(ب) قرار دهید TOP := NULL و ADD := Main .

(۲) تبدیل "مرحلهٔ K ، فراخوانی P "

(الف) مقادیر جاری پارامترها و متغیرهای محلی و آدرس بازگشت جاری ADD را به داخل پشته‌های مربوطه Push کنید.

(ب) با استفاده از مقادیر آرگومان‌جدید پارامترها را مقداردهی کنید و قرار دهید ADD := [Step] K+1 .

(ج) Go to Step 1 [ابتدای زیربرنامهٔ P]

(۳) تبدیل "مرحلهٔ J ، بازگشت Return"

(الف) اگر ADD = Main آنگاه Return [کنترل به برنامهٔ اصلی منتقل می‌شود.]

(ب) قرار دهید SAVE := ADD .

(ج) مقادیر بالای پشته‌ها را برگردانید، یعنی پارامترها و متغیرهای محلی را برابر مقادیر بالای پشته‌ها قرار دهید و ADD را برابر مقدار بالای پشتهٔ STADD قرار دهید.

(د) Go to Step SAVE .

این الگوریتم تبدیل را با الگوریتم بخش ۸ـ ۶ مقایسه کنید.

صفها، صفهای دوسره Deque

مسأله ۲۱ـ ۶ : صف کاراکترهای زیر را درنظر بگیرید که در آن QUEUE یک آرایهٔ چرخشی یا حلقوی است که شش خانهٔ حافظه به آن اختصاص داده شده است :

FRONT = 2, REAR = 4 QUEUE: __, A, C, D, __, __

جهت سهولت در نمادگذاری از "ـ" برای نمایش خانهٔ حافظهٔ خالی استفاده می‌کنیم. وضعیت صف را به‌محض انجام هر یک از عملیات زیر شرح دهید :

(الف) F به صف اضافه می‌شود. (و) دو حرف حذف می‌شود.

(ب) دو حرف حذف می‌شوند. (ز) S صف اضافه می‌شود.

(ج) K, L و M به صف اضافه می‌شوند. (ح) دو حرف حذف می‌شود.

(د) دو حرف حذف می‌شود. (ط) یک حرف حذف می‌شود.

(هـ) R به صف اضافه می‌شود. (ی) یک حرف حذف می‌شود.

حل: (الف) F به انتهای صف اضافه می‌شود، درنتیجه

FRONT = 2, REAR = 5 QUEUE: __, A, C, D, F, __

توجه دارید که REAR به اندازهٔ یک واحد افزایش می‌یابد.

(ب) دو حرف A و C حذف می‌شوند، باقی می‌ماند :

FRONT = 4, REAR = 5 QUEUE: __, __, __, D, F, __

توجه دارید که FRONT دو واحد افزایش می‌یابد.

(ج) K, L و M به انتهای صف اضافه می‌شوند. از آنجا که K در آخرین خانهٔ حافظهٔ QUEUE قرار می‌گیرد L و M در دو خانهٔ اول قرار می‌گیرند. درنتیجه :

FRONT = 4, REAR = 2 QUEUE: L, M, __, D, F, K

توجه دارید که REAR به اندازهٔ 3 واحد افزایش می‌یابد که حساب با باقیماندهٔ 6 است.

REAR = 5 + 3 = 8 = 2 (mod 6)

(د) دو حرف ابتدایی D و F حذف می‌شوند، باقی می‌ماند :

FRONT = 6, REAR = 2 QUEUE: L, M, __, __, __, K

(ه) R به انتهای صف اضافه می‌شود، درنتیجه :

FRONT = 6, REAR = 3 QUEUE: L, M, R, __, __, K

(و) دو حرف ابتدایی K و L حذف می‌شوند، باقی می‌ماند :

FRONT = 2, REAR = 3 QUEUE: __, M, R, __, __, __

توجه دارید که FRONT به اندازهٔ دو واحد افزایش می‌یابد که حساب با باقیماندهٔ 6 است :

FRONT = 6 + 2 = 8 = 2 (mod 6)

(ز) S به انتهای صف اضافه می‌شود، درنتیجه :

FRONT = 2, REAR = 4 QUEUE: __, M, R, S, __, __

(ح) دو حرف ابتدایی M و R حذف می‌شود، باقی می‌ماند :

FRONT = 4, REAR = 4 QUEUE: __, __, __, S, __, __

(ط) حرف ابتدا S حذف می‌شود. چون FRONT = REAR درنتیجه صف خالی است. از این‌رو NULL را در FRONT و REAR جایگزین می‌کنیم. بنابراین

FRONT = 0, REAR = 0 QUEUE: __, __, __, __, __, __

(ی) چون FRONT = NULL ، هیچ عمل حذفی نمی‌تواند صورت گیرد، زیرریزی اتفاق می‌افتد.

مسأله ۲۲ـ ۶ : فرض کنید هر ساختمان داده در یک آرایهٔ چرخشی با N خانهٔ حافظه ذخیره می‌شود.

(الف) تعداد عناصر NUMB صف را برحسب FRONT و REAR تعیین کنید.

(ب) تعداد عناصر NUMB صف دوسره یا Deque رابرحسب LEFT و RIGHT تعیین کنید.

(ج) چه وقت آرایه پر خواهد شد؟

حل : (الف) اگر $FRONT \leq REAR$ ، آنگاه $NUMB = REAR - FRONT + 1$. برای مثال صف زیر را با $N = 12$ درنظر بگیرید :

FRONT = 3, REAR = 9 QUEUE: ــ, ــ, *, *, *, *, *, *, *, *, ــ, ــ, ــ

آنگاه $NUMB = 9 - 3 + 1 = 7$, که در تصویر نشان داده شده است.

اگر $REAR < FRONT$ ، آنگاه $FRONT - REAR - 1$ تعداد خانهٔ خالی می‌باشد، بنابراین :

$$NUMB = N - (FRONT - REAR - 1) = N + REAR - FRONT + 1$$

برای مثال صف زیر با $N = 12$ را درنظر بگیرید :

FRONT = 9, REAR = 4 QUEUE: *, *, *, *, ــ, ــ, ــ, ــ, *, *, *, *

آنگاه $NUMB = 12 + 4 - 9 + 1 = 8$ ، که در تصویر نشان داده شده است.

با استفاده از حساب با باقیماندهٔ N تنها به یک فرمول زیر احتیاج داریم :

$$NUMB = REAR - FRONT + 1 \pmod N$$

(ب) همان نتیجه برای صفهای دوسره یا Deque ها نیز بدست می‌آید با این تفاوت که دراینجا به جای FRONT ، RIGHT جایگزین می‌شود یعنی

$$NUMB = RIGHT - LEFT + 1 \pmod N$$

(ج) زمانی که

$$FRONT = 1 \quad (i) \qquad FRONT = REAR + 1 \quad (ii) \qquad \text{یا} \qquad REAR = N \quad \text{و}$$

با یک صف آرایه پر است.

به همین ترتیب، وقتی

$$LEFT = 1 \quad (i) \qquad LEFT = RIGHT + 1 \quad (ii) \qquad \text{یا} \qquad RIGHT = N \quad \text{و}$$

با یک صف دوسره یا Deque آرایه پر است.

از هر یک از این شرایط نتیجه می‌شود $NUMB = N$.

مسأله ۲۳ـ ۶ : صف دوسره Deque کاراکترهای زیر را درنظر بگیرید که در آن DEQUE یک آرایهٔ چرخشی یا حلقوی است و شش خانهٔ حافظه به آن اختصاص داده شده است.

LEFT = 2, RIGHT = 4 DEQUE: ــ, A, C, D, ــ, ــ

وضعیت صف دوسره Deque را به محض انجام هر یک از عملیات زیر شرح دهید.

(الف) F در سمت راست به صف دوسره اضافه می‌شود.

(ب) دو حرف از سمت راست حذف می‌شوند.

(ج) K ، L و M در سمت چپ به صف دوسره اضافه می‌شوند.

(د) یک حرف از سمت چپ حذف می‌شود.

(ه) R در سمت چپ به صف دوسره اضافه می‌شود.

(و) S در سمت راست به صف دوسره اضافه می‌شود.

(ز) T در سمت راست به صف دوسره اضافه می‌شود.

حل : (الف) F در سمت راست اضافه می‌شود، درنتیجه :

LEFT = 2, RIGHT = 5 DEQUE: _, A, C, D, F, _

توجه دارید که RIGHT به اندازهٔ واحد اضافه می‌شود.

(ب) دو حرف سمت راست، F و D حذف می‌شوند، درنتیجه

LEFT = 2, RIGHT = 3 DEQUE: _, A, C, _, _, _

توجه دارید که RIGHT دو واحد افزایش می‌یابد.

(ج) K ، L و M در سمت چپ اضافه می‌شوند. چون K اولین خانهٔ حافظه و L آخرین خانهٔ حافظه و M یک خانه مانده به آخرین خانهٔ حافظه است. درنتیجه

LEFT = 5, RIGHT = 3 DEQUE: K, A, C, _, M, L

درنتیجه توجه دارید که LEFT به 3 واحد افزایش می‌یابد که حساب با باقیمانده 6 است :

$$LEFT = 2 - 3 = -1 = 5 \pmod 6$$

(د) حرف چپ، M ، حذف می‌شود و باقی می‌ماند :

LEFT = 6, RIGHT = 3 DEQUE: K, A, C, _, _, L

توجه دارید که LEFT به اندازهٔ واحد افزایش می‌یابد.

(ه) R در سمت چپ اضافه می‌شود، درنتیجه :

LEFT = 5, RIGHT = 3 DEQUE: K, A, C, _, R, L

توجه دارید که LEFT به اندازهٔ واحد افزایش می‌یابد.

(و) S در سمت راست به صف دوسره اضافه می‌شود، درنتیجه :

LEFT = 5, RIGHT = 4 DEQUE: K, A, C, S, R, L

(ز) چون LEFT = RIGHT + 1 ، آرایه پر است و از این‌رو T نمی‌تواند به صف دوسره اضافه شود، سرریزی اتفاق می‌افتد.

مسأله ۲۴ـ ۶ : یک صف دوسره Deque را در نظر بگیرید که در یک آرایهٔ چرخشی با N خانهٔ حافظه نگهداری می‌شود.

(الف) فرض کنید یک عنصر به صف دوسره **Deque** اضافه شده است **LEFT** یا **RIGHT** چگونه تغییر می‌کنند؟

(ب) فرض کنید یک عنصر از صف دوسره حذف می‌شود. **LEFT** یا **RIGHT** چگونه تغییر می‌کنند؟

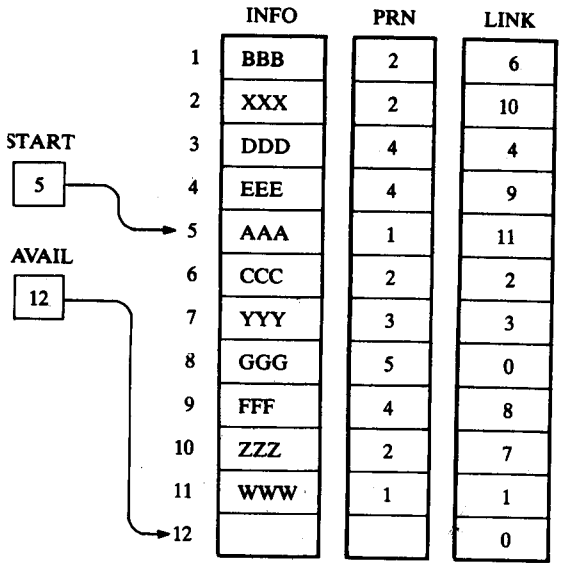
حل: (الف) اگر این عنصر در سمت چپ اضافه شود، آنگاه **LEFT** به اندازهٔ واحد **(mod N)** اضافه می‌شود. از طرف دیگر، اگر این عنصر در سمت راست اضافه شود، آنگاه **RIGHT** به اندازهٔ واحد **(mod N)** اضافه می‌شود.

(ب) اگر این عنصر از سمت چپ حذف شود، آنگاه **LEFT** به اندازهٔ واحد **(mod N)** افزایش می‌یابد و اگر این عنصر از سمت راست حذف شود، آنگاه **RIGHT** به اندازهٔ واحد **(mod N)** کاهش می‌یابد. درحالتی که **LEFT = RIGHT** قبل از عمل حذف (یعنی وقتی که صف **Deque** تنها یک عنصر دارد)، آنگاه هم **LEFT** و هم **RIGHT**، جایگزین می‌شود که بیانگر آن هستند صف دوسره، خالی است.

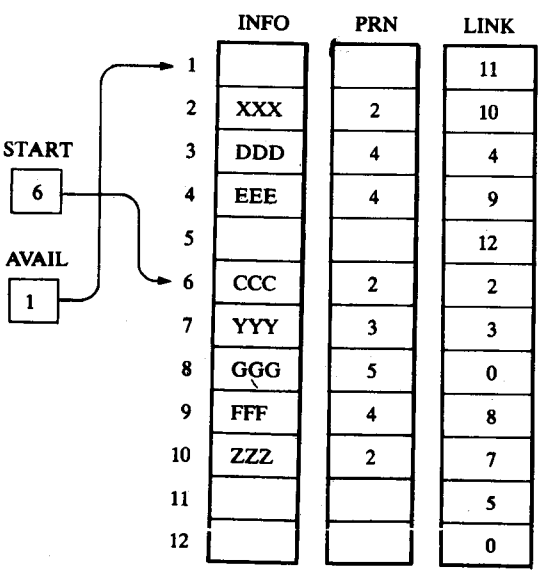
صفهای اولویت

مسأله ۲۵ـ ۶: صف اولویت شکل ۲۰ـ ۶ را درنظر بگیرید که به صورت یک لیست یکطرفه نگهداری می‌شود. **(الف)** این ساختمان داده را پس از اضافه‌شدن **(ZZZ, 2)**, **(YYY, 3)**, **(XXX, 2)** و **(WWW, 1)** به صف توصیف کنید. **(ب)** اگر پس از عملیات اضافه‌شدن قبلی، سه عنصر از این ساختمان داده حذف شوند وضعیت آن را توصیف کنید.

حل: (الف) لیست را پیمایش کرده اولین عنصری را که عدد اولویت آن بزرگتر از عدد اولویت **XXX** است پیدا می‌کنیم. این عنصر **DDD** است. از این‌رو **XXX** را قبل از **DDD** (بعد از **CCC**) در خانهٔ حافظهٔ خالی اول **INFO[2]** اضافه می‌کنیم. آنگاه لیست را پیمایش کرده اولین عنصری را که عدد اولویت آن بزرگتر از عدد اولویت **YYY** است پیدا می‌کنیم. این بار نیز این عنصر، **DDD** است. از این‌رو **YYY** را قبل از **DDD** (بعد از **XXX**) در خانهٔ خانهٔ خالی بعدی، **INFO[7]** اضافه می‌کنیم. آنگاه لیست را پیمایش کرده اولین عنصری را که عدد اولویت آن بزرگتر از عدد اولویت **ZZZ** است پیدا می‌کنیم. این عنصر، **YYY** است. از این‌رو **ZZZ** را قبل از **YYY** (بعد از **XXX**) در خانهٔ حافظهٔ بعدی، **INFO[10]** اضافه می‌کنیم. بالاخره، لیست را پیمایش کرده اولین عنصری را که عدد اولویت آن بزرگتر از عدد اولویت **WWW** است پیدا می‌کنیم. این عنصر، **BBB** است. از این‌رو **WWW** را قبل از **BBB** (بعد از **AAA**) در خانهٔ حافظهٔ خالی بعدی، **INFO[11]** اضافه می‌کنیم. نهایتاً ساختمان دادهٔ شکل ۲۶ـ ۶ **(الف)** نتیجه می‌شود.



(الف)



(ب)

شکل ۲۶ـ ۶

(ب) سه عنصر اول از لیست یکطرفه حذف می‌شوند. به‌طور مشخص، نخست **AAA** حذف می‌شود و خانهٔ حافظهٔ آن **INFO[5]** به لیست **AVAIL** اضافه می‌شود. آنگاه **WWW** حذف می‌شود و خانهٔ حافظهٔ آن **INFO[11]** به لیست **AVAIL** اضافه می‌شود. بالاخره **BBB** حذف می‌شود و خانهٔ حافظهٔ آن **INFO[1]** به لیست **AVAIL** اضافه می‌شود. نهایتاً ساختمان دادهٔ شکل ۲۶ـ ۶ (ب) نتیجه می‌شود.

توجه کنید : ملاحظه می‌کنید که **START** و **AVAIL** باتوجه به عملیات مسأله، تغییر می‌کنند.

مسأله ۲۶ـ ۶ : صف اولویت شکل ۲۲ـ ۶ را درنظر بگیرید که در آرایهٔ دوبعدی **QUEUE** نگهداری می‌شود. **(الف)** وضعیت این ساختمان داده را پس از اضافه‌شدن **(TTT, 1), (SSS, 4), (RRR, 3)** و **(VVV, 2), (UUU, 4)** به صف توضیح دهید. **(ب)** اگر پس از عملیات اضافه‌شدن قبلی، سه عنصر از این ساختمان داده حذف شوند وضعیت صف را توضیح دهید.

حل: (الف) هر عنصر را در ردیف اولویت مربوطه‌اش اضافه می‌کنیم، یعنی **RRR** را به عنوان عنصر انتهای ردیف **3** ، **SSS** را به عنوان عنصر انتهای ردیف **4** ، و **TTT** را به عنوان عنصر انتهای ردیف **1** ، **UUU** را به عنوان عنصر انتهای ردیف **4** ، و **VVV** را به عنوان عنصر انتهای ردیف **2** اضافه می‌کنیم با این عملیات به ساختمان دادهٔ شکل ۲۷ـ ۶ (الف) می‌رسیم. همانگونه که قبل از این متذکر شدیم، عملیات اضافه‌کردن در نمایش آرایه‌ای معمولاً ساده‌تر از عملیات اضافه‌کردن در نمایش لیست یکطرفه است.

	FRONT	REAR
1	2	3
2	1	4
3	1	1
4	5	3
5	4	4

QUEUE

	1	2	3	4	5	6
1		AAA	TTT			
2	BBB	CCC	XXX	VVV		
3	RRR					
4	FFF	SSS	UUU		DDD	EEE
5				GGG		

(الف)

	FRONT	REAR
1	0	0
2	2	4
3	1	1
4	5	3
5	4	4

QUEUE

	1	2	3	4	5	6
1						
2		CCC	XXX	VVV		
3	RRR					
4	FFF	SSS	UUU		DDD	EEE
5				GGG		

(ب)

شکل ۲۷ـ ۶

(ب) نخست عناصر ردیف 1 راکه دارای بیشترین الویت هستند حذف می‌کنیم. چون در ردیف 1 تنها دو عنصر، AAA و TTT وجود دارد، آنگاه عنصر ابتدای ردیف 2 ، BBB ، نیز باید حذف شود. نهایتاً این عملیات به ساختمان داده شکل ۲۷ـ ۶ (ب) منتهی می‌شود.

توجه کنید : در هر دو حالت ملاحظه می‌کنید که **FRONT** و **REAR** باتوجه به عملیات مسأله، تغییر می‌کنند.

مسأله‌های تکمیلی

پشته‌ها

مسأله ۲۷ـ ۶ : پشتهٔ زیر متشکل از نام شهرها را درنظر بگیرید :

STACK: London, Berlin, Rome, Paris, _____, _____

(الف) وضعیت پشته را به‌محض انجام عملیات زیر توضیح دهید.

(i) PUSH(STACK, Athens)
(ii) POP(STACK, ITEM)
(iii) POP(STACK, ITEM)
(iv) PUSH(STACK, Madrid)
(v) PUSH(STACK, Moscow)
(vi) POP(STACK, ITEM)

(ب) اگر عمل **POP(STACK, ITEM)** لندن را حذف کند وضعیت پشته را توضیح دهید.

مسأله ۲۸ـ ۶ : پشتهٔ زیر را درنظر بگیرید که در آن به **STACK**، **N = 4** خانهٔ حافظه اختصاص داده شده است.

STACK: AAA, BBB, _____, _____

وضعیت پشته را به محض انجام عملیات زیر توضیح دهید.

(الف) POP(STACK, ITEM)
(ب) POP(STACK, ITEM)
(ج) PUSH(STACK, EEE)
(د) POP(STACK, ITEM)
(ه) POP(STACK, ITEM)
(و) PUSH(STACK, GGG)

مسأله ۲۹ـ ۶ : پشتهٔ زیر متشکل از اعداد صحیح را در حافظه درنظر بگیرید که در آن به **STACK**، **N = 6** خانهٔ حافظه اختصاص داده شده است.

TOP = 3 **STACK:** 5, 2, 3, —, —, —

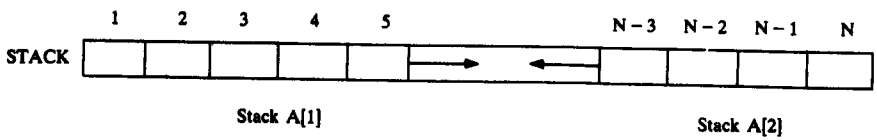
خروجی قطعه برنامهٔ زیر را مشخص کنید.

```
1. Call POP(STACK, ITEMA).
   Call POP(STACK, ITEMB).
   Call PUSH(STACK, ITEMB + 2).
   Call PUSH(STACK, 8).
   Call PUSH(STACK, ITEMA + ITEMB).
2. Repeat while TOP ≠ 0:
       Call POP(STACK, ITEM).
       Write: ITEM.
   [End of loop.]
```

مسأله ۳۰ـ ۶: فرض کنید پشته‌های A[1] و A[2] در آرایهٔ خطی STACK با N عنصر مانند شکل ۲۸ـ ۶ ذخیره شده‌اند. فرض کنید TOP[K] بالای پشتهٔ A[K] نمایش می‌دهد.

(الف) یک زیربرنامهٔ PUSH(STACK, N, TOP, ITEM, K) بنویسید تا ITEM را به داخل پشتهٔ A[K]، Push کند.

(ب) یک زیربرنامهٔ POP(STACK, TOP, ITEM, K) بنویسید تا عنصر بالای پشتهٔ A[K] را حذف کند و این عنصر را در متغیر ITEM جایگزین کند.



شکل ۲۸ـ ۶

عبارتهای محاسباتی، عبارتهای لهستانی

مسأله ۳۱ـ ۶: با بررسی و روش دستی، هر عبارت میانوندی زیر را به عبارت پسوندی معادل آن تبدیل کنید:

(الف) (A − B)/((D + E) * F) **(ب)** ((A + B)/D) ↑ ((E − F) * G)

مسأله ۳۲ـ ۶: با بررسی و روش دستی، هر عبارت میانوندی مسأله ۳۱ـ ۶ را به عبارت پیشوندی معادل آن تبدیل کنید.

مسأله ۳۳ـ ۶: هر یک از عبارت محاسباتی بدون پرانتز زیر را ارزیابی کنید.

(الف) 5 + 3 ↑ 2 − 8 / 4 * 3 + 6

(ب) 6 + 2 ↑ 3 + 9 / 3 − 4 * 5

مسأله ۳۴ـ ۶: عبارت محاسباتی بدون پرانتز زیر را درنظر بگیرید:

E: 6 + 2 ↑ 3 ↑ 2 − 4 * 5

با فرض‌های زیر E را ارزیابی کنید: (الف) همانگونه که در عبارتهای بالا داشتید عمل توان‌رسانی از چپ به راست اجرا شود. (ب) عمل توان‌رسانی از راست به چپ اجرا شود.

مسأله ۳۵ـ ۶ : هر یک از عبارتهای پسوندی زیر را درنظر بگیرید :

P_1: 5, 3, +, 2, *, 6, 9, 7, −, /, −
P_2: 3, 5, +, 6, 4, −, *, 4, 1, −, 2, ↑, +
P_3: 3, 1, +, 2, ↑, 7, 4, −, 2, *, +, 5, −

با بررسی و روش دستی، هر عبارت داده شده را به نماد میانوندی تبدیل کنید و آنگاه مقدار آنها را ارزیابی کنید.

مسأله ۳۶ـ ۶ : هر عبارت پسوندی مسأله ۳۵ـ ۶ را با استفاده از الگوریتم **6.3** ارزیابی کنید.

مسأله ۳۷ـ ۶ : با استفاده از الگوریتم **6.4** هر عبارت میانوندی را به عبارت پسوندی معادل آن تبدیل کنید.

(الف) (A − B)/((D + E) * F) (ب) ((A + B)/D) ↑ ((E − F) * G)

این مسأله را با مسأله ۳۱ـ ۶ مقایسه کنید.

زیربرنامه‌های بازگشتی

مسأله ۳۸ـ ۶ : فرض کنید J و K دو عدد صحیح باشند و Q(J, K) به صورت بازگشتی زیر

$$Q(J, K) = \begin{cases} 5 & \text{if } J < K \\ Q(J - K, K + 2) + J & \text{if } J \geq K \end{cases}$$

تعریف شده است Q(5, 3), Q(2, 7) و Q(15, 2) را محاسبه کنید.

مسأله ۳۹ـ ۶ : فرض کنید A و B دو عدد صحیح نامنفی باشند. فرض کنید تابع GCD به صورت بازگشتی زیر تعریف شده است.

$$GCD(A, B) = \begin{cases} GCD(B, A) & \text{if } A < B \\ A & \text{if } B = 0 \\ GCD(B, MOD(A, B)) & \text{otherwise} \end{cases}$$

در اینجا MOD(A, B) را به صورت "A modulo B" بخوانید که نمایش باقیمانده A تقسیم بر B است.

(الف) GCD(6, 15), GCD(20, 28) و GCD(540, 168) را محاسبه کنید. (ب) این تابع چه عملی انجام می‌دهد.

مسأله ۴۰ـ ۶ : فرض کنید N یک عدد صحیح است و H(N) به صورت بازگشتی زیر تعریف شده است :

$$H(N) = \begin{cases} 3 * N & \text{if } N < 5 \\ \text{otherwise} \end{cases}$$

(الف) معیار پایه H را پیدا کنید و (ب) H(2), H(8) و H(24) را محاسبه کنید.

مسأله ۴۱ـ ۶ : با استفاده از تعریف 6.3 برای تابع آکرمان، A(2, 2) را تعیین کنید.

مسأله ۴۲ـ ۶ : فرض کنید M و N دو عدد صحیح باشند و F(M, N) به صورت بازگشتی زیر تعریف شده است.

$$F(M, N) = \begin{cases} 1 & \text{if } M = 0 \text{ or } M \geq N \geq 1 \\ F(M-1, N) + F(M-1, N-1) & \text{otherwise} \end{cases}$$

(الف) F(1, 5), F(4, 2) و F(2, 4) را پیدا کنید. (ب) چه وقت F(M, N) قابل تعریف نیست؟

مسأله ۴۳ـ ۶ : فرض کنید A یک آرایهٔ N عنصری از اعداد صحیح باشد. فرض کنید X یک تابع صحیح تعریف شده به صورت زیر باشد :

$$X(K) = X(A, N, K) = \begin{cases} 0 & \text{if } K = 0 \\ X(K-1) + A(K) & \text{if } 0 < K \leq N \\ X(K-1) & \text{if } K > N \end{cases}$$

برای هر یک از آرایه‌های زیر X(5) را پیدا کنید :

(الف) N = 8, A: 3, 7, − 2, 5, 6, − 4, 2, 7 (ب) N = 3, A: 2, 7, − 4

این تابع چه عملی انجام می‌دهد؟

مسأله ۴۴ـ ۶ : نشان دهید برای n دیسک راه حل بازگشتی مسأله برج‌های هانوی بخش ۷ـ ۶ مستلزم $f(n) = 2^n - 1$ انتقال یا جابجایی است. نشان دهید هیچ راه حل دیگری وجود ندارد که از $f(n)$ کمتری در انتقال استفاده کند.

مسأله ۴۵ـ ۶ : فرض کنید S یک رشتهٔ N کاراکتری است، و SUB(S, J, L) نمایش زیررشته‌ای از S باشد که از مکان J شروع می‌شود و طول L دارد. و نیز فرض کنید A // B نمایش اتصال دو رشتهٔ A و B باشد و REV(S, N) به صورت بازگشتی زیر تعریف شده است :

$$REV(S, N) = \begin{cases} S & \text{if } N = 1 \\ SUB(S, N, 1) // REV(SUB(S, 1, N-1), N-1) & \text{otherwise} \end{cases}$$

(الف) وقتی (i) N = 3, S = abc و (ii) N = 5, S = ababc است REV(S, N) را محاسبه کنید. (ب) این تابع چه عملی انجام می‌دهد.

صفها، صفهای دوسره Deque

مسأله ۴۶ـ ۶ : صف زیر را درنظر بگیرید که در آن به QUEUE شش خانهٔ حافظهٔ اختصاص داده شده است :

FRONT = 2, REAR = 5 QUEUE: _____, London, Berlin, Rome, Paris, _____

به‌محض انجام عملیات زیر وضع صف و وضع **FRONT** و **REAR** را توضیح دهید.

(الف) **Athens** اضافه می‌شود. **(ب)** دو شهر حذف می‌شوند. **(ج)** **Madrid** اضافه می‌شود.

(د) **Moscow** اضافه می‌شود. **(ه)** سه شهر حذف می‌شوند **(و)** **Oslo** اضافه می‌شود.

مسأله ۴۷ـ ۶ : صف دوسره **Deque** زیر را درنظر بگیرید که در آن به **DEQUE**، **6** خانهٔ حافظه اختصاص داده شده است.

LEFT = 2, RIGHT = 5 **DEQUE:** _____, London, Berlin, Rome, Paris, _____

به‌محض انجام عملیات زیر وضع صف و وضع **LEFT** و **RIGHT** را توضیح دهید.

(الف) **Athens** در سمت چپ اضافه می‌شود. **(ه)** دو شهراز سمت راست حذف می‌شوند.

(ب) دو شهر از سمت راست حذف می‌شوند. **(و)** یک شهراز سمت‌چپ‌حذف می‌شود.

(ج) **Madrid** در سمت چپ اضافه می‌شود. **(ز)** **Oslo** در سمت چپ اضافه می‌شود.

(د) **Moscow** در سمت راست اضافه می‌شود.

مسأله ۴۸ـ ۶ : فرض کنید یک صف در آرایهٔ چرخشی یا حلقوی **QUEUE** با **N = 12** خانهٔ حافظه نگهداری می‌شود. تعداد عناصر صف **QUEUE** را در هر یک از حالات زیر محاسبه کنید.

(الف) **FRONT = 4, REAR = 8;** **(ب)** **FRONT = 10, REAR = 3;** **(ج)** **FRONT = 5, REAR = 6** و آنگاه دو عنصر حذف می‌شوند.

مسأله ۴۹ـ ۶ : صف اولویت شکل ۲۶ـ ۶ (ب) را درنظر بگیرید که در لیست یکطرفه نگهداری می‌شود.

(الف) اگر دو عنصر حذف شوند وضعیت این ساختمان داده را توضیح دهید.

(ب) اگر قبل از عمل حذف‌کردن، عنصرهای **(TTT, 3)**, **(SSS, 1)**, **(RRR, 3)** و **(UUU, 2)** به صف اضافه شوند وضعیت این ساختمان داده را توضیح دهید.

(ج) اگر قبل از عمل اضافه‌کردن، سه عنصر حذف شوند وضعیت این ساختمان داده را توضیح دهید.

مسأله ۵۰ـ ۶ : صف اولویت شکل ۲۷ـ ۶ (ب) را درنظر بگیرید که در آرایهٔ دوبعدی **QUEUE** نگهداری می‌شود.

(الف) اگر دو عنصر حذف شود وضعیت این ساختمان داده را توضیح دهید.

(ب) اگر قبل از عمل حذف‌کردن، عنصرهای **(LLL, 4)**, **(KKK, 1)**, **(JJJ, 3)** و **(MMM, 5)** به صف اضافه شوند وضعیت این ساختمان داده را توضیح دهید.

(ج) اگر پس از عمل اضافه‌کردن قبلی، شش عنصر حذف شوند وضعیت این ساختمان داده را توضیح دهید.

برای مسأله‌های زیر برنامه بنویسید

مسأله ۵۱ـ ۶ : **QuickSort** را به یک زیربرنامهٔ **QUICK(A, N)** تبدیل کنید که آرایهٔ **N** عنصری **A** را مرتب می‌کند. برنامه را با استفاده از داده‌های زیر آزمایش کنید :

(الف) 44, 33, 11, 55, 77, 90, 40, 60, 99, 22, 88, 66

(ب) D, A, T, A, S, T, R, U, C, T, U, R, E, S

مسأله ۵۲ـ ۶ : برنامه‌ای بنویسید تا راه حل مسأله برجهای هانوی را برای n دیسک به دست دهد. برنامه را با استفاده از (الف) $n = 3$ و (ب) $n = 4$ آزمایش کنید.

مسأله ۵۳ـ ۶ : الگوریتم **6.4** را به یک زیربرنامهٔ **POLISH(Q, P)** تبدیل کنید تا یک عبارت میانوندی **Q** را به عبارت پسوندی **P** معادل آن تبدیل کند. فرض کنید هر عملوند یک کاراکتر الفبایی منفرد است و از نمادهای متداول برای جمع (+)، تفریق (−) ضرب (*) و تقسیم (/) استفاده کنید اما برای توان از نماد ↑ یا $ استفاده کنید. (بعضی از زبانهای برنامه‌نویسی نماد ↑ را برای توان قبول نمی‌کنند.) برنامه را با استفاده از

(الف) ((A + B) * D) $ (E − F) (ب) A + (B * C − (D / E $ F) * G) * H

آزمایش کنید.

مسأله ۵۴ـ ۶ : فرض کنید یک صف اولویت مطابق شکل ۲۰ـ ۶ به صورت لیست یکطرفه نگهداری می‌شود.

(الف) یک زیربرنامهٔ **Procedure** بنویسید :

INSPQL(INFO, PRN, LINK, START, AVAIL, ITEM, N)

تا عنصر **ITEM** با عدد اولویت **N** را به صف اضافه کند. الگوریتم **6.4** را ببینید.

(ب) یک زیربرنامهٔ **Procedure** بنویسید :

DELPQL(INFO, PRN, LINK, START, AVAIL, ITEM)

تا یک عنصر را از صف حذف کند و آن را در متغیر **ITEM** جایگزین کند. الگوریتم **6.13** را ببینید.

این زیربرنامه‌های **Procedure** را با استفاده از داده‌های مسأله ۲۵ـ ۶ آزمایش کنید.

مسأله ۵۵ـ ۶ : فرض کنید یک صف اولویت مطابق شکل ۲۳ـ ۶ به صورت آرایهٔ دوبعدی نگهداری می‌شود.

(الف) یک زیربرنامهٔ **Procedure** بنویسید :

INSPQA(QUEUE, FRONT, REAR, ITEM, M)

تا عنصر **ITEM** با عدد اولویت **M** را به صف اضافه کند. الگوریتم **6.16** را ببینید.

(ب) یک زیربرنامهٔ **Procedure** بنویسید:

DELPQL(QUEUE, FRONT, REAR, ITEM)

تا یک عنصر را از صف حذف کند و آن را در متغیر **ITEM** جایگزین کند. الگوریتم **6.15** را ببینید.

این زیربرنامه‌های **Procedure** را با استفاده از داده‌های مسألهٔ ۳۶ـ ۶ آزمایش کنید. فرض کنید **QUEUE** تعداد **ROW** سطر و تعداد **COL** ستون دارد که در آن **ROW** و **COL** متغیرهای سراسری هستند.

فصل ۷

درختـها

۱ـ۷ مقدمه

تا اینجا، انواع مختلف ساختمان داده‌های خطی ازقبیل رشته‌ها، آرایه‌ها، لیستها، پشته‌ها و صفها مورد مطالعه و بررسی کامل قرار گرفته است. فصل حاضر یک ساختمان دادهٔ غیرخطی موسوم به **درخت** را تعریف می‌کند. این ساختمان اساساً برای نمایش داده‌هایی که شامل رابطهٔ سلسله مراتبی بین عناصر مانند رکوردها، درختهای خانوادگی و جدول فهرست مطالب کتاب است بکار می‌رود.

نخست یک نوع خاص از درخت موسوم به **درخت دودویی** را بررسی می‌کنیم که می‌توان آن را به‌سادگی در کامپیوتر نگهداری یا ذخیره کنیم. هرچند چنین درختی بنظر خیلی محدود می‌آید اما اندکی دیرتر در همین فصل خواهید دید درختهای عمومی‌تر را می‌توان به صورت درختهای دودویی مورد توجه و بررسی قرار داد.

۲ـ۷ درختهای دودویی

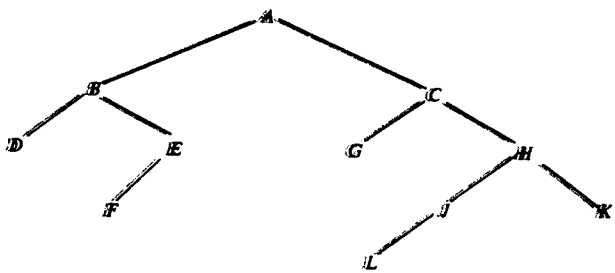
یک درخت دودویی T به صورت مجموعه‌ای متناهی از عناصر بنام **گره‌ها** تعریف می‌شود به‌گونه‌ای که :

(الف) T خالی است که به آن **درخت پوچ** یا **درخت تهی** می‌گویند یا

(ب) T شامل یک گره خاص R بنام ریشهٔ T است و سایر گره‌های T تشکیل یک زوج مرتب از درختهای دودویی مجزا T_1 و T_2 را می‌دهند.

اگر T شامل ریشهٔ R باشد، آنگاه دو درخت T_1 و T_2 به‌ترتیب زیردرختهای چپ و راست R نامیده می‌شوند. اگر T_1 غیرتهی باشد، آنگاه ریشهٔ آن را گرهٔ‌بعدی چپ R می‌گویند. به همین ترتیب اگر T_2 غیرتهی باشد آنگاه ریشهٔ آن را گرهٔ‌بعدی راست R می‌گویند.

اغلب اوقات یک درخت دودویی را به وسیلهٔ یک نمودار نمایش می‌دهند. به‌طور مشخص، نمودار شکل ۱۱ـ۷ زیر یک درخت دودویی را نشان می‌دهد.



شکل ۱۱ـ۷

(ii) T از ۱۱ گره تشکیل شده است که با حروف A تا L بجز I نمایش داده شده است.

(iii) ریشهٔ T گرهٔ A است که در بالای نمودار است. (iiii) یک خط از گرهٔ N که به طرف پائین و متمایل به چپ است مبین گرهٔ‌بعدی چپ N است و خط دیگر از گرهٔ N که به طرف پائین و متمایل به راست است مبین گرهٔ‌بعدی راست N است. ملاحظه می‌کنید که:

(الف) B یک گرهٔ‌بعدی چپ و C یک گرهٔ‌بعدی راست گرهٔ A است.

(ب) زیردرخت چپ ریشهٔ A متشکل از گره‌های B، D، E و F است و زیردرخت راست A متشکل از گره‌های C، G، H، J، K و L است.

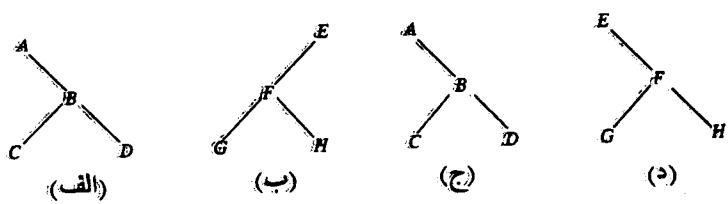
هر گرهٔ N در یک درخت دودویی T دارای 0، 1 یا 2 گرهٔ‌بعدی است. گره‌های A، B، C و H دو گرهٔ‌بعدی دارند و گره‌های E و J تنها یک گرهٔ‌بعدی دارند و گره‌های D، F، G، L و K هیچ گرهٔ‌بعدی ندارد. گره‌هایی که هیچ گرهٔ‌بعدی ندارند گره‌های انتهایی نام دارند.

تعریف بالا از درخت دودویی T بازگشتی است چون T برحسب زیردرختهای دودویی T_1 و T_2 تعریف می‌شود. به‌ویژه، معنی آن این است که هر گرهٔ N از T، یک زیردرخت چپ و راست دارد. علاوه براین، اگر N یک گرهٔ انتهایی باشد آنگاه زیردرختهای چپ و راست آن هر دو خالی هستند.

درختهای دودویی T و T' را مشابه یا متشابه می‌گویند اگر دارای یک ساختمان یا ساختار باشند. به عبارت دیگر اگر این درختها دارای یک شکل باشند. درختها را کپی هم گویند اگر این درختها متشابه بوده و محتوای گرههای متناظرِ آنها یکسان باشد.

مثال ۱ـ۷

چهار درخت دودویی شکل ۲ـ۷ را درنظر بگیرید. سه درخت (الف)، (ج) و (د) مشابه هستند. به‌ویژه، درختهای (الف) و (ج) کپی هم هستند چون دارای داده‌های مساوی و یکسان در گرههای متناظر هستند، درخت (ب) نه مشابهٔ درخت (د) است نه کپی آن، زیرا ما در یک درخت دودویی بین یک گرهٔ‌بعدی چپ و یک گرهٔ‌بعدی راست حتی وقتی تنها یک گرهٔ‌بعدی وجود داشته باشد تفاوت و تمایز قائل هستیم.



(الف) (ب) (ج) (د)

شکل ۲ـ۷

مثال ۲ـ۷ عبارتهای جبری

یک عبارت جبری E نظیر عبارت زیر را که تنها حاوی عملیات دوتایی است، درنظر بگیرید:

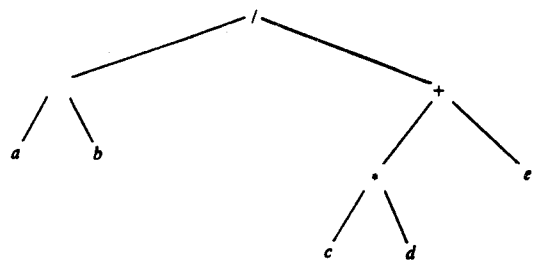
$$E = ((a - b) / ((c * d) + e))$$

E را می‌توان به وسیلهٔ درخت دودویی T به صورتی که در شکل ۳ـ۷ آمده است نمایش داد. به عبارت دیگر هر متغیر یا ثابت در E به صورت گرهٔ "داخلی" در T ظاهر شده است که زیر درختهای چپ و راست آن متناظر با عملوندهای عملگر هستند. برای مثال:

(الف) در آن عبارت E، عملوندهای + عبارتند از d * c و e

(ب) در درخت T، زیردرختهای گرهٔ + متناظر با زیرعبارتهای d * c و e هستند.

واضح است که هر عبارت جبری متناظر با یک درخت یکتا و منحصر بفرد است و برعکس.



$E = (a - b)/((c * d) + e)$. شکل ۳ـ ۷

چند اصطلاح

اصطلاحاتی که روابط خانوادگی را توصیف می‌کنند اغلب برای توصیف روابط بین گره‌ها در یک درخت T نیز بکار گرفته می‌شوند. به‌طور مشخص، فرض کنید N یک گرهٔ درخت T با گرهٔ‌بعدی چپ S_1 و گرهٔ‌بعدی راست S_2 باشد. آنگاه N را **پدر** S_1 می‌گویند. به‌طور مشابه S_1 **بچهٔ** (پسر) چپ N و S_2 را **بچهٔ** (پسر) **راست** N می‌گویند. علاوه براین S_1 و S_2 را **هم‌ردیف** یا **برادر** گویند. هر گرهٔ N در یک درخت دودویی T به‌استثنای ریشه، یک پدر منحصر بفرد دارند که **ریشهٔ قبلی** N می‌باشد.

اصطلاحات نسل و جَدْ دارای معنی متداول خودشان هستند. به عبارت دیگر گرهٔ L یک نسل گرهٔ N است و N یک جد گرهٔ L است اگر یک دنباله از بچه‌ها از N به L وجود داشته باشد. به‌ویژه، بسته به اینکه L به زیردرخت چپ تعلق داشته باشد یا زیردرخت راست. L یک **نسل چپ** یا **راست** گرهٔ N است.

اصطلاحات متداول در نظریهٔ گراف و علم باغبانی نیز، با یک درخت دودویی T بکار می‌رود. بطور مشخص، خطی که از گرهٔ N درخت T به یک گرهٔ‌بعدی رسم می‌شود یک **یال** و دنباله‌ای از یالهای متوالی یک **مسیر** نامیده می‌شود. یک گرهٔ انتهایی یک **برگ** نامیده می‌شود و مسیری که به یک برگ ختم می‌شود یک **شاخه** نام دارد.

به هر گره در یک درخت دودویی T یک **عدد سطح** به شرح زیر نسبت داده می‌شود. به ریشهٔ R درخت T **عدد سطح 0** نسبت داده می‌شود و به هر گرهٔ دیگر عدد سطحی نسبت داده می‌شود که یک واحد بیشتر از عدد سطح پدرش است. علاوه بر این، به آن دسته از گره‌ها که عدد **سطح** مساوی و یکسان دارند **متعلق به یک نسل** گویند.

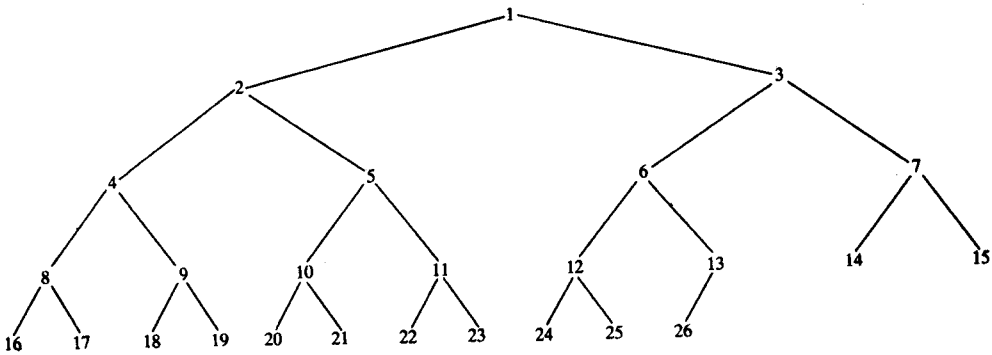
عمق یا **ارتفاع** درخت T حداکثر تعداد گره‌های یک شاخهٔ T است، ثابت می‌شود که عمق یک درخت یک واحد بیشتر از بزرگترین عدد سطح درخت T است. درخت T در شکل ۱ـ ۷ **عمق 5** دارد.

درختهای دودویی T و T′ **مشابه** هستند اگر دارای ساختمان یکسان باشند به بیان دیگر شکل

مساوی داشته باشند. درختها را **کپی هم** می‌گویند اگر مشابه بوده و محتوای گره‌های متناظر آنها یکی باشد.

درختهای دودویی کامل

درخت دودویی دلخواه **T** را درنظر بگیرید. هرگرهٔ **T** حداکثر می‌تواند **دو بچه** داشته باشد. بنابراین، می‌توان نشان داد سطح **r** از درخت **T** حداکثر می‌تواند 2^r گره داشته باشد. درخت **T** را **کامل** گویند اگر تمام سطح‌های آن بجز احتمالاً آخرین سطح، حداکثر تعداد گرهٔ ممکن را داشته باشد همچنین اگر تمام گره‌های آخرین سطح تا حد ممکن در سمت چپ و در دورترین مکان آن باشد. بنابراین برای یک درخت منحصر بفرد T_n دقیقاً **n** گره وجود دارد (البته، بدون درنظر گرفتن محتوای گره‌ها). درخت کامل T_{26} با **26** گره در شکل ۴ـ ۷ رسم شده است.



شکل ۴ـ ۷ درخت کامل T_{26}

گره‌های درخت دودویی کامل T_{26} شکل ۴ـ ۷ با هدف مشخص،با اعداد صحیح **1 ,2 ,3 , ...,26** از چپ به راست، نسل به نسل شماره‌گذاری شده است. با این نحوهٔ شماره‌گذاری، می‌توان به‌سادگی، بچه‌ها و پدر هر گرهٔ **K** را در یک درخت کامل T_n بدست آورد. بطور مشخص بچه‌های چپ و راست گرهٔ **K** به ترتیب $2 * K$ و $2 * K + 1$ هستند و پدر آن گرهٔ $\lfloor k/2 \rfloor$ است. برای مثال بچهٔ گرهٔ **9** گره‌های **18** و **19** هستند و پدرشان گرهٔ $\lfloor 9 / 2 \rfloor = 4$ است. عمق d_n درخت کامل T_n با **n** گره از رابطهٔ

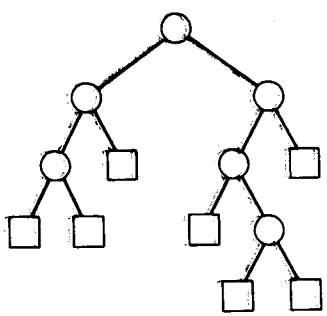
$$D_n = \lfloor \log_2 n + 1 \rfloor$$

بدست می‌آید. عدد بالا، عدد نسبتاً کوچکی است، برای مثال، اگر درخت کامل T_n دارای $n = 1000\,000$ گره باشد، آنگاه عمق آن $D_n = 21$ است.

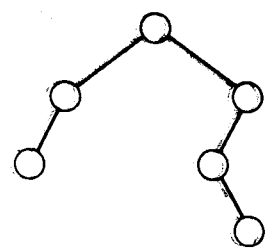
درختهای دودویی گسترش‌یافته : ۲ـ درخت

یک درخت دودویی T را یک ۲ـ درخت یا یک درخت دودویی گسترش‌یافته گویند اگر هر گرهٔ N، 0 یا 2 بچه داشته باشد. درچنین حالتی، گره‌هایی که دو بچه دارند گره‌های داخلی و گره‌هایی که 0 بچه دارند گره‌های خارجی نامیده می‌شوند. گاهی اوقات گره‌ها را در نمودار به صورت زیر ازهم تمیز می‌دهند: برای نمایش گره‌های داخلی از دایره و برای نمایش گره‌های خارجی از مربع استفاده می‌کنند.

اصطلاح "درخت دودویی کامل" ازعمل زیرمنشأ گرفته است. یک درخت دودویی دلخواه T نظیر درخت شکل ۵ـ ۷ (الف) رادرنظر بگیرید. آنگاه T را می‌توان با جانشین‌سازی هر زیردرخت خالی توسط یک گرهٔ جدید، مانند نمودار شکل ۵ـ ۷ (ب) به یک ۲ـ درخت تبدیل کرد. ملاحظه می‌کنید که درخت جدید، واقعاً یک ۲ـ درخت است. علاوه بر این، گره‌های درخت اصلی T اکنون گره‌های داخلی در درخت گسترش یافته هستند و گره‌های جدید گره‌های خارجی در درخت گسترش یافته هستند.



(الف) درخت دودویی T



(ب) ۲ـ درخت گسترش‌یافته

شکل ۵ـ ۷ تبدیل یک درخت دودویی T به یک ۲ـ درخت

یک مثال مهم و قابل توجه از یک ۲ـ درخت، درخت T متناظر با یک عبارت جبری دلخواه E است که تنها از عملیات دوتایی استفاده می‌کند. همانگونه که در شکل ۳ـ ۷ نشان داده شده است، متغیرهایی که داخل E هستند به صورت گره‌های خارجی در شکل ظاهر می‌شوند و عملیات داخل E به صورت گره‌های خارجی ظاهر خواهند شد.

۳ـ۷ نمایش درختهای دودویی در حافظه

فرض کنید T یک درخت دودویی باشد. این بخش دو روش نمایش T را در حافظه مورد بحث و بررسی قرار می‌دهد. روش اول و معمول، نمایش پیوندی درخت T است و مشابه روش لیست‌های

پیوندی است که در حافظه نمایش داده می‌شود. روش دوم که تنها از یک آرایه استفاده می‌کند نمایش ترتیبی Sequential درخت T است. اصلی‌ترین موضوع در هر نمایش درخت T، آن است که به ریشهٔ R درخت T دسترسی مستقیم داشته باشیم و یا معلوم‌بودن هر گرهٔ N از T، باید بتوان به هر بچهٔ N دسترسی پیدا کرد.

نمایش پیوندی درختهای دودویی

درخت دودویی T را درنظر بگیرید. درخت T در حافظه بوسیلهٔ یک نمایش پیوندی نگهداری می‌شود که از سه آرایهٔ موازی INFO، LEFT و RIGHT و یک متغیر اشاره‌گر به شرح زیر استفاده می‌کند، مگر آن که خلاف آن به صورت صریح یا ضمنی بیان شود.

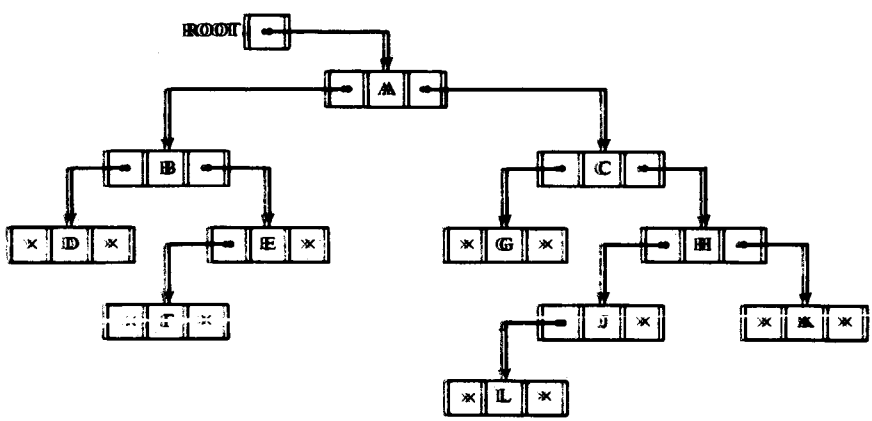
قبل از هر چیز، متذکر می‌شویم که هر گرهٔ N متناظر با یک مکان K است به گونه‌ای که:

((۱)) INFO[K] حاوی دادهٔ گرهٔ N است.

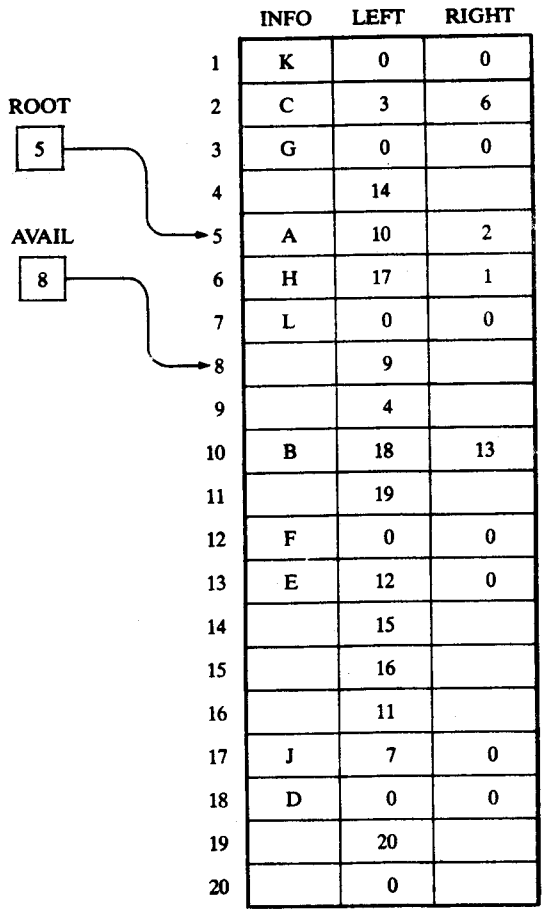
((۲)) LEFT[K] حاوی مکان بچه چپ گرهٔ N است.

((۳)) RIGHT[K] حاوی مکان بچهٔ راست گرهٔ N است.

علاوه براین، ROOT حاوی مکان ریشهٔ R درخت T است. اگر هر زیردرخت ـ ـاشد، آنگاه اشاره‌گر متناظر آن دارای مقدار پوچ NULL خواهد بود. اگر خود درخت T خالی ـ نگاه ROOT دارای مقدار پوچ NULL خواهد بود.



شکل ۶ـ۷



	INFO	LEFT	RIGHT
1	K	0	0
2	C	3	6
3	G	0	0
4		14	
5	A	10	2
6	H	17	1
7	L	0	0
8		9	
9		4	
10	B	18	13
11		19	
12	F	0	0
13	E	12	0
14		15	
15		16	
16		11	
17	J	7	0
18	D	0	0
19		20	
20		0	

شکل ۷ـ ۷

تذکر ۱ : اکثر مثالهای ما یک عنصر اطلاعاتی را در هر گره N از یک درخت دودویی T نشان خواهند داد. در کاربردهای عملی، ممکن است در گرهٔ N یک رکورد ذخیره شود. به بیان دیگر، **INFO** واقعاً ممکن است یک آرایهٔ خطی از رکوردها یا مجموعه‌ای از آرایه‌های موازی باشد.

تذکر ۲ : چون ممکن است گره‌ها به درختهای دودویی اضافه شوند یا از درختهای دودویی حذف شوند ما صراحتاً فرض می‌کنیم که مکانهای خالی در آرایه‌های **INFO** ، **LEFT** و **RIGHT** تشکیل یک لیست پیوندی با اشاره‌گر **AVAIL** را می‌دهند که در رابطه با لیستهای پیوندی در فصل ۵ مورد بحث و بررسی قرار گرفت. معمولاً فرض می‌کنند آرایهٔ **LEFT** دارای اشاره‌گرهایی برای لیست **AVAIL** است.

تذکر ۳: هر آدرس غیرمجازی که ممکن است برای اشاره‌گر پوچ **NULL** انتخاب شود با **NULL** نمایش داده می‌شود. در کاربردهای عملی، برای **NULL** از **0** یا یک عدد منفی استفاده می‌شود.

مثال ۳ـ ۷

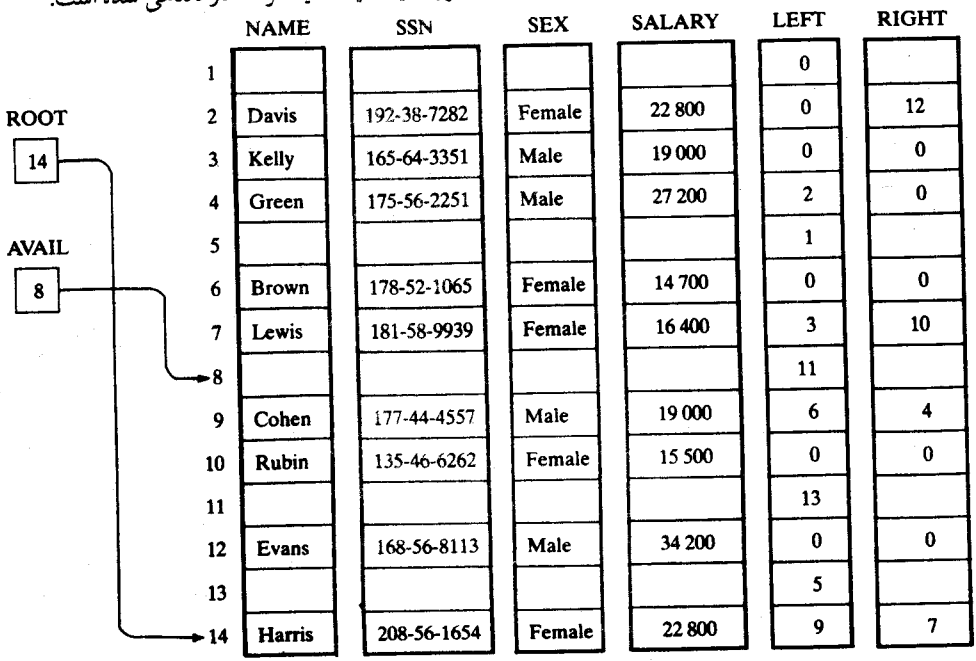
درخت دودویی **T** شکل ۱ـ ۷ را درنظر بگیرید. نمودار نمایش پیوندی **T** در شکل ۶ـ ۷ نشان داده شده است. ملاحظه می‌کنید که هر گره با سه فیلد آن و زیردرختهای خالی با استفاده از **X** برای ورودی‌های پوچ به تصویر درآمده است. شکل ۷ـ ۷ چگونگی نمایش این لیست پیوندی را در حافظه نشان می‌دهد. انتخاب **20** عنصر برای آرایه‌ها، دلخواه و اختیاری است. ملاحظه می‌کنید که لیست **AVAIL** با استفاده از آرایه **LEFT** به صورت لیست یکطرفه نگهداری می‌شود.

مثال ۴ـ ۷

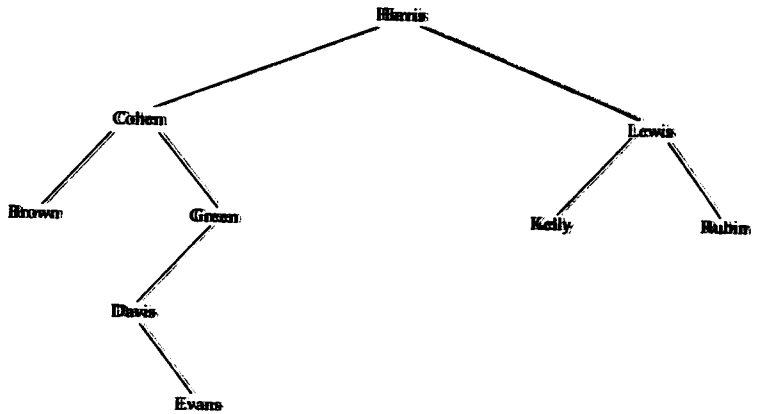
فرض کنید فایل پرسنلی یک شرکت کوچک با **9** کارمند از فیلدهای زیر تشکیل شده است:

Name, Social Security Number, Sex, Monthly Salary

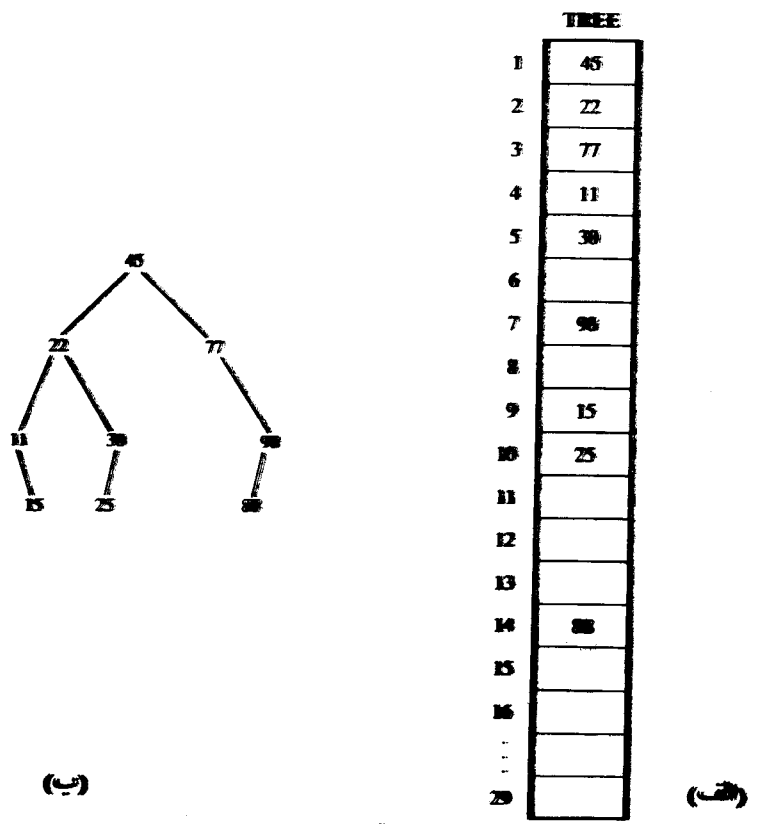
شکل ۸ـ ۷ چگونگی نگهداری فایل را در حافظه به صورت یک درخت دودویی نشان می‌دهد. این ساختمان داده را با شکل ۱۲ـ ۵ مقایسه کنید که در آن دقیقاً همین داده‌ها به صورت یک لیست یکطرفه سازماندهی شده است.



شکل ۸ـ ۷



شکل ۹_۷



شکل ۱۰_۷

فرض کنید بخواهیم نمودار یک درخت را رسم کنیم که متناظر با درخت دودویی شکل ۸ـ ۷ است. جهت سهولت در نمادگذاری، گره‌های نمودار درخت را تنها با مقادیر کلیدی NAME شماره‌گذاری می‌کنیم. درخت را به شرح زیر می‌سازیم :

(الف) مقدار ROOT = 14 بیانگر آن است که Harris ریشه درخت است.

(ب) LEFT[14] = 9 بیانگر آن است که Cohen بچهٔ راست Harris و RIGHT[14] = 7 بیانگر آن است که Lewis بچهٔ راست Harris است.

برای هر گرهٔ جدید داخل نمودار، با تکرار مرحلهٔ (ب) شکل ۹ـ ۷ بدست می‌آید.

نمایش ترتیبی درختهای دودویی

فرض کنید T یک درخت دودویی باشد که کامل یا تقریباً کامل است. آنگاه روش کارایی برای نگهداری T در حافظه وجود دارد که نمایش ترتیبی T نام دارد. این نمایش تنها از یک آرایهٔ خطی TREE به صورت زیر استفاده می‌کند.

(الف) ریشهٔ R درخت T در TREE[1] ذخیره می‌شود.

(ب) اگر گرهٔ N، TREE[K] را اشغال کند آنگاه بچهٔ چپ آن در TREE[2 * K] و بچهٔ راست آن در TREE[2 * K + 1] ذخیره می‌شود. مجدداً در NULL برای بیان یک زیردرخت خالی مورد استفاده قرار می‌گیرد. بعلاوه TREE[1] = NULL بیانگر آن است که درخت خالی است.

نمایش ترتیبی درخت دودویی T در شکل ۱۰ـ ۷ (الف) در شکل ۱۰ـ ۷ (ب) نشان داده شده است. ملاحظه می‌کنید که حتی اگر T تنها 9 گره داشته باشد در آرایهٔ TREE به 14 خانهٔ احتیاج است. در واقع، اگر درودی پوچ برای گره‌های انتهایی اختیار شود آنگاه واقعاً برای گرهٔ بعدی TREE[29] به TREE[14] احتیاج خواهیم داشت. به بیان کلی‌تر، نمایش ترتیبی یک درخت به عمق d، نیازمند آرایه‌ای با تقریباً 2^{d+1} عنصر خواهد بود. برطبق آن، این نمایش ترتیبی معمولاً از کارایی لازم برخوردار نیست همانگونه که در بالا بیان شده است، درخت دودویی T کامل یا تقریباً کامل است. برای مثال، درخت T در شکل ۱ـ ۷ دارای 11 گره و عمق 5 است. معنی آن، این است که نیازمند یک آرایه با تقریباً $2^6 = 64$ عنصر است.

۴ـ ۷ پیمایش درختهای دودویی

سه روش استاندارد برای پیمایش یک درخت دودویی T با ریشهٔ R وجود دارد. این سه الگوریتم که PreOrder، InOrder و PostOrder نامیده می‌شوند به شرح زیر می‌باشند :

روش PreOrder :

(۱) ریشهٔ R را پردازش کنید.

(۲) زیردرخت چپ R را به روش PreOrder پیمایش کنید.

(۳) زیردرخت راست R را به روش PreOrder پیمایش کنید.

روش Inorder :

(۱) زیردرخت چپ را به روش InOrder پیمایش کنید.

(۲) ریشهٔ R را پردازش کنید.

(۳) زیردرخت راست را به روش InOrder پیمایش کنید.

روش PostOrder :

(۱) زیردرخت چپ R را به روش PostOrder پیمایش کنید.

(۲) زیردرخت راست R را به روش PostOrder پیمایش کنید.

(۳) ریشهٔ R را پردازش کنید.

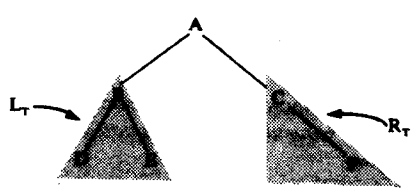
ملاحظه می‌کنید هر الگوریتم دارای همین سه مرحله است و زیردرخت چپ R همیشه قبل از زیردرخت راست پیمایش می‌شود. تفاوت بین این سه الگوریتم، در زمانی است که در آن ریشهٔ R پردازش می‌شود. بطور مشخص، در الگوریتم دارای "Pre" ، ریشهٔ R قبل از پیمایش زیر درخت‌ها پردازش می‌شود. در الگوریتم دارای "In" ، ریشهٔ R مابین پیمایش زیردرختها پردازش می‌شود و در الگوریتم دارای "Post" ریشهٔ R بعد از پیمایش زیردرختها پردازش می‌شود.

گاهی اوقات این سه الگوریتم را به ترتیب **پیمایشِ گره ـ چپ ـ راست (NLR) پیمایشِ چپ ـ گره ـ راست (LNR)** و پیمایشِ چپ ـ راست ـ گره **(LRN)** می‌گویند.

ملاحظه می‌کنید که هر یک از الگوریتم‌های پیمایش بالا به صورت بازگشتی تعریف می‌شود چون الگوریتم‌ها حاوی پیمایش زیردرختها با ترتیب داده شده **Given Order** است. بنابراین، انتظار داریم که هنگام پیاده‌سازی الگوریتم‌ها در کامپیوتر از یک پشته استفاده شود.

مثال ۵ـ ۷

درخت دودویی شکل ۱۱ـ ۷ را درنظر بگیرید.



شکل ۱۱ـ ۷

ملاحظه می‌کنید که A ریشهٔ این‌درخت است و زیردرخت چپ آن L_T از گره‌های B ، D و E و زیردرخت راست آن R_T از گره‌های C و F تشکیل شده است.

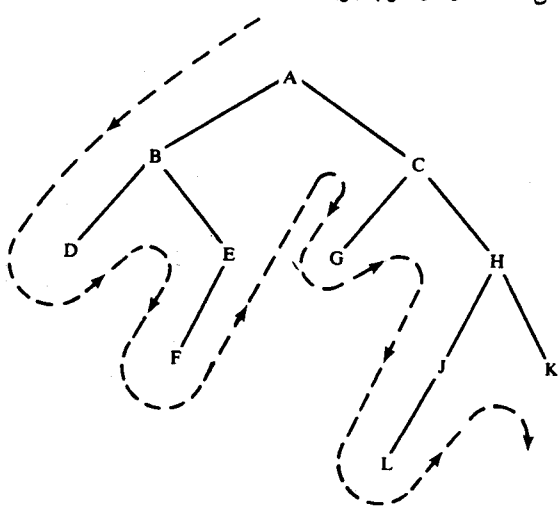
(الف) پیمایش PreOrder درخت T ، A را پردازش می‌کند، L_T را پیمایش می‌کند و R_T را پیمایش می‌کند. با وجود این، پیمایش PreOrder زیردرخت L_T ریشهٔ B را پردازش می‌کند و آنگاه D و E را پردازش می‌کند و پیمایش PreOrder زیردرخت R_T ریشهٔ C را پردازش می‌کند و آنگاه F را پردازش می‌کند. از این‌رو ABDECF پیمایش PreOrder درخت T است.

(ب) پیمایش InOrder درخت T زیردرخت L_T را پردازش می‌کند، A را پردازش می‌کند و R_T را پیمایش می‌کند. باوجود این، پیمایش InOrder از زیردرخت L_T ، D ، E و آنگاه E را پردازش می‌کنند و پیمایش InOrder زیردرخت R_T ، C و آنگاه F را پردازش می‌کند. از این‌رو DBEACF پیمایش InOrder درخت T است.

(ج) پیمایش PostOrder درخت T ، زیردرخت L_T را پیمایش می‌کند، R_T را پیمایش می‌کند و A را پردازش می‌کند. پیمایش PostOrder زیردرخت L_T ، D ، E و آنگاه B را پردازش می‌کند و پیمایش PostOrder زیردرخت R_T ، F و آنگاه C را پردازش می‌کند. بنابراین، DEBFCA پیمایش PostOrder درخت T است.

مثال ۶ـ ۷

درخت دودویی شکل ۱۲ـ ۷ را درنظر بگیرید.



شکل ۱۲ـ ۷

پیمایش **PreOrder** درخت **T** ، **ABDEFCGHJLK** است. این ترتیب، همان ترتیبی است که با جستجو و خواندن درخت از چپ، به صورتی که به وسیلهٔ مسیر شکل ۱۲ـ ۷ مشخص شده است به دست می‌آید یعنی عمل پیمایش را به طرف پائین در سمت چپ‌ترین شاخه انجام می‌دهیم تا گرهٔ انتهایی را ملاقات کنیم، آنگاه به عقب برمی‌گردیم تا به شاخهٔ بعدی برسیم و الی آخر. در پیمایش **PreOrder** سمت راست‌ترین گرهٔ انتهایی، گرهٔ **K** ، آخرین گره‌ای است که خوانده می‌شود. ملاحظه می‌کنید که زیردرخت چپ ریشهٔ **A** قبل از زیردرخت راست پیمایش می‌شود و هر دو بعد از **A** پیمایش می‌شوند. این مطلب برای هر گرهٔ دیگر زیردرختها برقرار است که خاصیت عمدهٔ یک پیمایش **PreOrder** است.

خواننده با بررسی می‌تواند تحقیق کند که دو روش دیگر پیمایش درخت دودویی شکل ۱۲ـ ۷ به صورت زیر هستند :

(Inorder) D B F E A G C L J H K
(Postorder) D F E B G L J K H C A

ملاحظه می‌کنید که گره‌های انتهایی **D** ، **F** ، **G** ، **L** و **K** با همان ترتیب، در تمام سه پیماش از چپ به راست پیمایش می‌شوند. تأکید می‌کنیم که این مطلب برای هر درخت دودویی **T** برقرار است.

مثال ۷ـ ۷

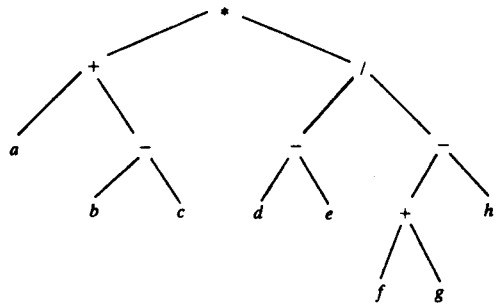
فرض کنید **E** نمایش عبارت جبری زیر باشد :

$$[a + (b - c)] * [(d - e)/(f + g - h)]$$

درخت دودویی **T** مربوط به آن در شکل ۱۳ـ ۷ نشان داده شده است. دانشجو با بررسی می‌تواند تحقیق کند که پیمایش **PreOrder** و **PostOrder** درخت **T** به شرح زیر است :

(Preorder) * + a − b c / − d e − + f g h
(Postorder) a b c − + d e − f g + h − / *

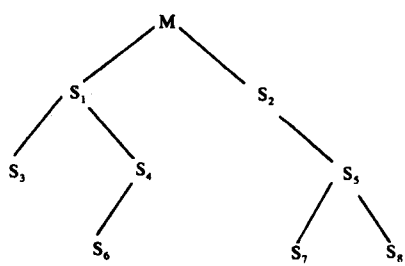
علاوه بر این دانشجو می‌تواند تحقیق کند که این ترتیبها دقیقاً متناظر با نماد پیشوندی و نماد لهستانی پسوندی **E** است که در بخش ۴ـ ۶ مورد بحث و بررسی قرار گرفت. تأکید می‌کنیم که برای هر عبارت جبری **E** این ترتیب برقرار است.



شکل ۱۳ـ ۷

مثال ۸ـ ۷

درخت دودویی **T** شکل ۱۴ـ ۷ را درنظر بگیرید :



شکل ۱۴ـ ۷

دانشجو می‌تواند تحقیق کند که نمایش **PostOrder** درخت **T** به شرح زیر است :

$S_3,\ S_6,\ S_4,\ S_1,\ S_7,\ S_8,\ S_5,\ S_2,\ M$

یک خاصیت اصلی این الگوریتم پیمایش، آن است که هر نسل از هر گره **N** قبل از گرهٔ **N** پردازش می‌شود. برای مثال، S_6 قبل از S_4، S_6 و S_4 قبل از S_1 پردازش می‌شود. بطور مشابه S_7 و S_8 قبل از S_5 و S_7 و S_8 و S_5 قبل از S_2 پردازش می‌شوند. علاوه براین، تمام گره‌های S_1، S_2، ..... ، S_8 قبل از ریشهٔ **M** پردازش می‌شوند.

توجه کنید : اگر درختی مانند دو مثال بالا تعداد نسبتاً کمی گره داشته باشد، دانشجو می‌تواند با بررسیِ سه پیمایش مختلف درخت دودویی **T** را پیاده‌سازی کند. هرگاه درخت **T** صدها یا هزارها گره داشته باشد پیاده‌سازی با بررسی، ممکن است امکان‌پذیر نباشد، به بیان دیگر، نیازمند روش منظم و اصولی