزاد را برای آرایه‌های خطی **SSN, SEX** و **SALARY** لیست می‌کند.

(ج) فضای در دسترس در آرایهٔ **CUSTOMER** شکل ۶ـ ۵ را می‌توان مانند شکل ۱۳ـ ۵ به هم پیوند

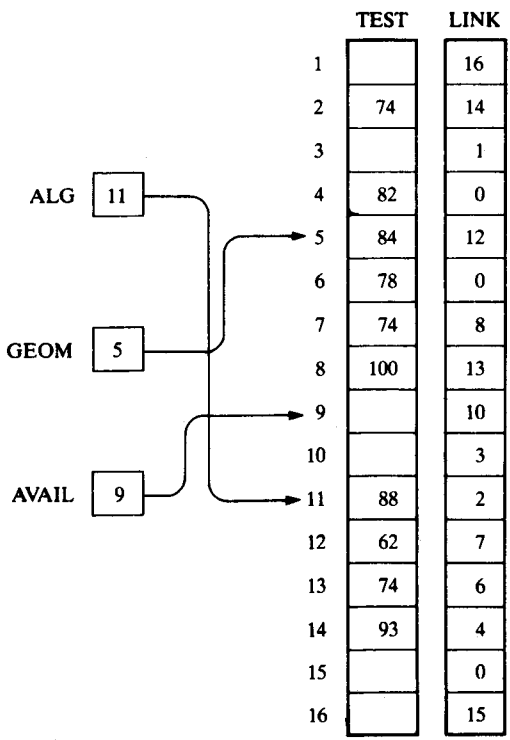
داد. تأکید می‌کنیم که هر یک از چهار لیست، می‌تواند از لیست AVAIL برای مشتری جدید استفاده کند.

مثال ۱۲ـ ۵

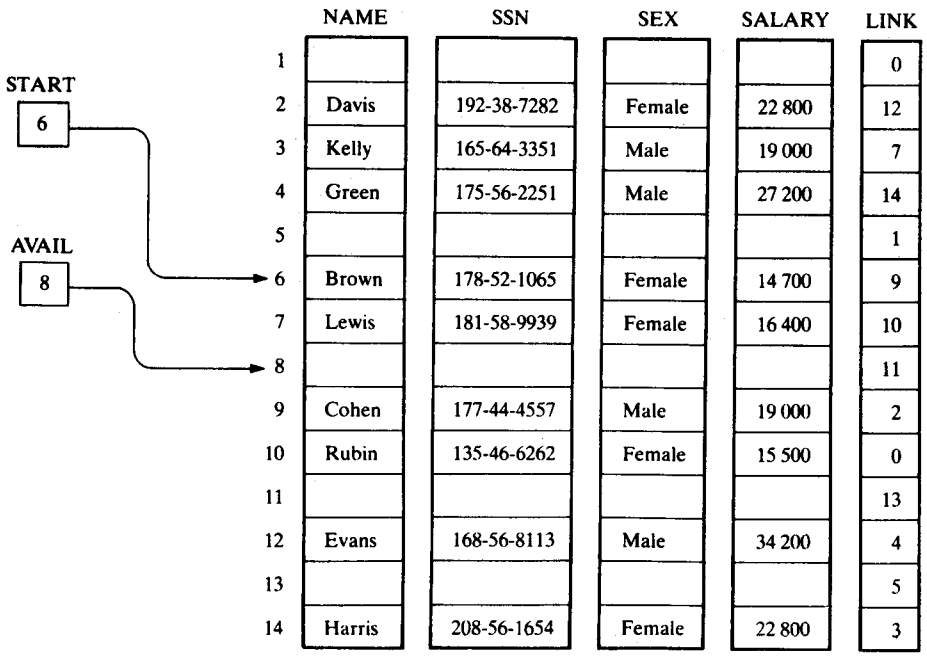
فرض کنید LIST(INFO, LINK, START, AVAIL) برای $n = 10$ گره فضای حافظه دارد. علاوه براین فرض کنید LIST از ابتدا خالی است، شکل ۱۲ـ ۵ مقدارهای LINK را به گونه‌ای نشان می‌دهد که لیست AVAIL از دنباله‌های

INFO[1], INFO[2], ..., INFO[10]

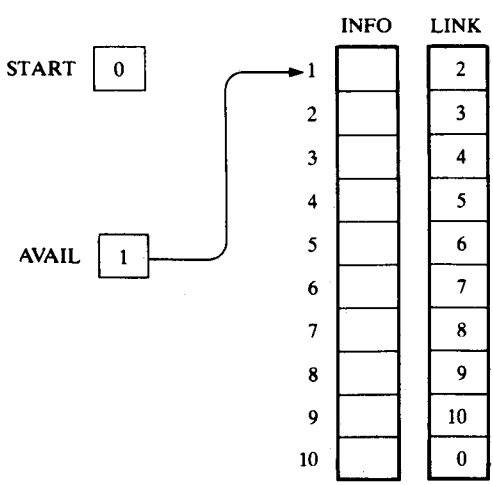
تشکیل شده است به عبارت دیگر لیست AVAIL از عناصر INFO با ترتیب معمول تشکیل شده است. ملاحظه می‌کنید که چون لیست خالی است START = NULL .



شکل ۱۰ـ ۵



شکل ۱۱ـ۵



شکل ۱۲ـ۵