



دانشگاه هرمزگان

اصول

ساختمان داده‌ها

سیمور لیپ شوتز / مهندس حسین ابراهیم زاده قلزم

چاپ هشتم

قابل استفاده دانشجویان مهندسی کامپیوتر

سخت افزار، نرم افزار، ریاضی کاربردی

ناشر برگزیده سال ۱۳۷۷

حاوی ۴۵۷ مساله حل شده

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

اصول ساختمان داده‌ها

نوشته: سیمور لیپ شوتز

ترجمه: مهندس حسین ابراهیم‌زاده قلزم

Lipschutz, Seymour

لیپشوتز، سیمور

اصول ساختمان داده‌ها / نوشته سیمور لیپ شوتز؛ ترجمه حسین ابراهیم‌زاده قلزم -
 [بندرعباس] : دانشگاه هرمزگان، ۱۳۷۴.
 (۸)، ۱، ۵۰ ص، مصور، جدول، نمودار.

فهرستنويسي براساس اطلاعات فنيا (فهرستنويسي پيش از انتشار).

عنوان روی جلد: اصول ساختمان داده‌ها، قابل استفاده دانشجویان مهندسی کامپیوتر،
 سخت افزار، ریاضی کاربردی، حاوی ۴۵۷ مسئله حل شده.

Schaums outline of theory
 and problems of data structures.
 عنوان اصلی:

واژه‌نامه.

چاپ هشتم، ۱۳۸۲

ISBN: 964 - 6426 - 35 - 2: ۳۴۰۰۰

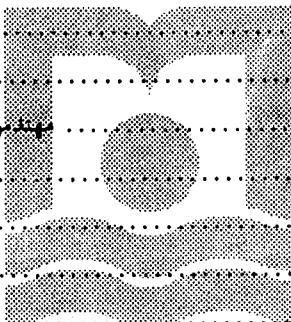
۱. ساختار داده‌ها. ۲. ساختار داده‌ها -- مسائل، تمرینها و غیره. الف. ابراهیم‌زاده قلزم،
 حسین، ۱۳۴۰ -، مترجم. ب. دانشگاه هرمزگان. ج. عنوان.

۰۰۵۷۳ Q۸۷۶/۹ س۲۱۹

۷۵-۱۰۵۸۱

۱۳۷۴

نام کتاب:	اصول ساختمان داده‌ها
ناشر:	دانشگاه هرمزگان
مؤلف:	سیمور لیپ شوتز
مترجم:	حسین ابراهیم‌زاده قلزم
لیتوگرافی:	کبریا
چاپ:	کبریا
نوبت چاپ:	هشتم / ۱۳۸۲
شمارگان:	۳۳۰۰
قیمت:	۳۴۰۰۰ ریال



ISBN: 964 - 6426 - 35 - 2

۹۶۴ - ۶۴۲۶ - ۳۵ - ۲

«حق چاپ محفوظ است»

بندرعباس - صندوق پستی ۳۹۹۵ - دانشگاه هرمزگان تلفن: (۰۷۶۱ - ۴۰۱۲۱ - ۰)

فروشگاه: تهران - خیابان انقلاب - مقابل دانشگاه تهران - پاساژ فروزنده - تلفن: ۶۹۵۰۵۵۳

مُهندسی کامپیوْتِر

سالهای است که به ترجمه و تألیف کتابهای ریاضی و کامپیوْتری اشتغال دارم که کاری است پررنج، با مشقت‌های فراوان. اما هیچ ترجمه و تالیف را شیرین‌تر و جذاب‌تر از اثر حاضر ندیده‌ام زیرا کامپیوْتر را علم پرفوسونی می‌دانم که معتقدم فسون آن در ساختمان داده‌ها متجلی است. ساختمان داده‌ها یکی از زیباترین مباحث و نیرومندترین ابزار در حل مسائل کامپیوْتری است. این درس شیفتگان بسیار در میان دانشجویان رشته کامپیوْتر دارد و نظر به اهمیت آن به عنوان درس رسمی دانشگاه‌های کشور پذیرفته شده است.

کتاب حاضر دربرگیرنده تمامی مطالب ساختمان داده‌های رشته‌های مهندسی کامپیوْتر - سخت‌افزار، نرم‌افزار و ریاضی کاربردی است. این کتاب در عین حال که به تئوری پرداخته اما عاری از لفاظی و توصیفهای ناضرور است. از کمتر نکته‌ای در آن چشم پوشیده شده است و آنچه در کتاب است، ساختمان داده‌ها است و بس. این کتاب را می‌توان به عنوان سرفصل چندین واحد درسی دانشجویان رشته‌های کامپیوْتر و ریاضی کاربردی مورد استفاده قرار داد و می‌تواند مرجعی مناسب و مطمئن برای درس‌های برنامه‌نویسی پیشرفت، ساختمان داده‌ها و الگوریتم‌ها، ساختمان و ریاضیات گستته، ذخیره و بازیابی اطلاعات و زبانهای برنامه‌نویسی باشد.

در ترجمه این کتاب برکسی منتی نمی‌گذارم و خویشن را سزاوار هیچ‌گونه پاداشی نمی‌دانم زیرا به ایران و مردم ایران مدیونم و اگر بتوانم از این رهگذر اندکی از دین معنوی خود بکاهم تا حدی به آرزوی خود رسیده‌ام.

حسین ابراهیم‌زاده قلزم
دانشگاه صنعتی امیرکبیر
(پلی‌تکنیک تهران)
تهران ۲۰ - مهر - ۱۳۷۴

پیشکش‌تار مولف

مطالعه و بررسی ساختمان داده‌ها جزء اصلی و اجباری برنامه هر دانشجوی لیسانس و فوق‌لیسانس علم کامپیوتر است. این کتاب که اساسی‌ترین مطالب و سرفصل‌های درس ساختمان داده‌ها را ارائه می‌دهد، می‌تواند به عنوان یک کتاب درسی برای یک درس رسمی ساختمان داده‌ها یا به عنوان یک کتاب مکمل برای تقریباً تمام کتب متعارف و موجود کامپیوتر در این زمینه مورد استفاده قرار گیرد.

فصل‌های کتاب عمده‌تاً براساس میزان پیچیدگی مطالب تنظیم شده‌اند. فصل ۱، معرفی و مژو رکلی بر مطالب تمام کتاب خواهد داشت و فصل ۲، زمینه ریاضی و نمادهای کامپیوتری را جهت ارائه و تجزیه و تحلیل الگوریتم‌ها فراهم می‌آورد. فصل ۳^۱ که در مورد تطبیق الگوست، فصل مستقلی است که به صورت حاشیه‌ای با متن ارتباط دارد و از این رو در مطالعه اول کتاب می‌توان این فصل را حذف کرد یا خواندن آن را تا زمان دیگر به تعویق انداخت. فصل‌های ۴ تا ۸ شامل مطالب اصلی درس ساختمان داده‌است. مخصوصاً این که در فصل ۴، آرایه‌ها *Arrays*، رکوردها *Records* مورد بررسی قرار می‌گیرد.

فصل ۵ راجع به لیستهای پیوندی *Linked Lists* است. فصل ۶ حاوی مطالبی در زمینه پشته‌ها *Stacks* و صفها *Queues* و همچنین زیربرنامه‌های بازگشته *Recursion* است، فصل ۷ در مورد درختهای دودویی *Binary Trees* و فصل ۸ در باره گرافها *Graphs* و کاربردهای آن است. اگرچه مرتب کردن *Sorting* و جستجوی اطلاعات *Searching* از ابتدا تا انتهای کتاب در داخل مجموعه‌ای از ساختمان داده مخصوصاً مانند جستجوی دودویی با آرایه‌های خطی، *QuickSort* با پشته‌ها و صفها، و *HeapSort* با درختهای دودویی، مورد بحث و بررسی قرار گرفته است اما در فصل ۹، که آخرین فصل کتاب رانیز تشکیل می‌دهد الگوریتم‌های اضافی و تکمیلی مرتب کردن اطلاعات و جستجوی آنها نظری ادغام و مرتب کردن *Merge - Sort* و درهم سازی *Hashing* معرفی و بحث و بررسی شده است.

الگوریتم‌ها به صورتی ارائه شده‌اند که مستقل از نوع کامپیوتر و زبان‌های برنامه‌نویسی است. علاوه

براین، این الگوریتم‌ها برای دستورهای کنترلی، عمدتاً با استفاده از قطعه برنامه‌های IF- THEN و REPEAT - WHILE نوشته شده‌اند و همچنین جهت سهولت در خواندن و درک و فهم ساده‌آنها، از الگوی تورفته و پله‌ای استفاده شده است. بنابراین، هر یک از الگوریتم‌های کتاب را تقریباً می‌توان به سادگی، به تمام زبانهای متعارف و موجود برنامه‌نویسی ترجمه کرد.

این کتاب با اتخاذ یک روش کاملاً ساده و مقدماتی و با ارائه مثال و نمودارهای متعدد، خوانندگان زیادی را بخود اختصاص خواهد داد و به ویژه به عنوان راهنمایی خودآموز، کاملاً مؤثر و مفید عمل خواهد کرد.

هر فصل از کتاب با بیان روشنی از تعاریف، اصول و عبارتهای توصیفی همراه است. بدنبال این مراحل، مجموعه‌های درجه‌بندی شده از مسایل حل شده و تکمیلی می‌آیند. مسائلهای حل شده، مطالب درسی را توضیح می‌دهند و نکات ظرفی را مطرح می‌سازند و در جهت تقویت آنها است در حالی که مسایل تکمیلی مروری کامل بر مطالب درسی این سرفصل‌ها دارند.

مايلم مراتب سپاسگزاری خود را از دوستان و همکاران بیشماری که به لحاظ ارائه پیشنهادات ارزنده‌شان و بازنگری انتقادی دست‌نوشته‌ها که وقت خود را مصروف آن نموده‌اند ابراز دارم. علاوه بر این مايلم از کارکنان نشر Mc Graw - Hill Schaum Jeffrey Mccartney به خاطر همکاری سودمندشان سپاسگزاری بنمایم.

در پایان از آقای دانلد. ا. کناث Donald. E. Knuth که متن جامع اولیه ساختمان داده‌ها را تدوین نموده‌اند و بدون شک تأثیر بسزایی در تألیف این کتاب داشته‌اند و همچنین موضوعات متعدد دیگر در این خصوص که از ناحیه ایشان صورت پذیرفته است صمیمانه تشکر می‌نمایم.

سیمور لیپ شوتز
دپارتمان کامپیوتر
دانشگاه تمپل

فهرست مطالب

صفحة	فصل ۱ معرفی و مرور سریع مطالب کتاب
۱	۱-۱ مقدمه
۱	۱-۲ اصطلاحات اصلی و سازماندهی ابتدایی داده‌ها
۳	۱-۳ ساختمان داده‌ها
۱۲	۱-۴ عملیات بر روی ساختمان داده‌ها
۱۴	۱-۵ الگوریتم‌ها: پیچیدگی، توازن بین زمان اجرا و حافظه
۱۷	۱-۶ مسئله‌های حل شده
۲۷	فصل ۲ مفاهیم مقدماتی
۲۸	۲-۱ مقدمه
۳۳	۲-۲ نمادگذاری ریاضی و تابعهای کامپیوتروی
۳۶	۲-۳ نمایش الگوریتمی
۴۳	۲-۴ دستورهای کنترلی
۴۷	۲-۵ پیچیدگی الگوریتم‌ها
۵۰	۲-۶ زیرالگوریتم‌ها
۵۲	۲-۷ متغیرها و انواع داده‌ها
۶۰	۲-۸ مسئله‌های حل شده
۶۰	۲-۹ مسئله‌های تکمیلی

فصل ۳ پردازش رشته‌ها

۳-۱ مقدمه

۲-۲ اصطلاحات پایه‌ای

۲-۳ ذخیره رشته‌ها

۲-۴ نوع داده کاراکتری

۲-۵ عملیات بر روی رشته‌ها

۲-۶ پردازش رشته

۲-۷ الگوریتم‌های تطبیق الگو

۲-۸ مسئله‌های حل شده

۲-۹ مسئله‌های تکمیلی

۳-۱ فصل ۴ آرایه‌ها، رکوردها و اشاره‌گرها

۴-۱ مقدمه

۴-۲ آرایه‌های خطی

۴-۳ نمایش آرایه‌های خطی در حافظه

۴-۴ پیمایش آرایه‌های خطی

۴-۵ اضافه کردن و حذف کردن عنصر

۴-۶ مرتب کردن، مرتب کردن جابجایی

۴-۷ جستجو کردن، جستجوی خطی

۴-۸ جستجوی دودویی

۴-۹ آرایه‌های چند بعدی

۴-۱۰ اشاره‌گرها، آرایه‌های نوع اشاره‌گر

۴-۱۱ رکوردها، ساختارهای رکوردی

۴-۱۲ نمایش رکوردها در حافظه، آرایه‌های موازی

۴-۱۳ ماتریسها

۴-۱۴ ماتریس‌های خلوت

۴-۱۵ مسئله‌های حل شده

۴-۱۶ مسئله‌های تکمیلی

۵-۱ فصل ۵ لیستهای پیوندی

۵-۱ مقدمه

۵-۲ لیستهای پیوندی

۵-۳ نمایش لیستهای پیوندی در حافظه

۵-۴ پیمایش یک لیست پیوندی

۶۴

۶۵

۷۱

۷۳

۷۶

۸۲

۸۹

۹۹

۱۰۳

۱۰۴

۱۰۷

۱۰۹

۱۱۰

۱۱۳

۱۱۷

۱۲۱

۱۲۵

۱۲۳

۱۲۹

۱۴۲

۱۴۵

۱۴۹

۱۵۱

۱۶۶

۱۷۳

۱۷۴

۱۷۶

۱۸۱

۱۸۳	۵-۵ جستجو در یک لیست پیوندی
۱۸۶	۵-۶ تخصیص حافظه، جمع آوری حافظه بلااستفاده
۱۹۱	۵-۷ اضافه کردن گره به یک لیست پیوندی
۲۰۰	۵-۸ حذف گره از یک لیست پیوندی
۲۰۸	۵-۹ لیستهای پیوندی دارای سریست
۲۱۴	۵-۱۰ لیستهای دو طرفه
۲۲۰	۵-۱۱ مسأله‌های حل شده
۲۲۰	۵-۱۲ مسأله‌های تکمیلی

فصل ۶ پشته‌ها، صفحه‌ها، زیربرنامه‌های بازگشته

۲۳۹	۶-۱ مقدمه
۲۴۰	۶-۲ پشته‌ها
۲۴۳	۶-۳ نمایش پشته‌ها به کمک آرایه
۲۴۶	۶-۴ عبارتهای محاسباتی، نمادگذاری لهستانی
۲۵۲	۶-۵ QuickSort، یک کاربرد از پشته‌ها
۲۵۸	۶-۶ زیربرنامه بازگشته
۲۶۴	۶-۷ برجهای هانوی
۲۶۸	۶-۸ پیاده‌سازی زیربرنامه‌های بازگشته به وسیله پشته‌ها
۲۷۶	۶-۹ صفحه
۲۸۲	۶-۱۰ صفهای دو سرمه
۲۸۳	۶-۱۱ صفهای اولویت
۲۸۸	۶-۱۲ مسأله‌های حل شده
۳۰۷	۶-۱۳ مسأله‌های تکمیلی

فصل ۷ درختها

۳۱۵	۷-۱ مقدمه
۳۱۵	۷-۲ درختهای دودویی
۳۲۰	۷-۳ نمایش درختهای دودویی در حافظه
۳۲۵	۷-۴ پیمایش درختهای دودویی
۳۳۰	۷-۵ الگوریتم‌های پیمایش به کمک پشته‌ها
۳۳۸	۷-۶ سرگرهای، گرهای نخ‌کشی شده
۳۴۳	۷-۷ درختهای جستجوی دودویی

۳۴۵	۷-۸ جستجو و وارد کردن یک عنصر در درختهای جستجوی دودویی
۳۵۱	۷-۹ حذف یک عنصر از یک درخت جستجوی دودویی
۳۵۷	HeapSort ، Heap ۷-۱۰
۳۶۵	۷-۱۱ طول مسیر، الگوریتم هافمن
۳۷۳	۷-۱۲ درختهای عمومی
۳۷۸	۷-۱۳ مسئله‌های حل شده
۳۹۲	۷-۱۴ مسئله‌های تکمیلی

فصل ۸ گرافها و کاربردهای آن

۴۰۳	۸-۱ مقدمه
۴۰۳	۸-۲ چند اصطلاح نظریه گراف
۴۰۸	۸-۳ نمایش ترتیبی گرافها، ماتریس مجاورت، ماتریس مسیر
۴۱۲	۸-۴ الگوریتم وارشال، کوتاهترین مسیر
۴۱۶	۸-۵ نمایش یک گراف با استفاده از لیست پیوندی
۴۲۰	۸-۶ عملیات بر روی گرافها
۴۲۷	۸-۷ پیمایش یک گراف
۴۳۲	۸-۸ مجموعه‌های جزوآ مرتب، مرتب سازی موضعی
۴۳۸	۸-۹ مسئله‌های حل شده
۴۵۱	۸-۱۰ مسئله‌های تکمیلی

فصل ۹ مرتب کردن و جستجوی اطلاعات

۴۶۱	۹-۱ مقدمه
۴۶۲	۹-۲ مرتب کردن
۴۶۷	۹-۳ مرتب کرن درجی
۴۷۰	۹-۴ مرتب کردن انتخابی
۴۷۲	۹-۵ ادغام کردن
۴۷۶	۹-۶ ادغام و مرتب کردن
۴۷۹	۹-۷ مرتب کردن مبنایی
۴۸۲	۹-۸ جستجوی اطلاعات و اصلاح داده‌ها
۴۸۳	۹-۹ درهم سازی
۴۹۱	۹-۱۰ مسئله‌های تکمیلی
۴۹۵	واژه‌نامه ریاضی و کامپیوتر

فصل ۱

معرفی و معرفه سریع مطالب کتاب

۱-۱ مقدمه

این فصل موضوع ساختمان داده‌ها را معرفی می‌کند و مروری سریع بر مطالب کتاب خواهد داشت. اصطلاحات و مفاهیم اصلی، تعریف می‌شوند و مثالهایی در رابطه با آنها ارائه می‌شود. مرور کلی بر سازماندهی داده‌ها و ساختمان داده‌های معین به همراه بحث و بررسی عملیات مختلفی که بر روی این ساختمان داده‌ها صورت می‌گیرد از جمله مطالب کتاب را تشکیل می‌دهد. در پایان، مفهوم الگوریتم و پیجیدگی آن معرفی می‌شود و توازن‌بین زمان اجرا و حافظه را که ممکن است در انتخاب یک الگوریتم خاص و ساختمان داده مربوط به یک مسئله معین اتفاق بیفتد مورد بررسی قرار خواهیم داد.

۱-۲ اصطلاحات اصلی و سازماندهی ابتدایی داده‌ها

داده‌ها، فقط مقادیر یا مجموعه‌هایی از مقادیر هستند. یک عنصر داده‌ای، به معنی واحد منحصر بفرد از مقادیر است. عنصرهای داده‌ای که به زیرعنصرهایی تقسیم شده باشد عنصرهای چند قسمتی نامیده می‌شوند و آن دسته از عناصر که چند قسمتی نیستند عنصرهای ابتدایی نامیده می‌شوند. برای مثال، اسم یک کارمند ممکن است از سه زیر عنصر تشکیل شده باشد، اسم اول، اسم وسط و اسم آخر، اما شماره تأمین اجتماعی SSN معمولاً به صورت یک عنصر منحصر بفرد مورد توجه قرار می‌گیرد.

غالباً مجموعه داده‌ها در سلسله مراتب فیلد، رکوردها و فایلها سازماندهی می‌شوند. برای این که مفهوم این اصطلاحات دقیق‌تر بیان شود، چند اصطلاح دیگر را معرفی می‌کنیم. یک موجودیت، چیزی است که دارای خصیصه‌ها یا خواص معین باشد و ممکن است مقادیری به آن نسبت داده شود. این مقادیر، ممکن است عددی یا غیر عددی باشند. برای مثال، برای موجودیتی نظیر کارمند یک سازمان معین، ممکن است خصیصه‌ها و مقادیر متناظر آنها به صورت زیر باشد:

Social Security Number	Sex	Age	Name	خصیصه‌ها :
134 - 24 - 5533	F	34	ROHLAND,GAIL	مقادیر :

موجودیت‌هایی که دارای خصیصه‌های مشابه هستند تمام کارمندان یک سازمان، تشکیل یک مجموعه موجودیت را می‌دهند. هر خصیصه از یک مجموعه موجودیت دارای یک دامنه از مقادیر است، این دامنه، مجموعه تمام مقادیر ممکنی است که می‌توان آنها را به خصیصه‌های خاص نسبت داد. اصطلاح اطلاعات گاهی اوقات به جای داده‌هایی با خصیصه‌های معین بکار می‌رود؛ به عبارت دیگر، به داده‌های دارای معنی یا داده‌های پردازش شده، اطلاعات می‌گویند.

روشی که در آن داده‌ها به سلسله مراتب فیلد، رکورد و فایل سازماندهی می‌شوند در رابطه بین خصیصه‌ها، موجودیت‌ها و مجموعه‌های موجودیت انعکاس پیدا می‌کنند؛ یعنی، یک فیلد یک واحد ابتدایی منحصر بفرد از اطلاعات است که یک خصیصه از یک موجودیت را به نمایش می‌گذارد. یک رکورد مجموعه‌ای از مقادیر فیلدی‌ای یک موجودیت معین است و یک فایل مجموعه‌ای از رکوردهای موجودیت در یک مجموعه از موجودیت معین است.

هر رکورد یک فایل ممکن است از چند فیلد تشکیل شود اما مقدار یک فیلد معین، می‌تواند به طور منحصر بفرد رکورد داخل فایل را مشخص کند. به یک چنین فیلدی که آن را K می‌نامیم کلید اولیه یا اصلی و به مقادیر $1, 2, \dots$ این فیلد، کلیدها یا مقادیر کلیدی می‌گویند.

مثال ۱-۱

(الف) فرض کنید یک بنگاه معاملات اتومبیل مشخصات اتومبیل‌ها را در یک فایل نگهداری می‌کند که هر رکورد آن شامل فیلدی‌ای زیر است:

Serial Number	شماره سریال	Type	نوع	Price	قیمت	Year	سال
فیلد شماره سریال را می‌توان به عنوان کلید اولیه فایل مورد استفاده قرار داد، زیرا هر اتومبیل شماره سریال منحصر بفرد دارد.							

(ب) فرض کنید یک سازمان مشخصات عضوهاش را در یک فایل نگهداری می‌کند که هر رکورد آن شامل فیلدی‌ای زیر است:

Name	نام	Address	آدرس	Telephone Number	شماره تلفن	Dues Owed	دیون بدھکار
------	-----	---------	------	------------------	------------	-----------	-------------

هر چند هر رکورد چهار عنصر داده‌ای دارد، نام و آدرس ممکن است عنصرهای چند قسمی باشند. در اینجا فیلد نام، یک کلید اولیه است. توجه دارید که فیلد های آدرس و شماره تلفن را نمی‌توان به عنوان کلید اولیه مورد استفاده قرار داد چون برخی از عضوها ممکن است متعلق به یک خانواده باشند و درنتیجه دارای یک آدرس و یک شماره تلفن هستند.

رکوردها را می‌توان بر حسب طول آنها نیز دسته‌بندی کرد. یک فایل می‌تواند دارای رکوردهایی با طول ثابت یا رکوردهایی با طول متغیر باشد. در رکوردهای با طول ثابت، تمام رکوردها دارای فیلد های برابر و یکسان هستند و به هر فیلد مقدار حافظه مساوی اختصاص می‌یابد. در رکوردهای با طول متغیر، رکوردهای فایل ممکن است طولهای مختلف داشته باشند؛ برای مثال، رکوردهای دانشجویی معمولاً طول متغیر دارند چون دانشجویان مختلف تعداد درس های متفاوتی را اختیار می‌کنند. معمولاً رکوردهای با طول متغیر، یک طول حداقل و یک طول حداقل دارند.

سازماندهی داده‌ها به شکل بالا به صورت فیلد، رکورد و فایل ممکن است آنقدر پیچیده نباشد تا مجموعه‌هایی از داده‌های معین را در حافظه ذخیره کند و به صورت موثر و کارا مورد پردازش قرار دهد. به همین دلیل، داده‌ها نیز به انواع پیچیده‌تری از ساختمانها تقسیم‌بندی می‌شوند. مطالعه چنین ساختمان داده‌هایی، که موضوع اصلی این کتاب را تشکیل می‌دهد، شامل سه مرحله زیر است:

(۱) توصیف منطقی یا ریاضی ساختمان داده

(۲) پیاده‌سازی این ساختمان داده بر روی یک کامپیوتر

(۳) تجزیه و تحلیل کمی این ساختمان داده که شامل تعیین مقدار حافظه موردنیاز برای ذخیره این ساختمان و زمان موردنیاز برای پردازش آن است.

بخش بعدی، برخی از این ساختمان داده‌ها را به شما معرفی می‌کند.

توجه کنید: مرحله دوم و سوم مطالعه ساختمان داده‌ها بستگی به این دارد که آیا داده‌ها (الف) در حافظه اصلی (اولیه) کامپیوتر ذخیره می‌شوند یا (ب) در واحد حافظه ثانویه (خارجی). این کتاب اساساً حالت اول را پوشش می‌دهد؛ یعنی آدرس دادن به یک خانه حافظه و زمان موردنیاز برای دسترسی به محتوای خانه حافظه، که بستگی به یک حافظه معین یا به محتوای حافظه‌ای که قبلاً به آن دسترسی پیدا کردیم ندارد. حالت دوم، که مدیریت فایل یا مدیریت پایگاه داده‌ها نیز نامیده می‌شود مطالب مربوط به خود را شامل می‌شود که بحث و بررسی آن از محدوده درس ساختمان داده‌ها خارج است.

۳-۱ ساختمان داده‌ها

داده‌ها می‌توانند به صورتهای متفاوتی سازماندهی شوند. مدل منطقی یا ریاضی سازماندهی داده‌ها

به یک صورت خاص، یک ساختمان داده نامیده می‌شود. انتخاب یک مدل خاص از داده‌ها بستگی به دو مسئله قابل توجه دارد. نخست این‌که، از نظر ساختمان باید به اندازه کافی غنی باشد تا بتواند رابطه‌های واقعی بین داده‌ها را در دنیای ما منعکس سازد، از سوی دیگر، این ساختمان باید به اندازه کافی ساده باشد تا بتواند به هنگام نیاز، داده‌ها را به صورت مؤثری پردازش کند. این بخش، برخی از ساختمان داده‌ای را که دیرتر در این کتاب به طور مژروح مورد بررسی قرار می‌گیرد، به ما معرفی می‌کند.

آرایه‌ها

یک آرایه خطی (یا یک بعدی) ساده‌ترین نوع ساختمان داده‌ها است. منظور از یک آرایه خطی، یک لیست متناهی با n عنصر داده‌ای مشابه است که به عناصر آن به ترتیب به کمک مجموعه‌ای از n عدد متوالی که معمولاً $1, 2, 3, \dots, n$ می‌باشد دسترسی پیدا می‌کنیم. اگر ما برای نام آرایه، A را اختیار کنیم، آنگاه عنصرهای آرایه A را با نماد اندیس‌گذاری شده

$$a_1, a_2, a_3, \dots, a_n$$

یا نماد پرانتزی

$$A(1), A(2), A(3), \dots, A(N)$$

یا با نماد کروشهای

$$A[1], A[2], A[3], \dots, A[N]$$

نمایش می‌دهیم.

صرفنظر از نوع نمادگذاری، عدد K در $A[K]$ یک اندیس و $[K]A$ یک متغیر اندیس دار نامیده می‌شود. توجه کنید: نماد پرانتزی و نماد کروشهای غالباً هنگامی مورد استفاده قرار می‌گیرند که نام آرایه بیش از یک حرف داشته باشد یا نام آرایه در یک الگوریتم بکار گرفته شود. هنگام استفاده از این نمادگذاری، معمولاً برای نام آرایه و اندیس آرایه همانگونه که در بالا بیان شده است از حروف بزرگ نظری A و N استفاده می‌کنیم. در غیر اینصورت، می‌توانیم از نمادگذاری متداول اندیسی کچ برای نام و اندیسهاش و از حروف کوچک، به صورتی که در بالا با a و n بیان شده است برای اندیسها استفاده کنیم. از نمادگذاری اول اغلب در کتب چاپی ریاضی استفاده می‌شود حال آنکه در اکثر کتب کامپیوتروی، نمادگذاری دوم دیده می‌شود.

مثال ۱ - ۲

آرایه خطی STUDENT از نام شش دانشجو، به صورتی که در شکل ۱ - ۱ نشان داده، تشکیل شده

است. در اینجا [1] STUDENT John Brown ، STUDENT[2] Sandra Gold و غیره را نمایش می‌دهد.

STUDENT	
1	John Brown
2	Sandra Gold
3	Tom Jones
4	June Kelly
5	Mary Reed
6	Alan Smith

شکل ۱-۱

آرایه‌های خطی، آرایه‌های یکبعدی نیز نامیده می‌شوند زیرا به هر عنصر چنین آرایه‌ای تنها می‌توان از طریق یک اندیس مراجعه کرد. یک آرایه دو بعدی، یک مجموعه از عناصر داده‌ای مشابه است که به هر عنصر آن از طریق دو اندیس مراجعه می‌کنیم. به آرایه دو بعدی در ریاضیات ماتریس و در امور تجاری جدول می‌گویند. آرایه‌های چند بعدی به طور مشابه تعریف می‌شوند. بحث و بررسی مسروح آرایه‌ها در فصل ۴ کتاب گنجانده شده است.

مثال ۱-۳

یک فروشگاه زنجیره‌ای از ۲۸ فروشگاه تشکیل شده است که هر فروشگاه ۴ بخش دارد و می‌توان فروش هفتگی آن را (بر حسب نزدیکترین عدد به دلار) مانند شکل ۱-۲ ارائه داد:

Dept. Store	1	2	3	4
1	2872	805	3211	1560
2	2196	1223	2525	1744
3	3257	1017	3686	1951
...
28	2618	931	2333	982

شکل ۱-۲

اصول ساختمان داده‌ها

این نوع داده‌ها را می‌توان با استفاده از یک آرایه دو بعدی در کامپیوتر ذخیره کرد که در آن اندیس اول شماره فروشگاه و اندیس دوم شماره بخش را نشان می‌دهد. اگر SALES نامی باشد که به این آرایه داده‌ایم، آنگاه

$$\text{SALES}[1, 1] = 2872, \quad \text{SALES}[1, 2] = 805, \quad \text{SALES}[1, 3] = 3211, \dots, \quad \text{SALES}[28, 4] = 982$$

اندازه این آرایه به صورت 28×4 (بخوانید ۲۸ در ۴) نمایش داده می‌شود زیرا این آرایه ۲۸ سطر (شماره‌های خطوط افقی) و ۴ ستون (شماره‌های خطوط عمودی) دارد.

لیستهای پیوندی

به کمک یک مثال لیستهای پیوندی را معرفی می‌کنیم. فرض کنید یک شرکت حق العمل کار، مشخصات مشتری‌های خود را در یک فایل نگهداری می‌کند که هر رکورد آن شامل نام مشتری و فروشنده Salesperson است و فرض کنید این فایل شامل داده‌هایی است که در شکل ۱-۳ Customer

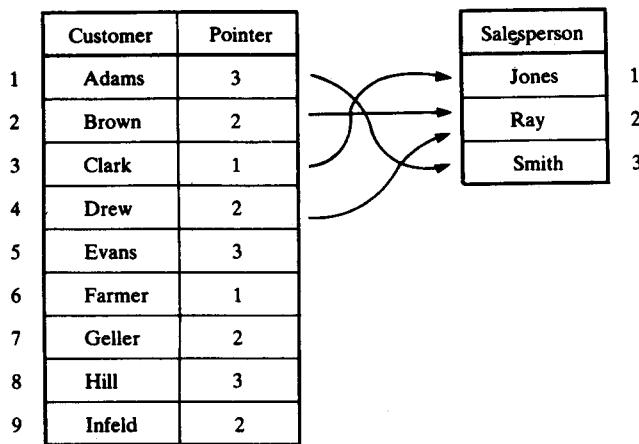
نشان داده شده است:

	Customer	Salesperson
1	Adams	Smith
2	Brown	Ray
3	Clark	Jones
4	Drew	Ray
5	Evans	Smith
6	Farmer	Jones
7	Geller	Ray
8	Hill	Smith
9	infeld	Ray

شکل ۱-۳

واضح است که این فایل می‌توانست به وسیله چنین جدولی یعنی به وسیله یک جدول دوستونه با نام در کامپیوتر ذخیره شود. با وجود این، همانگونه که بحث زیر نشان می‌دهد، این روش نمی‌تواند مفیدترین راه برای ذخیره داده‌ها باشد.

روش دیگر برای ذخیره داده‌های شکل ۱-۳، آن است که یک آرایه مستقل برای فروشنده‌ها و یک مدخل موسوم به اشاره‌گر در فایل مشتری داشته باشیم که به مکان فروشنده هر مشتری اشاره می‌کند. این روش ذخیره داده‌ها در شکل ۱-۴ نشان داده شده است که در آن تنها تعدادی از اشاره‌گرهای به وسیله



شکل ۱-۴

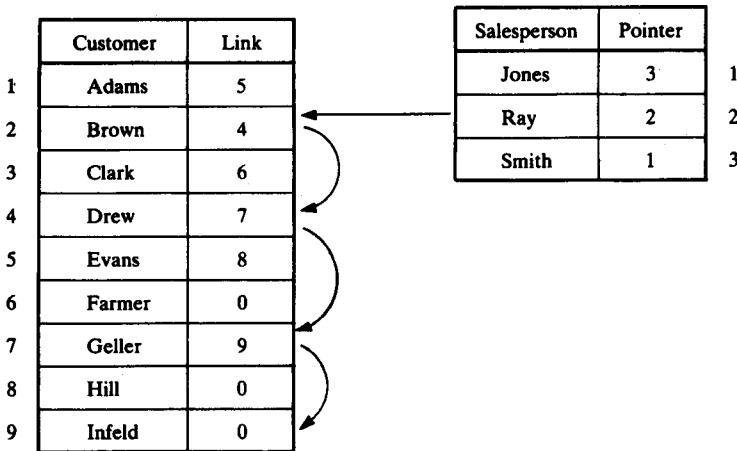
اما در کامپیوتر عملاً از یک عدد صحیح استفاده می‌شود. چون یک اشاره‌گر، نسبت به یک اسم به حافظه کمتری نیاز دارد. از این‌رو، این نحوه نمایش باعث صرفه‌جویی در حافظه می‌شود بمویژه این که اگر هر فروشنده چند صد مشتری داشته باشد، اهمیت صرفه‌جویی در حافظه بیشتر نمایان می‌شود. فرض کنید این شرکت بخواهد لیست تمام مشتریان یک فروشنده معین را به دست آورد. با استفاده از داده‌های نشان داده شده در شکل ۱-۴، شرکت مجبور است در میان فایل تمام مشتری‌ها، فروشنده موردنظر را جستجو کند. یک راه ساده برای چنین جستجویی، داشتن پیکانهایی نظیر شکل ۱-۵ است که به اطلاعات دیگر اشاره می‌کند. اکنون هر فروشنده یک مجموعه از اشاره‌گر دارد که به مکان مشتری‌های وی اشاره می‌کند. شکل ۱-۵ را ببینید.

	Salesperson	Pointer
1	Jones	3, 6
2	Ray	2, 4, 7, 9
3	Smith	1, 5, 8

شکل ۱-۵

عیب اصلی این نمایش آن است که هر فروشنده ممکن است چند اشاره‌گر داشته باشد و با اضافه و حذف شدن مشتریها، مجموعه اشاره‌گرها تغییر خواهد کرد.

یکی از راههای بسیار متداول ذخیره داده‌های شکل ۶-۱، در شکل ۶-۱ نشان داده شده است. در اینجا هر فروشنده یک اشاره‌گر دارد که به مشتری اول اشاره می‌کند که اشاره‌گر آن نیز به نوبه خود به مشتری دوم اشاره می‌کند و الی آخر، که در آن فروشنده مشتری آخر با یک **۰** مشخص شده است. این اشاره‌گرها برای فروشنده‌ای به نام **Ray** به وسیله پیکانهایی، در شکل ۶-۱ نشان داده شده است. با استفاده از این نمایش به سادگی می‌توان لیست تمام مشتریهای یک فروشنده معین را به دست آورد. همانگونه که در فصل ۵ خواهید دید به سادگی می‌توان عمل اضافه کردن و حذف مشتریها را انجام داد.



شکل ۶-۱

نمایش داده‌های شکل ۶-۱ یک مثال از لیستهای پیوندی است. اگرچه معمولاً اصطلاحات "اشاره‌گر" و "پیوند" به یک معنی بکار می‌روند. اما معنی می‌کنیم از اصطلاح "اشاره‌گر" در مواردی استفاده کنیم که یک عنصر از یک لیست به یک عنصر در لیست دیگر اشاره می‌کند و اصطلاح "پیوند" را برای مواردی نگه می‌داریم که یک عنصر از یک لیست به عنصر دیگری از همان لیست اشاره می‌کند.

درختها

داده‌ها اغلب شامل رابطه سلسله مرتبی بین عناصر مختلف هستند. ساختمان داده‌ای که این رابطه را

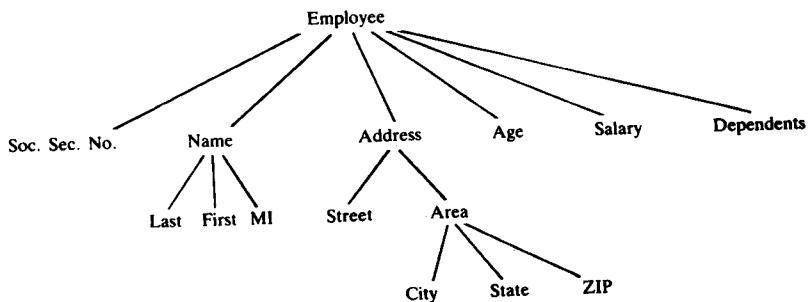
منعکس می‌کند یک گراف درختی ریشه‌دار یا فقط یک درخت نامیده می‌شود. درختها در فصل ۷ تعریف شده و به طور مفصل مورد بحث و بررسی قرار می‌گیرند. در اینجا ما با ارائه دو مثال تنها برخی از خواص اصلی آنها را بیان می‌کنیم.

مثال ۱ - ۴ : ساختمان رکورد

هر چند یک فایل را می‌توان به وسیله یک یا چند آرایه یا رکورد نگهداری کرد که هر یک از آنها، می‌توانند هم فیلد‌های چند قسمتی و هم فیلد‌های ابتدایی باشند. اما می‌توان آن را به وسیله یک ساختمان درختی به شکل بهتری بیان کرد. برای مثال، رکورد پرستنی یک کارمند می‌تواند شامل فیلد‌های زیر باشد :

Social Security Number, Name, Address, Age, Salary, Dependents

با وجود این، نام **Name** ممکن است یک فیلد چند قسمتی با فیلد‌های کوچک **Name**، نام آخر، **Last** نام اول **First** و نام وسط **MI** باشد. علاوه بر این، آدرس **Address** ممکن است یک فیلد چند قسمتی با فیلد‌های کوچک آدرس خیابان **Street** و آدرس منطقه **Area** باشد که در آن منطقه **Area** خود می‌تواند یک فیلد چند قسمتی با فیلد‌های کوچک شهر، **City**، ایالت **State** و شماره کد پستی **ZIP** باشد. این ساختمان سلسله مراتبی در شکل ۱-۷ (الف) به تصویر درآمده است.



شکل ۱-۷ (الف)

روش دیگر به نمایش گذاشتن یک ساختن درختی از این نوع، برحسب سطوح آن در شکل ۱-۷ (ب) نشان داده شده است.

```

01 Employee
  02 Social Security Number
  02 Name
    03 Last
    03 First
    03 Middle Initial
  02 Address
    03 Street
    03 Area
      04 City
      04 State
      04 ZIP
  02 Age
  02 Salary
  02 Dependents

```

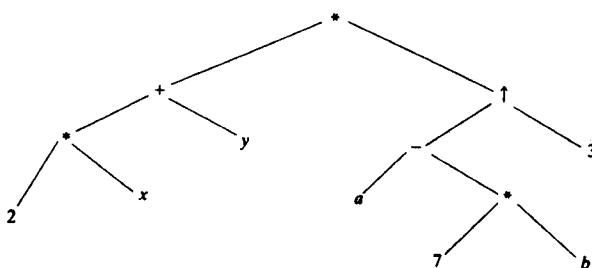
شکل ۱-۷ (ب)

مثال ۵-۱: عبارتهای جبری

عبارت جبری زیر را در نظر بگیرید:

$$(2x+y)(a-7b)^3$$

با استفاده از پیکان به طرف بالای \uparrow برای توان و ستاره * برای ضرب، می‌توانیم این عبارت را به صورت درخت شکل ۱-۸ نمایش دهیم.



شکل ۱-۸

ملاحظه می‌کنید که ترتیب اجرای عملیات در این نمودار نیز، انعکاس یافته است، عمل توان رساندن باید بعد از انجام عمل تفکیق عبارت داخل پرانتز اجرا شود و عمل ضرب نشان داده شده در بالای درخت باید آخر اجرا شود.

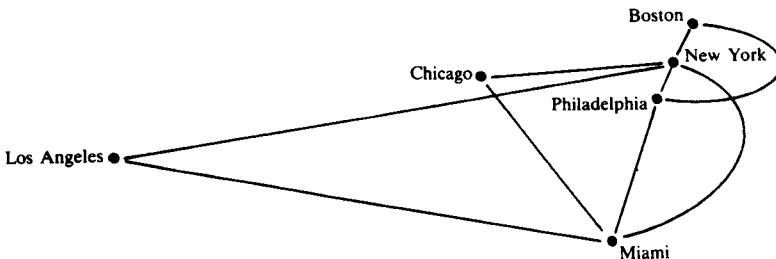
به غیر از آرایه‌ها، لیستهای پیوندی و درختها، ساختمان داده‌های دیگری وجود دارند که ما برخی از این ساختمانها را که به اختصار در زیر شرح داده‌ایم دیگر بطور مشروح مورد مطالعه قرار خواهند گرفت.
 (الف) پشته Stack: یک پشته که به آن یک سیستم LIFO یا آخرین ورودی اولین خروجی است، نیز

می‌گویند، یک لیست خطی است که در آن عملیات اضافه شدن عناصر تنها از یک انتهای آن موسوم به بالای پشته Top صورت می‌گیرد. این ساختمان، عملیاتی مشابه یک پشته از بشقابها بر روی یک سیستم فنری دارد که در شکل ۱-۹ (الف) نشان داده شده است. توجه دارید که بشقابهای جدید را می‌توان تنها از بالای پشته به آن اضافه کرد و عمل حذف یا برداشتن بشقابها تنها از بالای پشته امکان‌پذیر است.



(ب) صف انتظار اتوبوس

(الف) یک پشته از بشقابها



(ج) خطوط هواپیمایی

۱-۹

(ب) صف Queue: یک صف، که به آن یک سیستم FIFO یا اولین ورودی اولین خروجی است، نیز می‌گویند، یک لیست خطی است که در آن عملیات حذف عناصر تنها از یک انتهای لیست موسوم به ابتداء Front لیست و اضافه شدن عناصر به صف تنها از انتهای دیگر لیست، موسوم به انتهای Rear لیست صورت می‌گیرد. این ساختمان داده به همان صورتی عمل می‌کند که ما در صف انتظار یک ایستگاه اتوبوس عمل می‌کنیم. این وضعیت در شکل ۱-۹ (ب) به تصویر درآمده است: اولین نفر داخل صف اولین کسی است که سوار اتوبوس می‌شود. یک مثال دیگر از صف، مربوط به اتومبیل‌هایی است که در یک چهارراه در انتظار عبور به سر می‌برند، به محض سبز شدن چراغ راهنمایی، اولین اتومبیل داخل

صف اولین اتومبیلی است که از چهارراه عبور می‌کند.

(ج) گراف: گاهی اوقات داده‌ها یک رابطه بین جفت عناصر موجود در طبیعت را بیان می‌کنند که لزوماً به صورت سلسله مراتبی نیستند. برای مثال، خطوط هوایی شکل ۱-۹ (ج) را درنظر بگیرید که ارتباط بین شهرها تنها با رسم خطوط نشان داده شده است. ساختمان داده‌ای که این نوع رابطه را بیان می‌کند یک گراف نام دارد. گرافها به صورت رسمی در فصل ۸ تعریف شده، مورد مطالعه و بررسی کامل قرار می‌گیرند.

توجه کنید: برای عناصر یک ساختمان داده از اسم‌های مختلف و گوناگونی استفاده می‌شود. برخی از اسم‌های متداول عبارتند از: عنصر داده‌ای، فیلد، مجموعه‌ای از فیلدها، رکورد، گره و شیئی داده‌ای. اسم خاصی که مورد استفاده قرار می‌گیرد، بستگی به نوع ساختمان داده‌ای دارد که در یک متن یا یک مبحث، از آن ساختمان داده استفاده می‌شود و برنامه‌نویسان از آن اسم استفاده می‌کنند. ما برای این منظور اصطلاح "عنصر داده‌ای" را ترجیح می‌دهیم اما هنگام بحث و بررسی فایلها از اصطلاح رکورد و هنگام مطالعه لیستهای پیوندی، و همچنین درختها و گرافها از اصطلاح گره استفاده می‌کنیم.

۴-۱ عملیات بر روی ساختمان داده‌ها

داده‌هایی که در ساختمان داده‌ها ظاهر می‌شوند به وسیلهٔ عملیات مشخصی پردازش می‌شوند. در واقع، ساختمان داده خاصی که برنامه‌نویس برای یک مسئله معین انتخاب می‌کند بستگی زیاد به میزان عملیات خاصی دارد که در آن مسئله انجام می‌شود. در این بخش دانشجو با برخی از این عملیات که زیاد مورد استفاده قرار می‌گیرد، آشنا می‌شود.

در این کتاب چهار عمل زیر نقش اصلی را ایفا می‌کنند:

(۱) پیمایش: دسترسی به اطلاعات یک رکورد آن هم دقیقاً یکبار، پیمایش نامیده می‌شود، طوری که بتوان بعضی از فیلدهای رکورد را مورد پردازش قرار داد. گاهی اوقات به دسترسی و پردازش رکورد ملاقات نیز گفته می‌شود.

(۲) جستجو: به عمل یافتن مکان یک رکورد با یک مقدار کلیدی معین یا پیدا کردن مکان تمام رکوردهایی که در یک یا چند شرط صدق می‌کنند، جستجو می‌گویند.

(۳) اضافه کردن: افزودن رکورد جدید به ساختمان داده است.

(۴) حذف کردن: حذف یک رکورد از ساختمان داده است.

گاهی اوقات ممکن است از دو یا چند عمل در یک کار مشخص استفاده شود. مثلاً چنانچه بخواهیم یک رکورد با کلید معلوم را حذف کنیم لازم است نخست مکان آن رکورد را جستجو (پیدا) کنیم.

- دو عمل زیر نیز در وضعیت‌های خاص مورد استفاده قرار می‌گیرند که در زیر آنها را از نظر می‌گذرانیم:
- (۱) مرتب‌کردن: عبارت است از قراردادن رکوردها با یک نظم معین در کنار هم. مثلاً با یک نظم الفبایی براساس یک کلید نام **NAME** یا با یک نظم عددی بربطیک یک کلید عددی **NUMBER** مانند شماره تأمین اجتماعی یا شماره حساب بانکی.
 - (۲) ادغام‌کردن: ترکیب رکوردهای دو فایل مرتب‌شده مختلف و قراردادن آنها در یک فایل مرتب شده ادغام کردن نام دارد.
- عملیات دیگری مانند کپی‌کردن و اتصال اطلاعات دیرتر در این کتاب بررسی می‌شود.

مثال ۶-۱

یک سازمان، یک فایل برای اعضا‌یاش دارد که در آن فایل، هر رکورد یک عضو معین، دارای فیلدهای زیر است:

Name,	Address,	Telephone Number,	Age,	Sex
-------	----------	-------------------	------	-----

- (الف) فرض کنید این سازمان بخواهد یک گردبهم آبی برای اعضا‌یاش ترتیب دهد. در آن صورت لازم است که عمل پیمایش در فایل انجام شود تا نام و آدرس هر عضو بدست آید.
- (ب) فرض کنید این سازمان بخواهد نام تمام عضوهایی را که در یک منطقه معین زندگی می‌کنند به دست آورد. برای این حالت نیز لازم است عمل پیمایش در فایل انجام شود تا اطلاعات موردنیاز به دست آید.
- (ج) فرض کنید این سازمان بخواهد آدرس یک نام **Name** معین را بدست آورد. در آن صورت لازم است عمل جستجو در این فایل انجام شود تا رکورد مربوط به این نام **Name** به دست آید.
- (د) فرض کنید شخص جدیدی به این سازمان پیوسته است. در آن صورت لازم است رکورد وی به فایل اضافه شود.
- (ه) فرض کنید یک عضو این سازمان در گذشته است. در آن صورت لازم است رکورد وی از این فایل حذف شود.

- (و) فرض کنید یک عضو محل زندگی اش را تغییر داده است و دارای آدرس و شماره تلفن جدید است. با معلوم بودن نام این عضو، نخست لازم است رکورد او را در فایل جستجو کنیم. آنگاه لازم است عمل "تازه شدن" یا **Update** انجام شود یعنی بعضی از فیلدهای رکورد او را با توجه به مشخصات تازه‌ای او تغییر دهیم.
- (ز) فرض کنید این سازمان بخواهد تعداد اعضای خود را که ۶۵ سال یا بیشتر سن دارند به دست آورد.

برای این حالت نیز لازم است عمل پیمایش در فایل انجام شود و تعداد این عضوها شمرده شوند.

۵- ۱ الگوریتم‌ها: پیچیدگی الگوریتم، توازن بین زمان اجرا و حافظه

یک الگوریتم، یک لیست خوش تعریف از مراحلی است که برای حل یک مسأله معین در نظر گرفته می‌شود. یکی از اهداف اصلی این کتاب ارائه الگوریتم‌های کارا و مؤثر برای پردازش داده‌ها است. از زمان و حافظه به عنوان دو معیار اصلی در کارآیی یک الگوریتم استفاده می‌شود. پیچیدگی یک الگوریتم، تابعی است که زمان اجرا و / یا حافظه را بحسب تعداد داده‌های ورودی به دست می‌دهد. مفهوم پیچیدگی یک الگوریتم در فصل ۲ بررسی می‌شود.

هر یک از الگوریتم‌های این کتاب حاوی ساختمان داده خاصی است. براین اساس همیشه نمی‌توانیم از کاراترین الگوریتم استفاده کنیم چون انتخاب ساختمان داده به عوامل زیادی از قبیل، نوع داده‌ها و تعداد دفعاتی که با آن، عملیات داده‌ای متعدد انجام می‌شود بستگی دارد. گاهی اوقات در انتخاب ساختمان داده‌ها مسأله توازن (سبک و سنگینی) بین زمان اجرا و حافظه مورد توجه قرار می‌گیرد به این معنی که: جهت ذخیره داده‌ها، با افزایش مقدار حافظه، می‌توان زمان لازم برای پردازش داده‌ها را کاهش داد و بر عکس، این ایده‌ها را با دو مثال روشن می‌کنیم.

الگوریتم‌های جستجو

فایل اعضای سازمان مثال ۶-۱ را در نظر بگیرید که در آن هر رکورد به همراه چند فیلد دیگر، دارای دو فیلد نام Name و شماره تلفن Telephone Number است. فرض کنید که نام یک عضو داده شده است و بخواهیم شماره تلفن او را پیدا کنیم. یک راه برای این منظور جستجوی خطی در فایل است یعنی از الگوریتم زیر استفاده کنیم.

جستجوی خطی: در هر یک از رکوردهای فایل نام داده شده را جستجو کنید به محض پیدا شدن رکوردي با نام داده شده و درنتیجه با شماره تلفن متناظر با آن، به عمل جستجو خاتمه دهد.

قبل از هر کاری، واضح است که زمان موردنیاز اجرای این الگوریتم مناسب با تعداد مقایسه‌ها است. همچنین فرض کنید هر نام Name درون فایل، احتمال مساوی برای انتخاب شدن داشته باشد. به طور شهودی واضح است که میانگین تعداد مقایسه‌ها برای یک فایل با n رکورد برابر $2^n - 1$ است یعنی پیچیدگی الگوریتم جستجوی خطی به صورت $2^n = C(n)$ است.

چنانچه بخواهیم عمل جستجو را در میان لیستی شامل چند هزار نام، مانند دفترچه تلفن انجام دهیم استفاده از الگوریتم بالا عملاً غیرممکن است. با وجود این، اگر نامها به صورت الفبایی مرتب باشند

مانند دفترچه‌های تلفن، در آن صورت می‌توانیم از یک الگوریتم کارا به نام جستجوی دودویی استفاده کنیم. این الگوریتم به تفصیل در فصل ۴ شرح داده شده است ولی ایده کلی آن به اختصار در زیر شرح داده می‌شود.

جستجوی دودویی : نام داده شده را با نام وسط لیست مقایسه کنید. انجام این کار به ما می‌گوید که نام داده شده در کدام نیمه از لیست است. آنگاه نام داده شده را با نام وسط نیمة راست مقایسه کنید تا معلوم شود این نام در کدام ربع از لیست وجود دارد. عمل جستجو را تا پیداشدن نام مورد نظر در لیست ادامه دهید.

می‌توان نشان داد که پیچیدگی الگوریتم جستجوی دودویی از رابطه

$$C(n) = \log_2 n$$

به دست می‌آید. برای مثال، برای پیدا کردن یک نام **Name** داده شده در یک لیست با ۲۵۰۰۰ نام، به بیش از ۱۵ مقایسه نیاز نداریم.

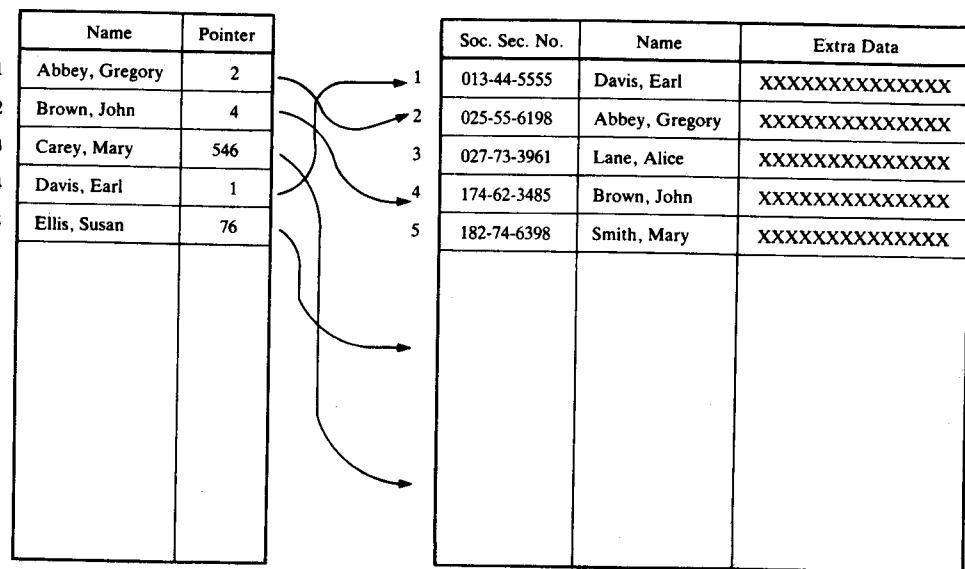
اگرچه الگوریتم جستجوی دودویی الگوریتم بسیار کارآیی است اما یک عیب اساسی دارد. به خصوص این که، این الگوریتم فرض می‌کند به نام وسط لیست یا لیستهای کوچک شده بعدی دسترسی مستقیم داریم به این معنی که لیست باید در نوعی از آرایه ذخیره شده باشد. متأسفانه، اضافه کردن یک عنصر در یک آرایه مستلزم آن است عناصر آرایه یک خانه به طرف پائین لیست انتقال داده شوند و حذف یک عنصر از یک آرایه نیز مستلزم انتقال عناصر آرایه یک خانه به طرف بالای لیست است.

شرکت تلفن، با چاپ یک دفترچه تلفن جدید در هر سال مسئله بالا را حل می‌کند و این در حالی است که شماره تلفن مشتریان جدید را به طور جداگانه در یک فایل موقت نگه می‌دارد یعنی شرکت تلفن هرساله فایل مشترکینش را به روز در می‌آورد. از طرف دیگر، یک بانک، می‌تواند مشخصات مشتری جدید را تقریباً به طور لحظه‌ای در فایل مربوطه اضافه کند. بنابراین، برای یک بانک یک لیست مرتب شده خطی بهترین ساختمان داده نیست.

یک مثال از توازن بین زمان اجرا و حافظه مصرفی

فرض کنید یک فایل از رکوردهایی شامل فیلدهای نام، شماره تأمین اجتماعی و چند فیلد اضافی دیگر تشکیل شده است. مرتب کردن این فایل به صورت الفایی با استفاده از یک جستجوی دودویی راه بسیار کارآیی در پیدا کردن یک رکورد با نام معین است. از طرف دیگر فرض کنید تنها شماره تأمین اجتماعی یک شخص داده شده است. در آن صورت مجبوریم برای این رکورد یک جستجوی دودویی انجام دهیم که این روش، برای فایلی با تعداد رکوردهای بسیار زیاد زمان اجرا را به میزان قابل توجهی بالا

می‌برد. برای حل اینگونه مسایل چه روشی می‌توان در پیش گرفت؟ یک راه آن است که فایل دیگری داشته باشیم که آن فایل براساس شماره تأمین اجتماعی به صورت عددی مرتب شده باشد. با وجود این، این روش حافظه موردنیاز برای مرتب کردن داده‌ها را به دو برابر افزایش می‌دهد. راه دیگر که در شکل ۱۰-۱ نشان داده شده است، آن است که فایل اصلی را به صورت مرتب شده عددی براساس شماره تأمین اجتماعی و یک آرایه کمکی تنها با دو ستون در اختیار داشته باشیم، که در این آرایه، ستون اول شامل نام‌ها با ترتیب الفبایی و ستون دوم شامل اشاره‌گرهایی باشد که این اشاره‌گرهای مکان رکوردهای متناظر آن در فایل اصلی اشاره کنند. این روش یکی از راه حل‌های این مسئله است که اغلب مورد استفاده قرار می‌گیرد، زیرا حافظه اضافی که تنها شامل دو ستون است برای مقدار اطلاعات اضافی تدارک دیده شده، حداقل حافظه می‌باشد.



شکل ۱۰-۱

آرایه کمکی که به صورت الفبایی مرتب شده است.

فایل اصلی
که براساس شماره تأمین اجتماعی
مرتب شده است. Soc. Sec. No

توجه کنید: فرض کنید یک فایل به صورت الفبایی براساس شماره تأمین اجتماعی مرتب شده است.

به محض اضافه شدن یک رکورد جدید در این فایل، رکوردها باید پیوسته به مکان جدید خود انتقال پیدا کنند تا ترتیب مرتب شده رکوردها حفظ شود. یک راه ساده برای به حداقل رساندن مقدار جابجایی رکوردها آن است که از شماره تأمین اجتماعی به عنوان آدرس هر رکورد استفاده کنیم. با این کار به هنگام اضافه شدن یک رکورد، نه تنها هیچ گونه جابجایی صورت نمی‌گیرد بلکه در هر لحظه می‌توان به هر رکوردي دسترسی پیدا کرد. بنابراین برای تنها چندصد یا احتمالاً چندهزار رکورد، این روش مرتب کردن اطلاعات به یک میلیارد (10⁹) خانه حافظه احتیاج دارد و واضح است که این مقدار حافظه در مقابل زمان اجرا مقرن به صرفه و اقتصادی نیست. یک روش دیگر برای این منظور، تعریف یکتابع H از مجموعه مقادیر کلید K یعنی، شماره تأمین اجتماعی به مجموعه L از آدرس خانه‌های حافظه است. به یک چنین تابع H ای، تابع درهم‌ساز Hashing می‌گویند. تابعهای درهم‌ساز و خواص آن مطالب فصل ۹ این کتاب را تشکیل می‌دهد.

مسئله‌های حل شده

اصطلاحات اصلی

مسئله ۱ - ۱ : یک استاد دانشگاه لیست دانشجویان یک کلاس را که شامل اطلاعات زیر است نگهداری می‌کند:

Name,	Major,	Student Number,	Test Scores,	Final Grade
-------	--------	-----------------	--------------	-------------

(الف) موجودیتها، خصیصه‌ها و مجموعه موجودیت لیست را تعیین کنید.

(ب) مقادیر فیلد، رکوردها و فایل‌ها را شرح دهید.

(ج) برای این لیست چه خصیصه‌هایی را می‌توان به عنوان کلیدهای اولیه مورد استفاده قرار داد؟

حل : (الف) هر دانشجو یک موجودیت است و مجموعه دانشجویان مجموعه موجودیت است. ویژگی‌های دانشجویان از قبل نام دانشجو، رشته او و غیره خصیصه آنها هستند.

(ب) مقادیر فیلد، مقادیری هستند که به خصیصه‌ها نسبت داده می‌شوند یعنی نامهای واقعی، نمرات امتحانات و غیره. مقادیر فیلد هر دانشجو یک رکورد را تشکیل می‌دهد و مجموعه‌ای از تمام رکوردهای دانشجویان فایل دانشجویی آنها است.

(ج) نام **Name** یا شماره دانشجویی **Student Number** را می‌توان به عنوان کلید اولیه مورد استفاده قرار داد، چون هر یک از این فیلد را رکورد دانشجویی را به طور منحصر بفرد مشخص می‌کنند. معمولاً استاد دانشگاه نام **Name** را به عنوان کلید اولیه به کار می‌برد. درحالی که اداره ثبت نام دانشگاه ممکن است از

شماره دانشجویی **Student Number** به عنوان کلید اولیه استفاده کند.

مسئله ۱-۲ : یک بیمارستان مشخصات بیماران را در یک فایل نگهداری می‌کند که در آن هر رکورد شامل فیلدهای زیر است :

Admission Date, Social Security Number, Room, Bed Number, Doctor

(الف) چه فیلدی را می‌توان به عنوان کلید اولیه مورد استفاده قرار داد؟

(ب) کدام جفت فیلد را می‌توان به عنوان کلید اولیه مورد استفاده قرار داد؟

(ج) کدامیک از فیلدها چندقسمتی هستند؟

حل : (الف) نام **Name** و شماره تأمین اجتماعی **Social Security Number** را می‌توان به عنوان کلید اولیه مورد استفاده قرار داد. فرض کردہ ایم هیچ دو بیماری همنام نیستند.

(ب) اتاق **Room** و شماره تخت **Bed Number** با هم به طور منحصر بفرد یک بیمار را مشخص می‌کنند.

(ج) نام **Name**، تاریخ پذیرش **Admission Date** و دکتر **Doctor** فیلدهای چندقسمتی هستند.

مسئله ۳-۱ : کدامیک از فیلدهای داده‌ای زیر هنگامی که به عنوان فیلد در رکورد گنجانده شوند می‌توانند منتهی به رکوردهایی با طول متغیر شوند: (الف) سن **age** (ب) جنس **sex** (ج) نام همسر (د) نام بچه‌ها (ه) میزان تحصیلات **education** (و) کارمندان سابق.

حل : چون (د) و (و) می‌توانند فیلدهای چندقسمتی با تعداد کم یا تعداد زیادی زیرفیلد باشند از این رو می‌توانند منتهی به رکوردهایی با طول متغیر شوند. علاوه بر این (ه) می‌تواند چند فیلد باشد مگر آنکه، فقط بالاترین سطح تحصیلات موردنظر باشد.

مسئله ۴-۱ : سیستم‌های پایگاه اطلاعات تنها به اختصار در این کتاب مطرح شده است. چرا؟

حل : چون سیستم‌های پایگاه اطلاعات بخشی از علم کامپیوتر است که به داده‌های ذخیره شده در حافظه ثانویه کامپیوتر مربوط می‌شود. پیاده‌سازی و تجزیه و تحلیل ساختمان داده‌های حافظه ثانویه بسیار متفاوت با ساختمان داده‌های حافظه اصلی کامپیوتر هستند. این کتاب اساساً ساختمان داده‌های حافظه اصلی را مورد بررسی قرار می‌دهد نه حافظه فرعی با ثانویه را.

ساختمان داده‌ها و عملیات بر روی آنها

مسئله ۵-۱ : هر یک از اصطلاحات زیر را به اختصار شرح دهید. (الف) پیمايش (ب) مرتب کردن و (ج) جستجوکردن.

حل : (الف) دقیقاً به یکبار دسترسی و پردازش اطلاعات هر رکورد پیمايش می‌گویند.

(ب) قراردادن داده‌ها با یک نظم و ترتیب خاص در کنار هم مرتب کردن نام دارد.

(ج) یافتن مکان رکورد با کلید یا کلیدهای معین جستجو کردن نام دارد.

مسئله ۶-۱: هر یک از اصطلاحات زیر را به اختصار شرح دهید. (الف) اضافه کردن رکورد (ب) حذف رکورد.

حل: (الف) افزودن یک رکورد جدید به یک ساختمان داده، که در آن معمولاً نظم قبلی حفظ می‌شود.

(ب) حذف یک رکورد خاص از ساختمان داده، حذف رکورد نام دارد.

مسئله ۷-۱: آرایه خطی NAME در شکل ۱۱-۱ را درنظر بگیرید که به ترتیب الفابی مرتباً است.

NAME	
1	Adams
2	Clark
3	Evans
4	Gupta
5	Jones
6	Lane
7	Pace
8	Smith

شکل ۱۱-۱

(الف) در اینجا NAME[2] ، NAME[4] ، NAME[7] را پیدا کنید.

(ب) هرگاه بخواهیم Davis را به این آرایه اضافه کنیم، چه تعداد نام بایستی به مکانهای جدید انتقال یابد؟

(ج) هرگاه بخواهیم Gupta را از این آرایه حذف کنیم، چه تعداد نام بایستی به مکانهای جدید انتقال یابد؟

حل: (الف) در اینجا NAME[K] ، K امین نام در لیست است. از این رو

$$\text{NAME}[2] = \text{Clark}, \quad \text{NAME}[4] = \text{Gupta}, \quad \text{NAME}[7] = \text{Pace}$$

(ب) از آنجاکه Davis در [3] جایگزین خواهد شد، نامهای Evans تا Smith باید جابجا شوند. از این رو شش نام باید به مکان جدید انتقال یابد.

(ج) نامهای Jones تا Smith باید به بالای آرایه انتقال یابد. از این رو چهار نام باید جابجا شود.

مسئله ۸-۱: آرایه خطی NAME در شکل ۱۲-۱ را درنظر بگیرید.

FIRST

	NAME	LINK
1	Rogers	7
2	Clark	8
3		
4	Hansen	10
5	Brooks	2
6	Pitt	1
7	Walker	0
8	Fisher	4
9		
10	Leary	6

شکل ۱۲-۱

مقادیر FIRST و LINK[K] در شکل، یک ترتیب خطی از نامها را به صورت زیر مشخص می‌کنند.
FIRST به مکان اولین نام درون لیست اشاره می‌کند و [K] به مکان نام بعدی NAME[K] اشاره می‌کند که در آن ۰ علامت پایان لیست است. ترتیب خطی نامها را مشخص کنید.

حل : ترتیب مورد نظر به صورت زیر به دست می‌آید :

$\text{FIRST} = 5$ ، از این رو اولین نام درون لیست NAME[5] یا Brooks است.

از این رو نام بعدی NAME[2] یا Clark است.

از این رو نام بعدی NAME[8] یا Fisher است.

از این رو نام بعدی NAME[4] یا Hansen است.

از این رو نام بعدی NAME[10] یا Leary است.

از این رو نام بعدی NAME[6] یا Pitt است.

از این رو نام بعدی NAME[1] یا Rogers است.

از این رو نام بعدی NAME[7] یا Walker است.

LINK[7] = ۰ ، که نشان‌دهنده پایان لیست است.

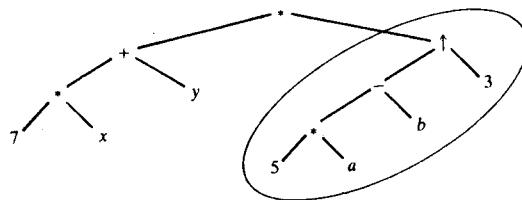
بنابراین، ترتیب خطی نام‌ها عبارت است از:

Brooks , Clark , Fisher , Hansen , Leary , Pitt, Rogers , Walker

توجه دارید که این همان ترتیب الفبایی نامها است.

مسئله ۹ - ۱ : عبارت جبری $(5a - b)^3 = 7x + y$ را درنظر بگیرید. (الف) نمودار درختی متناظر با آن را مانند مثال ۵ رسم کنید. (ب) دامنه تغییرات نماد توان رسانی را مشخص کنید. (دامنه تغییرات گره ۷ در یک درخت، زیردرختی است که شامل ۷ و گره‌های بعد از ۷ است).

حل : (الف) با استفاده از پیکان به طرف بالای \uparrow برای توان و ستاره * به جای ضرب، درخت شکل ۱-۱۳ به دست می‌آید.



شکل ۱-۱۳

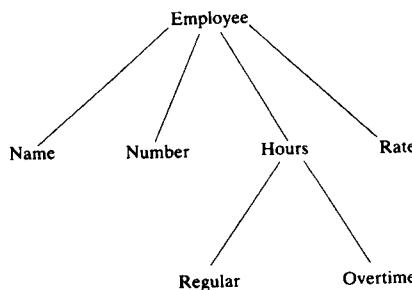
(ب) دامنه تغییرات نماد توان \uparrow در نمودار، زیردرختی است که با بیضی مشخص شده است و متناظر با عبارت $(5a - b)^3$ است.

مسئله ۱۰ - ۱ : اطلاعات زیر مربوط به یک ساختمان درختی است که به وسیله شماره‌های سطح آن، به صورتی که در مثال ۴ توصیف شده است مشخص شده است.

01 Employee	02 Name	02 Number	02 Hours	03 Regular	03 Overtime	02 Rate
-------------	---------	-----------	----------	------------	-------------	---------

نمودار درختی متناظر با آن را رسم کنید.

حل : نمودار درختی آن در شکل ۱۴ - ۱ نشان داده شده است.



شکل ۱-۱۴

در اینجا هر گرگه ۷، عنصر بعدی گره‌ای است که قبل از ۷ قرار دارد و شماره سطح آن پائین‌تر از ۷ است.
مسئله ۱۱-۱: مطلوب است بحث و بررسی این مطلب که از پشتی یا صفحه، کدامیک ساختاری مناسب برای تعیین ترتیبی است که در آن، عناصر در هر یک از وضعیت‌های زیر پردازش می‌شوند.

(الف) برنامه‌های کامپیوتری دسته‌ای Batch که به مرکز محاسبات ارائه می‌شود.

(ب) برنامه A زیر برنامه B را فرا می‌خواند که B به نوبه خود، زیر برنامه C را فرا می‌خواند و الى آخر.

(ج) کارمندان، قراردادی را به امضا می‌رسانند که در آن، روش خاص سیستم ارشدیت برای استخدام یا اخراج منظور می‌شود.

حل: (الف) صفحه، صرف نظر از حالت‌های اولویت داده شده، برنامه‌هایی که اول از همه می‌آیند در اولین مرحله سرویس می‌گیرند.

(ب) پشتی. آخرین زیر برنامه، اول اجرا می‌شود و نتیجه آن به برنامه‌ای داده می‌شود که قبل از برنامه آخر است که بعد از آن اجرا می‌شود و الى آخر، تا این که اولین برنامه فراغ‌خواننده اجرا شود.

(ج) پشتی. در یک سیستم ارشدیت، آخرین عضو استخدامی اولین فرد قابل اخراج محسوب می‌شود.

مسئله ۱۲-۱: پروازهای روزانه یک شرکت هوایی در شکل ۱۵-۱ نشان داده شده است. CITY لیست شهرها و [K] و ORIG[K] و DEST[K] به ترتیب شهرهای مبدأ و مقصد پرواز شماره NUMBER[K] را نمایش می‌دهد. گراف جهت دار متناظر با این اطلاعات را رسم کنید. (این گراف جهت دار است چون شماره‌های پرواز، پرواز از یک شهر به شهر دیگر را بدون برگشت نشان می‌دهد.)

	NUMBER	ORIG	DEST
1	701	2	3
2	702	3	2
3	705	5	3
4	708	3	4
5	711	2	5
6	712	5	2
7	713	5	1
8.	715	1	4
9	717	5	4
10	718	4	5

(ب)

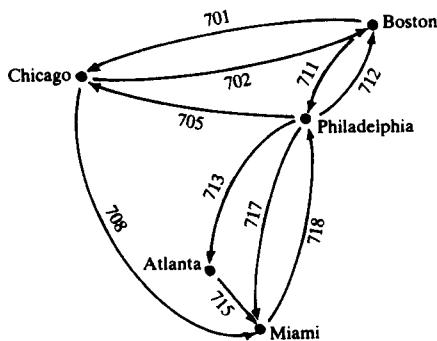
CITY

1	Atlanta
2	Boston
3	Chicago
4	Miami
5	Philadelphia

(الف)

شکل ۱۵-۱

حل: گره‌های این گراف از پنج شهر تشکیل شده است. اگر پروازی از شهر A به B وجود داشته باشد یک پیکان از شهر A به شهر B رسم کنید و پیکان را با شماره پرواز، شماره گذاری کنید. گراف جهت دار این مسئله در شکل ۱-۱۶ رسم شده است.



شکل ۱-۱۶

پیچیدگی و توازن بین زمان اجرا و حافظه

مسئله ۱-۱۳-۱: هر یک از مفاهیم زیر را به اختصار شرح دهید.

(الف) پیچیدگی یک الگوریتم (ب) توازن بین زمان اجرا و حافظه الگوریتم.

حل: (الف) پیچیدگی یک الگوریتم تابعی به صورت $(n)^f$ است که مدت زمان اجرا و/ یا مقدار حافظه استفاده شده توسط یک الگوریتم را بر حسب تعداد داده‌های ورودی n اندازه می‌گیرد.

(ب) توازن بین زمان اجرا و حافظه به انتخابی در میان جوابهای الگوریتمی یک مسئله پردازش داده‌ها مربوط می‌شود که اجازه می‌دهد زمان اجرای یک جواب الگوریتمی را با افزایش خانه‌های حافظه که داده‌ها را ذخیره می‌کند کاهش دهیم و برعکس.

مسئله ۱-۱۴-۱: فرض کنید S یک مجموعه n عنصری از داده‌ها باشد.

(الف) T_1 زمان اجرای الگوریتم جستجوی خطی را با T_2 زمان اجرای الگوریتم جستجوی دودویی مقایسه کنید هرگاه (i) $n = 1000$ و (ii) $n = 10000$.

(ب) فرض کنید S به صورت یک لیست پیوندی ذخیره شده است. جستجوی یک مشخص در S را مورد بررسی قرار دهید.

حل: از بخش ۵-۱ یادآور می‌شویم که زمان اجرای انتظاری الگوریتم جستجوی خطی برابر $2/n$ است.

و از آن الگوریتم جستجوی دودویی برابر $f(n) = \log_2 n$ است. درنتیجه (i) به ازای $n = 1000$ $T_1 = \log_2 1000 \approx 10$ و لی (ii) به ازای $n = 10000$ $T_1 = 5000$ ، $n = 10000$ $T_2 = \log_2 10000 \approx 14$ است.

(ب) الگوریتم جستجوی دودویی فرض می‌کند مستقیماً می‌توان به عنصر وسط مجموعه S دسترسی پیدا کرد اما در یک لیست پیوندی نمی‌توان مستقیماً به عنصر وسط مجموعه S دسترسی پیدا کرد. از این رو وقتی S

به صورت یک لیست پیوندی ذخیره می‌شود بهتر است از الگوریتم جستجوی خطی استفاده شود.

مسئله ۱۵ - ۱: داده‌های شکل ۱۵ - ۱ را در نظر بگیرید که اطلاعاتی راجع به پروازهای مختلف یک خط هوایپیمایی به دست می‌دهد. راههای مختلف ذخیره داده‌ها که زمان اجرا را در حالت‌های زیر پائین می‌آورد

موردن بررسی قرار دهید :

(الف) با معلوم بودن شماره پرواز، مبدأ و مقصد را پیدا کند.

(ب) با معلوم بودن دو شهر A و B ، تعیین کنید که آیا پرواز از A به B وجود دارد یا خیر و در صورت مثبت بودن جواب، شماره پرواز آن را پیدا کنید.

حل : (الف) داده‌های شکل ۱۵ - ۱ (ب) را در آرایه‌های **ORIG** و **DEST** ذخیره کنید که در آن همانند شکل ۱۷ - ۱ (الف) آن دیس آرایه، شماره پرواز است.

(ب) داده‌های شکل ۱۵ - ۱ (ب) را در آرایه دو بعدی **FLIGHT** ذخیره کنید که در آن **FLIGHT[J,K]** شامل شماره پرواز از شهر **CITY[J]** به شهر **CITY[K]** است یا هرگاه هیچ پروازی صورت نگیرد شامل ۰ است، که این وضعیت در شکل ۱۷ - ۱ (ب) نشان داده شده است.

FLIGHT	1	2	3	4	5
1	0	0	0	715	0
2	0	0	701	0	711
3	0	702	0	708	0
4	0	0	0	0	718
5	713	712	705	717	0

	ORIG	DEST
701	2	3
702	3	2
703	0	0
704	0	0
705	5	3
706	0	0
:	:	:
715	1	4
716	0	0
717	5	4
718	4	5

(ب)

شکل ۱۷ - ۱

(الف)

مسئله ۱-۱۶ : فرض کنید یک شرکت هواپیمایی به n شهر پرواز عرضه می‌کند. معایب نمایش داده‌ها را به صورتی که در شکل ۱-۱۷ (الف) و شکل ۱-۱۷ (ب) آمده است شرح دهید.

حل : (الف) فرض کنید شماره‌های پرواز با فاصله خیلی زیاد از آدرس صفر، در حافظه ذخیره شده است یعنی فرض کنید نسبت شماره‌های پرواز به شماره خانه‌های حافظه عدد بسیار کوچکی مثلًاً تقریباً ۰.۰۵ باشد. در آن صورت است که گرفتن فضای حافظه اضافی ممکن است مقرون به صرفه نباشد.

(ب) فرض کنید نسبت شماره پروازهای δ به شماره خانه‌های حافظه n در آرایه **FLIGHT** عدد بسیار کوچکی است یعنی آرایه **FLIGHT** به گونه‌ای است که شامل تعداد بسیار زیادی صفر است. (به این گونه آرایه‌ها ماتریس خلوت یا تُنک می‌گویند). در آن صورت است که گرفتن فضای حافظه اضافی ممکن است مقرون به صرفه نباشد.

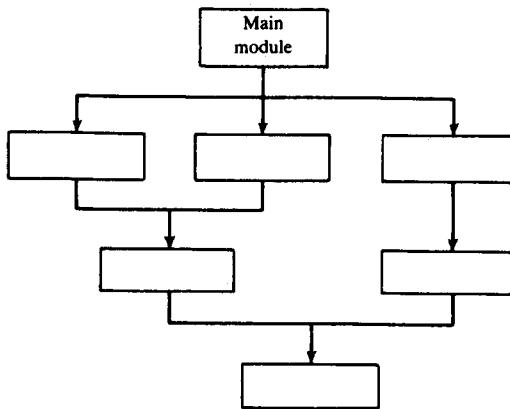
فصل ۲

مفاهیم مقدماتی

۱- مقدمه

به دست آوردن الگوریتم هایی برای ایجاد و پردازش ساختمان داده ها، ویژگی اصلی این کتاب را تشکیل می دهد. در این فصل با مثالهای ساده، فرمتهایی را توضیح می دهیم که به کمک آنها به ارائه الگوریتم ها می پردازیم. فرمتی که ما برای ارائه الگوریتم ها انتخاب کردہ ایم مشابه فرمتی است که دانلد کناث D.Knuth از آنها در کتاب معروفش، الگوریتم های بنیادی استفاده کرده است. اگرچه فرمت مورد استفاده ما مستقل از زبان برنامه نویسی است اما به اندازه کافی ساختیافته بوده، جزئیات آن بیان شده است و به سادگی می توان آنها را به یکی از زبان های برنامه نویسی نظیر **FORTRAN** ، **C** ، **PASCAL** یا **BASIC** ترجمه کرد. شایان ذکر است که برخی از الگوریتم های کتاب، در بخش مسایل به این زبانها نوشته شده است.

الگوریتم ها تا حدودی پیچیده هستند. درک پیاده سازی الگوریتم های پیچیده به صورت برنامه های کامپیوتری، زمانی با سادگی زیاد امکان پذیر است که بتوان این برنامه ها را به چند قطعه برنامه آن هم به صورت سلسله مراتبی، مشابه آنچه که در شکل ۱-۲ نشان داده شده است سازماندهی کرد. در یک چنین سازماندهی ای، در اول هر برنامه قطعه برنامه اصلی قرار دارد که الگوریتم را به صورت کلی شرح می دهد. قطعه برنامه اصلی چند زیر برنامه را احضار می کند که در آنها جزئیات بیشتری راجع



شکل ۱ - ۲ سلسله مراتب قطعه برنامه‌ها

به کار برنامه اصلی گنجانده شده است. هر زیر برنامه ممکن است به زیر برنامه‌های دیگری که دارای جزئیات بیشتری هستند رجوع کنند و الی آخر. سازماندهی یک برنامه به صورت چنین سلسله مراتبی از قطعه برنامه‌ها، معمولاً مستلزم استفاده از فلوچارت‌های مختلف و دستورهای منطقی است که عموماً در هر زیان برنامه‌نویسی ساختیافته وجود دارد. در این فصل چند فلوچارت و دستورهای منطقی از نظر گذرانده می‌شود.

فصل حاضر با شرح مختصری از چند تابع ریاضی شروع می‌شود که در مطالعه الگوریتم‌ها و به طور کلی در علوم کامپیوتر با آن‌ها مواجه می‌شوید و با بحث روی انواع مختلف متغیرهایی که می‌توانند در الگوریتم‌ها و برنامه‌ها ظاهر شوند به پایان می‌رسد.

مفهوم پیچیدگی یک الگوریتم نیز در این فصل گنجانده شده است. این معیار بسیار مهم در الگوریتم‌ها، ابزاری را در اختیار دانشجو قرار می‌دهد تا بتواند جوابهای مختلف الگوریتمی مسئله خاص نظری جستجو یا مرتب کردن اطلاعات را با هم مقایسه کند. مفهوم یک الگوریتم و پیچیدگی آن نه تنها در درس ساختمان داده‌ها، بلکه تقریباً در تمام شاخه‌های علوم کامپیوتر از اهمیت اساسی و بنیادی برخوردار است.

۲ - نمادگذاری ریاضی و تابعهای کامپیوترا

در این بخش چند تابع ریاضی معرفی می‌شود که به همراه نمادشان بکرات در تجزیه و تحلیل

الگوریتم‌ها و بهطور کلی در علم کامپیوتر ظاهر می‌شوند.

تابعهای کف و سقف

فرض کنید x یک عدد اعشاری باشد. در این صورت x بین دو عدد صحیح قرار دارد که کف و سقف x نامیده می‌شوند. به خصوص این که:

$[x]$ که کف x نامیده می‌شود، بزرگترین عدد صحیحی را نشان می‌دهد که بزرگتر از x نباشد.
 $\lceil x \rceil$ که سقف x نامیده می‌شود کوچکترین عدد صحیحی را نشان می‌دهد که کوچکتر از x نباشد.
اگر x یک عدد صحیح باشد، آنگاه $\lceil x \rceil = \lfloor x \rfloor$ در غیر اینصورت $\lceil x \rceil = \lfloor x \rfloor + 1$.

مثال ۲ - ۱

$$\begin{array}{llll} \lfloor 3.14 \rfloor = 3, & \lfloor \sqrt{5} \rfloor = 2, & \lceil -8.5 \rceil = -9, & \lceil 7 \rceil = 7 \\ \lceil 3.14 \rceil = 4, & \lceil \sqrt{5} \rceil = 3, & \lceil -8.5 \rceil = -8, & \lceil 7 \rceil = 7 \end{array}$$

تابع باقیمانده، حساب باقیمانده

فرض کنید K یک عدد صحیح دلخواه و M یک عدد صحیح مثبت باشد. آنگاه

$$k \pmod M$$

(بخوانید باقیمانده k بر M) هنگام تقسیم k بر M برای نمایش باقیمانده صحیح به کار می‌رود. به بیان دقیق‌تر، $K \pmod M$ برابر عدد صحیح منحصر بفرد r است طوری که

$$0 \leq r < M \quad \text{که در آن} \quad K = Mq + r$$

هرگاه K مثبت باشد تنها با تقسیم K بر M باقیمانده r به دست می‌آید. بنابراین:

$$25 \pmod 7 = 4, \quad 25 \pmod 5 = 0, \quad 35 \pmod 11 = 2, \quad 3 \pmod 8 = 3$$

در حالی که k منفی باشد مسئله ۲(ب) روشی را برای تعیین $(K \pmod M)$ نشان می‌دهد. از اصطلاح \pmod در رابطه ریاضی همنهشتی نیز استفاده می‌شود که به صورت زیر نمایش داده شده، تعریف می‌شود:
 $a \equiv b \pmod M$ اگر و فقط اگر M یک عامل $a - b$ باشد.

در اینجا M سنج یا هنگ نامیده می‌شود و $a \equiv b \pmod M$ به صورت " a همنهشت است با b به سنج M " خوانده می‌شود. ویژگی‌های زیر از رابطه همنهشتی اغلب مفید هستند:

$$a \pm M \equiv a \pmod M \quad \text{و} \quad 0 \equiv M \pmod M$$

منظور از حساب با باقیمانده M عملیات حسابی جمع، ضرب و تفریق است که در آن مقدار حسابی، جایگزین مقدار معادل آن در مجموعه

$$\{0, 1, 2, \dots, M-1\}$$

یا در مجموعه

$$\{0, 1, 2, \dots, M\}$$

می‌شود. برای مثال در حساب با باقیمانده 12 که گاهی اوقات حساب "ساعتی" نیز نامیده می‌شود داریم:

$$6 + 9 \equiv 3, \quad 7 \times 5 \equiv 11, \quad 1 - 5 \equiv 8, \quad 2 + 10 \equiv 0 \equiv 12$$

(استفاده از 0 یا M بستگی به کاربرد آن در محاسبه دارد.)

تابعهای مقدار صحیح و قدر مطلق

فرض کنید x یک عدد اعشاری دلخواه باشد. مقدار صحیح x که به صورت $\text{INT}(x)$ نوشته می‌شود عدد اعشاری x را با حذف (قطع) قسمت کسری آن به یک عدد صحیح تبدیل می‌کند. بنابراین

$$\text{INT}(3.14) = 3, \quad \text{INT}(\sqrt{5}) = 2, \quad \text{INT}(-8.5) = -8, \quad \text{INT}(7) = 7$$

ملاحظه می‌کنید بسته به اینکه x مثبت یا منفی باشد $\lfloor x \rfloor = \text{INT}(x)$ یا $\lceil x \rceil = \text{INT}(x)$.

قدر مطلق عدد اعشاری x که به صورت $\text{ABS}(x)$ یا $|x|$ نوشته می‌شود بنابراین تعریف برابر x است. از این رو $\text{ABS}(0) = 0$ و به ازای $x \neq 0$, $\text{ABS}(x) = x$, $\text{ABS}(x) = -x$. بسته به این که x مثبت یا منفی باشد. بنابراین

$$|-15| = 15, \quad |7| = 7, \quad |-3.33| = 3.33, \quad |4.44| = 4.44, \quad |-0.075| = 0.075$$

توجه دارید که $|x| = -x$ و به ازای $x \neq 0$, $|x|$ مثبت است.

نماد جمع؛ مجموعه

در اینجا نماد جمع Σ را معرفی می‌کنیم که علامت حرف یونانی سیگما است. دنباله a_1, a_2, a_3, \dots را در نظر بگیرید. آنگاه مجموع

$$a_m + a_{m-1} + \dots + a_n \quad \text{و} \quad a_1 + a_2 + \dots + a_n$$

به ترتیب به صورت زیر نمایش داده می‌شود:

$$\sum_{j=m}^n a_j \quad \text{و} \quad \sum_{j=1}^n a_j$$

حرف زد در بسطهای بالا، اندیس ظاهری یا متغیر ظاهری نامیده می‌شود. از حروف دیگری نظیر s , k , t نیز به عنوان متغیرهای ظاهری زیاد استفاده می‌شود.

مثال ۲-۲

$$\sum_{i=1}^n a_i b_i = a_1 b_1 + a_2 b_2 + \dots + a_n b_n$$

$$\sum_{j=2}^5 j^2 = 2^2 + 3^2 + 4^2 + 5^2 = 4 + 9 + 16 + 25 = 54$$

$$\sum_{j=1}^n j = 1 + 2 + \dots + n$$

اغلب اوقات، آخرین جمع مثال ۲ در کاربردها ظاهر می‌شود. مقدار آن برابر $2/(n(n+1))$ است. به عبارت دیگر:

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

به این ترتیب برای مثال:

$$1 + 2 + \cdots + 50 = \frac{50(51)}{2} = 1275$$

تابع فاکتوریل

ضرب اعداد صحیح مثبت از ۱ تا n و خود را با $n!$ نمایش داده، آن را $n!$ فاکتوریل می‌خوانند، به بیان دیگر:

$$n! = 1 \cdot 2 \cdot 3 \cdots (n-2)(n-1)n$$

بنابراین قرارداد $1! = 1$ تعریف می‌شود.

مثال ۲ - ۳

(الف)

$$2! = 1 \cdot 2 = 2; \quad 3! = 1 \cdot 2 \cdot 3 = 6; \quad 4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$$

(ب) به ازای n ، داریم $n! = |n(n-1)|$. از این رو

$$5! = 5 \cdot 4! = 5 \cdot 24 = 120; \quad 6! = 6 \cdot 5! = 6 \cdot 120 = 720$$

جایگشتها

یک جایگشت، یک مجموعه n عنصری از عناصر است که در آن، عناصر با یک نظم خاص مرتب شده باشند. برای مثال، جایگشتهای یک مجموعه با عناصر a, b, c به صورت زیر است:

$$abc, \quad acb, \quad bac, \quad bca, \quad cab, \quad cba$$

می‌توان ثابت کرد که: یک مجموعه n عنصری $n!$ جایگشت دارد. بنابراین یک مجموعه ۴ عنصری $= 4!$ جایگشت و یک مجموعه ۵ عنصری $= 5!$ جایگشت دارد و الى آخر.

توان رسانی و لگاریتم گیری

یادآوری می‌کنیم که تعریف‌های زیر برای توانهای صحیح a^m (که در آن m عدد صحیح مثبت است) همواره صادق است:

$$a^0 = 1, \quad a^{-m} = \frac{1}{a^m} \quad a^m = a \cdot a \cdots a \quad (\text{بار}) m$$

عمل توان رسانی تا جایی که شامل تمام اعداد گویا می‌شود قابل تعمیم است. بنابراین برای هر عدد گویای n داریم:

$$a^{m/n} = \sqrt[n]{a^m} = (\sqrt[n]{a})^m$$

مثال

$$2^4 = 16, \quad 2^{-4} = \frac{1}{2^4} = \frac{1}{16}, \quad 125^{2/3} = 5^2 = 25$$

در حقیقت، عمل توان رسانی تا جایی که شامل تمام اعداد حقیقی می‌شود نیز قابل تعمیم است. بنابراین تعریف برای هر عدد حقیقی x

$$\text{که در آن } r \text{ یک عدد گویا است} \quad a^x = \lim_{r \rightarrow x} a^r$$

برطبق آن، به ازای تمام اعداد حقیقی، تابع نمایی $a^x = f(x)$ تعریف می‌شود.
لگاریتم با توان به صورت زیر در ارتباط است. فرض کنید b یک عدد صحیح مثبت باشد. لگاریتم عدد مثبت و دلخواه x در مبنای b که به صورت زیر نوشته می‌شود.

$$\log_b x$$

برابر توانی است که b بایستی به آن توان برسد تا x بدست آید، به بیان دیگر

$$b^y = x \quad \text{و} \quad y = \log_b x$$

دو رابطه معادل هستند. بنابراین

$10^2 = 100$	چون	$\log_{10} 100 = 2$	چون	$\log_2 8 = 3$
$10^{-3} = 0.001$	چون	$\log_{10} 0.001 = -3$	چون	$\log_2 64 = 6$

علاوه بر این برای هر مبنای دلخواه b :

$$\begin{aligned} b^0 &= 1 & \log_b 1 &= 0 \\ b^1 &= b & \log_b b &= 1 \end{aligned}$$

لگاریتم اعداد منفی و لگاریتم صفر تعریف نشده است.

می‌توان تابعهای نمایی و لگاریتمی

$$g(x) = \log_b x \quad \text{و} \quad f(x) = b^x$$

را به عنوان توابع معکوس یکدیگر، مورد توجه قرار داد. بهموجب آن، نمودارهای این دو تابع، معکوس یکدیگر هستند. مسأله ۱۵-۲ را ببینید.

اغلب مقدار لگاریتم اعداد به صورت مقادیر تقریبی بیان می‌شوند. برای مثال با استفاده از جدول لگاریتمی، ماشین حساب یا کامپیوتر

$$\log_e 40 = 3.6889 \quad \text{و} \quad \log_{10} 300 = 2.4771$$

را به عنوان جوابهای تقریبی بدست می‌آوریم. در اینجا $e = 2.71828\dots$ لگاریتم‌هایی که در کار ما از اهمیت زیادی برخوردارند عبارتند از: لگاریتم در مبنای ۱۰ که لگاریتم معمولی نام دارد، لگاریتم در مبنای e که لگاریتم طبیعی نام دارد و لگاریتم در مبنای ۲ که لگاریتم دودویی نام دارد.

در بعضی از کتابها به جای \log_e^x می‌نویسند $\ln x$ و به جای \log_2^x می‌نویسند $\lg x$ یا $\log x$. این کتاب که درباره درس ساختمان داده‌هاست اساساً با لگاریتم‌های دودویی اعداد و عبارتها سر و کار دارد.

بنابراین منظور ما از اصطلاح $\log x$ در این کتاب $\log_2 x$ است مگر آن که خلاف آن بیان شود. اغلب ما فقط به مقادیر کف و سقف لگاریتم دودویی نیازمندیم. با ملاحظه توانهای ۲ می‌توانیم به این مقادیر دست یابیم. برای مثال:

$$\begin{array}{lll} 2^7 = 128 & 2^6 = 64 & \text{چون} \\ 2^9 = 1024 & 2^8 = 512 & \text{چون} \end{array} \quad \begin{array}{l} \lceil \log_2 100 \rceil = 6 \\ \lceil \log_2 1000 \rceil = 9 \end{array}$$

و الى آخر.

۳- ۲ نمایش الگوریتمی

به بیان شهودی، یک الگوریتم یک لیست متناهی و مرحله به مرحله از دستورات خوش تعریف است که برای حل یک مسئله خاص درنظر گرفته می‌شود. تعریف رسمی الگوریتم که از مفهوم ماشین تیورینگ بهره می‌گیرد بسیار پیچیده و در محدوده مطالب این درس قرار ندارد. این بخش فرمتی را توضیح می‌دهد که برای نمایش الگوریتم‌ها در سراسر کتاب از آن استفاده می‌شود بهتر دیدیم این نمایش الگوریتمی را با ارائه چند مثال توضیح دهیم.

مثال ۴-۲

آرایه **DATA** با مقادیر عددی در حافظه ذخیره شده است. می‌خواهیم **LOC** مکان و **MAX** مقدار بزرگترین عنصر آرایه **DATA** را پیدا کنیم. درباره **DATA** هیچ اطلاع دیگری نداریم. یک راه برای حل این مسئله به قرار زیر است:

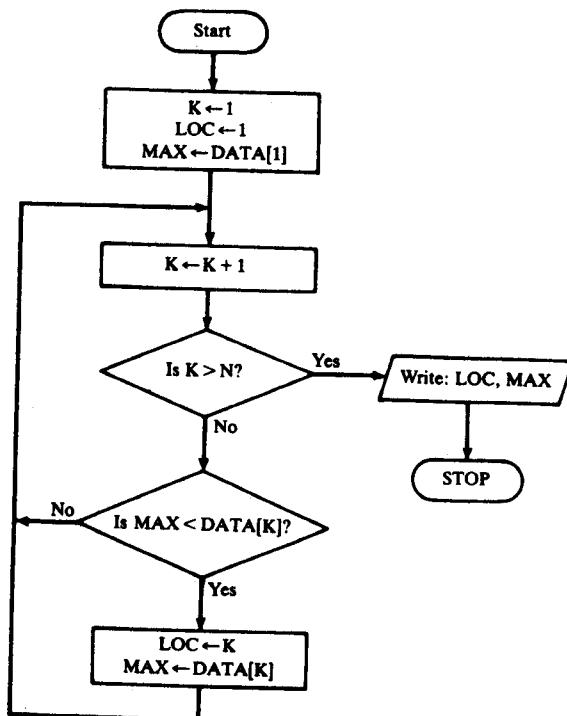
در آغاز، کار را با $LOC = 1$ و $MAX = DATA[1]$ شروع می‌کنیم. آنگاه **MAX** را با هر یک از عناصر بعدی **DATA** مقایسه می‌کنیم. اگر $DATA[K]$ بزرگتر از **MAX** باشد آنگاه **LOC** و **MAX** را تازه می‌کنیم طوری که $LOC = K$ و $MAX = DATA[K]$ شود. آخرین مقداری که در **LOC** و **MAX** قرار می‌گیرد مکان و مقدار بزرگترین عنصر **DATA** است.

نمایش رسمی این الگوریتم به صورت زیر است. فلوچارت آن در شکل ۲-۲ رسم شده است.

Algorithm 2.1: (Largest Element in Array) A nonempty array DATA with N numerical values is given. This algorithm finds the location LOC and the value MAX of the largest element of DATA. The variable K is used as a counter.

- Step 1. [Initialize.] Set $K := 1$, $LOC := 1$ and $MAX := DATA[1]$.
- Step 2. [Increment counter.] Set $K := K + 1$.
- Step 3. [Test counter.] If $K > N$, then:
 Write: LOC, MAX, and Exit.
- Step 4. [Compare and update.] If $MAX < DATA[K]$, then:
 Set $LOC := K$ and $MAX := DATA[K]$.
- Step 5. [Repeat loop.] Go to Step 2.

فرمت بالا برای نمایش رسمی یک الگوریتم از دو قسمت تشکیل شده است. در قسمت اول، متنی قرار دارد که اطلاعاتی راجع به هدف الگوریتم به دست می‌دهد و متغیرهایی را که در الگوریتم از آنها استفاده می‌شود معرفی می‌کند و لیست داده‌های ورودی را ارائه می‌دهد. قسمت دوم این الگوریتم شامل لیست مرحله‌هایی است که بایستی به ترتیب اجرا شوند.



شکل ۲ - ۲ فلوچارت الگوریتم ۱

در زیر چند قرارداد را که ما در ارائه الگوریتم‌ها از آن استفاده می‌کنیم به صورت خلاصه شده، می‌آوریم. برخی از دستورهای کنترلی در بخش بعد به کار می‌آیند.

شماره شناسایی

به هر الگوریتم به صورت زیر یک شماره شناسایی نسبت داده شده است: که منظور از الگوریتم ۳-۴ الگوریتم سوم از فصل ۴ و منظور از الگوریتم ۳-۵ الگوریتم مسأله ۵-۳ از فصل ۵ است. توجه دارید که حرف P مبین آن است که این الگوریتم در یک مسأله Problem عنوان شده است.

مرحله‌ها، کنترل، خروج از الگوریتم

مراحل یک الگوریتم یکی پس از دیگری اجرا می‌شوند. این مراحل با مرحله ۱ شروع می‌شود، مگر آن که خلاف آن گفته شود. دستور "برو به n، یا Go To n" کنترل کار را به دستور مرحله n از الگوریتم منتقل می‌کند. مثلاً در الگوریتم ۱-۲ مرحله ۵، کنترل اجرا را به مرحله ۲ منتقل می‌کند. به بیان کلی تر، دستورهای GO با بکارگیری تعدادی از دستورهای کنترلی که در بخش بعد مطرح می‌شوند عملأ و در ظاهر حذف می‌شوند. اگر در یک مرحله، از چند دستور استفاده شود نظر

`Set K := 1, LOC := 1, MAX := DATA[1].`

آنگاه این دستورها از چپ به راست اجرا می‌شوند.

الگوریتم زمانی به پایان می‌رسد که دستور خروج از الگوریتم

Exit

مشاهده شود به عبارت دیگر دستور Exit به معنی پایان الگوریتم است. این دستور مشابه دستور STOP در FORTRAN و فلوچارت‌ها است.

توضیحات Comments

هر مرحله ممکن است شامل یک توضیح در داخل کروشه باشد که هدف اصلی آن مرحله را بیان می‌کند. توضیح، معمولاً در آغاز یا پایان یک مرحله داده می‌شود.

اسامي متغيرها

در نام متغیرها نظیر MAX و DATA از حروف بزرگ استفاده می‌کنیم. از متغیرهای تک حرفی که در الگوریتم با حروف بزرگ نوشته شده‌اند مانند K و N، به عنوان شمارنده یا انديس استفاده می‌کنیم حتی

اگر برای این متغیرها در تجزیه و تحلیل‌ها یا عبارات ریاضی از حروف کوچک نظیر `k` و `n` استفاده شود، برای یادآوری، به بحث مربوط به نمادهایی با حروف کج یا حروف کرجک بخش ۳-۱ از فصل ۱ تحت عنوان آرایه‌ها مراجعه کنید.

دستورهای جایگزینی

برای دستورهای جایگزینی همانگونه که در زبان **PASCAL** رایج است از نماد کولن و تساوی `=` استفاده می‌شود. برای مثال:

```
Max := DATA[1]
```

مقدار `DATA[1]` را در متغیر **MAX** جایگزین می‌کند. بعضی از کتابها برای عمل جایگزینی از نماد پیکان `←` یا علامت تساوی `=` استفاده می‌کنند.

ورودی و خروجی

داده‌ها به کمک دستور **Read** به صورت زیر از ورودی دریافت شده، در متغیرها جایگزین می‌شوند:

نام چند متغیر : **Read**

به طور مشابه، پیغامهایی که در داخل علامت نقل قول یا کوتیشن قرار دارند به همراه داده‌های داخل

متغیرها، به کمک دستور **Write** یا **Print** به صورت زیر در خروجی چاپ می‌شوند:

نام چند متغیر یا / و یا پیغام : **Write**

زیربرنامه

اصطلاح "زیربرنامه" برای قطعه الگوریتم مستقلی به کار می‌رود که بخشی از یک مسئله را حل می‌کند. استفاده از واژه "زیربرنامه" یا "قطعه برنامه" به جای "الگوریتم" برای یک مسئله داده شده، تنها یک موضوع سلیقه‌ای است. به بیان کلی تر، واژه الگوریتم برای حل یک مسئله با نگرش کلی به کار می‌رود اما اصطلاح "زیربرنامه" برای توصیف و حل یک بخش از الگوریتم یا زیرالگوریتم مورد استفاده قرار می‌گیرد که در بخش ۶-۲ به تفصیل مورد بحث و بررسی قرار خواهد گرفت.

۴-۲ دستورهای کنترلی

الگوریتم‌ها و برنامه‌های کامپیوتری معادل آنها زمانی بهتر و ساده‌تر درک می‌شوند که در آنها اساساً از چند زیربرنامه و سه نوع منطق یا جریان کنترلی به شرح زیر استفاده شود:

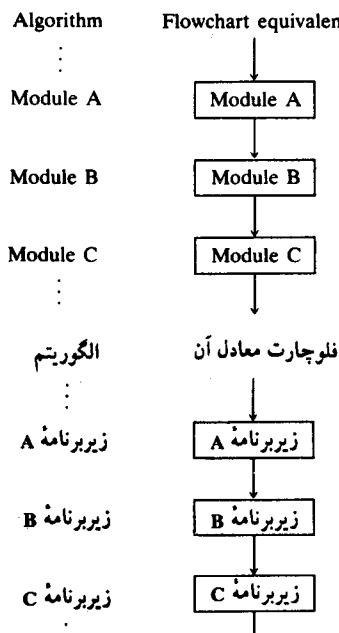
- (۱) منطق توالی یا جریان متواالی اجرای دستورات
- (۲) منطق انتخاب یا دستورات شرطی
- (۳) منطق تکرار یا جریان تکراری

سه منطق بالا در زیر، مورد بحث و بررسی قرار می‌گیرد و در هر حالت فلوچارت مربوط به آن رسم می‌شود.

منطق توالی (جریان متواالی اجرای دستورات)

منطق توالی قبل از مورد بررسی قرار گرفت بجز در مواردی که بعضی از دستورات، اجرای پشت سرهم یا متواالی دستورات را برهم می‌زنند. تمام دستورات برنامه‌ها و زیربرنامه‌ها به صورت متواالی و پشت‌سرهم اجرا می‌شوند. توالی دستورات ممکن است با استفاده از شماره مراحل به صورت صریح بیان شود و یا براساس ترتیبی که در آن دستورات زیربرنامه‌ها یا برنامه‌ها نوشته می‌شوند به صورت ضمنی باشد. (شکل ۳-۲ را ببینید).

در بیشتر پردازشها، حتی در مسایل پیچیده، عموماً از این الگوی مقدماتی جریان کنترلی، تبعیت می‌شود.



شکل ۳-۲. منطق توالی

منطق انتخاب (دستورات شرطی)

منطق انتخاب از تعدادی شرط استفاده می‌کند که منتهی به انتخاب یک زیربرنامه از میان چند زیربرنامه می‌شود. دستوراتی که ما برای پیاده‌سازی این منطق به کار می‌گیریم، دستورات شرطی یا دستورات IF است. برای روشی بیشتر مطلب، غالباً در پایان این گونه دستورات عبارت [End of If Structure. If] یا [پایان دستور IF].

یا با عبارت معادل آن نوشته می‌شود.

دستورات شرطی به سه نوع مختلف تقسیم می‌شوند که هر یک از این دستورات به طور جداگانه مورد بحث قرار می‌گیرد.

(۱) حالت یک وضعیتی: این دستور به صورت زیر است:

IF ، شرط THEN :

If condition, then:

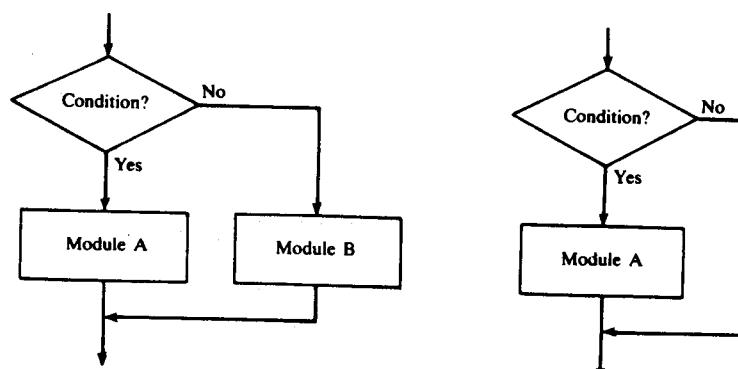
[زیربرنامه A]

[Module A]

[IF پایان دستور]

[End of If structure.]

منطق این دستور در فلوچارت شکل ۲-۴ (الف) نشان داده شده است. اگر شرط برقرار باشد، آنگاه زیربرنامه A اجرا می‌شود که خود می‌تواند شامل یک یا چند دستور باشد، در غیر اینصورت زیربرنامه A نادیده گرفته می‌شود و کنترل اجرای دستورات به مرحله بعدی الگوریتم داده می‌شود.



(ب) حالت دو وضعیتی

(الف) حالت یک وضعیتی

شکل ۲-۴

(۲) حالت دووضعیتی: این دستور به صورت زیر است:

IF ، شرط THEN :

If condition, then:

<p>[زیربرنامه A]</p> <p>ELSE :</p> <p>[زیربرنامه B]</p> <p>[IF] پایان دستور</p>	<p>با</p>	<p>[Module A]</p> <p>Else:</p> <p>[Module B]</p> <p>[End of If structure.]</p>
---	-----------	--

منطق این دستور در فلوچارت شکل ۴-۲ (ب) نشان داده شده است. همانگونه که فلوچارت بیان می‌کند اگر شرط برقرار باشد آنگاه زیربرنامه A اجرا می‌شود، در غیر اینصورت اجرا به زیربرنامه B داده می‌شود.

(۳) حالت چندوضعیتی :

<p>IF (۱) , شرط :</p> <p>[زیربرنامه A₁]</p> <p>ELSE IF (۲) , شرط :</p> <p>[زیربرنامه A₂]</p> <p>:</p> <p>Else IF (M) , شرط :</p> <p>[زیربرنامه A_M]</p> <p>ELSE :</p> <p>[زیربرنامه B]</p> <p>[IF] پایان دستور</p>	<p>If condition(1), then:</p> <p>[Module A₁]</p> <p>Else if condition(2), then:</p> <p>[Module A₂]</p> <p>:</p> <p>Else if condition(M), then:</p> <p>[Module A_M]</p> <p>Else:</p> <p>[Module B]</p> <p>[End of If structure.]</p>
---	---

منطق این دستور به گونه‌ای است که تنها اجازه اجرای یک زیربرنامه را می‌دهد. بهویژه این‌که، یا زیربرنامه پائین شرط اول در صورت برقراربودن اجرا می‌شود یا زیربرنامه‌ای که پائین آخرین دستور Else قرار دارد اجرا می‌شود. در عمل بهندرت اتفاق می‌افتد که بیش از سه شرط مداخل در زیربرنامه وجود داشته باشد.

مثال ۵-۲

ریشه‌های معادله درجه دوم

$$ax^2 + bx + c = 0$$

که در آن $a \neq 0$ ، با استفاده از دستور زیر به دست می‌آید:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

مقدار $D = b^2 - 4ac$ می‌بین **Discriminant** معادله درجه دوم نامیده می‌شود. اگر D منفی باشد، آنگاه معادله ریشه حقیقی ندارد. اگر $D = 0$ باشد، آنگاه معادله تنها یک ریشه مضاعف (دو ریشه مساوی) دارد. اگر D مثبت باشد، دستور بالا دو ریشه حقیقی متمایز را به دست می‌دهد. الگوریتم زیر را پیدا می‌کند.

Algorithm 2.2: (Quadratic Equation) This algorithm inputs the coefficients A, B, C of a quadratic equation and outputs the real solutions, if any.

```

Step 1. Read: A, B, C.
Step 2. Set D := B2 - 4AC.
Step 3. If D > 0, then:
        (a) Set X1 := (-B + √D)/2A and X2 := (-B - √D)/2A.
        (b) Write: X1, X2.
Else if D = 0, then:
        (a) Set X := -B/2A.
        (b) Write: 'UNIQUE SOLUTION', X.
Else:
        Write: 'NO REAL SOLUTIONS'.
[End of If structure.]
Step 4. Exit.

```

توجه کنید: ملاحظه می‌کنید که در مرحله 3 از الگوریتم 2.2 سه شرط دوبه دو متقابل وجود دارد که اجرای هر یک از آنها بستگی به آن دارد که آیا D مثبت است یا صفر یا منفی. در چنین وضعیتی، می‌توان به طور متناوب حالت‌های مختلف را به صورت زیر ارائه داد:

Step 3. (1) If $D > 0$, then:

.....

(2) If $D = 0$, then:

.....

(3) If $D < 0$, then:

.....

بیان دستور **If** مرحله 3 از الگوریتم 2.2، مشابه استفاده از دستور CASE در زبان PASCAL است.

منطق تکرار (جريان تکراری)

نوع سوم از منطق یا جریان کنترلی به یکی از دو نوع دستور حلقة تکرار زیر مربوط می‌شود. هر یک از این حلقه‌ها با یک دستور **Repeat** شروع می‌شود، بدنبال آن یک زیربرنامه قرار می‌گیرد که بدنۀ حلقة نامیده می‌شود. برای روشنی بیشتر مطلب، در پایان این گونه دستورات عبارت

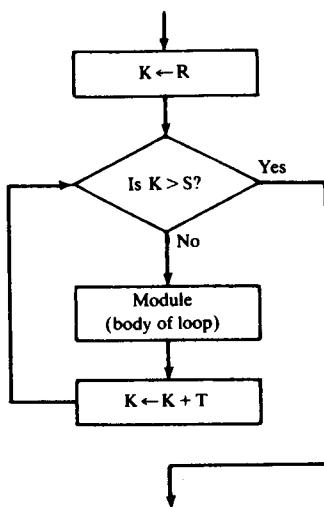
[End of loop.]

یا عبارت معادل آن نوشته می‌شود.

هر یک از این دو نوع دستور حلقه تکرار بطور جداگانه مورد بحث قرار می‌گیرد.
حلقه تکرار **Repeat – For** از یک متغیر شاخص **Index** نظیر K برای کنترل مراحل تکرار حلقه استفاده می‌کند. این حلقه معمولاً به صورت زیر است:

Repeat for $K = R$ to S by T :
 [Module]
 [End of loop.]

منطق این دستور در فلوچارت شکل ۲-۵ (الف) نشان داده شده است. در اینجا R مقدار اولیه و S مقدار پایانی یا مقدار آزمایشی حلقه و T نمو حلقه نام دارد.



شکل ۲-۵ (الف)

Repeat – For
دستور

ملاحظه می‌کنید که بدنه این حلقه ابتدا به ازای $R = K$ ، بعد به ازای $R = K + T$ و بدنال آن به ازای $R = K + 2T$ و غیره اجرا می‌شود. حلقه وقتی پایان می‌یابد که $K > S$ شود. این فلوچارت فرض می‌کند که نمو T مثبت است، اگر T منفی باشد و درنتیجه مقدار K دائمًا کاهش می‌یابد آنگاه حلقه وقتی پایان می‌یابد که $K < S$ باشد.

حلقه **Repeat – While** از یک شرط برای کنترل مراحل تکرار حلقه استفاده می‌کند. این حلقه معمولاً

به صورت زیر است:

Repeat While شرط :

[زیربرنامه]

یا

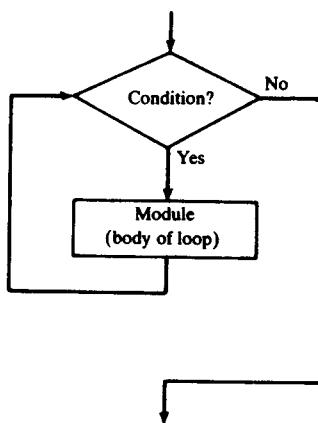
Repeat while condition:

[Module]

[پایان حلقه.]

[End of loop.]

منطق این دستور در فلوچارت شکل ۲-۵ (ب) نشان داده شده است.



شکل ۲-۵ (ب)

Repeat — While دستور

ملاحظه می‌کنید که اجرای این حلقه تا زمانی ادامه می‌یابد که شرط برقرار باشد. تأکید می‌کنیم که بایستی قبل از این دستور، دستوری وجود داشته باشد تا به متغیر کنترل کننده شرط حلقه، مقدار اولیه بدهد و برای اینکه اجرای حلقه بتواند درنهایت متوقف شود، بایستی در بدنه حلقه دستوری وجود داشته باشد تا شرط را تغییر دهد.

مثال ۶-۲

الگوریتم 2.1 بجای دستور **GO TO**، با استفاده از یک حلقه **Repeat — While** به صورت زیر نوشته می‌شود.

Algorithm 2.3: (Largest Element in Array) Given a nonempty array DATA with N numerical values, this algorithm finds the location LOC and the value MAX of the largest element of DATA.

1. [Initialize.] Set $K := 1$, $LOC := 1$ and $MAX := DATA[1]$.
2. Repeat Steps 3 and 4 while $K \leq N$:
3. If $MAX < DATA[K]$, then:
 Set $LOC := K$ and $MAX := DATA[K]$.
 [End of If structure.]
4. Set $K := K + 1$.
 [End of Step 2 loop.]
5. Write: LOC , MAX .
6. Exit.

الگوریتم 2.3 برخی از ویژگیهای دیگر آن را بیان می‌کند. معمولاً کلمه "مرحله" در الگوریتم حذف می‌شود و سعی می‌شود به جای دستورات **To** از دستورات **Repeat** استفاده شود. دستور **Repeat** می‌تواند به صورت صریح مراحلی را نشان دهد که بدئه حلقه را تشکیل می‌دهند. دستور "پایان حلقه زیر برنامه‌های درون دستورات منطقی معمولاً برای سادگی بیشتر در خواندن آنها، به صورت پله‌ای یا تورفته نوشته می‌شوند. این مطلب تأکیدی دیگر بر نوشتمند دستورات معمولی حلقه‌ها به صورت پله‌ای در زبانهای برنامه‌نویسی ساختیافه است.

هر نماد یا قرارداد جدید دیگری که در این کتاب آمده باشد یا بروشنا و واضح است یا هنگام استفاده از آن، توضیح داده می‌شود.

۲- پیچیدگی الگوریتم‌ها

تجزیه و تحلیل الگوریتم‌ها، وظيفة اصلی علم کامپیوتر است. برای مقایسه الگوریتم‌ها، باید معیارهایی برای اندازه‌گیری کارآیی الگوریتم در اختیار داشته باشیم. این موضوع مهم، و قابل توجه در این بخش از کتاب شرح داده می‌شود.

فرض کنید M یک الگوریتم و n تعداد داده‌های ورودی آن باشد. دو معیار اصلی برای الگوریتم M ، زمان و مقدار حافظه‌ای است که الگوریتم M از آنها استفاده می‌کند. زمان اجرا با شمارش و محاسبه تعداد عملیات کلیدی داخل الگوریتم اندازه‌گرفته می‌شود به عنوان مثال در الگوریتم‌های مرتب‌کردن و جستجوی اطلاعات، تعداد مقایسه‌ها عمل کلیدی الگوریتم است. به این علت که عملیات کلیدی چنان تعریف می‌شوند تا زمان مربوط به عملیات دیگر خیلی کوچکتر یا حداقل متناسب با زمان مربوط به عملیات کلیدی باشد. حافظه، با شمارش و محاسبه مانگریم خانه حافظه موردنیاز الگوریتم اندازه‌گیری می‌شود.

پیچیدگی الگوریتم M ، تابع $f(n)$ است که زمان اجرا و/یا حافظه موردنیاز الگوریتم را بر حسب n تعداد داده‌های داده‌پروردی به دست می‌دهد. اغلب، حافظه موردنیاز یک الگوریتم تنها مضری از n یا تعداد داده‌های ورودی است. بر طبق آن، منظور از اصطلاح "پیچیدگی" زمان اجرای الگوریتم است مگر آن که خلاف آن بیان شود یا بکار آید.

مثال زیر نشان می‌دهد تابع $f(n)$ که زمان اجرای یک الگوریتم را به دست می‌دهد نه تنها به n تعداد داده‌های ورودی بستگی دارد بلکه به نوع داده‌ای نیز وابسته است.

مثال ۲ - ۷

فرض کنید یک داستان کوتاه به نام TEXT در اختیار داریم و بخواهیم در TEXT اولین کلمه سه‌حرفی به نام W را با جستجو پیدا کنیم. اگر W کلمه سه‌حرفی the باشد، آنگاه به نظر می‌رسد که W در ابتدای داستان TEXT وجود داشته باشد از این‌رو $f(n)$ یک عدد کوچک خواهد بود. از طرف دیگر. اگر W کلمه سه‌حرفی zoo باشد آنگاه ممکن است این کلمه اصلاً در TEXT وجود نداشته باشد، درنتیجه $f(n)$ بسیار بزرگ خواهد بود.

بحث بالا منتهی به این سؤال می‌شود که تابع پیچیدگی $f(n)$ را در حالت‌های خاص پیدا کنیم. دو حالتی که معمولاً در نظریه پیچیدگی الگوریتم‌ها موردنوجه قرار می‌گیرد عبارتند از:

(۱) بدترین حالت: که ماگزیم مقدار $f(n)$ برای هر ورودی ممکن است.

(۲) حالت میانگین: که مقدار انتظاری $f(n)$ یا امید ریاضی E است.

گاهی اوقات، حداقل مقدار ممکن $f(n)$ نیز مدنظر قرار می‌گیرد که بهترین حالت نامیده می‌شود. تجزیه و تحلیل حالت میانگین فرض می‌کند داده‌های ورودی از یک توزیع احتمال مشخص پیروی می‌کند. در یک چنین فرضی ممکن است تمام جایگشت‌های ممکن مجموعه داده‌های ورودی با احتمال مساوی باشد. حالت میانگین از مفهوم زیر نیز در نظریه احتمالات استفاده می‌کند. فرض کنید اعداد n_1, n_2, \dots, n_k به ترتیب با احتمال P_1, P_2, \dots, P_k ظاهر شوند. آنگاه امید ریاضی یا مقدار میانگین E از رابطه زیر به دست می‌آید:

$$E = n_1 p_1 + n_2 p_2 + \cdots + n_k p_k$$

این ایده‌ها در مثال زیر نشان داده شده است!

مثال ۲ - ۸: جستجوی خطی

فرض کنید آرایه خطی DATA دارای n عنصر است و داده مشخص ITEM نیز داده شده است.

می خواهیم در صورت وجود ITEM در آرایه DATA ، LOC مکان آن را پیدا کنیم و در صورت عدم وجود، پیغامی نظری $LOC = 0$ بدهیم، تا نشان داده شود ITEM در DATA وجود ندارد. الگوریتم جستجوی خطی، این مسئله را از طریق مقایسه ITEM با تک تک عناصر آرایه DATA حل می کند. به عبارت دیگر، ITEM با [2] DATA[1] و الى آخر مقایسه می شود تا LOC پیدا شود طوری که $ITEM = DATA[LOC]$ باشد. نمایش رسمی این الگوریتم به قرار زیر است:

Algorithm 2.4: (Linear Search) A linear array DATA with N elements and a specific ITEM of information are given. This algorithm finds the location LOC of ITEM in the array DATA or sets LOC = 0.

1. [Initialize] Set $K := 1$ and $LOC := 0$.
2. Repeat Steps 3 and 4 while $LOC = 0$ and $K \leq N$.
 3. If $ITEM = DATA[K]$, then: Set $LOC := K$.
 4. Set $K := K + 1$. [Increments counter.]
5. [End of Step 2 loop.]
6. [Successful?]
 - If $LOC = 0$, then:
Write: ITEM is not in the array DATA.
 - Else:
Write: LOC is the location of ITEM.
7. [End of If structure.]
8. Exit.

پیچیدگی الگوریتم جستجو با C ، تعداد مقایسه های انجام شده بین ITEM و DATA[K] به دست می آید. $C(n)$ را در بدترین حالت و حالت میانگین تعیین می کنیم.

بدترین حالت

واضح است که بدترین حالت وقتی اتفاق می افتد که ITEM آخرین عنصر آرایه DATA باشد یا اصلاً در آرایه وجود نداشته باشد. در هر دو حالت داریم :

$$C(n) = n$$

بنابراین $n = C(n)$ پیچیدگی بدترین حالت الگوریتم جستجوی خطی است.

حالت میانگین

در اینجا فرض می کنیم که ITEM در آرایه DATA و با احتمال مساوی در هر مکانی از آرایه وجود دارد. به موجب آن، تعداد مقایسه ها را می توان هر یک از اعداد $1, 2, 3, \dots, n$ دانست و احتمال وقوع هر عدد برابر $P = 1/n$ است. آنگاه

$$\begin{aligned}
 C(n) &= 1 \cdot \frac{1}{n} + 2 \cdot \frac{1}{n} + \cdots + n \cdot \frac{1}{n} \\
 &= (1 + 2 + \cdots + n) \cdot \frac{1}{n} \\
 &= \frac{n(n+1)}{2} \cdot \frac{1}{n} = \frac{n+1}{2}
 \end{aligned}$$

این نتیجه با احساس شهودی ما نیز تطابق دارد که میانگین تعداد مقایسه‌های موردنیاز برای تعیین مکان ITEM تقریباً برابر نصف تعداد عناصر لیست خطا DATA است.

توجه کنید: پیچیدگی حالت میانگین یک الگوریتم، معمولاً بسیار پیچیده‌تر از تجزیه و تحلیل پیچیدگی بدترین حالت است. علاوه بر این، توزیع احتمالی که برای حالت میانگین درنظر گرفته می‌شود عملاً در وضعیتها حقیقی ممکن نیست بکار گرفته شود. به موجب آن، منظور از پیچیدگی یک الگوریتم، تابعی است که زمان اجرای الگوریتم را در بدترین حالت بر حسب تعداد داده‌های ورودی به دست می‌دهد، مگر آنکه خلاف آن بیان شود یا بکار آید. این فرض آنقدرها که فکر می‌کنیم قوی نیست، چون پیچیدگی حالت میانگین در مورد بسیاری از الگوریتم‌ها متناسب با بدترین حالت است.

آهنگ (نرخ) رشد، نماد O بزرگ (Big O)

فرض کنید M یک الگوریتم و n تعداد داده‌های ورودی آن باشد. واضح است که پیچیدگی $f(n)$ الگوریتم M با زیاد شدن تعداد داده‌های ورودی n افزایش می‌یابد. معمولاً ما مایل به تعیین آن، آهنگ افزایش $f(n)$ هستیم. این کار معمولاً از مقایسه $f(n)$ با چند تابع استاندارد نظری

$$\log_2 n, \quad n, \quad n \log_2 n, \quad n^2, \quad n^3, \quad 2^n$$

حاصل می‌شود. آهنگهای رشد این توابع استاندارد، در شکل ۶ نشان داده شده است که مقادیر تقریبی آنها را، به ازای چند مقدار معین n به دست می‌دهد.

$n \backslash g(n)$	$\log n$	n	$n \log n$	n^2	n^3	2^n
5	3	5	15	25	125	32
10	4	10	40	100	10 ³	10 ³
100	7	100	700	10 ⁴	10 ⁶	10 ³⁰
1000	10	10 ³	10 ⁴	10 ⁶	10 ⁹	10 ³⁰⁰

شکل ۶. آهنگ رشد چند تابع استاندارد

ملاحظه می شود که تابعها، به ترتیب آهنگ رشدشان ارائه شده اند. تابع لگاریتمی $\log_2 n$ رشد نسبتاً کنندی دارد، تابع نمایی e^n رشد نسبتاً تندی دارد و تابع چندجمله‌ای n^c با توجه به مقدار توان c رشد می‌کند. یک راه برای مقایسه تابع $f(n)$ با این تابعهای استاندارد، استفاده از نماد تابعی O است که به صورت زیر تعریف می‌شود:

فرض کنید $f(n)$ و $g(n)$ دو تابعی باشند که بر روی اعداد صحیح مثبت تعریف شده، و دارای این خاصیت هستند که $f(n)$ به ازای تمام مقادیر n توسط مضربی از $g(n)$ محدود شده است. به عبارت دیگر، فرض کنید که عدد صحیح مثبت n_0 و عدد مثبت M وجود دارد به گونه‌ای که به ازای تمام $n > n_0$ داریم:

$$|f(n)| \leq M|g(n)|$$

در آن صورت می‌توان نوشت:

$$f(n) = O(g(n))$$

که خوانده می‌شود $f(n)$ از مرتبه $g(n)$ است. برای چندجمله‌ای $p(n)$ که از درجه m است در مسأله ۲-۱۰ نشان می‌دهیم $P(n) = O(n^m)$ ، به عنوان مثال

$$8n^3 - 576n^2 + 832n - 248 = O(n^3)$$

همچنین می‌توان نوشت:

$$f(n) - h(n) = O(g(n)) \quad f(n) = h(n) + O(g(n))$$

(این نماد "O" بزرگ یا "Big O" نام دارد چون $f(n) = O(g(n))$ معنی کاملاً متفاوتی دارد). برای این که نشان دهیم این نمادگذاری چقدر مناسب است پیچیدگی چند الگوریتم معروف جستجو و مرتب کردن را ارائه می‌دهیم:

(الف) جستجوی خطی $O(n)$

(ب) جستجوی دودویی $O(\log n)$

(ج) مرتب کردن حبابی $O(n^2)$

(د) مرتب کردن با ادغام $O(n\log n)$

این نتایجه‌ها در فصل ۹ در مبحث مرتب کردن و جستجوی اطلاعات به تفصیل مورد بررسی قرار می‌گیرد.

۶- ۲ زیرالگوریتم‌ها

یک زیرالگوریتم یک قطعه برنامه الگوریتمی کامل و به طور مستقل تعریف شده، است که به وسیله الگوریتم اصلی یا الگوریتم دیگری مورد استفاده قرار می‌گیرد (احضار یا صدازده می‌شود). یک

زیرالگوریتم، مقادیر خود را که آرگومان نامیده می‌شوند از الگوریتم اصلی یا فراخواننده دریافت می‌کند، محاسبات لازم را انجام می‌دهد و آنگاه نتیجه را به الگوریتم فراخواننده بر می‌گرداند. زیرالگوریتم به این علت به صورت مستقل تعریف می‌شود تا الگوریتم‌های مختلف بتوانند آن را احضار کنند یا در همان الگوریتم در زمانهای مختلف بتوانند آن را صدا بزنند. رابطه بین یک الگوریتم و یک زیرالگوریتم مشابه رابطه بین برنامه اصلی و یک زیربرنامه در زبانهای برنامه‌نویسی است. اختلاف اصلی بین فرمت یک زیرالگوریتم و یک الگوریتم در آن است که زیرالگوریتم معمولاً دارای عنوانی به صورت زیر است :

NAME(PAR₁, PAR₂, ..., PAR_k)

که در اینجا منظور از NAME نام زیرالگوریتم است و هنگامی مورد استفاده قرار می‌گیرد که زیرالگوریتم احضار شده باشد و PAR₁, ..., PAR_k پارامترهایی هستند که برای انتقال یا جابجایی داده‌ها بین زیرالگوریتم و الگوریتم فراخواننده بکار می‌روند.

تفاوت دیگر، آن است که در زیرالگوریتم به جای دستور Exit دستور Return داریم. تأکید می‌کنیم کنترل اجرا، زمانی به برنامه فراخواننده داده می‌شود که اجرای زیرالگوریتم کامل شده باشد. زیرالگوریتم‌ها به دو دسته اصلی تقسیم می‌شوند : زیرالگوریتم‌های تابع FUNCTION و زیرالگوریتم‌های PROCEDURE. به کمک چند مثال شباهتها و تفاوت‌های این دو نوع زیرالگوریتم را مورد بررسی قرار می‌دهیم. یک تفاوت اساسی این دو زیرالگوریتم آن است که زیرالگوریتم تابع تنها یک مقدار را به الگوریتم فراخواننده بر می‌گرداند، در حالی که زیرالگوریتم PROCEDURE می‌تواند بیش از یک مقدار برگرداند.

مثال ۲-۹

زیرالگوریتم تابع MEAN زیر، میانگین AVE سه عدد A، B و C را پیدا می‌کند.

Function 2.5: MEAN(A, B, C)

1. Set AVE := (A + B + C)/3.
2. Return(AVE).

توجه دارید که MEAN نام زیرالگوریتم است و A، B و C پارامترهای آن هستند. دستور Return مقدار متغیر AVE را که در داخل پرانتز قرار دارد به برنامه فراخواننده بر می‌گرداند. به همان صورتی که یک زیربرنامه تابع به وسیله یک برنامه فراخواننده احضار می‌شود، زیرالگوریتم MEAN نیز توسط یک الگوریتم احضار می‌شود. برای مثال، فرض کنید یک الگوریتم شامل دستور

TEST := MEAN(T₁, T₂, T₃)

است که در آن T₁, T₂ و T₃ نمره‌های آزمون دانشجویان هستند. مقدار آرگومان T₁, T₂ و T₃ در زیرالگوریتم به پارامترهای A, B و C داده می‌شود. زیرالگوریتم MEAN اجرامی شود و بدنیال آن مقدار AVE به برنامه برگردانده می‌شود و جایگزین (T₁, T₂, T₃) در دستور بالا می‌شود. درنتیجه آن، میانگین T₁ و T₂ و T₃ در TEST جایگزین می‌شود.

مثال ۲-۱۰

زیربرنامه Procedure در زیر با نام SWITCH مقدار متغیرهای AAA و BBB را جابجا می‌کند.

Procedure 2.6: SWITCH(AAA, BBB)

1. Set TEMP := AAA, AAA := BBB and BBB := TEMP.
2. Return.

این زیربرنامه توسط دستور CALL صدا زده می‌شود. برای مثال با دستور CALL زیر

Call SWITCH(BEG, AUX)

مقادیر متغیرهای BEG و AUX جابجا می‌شوند. به ویژه این که، هنگام احضار زیربرنامه SWITCH، آرگومان BEG و AUX به ترتیب به پارامترهای AAA و BBB منتقل می‌شود، زیربرنامه Procedure اجرا می‌شود، که درنتیجه آن مقادیر AAA و BBB جابجا می‌شوند و بدنیال آن مقادیر جدید AAA و BBB به ترتیب به BEG و AUX داده می‌شود.

توجه کنید: هر زیرالگوریتم تابع را به سادگی می‌توان به یک زیرالگوریتم Procedure معادل آن تبدیل کرد، به این صورت که، کافی است به الگوریتم فراخواننده یک پارامتر دیگر اضافه کنیم تا از آن برای برگرداندن مقدار محاسبه شده استفاده شود. مثلاً می‌توان تابع 2.5 Function را به صورت زیربرنامه Procedure

MEAN(A, B, C, AVE)

نوشت که در پارامتر AVE آن میانگین A, B و C جایگزین می‌شود. آنگاه دستور

Call MEAN(T₁, T₂, T₃, TEST)

دارای همان اثر جایگزینی میانگین T₁, T₂ و T₃ در TEST است. به بیان کلی‌تر، به جای زیرالگوریتم‌های تابع می‌توانیم از زیرالگوریتم‌های Procedure استفاده کنیم.

۷-۲ متغیرها و انواع داده‌ها

هر متغیر در یک الگوریتم یا برنامه، دارای یک نوع داده‌ای است که، آن نوع داده، گُدی را مشخص می‌کند که از آن برای ذخیره مقدار متغیر استفاده می‌شود. چهار نوع داده استاندارد به قرار زیر است:

- (۱) نوع داده کاراکتری: در این حالت، داده‌ها با استفاده از نوعی کد کاراکتری مانند EBCDIC یا کد ASCII ذخیره می‌شوند. کد ۸ بیتی EBCDIC تعدادی از کاراکترها، در شکل ۷-۲ ارائه شده است.

Char.	Zone	Numeric	Hex	Char.	Zone	Numeric	Hex	Char.	Zone	Numeric	Hex
A	1100	0001	C1	S	1110	0010	E2	blank	0100	0000	40
B		0010	C2	T		0011	E3			1011	4B
C		0011	C3	U		0100	E4	<		1100	4C
D		0100	C4	V		0101	E5	(1101	4D
E		0101	C5	W		0110	E6	+	0100	1110	4E
F		0110	C6	X		0111	E7	&	0101	0000	50
G		0111	C7	Y		1000	E8	\$		1011	5B
H		1000	C8	Z	1110	1001	E9	*		1100	5C
I	1100	1001	C9	0	1111	0000	F0)		1101	5D
J	1101	0001	D1	1		0001	F1	:	0101	1110	5E
K		0010	D2	2		0010	F2	-	0110	0000	60
L		0011	D3	3		0011	F3	/		0001	61
M		0100	D4	4		0100	F4	%		1011	6B
N		0101	D5	5		0101	F5	>		1100	6C
O		0110	D6	6		0110	F6	?	0110	1111	6E
P		0111	D7	7		0111	F7	:	0111	1010	7A
Q		1000	D8	8		1000	F8	#		1011	7B
R	1101	1001	D9	9	1111	1001	F9	=	0111	1110	7C

شکل ۷-۲. بخشی از کد EBCDIC

معمولًا یک کاراکتر در یک بایت حافظه ذخیره می‌شود.

(۲) نوع داده اعشاری (یا نقطه شناور): در این حالت داده‌های عددی با استفاده از صورت نمایی داده‌ها ذخیره می‌شوند.

(۳) نوع داده صحیح (یا نقطه ثابت): در این حالت اعداد صحیح مثبت با استفاده از نمایش دودویی ذخیره می‌شوند و اعداد صحیح منفی با یک تبدیل دودویی مانند مکمل ۲ ذخیره می‌شوند.

(۴) نوع داده منطقی: در این حالت از کدگذاری، متغیر می‌تواند تنها مقدار True و False داشته باشد. از این رو متغیر تنها با استفاده از یک بیت کدگذاری می‌شود. ۱ برای True و ۰ برای False. گاهی اوقات از بایتها ۱۱۱۱ و ۰۰۰۰ به ترتیب برای True و False استفاده می‌شود.

در الگوریتم‌های این کتاب، نوع متغیرها مانند برنامه‌های کامپیوتروی به صورت صریح بیان نمی‌شود اما در درون متن، به صورت ضمنی بیان می‌شود.

مثال ۲-۱۱

فرض کنید X یک خانه حافظه ۳۲ بیتی با دنباله بیتهاي زیر باشد:

0110 1100 1100 0111 1101 0110 0110 1100

جز در حالتی که نوع متغیر X بیان شده است به هیچ طریق نمی‌توان هیچگونه اطلاع دقیقی از محتوای این خانه حافظه به دست آورد.

(الف) فرض کنید متغیر X دارای نوع کاراکتری است و برای آن از گذگاری **EBCDIC** هم استفاده شده است. در آن صورت چهار کاراکتر %GO در متغیر X قابل ذخیره است.

(ب) فرض کنید X دارای نوع دیگری نظیر صحیح یا اعشاری است. در آن صورت می‌توان یک عدد صحیح یا اعشاری را در X ذخیره کرد.

متغیرهای محلی و سراسری

سازماندهی یک برنامه کامپیوتروی به صورت یک برنامه اصلی و چند زیربرنامه متنهی به مفهوم متغیرهای محلی و سراسری شده است. معمولاً یک قطعه برنامه شامل لیستی از چند متغیر مربوط به خود موسوم به متغیرهای محلی است که فقط توسط این قطعه برنامه قابل دسترسی است. علاوه بر این، قطعه‌هایی که به صورت زیربرنامه هستند، می‌توانند شامل چند پارامتر، باشند این پارامترها، متغیرهایی هستند که داده‌ها را بین یک زیربرنامه و برنامه فراخواننده انتقال می‌دهند.

مثال ۲-۱۲

زیربرنامه **Procedure** مثال ۱۰-۲ یعنی (AAA , BBB) **SWITCH** را در نظر بگیرید. متغیرهای AAA و BBB پارامتر هستند. از این پارامترها برای انتقال داده‌ها بین زیربرنامه **Procedure** و الگوریتم فراخواننده استفاده می‌شود. از طرف دیگر متغیر TEMP در زیربرنامه **Procedure** یک متغیر محلی است. محل "زنده‌گی" یا "محیط زیست" این متغیر تها در زیربرنامه **Procedure** است یعنی مقدار آن تنها با اجرای این زیربرنامه قابل دسترسی و تغییر است. حقیقت این است که از نام TEMP می‌توان در هر قطعه برنامه دیگر تحت عنوان نام یک متغیر استفاده کرد و استفاده از این نام در قطعه برنامه دیگر هیچ تأثیری در اجرای زیربرنامه **SWITCH** ندارد.

طراحان زبانهای برنامه‌نویسی بر این باورند که در یک برنامه کامپیوتروی بهتر است از چند متغیر مشخص استفاده شود، تا یک یا حتی تمام قطعه برنامه‌ها بتوانند به آن متغیرها دسترسی داشته باشند. متغیرهایی که تمام قطعه برنامه‌ها به آنها دسترسی دارند متغیرهای سراسری نام دارند و متغیرهایی که قطعه برنامه‌های معین به آنها دسترسی دارند متغیرهای غیر محلی نام دارند. هر زبان برنامه‌نویسی برای معرفی چنین متغیرهایی، ساختار دستوری مختص به خود دارد. برای مثال، در زبان FORTRAN از دستور COMMON برای معرفی متغیرهای سراسری استفاده می‌شود و زبان PASCAL از قاعدة دامنه تغییرات متغیرها SCOPE برای معرفی متغیرهای سراسری و غیر محلی استفاده می‌کند.

بنابراین، برای انتقال اطلاعات بین قطعه برنامه‌ها دو روش اساسی زیر وجود دارد:

(۱) روش مستقیم، به کمک پارامترهایی که خوش تعریف‌اند.

(۲) روش غیرمستقیم، به کمک متغیرهای غیر محلی و سراسری.

تغییر غیرمستقیم مقدار متغیر یک قطعه برنامه توسط قطعه برنامه دیگر اثر جانبی یا عوارض جانبی Side Effect نام دارد. دانشجویان می‌باید به هنگام استفاده از متغیر غیر محلی و سراسری نهایت دقت را داشته باشند چون خطاهای ایجاد شده توسط اثر جانبی به سختی قابل کشف است.

مسائلهای حل شده

نمادگذاری ریاضی و تابعهای کامپیوتروی

مسئله ۱ - ۲: مطلوب است تعیین

$$(\text{الف}) \quad [7.5], [-7.5], [-18], [\sqrt{30}], [\pi], [\sqrt[3]{30}]$$

$$(\text{ب}) \quad [7.5], [-7.5], [-18], [\sqrt{30}], [\sqrt[3]{30}], [\pi]$$

حل: (الف) بنابر تعريف، $[x]$ بزرگترین عدد صحیحی را نمایش می‌دهد که بزرگتر از x نباشد و کف x نامیده می‌شود. درنتیجه:

$$\begin{array}{lll} [7.5] = 7 & [-7.5] = -8 & [-18] = -18 \\ [\sqrt{30}] = 5 & [\sqrt[3]{30}] = 3 & [\pi] = 3 \end{array}$$

(ب) بنابر تعريف، $[x]$ کوچکترین عدد صحیحی را نمایش می‌دهد که کوچکتر از x نباشد و سقف x نامیده می‌شود. درنتیجه:

$$\begin{array}{lll} [7.5] = 8 & [-7.5] = -7 & [-18] = -18 \\ [\sqrt{30}] = 6 & [\sqrt[3]{30}] = 4 & [\pi] = 4 \end{array}$$

مسئله ۲-۲ :

(الف) مطلوب است تعیین $26 \pmod{7}$, $34 \pmod{8}$, $2345 \pmod{6}$, $495 \pmod{11}$

(ب) مطلوب است تعیین $-26 \pmod{7}$, $-2345 \pmod{6}$, $-371 \pmod{8}$, $-39 \pmod{3}$

(ج) با استفاده از حساب با باقیمانده ۱۵، حاصل $10 - 9, 2 + 11, 4 + 9$ را به دست آورید.

حل : (الف) چون k مثبت است تنها با تقسیم k بر M باقیمانده r بدست می آید. در آن صورت $r \equiv K \pmod{M}$. بنابراین

$$5 = 26 \pmod{7}$$

$$2 = 34 \pmod{8}$$

$$5 = 2345 \pmod{6}$$

$$0 = 495 \pmod{11}$$

(ب) هرگاه k منفی باشد، با تقسیم $|k|$ بر M باقیمانده r' به دست می آید. در آن صورت $.k \equiv -r' \pmod{M}$. بنابراین هرگاه $0 \neq r'$ از این رو است که $.k \pmod{M} = M - r' = 8 - 3 = 5$

$$-26 \pmod{7} = 7 - 5 = 2 \quad -371 \pmod{8} = 8 - 3 = 5$$

$$-2345 \pmod{6} = 6 - 5 = 1 \quad -39 \pmod{3} = 0$$

(ج) از $a \pm M \equiv a \pmod{M}$ استفاده کنید.

$$\begin{array}{ll} 9 + 13 = 22 \equiv 22 - 15 = 7 & 7 + 11 = 18 \equiv 18 - 15 = 3 \\ 4 - 9 = -5 \equiv -5 + 15 = 10 & 2 - 10 = -8 \equiv -8 + 15 = 7 \end{array}$$

مسئله ۲-۳ : تمام جایگشت‌های سه عدد ۱، ۲، ۳ و ۴ را بنویسید.

حل : ابتدا توجه داشته باشید که تعداد کل این جایگشت‌ها $= 4! = 24$ است.

1234	1243	1324	1342	1423	1432
2134	2143	2314	2341	2413	2431
3124	3142	3214	3241	3412	3421
4123	4132	4213	4231	4312	4321

ملاحظه می‌کنید که شش جایگشت سطر اول با یک، شش جایگشت سطر دوم با ۲ شروع می‌شوند همینطور تا آخر.

مسئله ۲-۴ : مطلوب است تعیین (الف)

(ب) $[log_2 1000]$, $[log_2 0.01]$ (ج) $\log_2 32$, $\log_{10} 1000$, $\log_2 (1/16)$;

حل : (الف) $2^{-5} = 1/2^5 = 1/32$; $8^{2/3} = (\sqrt[3]{8})^2 = 2^2 = 4$; $25^{-3/2} = 1/25^{3/2} = 1/5^3 = 1/125$

(ب) $10^3 = 1000$, $\log_{10} 1000 = 3$ چون، $2^5 = 32$ $\log_2 32 = 5$

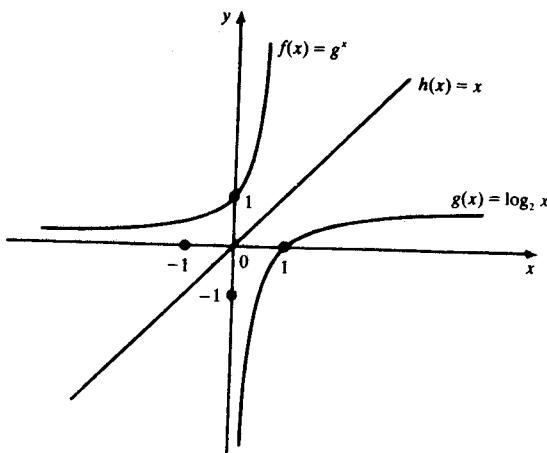
(ج) $2^{-4} = 1/2^4 = 1/16$ چون، $\log_2(1/16) = -4$

(ج) $2^{10} = 1024$ ، اما $2^9 = 512$, چون $[\log_2 1000] = 9$

(ج) $2^{-7} = 1/128 < 0.01 < 2^{-6} = 1/64$ چون $[\log_2 0.01] = -7$

مسئله ۲-۵ : نمودار تابع نمایی $f(x) = 2^x$, تابع لگاریتمی $f(x) = \log_2 x$ و تابع خطی $h(x) = x$ را برابر

روی یک محور مختصات رسم کنید. (الف) خاصیت هندسی نمودار $f(x)$ و $g(x)$ را بیان کنید. (ب) بهازای عدد مثبت دلخواه c ، چه رابطه‌ای بین $f(c)$ ، $g(c)$ و $h(c)$ برقرار است؟
حل: در شکل ۸-۲ نمودار این سه تابع رسم شده است.



شکل ۸-۲

(الف) چون $x^2 = f(x)$ و $x = g(x) = \log_2 x$ معکوس یکدیگر هستند. از این‌رو نمودار این دو تابع نسبت به خط $x = y$ متقارن است.

(ب) بهازای هر عدد مثبت c ، داریم:

$$g(c) < h(c) < f(c)$$

واقعیت این است که با افزایش مقدار c ، فاصله قائم بین تابعهای

$$f(c) - h(c), \quad \text{و} \quad h(c) - g(c)$$

از نظر مقدار افزایش می‌باشد، علاوه بر این تابع لگاریتمی $g(x)$ در مقایسه با تابع خطی $h(x)$ رشد کنده دارد و رشد تابع نمایی $f(x)$ در مقایسه با تابع خطی $h(x)$ بسیار تند است.

الگوریتم‌ها، پیچیدگی آنها

مسأله ۶-۲: الگوریتم 2.3 را درنظر بگیرید که مکان LOC و مقدار MAX بزرگترین عنصر آرایه DATA با عنصر را پیدا می‌کند. (n) پیچیدگی تابع را درنظر بگیرید که تعداد دفعاتی را که LOC و MAX در

مرحله ۳ تازه می‌شوند اندازه می‌گیرد. تعداد مقایسه‌ها مستقل از ترتیب قرارگیری عناصر در DATA است.

(الف) بدترین حالت را تشریح کنید و $C(n)$ آن را پیدا کنید.

(ب) بهترین حالت را تشریح کنید و $C(n)$ آن را پیدا کنید.

(ج) بعازای $3 = n$ ، $C(n)$ را برای حالت میانگین پیدا کنید، فرض کنید ترتیب قرارگیری تمام عناصر در DATA با احتمال مساوی است.

حل : (الف) بدترین حالت وقتی اتفاق می‌افتد که عناصر DATA به ترتیب صعودی باشند که در آن، مقایسه MAX با $[K]$ باعث می‌شود LOC و MAX تازه شوند، در این حالت $n-1 = C(n)$.

(ب) بهترین حالت وقتی اتفاق می‌افتد که بزرگترین عنصر آرایه، اولین عنصر آن باشد و از این‌رو هنگام مقایسه MAX با $[K]$ ، DATA[LOC] و MAX هرگز تازه نمی‌شوند. بنابراین در این حالت $0 = C(n)$.

(ج) فرض کنید $1, 2$ و 3 به ترتیب بزرگترین عنصر، دومین عنصر بزرگ آرایه و کوچکترین عنصر آرایه DATA باشند. عناصر در آرایه DATA می‌توانند به شش طریق ممکن ظاهر شوند که متناظر با جایگشت‌های سه عدد $1, 2, 3$ یعنی $= 3!$ است. برای هر جایگشت P ، فرض کنید p نمایش تعداد دفعات تازه شدن LOC و MAX باشد، که الگوریتم با ورودی P اجرا می‌شود. شش جایگشت P و مقادیر p متناظر آن به صورت زیر است :

جایگشت P :	123	132	213	231	312	321	:
مقادیر p :	0	0	1	1	1	2	

با فرض این که تمام جایگشت‌ها با احتمال مساوی باشند :

$$C(3) = \frac{0+0+1+1+1+2}{6} = \frac{5}{6}$$

محاسبه مقدار میانگین $C(n)$ برای n دلخواه، خارج از حدود مطالب درس ساختمان داده‌ها است. یکی از اهداف این مسئله، آن است که نشان دهیم هنگام تعیین پیچیدگی حالت میانگین یک الگوریتم، با چه مشکلاتی دست به گردیان هستیم.

مسئله ۲-۷ : فرض کنید M واحد زمانی، لازم است تا قطعه برنامه A اجرا شود که در آن M یک مقدار ثابت است. هرگاه n تعداد داده‌های ورودی و m عدد مثبتی بزرگتر از یک باشد، $C(n)$ پیچیدگی هر یک از دو الگوریتم زیر را به دست آورید.

(الف) الگوریتم P 2-7 A :

Algorithm P2.7A:

1. Repeat for I = 1 to N:
2. Repeat for J = 1 to N:
3. Repeat for K = 1 to N:
4. Module A.
- [End of Step 3 loop.]
- [End of Step 2 loop.]
- [End of Step 1 loop.]
5. Exit.

(ب) الگوریتم P 2-7 B :

Algorithm P2.7B:

1. Set J := 1.
2. Repeat Steps 3 and 4 while J ≤ N:
3. Module A.
4. Set J := B × J.
- [End of Step 2 loop.]
5. Exit.

ملاحظه می‌کنید که در این الگوریتم به جای n از N و به جای b از B استفاده شده است.

حل : (الف) در اینجا

$$C(n) = \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n M$$

تعداد دفعاتی که M در این مجموع ظاهر می‌شود برابر تعداد سه تابی‌های (i,j,k) است که در آن i, j,

k اعداد صحیح از 1 تا خود n هستند. تعداد n^3 سه تابی از این نوع وجود دارد. از این رو

$$C(n) = Mn^3 = O(n^3)$$

(ب) ملاحظه می‌کنید که مقادیر شاخص حلقه J توانهایی از b هستند:

$$1, b, b^2, b^3, b^4, \dots$$

بنابراین قطعه برنامه A دقیقاً T بار تکرار می‌شود که در آن T اولین توانی از b است که

$$b^T > n$$

$$T = \lceil \log_b n \rceil + 1$$

$$C(n) = MT = O(\log_b n)$$

در آن

درنتیجه :

بنابراین :

مسئله - ۲ : (الف) یک زیربرنامه FIND(DATA, N, LOC1, LOC2) به نام Procedure بنویسید که مکان بزرگترین عنصر و LOC2 مکان دومین عنصر بزرگ آرایه DATA را برای $n > 1$ عنصر پیدا کند.

حل : (الف) عناصر DATA را یک به یک مورد بررسی قرار دهید. در خلال اجرای این زیربرنامه procedure و SECOND به ترتیب مقادیر بزرگترین عنصر و دومین عنصر بزرگ آرایه را که قبلًا مورد بررسی قرار گرفته نمایش می‌دهند. هر عنصر جدید DATA[K] به صورت زیر آزمایش می‌شود که اگر

$$\text{SECOND} \leq \text{FIRST} < \text{DATA}[K]$$

آنگاه عنصر FIRST عنصر SECOND جدید و DATA[K] عنصر FIRST جدید می‌شود. از طرف دیگر اگر

$$\text{SECOND} < \text{DATA}[K] \leq \text{FIRST}$$

آنگاه [2] عنصر DATA[K] جدید می‌شود. در آغاز قرار دهید FIRST := DATA[1] و SECOND := DATA[2] و بررسی کنید که آیا این عناصرها با ترتیب درست قرار گرفته‌اند یا خیر. نمایش رسمی این زیربرنامه به صورت زیر است :

Procedure P2.8: FIND(DATA, N, LOC1, LOC2)

1. Set FIRST := DATA[1], SECOND := DATA[2], LOC1 := 1, LOC2 := 2.
 2. [Are FIRST and SECOND initially correct?]
 - If FIRST < SECOND, then:
 - (a) Interchange FIRST and SECOND,
 - (b) Set LOC1 := 2 and LOC2 := 1.
 - [End of If structure.]
 3. Repeat for K = 3 to N:
 - If FIRST < DATA[K], then:
 - (a) Set SECOND := FIRST and FIRST := DATA[K].
 - (b) Set LOC2 := LOC1 and LOC1 := K.
 - Else if SECOND < DATA[K], then:
 - Set SECOND := DATA[K] and LOC2 := K.
 - [End of If structure.]
 - [End of loop.]
4. Return.

(ب) استفاده از پارامتر اضافی FIRST و SECOND به نظر زاید می‌آید چون LOC1 و LOC2 خود به خود به برنامه فراخوانده می‌گویند که DATA[LOC1] و DATA[LOC2] به ترتیب مقادیر بزرگترین عنصر و دومین عنصر بزرگ آرایه DATA هستند.

مسئله ۹ - ۲ : عدد صحیح $n > 1$ یک عدد اول نامیده می‌شود اگر مقصوم علیه‌های مثبت آن تنها ۱ و n باشند در غیر اینصورت به n یک عدد مرکب می‌گویند، برای مثال، در زیر لیست اعداد اول کوچکتر از ۲۰ ارائه شده است :

$$2, 3, 5, 7, 11, 13, 17, 19$$

اگر $n > 1$ اول نباشد، یعنی اگر n یک عدد مرکب باشد آنگاه n باید مقصوم علیه‌ی مانند $K \neq 1$ داشته باشد بطوری که $k^2 \leq n$ یا به بیان دیگر $k^2 \leq n$

فرض کنید خواسته باشیم تمام اعداد اول کوچکتر از یک عدد معلوم m مانند ۳۰ را به دست آوریم.

این کار با روشی موسوم به "روش غربال اراتسین" عملی است که از مراحل زیر تشکیل شده است.
لیست تمام اعداد از یک تا ۳۰ را بنویسید :

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30

عدد ۱ و تمام مضربهای عدد ۲ به غیر از خود ۲ را به صورت زیر حذف کنید :

~~1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15~~
~~16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30~~

حال چون عدد ۳ اولین عدد بعد از ۲ است که حذف نشده است در این مرحله تمام مضربهای عدد ۳ غیر از خود ۳ را به صورت زیر از لیست حذف کنید :

~~1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15~~
~~16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30~~

حال چون عدد ۵ اولین عدد بعد از ۳ است که حذف نشده است در این مرحله تمام مضربهای عدد ۵ غیر از خود ۵ را به صورت زیر از لیست حذف کنید :

~~1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15~~
~~16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30~~

حال چون عدد ۷ اولین عدد بعد از ۵ است که حذف نشده است و چون $3^2 > 30$. این الگوریتم به پایان رسیده است و اعداد باقیمانده در لیست، اعداد اول کوچکتر از ۳۰ هستند :

2, 3, 5, 7, 11, 13, 17, 19, 23, 29

بازدیده به توضیحات فوق، روش غربال اراتسین برای پیدا کردن کلیه اعداد اول کوچکتر از عدد معلوم n را به یک الگوریتم تبدیل کنید.

حل : ابتدا آرایه A را به گونه‌ای تعریف می‌کنیم که

$$A[1] = 1, \quad A[2] = 2, \quad A[3] = 3, \quad A[4] = 4, \dots, A[N-1] = N-1, \quad A[N] = N$$

عدد صحیح L را با دستور جایگزینی $A[L] = 1$ از لیست حذف می‌کنیم زیر برنامه **Procedure CROSSOUT** :

$$A[2K] = 1, \quad A[3K] = 1, \quad A[4K] = 1, \dots$$

یعنی مضربهای K از لیست حذف شده‌اند :

Procedure P2.9A: CROSSOUT(A, N, K)

1. If $A[K] = 1$, then: Return.
2. Repeat for $L = 2K$ to N by K :
 - Set $A[L] := 1$.
 - [End of loop.]
3. Return.

روش غربال را اکنون می‌توان به صورت ساده‌زیر نوشت:

Algorithm P2.9B: This algorithm prints the prime numbers less than N .

1. [Initialize array A.] Repeat for $K = 1$ to N :
 - Set $A[K] := K$.
2. [Eliminate multiples of K .] Repeat for $K = 2$ to \sqrt{N} .
 - Call CROSSOUT(A, N, K).
3. [Print the primes.] Repeat for $K = 2$ to N :
 - If $A[K] \neq 1$, then: Write: $A[K]$.
4. Exit.

مسئله ۱۰ - ۲: فرض کنید $P(n) = a_0 + a_1n + a_2n^2 + \dots + a_m n^m$ یعنی فرض کنید درجه n برابر باشد. ثابت کنید که $P(n) = O(n^m)$

حل: اگر $n \geq 1$ آنگاه به ازای $b_0 = |a_0|, b_1 = |a_1|, \dots, b_m = |a_m|$.

$$\begin{aligned} P(n) &\leq b_0 + b_1n + b_2n^2 + \dots + b_m n^m = \left(\frac{b_0}{n^m} + \frac{b_1}{n^{m-1}} + \dots + b_m \right) n^m \\ &\leq (b_0 + b_1 + \dots + b_m) n^m = Mn^m \end{aligned}$$

که در آن $P(n) = O(n^m)$ از این رو $M = |a_0| + |a_1| + \dots + |a_m|$.

$x^5 - 4000000x^2 = O(x^5)$ و $5x^3 + 3x = O(x^3)$ برای مثال

متغیرها، انواع داده‌ای

مسئله ۱۱ - ۲: به اختصار تفاوت بین متغیرهای محلی، پارامترها و متغیرهای سراسری را بیان کنید.

حل: متغیرهای محلی متغیرهایی هستند که تنها در درون یک برنامه خاص یا یک زیربرنامه، قابل دسترس هستند. پارامترها، متغیرهایی هستند که برای انتقال داده‌ها بین یک زیربرنامه و برنامه فراخواننده بکار می‌رود. متغیرهای سراسری، متغیرهایی هستند که تمام قطعه برنامه‌های یک برنامه کامپیوتری می‌توانند به آنها دسترسی داشته باشند. هر زبان برنامه‌نویسی که استفاده از متغیرهای سراسری را مجاز دانسته باشد دارای دستوراتی برای معرفی این نوع متغیرها است.

مسئله ۱۲ - ۲: فرض کنید NUM معرف تعداد رکوردهای یک فایل باشد. مزایای تعریف NUM را به صورت یک متغیر سراسری شرح دهید. معایب استفاده از متغیرهای سراسری را در حالت کلی شرح

دهید.

حل : در بسیاری از زیربرنامه‌های **Procedure** پردازش رکوردهای یک فایل با استفاده از نوعی حلقه صورت می‌گیرد. از آنجاکه تمام این زیربرنامه‌ها، تنها از یک نام برای رکورد، یعنی، **NUM** استفاده می‌کنند که این یکی از عیبهای معروف **NUM** به صورت یک متغیر سراسری است، به بیان کلی‌تر، متغیرهای سراسری و غیر محلی ممکن است در اثر عوارض جانبی **Side Effect** منجر به خطاها بیشتر **errors** شوند که احتمالاً کشف آنها مشکل است.

مسأله ۱۳-۲ : فرض کنید **AAA** یک خانه حافظه ۳۲ بیتی با دنباله بیتها زیر باشد :

0100 1101	1100 0001	1110 1001	0101 1101
-----------	-----------	-----------	-----------

مطلوب است تعیین داده ذخیره شده در **AAA**.

حل : بجز در حالتی که نوع داده‌ای **AAA** را می‌دانیم، هیچ راهی برای تعیین داده ذخیره شده در **AAA** وجود ندارد. اگر **AAA** یک متغیر کاراکتری باشد و از کدگذاری **EBCDIC** نیز برای ذخیره داده‌ها استفاده شود آنگاه (**AZ**) در **AAA** ذخیره می‌شود. اگر **AAA** یک متغیر صحیح باشد، آنگاه یک عدد صحیح با نمایش دو دویی بالا در **AAA** ذخیره می‌شود.

مسأله ۱۴-۲ : از نظر ریاضی، اعداد صحیح را می‌توان مانند اعداد حقیقی (اعشاری) مورد بررسی قرار داد. دلایلی برای داشتن دو نوع داده مختلف صحیح و اعشاری ارائه دهید.

حل : حساب اعداد صحیح که با استفاده از نوعی نمایش دو دویی در کامپیوتر ذخیره می‌شوند خیلی ساده‌تر از حساب اعداد اعشاری است که به صورت نمایی ذخیره می‌شوند. علاوه بر این خطای ناشی از گردکردن که در اعداد اعشاری اتفاق می‌افتد در حساب اعداد صحیح وجود ندارد.

مسأله‌های تكمیلی

نمادهای ریاضی و تابعهای کامپیوتري

مسأله ۱۵-۲ : مطلوب است تعیین

(الف) $[3.4], [-3.4], [-7], [\sqrt{75}], [\sqrt[3]{75}], [e]$

(ب) $[3.4], [-3.4], [-7], [\sqrt{75}], [\sqrt[3]{75}], [e]$

مسأله ۱۶-۲ : (الف) مطلوب است محاسبه $48 \pmod{5}, 48 \pmod{7}, 1397 \pmod{11}, 2468 \pmod{9}$

(ب) مطلوب است محاسبه $-48 \pmod{5}, -152 \pmod{7}, -358 \pmod{11}, -1326 \pmod{13}$

(ج) با استفاده از حساب با باقیمانده ۱۳، مطلوب است محاسبه

$$9 + 10, \quad 8 + 12, \quad 3 + 4, \quad 3 - 4, \quad 2 - 7, \quad 5 - 8$$

مسئله ۱۷-۲: مطلوب است تعیین

(الف) $|3 + 8|, |3 - 8|, |-3 + 8|, |-3 - 8|$

(ب) $7!, 8!, 14!/12!, 15!/16!$

مسئله ۱۸-۲: مطلوب است محاسبه

(الف) $3^{-4}, 4^{7/2}, 27^{-2/3}$

(ب) $\log_2 64, \log_{10} 0.001, \log_2 (1/8)$

(ج) $\lfloor \lg 1\,000\,000 \rfloor, \lfloor \lg 0.001 \rfloor$

الگوریتم‌ها، پیچیدگی آنها

مسئله ۱۹-۲: تابع پیچیدگی $C(n)$ را در نظر بگیرید که تعداد دفعاتی را که LOC در مرحله ۳ ای الگوریتم ۲.۳ تازه می‌شود اندازه می‌گیرد. بهمازای $n = 4$ ، حالت میانگین $C(n)$ را پیداکنید. فرض می‌شود ترتیب تمام چهار عنصر داده شده با احتمال برابر است. مسئله ۱۹-۲ را نیز بپسندید.

مسئله ۲۰-۲: زیربرنامه Procedure P2.8 را در نظر بگیرید که LOC1 مکان بزرگترین عنصر و LOC2 مکان دومین عنصر بزرگ آرایه DATA را که $1 < n$ ، پیدا می‌کند. فرض کنید $C(n)$ نمایش تعداد مقایسه‌ها در خلال اجرای این Procedure باشد.

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

(ب) $C(n)$ را در بدترین حالت پیدا کنید.

(ج) بهمازای $n = 4$ ، $C(n)$ را در حالت میانگین پیدا کنید. فرض می‌شود ترتیب تمام عناصر داده شده در DATA با احتمال برابر است.

مسئله ۲۱-۲: مسئله ۲۰-۲ را مجدداً حل کنید با این تفاوت که اکنون فرض کنید که $C(n)$ نمایش تعداد دفعاتی باشد که باید مقدارهای FIRST و SECOND (یا LOC1 و LOC2) تازه شوند.

مسئله ۲۲-۲: فرض کنید که زمان اجرای قطعه برنامه A مقدار ثابت M است. مرتبه بزرگی تابع پیچیدگی $C(n)$ را به دست آورید که زمان اجرای الگوریتم‌های زیر را که در آن n تعداد داده‌های ورودی است پیدا می‌کند. n در الگوریتم‌ها با N نشان داده شده است.

Procedure P2.22A: 1. Repeat for I = 1 to N: (الف)

2. Repeat for J = 1 to I:

3. Repeat for K = 1 to J:

4. Module A.

[End of Step 3 loop.]

[End of Step 2 loop.]

[End of Step 1 loop.]

5. Exit.

Procedure P2.22B:

1. Set $J := N$.
2. Repeat Steps 3 and 4 while $J > 1$.
3. Module A.
4. Set $J := J/2$.
5. [End of Step 2 loop.]
6. Return.

(ب)

برای مسئله های زیر، برنامه بنویسید

مسئله ۲-۲۳ : یک زیر برنامه تابع $DIV(J,K)$ بنویسید که در آن J و K اعداد صحیح مثبت هستند طوری که اگر J یک عامل K باشد $1 = DIV(J,K)$ ، در غیر اینصورت $0 = DIV(J,K)$. (برای مثال $DIV(3,15) = 1$ اما $DIV(3,16) = 0$).

مسئله ۲-۲۴ : با استفاده از $DIV(J,K)$ برنامه ای بنویسید که یک عدد صحیح مثبت $N > 10$ را بخواند و تعیین کند که آیا N اول است یا خیر.

راهنمایی : اول است اگر $DIV(2,N) = 0$ (i) یعنی N فرد باشد و (ii) به ازای تمام اعداد صحیح فرد K که $N \leq K^2$ که $DIV(K,N) = 0$ ، $1 < K^2$

مسئله ۲-۲۵ : زیر برنامه Procedure 2.8 را به یک برنامه کامپیوترا تبدیل کنید یعنی برنامه ای بنویسید که LOC1 مکان بزرگترین عنصر و LOC2 مکان دومین عنصر بزرگ آرایه DATA را که دارای $N > 1$ عنصر است پیدا کند. برنامه را با استفاده از $70, 30, 25, 80, 50, 60, 30, 75, 25$ و 60 آزمایش کنید.

مسئله ۲-۲۶ : روش غربال اراتسین برای تعیین اعداد اول را که در مسئله ۲-۹ شرح داده شده است به یک برنامه کامپیوترا تبدیل کنید تا اعداد اول کوچکتر از N را پیدا کند. برنامه را با استفاده از (الف) $N = 1000$ و (ب) $N = 10000$ آزمایش کنید.

مسئله ۲-۲۷ : فرض کنید C نمایش تعداد دفعاتی باشد که LOC در الگوریتم ۲-۳ برای تعیین بزرگترین عنصر آرایه N عنصری A، تازه می شود.

(الف) یک زیر برنامه COUNT(A,N,C) برای تعیین C بنویسید.

(ب) یک زیر برنامه Procedure P2.27 بنویسید که (i) عدد تصادفی بین ۰ و ۱ در آرایه A بخواند و (ii) با استفاده از COUNT(A,N,C) مقدار C را پیدا کند.

(ج) برنامه ای بنویسید که زیر برنامه Procedure P2.27 را ۱۰۰۰ بار اجرا کند و میانگین هزار C را پیدا کند. (i) به ازای $N = 3$ برنامه را آزمایش کنید و نتیجه را با مقدار بدست آمده در مسئله ۲-۶ مقایسه کنید. (ii) به ازای $N = 4$ برنامه را اجرا کنید و نتیجه را با مقدار بدست آمده در مسئله ۲-۲۹ مقایسه کنید.

فصل ۳

پردازش رشته‌ها

۱- مقدمه

از نظر تاریخی، از کامپیوتر نخست به منظور پردازش داده‌های عددی استفاده می‌شد. امروزه، اغلب کامپیوتر برای پردازش داده‌های غیرعددی موسوم به داده‌های کاراکتری مورد استفاده قرار می‌گیرد. در این فصل چگونگی ذخیره داده‌های کاراکتری و پردازش آنها بررسی می‌شود.

امروزه یکی از کاربردهای اصلی کامپیوتر در عرصه پردازش کلمات یا رشته‌ها است. معمولاً چنین پردازش‌هایی شامل نوعی از تطبیق الگو است نظیر تحقیق در مورد این که آیا یک کلمه خاص مانند **S** در متن داده شده **T** وجود دارد یا خیر. ما این مسأله تطبیق الگو را به طور مسروچ بررسی می‌کنیم و علاوه بر این دو الگوریتم مختلف برای تطبیق الگوها ارائه خواهیم داد.

در دروس کامپیوتری معمولاً برای دنباله‌ای از کاراکترها به جای اصطلاح "کلمه" از اصطلاح "رشته" استفاده می‌شود چون "کلمه" دارای معنی دیگری در علم کامپیوتر است. به همین دلیل، در بسیاری از متنها و کتابهای کامپیوتری از عبارت "پردازش رشته‌ها"، "عملیات بر روی رشته‌ها" یا "ویرایش متن" به جای عبارت "پردازش کلمه" استفاده می‌شود.

مطلوب این فصل اساساً جهت یادآوری بیان شده و مستقل از بقیه مطالب کتاب است. بنابراین دانشجو یا استاد می‌تواند در مطالعه اول این فصل را حذف کند و یا مطالعه آن را تا زمان دیگر به تعویق بیندازد.

۲-۳ اصطلاحات پایه‌ای

هر زبان برنامه‌نویسی دارای یک مجموعه از کاراکترها است که از آنها برای برقاری ارتباط با کامپیوتر استفاده می‌کند. این مجموعه معمولاً شامل کاراکترهای زیر است:

کاراکترهای الفبایی : ABCDEFGHIJKLMNOPQRSTUVWXYZ

کاراکترهای رقمی : ۰ ۱ ۲ ۳ ۴ ۵ ۶ ۷ ۸ ۹

کاراکترهای مخصوص : + - / * () , . \$ = ' □

مجموعه کاراکترهای مخصوص که شامل فضای خالی نیز است غالباً با □ نمایش داده می‌شود که اندکی از یک زبان برنامه‌نویسی تا زبان دیگر در تغییر است.

رشته یک دنباله متناهی از صفر، یک یا چند کاراکتر است. تعداد کاراکترهای یک رشته، طول رشته نام دارد. رشته‌ای که هیچ کاراکتری نداشته باشد، رشته‌تهی یا رشته پوج نام دارد. رشته‌ها را با مخصوص کردن کاراکترهای آن بین یک علامت نقل قول یا دو علامت نقل قول نمایش می‌دهند. علامت نقل قول یا کوتیشن Quotation به عنوان علائم ابتدا و انتهای رشته یا محدود کننده رشته نیز بکار می‌رود. از این رو

'THE END' 'TO BE OR NOT TO BE' ' ', '□□'

رشته‌هایی به ترتیب با طول ۷، ۱۸، ۰ و ۲ هستند. تأکید می‌کنیم که فضای خالی، یک کاراکتر است و درنتیجه در محاسبه طول رشته‌ها شرکت می‌کند. گاهی اوقات علامت کوتیشن را می‌توان حذف کرد و این هنگامی است که از خود متن چنین استنباط شود که عبارت داده شده یک رشته است. فرض کنید S_1 و S_2 دور رشته باشند. رشته‌ای که از ترکیب کاراکترهای S_1 و بدنبال آن کاراکترهای رشته S_2 حاصل می‌شود پیوند یا اتصال S_1 و S_2 نامیده می‌شود. ما در این کتاب اتصال دو رشته S_1 و S_2 را به صورت $S_1//S_2$ نمایش می‌دهیم. برای مثال:

'THE' // 'END' = 'THEEND' اما 'THE' // '□' // 'END' = 'THE END'

واضح است که طول $S_1//S_2$ برابر مجموع طول های دو رشته S_1 و S_2 است. رشته Y زیر رشته، رشته نامیده می‌شود هرگاه رشته‌های مانند X و Z وجود داشته باشد طوری که:
 $S = X//Y//Z$
اگر X یک رشته‌تهی باشد، آنگاه Y یک زیر رشته اولیه یا ابتدایی S نامیده می‌شود و اگر Z یک رشته تهی باشد آنگاه Y یک زیر رشته انتهایی S نامیده می‌شود. برای مثال:

است. 'TO BE OR NOT TO BE' یک زیر رشته 'BE OR NOT'

است. 'THE END' یک زیر رشته ابتدایی 'THE'

واضح است که اگر Y یک زیر رشته S باشد آنگاه طول Y نمی‌تواند بیشتر از طول S باشد.

توجه کنید: کاراکترها در کامپیوتر به کمک یک گُد ۶ بیتی، یک کد ۷ بیتی، یا یک کد ۸ بیتی ذخیره می‌شوند. واحدی که برابر تعداد بیت‌های موردنیاز برای نمایش یک کاراکتر است یک بایت نامیده می‌شود. کامپیوتري که بتواند به یک بایت از حافظه دسترسی پیدا کند، یک ماشین با قابلیت آدرس دهی بایتی نامیده می‌شود.

۳-۳ ذخیره رشته‌ها

در حالت کلی، رشته‌ها را می‌توان با سه ساختار مختلف ذخیره کرد. (۱) ساختارهایی که طول ثابت دارند. (۲) ساختارهایی با طول متغیر که مانند طول آن ثابت است و (۳) ساختارهای پیوندی. هر نوع ساختار به طور جداگانه مورد بررسی قرار می‌گیرد، مزایا و معایب آنها بیان می‌شود.

رکورد، حافظه با طول ثابت

در حافظه با طول ثابت هر خط چاپ، یک رکورد محسوب می‌شود که در آن تمام رکوردها دارای طول برابر هستند، یعنی در هر رکورد تعداد کاراکترها برابر می‌باشند. از آنجا که داده‌ها اغلب به صورت تصویرهای ۸۰ ستونی به عنوان ورودی به ترمینالها داده می‌شوند یا از کارتهای ۸۰ ستونی استفاده می‌کنند، فرض می‌کنیم که رکوردها طول ۸۰ دارند، در غیراینصورت طول رکور به صورت صریح یا ضمنی بیان می‌گردد.

مثال ۳-۱

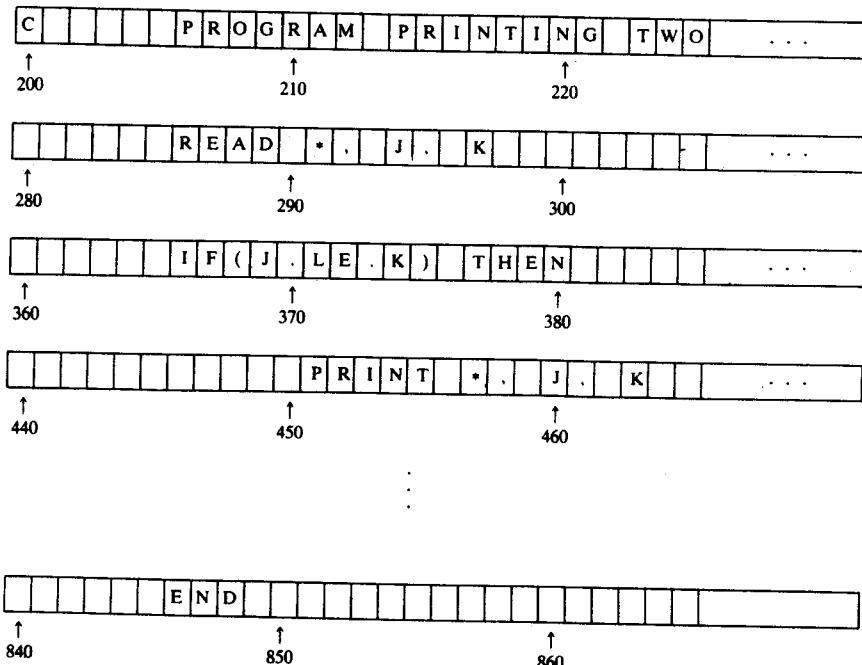
فرض کنید داده ورودی کامپیوترا، برنامه **FORTRAN** شکل ۱-۳ است.

```
C PROGRAM PRINTING TWO INTEGERS IN INCREASING ORDER
READ *, J, K
IF(J.LE.K) THEN
    PRINT *, J, K
ELSE
    PRINT *, K, J
ENDIF
STOP
END
```

شکل ۱-۳. داده ورودی

با استفاده از یک رکورد برای محیط حافظه با طول ثابت، داده ورودی در حافظه به صورتی که در

شکل ۲-۳ ارائه شده است نمایش داده می‌شود که در آن فرض شده است ۲۰۰، آدرس اولین کاراکتر برنامه است.



شکل ۲-۳. رکوردها به صورت متوالی در کامپیوتر ذخیره می‌شوند.

مزایای اصلی روش بالا در ذخیره رشته‌ها عبارت است از:

(۱) دسترسی آسان به اطلاعات هر رکورد معین

(۲) تازه‌سازی آسان اطلاعات هر رکورد معین (مشروط بر این که طول اطلاعات جدید بیشتر از طول رکورد نباشد)

مزایای عمدۀ روش بالا عبارت است از:

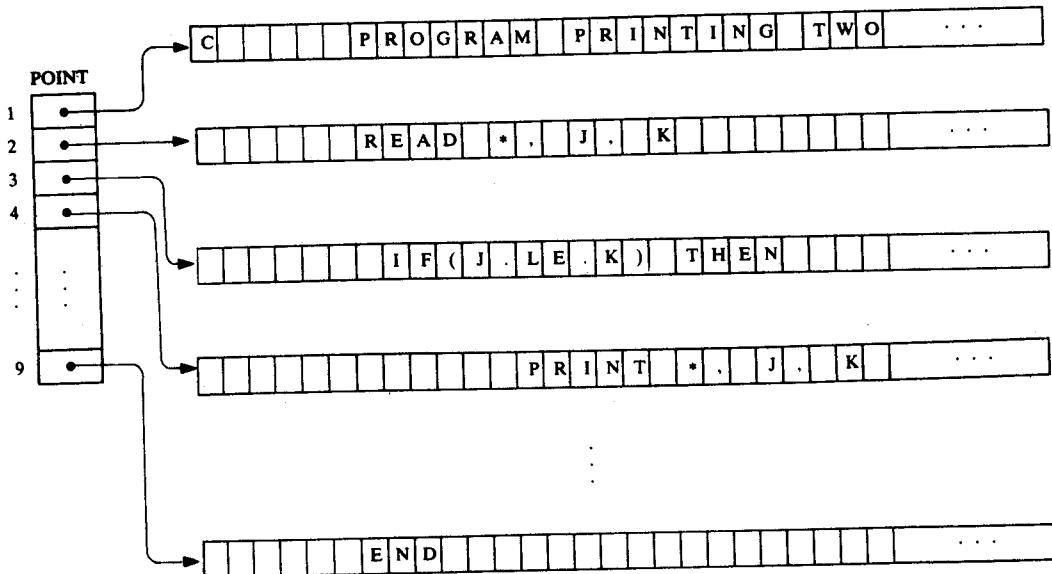
(۱) هرگاه اکثر حافظه از فضاهای خالی تشکیل شده باشد خواندن تمام رکورد باعث بهدر رفتن زمان می‌شود.

(۲) بعضی از رکوردها به حافظه بیشتر از حافظه در دسترس، نیاز دارند.

(۳) هرگاه در متن اصلی احتیاج به تصحیح یک یا چند کاراکتر باشد، آنگاه برای تغییر یک کلمه نادرست

لازم است تمام رکوردها تغییر کند.

توجه کنید: فرض کنید بخواهیم در مثال ۱-۳ یک رکورد جدید اضافه کنیم. این کار مستلزم آن است که تمام رکوردهای بعدی به خانه‌های حافظه جدید منتقل شوند. با وجود این، این عیب را می‌توان به صورتی که در شکل ۳-۳ بیان شده است، برطرف کرد.



شکل ۳-۳. ذخیره‌سازی رکوردها با استفاده از اشاره‌گرها

یعنی می‌توان از یک آرایه خطی POINT که آدرس هر رکورد متوالی را به دست می‌دهد استفاده کرد طوری که لازم نباشد رکوردها در خانه‌های متوالی حافظه ذخیره شوند. بدین ترتیب اضافه کردن یک رکورد جدید تنها مستلزم تازه‌سازی آرایه POINT است.

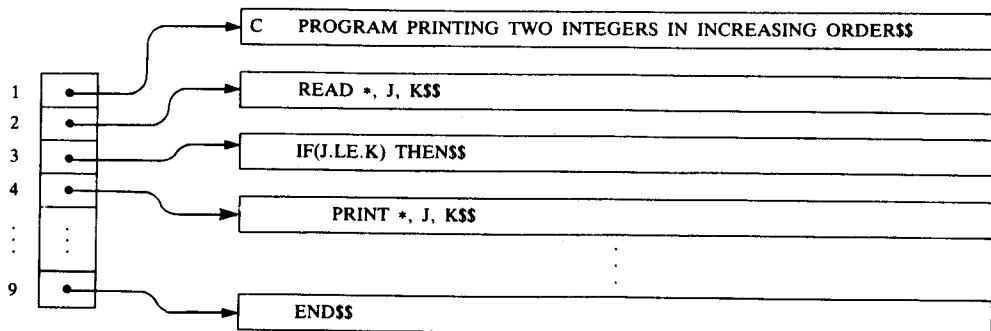
حافظه با طول متغیر، که ماگزینم طول آن ثابت است

اگرچه مانند بالا رشته‌ها را می‌توان با طول ثابت در خانه‌های حافظه ذخیره کرد اما بیان طول واقعی هر رشته دارای معایبی است. مثلاً هنگامی که رشته تنها بخش آغازین خانه حافظه را اشغال می‌کند در آن صورت نباید تمام رکورد خوانده شود. علاوه بر این بعضی از عملیات روی رشته‌ها (که در بخش ۳-۴ بررسی می‌شود) بستگی به داشتن رشته‌هایی با طول متغیر دارند.

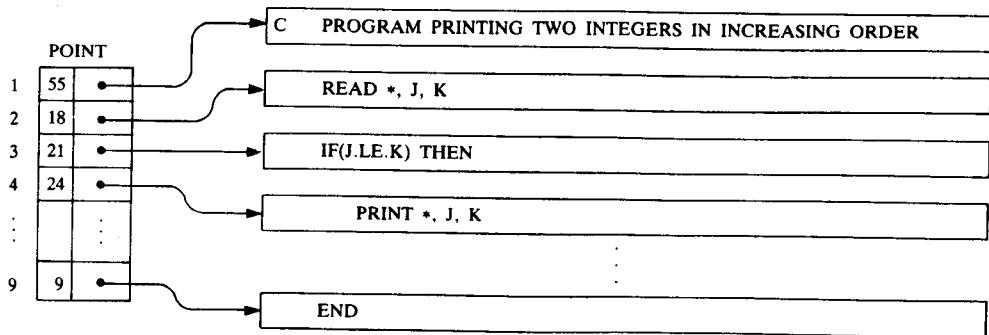
ذخیره رشته‌هایی با طول متغیر در حافظه را که طول ثابت دارند می‌توان به دو روش کلی زیر انجام داد:

(۱) می‌توان برای پایان رشته از یک علامت، نظیر دو علامت دلار \$ استفاده کرد.

(۲) می‌توان طول رشته را مثلاً به عنوان یک فیلد اضافی در آرایه اشاره گر بیان کرد. با استفاده از داده‌های شکل ۳-۱، روش اول در نمودار ۴-۳ (الف) و روش دوم در نمودار ۴-۳ (ب) به تصویر کشیده شده است.



(الف) رکوردها به همراه علامت پایان رشته



(ب) رکوردهایی که طولشان به صورت پیوندی هستند.

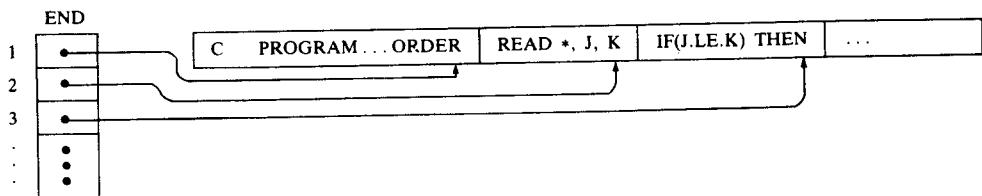
شکل ۴-۳

توجه کنید: بعضی از دانشجویان ممکن است دچار این وسوسه اغوا آمیز شوند که بخواهند رشته‌ها را

یکی بعد از دیگری با استفاده از یک علامت خاص نظیر دو علامت دلار \$\$ شکل ۳-۵ (الف) یا با استفاده از یک آرایه اشاره گر که مکان رشته‌ها را مانند شکل ۳-۵ (ب) به دست می‌دهد ذخیره کنند. واضح است که این‌گونه روشهای ذخیره رشته‌ها باعث صرفه‌جویی در مقدار حافظهٔ مصرفی می‌شود، و گاهی اوقات هنگامی که رکوردها دائمی هستند و به ندرت دستخوش تغییرات کوچک می‌شوند در حافظهٔ ثانویه مورد استفاده قرار می‌گیرد. با وجود این، هنگامی که رشته‌ها و طول آنها غالباً در حال تغییر است، این روشن ذخیره‌سازی معمولاً غیرکارا و نامؤثر است.

C	PROGRAM ... ORDER\$S\$	READ *, J, K\$S	IF(J.LE.K) THEN\$S	...
---	------------------------	-----------------	--------------------	-----

(الف)



(ب)

شکل ۳-۳. ذخیره رکوردها یکی بعد از دیگری

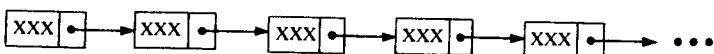
ذخیره رشته‌ها به صورت پیوندی

امروزه از کامپیوتر به میزان زیادی در پردازش کلمات یا رشته‌ها استفاده می‌شود یعنی متن به عنوان ورودی به کامپیوتر داده می‌شود آنگاه پردازش شده و به صورت چاپ شده رخروجی ارائه می‌شود. بنابراین کامپیوتر باید توانایی تصحیح و ویرایش متن چاپ شده را نیز داشته باشد که منظور از ویرایش، حذف، تغییر و اضافه کردن کلمات، یا رشته‌ها عبارات، جملات و حتی پاراگراف‌های یک متن می‌باشد. با وجود این، حافظه‌های با طول ثابت که در بالا مورد بررسی قرار گرفت برای عملیات فوق الذکر مناسب نیستند. بنابراین، در اکثر کاربردهای پردازش کلمه، رشته‌ها با استفاده از لیستهای پیوندی ذخیره می‌شوند. لیستهای پیوندی و چگونگی حذف و اضافه کردن داده‌ها در آن، به طور مشروح در فصل ۵ بررسی

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

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

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



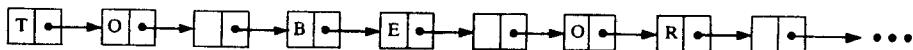
شکل ۶-۳. لیست پیوندی

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

بودن یا نبودن، مسأله این است.

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

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



شکل ۷-۳ (الف). هر کاراکتر در یک گره

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



شکل ۷-۳ (ب). چهار کاراکتر در یک گره

۴-۳ نوع داده کاراکتری

در این بخش مورکلی بروشهایی داریم که زبانهای برنامه‌نویسی مختلف داده نوع کاراکتری را مورد استفاده قرار می‌دهند. همانگونه که در فصل قبل (در بخش ۷-۲) متذکر شدیم، هر نوع داده‌ای دارای فرمول خاص خود جهت از کد در آوردن یک دنباله از بیتها، در حافظه است.

ثابت‌ها

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

'TO BE OR NOT TO BE' و 'THE END'

به ترتیب ثابت‌های رشته‌ای با طول ۷ و ۱۸ کاراکتر هستند. الگوریتم‌های این کتاب نیز ثابت‌های کاراکتری را به همین صورت تعریف می‌کند.

متغیرها

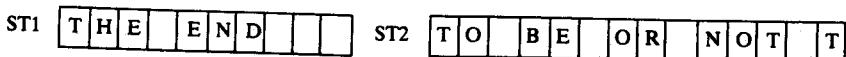
هر زبان برنامه‌نویسی دارای قانون و قاعده خاص خود برای ساختن متغیرهای کاراکتری است. با وجود این، متغیرهای کاراکتری به یکی از سه دسته زیر تقسیم می‌شوند: ایستا، نیمه‌ایستا و پویا. منظور از یک متغیر کاراکتری ایستا، متغیری است که طول آن قبل از اجرای برنامه تعريف شده و در سراسر برنامه ثابت است. منظور از یک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

مثال ۲-۲

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

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

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



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

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

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

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

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

(ج) زبان SNOBOL از متغیرهای کاراکتری پویا استفاده می‌کند. برای مثال، قطعه برنامه SNOBOL

زیر:

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

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

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

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

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

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

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

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

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

۳-۵ عملیات بر روی رشته‌ها

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

برای مثال رشته زیر را در نظر بگیرید:

'TO BE OR NOT TO BE'

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

از سوی دیگر، یک آرایه خطی 18 عنصری با 18 عدد صحیح زیر را در نظر بگیرید:

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

واحد اصلی قابل دسترس در چنین آرایه‌ای، معمولاً تک‌تک عناصر آن آرایه است. گروههای متوالی

از عناصر این آرایه، معمولاً معنی خاصی ندارند. بنابراین بالا، عملیات متعددی روی رشته‌ها تعریف شده است که معمولاً با انواع دیگر آرایه‌ها بکار برده نمی‌شود. در این بخش، این عملیات روی رشته‌ها توضیح داده می‌شود. بخش بعد، چگونگی استفاده از این عملیات را در پردازش کلمات یا رشته‌ها نشان می‌دهد. بجز در حالی که به صورت صریح بیان می‌شود یا به طور ضمنی توضیح داده می‌شود، فرض می‌کیم متغیرهای نوع کاراکتری پویا هستند و طول متغیر دارند و طول آن، در متنی که از متغیر استفاده می‌کند، تعیین می‌شود.

زیر رشته

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

SUBSTRING(string, initial, length)

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

مثال ۳-۳

(الف) با استفاده از تابع بالا داریم:

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

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

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

شاخص گذاری

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

و می‌نویسیم :

INDEX(text, pattern)

اگر الگوی P در متن T ظاهر نشود، آنگاه در **INDEX** مقدار ۰ جایگزین می‌شود. آرگومان‌های $text$ و $pattern$ در عمل بالا، می‌توانند ثابت‌های رشته‌ای یا متغیرهای رشته‌ای باشند.

مثال ۳-۴

(الف) فرض کنید T شامل متن

'HIS FATHER IS THE PROFESSOR'

باشد آنگاه : $INDEX(T, ' \square THE \square ') = INDEX(T, 'THE') = INDEX(T, 'THEN')$ به ترتیب مقادیر ۷، ۰ و ۱۴ دارند.

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

PL/1: **INDEX(text, pattern)**

UCSD Pascal: **POS(pattern, text)**

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

اتصال دو رشته

فرض کنید S_1 و S_2 دو رشته باشند. از بخش ۲-۳ یادآور می‌شویم که اتصال S_1 و S_2 که ما آن را با $S_1 // S_2$ نمایش می‌دهیم، رشته‌ای است که شامل کاراکترهای S_1 و بدنال آن کاراکترهای S_2 است.

مثال ۳-۵

(الف) فرض کنید 'TWAIN' = S_2 باشد. آنگاه $S_1 = 'MARK' // S_2$

$S_1 // ' \square ' // S_2 = 'MARK TWAIN'$ اما $S_1 // S_2 = 'MARKTWAIN'$

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

PL/1: $S_1 || S_2$

FORTRAN 77: $S_1 // S_2$

BASIC: $S_1 + S_2$

SNOBOL: $S_1 S_2$

(بین S_1 و S_2 یک فضای خالی وجود دارد)

طول رشته

تعداد کاراکترهای داخل یک رشته، طول آن رشته نامیده می‌شود. برای تعیین طول یک رشته معلوم

می‌نویسیم:

`LENGTH(string)`

بدین ترتیب:

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

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

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

و **PASCAL** و **FORTRAN** استاندارد که از متغیرهای رشته‌ای با طول ثابت استفاده می‌کنند و هیچ تابع کتابخانه‌ای **LENGTH** برای تعیین طول رشته‌ها ندارند. با وجود این، هرگاه از تمام فضای خالی انتهایی رشته صرف‌نظر کنیم، می‌توان چنین متغیرهایی را متغیرهایی با طول متغیر در نظر گرفت. بنابراین می‌توان

در این زبانها، یک زیربرنامه **LENGTH** نوش特 طوری که: `LENGTH('MARC')=4`

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

`TRIM('ERIK ')='ERIK'`

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

۶-۳ پردازش کلمه یا رشته

در گذشته، داده‌های کاراکتری پردازش شده توسط کامپیوتز، اساساً از عناصر داده‌ای نظیر نام و آدرس تشکیل می‌شد. امروزه کامپیوتر، کارهای تایپی و نشریاتی از قبیل نامه‌ها، مقالات و گزارشها را پردازش می‌کند. بخاطر متنها و مطالب بعدی است که از اصطلاح "پردازش کلمه" یا رشته استفاده می‌کنیم. متن تایپ شده‌ای داده شده است. معمولاً عملیات مرتبط با پردازش کلمه یا رشته به قرار زیر است:

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

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

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

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

اضافه کردن

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

`INSERT(text, position, string)`

نمایش می‌دهیم. برای مثال:

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

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

$\text{INSERT}(T, K, S) = \text{SUBSTRING}(T, 1, K - 1) // S // \text{SUBSTRING}(T, K, \text{LENGTH}(T) - K + 1)$

یعنی زیررشته ابتدایی T که قبل از مکان K است و طول $1 - K$ دارد، به رشته S متصل شده است و نتیجه به بقیه قسمت T متصل می‌شود که از مکان K شروع می‌شود و طول $\text{LENGTH}(T) - (K - 1) = \text{LENGTH}(T) - K + 1$ دارد. ما به صورت ضمنی فرض کرده‌ایم که T یک متغیر پویا است و اندازه T خیلی بزرگ نمی‌شود.

حذف کردن

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

`DELETE(text, position, length)`

برای مثال

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

فرض می کنیم که عمل حذف زمانی که مکان $0 = K$ است انجام نمی شود. بنابراین اهمیت این "حالت صفر" دیرتر مشاهده می شود.

تابع DELETE را می توان با استفاده از عملیات بر روی رشته ها، که در بخش قبل به صورت زیر داده شده است پیاده سازی کرد:

$$\begin{aligned} \text{DELETE}(T, K, L) = \\ \text{SUBSTRING}(T, 1, K - 1) // \text{SUBSTRING}(T, K + L, \text{LENGTH}(T) - K - L + 1) \end{aligned}$$

يعنى زیررشته ابتدائي T که قبل از مکان K است به زيررشته انتهاي T که از مکان $K + L$ شروع می شود متصل شده است. طول زيررشته ابتدائي $1 - K$ است و طول زيررشته انتهاي برابر است با:

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

علاوه بر اين فرض می کنیم به ازای $0 = K$ DELETE(T, K, L) = T ،

حال فرض کنيد متن T و الگوی P داده شده است و بخواهیم از متن T اولین وقوع الگوی P را حذف

كنیم. اين عمل با استفاده از تابع DELETE بالا به صورت زير انجام می شود:

$$\text{DELETE}(T, \text{INDEX}(T, P), \text{LENGTH}(P))$$

يعنى در متن T ، نخست INDEX(T, P) را محاسبه می کنیم يعنی مکانی را که الگوی P برای اولین بار در T ظاهر شده است پیدا می کنیم و بدنبال آن LENGTH(P) يعنی تعداد کاراکتر های P را محاسبه می کنیم. يادآور می شویم که وقتی $0 = \text{INDEX}(T, P)$ (يعنى هنگامی که P در T ظاهر نشده باشد) متن T تغیير نمی کند.

مثال ۶-۳

(الف) فرض کنيد 'ABCDEFG' از $T = 'ABCDEFG'$ و $P = 'CD'$ و $\text{INDEX}(T, P) = 3$ و $\text{LENGTH}(P) = 2$ از اين رو

$$\text{DELETE}('ABCDEFG', 3, 2) = 'ABEFG'$$

(ب) فرض کنيد 'ABCDEFG' از $T = 'ABCDEFG'$ و $P = 'CD'$ و $\text{INDEX}(T, P) = 0$ آنگاه $0 = \text{INDEX}(T, P)$ و $\text{LENGTH}(P) = 2$ از اين رو، بنا به "حالت صفر":

$$\text{DELETE}('ABCDEFG', 0, 2) = 'ABCDEFG'$$

که همان نتیجه مورد انتظار است.

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

$$\text{DELETE}(T, \text{INDEX}(T, P), \text{LENGTH}(P))$$

انجام داد تا این که $\text{INDEX}(T, P) = 0$ (یعنی به محض این که P در T ظاهر نشده است).

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

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

1. [Find index of P .] Set $K := \text{INDEX}(T, P)$.
2. Repeat while $K \neq 0$:
 - (a) [Delete P from T .] Set $T := \text{DELETE}(T, \text{INDEX}(T, P), \text{LENGTH}(P))$
 - (b) [Update index.] Set $K := \text{INDEX}(T, P)$.
3. Write: T .
4. Exit.

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

مثال ۳-۷

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

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

آنگاه حلقة داخلی الگوریتم دوبار اجرا خواهد شد. در خلال اجرای اول الگوریتم، اولین وقوع AB از T حذف خواهد شد که درنتیجه آن $T = XYABZ$ به صورت $T = XYABZ$ درخواهد آمد. در خلال اجرای دوم، وقوع باقیمانده AB از T حذف خواهد شد. از این‌رو $T = XYZ$. بنابراین XYZ خروجی الگوریتم است.

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

$$T = XAAABBBY, \quad P = AB$$

ملاحظه می‌کنید که الگوی AB تنها یکبار در T ظاهر شده است اما حلقه الگوریتم سه بار اجرا می‌شود. به ویژه این‌که، پس از حذف AB در مرحله اول از T داریم $T = XAABBY$ و از این‌رو AB مجدداً در T ظاهر شده است. پس از حذف AB در مرحله دوم از T ملاحظه می‌کنید که $T = XABY$ و AB همچنان در T وجود دارد. بالاخره پس از حذف AB در مرحله سوم از T داریم $T = XY$ و AB در T ظاهر نشده است و بدین ترتیب $0 = \text{INDEX}(T, P)$. از این‌رو XY خروجی الگوریتم است.

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

جانشین‌سازی

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

$\text{REPLACE}(\text{text}, \text{pattern}_1, \text{pattern}_2)$

برای مثال:

$\text{REPLACE('XABYABZ', 'AB', 'C')} = 'XCYABZ'$
 $\text{REPLACE('XABYABZ', 'BA', 'C')} = 'XABYABZ'$

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

```
K := INDEX(T, P1)
T := DELETE(T, K, LENGTH(P1))
INSERT(T, K, P2)
```

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

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

$\text{REPLACE}(T, P, Q)$

تا وقتی که $0 = \text{INDEX}(T, P)$ می‌توان انجام داد (یعنی تا وقتی که الگوی P در T وجود نداشته باشد).

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

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

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

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

مثال ۳-۸

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

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

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

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

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

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

$T = XABY$ در پایان اجرای اول حلقه است.

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

.....

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

چون P یک زیرشته Q است از این‌رو در اینجا حلقة بی‌پایان **Infinite Loop** خواهیم داشت.

۷-۳. الگوریتم‌های تطبیق الگو

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

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

$$cdcdcd \quad aabbabbb \quad a^2b^3ab^2 \quad \text{به جای } (cd)^3$$

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

الگوریتم اول تطبیق الگو

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

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

باشد. به عبارت دیگر فرض کنید W_k زیررشته‌ای از T با همان طول P و با شروع از K امین کاراکتر T باشد. نخست P را کاراکتر با اولين زيرشته، W_1 مقایسه می‌کنیم. اگر تمام کاراکترها مساوی باشند در آن صورت $W_1 = P$ و درنتیجه P در T ظاهر شده است و $\text{INDEX}(T, P) = 1$. از سوی دیگر، فرض کنید به این نتیجه رسیدیم که یک کاراکتر P همان کاراکتر متناظر در W_1 نیست. در آن صورت \neq W_1 و بلاfacله می‌توانیم به زیرشته بعدی W_2 برویم به عبارت دیگر، در مرحله بعد P را با W_2 مقایسه می‌کنیم. اگر $W_2 = P$ ، در آن صورت P را با W_3 مقایسه می‌کنیم و الی آخر. عمل مقایسه متوقف می‌شود:

(الف) هرگاه به یک تطبیق P با زیرشته W_k برسیم و از این‌رو P در متن T وجود دارد و $\text{INDEX}(T, P) = K$ با (ب) هرگاه به هیچ تطبیق P با زیرشته W_k نرسیم و از این‌رو P در متن T وجود نخواهد داشت.

MAX مقدار ماگزینیم اندیس K برابر است با $\text{LENGTH}(T) - \text{LENGTH}(P) + 1$. حال به عنوان مثال فرض کنید که P یک رشته ۴ کاراکتری و T یک رشته ۲۰ کاراکتری باشد و فرض کنید P و T در حافظه به صورت آرایه‌های خطی نمایش داده شده‌اند که در هر عنصر آرایه یک کاراکتر ذخیره شده است. یعنی

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

آنگاه P با هر یک از زیررشته‌های ۴ کاراکتری بعدی T مقایسه می‌شود:

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

توجه داشته باشید که تعداد $17 = MAX = 20 - 4 + 1$ زیررشته ۴ کاراکتری در T وجود دارد. نمایش رسمی این الگوریتم که در آن P یک رشته R کاراکتری و T یک رشته S کاراکتری است در الگوریتم ۳.۳ نشان داده شده است.

ملحوظه می‌کنید الگوریتم ۳.۳ شامل دو حلقه است، یکی از حلقه‌ها داخل حلقه دیگر قرار دارد. حلقه خارجی برای هر زیررشته R کاراکتری متوالی

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

رشته T اجرا می‌شود. حلقه داخلی P را با W_k ، کاراکتر به کاراکتر مقایسه می‌کند. اگر هیچ کاراکتری تطبیق نکند، در آن صورت کنترل به مرحله S داده می‌شود که K را افزایش می‌دهد و پس از آن متنه به زیررشته بعدی T می‌شود. اگر تمام R کاراکتر P با کاراکترهای W_k تطابق داشته باشد، آنگاه P در T وجود دارد و K اندیس INDEX الگوی P در T است. از سوی دیگر، اگر حلقه خارجی به پایان تمام مراحل خود برسد ولی در T هیچ P ظاهر نشود در آن صورت $0 = \text{INDEX}$.

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

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

پیچیدگی این الگوریتم تعیین متن به وسیله C تعداد مقایسه بین کاراکترهای الگری P و کاراکترهای متن T اندازه گرفته می‌شود. برای پیدا کردن C ، فرض می‌کنیم N_k نمایش تعداد مقایسه‌هایی باشد که هنگام مقایسه P با W_k در حلقه داخلی اتفاق می‌افتد. در آن صورت

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

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

مثال بعدی C را برای یک P ی مشخص و T محاسبه می‌کند که در آن $4 = \text{LENGTH}(P)$ و $\text{MAX} = 20 - 4 + 1 = 17$ و $\text{LENGTH}(T) = 20$

مثال ۳-۹

(الف) فرض کنید $T = \text{cdcd...cd}^{10}$. واضح است که P در T ظاهر نشده است. علاوه بر این، برای هر ۱۷ بار اجرای حلقه، $N_k = 1$ ، چون اولین کاراکتر P با W_k مطابقت نمی‌کند. از این رو

$$C = 1 + 1 + 1 + \dots + 1 = 17$$

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

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

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

$$C = 4 + 4 + \dots + 4 = 17.4 = 68$$

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

$$n = r + s$$

بدترین حالت وقتی اتفاق می‌افتد که همانند مثال ۳-۹ (ج) تمام کاراکترهای P بجز آخرین کاراکتر با هر زیرشته W_k مطابقت داشته باشد. در این حالت، $(r(s - r + 1) + C(n) = r(s - r + 1) + n)$ برای مقدار ثابت n داریم

طوری که $s = n - r$

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

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

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

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

الگوریتم دوم تطبیق الگو

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

$$P = aaba$$

نخست دلیلی برای درایه‌های جدول اقامه می‌کنیم و چگونگی استفاده از آنها را شرح می‌دهیم. فرض کنید ... $T = T_1 T_2 T_3 \dots T$ که در آن T_i کاراکتر i ام T را نمایش می‌دهد و فرض کنید کاراکتر اول T با دو کاراکتر اول P مطابقت می‌کند یعنی فرض کنید $T = aa$. آنگاه T دارای یکی از سه صورت زیر است:

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

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

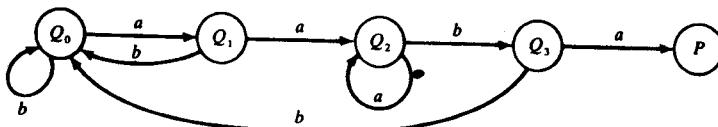
دو نکته قابل توجه و مهم در روش بالا وجود دارد. نخست، هنگام خواندن T_3 ، تنها لازم است T_3 با

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

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

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

(الف) جدول تطبیق الگو



(ب) گراف تطبیق الگو

شکل ۳-۸

طرز تشکیل جدول به صورت زیر است. قبل از هر کاری، فرض می‌کنیم Q_i نمایش زیرشته اولیه P با طول i باشد. از این رو

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

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

$$f(Q_1, t)$$

درایه سطر Q_1 و ستون t را نمایش دهد که در آن t می‌تواند هر کاراکتر دلخواهی باشد. درایه $f(Q_1, t)$ بنابراین بزرگترین Q ای است که در رشته t ، Q_1 ، یعنی درنتیجه اتصال Q_1 و t به صورت زیررشته نهایی ظاهر می‌شود. برای مثال:

$$a^2 \text{ بزرگترین } Q \text{ ای است که یک زیررشته پایانی } a^3 = Q_2a = Q_2 \text{ است از این رو } Q_2 = f(Q_2, a)$$

$$\wedge \text{ بزرگترین } Q \text{ ای است که یک زیررشته پایانی } Q_1b = ab = Q_0 \text{ است از این رو } Q_0 = f(Q_1, b)$$

$$\wedge \text{ بزرگترین } Q \text{ ای است که یک زیررشته پایانی } Q_0a = a = Q_1 \text{ است از این رو } Q_1 = f(Q_0, a)$$

$$\wedge \text{ بزرگترین } Q \text{ ای است که یک زیررشته پایانی } Q_3x = a^2bx = Q_3 \text{ است از این رو } Q_3 = f(Q_3, x)$$

$$\text{و الى آخر، اگرچه } a = Q_1 = \text{ زیررشته پایانی } Q_2a = a^3 \text{ است، داریم } f(Q_2, a) = Q_2 \text{ نیز یک}$$

$$\text{زیررشته پایانی } Q_2a = a^3 \text{ است و } Q_2 \text{ بزرگتر از } Q_1 \text{ است. توجه دارید که به ازای هر } Q,$$

$$Q = f(Q_1, x) \text{ چون } x \text{ در الگوی } P \text{ ظاهر نشده است، بنابراین، ستون متناظر با } x \text{ معمولاً از جدول حذف می‌شود.}$$

جدول را می‌توان با استفاده از گراف جهت دار برچسب‌گذاری شده شکل ۳-۸ (ب) رسم کرد. گراف به صورت زیر به دست می‌آید. نخست، متناظر با هر زیررشته اولیه Q_1 از P یک گره در گراف وجود دارد. Q ها حالتهای گراف نامیده می‌شوند و Q حالت اولیه نام دارد. دوم، متناظر با هر درایه در جدول، یک پیکان (یک یال جهت دار) در گراف وجود دارد. به ویژه این که، اگر

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

آنگاه یک پیکان با برچسب t از Q_1 به Q وجود دارد. برای مثال، $Q_3 = f(Q_2, b)$ از این رو یک پیکان با برچسب b از Q_2 به Q_3 وجود دارد. جهت سهولت در نمادگذاری، تمام پیکان‌های با برچسب x را که اجباراً متنهی به حالت اولیه Q می‌شوند حذف کرده‌ایم. اکنون آماده‌ایم که برای الگوی $P = aaba$ الگوریتم دوم تطبیق الگو را ارائه دهیم. توجه دارید که در بحث زیر، برای نام تمام متغیرهای یک حرکی داخل الگوریتم، از حروف بزرگ استفاده می‌کنیم. فرض کنید $T = T_1T_2T_3\dots T_n$ نمایش متن رشته کاراکتری باشد که در آن برای یافتن الگوی P عمل جستجو انجام می‌شود. با شروع از حالت اولیه Q و استفاده از متن T به یک دنباله از حالتهای S_1, S_2, S_3, \dots به صورت زیر دست می‌یابیم. فرض کنید $S_1 = Q$ و اولین کاراکتر T_1 را می‌خوانیم. چه از طریق جدول عمل کنیم و چه از گراف شکل ۳-۸ استفاده کنیم، زوج (S_1, T_1) حالت دوم S_2 را نتیجه می‌دهد یعنی $S_2 = f(S_1, T_1)$ ، کاراکتر بعدی T_2 را می‌خوانیم، زوج (S_2, T_2) حالت S_3 را نتیجه می‌دهد و الى آخر. دو حالت ممکن وجود دارد:

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

است. $K - \text{LENGTH}(P)$

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

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

مثال ۱۰ - ۳

(الف) فرض کنید $T = aabcaba$. با شروع از Q_0 و استفاده از کاراکترهای T و گراف (یا جدول) شکل ۳-۸، دنباله حالت‌های زیر به دست می‌آید:

$$Q_0 \xrightarrow{C_a} Q_1 \xrightarrow{C_a} Q_2 \xrightarrow{C_b} Q_3 \xrightarrow{C_c} Q_0 \xrightarrow{C_a} Q_1 \xrightarrow{C_b} Q_0 \xrightarrow{C_a} Q_1$$

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

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

$$Q_0 \xrightarrow{C_a} Q_1 \xrightarrow{C_b} Q_0 \xrightarrow{C_c} Q_0 \xrightarrow{C_a} Q_1 \xrightarrow{C_a} Q_2 \xrightarrow{C_b} Q_3 \xrightarrow{C_a} P$$

در اینجا، در حالت S_8 به الگوی P دست می‌یابیم. از این‌رو P در T ظاهر شده است و اندیس آن ۸ - $\text{LENGTH}(P) = 4$ است.

بيان رسمي الگوریتم دوم تطبیق الگو به صورت زیر است:

Algorithm 3.4: (Pattern Matching). The pattern matching table $F(Q_i, T)$ of a pattern P is in memory, and the input is an N -character string $T = T_1 T_2 \cdots T_N$. This algorithm finds the INDEX of P in T .

1. [Initialize.] Set $K := 1$ and $S_1 = Q_0$.
2. Repeat Steps 3 to 5 while $S_K \neq P$ and $K \leq N$.
 3. Read T_K .
 4. Set $S_{K+1} := F(S_K, T_K)$. [Finds next state.]
 5. Set $K := K + 1$. [Updates counter.]
 [End of Step 2 loop.]
6. [Successful?]
 - If $S_K = P$, then:
 - INDEX = $K - \text{LENGTH}(P)$.
 - Else:
 - INDEX = 0.
 [End of If structure.]
7. Exit.

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

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

توجه کنید: یک مسأله ترکیباتی را با زمان چندجمله‌ای، حل پذیر گویند اگر به ازای بعضی از مقادیر m یک جواب الگوریتمی با پیچیدگی ای برابر $O(m^n)$ ، وجود داشته باشد و با زمان خطی، حل پذیر گویند اگر یک جواب الگوریتمی با پیچیدگی ای برابر $O(n^m)$ وجود داشته باشد که در آن n تعداد داده‌ها است. بدین ترتیب حالت دوم دو الگوریتم تطبیق الگو که در این بخش تشریح شده است با زمان خطی حل پذیر هستند. الگوریتم اول تطبیق الگو با زمان چندجمله‌ای حل پذیر است.

مسأله‌های حل شده

اصطلاحات، ذخیره رشته‌ها

مسأله ۱-۳: فرض کنید W رشتة $ABCD$ است. (الف) طول W را پیدا کنید. (ب) تمام زیررشته‌های W را بنویسید. (ج) تمام زیررشته‌های اولیه W را بنویسید.

حل : (الف) تعداد کاراکترهای رشتة W طول آن است، بنابراین طول W برابر ۴ است.

(ب) هر زیردنباله‌ای از کاراکترهای W یک زیرشتة W است. در W ، ۱۱ زیرشتة وجود دارد:

Zیرشتة‌ها :	<u>ABCD</u> ,	<u>ABC</u> , <u>BCD</u> ,	<u>AB</u> , <u>BC</u> , <u>CD</u> ,	<u>A</u> , <u>B</u> , <u>C</u> , <u>D</u> ,	Λ
-------------	---------------	---------------------------	-------------------------------------	---	-----------

طول آنها :

4

3

2

1

0

در اینجا Λ معرف رشتة تهی است.

(ج) زیرشتة‌های اولیه عبارتند از $ABCD$ ، ABC ، BCD ، AB ، CD ، A ، B ، C ، D ، Λ ، یعنی هم رشتة تهی و هم زیرشتة‌های اولیه با A شروع می‌شوند.

مسأله ۲-۳: فرض کنید یک زبان برنامه‌نویسی حداقل از ۴۸ کاراکتر شامل ۲۶ حرف، ۱۰ رقم و حداقل ۱۲ کاراکتر مخصوص استفاده می‌کند. حداقل تعداد بیتها و تعداد بیتها متبادل برای نمایش یک کاراکتر در حافظه کامپیوتر را به دست آورید.

حل : از آنجا که $2^6 < 48 < 2^7$ ، حداقل یک کد ۶ بیتی برای نمایش این ۴۸ کاراکتر موردنیاز است. معمولاً یک کامپیوتر از یک گُدد ۷ بیتی نظیر ASCII یا یک کد ۸ بیتی نظیر کد EBCDIC برای نمایش کاراکترها استفاده می‌کند. این کدگذاری‌ها به کامپیوتر اجازه می‌دهند کاراکترهای مخصوص بسیار زیادتری را در

حافظه ایجاد کرده به نمایش و پردازش آنها بپردازند.

مسئله ۳-۳: به اختصار سه نوع ساختار مختلف را که برای ذخیره رشته‌ها بکار می‌روند توضیح دهید.

حل: (الف) ساختارهای حافظه با طول ثابت. در اینحالت رشته‌هایی که در خانه‌های حافظه ذخیره می‌شود همگی دارای طول مساوی هستند، معمولاً طول حافظه برای رشته‌ها ۸۰ کاراکتر است.

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

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

مسئله ۳-۴: رشته ذخیره شده در شکل ۳-۹ را تعیین کنید. فرض می‌شود مقدار پیوند ۰ علامت پایان لیست پیوندی است.

	CHAR	LINK
1	O Y F	10
2	ING	7
3		
4	A TH	2
5		
6	ER.	0
7	OF B	11
8	A J	1
9		
10	OREV	6
11	EAUT	12
12	Y IS	8

شکل ۳-۹

حل: در اینجا رشته در یک ساختمان لیست پیوندی ذخیره می‌شود که هر گره آن شامل ۴ کاراکتر است.

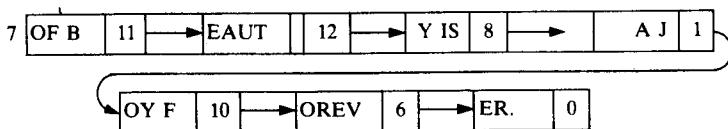
مقدار START، مکان گره اول لیست را به دست می‌دهد:

4	A TH	2
---	------	---

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

2	ING	7
---	-----	---

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



بنابراین رشته ذخیره شده عبارت است از:

A THING OF BEAUTY IS A JOY FOREVER.

مسئله ۵-۳: برخی از (الف) مزایا (ب) معایب استفاده از حافظه پیوندی برای ذخیره رشته‌ها را بیان کنید.

حل: (الف) هنگام استفاده از حافظه پیوندی به راحتی می‌توان عملیات اضافه کردن، حذف کردن، اتصال و جابجایی زیررشته‌ها را انجام داد.

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

مسئله ۶-۳: به اختصار درباره معنی (الف) متغیرهای کاراکتری ایستا (ب) نیمه‌ایستا (ج) پویا توضیح دهید.

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

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

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

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

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

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

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

(الف)

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

(ب)

عملیات بر روی رشته‌ها

در مسئله‌های ۳-۸ تا ۱۱-۳ و ۱۳-۳ فرض کنید S و T متغیرهای کاراکتری باشند، به گونه‌ای که :

$$S = 'JOHN PAUL JONES'$$

$$T = 'A THING OF BEAUTY IS A JOY FOREVER.'$$

مسئله ۳-۸ : یادآور می‌شویم که برای طول یک رشته از LENGTH(string) استفاده می‌کنیم.
 (الف) این تابع در زبانهای (الف) PL/I ، BASIC (ii) ، UCSD PASCAL (iii) ، SNOBOL (iv) و (v) FORTTRAN چگونه نمایش داده می‌شود؟

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

حل : (الف) .SIZE(string) (iv) . LENGTH(string) (iii) . LEN(string) (ii) . LENGTH(string) (i) .
 (v) FORTTRAN هیچ تابع طول برای رشته‌ها ندارد. چون این زبان تنها از متغیرهایی با طول ثابت استفاده می‌کند.

(ب) فرض می‌کیم بین کلمات تنها یک کاراکتر فضای خالی وجود دارد. از این‌رو
 $LENGTH(T) = 35$ و $LENGTH(S) = 15$

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

- مطلوب است تعیین (الف) SUBSTRINGS(S, 4, 8) و (ب) SUBSTRING(T, 10, 5)
 حل : (الف) با شروع از کاراکتر چهارم و ثبت ۸ کاراکتر، داریم :

$$SUBSTRING(S, 4, 8) = 'N\BoxPAUL\BoxJ'$$

$$SUBSTRING(T, 10, 5) = 'F\BoxBEAU'$$
(ب) به طور مشابه :

مسئله ۱۰ - ۳: یادآور می‌شویم که برای نمایش مکانی که یک الگو برای نخستین بار در متن T ظاهر می‌شود، از INDEX(TEXT, PATTERN) استفاده می‌کنیم. در این تابع مقدار ۰ جایگزین می‌شود اگر الگو در متن ظاهر نشده باشد. مطلوب است تعیین

$$\text{INDEX}(S, '□JO'), \quad \text{(ب)} \quad \text{INDEX}(S, 'JO'), \quad \text{(ج)} \quad \text{INDEX}(S, 'JO'), \\ \text{INDEX}(T, 'THE'), \quad \text{(د)} \quad \text{INDEX}(T, '□A□') \quad \text{(ه)} \quad \text{INDEX}(T, 'A')$$

حل: (الف) $\text{INDEX}(S, '□JO') = 10$, (ب) $\text{INDEX}(S, 'JO') = 1$, (ج) $\text{INDEX}(S, 'JO') = 0$,

$$\text{INDEX}(T, 'THE') = 0 \quad \text{(د)} \quad \text{INDEX}(T, '□A□') = 21 \quad \text{(ه)} \quad \text{INDEX}(T, 'A') = 1, \quad \text{(و)}$$

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

مسئله ۱۱ - ۳: یادآور می‌شویم که برای نمایش اتصال دو رشته S_1 و S_2 از $S_1 // S_2$ استفاده می‌کنیم.

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

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

(ج) مطلوب است تعیین (i) SUBSTRING(S, 11, 5) // '□' // SUBSTRING(S, 1, 9) و
SUBSTRING(T, 28, 3) // 'GIVEN' (ii)

حل: (الف) (i) $S_1 S_2$ (ii) $S_1 + S_2$, (iii) $S_1 // S_2$, (iv) $S_1 \parallel S_2$, (v) $\text{CONCAT}(S_1, S_2)$

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

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

مسئله ۱۲ - ۳: یادآور می‌شویم که برای نمایش اضافه کردن یک رشته S در یک متن داده شده T با شروع از مکان K از INSERT(text, position, string) استفاده می‌کنیم.

(الف) مطلوب است تعیین (i) INSERT('AAAAAA', 1, 'BBB'), (ii) و INSERT('AAAAAA', 6, 'BBB') (iii)

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

حل: (الف) (i) AAAAABBB (ii) BBBAAAAA (iii) و

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

یا (ii) $\text{INSERT}(\text{INSERT}(T, 19, '□ TODAY'), 15, '□ VERY')$ (iii) $\text{INSERT}(T, 19, '□ TODAY')$

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

مسئله ۱۳-۳: مطلوب است تعیین

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

و **REPLACE('AAABBB', 'AA', 'BB')** (ب)

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

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

(ب) REPLACE(T, P₁, P₂) در متن T اولین وقوع الگوی P₁ توسط الگوی P₂ جانشین می‌شود. از این رو جوابها عبارتند از BBABBB و JOHN DAVID JONES.

پردازش کلمه

در مسئله‌های ۱۴-۳ تا ۱۷-۳، S یک داستان کوتاه است که در یک آرایه خطی LINE با n عنصر به گونه‌ای ذخیره شده است که در آن هر LINE[K] یک متغیر کاراکتری است با قدرت ذخیره‌سازی 80 کاراکتر است و یک خط از داستان را نمایش می‌دهد. همچنین LINE[1] یعنی اولین خط داستان تنها شامل عنوان داستان است و LINE[N] آخرین خط داستان تنها شامل نام نویسنده است. علاوه براین، هر پاراگراف با ۵ فضای خالی شروع می‌شود و در هیچ کجا داستان بجز احتمالاً در عنوان [1] یا نام نویسنده [N] هیچ تورفتگی مانند ابتدای پاراگراف وجود ندارد.

مسئله ۱۴-۳: یک زیربرنامه Procedure بنویسید تا NUM تعداد پاراگراف‌های داستان کوتاه S را بشمارد.

حل: از خط [2] LINE[2] شروع می‌کنیم و با خط [1-N] LINE شمارش تعداد خطوط را که با 5 فضای

خالی شروع می‌شود به پایان می‌بریم. زیربرنامه به صورت زیر است:

Procedure P3.14: PAR(LINE, N, NUM)

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

مسئله ۱۵ - ۳: یک زیربرنامه procedure NUM تعداد دفعاتی را که کلمه the در این داستان کوتاه S ظاهر می‌شود بشمارد. the در کلمه mother را در شمارش شرکت ندهید و فرض می‌شود که هیچ جمله‌ای با کلمه the پایان نمی‌یابد.

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

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

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

مسئله ۱۶ - ۳: چنانچه بخواهیم تعداد دفعات وقوع کلمه دلخواهی مانند W را با طول r بشماریم توضیح دهید چه تغییراتی باید در Procedure P3.15 بدیم.

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

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

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

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

W, W; W. W?

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

مسئله ۱۷ - ۳: الگوریتمی را شرح دهید که پاراگرافهای K ام و L ام داستان کوتاه S را جابجا می‌کند.

حل: الگوریتم را به دو زیربرنامه Procedure تجزیه می‌کنیم.

زیربرنامه A: مقادیر آرایه‌های BEG و END را پیدا می‌کند که در آن

LINE[END[K]] و LINE[BEG[K]]

به ترتیب شامل اولین و آخرین خط پاراگراف K است. زیربرنامه B : با استفاده از مقادیر [END[K] و BEG[K] و مقادیر [BEG[L] و END[L]] بلک خطوط پاراگراف K با بلاک خطوط پاراگراف L جابجا می‌شود.

تطبيق الگو

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

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

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

حل : خاطرنشان می‌کنیم که در آن N_k تعداد مقایسه‌هایی را نشان می‌دهد که هنگام مقایسه P با W_k در حلقه داخلی انجام می‌شود. (الف) نخست توجه داشته باشید که تعداد

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

زیرشنوند W_k وجود دارد. داریم :

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

و $\text{INDEX}(T, P) = 0$ ، چون P در T ظاهر نشده است.

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

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

و $\text{INDEX}(T, P) = 0$

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

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

و $\text{INDEX}(T, P) = 0$

(د) داریم :

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

و $\text{INDEX}(T, P) = 7$

مسئله ۱۹-۳ : فرض کنید P یک رشته r کاراکتری و T یک رشته s کاراکتری باشد و فرض کنید هنگامی که الگوریتم ۳-۲ بر P و T اعمال می‌شود $C(n)$ نمایش تعداد مقایسه‌های صورت گرفته باشد. در اینجا

$$n = r + s$$

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

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

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

(ب) بنا به بحث بخش ۳-۷،

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

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

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

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

مسئله ۳-۲۰: الگوی $P = aaabb$ را در نظر بگیرید. جدول و گراف جهت دار برچسب‌گذاری شده متناظر با آن را که در الگوریتم "سریع" یا الگوریتم دوم تطبیق الگو مورد استفاده قرار گرفت، رسم کنید.

حل: نخست لیست قطعه‌های اولیه P را می‌نویسیم:

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

برای هر کارکتر t ، درایه $f(Q_i, t)$ جدول بزرگترین Q ای است که به صورت یک زیررشته انتهایی در رشته Q_it ظاهر می‌شود. محاسبه می‌کنیم:

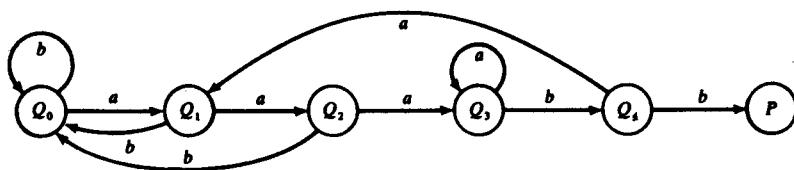
$$\begin{aligned} f(\Lambda, a) &= a, & f(a, a) &= a^2, & f(a^2, a) &= a^3, & f(a^3, a) &= a^3, & f(a^3b, a) &= a \\ f(\Lambda, b) &= \Lambda, & f(a, b) &= \Lambda, & f(a^2, b) &= \Lambda, & f(a^3, b) &= a^3b, & f(a^3b, b) &= P \end{aligned}$$

از این رو جدول موردنظر در شکل ۳-۱۰ (الف) ارائه شده است.

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

شکل ۳-۱۰ (الف)

گراف متناظر با آن در شکل ۳-۱۰ (ب) رسم شده است که در آن برای هر Q_i یک گره وجود دارد و پیکان از Q_i و Q_j برای هر درایه $f(Q_i, t) = Q_j$ داخل جدول باکاراکتر t برچسب‌گذاری شده است.



شکل ۳-۱۰ (ب)

مسئله ۳-۲۱: جدول و گراف متناظر با آن را برای الگوریتم دوم تطبیق الگو که در آن $P = ababab$ الگو است رسم کنید.

حل: زیرشته‌های اولیه P عبارتند از:

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

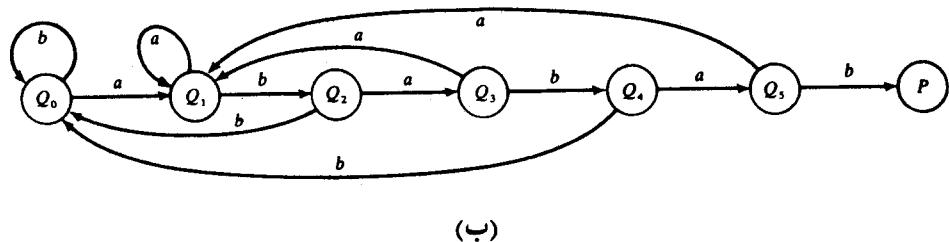
تابع f درایه‌های جدول را به صورت زیر به دست می‌دهد:

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

جدول در شکل ۳-۱۱ (الف) و گراف متناظر با آن در شکل ۳-۱۱ (ب) آمده است.

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

(الف)



شکل ۳-۱۱

مسأله‌های تكمیلی

رشته‌ها

مسأله ۳-۲۲: رشته ذخیره شده در شکل ۳-۱۲ را به دست آورید.

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

شکل ۳-۱۲

مسأله ۳-۲۳: رشته 'XYZST' = W را در نظر بگیرید. (الف) تمام زیررشته‌های W و

(ب) تمام زیررشه‌های ابتدایی W را بنویسید.

مسئله ۳-۲۴: فرض کنید W یک رشته به طول n است. مطلوب است تعیین تعداد (الف) زیررشه‌های W و (ب) زیررشه‌های ابتدایی W .

مسئله ۳-۲۵: فرض کنید STATE یک متغیر کاراکتری با طول ثابت 12 است. محتوای STATE را پس از جایگزینی‌های زیر توضیح دهید.

STATE := 'NEW YORK'، (الف)

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

عملیات بر روی رشته‌ها

در مسئله‌های ۳-۲۶ تا ۳-۳۱ فرض کنید S و T متغیرهای کاراکتری هستند به‌طوری که

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

مسئله ۳-۲۶: طول S و T را تعیین کنید.

مسئله ۳-۲۷: مطلوب است تعیین (الف) $\text{SUBSTRING}(S, 4, 8)$ و (ب) $\text{SUBSTRING}(T, 10, 5)$

مسئله ۳-۲۸: مطلوب است تعیین (الف) $\text{INDEX}(S, 'E')$ (ب) $\text{INDEX}(S, 'P')$ (ج) $\text{INDEX}(T, 'THEN')$ (ه) $\text{INDEX}(T, 'THE')$ (د) $\text{INDEX}(S, 'THE')$ (و) $\text{INDEX}(T, 'TE')$ و

مسئله ۳-۲۹: با استفاده از $S_1 // S_2$ به عنوان اتصال دو رشته S_1 و S_2 ، مطلوب است تعیین (الف)، 'NO' // 'EXIT' (ب) 'NO' // 'NO' و

(ج) $\text{SUBSTRING}(S, 4, 10) // \square \text{ARE} \square // \text{SUBSTRING}(T, 8, 6)$

مسئله ۳-۳۰: مطلوب است تعیین (الف) $\text{DELETE('AAABBB', 3, 3)}$,

(ب) $\text{DELETE}(T, 1, 7)$, (د) $\text{DELETE}(S, 1, 3)$ و (ج) $\text{DELETE('AAABBB', 1, 4)}$

مسئله ۳-۳۱: مطلوب است تعیین (الف) $\text{REPLACE('ABABAB', 'B', 'BAB')}$,

(ب) $\text{REPLACE}(T, 'THE', 'THESE')$ و (ج) $\text{REPLACE}(S, 'WE', 'ALL')$

مسئله ۳-۳۲: مطلوب است تعیین (الف) $\text{INSERT('AAA', 2, 'BBB')}$,

(ب) $\text{INSERT('THE BOY', 5, 'BIG \square')}$ و (ج) $\text{INSERT('ABCDE', 3, 'XYZ')}$

مسئله ۳-۳۳: فرض کنید U متن 'MARC STUDIES MATHEMATICS' باشد. با استفاده از INSERT

MARC STUDIES ONLY MATHEMATICS متن U را طوری تغییر دهید که (الف) $\text{MARC STUDIES APPLIED}$

(ج) $\text{MARC STUDIES MATHEMATICS AND PHYSICS}$ (ب) MATHEMATICS را بخواهد.

تطبیق الگو

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

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

مسئله ۳-۳۵: الگوی $a^5b = P$ را در نظر بگیرید. مسئله ۳-۳۴ را با هر یک از منتهای T زیر مجدداً حل کنید:

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

مسئله ۳-۳۶: الگوی $a^3ba = P$ را در نظر بگیرید. جدول و گراف جهت دار برچسب‌گذاری شده متناظر با آن را که در الگوریتم تطبیق الگو "سریع" مورد استفاده قرار گرفت رسم کنید.

مسئله ۳-۳۷: مسئله ۳-۳۶ را برای الگوی $b = abu^2b = P$ مجدداً حل کنید.

برای مسئله‌های زیر برنامه بنویسید

در مسئله‌های ۳-۳۸ تا ۳-۴۰، فرض می‌شود که پیشگفتار این متن در یک آرایه خطی LINE ذخیره شده است به طوری که LINE[K] یک متغیر کاراکتری است با قدرت ذخیره ۸ کاراکتر است و یک خط از پیشگفتار را نمایش می‌دهد. فرض کنید هر پاراگراف با ۵ فضای خالی شروع می‌شود و در هیچ جای دیگر پیشگفتار تورفتگی وجود ندارد. علاوه بر این فرض می‌شود یک متغیر NUM وجود دارد که تعداد خطوط پیشگفتار را به دست می‌دهد.

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

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

مسئله ۳-۴۰: یک برنامه بنویسید که پاراگرافهای J ام و K ام را جابجا کند. برنامه را با استفاده از $J = 2$ و $K = 4$ آزمایش کنید.

در مسئله‌های ۳-۴۱ تا ۳-۴۶ فرض می‌شود که پیشگفتار این متن در یک متغیر کاراکتری TEXT ذخیره شده است. فرض کنید ۵ فضای خالی میان یک پاراگراف جدید است.

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

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

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

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

مسئله ۴۵-۳: یک زیربرنامه **INST(TEXT, NEW, K)** بنویسید که رشته **NEW** را در داخل متن **TEXT** در آغاز **[K]** اضافه کند.

مسئله ۴۶-۳: یک زیربرنامه **PRINT(TEXT, K)** بنویسید که رشته کاراکتری **TEXT** را در خطوطی با حداکثر **K** کاراکتر چاپ کند. هیچ کلمه‌ای به دو قسمت تقسیم نشده و در دو خط ظاهر نمی‌شود، از این‌رو برخی از خطوط می‌توانند شامل چند فضای خالی در انتهایشان باشند. هر پاراگراف با خط مربوطه اش شروع شده و به کمک ۵ فضای خالی تورفتگی دارد. برنامه را با استفاده از (الف) **K = 80** (ب) **K = 70** و (ج) **K = 60** آزمایش کنید.

فصل ۲

آرایه‌ها، رکوردها و اشاره‌گرها

۱- مقدمه

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

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

(الف) نمایش. پردازش هر عنصر داخل لیست را پیماش گویند.

(ب) جستجوکردن. پیدا کردن مکان یک عنصر با یک مقدار داده شده یا رکورد با یک کلید معین را جستجو کردن گویند.

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

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

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

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

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

این فصل یک ساختار خطی کاملاً متداولی را مورد بررسی قرار می‌دهد که آرایه نام دارد. از آنجا که پیمایش، جستجو و مرتب کردن آرایه معمولاً ساده است از آنها اغلب برای ذخیره مجموعه هایی از داده های نسبتاً دائمی استفاده می شود. از سوی دیگر، اگر اندازه ساختار و داده های آن پیوسته در حال تغییر باشد آنگاه آرایه نمی تواند به خوبی ساختاری نظری لیست پیوندی باشد که در فصل ۵ بررسی می شود.

۴- آرایه های خطی

یک آرایه خطی لیستی از n یا تعداد متناهی عنصر داده ای همجناس است (یعنی عناصر داده ای از یک نوع هستند) بطوری که :

(الف) به عناصر آرایه به ترتیب به کمک یک مجموعه از اندیسها که شامل n عدد متوالی است رجوع می شود.

(ب) عناصر آرایه به ترتیب در خانه های متوالی حافظه ذخیره می شوند.

n یا تعداد عناصر آرایه طول یا اندازه آرایه نام دارد. اگر به صراحت بیان نشود فرض می کنیم که مجموعه اندیس ها شامل اعداد صحیح $1, 2, \dots, n$ است. در حالت کلی، طول یا تعداد عناصر داده ای آرایه را می توان از مجموعه اندیسها به کمک فرمول زیر به دست آورده:

$$(4-1) \quad \text{طول} = UB - LB + 1$$

که در آن UB بزرگترین اندیس یا کران بالای آرایه و LB کوچکترین اندیس یا کران پائین آرایه است. توجه دارید که وقتی $1 = LB$ باشد طول آرایه برابر UB است.

عناصر یک آرایه A را می توان با نماد اندیس گذاری

$$A_1, A_2, A_3, \dots, A_n$$

یا با نماد پرانتز گذاری (که در PL / I ، $FORTRAN$ و $BASIC$ بکار می رود)

$$A(1), A(2), \dots, A(N)$$

یا با نماد کروشهای (که در PASCAL بکار می‌رود)

$A[1], A[2], A[3], \dots, A[N]$

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

مثال ۴-۱

(الف) فرض کنید DATA یک آرایه خطی 6 عنصری از اعداد صحیح باشد به طوری که

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

گاهی اوقات ما این آرایه را فقط با نوشت

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

نمایش می‌دهیم. آرایه DATA غالباً به صورت شکل ۴-۱ (الف) یا شکل ۱-۴ (ب) نمایش داده می‌شود.

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

(ب)

(الف)

شکل ۴-۱

(ب) یک شرکت فروشنده اتومبیل از یک آرایه AUTO برای ثبت تعداد اتومبیل‌های فروخته شده هر سال از 1932 تا 1984 استفاده می‌کند. بجای اینکه مجموعه اندیسها از 1 شروع شود، بهتر است از 1932 شروع شود، طوری که :

AUTO[K] = تعداد اتومبیل‌های فروخته شده در سال K

آنگاه $LB = 1932$ کران پائین و $UB = 1984$ کران بالای آرایه **AUTO** است.

با توجه به فرمول (۴-۱)

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

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

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

مثال ۴-۲

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

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

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

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

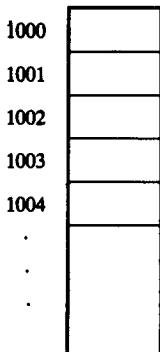
FORTRAN 77	INTEGER AUTO(1932:1984)
PL/I:	DECLARE AUTO(1932:1984) FIXED;
Pascal	VAR AUTO: ARRAY[1932..1984] of INTEGER

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

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

۴-۳ نمایش آرایه‌های خطی در حافظه

فرض کنید LA یک آرایه خطی در حافظه کامپیوتر باشد. یادآوری می‌کنیم که حافظه کامپیوتر فقط یک دنباله از مکانهای دارای آدرس است که در شکل ۲-۴ به تصویر کشیده شده است.



شکل ۲-۴. حافظه کامپیوتر

اجازه دهید از نماد زیر استفاده کنیم:

$$\text{LA}[\text{K}] = \text{LOC}(\text{LA}[\text{K}])$$

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

$$\text{Base(LA)}$$

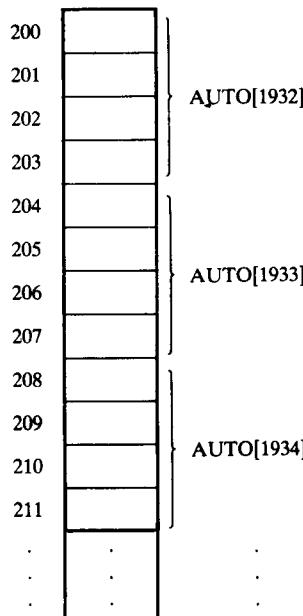
و آن را آدرس پایه یا مبنای LA می‌نامیم. با استفاده از آدرس پایه Base(LA)، کامپیوتر آدرس تمام عناصر دیگر آرایه LA را به کمک فرمول زیر محاسبه می‌کند:

$$\text{LOC(LA[K])} = \text{Base(LA)} + w(K - \text{lower bound}) \quad (4-2)$$

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

مثال ۴-۳

آرایه **AUTO** مثال ۱-۴ (ب) را در نظر بگیرید که تعداد اتمبیلهای فروخته شده هر سال از ۱۹۳۲ تا ۱۹۸۴ را ثبت می‌کند. فرض کنید **AUTO** در حافظه به صورت شکل ۴-۳ به نمایش در آمده است.



شکل ۴-۳

یعنی $\text{Base}(\text{AUTO}) = 200$ و $W=4$ کلمه در خانه حافظه برای **AUTO** باشد، آنگاه $\text{LOC}(\text{AUTO}[1932]) = 200$, $\text{LOC}(\text{AUTO}[1933]) = 204$, $\text{LOC}(\text{AUTO}[1934]) = 208$, ...

آدرس هر عنصر آرایه برای سال ۱۹۶۵ = K را می‌توان با استفاده از رابطه ۴-۲ بدست آورد.

$\text{LOC}(\text{AUTO}[1965]) = \text{Base}(\text{AUTO}) + w(1965 - \text{lower bound}) = 200 + 4(1965 - 1932) = 332$

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

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

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

۴-۴ پیمايش آرایه‌های خطی

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

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

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

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

علاوه بر این در شکل دیگری از این الگوریتم می‌توان از یک حلقه تکرار For – Repeat به جای حلقه Repeat – While استفاده نمود.

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

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

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

مثال ۴ - ۴

آرایه **AUTO** در مثال ۴ - ۴ (ب) را درنظر بگیرید که تعداد اتومبیلهای فروخته شده هر سال از ۱۹۳۲ تا ۱۹۸۴ را ثبت می‌کند. هر یک از قطعه برنامه‌های زیر که عمل داده شده را انجام می‌دهند شامل پیمایش **AUTO** است.

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

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

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

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

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

۵ - ۴ اضافه کردن و حذف کردن یک عنصر

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

هرگاه فضای حافظه اختصاص یافته برای یک آرایه خطی به اندازه کافی بزرگ باشد تا بتواند یک عنصر اضافی را در خود جای دهد عمل اضافه کردن یک عنصر در "پایان" آرایه خطی را می‌توان به سادگی انجام داد. از طرف دیگر فرض کنید بخواهیم یک عنصر به وسط آرایه اضافه کنیم. در آن صورت، به طور متوسط باید نصف عناصر به پائین انتقال (شیفت!) داده شوند در مکانهای جدید، عناصر جدید

قرار گیرند و نظم عناصر دیگر آرایه نیز حفظ شود.
به طور مشابه، حذف عنصری که در "پایان" یک آرایه است مشکل به نظر نمی‌آید اما حذف یک عنصر در مکان‌هایی مانند وسط آرایه نیازمند آن است که هر عنصر بعدی به مکان جدیدی در بالای آرایه انتقال داده شوند تا فضای خالی شده بالای آرایه را پر کند.

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

مثال ۵-۴

فرض کنید TEST به صورت یک آرایه ۵ عنصری معرفی شده است اما داده‌ها تنها در خانه‌های TEST[۱]، TEST[۲] و TEST[۳] ذخیره شده است. اگر X مقدار محتوای خانه بعدی TEST باشد، آنگاه تنها با دستور جایگزینی

TEST[۴] := X

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

TEST[۵] := Y

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

مثال ۶-۴

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

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

	NAME
1	Brown
2	Ford
3	Johnson
4	Smith
5	Taylor
6	Wagner
7	
8	

(5)

	NAME
1	Brown
2	Davis
3	Ford
4	Johnson
5	Smith
6	Taylor
7	Wagner
8	

(ج)

	NAME
1	Brown
2	Davis
3	Ford
4	Johnson
5	Smith
6	Wagner
7	
8	

(ب)

NAME
Brown
Davis
Johnson
Smith
Wagner

(الف)

شکل ۴-۴

الگوریتم زیر یک عنصر داده‌ای ITEM را در مکان K ام آرایه خطی LA که دارای N عنصر است اضافه می‌کند. چهار مرحله اول، هر عنصر از مکان K ام به بعد را یک خانه به پائین منتقل می‌کند و در آرایه LA فضای خالی ایجاد می‌کند. تأکید می‌کنیم که این عناصر به ترتیب عکس انتقال می‌یابند یعنی اول LA[N-1] و ... و بالاخره LA[K] در غیر اینصورت داده‌ها ممکن است پاک شوند. (مسئله ۳-۴ را بینید). مشروط حاصل این که، نخست قرار می‌دهیم $N = J$: آنگاه با استفاده از J به عنوان یک شمارنده، با افزایش J هر بار، حلقه اجرا می‌شود تا اینکه J به K برسد. در مرحله بعدی یعنی مرحله ۵، ITEM داخل آرایه در فضایی که اکنون ایجاد شده است اضافه می‌شود. قبل از پایان (خروج از) الگوریتم، N تعداد عناصر آرایه LA یک واحد افزایش می‌یابد تا عنصر جدید را درپر بگیرد.

Algorithm 4.2: (Inserting into a Linear Array) INSERT(LA, N, K, ITEM)

Here LA is a linear array with N elements and K is a positive integer such that $K \leq N$. This algorithm inserts an element ITEM into the Kth position in LA.

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

الگوریتم زیر عنصر K از آرایه خطی LA حذف می‌کند و آن را در متغیر ITEM جایگزین می‌کند:

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

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

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

۶-۴ مرتب کردن، مرتب کردن حبابی

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

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

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

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

پس از مرتب کردن، A به صورت لیست زیر درخواهد آمد:

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

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

توجه کنید: تعریف بالا از مرتب کردن عبارت است از قراردادن داده‌های عددی به ترتیب صعودی در کنار هم. این محدودیت فقط برای سهولت در نمایندگزاری است. راجح است که: قراردادن داده‌های عددی به ترتیب نزولی در کنار هم یا قراردادن داده‌های غیر عددی به ترتیب الفبایی در کنار هم نیز مرتب کردن می‌گویند. در حقیقت A اغلب یک فایل از رکوردها است و منظور از مرتب کردن A، عبارت است از قراردادن رکوردهای A کنار هم به طوری که بر حسب مقدارهای یک کلید داده مرتب باشند.

مرتب کردن حبابی

فرض کنید لیستی از اعداد $A[1], A[2], \dots, A[N]$ در حافظه است. الگوریتم مرتب کردن حبابی به صورت زیر کار می‌کند:

مرحله ۱ . $A[1] \text{ و } A[2]$ را با هم مقایسه کنید و آنها را با ترتیب خواسته شده کنار هم قرار دهید طوری که $A[2] < A[1]$ آنگاه $A[2]$ و $A[3]$ را با هم مقایسه کنید و آنها را طوری کنار هم قرار دهید که $A[3] < A[2]$ آنگاه $A[3]$ و $A[4]$ را با هم مقایسه کنید و آنها را طوری کنار هم قرار دهید که $A[4] < A[3]$ این کار را تا آنجا ادامه دهید که $A[N - 1] \text{ و } A[N]$ با هم مقایسه شده باشند و آنها را طوری کنار هم قرار دهید که $A[N - 1] < A[N]$.

ملاحظه می‌کنید که در مرحله ۱ ، $1 - n$ مقایسه انجام می‌شود. طی مرحله ۱ ، بزرگترین عنصر در مکان n ام در وضعیت صعودی یا در وضعیت نزولی است. پس از کامل شدن مرحله ۱ ، $A[N]$ شامل بزرگترین عنصر خواهد بود.

مرحله ۲ . مرحله ۱ را با یک مقایسه کمتر تکرار کنید. یعنی اکنون پس از مقایسه و احتمالاً قرارگرفتن $A[N - 1] \text{ و } A[N]$ کنار هم کار را متوقف می‌کنیم. مرحله ۲ شامل $N - 2$ مقایسه است و پس از کامل شدن مرحله ۲ ، دومین عنصر بزرگ در مکان $1 - A[N]$ جای خواهد گرفت.

مرحله ۳. مرحله ۱ را با دو مقایسه کمتر تکرار کنید یعنی پس از مقایسه و احتمالاً قرارگرفتن $A[N - 2] \text{ و } A[N - 3]$ کنار هم کار را متوقف می‌کنیم.

.....

مرحله $1 - N$. $A[1] \text{، } A[2] \text{، } \dots \text{، } A[N]$ را با هم مقایسه کنید و آنها را طوری کنار هم قرار دهید که $A[1] < A[2] < \dots < A[N]$. پس از $1 - n$ مرحله، لیست به صورت صعودی مرتب خواهد شد. فرایند متوالی پیمایش تمام یا قسمتی از یک لیست غالباً یک "گذر" یا "مرحله" Pass نامیده می‌شود. از این رو هر یک از مراحل بالا یک گذر نامیده می‌شود. بنابراین الگوریتم مرتب کردن حبابی به $1 - n$ گذر احتیاج دارد که در آن n تعداد داده‌های ورودی است.

مثال ۴-۷

فرض کنید اعداد زیر در آرایه A ذخیره شده‌اند:

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

مرتب کردن حبابی را بر روی آرایه A بکار می‌بندیم و هر گذر را به طور جداگانه مورد بحث و بررسی

قرار می‌دهیم:

گذر ۱. مقایسه‌های زیر را داریم.

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

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

$$32, \textcircled{27}, \textcircled{51}, 85, 66, 23, 13, 57$$

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

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

$$32, 27, 51, \textcircled{66}, \textcircled{85}, 23, 13, 57$$

(ه) A_5 و A_6 را مقایسه کنید. از آنجا که $23 < 85$ ، $85 < 23$ را به صورت زیر جابجا کنید:

$$32, 27, 51, 66, \textcircled{23}, \textcircled{85}, 13, 57$$

(و) A_6 و A_7 را مقایسه کنید. از آنجا که $13 < 85$ ، با جابجا بی 85 و 13 نتیجه می‌شود:

$$32, 27, . 51, 66, 23, \textcircled{13}, \textcircled{85}, 57$$

(ز) A_7 و A_8 را مقایسه کنید. از آنجا که $57 < 85$ ، با جابجا بی 85 و 57 نتیجه می‌شود:

$$32, 27, 51, 66, 23, 13, \textcircled{57}, \textcircled{85}$$

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

در مورد گذرهای باقیمانده ما تنها به نشان دادن جابجا بی اکتفا می‌کنیم:

گذر ۲

$$\textcircled{27}, \textcircled{33}, 51, 66, 23, 13, 57, 85$$

$$27, 33, 51, \textcircled{23}, \textcircled{66}, 13, 57, 85$$

$$27, 33, 51, 23, \textcircled{13}, \textcircled{66}, 57, 85$$

$$27, 33, 51, 23, 13, \textcircled{57}, \textcircled{66}, 85$$

در پایان گذر ۲، دومین عنصر بزرگ آرایه یعنی ۶۶ به پائین و در همسایگی آخرین عنصر آرایه انتقال می‌یابد.
گذر ۳.

27, 33, $\textcircled{23}$, $\textcircled{51}$, 13, 57, 66, 85
27, 33, 23, $\textcircled{13}$, $\textcircled{51}$, 57, 66, 85

گذر ۴.

27, $\textcircled{23}$, $\textcircled{33}$, 13, 51, 57, 66, 85
27, 23, $\textcircled{13}$, $\textcircled{33}$, 51, 57, 66, 85

گذر ۵.

$\textcircled{23}$, $\textcircled{27}$, 13, 33, 51, 57, 66, 85
23, $\textcircled{13}$, $\textcircled{27}$, 33, 51, 57, 66, 85

گذر ۶.

$\textcircled{13}$, $\textcircled{23}$, 27, 33, 51, 57, 66, 85

گذر ۶ در حقیقت دارای دو مقایسه A_1 با A_2 و A_3 است. مقایسه دوم باعث جابجایی نمی‌شود.
گذر ۷. بالاخره A_1 با A_2 مقایسه می‌شود. چون $23 < 23$. جابجایی صورت نمی‌گیرد. از آنجاکه لیست دارای ۸ عنصر است، پس از گذر هفتم به صورت مرتب شده درخواهد آمد. ملاحظه می‌کنید که در این مثال، لیست واقعاً پس از گذر ششم مرتب شده خواهد بود. این وضعیت در پایان بخش بررسی می‌شود.
اکنون به صورت رسمی، الگوریتم مرتب کردن حبابی را بیان می‌کنیم.

Algorithm 4.4: (Bubble Sort) BUBBLE(DATA, N)

Here DATA is an array with N elements. This algorithm sorts the elements in DATA.

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

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

پیچیدگی الگوریتم مرتب‌کردن حبابی

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

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

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

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

۴-۷. جستجو کردن، جستجوی خطی

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

اگلی ممکن است بخواهیم پس از یک جستجوی ناموفق ITEM در DATA، عنصر ITEM را به DATA اضافه کنیم. آنگاه به جای تنها یک الگوریتم جستجو، از الگوریتم جستجو و اضافه کردن استفاده می‌کنیم. الگوریتم‌های جستجو و اضافه کردن در بخش مسائل به طور مسروچ توضیح داده می‌شود. الگوریتم‌های جستجوی بسیار متفاوتی وجود دارد. معمولاً الگوریتمی که انتخاب می‌شود بستگی به سازماندهی اطلاعات در DATA دارد. عمل جستجو به تفصیل در فصل ۹ شرح داده می‌شود. این بخش الگوریتم ساده‌ای را توضیح می‌دهد که جستجوی خطی نام دارد و در بخش بعد الگوریتم معروف جستجوی دودویی را مورد بحث و بررسی قرار می‌دهیم.

بیچیدگی الگوریتم‌های جستجو بر حسب تعداد مقایسه‌های موردنیاز (n^f) برای پیدا کردن ITEM در DATA اندازه‌گیری می‌شود که در آن DATA شامل n عنصر است. نشان خواهیم داد که جستجوی خطی یک الگوریتم دارای زمان خطی است اما جستجوی دودویی، الگوریتمی با کارآیی به مرتب بیشتر است و زمان آن متناسب با $\log_2 n$ است. از سوی دیگر معایب اعتماد بیش از حد به الگوریتم جستجوی دودویی را نیز بررسی می‌کنیم.

جستجوی خطی

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

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

$$LOC = N + 1$$

است که در آن LOC مکانی را نشان می‌دهد که ITEM برای اولین بار در DATA ظاهر شده است و مبنی ناموفق بودن عمل جستجو است. هدف از این جایگزینی اولیه اجتناب از آزمایش تکراری است که آیا به انتهای آرایه DATA رسیده‌ایم یا خیر. با این روش در نهایت باید عمل جستجو "موفق" باشد. نمایش رسمی جستجوی خطی در الگوریتم ۴-۵ نشان داده شده است.

Algorithm 4.5: (Linear Search) LINEAR(DATA, N, ITEM, LOC)

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

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

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

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

از طرف دیگر، برای استفاده از مرحله ۱، باید تضمین کنیم که در پایان آرایه DATA یک خانه حافظه استفاده نشده وجود دارد، در غیر اینصورت باید از الگوریتم جستجوی خطی که در الگوریتم ۴-۲ شرح داده شده، استفاده کنیم.

مثال ۴-۸

آرایه NAME شکل ۴-۵ (الف) را که در آن $n = 6$ است، درنظر بگیرید.

(الف) فرض کنید بخواهیم تحقیق کنیم که آیا Paula در آرایه وجود دارد یا خیر؟ در صورت مثبت بودن جواب، مکان آن را پیدا کنیم. الگوریتم به طور موقت Paula را در پایان آرایه قرار می‌دهد. این وضعیت در شکل ۴-۵ (ب) با قراردادن $NAME[7] = paula$ نشان داده شده است. آنگاه این الگوریتم عمل جستجو در آرایه را از بالا تا پائین انجام می‌دهد. از آنجا که Paula برای نخستین بار در $NAME[N+1]$ ظاهر شده است، $NAME[7]$ در آرایه اصلی وجود ندارد.

(ب) فرض کنید بخواهیم تحقیق کنیم که آیا Susan در آرایه وجود دارد یا خیر؟ در صورت مثبت بودن جواب، مکان آن را پیدا کنیم. الگوریتم به طور موقت Susan را در پایان آرایه قرار می‌دهد. این وضعیت در شکل ۴-۵ (ب) با قراردادن $NAME[7] = Susan$ نشان داده شده است. آنگاه این الگوریتم عمل

جستجو در آرایه را از بالا تا پائین انجام می‌دهد. از آنجاکه **Susan** برای نخستین بار در [4] که $n \leq 4$ است) ظاهر شده است، می‌فهمیم که **Susan** در آرایه اصلی وجود دارد.

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

(ج)

(ب)

(الف)

شکل ۴-۵

پیچیدگی الگوریتم جستجوی خطی

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

واضح است که بدترین حالت وقتی اتفاق می‌افتد که عمل جستجو در تمام آرایه **DATA** انجام شود و

DATA در **ITEM** نشده باشد. در این حالت، الگوریتم نیازمند

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

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

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

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

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

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

$$\begin{aligned} f(n) &= 1 \cdot \frac{1}{n} + 2 \cdot \frac{1}{n} + \dots + n \cdot \frac{1}{n} + (n+1) \cdot 0 = (1+2+\dots+n) \cdot \frac{1}{n} \\ &= \frac{n(n+1)}{2} \cdot \frac{1}{n} = \frac{n+1}{2} \end{aligned}$$

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

۴-۸ جستجوی دودویی

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

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

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

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

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

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

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

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

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

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

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

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

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

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

در آغاز، کار را با تمام آرایه DATA یعنی با **1 = BEG** و **n = END** یا با بیان کلی تر با **BEG = LB** و

END = UB شروع می‌کنیم.

اگر ITEM در DATA ظاهر نشده باشد آنگاه نهایتاً به

END < BEG

می‌رسیم که اعلام می‌کند جستجو ناموفق است و در چنین حالتی دستور جایگزینی **LOC = NULL** را داریم. در اینجا NULL مقداری است که در محدوده مجموعه اندیشهای DATA قرار ندارد. در بیشتر

موارد می‌توانیم NULL را برابر صفر اختیار کنیم. (**NULL = 0**)

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

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

Algorithm 4.6: (Binary Search) **BINARY(DATA, LB, UB, ITEM, LOC)**

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

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

مثال ۴-۹

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

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

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

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

(۱) در آغاز 1 و BEG = 13 و END = 55 از این رو

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

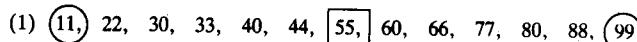
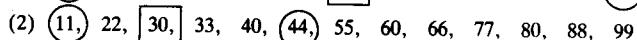
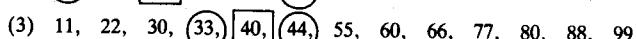
(۲) چون 55 < 40، با مقدار END = MID - 1 = 6 تغییر می‌کند. از این رو

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

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

$\text{DATA[MID]} = 40$ و در نتیجه $\text{MID} = \text{INT}[(4 + 6) / 2] = 5$

را در مکان $\text{LOC} = \text{MID} = 5$ ITEM پیدا کرده‌ایم.

- (1) 
- (2) 
- (3) 

[موفق]

شکل ۶-۴. جستجوی دودویی برای $\text{ITEM} = 40$

(ب) فرض کنید $\text{ITEM} = 85$. جستجو برای ITEM در آرایه DATA در شکل ۶-۷ به تصویر درآمده است. در اینجا BEG و END مقادیر متوالی زیر را دارا هستند:

• $\text{DATA[MID]} = 55$ و $\text{MID} = 7$, $\text{END} = 13$, $\text{BEG} = 1$ (۱) مجدداً در آغاز ۱

(۲) چون $55 < 85$, BEG با مقدار 8 تغییر می‌کند. از این رو

$\text{DATA[MID]} = 77$ و در نتیجه $\text{MID} = \text{INT}[(8 + 13) / 2] = 10$

(۳) چون $77 < 85$, BEG با مقدار 11 تغییر می‌کند. از این رو

$\text{DATA[MID]} = 88$ و در نتیجه $\text{MID} = \text{INT}[(11 + 13) / 2] = 12$

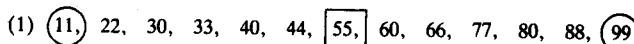
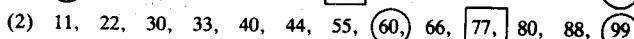
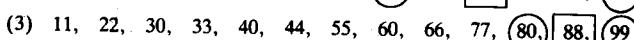
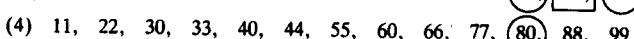
(۴) چون $88 < 85$, END با مقدار 11 تغییر می‌کند. از این رو

$\text{DATA[MID]} = 80$ و در نتیجه $\text{MID} = \text{INT}[(11 + 11) / 2] = 11$

• $\text{BEG} = \text{END} = \text{MID} = 11$ ملاحظه می‌کنید که اکنون

چون $80 < 85$, BEG با مقدار 12 تغییر می‌کند. اما اکنون $\text{BEG} > \text{END}$. از این رو

در DATA ITEM وجود ندارد.

- (1) 
- (2) 
- (3) 
- (4) 

[ناموفق]

شکل ۶-۷. جستجوی دودویی برای $\text{ITEM} = 85$

پیچیدگی الگوریتم جستجوی دودویی

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

$$f(n) = \lfloor \log_2 n \rfloor + 1 \quad \text{یا معادل آن} \quad 2^{f(n)} > n$$

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

مثال ۴-۱۰

فرض کنید DATA شامل 1000 000 عنصر باشد. ملاحظه می‌کنید که :

$$2^{20} > 1000^2 = 1\ 000\ 000 \quad \text{واز این‌رو} \quad 1024 > 1000 = 2^{10}$$

بنابراین با استفاده از الگوریتم جستجوی دودویی تنها به حدود 20 مقایسه احتیاج است تا مکان یک عنصر ITEM در آرایه DATA با 1000 000 عنصر پیدا شود.

محدودیتهای الگوریتم جستجوی دودویی

از آنجا که الگوریتم جستجوی دودویی از کارآیی بالایی برخوردار است مثلاً برای یک لیست اولیه 1000 000 عنصری تنها حدوداً به 20 مقایسه احتیاج است، چرا با وجود این، می‌خواهیم از الگوریتم جستجوی دیگری استفاده کنیم؟ ملاحظه کردید که این الگوریتم احتیاج به دو شرط دارد: (۱) لیست بایستی مرتب شده، باشد و (۲) بایستی به عنصر وسط هر زیرلیست، دسترسی مستقیم داشته باشیم. به این معنی که اساساً باید از یک آرایه مرتب شده برای نگهداری داده‌ها استفاده کرد. اما وقتی عملیات اضافه و حذف کردن زیادی در آرایه مورد نیاز باشد، نگهداری داده‌ها در یک آرایه مرتب شده معمولاً پرهزینه است. بنابراین در چنین وضعیتها بایی می‌توانیم از ساختمان داده متفاوتی نظری لیستهای پیوندی یا درخت جستجوی دودویی استفاده کرده تا داده‌ها را در آنها ذخیره کنیم.

۴-۹ آرایه‌های چندبعدی

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

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

آرایه‌های دو بعدی

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

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

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

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

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

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

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

شکل ۸-۴. آرایه 3×4 دو بعدی

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

مثال ۱۱-۴

فرض کنید تمام دانشجویان یک کلاس ۲۵ نفره در ۴ آزمون امتحانی شرکت کرده‌اند. فرض می‌شود

دانشجویان از ۱ تا ۲۵ شماره گذاری شده‌اند. نمرات آزمون را می‌توان در یک آرایه ماتریسی 4×25 بنام SCORE به صورتی که در شکل ۴-۹ نشان داده شده است جایگزین کرد.

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

شکل ۴-۹. آرایه SCORE

بنابراین $SCORE[K, L]$ حاوی نمره دانشجوی K ام در آزمون L ام است.
 $SCORE[2, 1]$, $SCORE[2, 2]$, $SCORE[2, 3]$, $SCORE[2, 4]$

حاوی ۴ نمره آزمون دانشجوی دوم است.

فرض کنید A یک آرایه $m \times n$ دو بعدی باشد. اولین بعد A شامل مجموعه اندیشهای ۱، ۲، ..., m با کران پائین ۱ و کران بالای m است و دومین بعد A شامل مجموعه اندیشهای ۱، ۲، ..., n با کران پائین ۱ و کران بالای n است. طول یک بعد، تعداد اعداد صحیح موجود در مجموعه اندیشهای آن است. جفت طول‌های $m \times n$ (بخوانید m در n) اندازه آرایه نامیده می‌شود.

برخی از زبان‌های برنامه‌نویسی به برنامه‌نویسی که آرایه‌های چند بعدی تعریف کنند که آرایه‌های چند بعدی نظم یا نامنظم نامیده می‌شوند. با وجود این، مجموعه اندیشهای هر بعد همچنان متشكل از اعداد صحیح متوالی از کران پائین تا کران بالای بعد است. طول یک بعد معین یعنی تعداد اعداد صحیح موجود در مجموعه اندیشهای آن، از فرمول زیر بدست می‌آید:

$$\text{Length} = \text{upper bound} - \text{lower bound} + 1 \quad (4-3)$$

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

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

مثال فرض کنید **DATA** یک آرایه 8×4 دو بعدی با عناصری از نوع اعشاری باشد. **FORTRAN**, **I**, **PL / I** و **PASCAL** چنین آرایه‌ای را به صورت زیر معرفی می‌کنند:

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

ملاحظه می‌کنید که در پاسکال کرانهای پائین حتی اگر این کران 1 باشد هنگام معرفی آرایه ذکر می‌شود. توجه کنید: زبانهای برنامه‌نویسی ای که توانایی معرفی آرایه‌های نامنظم را دارند معمولاً برای جداسازی کران پائین از کران بالای هر بُعد از یک کولن و برای جداسازی ابعاد، از هم از یک کاما استفاده می‌کنند. برای مثال در **FORTRAN**:

INTEGER NUMB(2:5, - 3:1)

NUMB را به صورت یک آرایه دو بعدی از نوع صحیح معرفی می‌کند. در اینجا مجموعه اندیسه‌های بُعدها به ترتیب از اعداد صحیح

- 3, - 2, - 1, 0, 1 2, 3, 4, 5

تشکیل شده‌اند. بنابراین فرمول $(-3 - 4) \times (2 - 5) = 4 \times 5 = 20$ و طول بعد اول برابر $4 + 1 = 5$ و طول بعد دوم برابر $5 - 1 = 4$ است. بنابراین NUMB دارای $4 \times 5 = 20$ عنصر است.

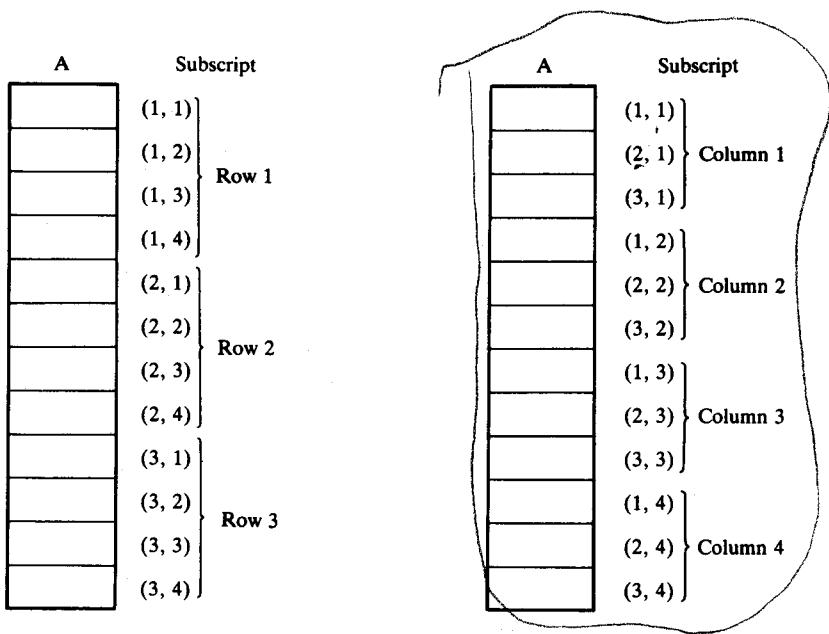
نمایش آرایه‌های دو بعدی در حافظه

فرض کنید A یک آرایه $m \times n$ دو بعدی باشد. هر چند A به صورت یک آرایه مستطیلی از عناصر با m سطر و n ستون به تصویر کشیده شده است، اما این آرایه در حافظه کامپیوتر توسط یک بلاک با $m \times n$ خانه متواالی حافظه نمایش داده می‌شود. بدین معنی که، زبان برنامه‌نویسی آرایه را به دو صورت ۱- ستون به ستون نیز نامیده می‌شود. ۲- سطر به سطر که روش سطروی نیز نامیده می‌شود. ذخیره می‌کنند. شکل ۱۰-۴ وقته که A یک آرایه دو بعدی 4×3 است را با این دو روش نشان می‌دهد. تأکید می‌کنیم که انتخاب هر یک از این دو نمایش بستگی به زبان برنامه‌نویسی دارد نه به کاربر یا برنامه‌نویس.

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

* - زبان FORTRAN آرایه‌ها را به صورت ستونی و زبان‌های Algol - like مانند زبان C و Pascal، آرایه‌ها را به صورت سطروی، در آرایه یک بعدی ذخیره می‌کنند. متوجه

$$\text{LOC}(\text{LA}[K]) = \text{Base}(\text{LA}) + w(K - 1)$$



(ب) روش سطري

(الف) روش ستونی

شکل ۴-۱۰

برای پیدا کردن آدرسی $\text{LA}[K]$ به موقع و بدون در نظر گرفتن K استفاده می‌کند. در اینجا W تعداد کلمات در حافظه یا طول کلمه برای آرایه LA است و یک، کران پائین مجموعه‌اندیس LA است. از وضعیت مشابه وضعیت بالا برای نگهداری هر آرایه دو بعدی A که $m \times n$ است استفاده می‌شود. به عبارت دیگر کامپیوتر آدرس پایه $\text{Base}(A)$ آدرس اولین عنصر A یعنی $A[1, 1]$ را نگه می‌دارد و آدرس $\text{LOC}(A[J, K])$ را با استفاده از فرمول

$$\text{LOC}(A[J, K]) = \text{Base}(A) + w[M(K - 1) + (J - 1)] \quad (4-4)$$

و فرمول

$$\text{LOC}(A[J, K]) = \text{Base}(A) + w[N(J - 1) + (K - 1)] \quad (4-5)$$

محاسبه می‌کند. در این جایز W تعداد کلمات خانه حافظه یا طول کلمه را برای آرایه A نمایش می‌دهد.

توجه دارید که فرمولها بر حسب J و K خطی هستند و می‌توان آدرس $LOC(A[J, K])$ را به موقع بدون در نظر گرفتن K به دست آورد.

مثال ۱۲-۴

آرایه‌ماتریسی $SCORE$ در مثال ۱۱-۴ را که 4×25 است در نظر بگیرید. فرض کنید $W=4$ کلمه در خانه حافظه وجود داشته باشد. علاوه بر این فرض می‌شود زبان برنامه‌نویسی آرایه‌های دو بعدی را با استفاده از روش سطري ذخیره می‌کند. آنگاه آدرس $SCORE[12, 3]$ یعنی نمودار آزمون سوم دانشجوی دوازدهم، به صورت زیر محاسبه می‌شود:

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

مالحظه می‌شود که در محاسبه آدرس تنها از رابطه (۴-۵) استفاده شده است.

بديهی است که آرایه‌های چند بعدی تفاوت بين نمايشهای منطقی و فيزيكی داده‌ها را بهتر بيان می‌کنند. شکل ۴-۸ چگونگي نمايش منطقی آرایه ماتریسی A 4×3 را به صورت يك آرایه مستطييلی از داده‌ها که در آن $A[J, K]$ در سطر J و ستون K ظاهر می‌شود نشان می‌دهد. از طرف ديگر، داده‌ها به صورت فيزيكی به وسیله يك مجموعة خطی از خانه‌های حافظه یا به صورت يك بعدی در حافظه کامپیوتر ذخیره می‌شوند. اين وضعیت در سراسر کتاب مورد توجه ماست. مثلاً برخی از ساختمان داده‌ها نظیر درختها یا گرافها را می‌توان به صورت منطقی در نظر گرفت در حالی که از نظر فيزيكی، به صورت خطی در حافظه کامپیوتر ذخیره می‌شوند.

حالت کلی آرایه‌های چند بعدی

آرایه‌های چند بعدی در حالت کلی مانند آرایه دو بعدی تعریف می‌شود. بعویظه این که، يك آرایه n بعدی $m_1 \times m_2 \times \dots \times m_n$ بنام B مجموعه‌ای از $m_1 \cdot m_2 \cdot \dots \cdot m_n$ عنصر داده‌ای است که در آن هر عنصر به وسیله لیستی از n عدد صحیح نظیر k_1, k_2, \dots, k_n مشخص می‌شود که اندیس نام دارند و دارای این خاصیت است که

$$1 \leq K_1 \leq m_1, \quad 1 \leq K_2 \leq m_2, \dots, \quad 1 \leq K_n \leq m_n$$

عنصر B با اندیسه‌های k_1, k_2, \dots, k_n به صورت زیر نمايش داده می‌شود:

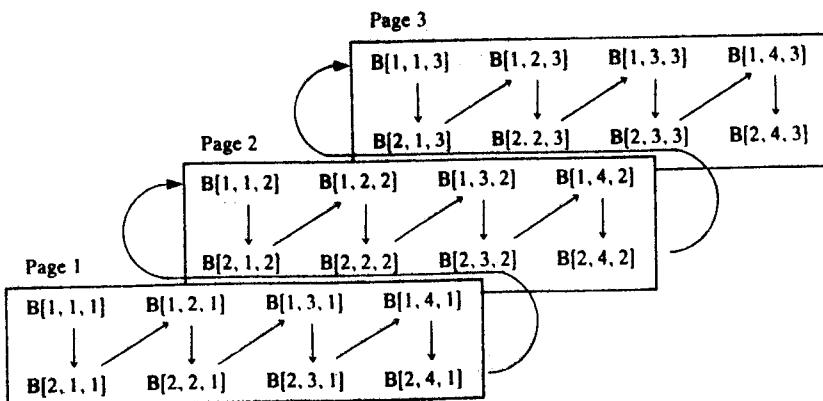
$$B[K_1, K_2, \dots, K_n] \quad \text{یا} \quad B_{K_1, K_2, \dots, K_n}$$

آرایه در حافظه، در دنباله‌ای از خانه‌های حافظه پشت سرهم ذخیره می‌شوند. به خصوص اینکه،

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

مثال ۴-۱۳

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



شکل ۴-۱۱

دو روش ذخیره B در آرایه یک بعدی شکل ۴-۱۲ نشان داده شده است. ملاحظه می‌کنید که پیکانها در شکل ۴-۱۱ روش ستونی قرار گرفتن عناصر را نمایش می‌دهند.

B	Subscripts	B	Subscripts
	(1, 1, 1)		(1, 1, 1)
	(1, 1, 2)		(2, 1, 1)
	(1, 1, 3)		(1, 2, 1)
	(1, 2, 1)		(2, 2, 1)
	(1, 2, 2)		(1, 3, 1)
:	:	:	:
	(2, 4, 2)		(1, 4, 3)
	(2, 4, 3)		(2, 4, 3)

(ب) روش سطري

(الف) روش ستوني

شکل ۴-۱۲

در حالت کلی تعریف آرایه‌های چندبعدی اجازه می‌دهد که کرانهای پائین مخالف ۱ باشد. فرض کنید C چنین آرایه n بعدی‌ای باشد. همانند قبل، مجموعه اندیس برای هر بعد C شامل اعداد صحیح متوالی از کران پائین تا کران بالای بعد است. طول i آرایه C برابر تعداد عناصر مجموعه اندیس‌ها است همچنین L_i را می‌توان مانند قبل از فرمول

$$L_i = \text{upper bound} - \text{lower bound} + 1 \quad (4-6)$$

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

$$E_i = K_i - \text{lower bound} \quad (4-7)$$

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

$$\text{Base}(C) + w[((...((E_{N-1}L_{N-1} + E_{N-2})L_{N-2}) + \dots + E_3)L_2 + E_2)L_1 + E_1] \quad (4-8)$$

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

$$\text{Base}(C) + w[((E_1L_2 + E_2)L_3 + E_3)L_4 + \dots + E_{N-1})L_N + E_N] \quad (4-9)$$

به دست آورد. یکبار دیگر مذکور می‌شویم که $\text{Base}(C)$ آدرس عنصر اول C و W تعداد کلمات در خانه حافظه یا طول کلمه را نشان می‌دهد.

مثال ۴-۱۴

فرض کنید MAZE یک آرایه سه بعدی باشد که به صورت زیر تعریف می‌شود:

$$\text{MAZE}(2:8, -4:1, 6:10)$$

در آن صورت طول سه بعد MAZE به ترتیب عبارتند از:

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

بنابراین MAZE دارای $L_1, L_2, L_3 = 7 \cdot 6 \cdot 5 = 210$ عنصر است.

فرض کنید زبان برنامه‌نویسی، MAZE را به روش سطروی در حافظه ذخیره می‌کند، همچنین فرض کنید $200 = W = 4$ کلمه در خانه حافظه وجود داشته باشد. آدرس یک عنصر MAZE به عنوان مثال MAZE[5, -1, 8] به صورت زیر محاسبه می‌شود. اندیشهای مؤثر به ترتیب عبارتند از:

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

با استفاده از فرمول (۹-۴) برای روش سطروی داریم:

$$\begin{aligned} E_1 L_2 &= 3 \cdot 6 = 18 \\ E_1 L_2 + E_2 &= 18 + 3 = 21 \\ (E_1 L_2 + E_2) L_3 &= 21 \cdot 5 = 105 \\ (E_1 L_2 + E_2) L_3 + E_3 &= 105 + 2 = 107 \end{aligned}$$

بنابراین:

$$\text{LOC(MAZE}[5, -1, 8]\text{)} = 200 + 4(107) = 200 + 428 = 628$$

۴-۱۰ اشاره‌گرها، آرایه‌های نوع اشاره‌گر

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

سازمانی را در نظر بگیرید که لیست اعضایش را به چهار گروه تقسیم می‌کند که در آن هر گروه شامل یک لیست الفبایی از اعضایی است که در یک منطقه معین زندگی می‌کنند. شکل ۴-۱۳ یک چنین لیستی را نشان می‌دهد.

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

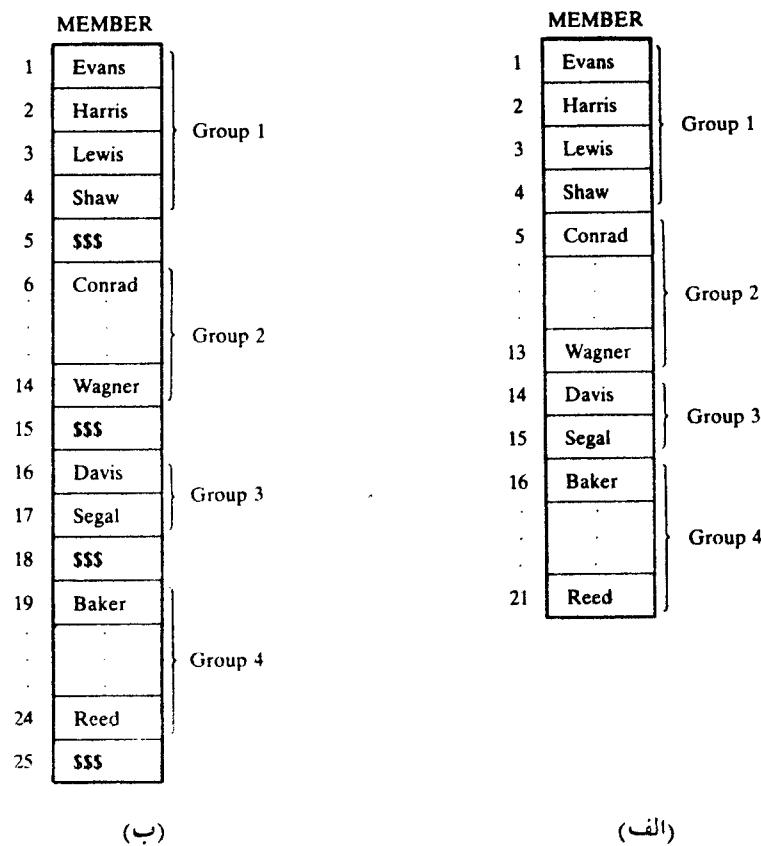
شکل ۱۳ - ۴

ملاحظه می‌کنید که در لیست بالا 21 نفر وجود دارد و هر گروه بترتیب شامل 4، 9، 2 و 6 نفر است. فرض کنید لیست اعضا با حفظ گروههای مختلف در حافظه ذخیره می‌شود. یک راه برای رسیدن به این منظور استفاده از یک آرایه $n \times 4$ ، دو بعدی است که هر سطر شامل یک گروه است یا یک آرایه $n \times 4$ ، دو بعدی است که هر ستون شامل یک گروه است. هرچند این ساختمان داده اجازه دسترسی به تک تک گروهها را می‌دهد ولی هنگامی که اندازه گروه خیلی بزرگ می‌شود بیهوده حافظه زیادی مصرف خواهد شد. بهویژه اینکه داده‌های شکل ۱۳ - ۴ احتیاج به دستکم یک آرایه 9×4 یا 4×9 عنصری برای ذخیره 21 نام خواهد داشت که تقریباً دو برابر حافظه موردنیاز است. شکل ۱۴ - ۴ نمایش آرایه 9×4 را نشان می‌دهد. در این نمودار ستاره‌ها عناصر داده‌ای و صفرها خانه‌های حافظه بلااستفاده را نشان می‌دهند. آرایه‌هایی که سطرهایا یا ستون‌هایش با تعداد مختلفی از عناصر داده‌ای شروع می‌شود و با خانه‌های حافظه مصرف نشده پایان می‌یابد آرایه‌های پله‌ای یا دندانه‌دار نام دارند.

*	*	*	*	0	0	0	0	0
*	*	*	*	*	*	*	*	*
*	*	0	0	0	0	0	0	0
*	*	*	*	*	*	0	0	0

شکل ۱۴ - ۴ آرایه‌های دندانه‌دار (پله‌ای)

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



شکل ۱۵-۴

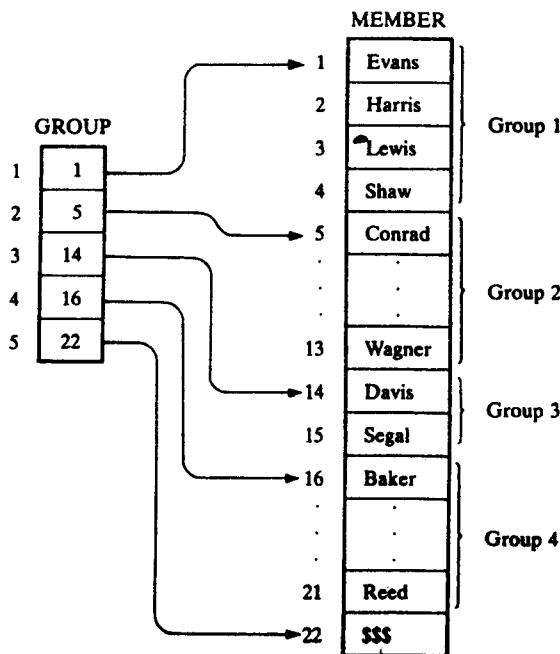
واضح است از نظر حافظه این روش کارا است. همچنین تمام لیست قابل پردازش است به عنوان مثال براحتی می‌توان تمام نام‌های داخل لیست را چاپ کرد. از طرف دیگر، هیچ راهی برای دسترسی به یک گروه خاص وجود ندارد مثلاً هیچ راهی برای پیدا کردن و چاپ نامهای داخل گروه سوم به تنها برای وجود ندارد.

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

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

آرایه‌های نوع اشاره‌گر

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



شکل ۱۶-۴

ملحوظه می‌کنید که $GROUP[L + 1] - 1$ به ترتیب شامل عنصرهای اول و آخر

گروه L است. دیده می‌شود که GROUP[5] به نگهبان لیست اشاره می‌کند و ۱ - GROUP[L] مکان عنصر آخر گروه ۴ را به دست می‌دهد.

مثال ۴-۱۵

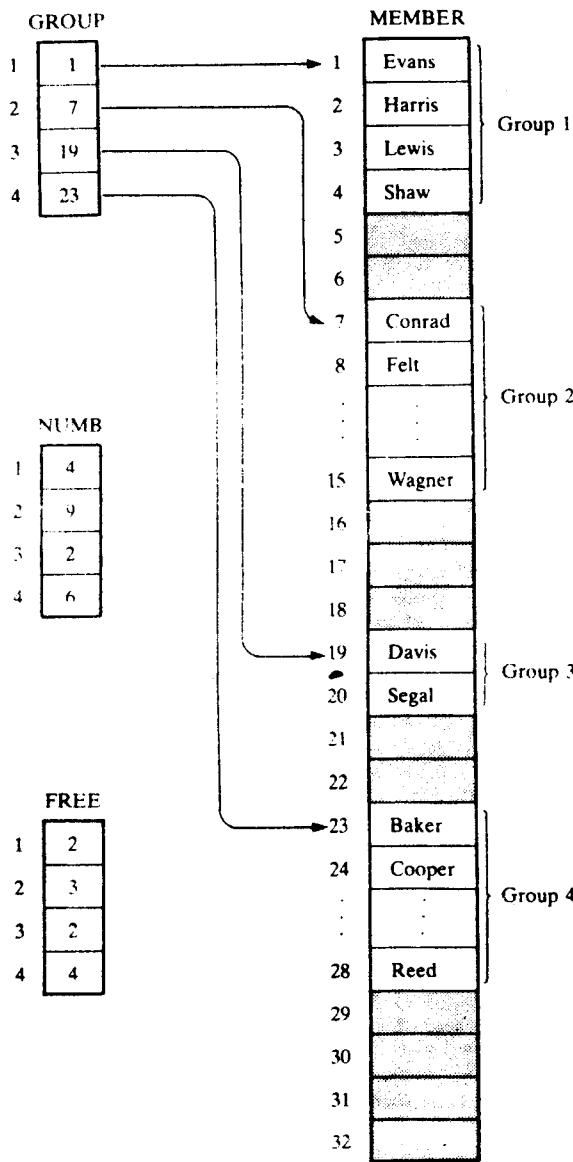
فرض کنید بخواهیم تنها نامهای گروه L ام شکل ۴-۱۶ را چاپ کنیم که مقدار L بخشی از ورودی است. از آنجاکه [L + ۱] - ۱ - GROUP[L] به ترتیب شامل مکانهای نام اول و آخر گروه L است. قطعه برنامه زیر این کار را انجام می‌دهد:

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

سادگی این قطعه برنامه از این واقعیت ناشی می‌شود که آرایه نوع اشاره‌گر GROUP گروه L ام را مشخص می‌کند. متغیرهای FIRST و LAST اساساً برای سهولت در نمادگذاری بکار گرفته شده‌اند. ساختمان داده شکل ۴-۱۶ با یک تغییر کوچک در شکل ۴-۱۷ نشان داده شده است که در آن خانه‌های حافظه بلااستفاده با سایه مشخص شده است. ملاحظه می‌کنید که اکنون بین گروه‌ها چند خانه خالی وجود دارد. بنابراین یک عنصر جدید می‌تواند در یک گروه اضافه شود بدون آن که احتیاج به جابجایی عناصر گروه دیگری باشد. با استفاده از این ساختمان داده به یک آرایه NUMB احتیاج است که تعداد عناصر هر گروه را به دست می‌دهد. ملاحظه می‌شود که GROUP[K + ۱] - GROUP[K] تعداد کل حافظه موجود برای گروه K است. بنابراین

$$\text{FREE}[K] = \text{GROUP}[K + 1] - \text{GROUP}[K] - \text{NUMB}[K]$$

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



شکل ۱۷ - ۴

مثال ۱۶ - ۴

مجددأً فرض کنید بخواهیم تنها نامهای گروه ۱ را چاپ کنیم که در آن باخشی از ورودی است اما اکنون به صورت شکل ۱۷ - ۴ ذخیره می‌شوند. ملاحظه می‌کنید که

GROUP[L] + NUMB[L] - 1 , **GROUP[L]**

به ترتیب شامل مکان نامهای اول و آخر گروه A است. بنابراین قطعه برنامه زیر، این عمل را نجات می‌دهد.

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

متغیرهای FIRST و LAST اساساً برای سهولت در نمادگذاری یکار رفته‌اند.

۱۱-۴. رکوردها؛ ساختارهای رکوردی

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

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

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

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

تحت رابطه تقسيم فيلد به زير فيلد ها، فيلد های يک رکورد تشکیل ساختار سلسله مرانجی را می دهند که می تواند به وسیله شماره های "سطح" به صورتی که درمثال $17 - 4 - 18 - 4$ داده شده است بیان گردد.

BROWN, JOHN M.	M	041684 M'D Y	BROWN, ROBERT S.	26	BROWN, SUSAN B.	22
			Name	Age	Name	Age
			Father		Mother	
			Sex	Birthday		

شکل ۱۸-۴

مثال ۴-۱۷

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

```

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

ساختار رکورد بالا معمولاً به صورت فوق بیان می‌شود. توجه دارید که نام Name در این رکورد سه بار و سن Age دو بار ظاهر شده است: عدد سمت چپ هر شناسه شماره سطح نامیده می‌شود. ملاحظه می‌کنید که پس از هر فیلد مرکب، زیرفیلدهای آن نوشته شده است و سطح زیرفیلدها، واحد بیشتر از سطح فیلدهای مرکب است. علاوه بر این یک فیلد در فیلد مرکب واقع است اگر و فقط اگر بلافاصله با شماره سطح بزرگتر پس از فیلد مرکب باشد.

بعضی از شناسه‌ها در یک ساختار رکوردي می‌توانند به آرایه‌هایی از عناصر مرتبط باشند. در واقع فرض کنید به جای خط اول ساختار بالا

1 Newborn(20)

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

Newborn₁, Newborn₂, newborn₃, ...

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

یا

که رکوردهای مختلف یک فایل را نشان می‌دهد.

مثال ۴-۱۸

رکوردهای دانشجویی یک کلاس به صورت زیر سازماندهی شده است:

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

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

مشخص کردن فیلدهای یک رکورد

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

مثال ۴-۱۹

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

Father.Age

یا بطور خلاصه

Newborn. Father. Age

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

(ب) فرض کنید جای خط اول در ساختار رکوردي مثال ۱۰-۴ را با عبارت زیر عوض کردہ‌ایم:

1 Newborn(20)

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

Sex[6] Newborn.Sex[6]
يا بطور خلاصه

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

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

(ج) ساختار رکوردي دانشجویان را در مثال ۱۸-۴ درنظر بگیرید. از آنجاکه دانشجو Student یک فایل 20 عنصری است تمام فیلدها به طور اتوماتیک آرایه‌های 20 عنصری می‌شوند. علاوه براین Test یک آرایه دو بعدی می‌شود. به خصوص این که، به دوم دانشجوی ششم به صورت زیر دسترسی پیدا می‌کنیم:

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

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

Test[3, 1]

به آزمون سوم دانشجوی اول دسترسی پیدا نمی‌کنیم بلکه به آزمون اول دانشجوی سوم دسترسی داریم. توجه کنید: در بعضی از کتاب‌ها برای بیان وضعیت شناسه‌های رکوردها، به جای نماد نقطه گذاری بین فیلدها، از نماد تابعی زیر استفاده می‌شود. برای مثال

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

می‌نویستند.

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

۱۲-۴ نمایش رکوردها در حافظه، آرایه‌های موازی

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

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

مثال ۴-۲۰

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

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

ملحوظه می‌کنید که متغیرهای جنس SEX و سال YEAR منحصر بفرد هستند، از این‌رو برای مراجعه به آنها احتیاج به بیان وضعیت نیست. از طرف دیگر AGE منحصر بفرد نیست. بتایراین از نماد MOTHER.AGE یا FATHER.AGE استفاده می‌کند.

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

مثال ۴-۲۱

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

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

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

شکل ۴-۱۹

مثال ۴-۲۲

بار دیگر رکورد **Newborn** مثال ۴-۱۷ را در نظر بگیرید. می‌توان فایلی مت Shank از این گونه رکوردها در نه آرایه خطی به صورت زیر ذخیره کرد:

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

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

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

رکودهایی با طول متغیر

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

زیر باشد:

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

در اینجا XXX میان فوت شدن پدر است یا این که با دانش آموز زندگی نمی‌کند یا دانش آموز برادر یا خواهری در مدرسه ندارد.

رکوردهای بالا مثالی از یک رکورد با طول متغیر است، چون عنصر داده‌ای دیگر افراد خانواده می‌تواند شامل هیچ یا چند اسم باشد. یک راه ذخیره فایل در آرایه‌ها در شکل ۴-۲۰ به تصویر درآمده است که در آن

	NAME	PHONE	FATHER	MOTHER	NUMB	PTR	SIBLING
1	Adams, John	345-6677	Richard	Mary	3	5	1
2	Bailey, Susan	222-1234	Steven	Sheila	0	0	2
3	Clark, Bruce	567-3344	XXXX	Barbara	2	2	3
							4
							5
							6
							7
							8

شکل ۴-۲۰

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

۴-۱۳ ماتریسها

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

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

$$V = (V_1, V_2, \dots, V_n)$$

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

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

در مباحث مربوط به بردارها و ماتریسها برای اشاره به تک تک اعداد از اصطلاح اسکالار استفاده می‌شود.

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

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

در بخش بعد تعدادی از عملیات جبری مربوط به بردارها و ماتریسها مورد بررسی قرار خواهد گرفت. آنگاه بدنبال این بخش راههای کارا و موثر ذخیره انواع معین از ماتریسها، موسوم به ماتریس‌های شکل یا خلوات Sparse بررسی خواهند شد.

جبر ماتریسها

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

بهتر است ضرب ماتریس را تاخته با تعریف ضرب اسکالار دو بردار شرح دهیم.

فرض کنید U و V دو بردار n عنصری باشند. آنگاه حاصلضرب اسکالار U و V که به صورت $U \cdot V$ نوشته می‌شود عددی اسکالار است که از ضرب عناصرهای U در عناصرهای متناظرش در V و آنگاه جمع آنها به دست می‌آید:

$$U \cdot V = U_1 V_1 + U_2 V_2 + \cdots + U_n V_n = \sum_{k=1}^n U_k V_k$$

تاکید می‌کنیم که $U \cdot V$ یک عدد یا یک اسکالار است نه یک بردار.

اکنون فرض کنید A یک ماتریس $P \times M$ و B یک ماتریس $M \times n$ باشد. حاصلضرب A و B زا به صورت AB می‌نویسیم که یک ماتریس $m \times n$ بنام C است که عنصر i -ام آن به صورت زیر به دست می‌آید:

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

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

مثال ۲۳ - ۴

(الف) فرض کنید

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

آنگاه

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

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

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

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

$$U \cdot W = 1 \cdot 3 + (-3) \cdot (-5) + 4 \cdot 2 + 5 \cdot (-1) = 3 + 15 + 8 - 5 = 21$$

(ج) فرض کنید

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

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

$$\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} \begin{pmatrix} 2 & 0 & -4 \\ 3 & 2 & 6 \end{pmatrix} = \begin{pmatrix} 1 \cdot 2 + 3 \cdot 3 & 1 \cdot 0 + 3 \cdot 2 & 1 \cdot (-4) + 3 \cdot 6 \\ 2 \cdot 2 + 4 \cdot 3 & 2 \cdot 0 + 4 \cdot 2 & 2 \cdot (-4) + 4 \cdot 6 \end{pmatrix} = \begin{pmatrix} 11 & 6 & 14 \\ 10 & 8 & 16 \end{pmatrix}$$

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

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

به عبارت دیگر :

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

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

Algorithm 4.7: (Matrix Multiplication) MATMUL(A , B , C , M , P , N)

Let A be an $M \times P$ matrix array, and let B be a $P \times N$ matrix array. This algorithm stores the product of A and B in an $M \times N$ matrix array C .

1. Repeat Steps 2 to 4 for $I = 1$ to M :
2. Repeat Steps 3 and 4 for $J = 1$ to N :
 3. Set $C[I, J] := 0$. [Initializes $C[I, J]$.]
 4. Repeat for $K = 1$ to P :

$$C[I, J] := C[I, J] + A[I, K] * B[K, J]$$

[End of inner loop.]

[End of Step 2 middle loop.]

[End of Step 1 outer loop.]
5. Exit.

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

$$C = m \cdot n \cdot p$$

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

مثال ۴-۲۴

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

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

در الگوریتم ۴.۷ حاصلضرب ماتریسی AB با استفاده از $8 = 2 \cdot 2 \cdot 2 = C$ عمل ضرب بدست می‌آید. از طرف دیگر AB را می‌توان به صورت زیر به دست آورد که در آن تنها از ۷ عمل ضرب استفاده می‌شود.

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

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

بعضی از نسخه‌های زبان برنامه‌نویسی **BASIC** دارای عملیات ماتریسی تعبیه شده در زبان هستند. به ویژه اینکه، در زیر چند دستور مجاز زبان **BASIC** ارائه شده است که در آن **A** و **B** آرایه‌های دو بعدی هستند که دارای ابعاد مناسب با هم می‌باشند و **K** یک اسکالر است:

$$\text{MAT } C = A + B$$

$$\text{MAT } D = (K) * A$$

$$\text{MAT } E = A * B$$

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

۴-۴ ماتریسهای خلوت

ماتریسهایی که دارای تعداد نسبتاً زیادی درایه صفر باشد ماتریس خلوت یا ماتریس **leck** نام دارد. دو نمونه کلی از ماتریسهای خلوت n مربعی که در اکثر کاربردها ظاهر می‌شوند در شکل ۲۱-۴ نشان داده شده است. گاهی اوقات رسم بر این است که بلاک‌های دارای صفر این‌گونه ماتریسهای را مانند شکل ۲۱-۴ حذف می‌کنند.

$$\left(\begin{array}{cccccc} 5 & -3 & & & & \\ 1 & 4 & 3 & & & \\ & 9 & -3 & 6 & & \\ & 2 & 4 & -7 & & \\ & 3 & -1 & 0 & & \\ & 6 & -5 & 8 & & \\ & 3 & -1 & & & \end{array} \right) \quad \left(\begin{array}{ccccc} 4 & & & & \\ 3 & -5 & & & \\ 1 & 0 & 6 & & \\ -7 & 8 & -1 & 3 & \\ 5 & -2 & 0 & 2 & -8 \end{array} \right)$$

(ب) ماتریس سه قطری

(الف) ماتریس مثلثی

شکل ۴-۲۱

اگر A اول لکه 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 بلا فاصله بالا و پائین قطر ظاهر شوند یک ماتریس سه قطری نام دارد.

ممکن است روش طبیعی نمایش ماتریسها در حافظه به صورت آرایه‌های دو بعدی، برای ماتریسها خلوت مناسب نباشد. یعنی، تنها با ذخیره درایه‌هایی که غیرصفر هستند می‌توان در مصرف حافظه صرفه جویی کرد. برای ماتریسها مثلثی این عمل در مثال زیر نشان داده شده است. غالتهای دیگر در قسمت مسئله‌های حل شده توضیح داده خواهد شد.

مثال ۴-۲۵

فرض کنید خواسته باشیم آرایه مثلثی A شکل ۴-۲۲ را در حافظه کامپیوتر ذخیره کنیم. واضح است که ذخیره درایه‌های بالای قطر اصلی A کاری بیهوده است چون می‌دانیم تمام این عناصر صفر هستند. از این رو تنها درایه‌های دیگر A را که در شکل ۴-۲۲ با پیکان مشخص شده است در آرایه خطی B ذخیره می‌کنیم، یعنی قوار می‌دهیم:

$$B[1] = a_{11} \quad B[2] = a_{21}, \quad B[3] = a_{22}, \quad B[4] = a_{31}, \quad \dots$$

نخست ملاحظه می‌کنید که B شامل

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

عنصر است که تقریباً نصف عنصر یک آرایه $n \times n$ دو بعدی است. از آنجا که در برنامه‌های خود به مقدار

احتیاج داریم، از این‌رو فرمولی را به دست می‌آوریم که عدد صحیح L را بر حسب J ، K معین می‌کند که در آن

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

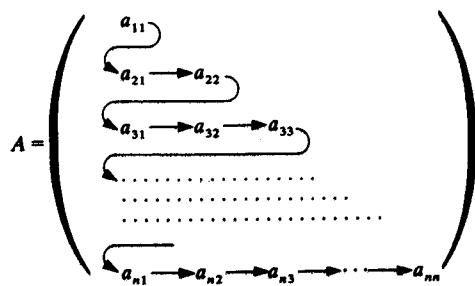
ملاحظه می‌شود که L تعداد عناصر داخل لیست حداکثر تا a_{JK} و خود آن را نمایش می‌دهد. اکنون تعداد

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

عنصر در سطرهای بالای a_{JK} وجود دارد و K عنصر در سطر زوجود 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 آرایه خطی B دسترسی پیدا کرد.



شکل ۴-۲۲

مسئله‌های حل شده

آرایه‌های خطی

مسئله ۱ - ۴ : آرایه‌های خطی (4:50) ، (5:50) و (18:50) را در نظر 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 $AAA[55]$ و $AAA[35]$ را پیدا کنید.

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

$$\text{Length} = UB - LB + 1$$

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 $\text{Length(CCC)} = \text{UB}$ چون 1

(ب) از فرمول زیر استفاده می‌کنیم :

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

از این رو

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

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

AAA عنصر AAA نیست چون ۵۵ بزرگتر از $50 = \text{UB}$ است.

مسئله ۲-۴: فرض کنید یک شرکت یک آرایه 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 L – 1984 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.

مسئله ۳-۴: فرض کنید A یک آرایه 10 عنصری با مقادیر a_1, a_2, \dots, a_{10} باشد. مقادیر A را پس از هر حلقه تعیین کنید.

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

Repeat for K = 9 to 1 by -1:
 Set A[K + 1] := A[9].
 [End of loop.] (ب)

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

(الف) نخست $A[1] := A[2]$ که $a_1 = a_2$ می‌شود که a_1 مقدار $A[1]$ است.
آنگاه $A[2] := A[3]$ که $a_1 = a_3$ می‌شود که a_1 مقدار فعلی $A[2]$ است.
آنگاه $A[3] := A[4]$ که $a_1 = a_4$ می‌شود که a_1 مقدار فعلی $A[3]$ است.
بدین ترتیب هر عنصر آرایه A دارای مقدار a_1 است که همان مقدار اولیه $A[1]$ است.

(ب) نخست $A[9] := A[10]$ که $a_9 = A[10]$ 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 خانه بعدی منتقل می‌شود. در پایان حلقه همچنان داریم $A[1] = a_1$.
توجه کنید: این مثال توضیحی بر این واقعیت است که در الگوریتم اضافه کردن یعنی الگوریتم ۴-۴ درست مانند حلقه (ب) در بالا، عنصرها به ترتیب عکس به طرف پائین منتقل می‌شوند.

مسئله ۴-۴: آرایه خطی NAME شکل ۲-۲۳ را که به صورت الفبایی مرتب شده درنظر بگیرید.

(الف) هرگاه بخواهیم Peters, Johnson, Brown 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 اضافه کردن را در دفترچه راهنمای تلفن چگونه انجام می‌دهد؟

حل: (الف) اضافه کردن Brown مستلزم جابجا شدن 13 عنصر است و عمل اضافه کردن Johnson مستلزم جابجا شدن 1 عنصر و عمل اضافه کردن Peters مستلزم جابجا یعنی 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 تازه‌می‌کند.

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 ۲۳-۴

جستجوکردن، مرتب کردن

مسئله ۵-۴: آرایه خطی مرتب شده الفبایی NAME شکل ۲۳-۴ را در نظر بگیرید.

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

(ب) تعیین کنید برای چنین آرایه 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) $C(Hobbs) = 6$ با تمام نامها مقایسه می‌شود این مقایسه از نام Allen شروع شده تا این که Hobbs در NAME[6] پیدا می‌شود.

$$C(Morgan) = 10 \text{ چون Morgan در NAME[10] ظاهر می‌شود.}$$

$C(Fisher) = 15$ 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 این رو جستجو ناموفق است.

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

مسئله ۶-۴: فرض کنید الگوریتم جستجوی دودویی، الگوریتم ۴.۶، روی آرایه NAME در شکل ۴-۲۳ به کار بسته شده است تا مکان Goodman پیدا شود. BEG و END انتهايی و MID و سط را برای قطعه آزمایشي در هر مرحله از الگوریتم پیدا کنيد.

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

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

مرحله ۲ : چون Goodman < Irwin قرار دهد $MID = 3[Dickens]$. از این رو $MID = 3[Dickens]$.

مرحله ۳ : چون Goodman > Dickens قرار دهد $MID = 5[Goodman]$. از این رو $MID = 5[Goodman]$.

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

مسئله ۷-۴: الگوریتم جستجوی دودویی، الگوریتم ۴.۶ را به گونه‌ای اصلاح کنید تا تبدیل به الگوریتم جستجو و اضافه کردن شود.

حل : در چهار مرحله اول الگوریتم به هیچ تغییری نیاز نیست. تنها وقتی ITEM در DATA ظاهر نشود الگوریتم کنترل کار را به مرحله ۵ 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.

مسئله ۴-۸: فرض کنید A یک آرایه مرتب شده 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 $(n)g$ را برای پیدا کردن 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 $1 + 2 + 4 + \dots + 64 = 127 = 200 - 73$ باقی می‌ماند. زمان اجرای بدترین حالت $f(n) = 8$ است. زمان اجرای حالت میانگین به صورت زیر محاسبه

می‌شود:

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

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

مسئله ۹-۴: با استفاده از الگوریتم مرتب کردن حبابی، الگوریتم 4.4، C تعداد مقایسه‌ها و D تعداد جابجایی‌ها را پیدا کنید که $n = 6$ حرف کلمه PEOPLE را به صورت حرفی مرتب می‌کند.

حل: جفت حرفهایی که در هر یک از $5 - 1 = 4$ گذر به ترتیب با هم مقایسه می‌شوند به صورت زیر است: جفت حرفهای مورد مقایسه و جابجا شونده با مرتع مشخص شده‌اند و جفت حرفهایی که با هم مقایسه ولی جابجا نمی‌شوند با دائیره مشخص شده است.

Pass 1.	P E O P L E,	E [] O P L E,	E O P P L E
	E O P P [] E,	E O P L [] 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 ، 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.
 مسئله ۱۰-۴: اتحاد زیر را که در تجزیه و تحلیل انواع الگوریتم‌های مرتب‌سازی و جستجو مورد استفاده قرار می‌گیرد ثابت کنید.

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

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

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

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

مجموع دو مقدار S جمع بالا به صورت زیر است:

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

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

آرایه‌های چندبعدی، ماتریسها

مسئله ۱۱-۴: فرض کنید آرایه‌های چندبعدی A و B با استفاده از نمادهای $B(1:8, -5:5, -10:5)$ و $A(-2:2, 2:22)$

تعریف شده‌اند.

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

(ب) عنصر [3, 3] در B را در نظر بگیرید. اندیشهای مؤثر E₁, E₂, E₃ و آدرس این عنصر را با فرض

$W = 400$ و $Base(B) = 4$ کلمه در خانه 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 این رو طول L_i ابعاد آرایه A عبارت است از

$$L_2 = 22 - 2 + 1 = 21 \quad L_1 = 2 - (-2) + 1 = 5$$

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

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

بنابراین B دارای $8.11.16 = 1408$ عنصر است.

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

درنتیجه

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

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

$$\begin{aligned} E_3L_2 &= 13 \cdot 11 = 143 & E_3L_2 + E_2 &= 143 + 8 = 151 \\ (E_3L_2 + E_2)L_1 &= 151 \cdot 8 = 1208 & (E_3L_2 + E_2)L_1 + E_1 &= 1208 + 2 = 1210 \end{aligned}$$

بنابراین

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

مسئله ۱۲ - ۴ : فرض کنید A یک آرایه ماتریسی مربعی $n \times n$ باشد. قطعه برنامه‌ای بنویسید که

(الف) NUM تعداد عناصر غیر صفر 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 ($a_{11}, a_{22}, \dots, a_{nn}$) را پیدا کنید.

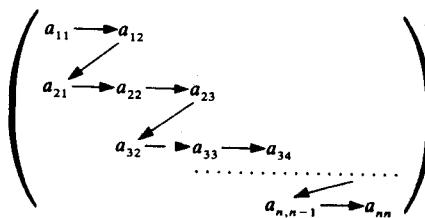
حل : (الف)

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.

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.
- (ب)

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.
- (ج)

مسئله ۱۳ - ۴: فرض کنید A یک آرایه سه قطری $n \times n$ مربعی به صورتی باشد که در شکل ۲۴ - ۴ نشان داده شده است.



شکل ۲۴ - ۴ آرایه سه قطری

توجه دارید که روی قطر A_{1-n} عنصر و در بالا و پائین قطر $1 - n$ عنصر می‌باشد. از این رو A_{1-2} - $3n$ عنصر غیرصفر دارد. فرض کنید خواسته باشیم A را در یک آرایه خطی B به صورتی که بوسیله پیکانها در شکل ۲۴ - ۴ مشخص شده است ذخیره کنیم یعنی فرمولی پیدا کنید که L را بر حسب J و K به

صورتی به دست دهد که $B[L] = A[J, K]$

$$B[1] = a_{11}, \quad B[2] = a_{12}, \quad B[3] = a_{21}, \quad B[4] = a_{22}, \quad \dots \quad \text{یعنی}$$

طوری که بتوان به مقدار $A[J, K]$ از طریق آرایه B دسترسی پیدا کرد.

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

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

مسئله ۱۴ - ۴: یک آرایه ماتریسی $n \times n$ مربعی A را مقارن گویند اگر بهزاری 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 ذخیره 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 ذخیره A و B در حافظه بیان کنید.

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

(ب) چون $[J, K] = A[J, K]$ ، تنها ذخیره 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 ۴-۲۵ 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 بالای قطر را ذخیره کنیم. بنابر این A و B را می‌توان در یک آرایه $n \times (n+1)$ به صورتی که در شکل ۴-۲۵ نشان داده شده است ذخیره کرد که در آن $C[J, K] = A[J, K]$ و قسمی $J \leq K \leq n$ اما

$$C[J, K] = B[J, K - 1]$$

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

شکل ۴-۲۵

آرایه‌های نوع اشاره‌گر، ساختارهای رکوردي

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

(الف) در اینجا آرایه مرتب شده الفبایی CLIENT و آرایه LAWYER به گونه‌ای وجود دارد که $LAWYER[K]$ وکیل مربوط به $[K]$ است.

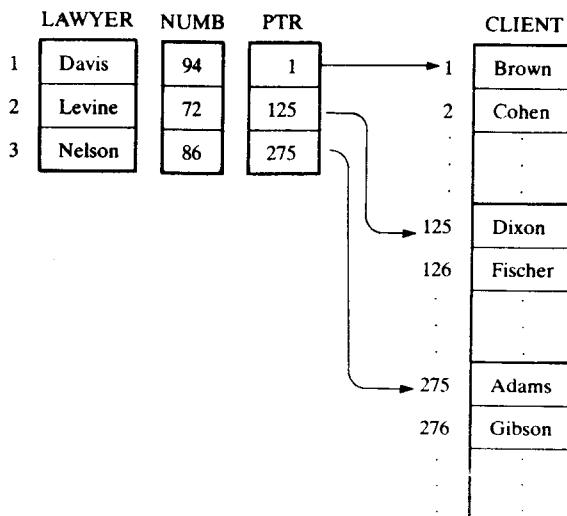
(ب) در اینجا سه آرایه مجزا NELSON و LEVINE, DAVIS وجود دارد که هر آرایه شامل لیست موکلین است.

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

	DAVIS	LEVINE	NELSON	CLIENT	LAWYER
1.	Brown	1. Dixon	1. Adams	1. Adams	Nelson
2	Cohen	2. Fischer	2. Gibson	2. Brown	Davis
3	Eisen	.	3. Harris	3. Cohen	Davis
.	.	.	.	4. Dixon	Levine
.	.	.	.	5. Eisen	Davis
.	.	.	.	6. Fischer	Levine
.	.	.	.	7. Gibson	Nelson
.	.	.	.	8. Harris	Nelson
.
.

(ب)

(الف)



(ج)

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 وجود 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 ۴-۲۶ (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 سیاهه موکلین (دفتر ثبت مشخصات موکلین نزد منشی) مختلف است. آنگاه شکل ۴-۲۶ (ب) احتمالاً مفیدترین ساختمان داده خواهد بود.

فرض کنید که دارالوکاله، کلیه امور موکلین را به طور متناوب رسیدگی کرده و هر یک از وکلا موظف است که در این برهمه زمانی به پرونده موکلین خود رسیدگی کند. آنگاه شکل ۴-۲۶ (ج) احتمالاً مفیدترین ساختمان داده خواهد بود.

مسئله ۴-۱۶: در زیر لیست فیلد های 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 فیلد ها، فیلد های ابتدایی هستند.

حل: (الف) هر چند فیلد ها به صورت خطی ارائه شده اند با این وجود شماره سطح ها رابطه سلسله مراتبی بین فیلد ها را تشان می دهند. 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: **Verbal** و **Number**, **last**, **First**, **MI**, **Sex**, **Day**, **Month**, **Year**, **Math** ابتدایی است فقط اگر بعد از آن فیلد با شماره سطح بالاتر وجود نداشته باشد.

مسئله ۱۷ - ۴: یک استاد دانشگاه مشخصات داده‌ای هر دانشجو در یک کلاس ۲۰ نفره را به صورت زیر نگه می‌دارد.

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

در اینجا **Grade** یک درایه ۲ کاراکتری مثلاً **B+** یا **C** یا **-A** است. یک ساختار در زبان **I / PL** برای ذخیره داده‌ها بیان کنید.

حل: یک عنصر در یک ساختار رکوردهای خود می‌تواند یک آرایه باشد. به جای ذخیره جداگانه 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);
```

مسئله ۱۸ - 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]** معروف مدل نمرات سال **K** ام و **CUM** معروف مدل کل است.

(الف) در فایل چند فیلد ابتدایی وجود دارد؟

(ب) چگونه می‌توان به (i) رشته **major** دانشجوی هجدهم و (ii) مدل نمرات **GPA** ای 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 ۲۲۰۰ فیلد مقدماتی وجود دارد.

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

(ج) (i) در اینجا Name[15] معرف نام دانشجوی پانزدهم است. اما NAME یک فیلد مرکب یا چند قسمتی است. از این‌رو First[15] ، LAST[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 آرایه GPA دومنی دانشجو است. از این‌رو

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

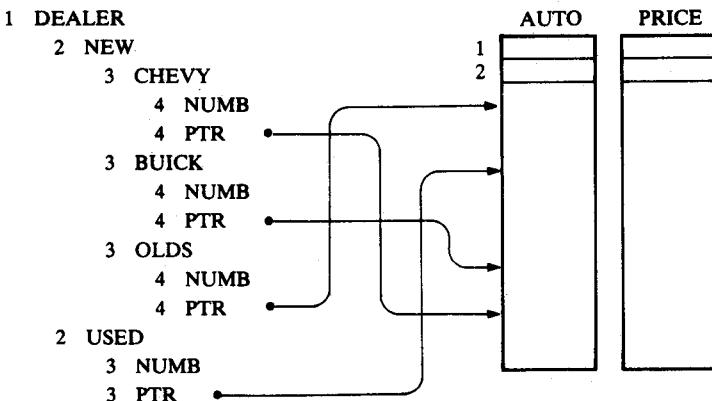
چاپ می‌شوند.

(iv) GPA[1, 3] یک فیلد یعنی GPA سال سوم اولین دانشجو است. یعنی تنها ۳ GPA[1, 3] چاپ می‌شود.

مسئله ۱۹ - 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 داده شکل ۴-۲۷ نیز استفاده می‌کند که ترکیبی از یک ساختار رکوردهای با متغیرهای اشاره‌گر است. Chevy نو و 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 زیربرنامه procedure برای چاپ شماره‌های سریال تمام Buick های نو که کمتر از \$10000 قیمت دارند بنویسید.



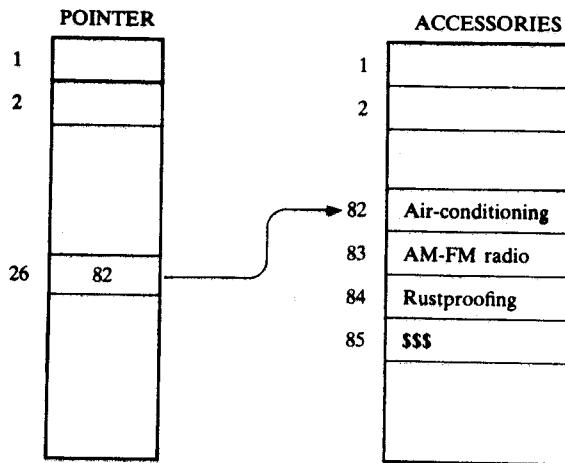
شکل ۴-۲۷

حل : (الف) چون PTR بیش از یکبار در ساختار رکوردی ظاهر می شود از این رو باید از BUICK.PTR برای مراجعه به مکان لیست Buick نو در AUTO استفاده کنید.

(ب) باید لیست را traverse پیماش کنید اما تنها آن دسته از Buick ها را چاپ کنید که قیمت آن کمتر از \$10000 است. زیر برنامه 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 ۴-۲۸

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

حل : این کار را می‌توان همانند شکل ۴-۲۸ انجام داد. یعنی علاوه بر AUTO و PRICE یک آرایه ACCESSORIES وجود دارد به گونه‌ای که [K] مکان لیست لوازم همراه در آرایه [K] مکان لیست AUTO در [K] در برابر با نگهبان (\$\$) باشد.

مسئله‌های تكمیلی

آرایه‌ها

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

مسئله ۴-۲۲ : 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 $W = \text{Base}(Y) = 400$ و $4 = Y$ کلمه در خانه حافظه وجود داشته باشد. اندیشهای مؤثر E_1, E_2, E_3 و آدرس $[5, 10, 15]$ را پیدا کنید. با این فرض که (i) Y با روش سطحی و (ii) Y با روش ستونی ذخیره شده باشد.

مسئله ۲۳-۴: آرایه 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.

مسئله ۲۴-۴: فرض کنید A یک آرایه خطی با n مقدار عددی باشد. یک زیربرنامه **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, \dots, x_n با توجه

به تعریف، به صورت زیر محاسبه می‌شود:

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

مسئله ۲۵-۴: هر دانشجوی یک کلاس 30 نفره در 6 آزمون شرکت می‌کند که در آن دامنه نمرات بین 0 و 100 است. فرض کنید نمرات آزمونها در یک آرایه 30×6 بنام **TEST** ذخیره شده است، یک قطعه برنامه بنویسید که:

(الف) معدل نمرات هر آزمون را پیدا کند.

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

(ج) تعداد **NUM** دانشجویانی که نمره قبولی نگرفته‌اند یعنی دانشجویانی که نمره نهایی شان کمتر از 60 است را پیدا کنید.

(د) میانگین نمرات نهایی را پیدا کنید.

آرایه‌های نوع اشاره‌گر، ساختارهای رکوردي

مسئله ۲۶-۴: داده‌های شکل ۲۶-۴ (ج) را در نظر بگیرید. یک زیربرنامه **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 به گونه‌ای که **FREE[K]** شامل تعداد حافظه‌های خالی بعد از لیست موکلین مربوط به **LAWYER[K]** باشد.

مسئله ۲۷-۴: در زیرلیست فیلد رکوردهای یک فایل از کارمندان با شماره 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 رارسم کنید.

(ب) کدامیک از فیلد ها، فیلد های ابتدایی هستند؟

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

مسئله ۲۸ - ۴ : ساختمان داده شکل ۲۷ - ۴ را درنظر بگیرید. یک زیربرنامه **Procedure** برای انجام هر یک از عملیات زیر بنویسید :

(الف) تعداد **Oldsmobiles** نو را که با قیمت کمتر از \$10 000 به فروش می رستند پیدا کند.

(ب) تعداد **automobiles** نو را که با قیمت کمتر از \$10 000 به فروش می رستند پیدا کند.

(ج) تعداد **automobiles** را که با قیمت کمتر از \$10 000 به فروش می رستند پیدا کند.

(د) تمام **automobiles** را که زیر \$10 000 به فروش می رستند پیدا کند.

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

مسئله ۲۹ - 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 **I** / **PL** یا یک رکورد **PASCAL** برای ذخیره داده ها 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)

مسئله ۳۰ - ۴ : ساختمان داده مسئله ۱۸ - ۴ را درنظر بگیرید. یک زیربرنامه **Procedure** بنویسید که :

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

(ب) تعداد دانشجویان رشته زیست‌شناسی را پیدا کند.

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

برای مسئله‌های زیر برنامه بنویسید

آرایه‌ها

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

مسئله ۳۱-۴: 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.

مسئله ۳۲-۴: الگوریتم جستجوی خطی را به صورت یک زیر برنامه 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.

مسئله ۳۳-۴: الگوریتم جستجوی دودویی و اضافه کردن را به صورت یک زیر برنامه 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 را باید به ARRAY اضافه شود.

مسئله ۳۴-۴: یک برنامه بنویسید که شماره 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) ۰۷۳۲ - ۵۸ - ۱۷۴ (ب) ۵۵۵۴ - ۵۵ - ۱۷۲ و (ج) ۱۲۶ - ۶۳ - ۶۳۸۲ آزمایش کنید.

مسئله ۳۵-۴: 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) Bailey و (ج) Johnson آزمایش کنید.

مسئله ۳۶-۴: 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 ۴-۱

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
186-58-0490	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

مسئله ۳۷ - 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.

مسئله ۳۸ - ۴ : برای هر یک از حالت‌های زیر یک برنامه بنویسید:
 (الف) با استفاده از آرایه SSN آرایه‌های NUMBER و PTR را به گونه‌ای تعریف کنید که NUMBER آرایه مرتب شده‌ای از عناصر SSN باشد و PTR شامل مکان NUMBER[K] در SSN باشد.

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

آرایه‌های نوع اشاره‌گر

فرض کنید داده‌های داخل جدول ۴-۲ در یک آرایه خطی CLASS (با فضایی برای ۵۰ نام) ذخیره شده‌اند. علاوه بر این فرض کنید بین بخش Section ها، ۲ فضای خالی وجود دارد و آرایه خطی NUMB و FREE طوری تعریف شده‌اند که NUMB[K] حاوی تعداد عناصر در بخش K است و

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

جدول ۴-۲

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		

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

مسئله ۴-۴۰ : 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 (ج) Lopez آزمایش کنید.

مسئله ۴-۴۱ : یک برنامه بنویسید که نام‌ها را به صورت ستونی، به صورتی که در جدول ۴-۲ داده شده است چاپ کند.

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

مسئله ۴-۴۳ : 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.

مسئله‌های گوناگون

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

مسئله ۴۵-۴: الگوریتم ضرب دو ماتریس، یعنی الگوریتم ۴.۷ را به صورت یک زیربرنامه

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

بنویسید که C حاصلضرب ماتریس $m \times p$ ، A و B ماتریس $n \times m$ را پیدا کند. برنامه را با استفاده از آزمایش کنید.

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

مسئله ۴۶-۴: چند جمله‌ای

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

را در نظر بگیرید. محاسبه 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 محاسبه مقدار یک چندجمله‌ای روش هورنر نام دارد.

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

(ب) فرض کنید ضریبهای یک چندجمله‌ای در یک آرایه خطی $(1, N + 1, A(N))$ در حافظه هستند، یعنی [1] ضرب x^n ، x^{n-1} و \dots x^1 و $[N + 1, A(N)]$ مقدار ثابت چندجمله‌ای است. یک زیربرنامه **HORNER(A, N + 1, X, Y)** بنویسید که مقدار چندجمله‌ای $Y = F(X)$ را به ازای 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

۱- مقدمه

استفاده از اصطلاح "لیست" در زندگی روزمره به یک مجموعه خطی از اقلام داده‌ای، مربوط می‌شود. شکل ۱-۵ (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 ۱-۵ (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 ذخیره چنین داده‌هایی است که در فصل ۴ مورد بحث و بررسی کامل قرار گرفت. یادآوری می‌کنیم که رابطه خطی بین عناصر داده‌ای یک آرایه به وسیله رابطه فیزیکی داده‌ها در حافظه منعکس می‌شود نه به وسیله اطلاعات خود 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 حافظه اضافی به راحتی نمی‌توان اندازه یک آرایه را دو برابر یا سه برابر کرد. به همین خاطر به آرایه‌ها، لیستهای فشرده یا متراکم نیز می‌گویند. علاوه بر این به آرایه‌ها، ساختمند داده ایستا نیز گفته می‌شود.

milk	milk
eggs	eggs
butter	butter
tomatoes	tomatoes
apples	apples
oranges	oranges
bread	bread
chicken	
corn	
lettuce	

(ب)

(الف)

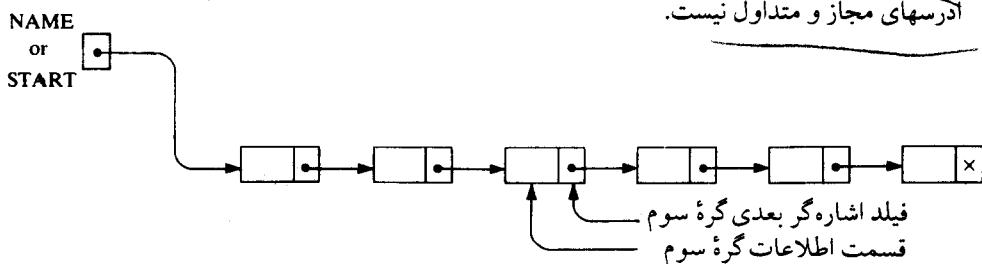
شکل ۵-۱

راه دیگر ذخیره 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 اشاره گر نامیده می‌شود قرار می‌دهیم. بدین ترتیب 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 علاوه‌الای 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 اشاره گر ذخیره کنیم. ساختمان داده 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.

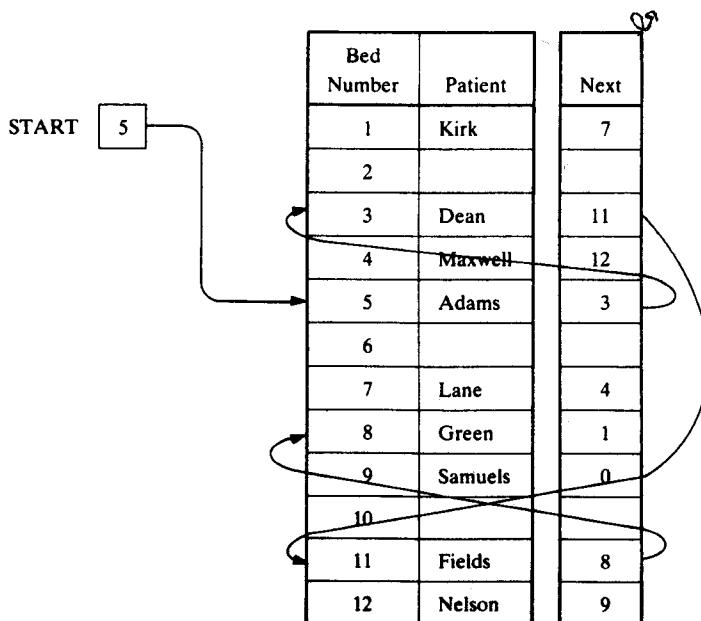
۵-۲ لیستهای پیوندی

یک لیست پیوندی یا یک لیست یکطرفه مجموعه‌ای خطی از عناصر داده‌ای بنام گره‌ها است که در آن ترتیب خطی توسط اشاره گرها 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 اشاره گر بعدی نام دارد شامل آدرس گره بعدی در لیست است.

شکل ۲-۵ یک نمودار از یک لیست پیوندی با ۶ 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 رکوردهای مانند (NAME, ADDRESS,...) باشد. قسمت راست، فیلد اشاره‌گر بعدی گره را نمایش می‌دهد و یک پیکان از آن تا گره 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 پوج یا NULL نام دارد که هیچیک از آدرسهای مجاز و متداول نیست.



شکل ۲-۵ لیست پیوندی با ۶ گره



شکل ۲-۳

در عمل ۰ یا یک عدد منفی برای اشاره گر پوچ یا **NULL** مورد استفاده قرار می‌گیرد. اشاره گر پوچ که در نمودار با X نمایش داده شده است علامت پایان لیست است. علاوه بر این لیست پیوندی دارای یک متغیر اشاره گر لیست بنام **NAME** یا **START** است که محتوای آن آدرس گره اول لیست است. از این رو یک پیکان از **START** به گره 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 اشاره گر **NULL** در متغیر **START** نمایش داده می‌شود.

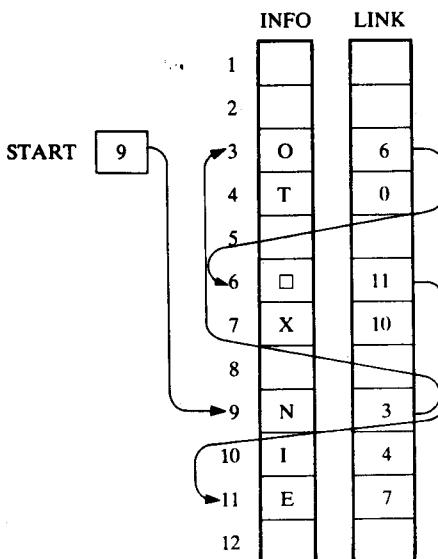
مثال ۵ -۱

اطاق یک بیمارستان دارای ۱۲ تخت است که از میان آنها ۹ تخت به صورتی که در شکل ۳-۵ 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 اشاره گر که در نمودار **Next** نام‌گذاری شده است داده شود. از متغیر **START** برای اشاره به بیمار اول استفاده می‌کنیم. از این رو **START** شامل ۵ است چون بیمار اول **Adams** تخت شماره ۵ را اشغال کرده است. اشاره گر **Adams** برابر ۳ است چون **Dean**، بیمار بعدی تخت شماره ۳ را اشغال کرده است. اشاره گر **Dean** برابر ۱۱ است چون **Fields** بیمار بعدی تخت شماره ۱۱ را اشغال کرده است و الی آخر. ورودی مربوط به بیمار آخر (**Samuels**) شامل اشاره گر پوچ است که آن را با **NULL** نمایش می‌دهد. در نمودار فقط برای بیان لیست چند بیمار اول پیکان رسم شده است.

۳-۵ 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 آرایه خطی است که ما آنها را در اینجا **INFO** و **LINK[K]** می‌نامیم. نظری **INFO[K]** و **LINK[K]** که به ترتیب شامل قسمت اطلاعات و فیلد اشاره گر بعدی یک گره از **LIST** است. همانگونه که در بالا مذکور شدیم **LIST** نیز نیازمند یک نام متغیر نظری **START** است که شامل مکان ابتدای لیست است و نگهبان اشاره گر بعد که با **NULL** نمایش داده می‌شود می‌بین انتهای لیست است. از آنجا که اندیشهای آرایه‌های **INFO** و **LINK** معمولاً مثبت هستند ما **0 = NULL** 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 اشاره گر مربوط به خود را دارا باشد که مکان گره اولش را به دست می‌دهد.



شکل ۵-۴

مثال ۵-۲

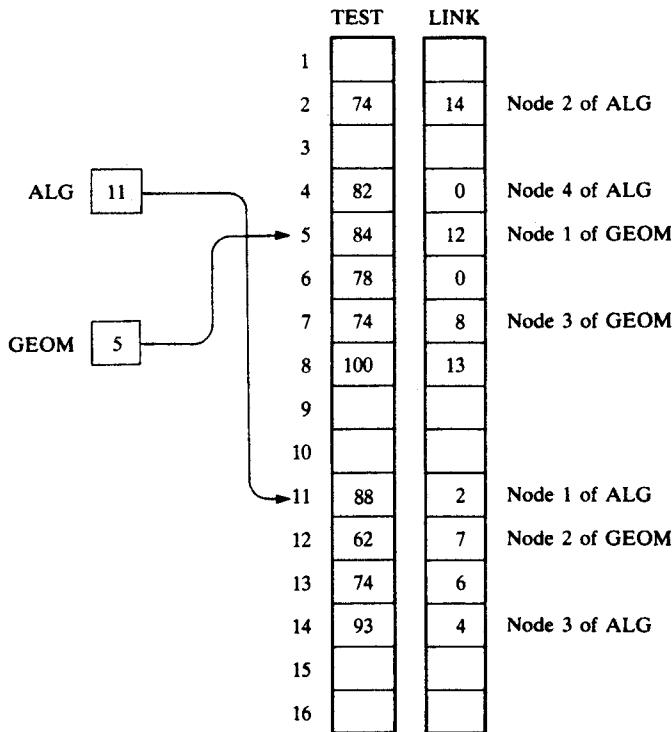
شکل ۵-۴ تصویر یک لیست پیوندی را در حافظه نشان می‌دهد که در آن هر گره لیست شامل یک کاراکتر است. می‌توانیم لیست واقعی کاراکترها، یا به بیان دیگر رشته را به صورت زیر به دست آوریم.
9 از این رو **INFO[9] = N** اولین کاراکتر است.
3 از این رو **INFO[3] = O** دومین کاراکتر است.
6 از این رو **INFO[6] = □** فضای خالی سومین کاراکتر است.
11 از این رو **INFO[11] = E** چهارمین کاراکتر است.
7 از این رو **INFO[7] = X** پنجمین کاراکتر است.
10 از این رو **INFO[10] = I** ششمین کاراکتر است.
4 از این رو **INFO[4] = T** هفتمین کاراکتر است.
0 **LINK[4] = 0** یعنی مقدار آن **NULL** است، از این رو لیست پایان یافته است.
به عبارت دیگر، **NO EXIT** رشته کاراکتری موردنظر است.

مثال ۵-۳

شکل ۵-۵ 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 ۱۱، مکان اولین گره آن و GEOM حاوی ۵، مکان اولین گره آن است. با تعقیب اشاره‌گرها ملاحظه می‌کنید که ALG شامل نمرات آزمون زیر است:

88, 74, 93, 82



شکل ۵-۵

و GEOM شامل نمرات آزمون

84, 62, 74, 100, 74, 78

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

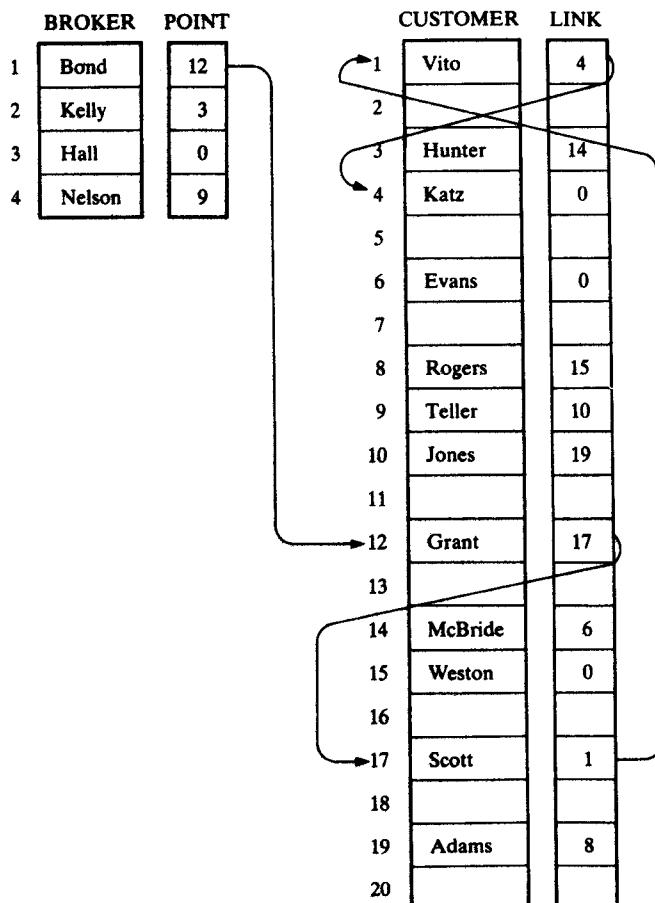
مثال ۵-۴

یک شرکت فروش کامپیوتر را در نظر 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 ۵-۵ سازماندهی کرد،

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

بنابراین لیست مشتری‌های Bond که با پیکان مشخص شده است شامل

Grant, Scott, Vito, Katz



شکل ۶-۵

Hunter, McBride, Evans

و لیست Nelson شامل

Teller, Jones, Adams, Rogers, Weston

و لیست Hall یک لیست خالی است، چون اشاره گر NULL یا ۰ در POINT[۳] ظاهر شده است. به بیان کلی، قسمت اطلاعات یک گره می‌تواند یک رکورد باشیش از یک عنصر داده‌ای باشد. در چنین مواردی، داده‌ها بایستی در نوعی از ساختار رکوردي یا در یک مجموعه از آرایه‌های موازی مانند آنچه که در مثال زیر بیان شده است ذخیره شوند.

مثال ۵-۵

فایل پرسنلی یک شرکت کوچک را در نظر بگیرید که دارای داده‌های زیر برای نمایش کارمند است:

Name, Social Security Number, Sex, Monthly Salary

معمولًاً چهار آرایه موازی مانند NAME, SSN, SEX, SALARY برای ذخیره داده‌ها به صورتی که در بخش ۱۲-۴ مورد بررسی قرار گرفت مورد نیاز است. شکل ۵-۷ چگونگی ذخیره داده‌ها را به صورت یک لیست پیوندی مرتب شده (الفبایی) نشان می‌دهد که در آن تنها از یک آرایه اضافی LINK برای فیلد اشاره گر بعدی لیست استفاده شده است و متغیر START به اولین رکورد لیست اشاره می‌کند. ملاحظه می‌کنید که ۰ به عنوان اشاره گر پوج NULL مورد استفاده قرار گرفته است.

	NAME	SSN	SEX	SALARY	LINK
1					
2	Davis	192-38-7282	Female	22 800	12
3	Kelly	165-64-3351	Male	19 000	7
4	Green	175-56-2251	Male	27 200	14
5					
6	Brown	178-52-1065	Female	14 700	9
7	Lewis	181-58-9939	Female	16 400	10
8					
9	Cohen	177-44-4557	Male	19 000	2
10	Rubin	135-46-6262	Female	15 500	0
11					
12	Evans	168-56-8113	Male	34 200	4
13					
14	Harris	208-56-1654	Female	22 800	3

۴-۵ پیمایش یک لیست پیوندی

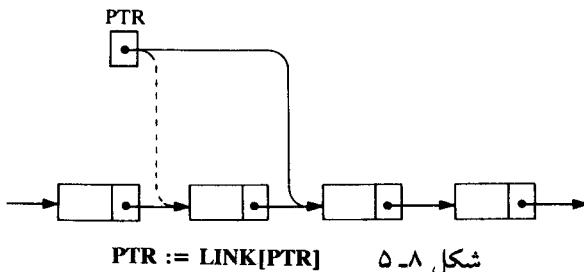
فرض کنید LIST یک لیست پیوندی در حافظه باشد و در آرایه‌های خطی INFO و LINK ذخیره شده است که START به عنصر اول اشاره می‌کند و NULL پایان لیست را نشان می‌دهد. فرض کنید خواسته باشیم برای پردازش هر گره دقیقاً یکبار LIST را پیمایش کنیم. این بخش الگوریتمی را ارائه می‌دهد که این عمل را انجام می‌دهد و آنگاه از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

الگوریتم پیمایش از متغیر اشاره گر PTR که به گره در حال پردازش اشاره دارد استفاده می‌کند. بنابراین

LINK[PTR] به گره بعدی اشاره می‌کند که قرار است پردازش شود. بنابراین دستور جایگزینی

PTR := LINK[PTR]

اشارة گر را به گره بعدی در لیست، به صورتی که در نمودار ۵-۸ نشان داده شده است، انتقال می‌دهد.



شکل ۵-۸

جزئیات این الگوریتم به صورت زیر است: به START یا PTR مقدار اولیه می‌دهیم. آنگاه **INFO[PTR]** یا اطلاعات گرۀ اول را پردازش می‌کنیم. PTR را با دستور جایگزینی **INFO[PTR] := PTR := LINK[PTR]** تازه می‌کنیم. از این‌رو PTR به گره دوم اشاره می‌کند. آنگاه **INFO[PTR]** یعنی اطلاعات گرۀ دوم را با دستور جایگزینی **PTR := LINK[PTR]** تازه می‌کنیم. مجدداً PTR را با دستور جایگزینی **INFO[PTR] := PTR := LINK[PTR]** تازه می‌کنیم. آنگاه **INFO[PTR]** یعنی اطلاعات گرۀ 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 شباخت بین الگوریتم ۵.۱ و الگوریتم ۴.۱ توجه کنید که یک آرایه خطی را پیمایش می‌کند.
شباخت 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.

مواظیب باشید : همانند آرایه‌های خطی، عمل **PROCESS** در الگوریتم ۵.۱ ممکن است از چند متغیر استفاده کند که باید قبل از اعمال **PROCESS** بر هر یک از عناصر داخل **LIST**، مقدار اولیه بگیرد.
درنتیجه، ممکن است الگوریتم با مرحله مقدار اولیه دادن دنبال شود.

مثال ۵-۶

زیربرنامه **PROCEDURE** زیر، اطلاعات هر گره لیست پیوندی را چاپ می‌کند. از آنجاکه زیربرنامه باید لیست را پیمایش کند، از این رو شباخت زیادی با الگوریتم ۵.۱ 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.]
5. [End of Step 2 loop.]
5. Return.

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

Write: INFO[PTR]

برای مرحله پردازش در الگوریتم ۵.۱ به دست آید.

مثال ۵-۷

زیربرنامه **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.]
6. [End of Step 3 loop.]
6. Return.

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

این رو، این زیربرنامه خیلی به الگوریتم پیمایش بالا یعنی الگوریتم ۵.۱ شبیه است. با وجود این در اینجا قبل از پیمایش لیست لازم است به متغیر **NUM** در یک مرحله مقدار اولیه داده شود. به بیان 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. مسئله ۳-۵ را ببینید.

۵-۵ جستجو در یک لیست پیوندی

فرض کنید یک لیست پیوندی **LIST** به صورت نشان داده شده در بخش‌های ۳-۵ و ۴-۵، در حافظه ذخیره شده است. فرض کنید عنصر مشخص **ITEM** داده شده است. این بخش دو الگوریتم جستجو برای تعیین مکان **LOC** گرمای را که 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** در واقع یک مقدار کلیدی باشد و بخواهیم در یک فایل رکوردهای را جستجوی کنیم که شامل **ITEM** است، آنگاه **ITEM** می‌تواند تنها یک بار در **LIST** ظاهر شود.

لیست **LIST** مرتب شده 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 مقایسه **ITEM** با محتوای **INFO[PTR]** هر گره لیست **LIST** یک گره به یک گره انجام می‌دهیم. قبل از آن که اشاره‌گر **PTR** را با دستور

PTR := LINK[PTR]

تاژه 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 منفی بودن جواب، باید تحقیق کنیم که آیا

INFO[PTR] = ITEM

یا خیر. این دو آزمایش را نمی‌توان در یک زمان انجام داد چون وقتی $\text{INFO}[\text{PTR}] = \text{NULL}$ باشد $\text{PTR} = \text{NULL}$ 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 \neq 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 ۷-۴ ببررسی شد. به بیان دیگر، زمان اجرای بدترین حالت، متناسب با n^2 تعداد عناصر لیست LIST است و زمان اجرای حالت میانگین تقریباً با $2n$ متناسب است. با این شرط که ITEM در لیست LIST یکبار ظاهر شده است که در هر گره لیست LIST احتمال مساوی دارد.

مثال ۵

فایل پرستنی شکل ۷-۵ را درنظر بگیرید. قطعه برنامه زیر شماره تأمین اجتماعی NNN یک کارمند را می‌خواند و آنگاه حقوق کارمند را ۵ درصد افزایش می‌دهد.

1. Read: NNN.
2. Call SEARCH(SSN, LINK, START, NNN, LOC).
3. If LOC \neq 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 شماره 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 را با پیمایش لیست با استفاده از متغیر اشاره‌گر PTR و مقایسه ITEM با محتوای INFO[PTR] هر گره 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^2 تعداد عناصر لیست LIST است و زمان اجرای حالت میانگین تقریباً با $2n$ متناسب است.

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

مثال ۵-۹

بار دیگر فایل پرسنلی شکل ۵-۷ را در نظر بگیرید. قطعه برنامه زیر نام EMP یک کارمند را می‌خواند و حقوق کارمند را ۵ درصد افزایش می‌دهد. این مثال را با مثال ۵-۸ 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 ۵.۳ را مورد استفاده قرار دهیم چون لیست به صورت الفبایی مرتب شده است.

۶-۵ تخصیص حافظه، جمع‌آوری حافظه بلااستفاده

از جمله موارد تعمیر و نگهداری لیستهای پیوندی در حافظه امکان اضافه کردن گره‌های جدید در لیست است و این امر مستلزم مکانیسمی است که حافظه بلااستفاده را به گره‌های جدید اختصاص دهد. به طور مشابه، به مکانیسمی موردنیاز است که به وسیله آن حافظه گره‌های حذف شده برای استفاده آتی در دسترس باشد. این مباحث در این بخش بررسی می‌شود، حال آن که بحث کلی اضافه کردن 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 می‌شود. خانه‌های حافظه بلااستفاده دارای اشاره‌گر مخصوص به خود است که لیست حافظه موجود یا لیست حافظه آزاد یا مخزن حافظه آزاد نامیده می‌شود.

فرض کنید لیستهای پیوندی به وسیله آرایه‌های موازی به صورتی که در بخش‌های قبل بیان شد پیاده‌سازی شده است و بخواهیم عملیات اضافه کردن 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 حافظه آزاد لیست **AVAIL** نیز می‌گویند. چنین ساختمان داده‌ای را اغلب به صورت

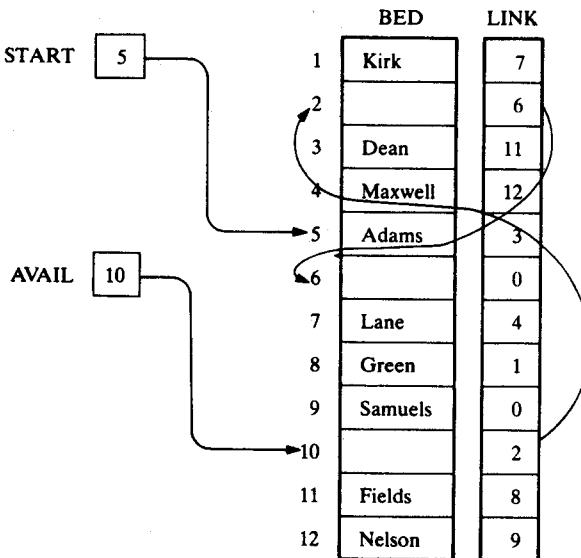
LIST(INFO, LINK, START, AVAIL)

نمایش می‌دهند.

مثال ۱۰-۵

فرض کنید لیست بیماران مثال ۱-۵ در آرایه‌های خطی **BED** و **LINK** ذخیره شده است طوری که بیمار تخت K ام در **BED[K]** جایگزین می‌شود. در آن صورت فضای موجود در آرایه خطی **BED** را می‌توان مانند شکل ۹-۵ 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]** اشاره گر پوج فیلد اشاره گر بعدی اش را دارد است یعنی $\text{LINK}[6] = 0$.



شکل ۵-۹

مثال ۵-۱۱

(الف) فضای در دسترس آرایه خطی **TEST** در شکل ۵-۵ را می‌توان مانند شکل ۵-۱۰ به هم پیوند داد. ملاحظه می‌کنید که هر یک از لیستهای **ALG** و **GEOM** می‌توانند از لیست **AVAIL** استفاده کنند. توجه دارید که ۹ از این رو **TEST[9]** اولین گره آزاد در لیست **AVAIL** است چون **TEST[10]**, **LINK[AVAIL] = LINK[9] = 10** است و الی آخر.

(ب) فایل پرسنلی در شکل ۵-۷ را در نظر بگیرید. فضای در دسترس در آرایه خطی **NAME** را می‌توان مانند شکل ۵-۱۱ به هم پیوند داد. ملاحظه می‌کنید که لیست حافظه آزاد در **NAME** از **NAME[1]** تشكیل شده است. علاوه بر این ملاحظه می‌کنید که مقدارها در **LINK** به صورت همزمان فضای حافظه آزاد را برای آرایه‌های خطی **SSN** و **SEX** لیست می‌کند.

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

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

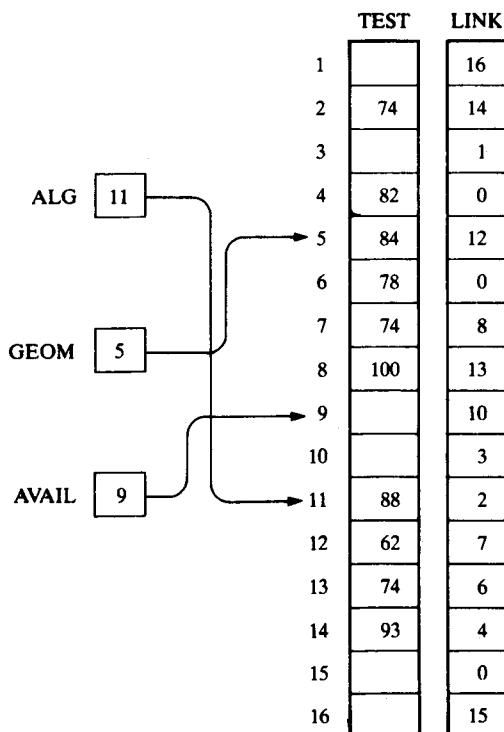
مثال ۵-۱۲

فرض کنید (LIST(INFO, LINK, START, AVAIL) برای $n = 10$ گره فضای حافظه دارد. علاوه براین

فرض کنید LIST از ابتدا خالی است، شکل ۵-۱۲ را به گونه‌ای نشان می‌دهد که لیست LINK از دنباله‌های AVAIL

$\text{INFO}[1], \text{INFO}[2], \dots, \text{INFO}[10]$

تشکیل شده است به عبارت دیگر لیست AVAIL از عناصر INFO با ترتیب معمول تشکیل شده است.
• $\text{START} = \text{NULL}$ ملاحظه می‌کنید که چون لیست خالی است



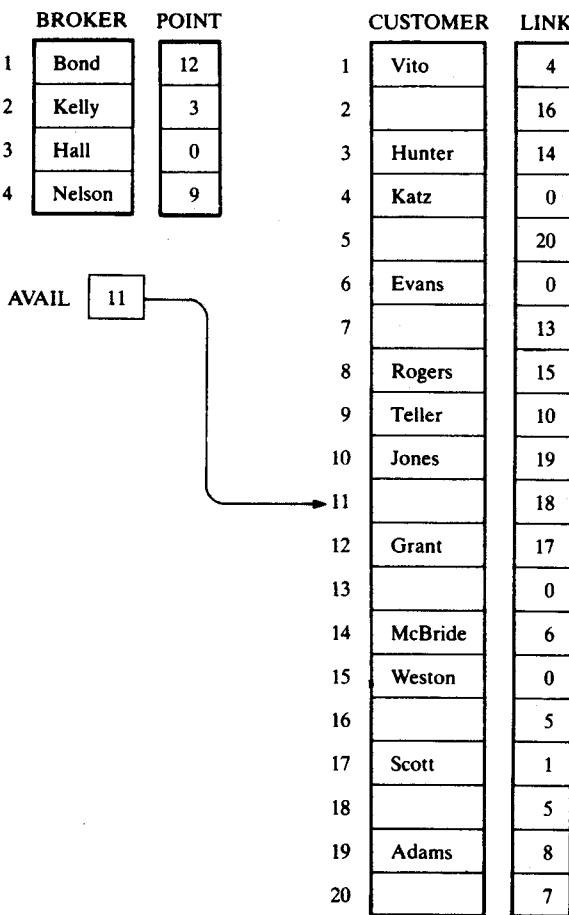
شکل ۵-۱۰

	NAME	SSN	SEX	SALARY	LINK
1					0
2	Davis	192-38-7282	Female	22 800	12
3	Kelly	165-64-3351	Male	19 000	7
4	Green	175-56-2251	Male	27 200	14
5					1
6	Brown	178-52-1065	Female	14 700	9
7	Lewis	181-58-9939	Female	16 400	10
8					11
9	Cohen	177-44-4557	Male	19 000	2
10	Rubin	135-46-6262	Female	15 500	0
11					13
12	Evans	168-56-8113	Male	34 200	4
13					5
14	Harris	208-56-1654	Female	22 800	3

شکل ۱۱-۵

	INFO	LINK
1		2
2		3
3		4
4		5
5		6
6		7
7		8
8		9
9		10
10		0

شکل ۱۲-۵



شکل ۵-۱۳

جمع آوری حافظه های بلا استفاده

فرض کنید بخاطر حذف یک گره از یک لیست یا حذف تمام لیست از یک برنامه می توان از فضای حافظه آن مجددًا استفاده کرد. واضح است که می خواهیم از فضای حافظه در کار بعدی استفاده کنیم. یک راه برای رسیدن به این منظور آن است که این فضا را بی درنگ در لیست حافظه آزاد اضافه کنیم. این همان کاری است که ما هنگام پیاده سازی لیستهای پیوندی به وسیله آرایه های خطی انجام می دهیم. با وجود این، این روش برای سیستم عامل یک کامپیوتر بسیار وقت گیر است که برای این منظور از روش دیگری نظری روش زیر استفاده می شود.

سیستم عامل یک کامپیوتر می‌تواند به طور متناوب تمام فضای حاصل از حذف یک یا چند گره یا تمام لیست را درون لیست حافظه آزاد جمع آوری کند. هر روشی که این جمع آوری را انجام می‌دهد جمع آوری حافظه بلااستفاده یا جمع آوری زائد های کامپیوترا نام دارد. جمع آوری حافظه بلااستفاده معمولاً در دو مرحله انجام می‌شود. نخست کامپیوتر تمام لیست را پیمایش می‌کند خانه‌هایی از حافظه را که در حال حاضر مورد استفاده می‌باشدند علامت می‌زند، آنگاه حافظه را پیمایش می‌کند و تمام فضای حافظه درون لیست حافظه آزاد را که برجسب نخورده‌اند علامت می‌زند. جمع آوری حافظه بلااستفاده می‌تواند زمانی انجام شود که تنها مقدار بسیار اندکی از حافظه آزاد موجود باشد یا هیچ حافظه‌ای در لیست حافظه آزاد باقی نمانده باشد یا وقتی که CPU بلا تکلیف(بیکار) است و وقت برای جمع آوری حافظه بلااستفاده در اختیار دارد. به بیان کلی‌تر، جمع آوری حافظه بلااستفاده از دید برنامه‌نویس پنهان است. بحث بیشتر درباره حافظه بلااستفاده خارج از حدود مطالب این کتاب است.

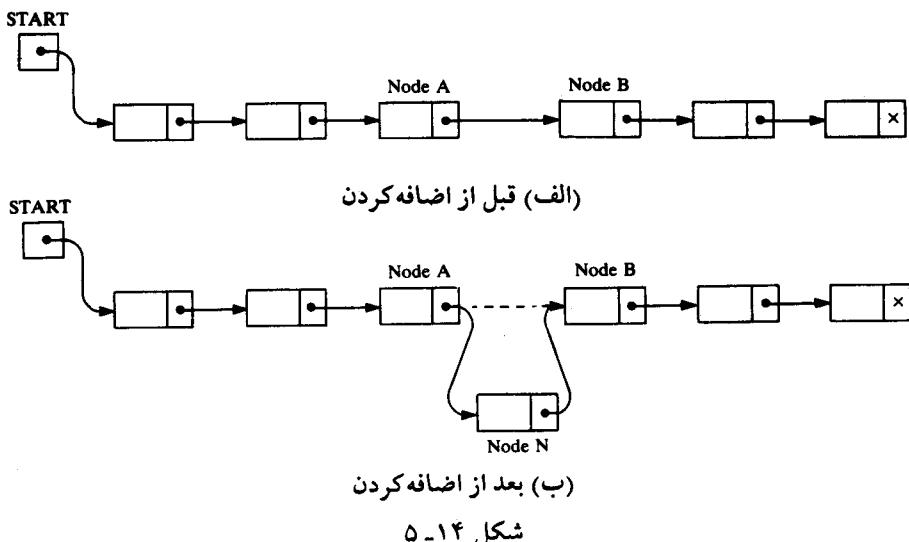
UnderFlow و زیرریزی OverFlow

گاهی اوقات نیاز است که اطلاعات جدید در یک ساختمان داده اضافه شود اما هیچ فضای آزاد در اختیار نداریم. یعنی لیست حافظه آزاد خالی است معمولاً به این وضعیت سرریزی می‌گویند. برنامه‌نویس می‌تواند با چاپ پیغام **OVERFLOW** با مسئله سرریزی برخورد کند. در چنین مواردی، برنامه‌نویس می‌تواند با افزودن حافظه به آرایه‌های مورد بررسی برنامه را اصلاح کند. ملاحظه می‌کنید که سرریزی در لیستهای پیوندی وقتی اتفاق می‌افتد که **NULL = AVAIL** و خواسته باشیم یک عنصر جدید به لیست اضافه کنیم.

به طور مشابه، اصطلاح زیرریزی مربوط به وضعیتی است که خواسته باشیم اطلاعاتی را از یک ساختمان داده خالی حذف کنیم. برنامه‌نویس می‌تواند با چاپ پیغام **START = NULL** مسئله زیرریزی را مورد توجه قرار دهد. ملاحظه می‌کنید که زیرریزی در لیستهای پیوندی وقتی اتفاق می‌افتد که **START = NULL** و خواسته باشیم یک عنصر را از لیست حذف کنیم.

۷-۵ اضافه کردن گره در یک لیست پیوندی

فرض کنید **LIST** یک لیست پیوندی باشد که دارای گره‌های متوالی **A** و **B** به صورت نشان داده شده در شکل ۱۴-۵ (الف) است. فرض کنید گره **N** قرار است به این لیست اضافه شود و بین گره‌های **A** و **B** قرار گیرد. نمودار اضافه کردن این گره در شکل ۱۴-۵ (ب) نشان داده شده است.

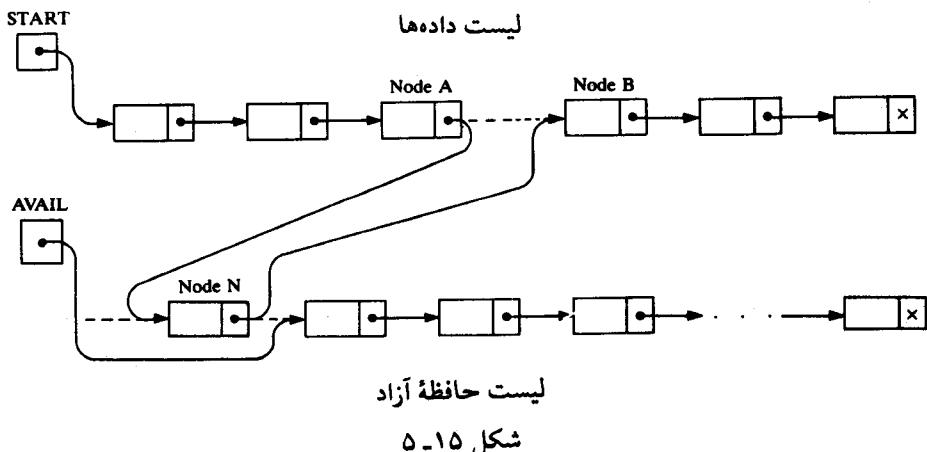


به بیان دیگر گره A اکنون به گره جدید N اشاره می‌کند و گره N به گره B اشاره می‌کند که A پیش از این به آن اشاره می‌کرد.

فرض کنید لیست پیوندی ما به صورت زیر در حافظه نگهداری می‌شود:

LIST(INFO, LINK, START, AVAIL)

شکل ۵-۱۴-۵ درنظر نمی‌گیرد که فضای حافظه برای گره جدید N از لیست AVAIL گرفته می‌شود. بخصوص این که، جهت سهولت در پردازش، گره اول لیست AVAIL برای گره جدید N بکار گرفته می‌شود. بدین ترتیب نمودار دقیقتر چنین اضافه کردنی در شکل ۵-۱۵ ارائه شده است.



ملاحظه می‌کنید که سه فیلد اشاره‌گر به صورت زیر تغییر می‌کنند:

- (۱) فیلد اشاره‌گر بعدی گره **A** اکنون به گره جدید **N** اشاره می‌کند که قبلًا **AVAIL** به آن اشاره می‌کرد.
- (۲) **AVAIL** اکنون به گره دوم در مخزن حافظه آزاد اشاره می‌کند که قبلًا **N** به آن اشاره می‌کرد.
- (۳) فیلد اشاره‌گر بعدی گره **N** اکنون به گره **B** اشاره می‌کند که قبلًا **A** به آن اشاره می‌کرد. دو حالت خاص نیز وجود دارد. اگر گره جدید **N** اولین گره لیست باشد، آنگاه **START** به **N** اشاره خواهد کرد و اگر گره جدید **N** آخرین گره لیست باشد، آنگاه **N** شامل اشاره‌گر پوج **NULL POINTER** خواهد بود.

مثال ۵-۱۳

(الف) شکل ۵-۹ یعنی، لیست الفبایی بیماران یک اتاق بیمارستان را در نظر بگیرید. فرض کنید **Hughes** بیمار پذیرفته شده در این اتاق باشد. ملاحظه می‌کنید که **Hughes** (i) روی تخت ۱۰ خوابانده می‌شود که اولین تخت آزاد است. **Hughes** (ii) در لیست بین **Green** و **Kirk** اضافه می‌شود. سه تغییر در فیلد های اشاره‌گر به صورت زیر است:

$$1 \quad -1 \quad \text{اکنون } \text{LINK}[8] = 10 \quad \text{اشاره می‌کند.}$$

$$2 \quad -2 \quad \text{اکنون } \text{LINK}[10] = 1 \quad \text{اشاره می‌کند.}$$

$$3 \quad -3 \quad \text{اکنون } \text{AVAIL} = 2 \quad \text{به تخت آزاد بعدی اشاره می‌کند.}$$

(ب) شکل ۵-۱۳ یعنی لیست مأموران فروش و مشتری های آنها را در نظر بگیرید. از آنجا که لیست های مشتری ها مرتب شده نیستند، فرض می‌کنیم که هر مشتری جدید به آغاز آن لیست اضافه می‌شود. فرض کنید **Gordan** یک مشتری جدید **Kelly** باشد. ملاحظه می‌کنید که **Gordan** (i) در **CUSTOMER[11]** **Gordan** (ii) قبل از **Hunter** اضافه می‌شود که اولین مشتری قبلی **Kelly** است.

سه تغییر در فیلد های اشاره‌گر به صورت زیر است:

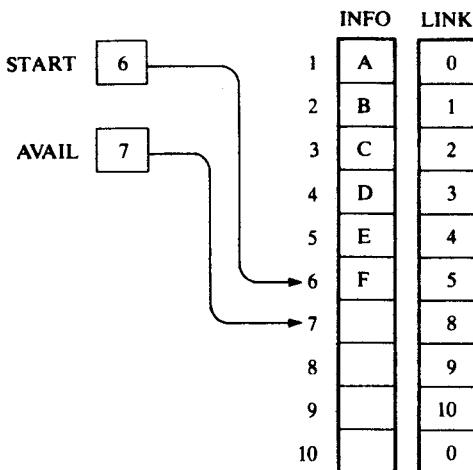
$$1 \quad -1 \quad \text{اکنون لیست به } \text{Gordan} \text{ شروع می‌شود.}$$

$$2 \quad -2 \quad \text{اکنون } \text{LINK}[11] = 3 \quad \text{اشاره می‌کند.}$$

$$3 \quad -3 \quad \text{اکنون } \text{AVAIL} = 18 \quad \text{به گره آزاد بعدی اشاره می‌کند.}$$

(ج) فرض کنید عناصر داده ای **E**, **F**, **A**, **B**, **C**, **D** در لیست خالی شکل ۵-۱۲ می‌باشند. یکی پس از دیگری اضافه می‌شوند. مجدداً فرض کنید هر گره جدید در ابتدای لیست اضافه می‌شود. بنابراین پس از شش اضافه کردن، **B** به **A**, **C** به **B**, **D** به **C**, **E** به **D**, **F** به **E** اشاره می‌کند همچنین **AVAIL** اولین گره در دسترس پس از شش حاوی اشاره‌گر پوج یا صفر است، علاوه بر این $7 = \text{AVAIL}$ اولین گره در دسترس پس از شش

اضافه کردن و $6 = \text{START}$ مکان اولین گره یعنی F است. شکل ۱۶-۵ لیست جدید (که در آن $n = 10$) را نشان می‌دهد.



شکل ۱۶-۵

الگوریتم‌های اضافه کردن گره

الگوریتم‌هایی که گره‌ها را به درون لیست پیوندی اضافه می‌کنند در وضعیت‌ها و شرایط متعدد می‌توانند مطرح شوند. ما در اینجا سه الگوریتم از این نوع را مورد بحث و بررسی قرار می‌دهیم. در الگوریتم اول یک گره به ابتدای لیست اضافه می‌شود، الگوریتم دوم گره را پس از گره‌ای که در مکان معینی قرار دارد اضافه می‌کند و الگوریتم سوم یک گره را به لیست مرتب شده اضافه می‌کند. در تمام این الگوریتم‌ها فرض می‌شود لیست پیوندی به صورت **(LIST(INFO, LINK, START, AVAIL)** در حافظه ذخیره شده است است و متغیر **ITEM** شامل اطلاعات جدیدی است که قرار است به لیست اضافه شود. از آنجاکه این الگوریتم‌ها از یک گره لیست **AVAIL** استفاده می‌کنند، از این‌رو تمام الگوریتم‌ها حاوی مراحل زیر است:

- (الف) تحقیق می‌کند که آیا فضای آزاد در لیست **AVAIL** وجود دارد یا خیر. اگر جواب منفی است یعنی **AVAIL = NULL** آنگاه الگوریتم پیغام **OVERFLOW** را چاپ می‌کند.
- (ب) گره اول را از لیست **AVAIL** حذف می‌کند. با استفاده از متغیر **NEW** مکان گره جدید را نگه می‌دارد.

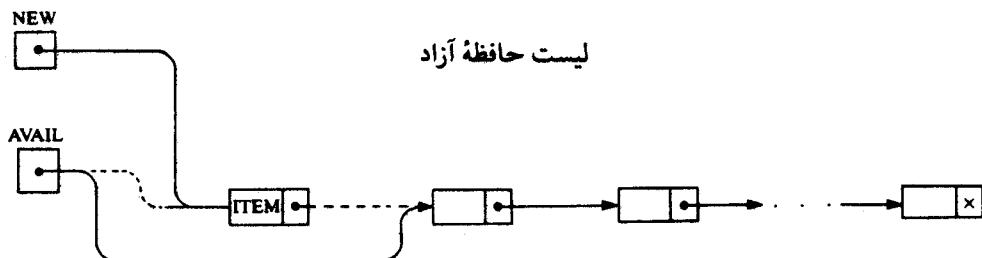
این مرحله را می‌توان با جفت دستور جایگزینی زیر پیاده‌سازی کرد:

NEW := AVAIL, AVAIL := LINK[AVAIL]

(ج) اطلاعات جدید را در گره جدید کپی می‌کند. به بیان دیگر:

INFO[NEW] := ITEM

در شکل ۵-۱۷ نمودار دو مرحله اخیر رسم شده است:



شکل ۵-۱۷

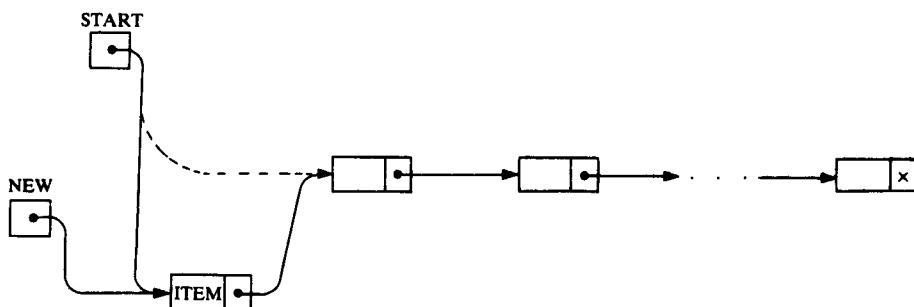
اضافه کردن یک گره به ابتدای لیست

فرض کنید لیست پیوندی ما الزاماً مرتب شده نیست و دلیلی وجود ندارد که یک گره جدید در محل خاصی از لیست اضافه شود. آنگاه بهترین محل برای اضافه کردن گره، ابتدای لیست است. الگوریتمی که این کار را انجام می‌دهد به شرح زیر است:

Algorithm 5.4: INSFIRST(INFO, LINK, START, AVAIL, ITEM)
This algorithm inserts ITEM as the first node in the list.

1. [OVERFLOW?] If AVAIL = NULL, then: Write: OVERFLOW, and Exit.
2. [Remove first node from AVAIL list.]
Set NEW := AVAIL and AVAIL := LINK[AVAIL].
3. Set INFO[NEW] := ITEM. [Copies new data into new node.]
4. Set LINK[NEW] := START. [New node now points to original first node.]
5. Set START := NEW. [Changes START so it points to the new node.]
6. Exit.

مرحله ۱ تا ۳ قبل ابررسی شده است و نمودار مرحله‌های ۲ و ۳ در شکل ۵-۱۷ رسم شده است.
نمودار مرحله‌های ۴ و ۵ در شکل ۵-۱۸ ارائه شده است.



شکل ۵-۱۸ اضافه کردن گره به ابتدای لیست

مثال ۵-۱۴

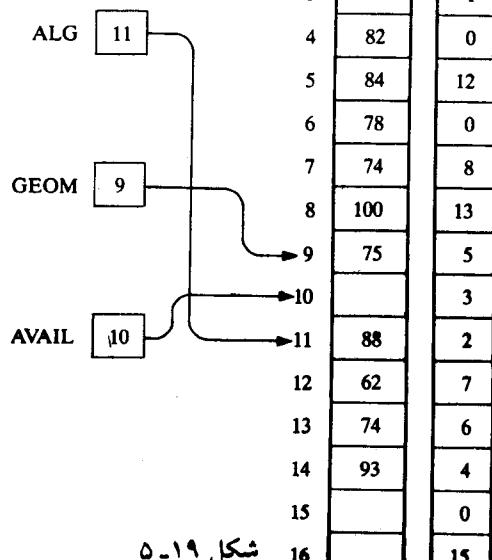
لیست آزمونهای شکل ۵-۱۰ را درنظر بگیرید. فرض کنید قرار است نمره آزمون ۷۵ را به ابتدای لیست هندسه GEOM اضافه کنیم. الگوریتم ۵.۴ را شبیه‌سازی می‌کنیم. ملاحظه کنید که

$$\text{START} = \text{GEOM}, \text{ITEM} = 75, \text{INFO} = \text{TEST}$$

شکل ۵-۱۹ این ساختمان داده را پس از اضافه کردن ۷۵ به لیست هندسه GEOM نشان می‌دهد. ملاحظه کنید که تنها سه اشاره‌گر GEOM، AVAIL و LINK[۹] تغییر کرده‌اند.

INSFIRST(TEST, LINK, GEOM, AVAIL, ITEM)

- Since AVAIL \neq NULL, control is transferred to Step 2.
- NEW = 9, then AVAIL = LINK[9] = 10.
- TEST[9] = 75.
- LINK[9] = 5.
- GEOM = 9.
- Exit.



اضافه کردن یک گره پس از یک گره معلوم

فرض کنید مقدار LOC داده شده است که در آن LIST گره A در لیست پیوندی است یا $LOC = NULL$. در زیر الگوریتمی ارائه شده است که ITEM را به درون لیست LIST به صورتی اضافه می‌کند که ITEM بعد از گره A قرار می‌گیرد یا وقتی که $ITEM = LOC = NULL$ گره A اول باشد. فرض کنید N نمایش گره جدید باشد که مکان آن NEW است. اگر $LOC = NULL$ آنگاه N به عنوان گره اول مانند الگوریتم 5.4 به لیست LIST اضافه می‌شود. در غیر این صورت، همانگونه که در شکل ۵-۱۵ رسم شده است، با دستور جایگزینی

LINK[NEW] := LINK[LOC]

گره N را به گره B اشاره می‌دهیم. که در آغاز بعد از گره A است. اکنون با دستور جایگزینی

LINK[LOC] := NEW

گره A به گره جدید N اشاره می‌کند.

بیان رسمی این الگوریتم به صورت زیر است :

Algorithm 5.5: INSLOC(INFO, LINK, START, AVAIL, LOC, ITEM)

This algorithm inserts ITEM so that ITEM follows the node with location LOC or inserts ITEM as the first node when LOC = NULL.

1. [OVERFLOW?] If AVAIL = NULL, then: Write: OVERFLOW, and Exit.
2. [Remove first node from AVAIL list.]
Set NEW := AVAIL and AVAIL := LINK[AVAIL].
3. Set INFO[NEW] := ITEM. [Copies new data into new node.]
4. If LOC = NULL, then: [Insert as first node.]
Set LINK[NEW] := START and START := NEW.
Else: [Insert after node with location LOC.]
Set LINK[NEW] := LINK[LOC] and LINK[LOC] := NEW.
[End of If structure.]
5. Exit.

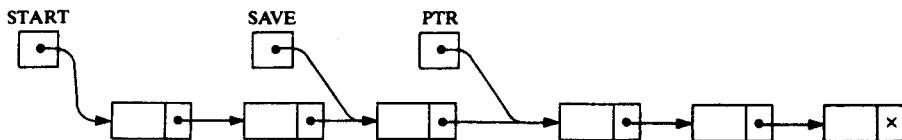
اضافه کردن گره در یک لیست پیوندی مرتب شده

فرض کنید خواسته باشیم ITEM را به لیست پیوندی مرتب شده LIST اضافه کنیم. آنگاه ITEM باید بین گره‌های A و B اضافه شود به گونه‌ای که

$$INFO(A) < ITEM \leq INFO(B)$$

در زیر، زیربرنامه Procedure ای ارائه شده است که مکان LOC گره A را پیدا می‌کند یعنی مکان

آخرین گره لیست LIST را پیدا می‌کند که مقدار آن کوچکتر از ITEM است. با استفاده از متغیر اشاره گر PTR لیست را پیمایش کنید و ITEM را با INFO[PTR] ای هر گره مقایسه کنید. هنگام پیمایش مکان گره قبلی را با استفاده از متغیر اشاره گر SAVE به صورتی که در شکل ۵-۲۰ آمده است نگهدارید.



شکل ۵-۲۰

بنابراین SAVE و PTR با دستورهای جایگزینی

$$\text{SAVE} := \text{PTR} \quad \text{PTR} := \text{LINK}[\text{PTR}]$$

تازه می‌شوند. عمل پیمایش تا زمانی که $\text{INFO}[\text{PTR}] > \text{ITEM}$ ادامه می‌یابد. به بیان دیگر به محض اینکه $\text{ITEM} \leq \text{INFO}[\text{PTR}]$ ، پیمایش متوقف می‌شود، آنگاه PTR به گره B اشاره می‌کند، از این‌رو SAVE حاوی مکان گره A خواهد بود.

بیان رسمی این Procedure در زیر ارائه شده است. در حالت‌هایی که لیست خالی است یا $\text{ITEM} < \text{INFO}[\text{START}]$ ، درنتیجه LOC = NULL، مستقلاً بررسی می‌شوند، چون از متغیر SAVE استفاده نمی‌کند.

Procedure 5.6: FINDA(INFO, LINK, START, ITEM, LOC)

This procedure finds the location LOC of the last node in a sorted list such that $\text{INFO}[LOC] < \text{ITEM}$, or sets LOC = NULL.

1. [List empty?] If START = NULL, then: Set LOC := NULL, and Return.
2. [Special case?] If ITEM < INFO[START], then: Set LOC := NULL, and Return.
3. Set SAVE := START and PTR := LINK[START]. [Initializes pointers.]
4. Repeat Steps 5 and 6 while PTR ≠ NULL.
5. If ITEM < INFO[PTR], then:
 Set LOC := SAVE, and Return.
 [End of If structure.]
6. Set SAVE := PTR and PTR := LINK[PTR]. [Updates pointers.]
 [End of Step 4 loop.]
7. Set LOC := SAVE.
8. Return.

اکنون تمام مؤلفه‌های موردنیاز را برای ارائه یک الگوریتم جهت اضافه کردن ITEM در لیست

پیوندی در اختیار داریم. سادگی این الگوریتم از این واقعیت ناشی می‌شود که در آن از دو زیربرنامه **Procedure** قبلی استفاده شده است.

Algorithm 5.7: INSSRT(INFO, LINK, START, AVAIL, ITEM)
This algorithm inserts ITEM into a sorted linked list.

1. [Use Procedure 5.6 to find the location of the node preceding ITEM.]
Call FINDA(INFO, LINK, START, ITEM, LOC).
2. [Use Algorithm 5.5 to insert ITEM after the node with location LOC.]
Call INSLOC(INFO, LINK, START, AVAIL, LOC, ITEM).
3. Exit.

۵-۱۵ مثال

لیست الفبایی بیماران شکل ۹-۵ را در نظر بگیرید. فرض کنید Jones قرار است به لیست بیماران اضافه شود. الگوریتم ۵.۷ را شبیه‌سازی می‌کنیم یا به بیان خصوصی تر زیربرنامه **Procedure 5.6** و آنگاه الگوریتم ۵.۵ را شبیه‌سازی می‌کنیم. ملاحظه می‌کنید که INFO = BED و ITEM = Jones و

FINDA(BED, LINK, START, ITEM, LOC) (الف)

1. Since START ≠ NULL, control is transferred to Step 2.
2. Since BED[5] = Adams < Jones, control is transferred to Step 3.
3. SAVE = 5 and PTR = LINK[5] = 3.
4. Steps 5 and 6 are repeated as follows:
 - (a) BED[3] = Dean < Jones, so SAVE = 3 and PTR = LINK[3] = 11.
 - (b) BED[11] = Fields < Jones, so SAVE = 11 and PTR = LINK[11] = 8.
 - (c) BED[8] = Green < Jones, so SAVE = 8 and PTR = LINK[8] = 1.
 - (d) Since BED[1] = Kirk > Jones, we have:
LOC = SAVE = 8 and Return.

INSLOC(BED, LINK, START, AVAIL, LOC, ITEM) [Here LOC = 8.] (ب)

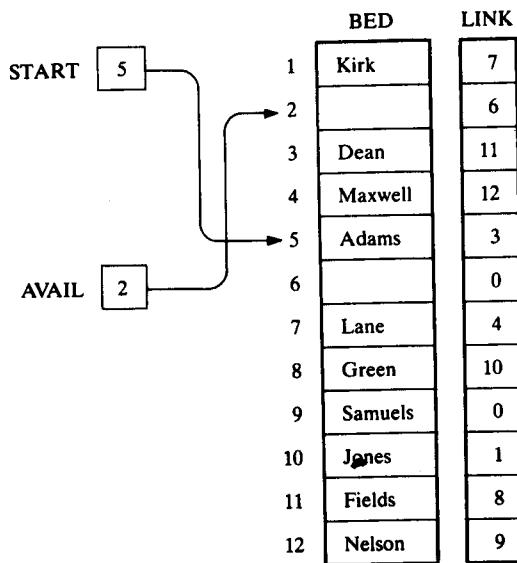
1. Since AVAIL ≠ NULL, control is transferred to Step 2.
2. NEW = 10 and AVAIL = LINK[10] = 2.
3. BED[10] = Jones.
4. Since LOC ≠ NULL we have:
LINK[10] = LINK[8] = 1 and LINK[8] = NEW = 10.
5. Exit.

شکل ۹-۵ این ساختمان داده را پس از اضافه شدن Jones به لیست بیماران نشان می‌دهد. تأکید می‌کنیم که تنها سه اشاره گر **LINK[10]**, **LINK[8]** و **AVAIL** تغییر کرده‌اند.

کپی کردن

فرض کنید خواسته باشیم تمام یا قسمت‌هایی از یک لیست معلوم را در لیست دیگری کپی کنیم یا لیست جدیدی ایجاد کنیم که از اتصال دو لیست داده شده بوجود آید. این کار را می‌توان با تعریف یک

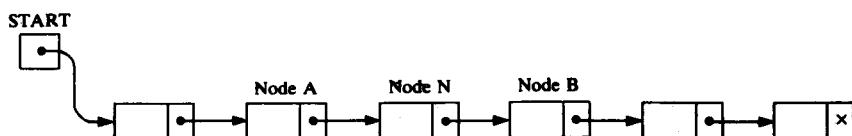
لیست خالی و آنگاه اضافه کردن یک به یک عناصر موردنظر به لیست، به کمک الگوریتمهای متعدد اضافه کردن انجام داد. یک لیست خالی تنها با انتخاب یک نام متغیر یا اشاره گر برای لیست نظیر **NAME := NULL** تعریف می‌شود و آنگاه قرار می‌دهیم **NAME := NULL**. این الگوریتم‌ها در بخش مسائل ارائه شده است.



شکل ۵-۲۱

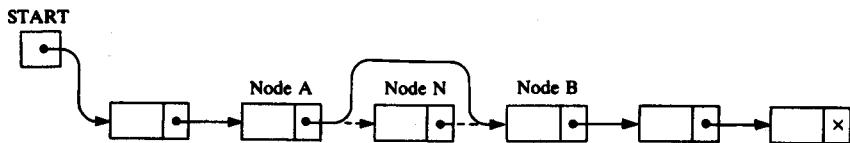
۵-۸ حذف گره از یک لیست پیوندی

فرض کنید لیست LIST یک لیست پیوندی باشد که همانند شکل ۵-۲۲ (الف) گره N آن بین دو گره A و B است.



شکل ۵-۲۲ (الف)

فرض کنید گره N را می‌خواهیم از لیست پیوندی حذف کنیم. نمودار چنین عمل حذفی، در شکل ۵-۲۲ (ب) آمده است.



شکل ۵-۲۲ (ب)

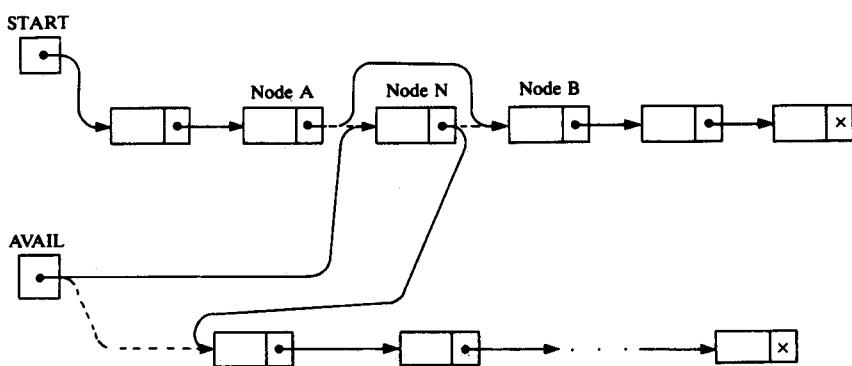
عمل حذف زمانی انجام می‌شود که فیلد اشاره‌گر بعدی گره A به گونه‌ای تغییر کند که به گره B اشاره کند. بنابراین هنگام انجام عمل حذف، باید آدرس گره‌ای را که گره حذف شده به آن اشاره می‌کرد در جایی نگهداریم.

فرض کنید لیست پیوندی به صورت زیر در حافظه نگهداری شده است :

LIST(INFO, LINK, START, AVAIL)

شکل ۵-۲۲-۵ این واقعیت را که حذف گره N از لیست، حافظه آن را بسیار بزرگ به لیست **AVAIL** بر می‌گرداند در نظر نمی‌گیرد. بهویژه این که، برای پردازش ساده‌تر، به ابتدای لیست **AVAIL** برگردانده می‌شود. بنابراین نمودار دقیق‌تر چنین عمل حذفی در شکل ۵-۲۳ نموده است.

لیست داده‌ها



لیست حافظه آزاد

شکل ۵-۲۳

مالحظه می‌کنید که سه فیلد اشاره‌گر به صورت زیر تغییر کرده‌اند:

- (۱) فیلد اشاره‌گر بعدی گره A اکنون به گره B اشاره می‌کند که قبلًاً به گره N اشاره می‌کرد.
- (۲) فیلد اشاره‌گر بعدی گره N اکنون به اولین گره اصلی مخزن حافظه اشاره می‌کند که قبلًاً به گره

اشاره می‌کرد.

(۳) AVAIL اکنون به گرۀ حذف شده N اشاره می‌کند.
همچنین دو حالت خاص وجود دارد. اگر گرۀ حذف شده N اولین گرۀ لیست باشد، آنگاه START به گرۀ B اشاره می‌کند و اگر گرۀ حذف شده N آخرین گرۀ لیست باشد. آنگاه گرۀ A حاوی اشاره گر NULL خواهد بود.

مثال ۵-۱۶

(الف) شکل ۵-۲۱، لیست بیماران در اتاق بیمارستان را درنظر بگیرید. فرض کنید بیمار Green از بیمارستان مرخص شده است از این رو تخت BED[8] اکنون خالی است، آنگاه برای نگهداری لیست پیوندی، باید سه تغییر زیر در فیلد اشاره گر داده شود:

$$\text{LINK}[11] = 10 \quad \text{LINK}[8] = 2 \quad \text{AVAIL} = 8$$

با تغییر اول، Fields که در آغاز قبل از Green بود اکنون به Jones اشاره می‌کند که در آغاز بعد از قرار داشت. تغییر دوم و سوم تخت خالی جدید را به لیست AVAIL اضافه می‌کند. تأکید می‌کنیم که قبل از عمل حذف باید گرۀ [BED[11]] را پیدا کنیم که در آغاز به گرۀ حذف شده BED[8] اشاره می‌کرد.

(ب) شکل ۵-۱۳ را درنظر بگیرید که لیست مأموران فروش و مشتری آنها را ارائه می‌دهد. فرض کنید اولین مشتری Nelson، از لیست مشتری‌ها حذف شده است. آنگاه برای نگهداری لیستهای پیوندی باید سه تغییر زیر در فیلد اشاره گرها داده شود:

$$\text{POINT}[4] = 10 \quad \text{LINK}[9] = 11 \quad \text{AVAIL} = 9$$

با تغییر اول، Nelson اکنون به دومین مشتری او یعنی Jones اشاره می‌کند. تغییر دوم و سوم گرۀ خالی جدید را به لیست AVAIL اضافه می‌کند.

(ج) فرض کنید عناصر داده‌ای B، E و C یکی پس از دیگری از لیست شکل ۵-۱۶ حذف شده‌اند. لیست جدید در شکل ۵-۲۴ رسم شده است.

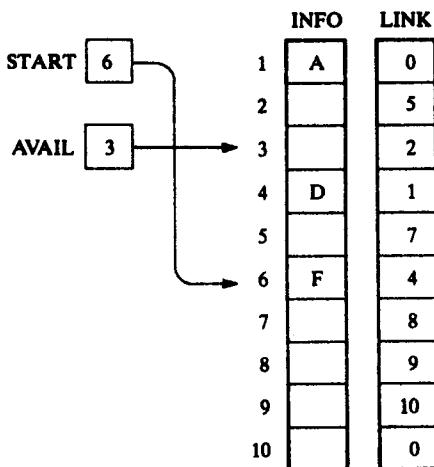
ملاحظه می‌کنید که اکنون سه گرۀ در دسترس اول به قرار زیرند:

INFO[3] که در آغاز حاوی C است.

INFO[2] که لر آغاز حاوی B است.

INFO[5] که در آغاز حاوی E است.

ملاحظه می‌کنید که ترتیب گرۀ‌ها در لیست AVAIL عکس ترتیبی است که در آن گرۀ‌ها از لیست حذف شده‌اند.



شکل ۵-۲۴

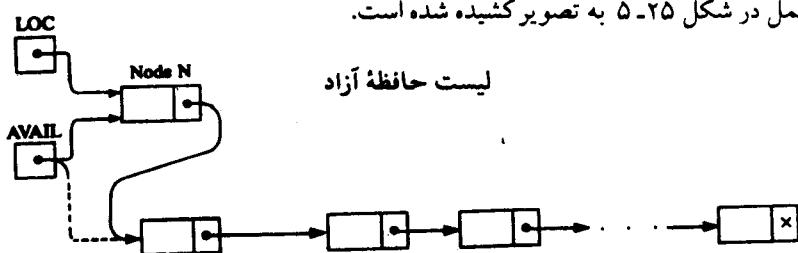
الگوریتم‌های حذف گره‌ها

الگوریتم‌هایی که گره‌هایی را از لیستهای پیوندی حذف می‌کنند در وضعیت‌ها و شرایط مختلف می‌توانند مطرح شوند. الگوریتم اول، گره بعد از یک گره داده شده را حذف می‌کند. الگوریتم دوم، گره‌ای با اطلاعات معلوم ITEM را حذف می‌کند. تمام این الگوریتم‌ها فرض می‌کنند که لیست پیوندی به صورت $\text{LIST}(\text{INFO}, \text{LINK}, \text{START}, \text{AVAIL})$ در حافظه است.

تمام الگوریتم‌های حذف گره‌ها فضای حافظه گره حذف شده N را به ابتدای لیست AVAIL برمی‌گردانند. بنابراین، تمام این الگوریتم‌ها شامل جفت دستور جایگزینی زیر هستند که در آنها LOC مکان گره حذف شده N است:

$$\text{AVAIL} := \text{LOC} \quad \text{LINK}[\text{LOC}] := \text{AVAIL}$$

این دو عمل در شکل ۵-۲۵ به تصویر کشیده شده است.

شکل ۵-۲۵ $\text{AVAIL} := \text{LOC} \quad \text{LINK}[\text{LOC}] := \text{AVAIL}$

در بعضی از این الگوریتم‌ها می‌خواهیم گره اول یا گره آخر از لیست حذف شود. الگوریتمی که این کار را انجام می‌دهد باید تحقیق کند که آیا در لیست گره‌ای وجود دارد یا خیر. در صورت منفی بودن جواب، اگر $START = NULL$ آنگاه الگوریتم پیغام UNDERFLOW را چاپ خواهد کرد.

حذف گره‌ای که بعد از یک گره معلوم قرار دارد

فرض کنید LIST یک لیست پیوندی در حافظه باشد. فرض کنید LOC مکان گره N در LIST داده شده است. علاوه بر این فرض کنید مکان LOCP گره قبل از N یا وقتی N گره اول است. $LOCP = NULL$ داده شده است. الگوریتم زیر N را از لیست حذف می‌کند.

Algorithm 5.8: DEL(INFO, LINK, START, AVAIL, LOC, LOCP)

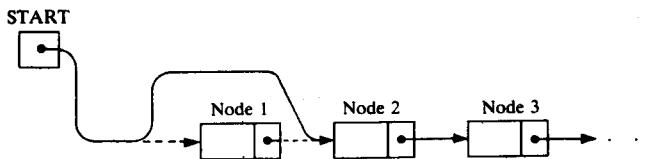
This algorithm deletes the node N with location LOC. LOCP is the location of the node which precedes N or, when N is the first node, LOCP = NULL.

1. If $LOCP = NULL$, then:
Set $START := LINK[START]$. [Deletes first node.]
- Else:
Set $LINK[LOCP] := LINK[LOC]$. [Deletes node N.]
[End of If structure.]
2. [Return deleted node to the AVAIL list.]
Set $LINK[LOC] := AVAIL$ and $AVAIL := LOC$.
3. Exit.

شکل ۵-۲۶ نمودار جایگزینی

$START := LINK[START]$

را نشان می‌دهد که به صورت مؤثری اولین گره را از لیست حذف می‌کند.



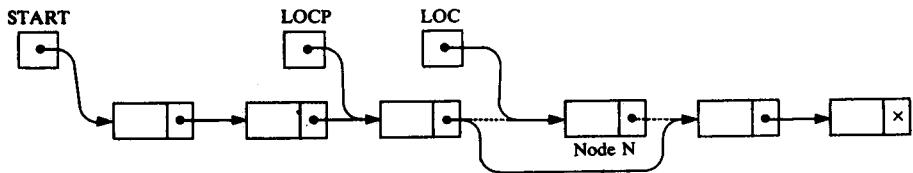
شکل ۵-۲۶

این نمودار شامل حالتی است که N گره اول است.

شکل ۵-۲۷ نمودار جایگزینی

$LINK[LOCP] := LINK[LOC]$

را نشان می‌دهد که به صورت مؤثری گره N را در حالتی که گره اول نیست حذف می‌کند.

شکل ۵-۲۷ $\text{LINK}[\text{LOC}_P] := \text{LINK}[\text{LOC}]$

سادگی این الگوریتم از این واقعیت ناشی می‌شود که پیشتر **LOC_P**، مکان‌گرهای که قبل از گره **N** است داده شده است. در بسیاری از کاربردها، نخست باید **LOC_P** را پیدا کنیم.

حذف یک گره با اطلاعات معلوم ITEM

فرض کنید **LIST** یک لیست پیوندی در حافظه باشد. فرض کنید اطلاعات **ITEM** داده شده است و می‌خواهیم **N** اولین گره **LIST** را حذف کنیم که حاوی اطلاعات **ITEM** است. اگر **ITEM** یک مقدار کلیدی باشد. آنگاه تنها یک گره شامل اطلاعات **ITEM** است. یادآوری می‌کنیم که برای حذف **N** از لیست لازم است مکان‌گره قبل از **N** را بدانیم. بنابراین نخست یک زیر برنامه **Procedure** ارائه می‌دهیم که مکان‌گره حاوی **ITEM** و **LOC_P** مکان‌گره قبل از گره **N** را پیدا می‌کند. اگر **N** اولین گره لیست باشد، قرار می‌دهیم **LOC_P = NULL** و اگر **ITEM** در لیست **LIST** وجود نداشته باشد قرار می‌دهیم **LOC = NULL** این زیر برنامه **Procedure** شبیه زیر برنامه ۵.۶ است.

با استفاده از متغیر اشاره گر **PTR** و مقایسه **ITEM** با **INFO[PTR]** هر گره لیست را پیمایش کنید. هنگام پیمایش، مکان‌گره قبلی با استفاده از متغیر اشاره گر **SAVE** به صورتی که در شکل ۵-۲۰ رسم شده است نگهدارید. بدین ترتیب **SAVE** و **PTR** با دستورهای جایگزینی زیر

$$\text{PTR} := \text{LINK}[\text{PTR}] \quad \text{SAVE} := \text{PTR}$$

تازه می‌شوند. پیمایش تا آنجا ادامه پیدا می‌کند که $\text{INFO}[\text{PTR}] \neq \text{ITEM}$ یا به بیان دیگر هرگاه **ITEM = INFO[PTR]** باشد پیمایش متوقف می‌شود. آنگاه **PTR** شامل **LOC** مکان‌گره **N** و **SAVE** شامل **LOC_P** مکان‌گره قبل از **N** است.

بیان رسمی این **Procedure** به صورت زیر است. حالتهایی که لیست خالی است یا $\text{INFO}[\text{START}] = \text{ITEM}$ (یعنی گره **N** اولین گره لیست است) مستقلًاً مورد بررسی قرار می‌گیرد چون از متغیر **SAVE** استفاده نمی‌کند.

Procedure 5.9: FINDB(INFO, LINK, START, ITEM, LOC, LOCP)

This procedure finds the location LOC of the first node N which contains ITEM and the location LOCP of the node preceding N. If ITEM does not appear in the list, then the procedure sets LOC = NULL; and if ITEM appears in the first node, then it sets LOCP = NULL.

1. [List empty?] If START = NULL, then:
 Set LOC := NULL and LOCP := NULL, and Return.
 [End of If structure.]
2. [ITEM in first node?] If INFO[START] = ITEM, then:
 Set LOC := START and LOCP = NULL, and Return.
 [End of If structure.]
3. Set SAVE := START and PTR := LINK[START]. [Initializes pointers.]
4. Repeat Steps 5 and 6 while PTR ≠ NULL.
5. If INFO[PTR] = ITEM, then:
 Set LOC := PTR and LOCP := SAVE, and Return.
 [End of If structure.]
6. Set SAVE := PTR and PTR := LINK[PTR]. [Updates pointers.]
 [End of Step 4 loop.]
7. Set LOC := NULL. [Search unsuccessful.]
8. Return.

اگر نون به راحتی می‌توانیم الگوریتمی ارائه دهیم که N گرده اول را از لیست پیوندی‌ای که شامل اطلاعات معلوم ITEM است حذف کند. سهولت این الگوریتم از این واقعیت ناشی می‌شود که کار پیداکردن مکان N و مکان گرده قبل از آن قبل‌از زیربرنامه 5.9 انجام شده است.

Algorithm 5.10: DELETE(INFO, LINK, START, AVAIL, ITEM)

This algorithm deletes from a linked list the first node N which contains the given ITEM of information.

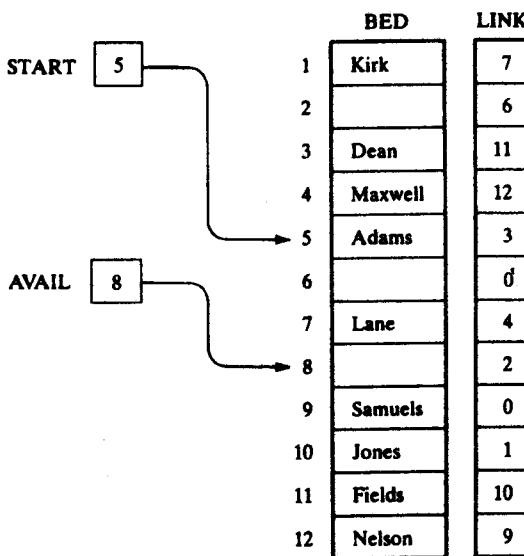
1. [Use Procedure 5.9 to find the location of N and its preceding node.]
Call FINDB(INFO, LINK, START, ITEM, LOC, LOCP)
2. If LOC = NULL, then: Write: ITEM not in list, and Exit.
3. [Delete node.]
 If LOCP = NULL, then:
 Set START := LINK[START]. [Deletes first node.]
 Else:
 Set LINK[LOCP] := LINK[LOC].
 [End of If structure.]
4. [Return deleted node to the AVAIL list.]
 Set LINK[LOC] := AVAIL and AVAIL := LOC.
5. Exit.

توجه کنید: دانشجو ممکن است به این نکته توجه کرده باشد که مراحل ۳ و ۴ در الگوریتم 5.10 قبل از الگوریتم 5.8 وجود داشته است. به بیان دیگر می‌توانیم دستور CALL زیر را جایگزین این مراحل کنیم:

این کار با سبک ماجولاریتی یا پیمانه‌ای در برنامه‌نویسی سازگاری دارد.

مثال ۵-۱۷

لیست بیماران شکل ۵-۲۱ را درنظر بگیرید. فرض کنید بیماری به نام **Green** از بیمارستان مرخص شده است. زیربرنامه ۵.۹ **Procedure** را شبیه‌سازی می‌کنیم تا **LOC** مکان این بیمار یعنی **Green** و **LOCOP** مکان بیمار قبل از **Green** را پیدا کند. بدنبال آن الگوریتم ۵.۱۰ را شبیه‌سازی می‌کنیم تا **Green** را از لیست حذف کند. در اینجا **ITEM = Green**, **INFO = BED**, **START = 5** و **AVAIL = 2**



شکل ۵-۲۸

(الف)

FINDB(BED, LINK, START, ITEM, LOC, LOCP)

1. Since **START ≠ NULL**, control is transferred to Step 2.
2. Since **BED[5] = Adams ≠ Green**, control is transferred to Step 3.
3. **SAVE = 5** and **PTR = LINK[5] = 3**.
4. Steps 5 and 6 are repeated as follows:
 - (a) **BED[3] = Dean ≠ Green**, so **SAVE = 3** and **PTR = LINK[3] = 11**.
 - (b) **BED[11] = Fields ≠ Green**, so **SAVE = 11** and **PTR = LINK[11] = 8**.
 - (c) **BED[8] = Green**, so we have:
LOC = PTR = 8 and **LOCOP = SAVE = 11**, and Return.

- DEALLOC(BED, LINK, START, AVAIL, ITEM)
1. Call FINDB(BED, LINK, START, ITEM, LOC, LOCP). [Hence LOC = 8 and LOCP = 11.]
 2. Since LOC ≠ NULL, control is transferred to Step 3.
 3. Since LOCP ≠ NULL, we have:
 $\text{LINK}[11] = \text{LINK}[8] = 10.$
 4. $\text{LINK}[8] = 2$ and $\text{AVAIL} = 8.$
 5. Exit.

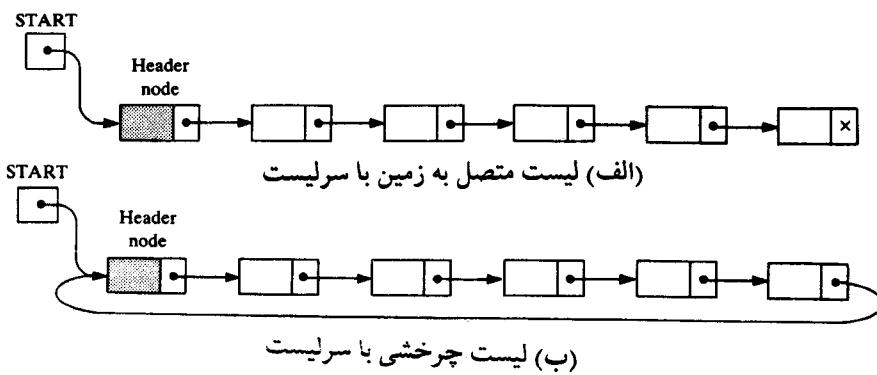
شکل ۵-۲۸ این ساختمان داده را پس از حذف Green از لیست بیماران نشان می دهد. تأکید می کنیم که تنها سه اشاره گر $\text{LINK}[8]$, $\text{LINK}[11]$ و AVAIL تغییر می کنند.

۵-۹ لیست پیوندی با سر لیست

یک لیست پیوندی با سر لیست یک لیست پیوندی است که همیشه دارای یک گره خاص موسوم به سرگره Header در ابتدای لیست است. در زیر دو نوع لیست دارای سر لیست که زیاد مورد استفاده قرار می گیرند ارائه می شود:

- (۱) لیست متصل به زمین با سر لیست، یک لیست دارای سرگره است که آخرین گره آن حاوی اشاره گر پوچ است. اصطلاح متصل به زمین از این واقعیت ناشی شده است که بسیاری از نمادهای اتصال به زمین جریان الکتریکی برای بیان اشاره گر پوچ استفاده می کنند.
- (۲) یک لیست چرخشی یا حلقوی با سر لیست، یک لیست دارای سرگره است که آخرین گره آن به سرگره لیست اشاره می کند.

شکل ۵-۲۹ نمودارهای این لیستهای دارای سر لیست را نشان می دهد. لیستهای دارای سر لیست همیشه چرخشی هستند مگر آن که خلاف آن به صورت صریح یا ضمنی بیان گردد. بنابراین در چنین مواردی سرگره نیز به عنوان نگهبانی که مبین پایان لیست است عمل می کند.



شکل ۵-۲۹

ملاحظه می‌کنید که اشاره‌گر لیست **START** همچه سرگره اشاره می‌کند. بنابراین **LINK[START] = NULL** میان آن است که لیست متصل به زمین دارای سرلیست خالی است و **LINK[START] = START** بیانگر آن است که لیست چرخشی دارای سرلیست خالی است. اگرچه داده‌هایمان به وسیله لیست‌های دارای سرلیست در حافظه ذخیره می‌شود، با وجود این لیست **AVAIL** همیشه به صورت یک لیست پیوندی معمولی نگهداری می‌شود.

۵-۱۸ مثال

فایل پرسنلی شکل ۱۱-۵ را درنظر بگیرید. داده‌ها را می‌توان به صورت یک لیست دارای سرلیست مانند شکل ۵-۳۰ سازمان داد. ملاحظه می‌کنید که **LOC = 5** اکنون مکان سر رکورد است. بنابراین، **START = 5** و چون **Rubin** آخرین کارمند است، **LINK[10] = 5**. سررکورد را می‌توان برای ذخیره اطلاعات تقریباً تمام فایل نیز بكارگرفت. برای مثال، فرض می‌کنیم **SSN[5] = 009** بیانگر تعداد کارمندان و **SALARY[5] = 191 600** بیانگر حقوق کل پرداخت شده به کارمندان باشد.

	NAME	SSN	SEX	SALARY	LINK
1					0
2	Davis	192-38-7282			12
3	Kelly	165-64-3351	Female	22 800	7
4	Green	175-56-2251	Male	19 000	14
5		009	Male	27 200	11
6	Brown	178-52-1065	Female	191 600	9
7	Lewis	181-58-9939	Female	14 700	10
8					11
9	Cohen	177-44-4557	Male	16 400	2
10	Rubin	135-46-6262	Female	19 000	5
11					13
12	Evans	168-56-8113	Male	15 500	4
13					1
14	Harris	208-56-1654	Female	34 200	3
				22 800	

شکل ۵-۳۰

اصطلاح "گره" به تهایی هنگامی که با لیست دارای سرلیست بکار گرفته شود اشاره به گره معمولی دارد نه سرگره. بدین ترتیب گره اول در یک لیست دارای سرلیست، گره‌ای است که بعد از سرگره قرار می‌گیرد و مکان گره اول **LINK[START]** است نه **START** به صورتی که با لیستهای پیوندی معمولی بکار گرفته می‌شد.

الگوریتم 5.11 که از یک متغیر اشاره گر **PTR** برای پیمایش یک لیست چرخشی با سرلیست استفاده می‌کند ذاتاً همان الگوریتم 5.1 است که یک لیست پیوندی معمولی را پیمایش می‌کند. اما اکنون این تفاوت وجود دارد که الگوریتم (۱) با $PTR = LINK[START] = START$ شروع می‌شود نه با $PTR = PTR$ و (۲) با $PTR = START$ به پایان می‌رسد نه با $PTR = NULL$.

لیستهای چرخشی دارای سرلیست اغلب به جای لیستهای پیوندی معمولی بکار گرفته می‌شوند چون اغلب با استفاده از لیستهای دارای سرلیست بهتر و ساده‌تر بیان شده و پیاده‌سازی می‌شوند. این مطلب از دو خاصیت زیر از لیستهای چرخشی دارای سرلیست ناشی می‌شود:

- (۱) اشاره گر پوج مورد استفاده قرار نمی‌گیرد و از این‌رو تمام اشاره گرها دارای آدرس مجاز هستند.
- (۲) هر گره (معمولی) یک گره پیشین دارد از این‌رو گره اول نمی‌تواند نیازمند حالت خاص باشد. مثال بعدی فایده این دو خاصیت را بیان می‌کند.

Algorithm 5.11: (Traversing a Circular Header List) Let LIST be a circular header list in memory. This algorithm traverses LIST, applying an operation PROCESS to each node of LIST.

1. Set $PTR := LINK[START]$. [Initializes the pointer PTR.]
2. Repeat Steps 3 and 4 while $PTR \neq START$:
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.

مثال ۵-۱۹

فرض کنید LIST یک لیست پیوندی در حافظه باشد و ITEM یعنی یک اطلاع خاص داده شده است.

(الف) الگوریتم 5.2 مکان گره اول در LIST را پیدا می‌کند که شامل ITEM است مشروط بر این که LIST یک لیست پیوندی معمولی باشد. الگوریتم زیر از این‌گونه الگوریتم است که در آن LIST یک لیست چرخشی با سرلیست است.

Algorithm 5.12: SRCHHL(INFO, LINK, START, ITEM, LOC)
 LIST is a circular header 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 := LINK[START].
2. Repeat while INFO[PTR] ≠ ITEM and PTR ≠ START:
 Set PTR := LINK[PTR]. [PTR now points to the next node.]
 [End of loop.]
3. If INFO[PTR] = ITEM, then:
 Set LOC := PTR.
 Else:
 Set LOC := NULL.
 [End of If structure.]
4. Exit.

دو آزمونی که حلقه جستجو را کنترل می‌کند (مرحله ۲ در الگوریتم ۵.12) به صورت هم زمان در الگوریتم برای لیستهای پیوندی معمولی اجرا نمی‌شوند یعنی الگوریتم ۵.۲ اجازه ندارد از دستوری شبیه $PTR \neq NULL$ و **Repeat while $INFO[PTR] \neq ITEM$** استفاده کند زیرا برای لیستهای پیوندی معمولی هنگامی که $PTR = NULL$ باشد $INFO[PTR]$ تعریف نشده است.

(ب) زیربرنامه ۵.۹ مکان LOC گرده اول N را پیدا می‌کند که شامل ITEM است و همچنین LOC مکان گرده قبل از N را وقتی LIST یک لیست پیوندی معمولی است پیدا می‌کند. هنگامی که LIST یک لیست چرخشی با سرلیست است زیربرنامه به صورت زیر است:

Procedure 5.13: FINDBHL(INFO, LINK, START, ITEM, LOC, LOCP)

1. Set SAVE := START and PTR := LINK[START]. [Initializes pointers.]
2. Repeat while INFO[PTR] ≠ ITEM and PTR ≠ START:
 Set SAVE := PTR and PTR := LINK[PTR]. [Updates pointers.]
 [End of loop.]
3. If INFO[PTR] = ITEM, then:
 Set LOC := PTR and LOCP := SAVE.
 Else:
 Set LOC := NULL and LOCP := SAVE.
 [End of If structure.]
4. Exit.

به سادگی این زیربرنامه Procedure در مقایسه با ۵.۹ توجه کنید. در اینجا نباید حالت خاصی را که ITEM در گرده اول ظاهر می‌شود مورد توجه قرار دارد و از این‌رو می‌توان در یک زمان دو آزمونی را که حلقه را کنترل می‌کنند انجام داد.

(ج) الگوریتم ۵.۱۰ گرده اول N را حذف می‌کند که وقتی ITEM یک لیست پیوندی معمولی است شامل LIST است. هنگامی که LIST یک لیست چرخشی با سرلیست است الگوریتم زیر این‌گونه است.

Algorithm 5.14: DELLOCHL(INFO, LINK, START, AVAIL, ITEM)

1. [Use Procedure 5.13 to find the location of N and its preceding node.]
Call LINDBHL(INFO, LINK, START, ITEM, LOC, LOCP).
2. If LOC = NULL, then: Write: ITEM not in list, and Exit.
3. Set LNK[LOC] := LINK[LOC]. [Deletes node.]
4. [Return deleted node to the AVAIL list.]
Set LINK[LOC] := AVAIL and AVAIL := LOC.
5. Exit.

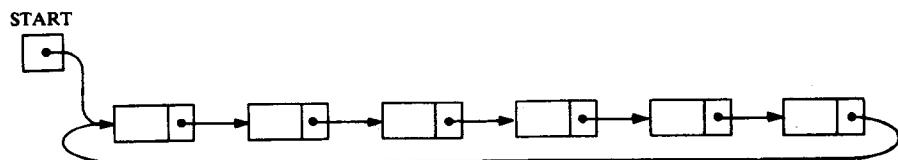
مجدداً نباید حالت خاصی را که ITEM در گره اول ظاهر می‌شود و به صورتی که در الگوریتم 5.10 عمل کردیم درنظر بگیریم.

توجه کنید: دو نوع دیگر از لیستهای پیوندی وجود دارد که بعضی وقتها در برخی از کتابهای ساختمان داده‌ها مطرح می‌شود:

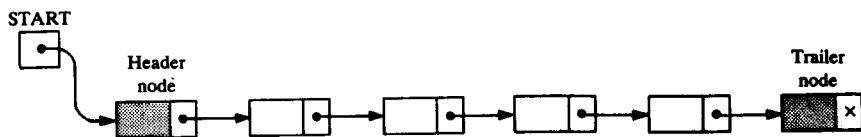
(۱) یک لیست پیوندی که گره آخرش به جای این که شامل اشاره‌گر پوج باشد به گره اول اشاره می‌کند که به آن لیست چرخشی یا حلقوی می‌گویند.

(۲) یک لیست پیوندی که شامل سرگره خاصی در ابتدای لیست و هم گره انتهایی خاص دیگری در پایان لیست است.

شکل ۵-۳۱ نمودارهای این لیستها را نشان می‌دهد.



(الف) لیست پیوندی چرخشی



(ب) لیست پیوندی با سرگره و گره انتهایی

شکل ۵-۳۱

چند جمله‌ایها

لیستهای پیوندی دارای سرلیست غالباً برای ذخیره چند جمله‌ایها در حافظه مورد استفاده قرار

می‌گیرند. سریست قسمت مهم این نمایش را تشکیل می‌دهد زیرا به آن در نمایش چند جمله‌ای صفر نیاز داریم. این نمایش چند جمله‌ای‌ها در قالب یک مثال خاص ارائه می‌شود.

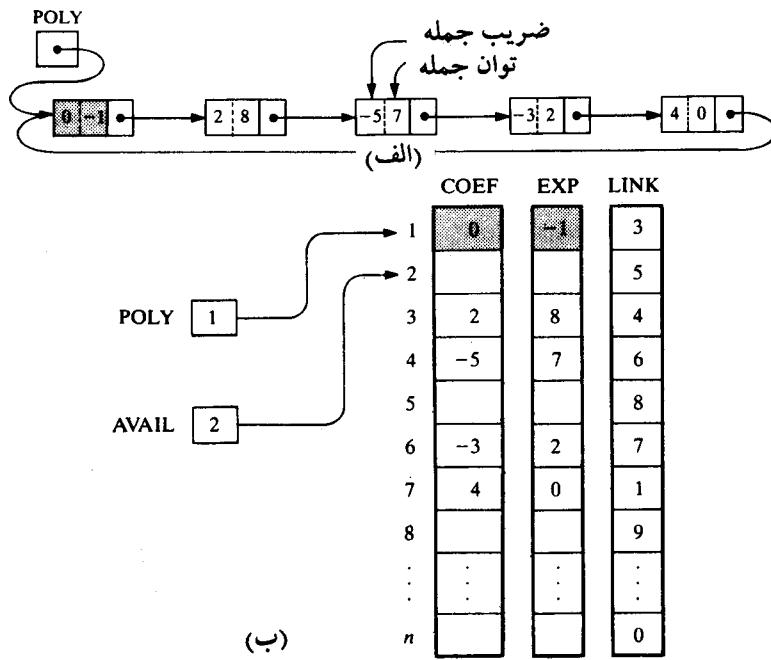
مثال ۵-۲۰

فرض کنید (x, p) نمایش چندجمله‌ای یک متغیره زیر باشد که شامل چهار جمله غیر صفر است:

$$p(x) = 2x^8 - 5x^7 - 3x^2 + 4$$

آنگاه (x) را می‌توان به وسیله لیست دارای سرلیست مانند شکل ۵-۳۲ (الف) نمایش داد که در آن هر گره متناظر با یک جمله غیر صفر از (x) است به ویژه این که قسمت اطلاعات گره به دو فیلد تقسیم شده است که به ترتیب ضریب و توان جمله متناظر آن را نمایش می‌دهند و گره‌ها با توجه به نزول درجه‌ها، پیوند خورده‌اند.

مالحظه می کنید که لیست متغیر اشاره گر POLY به سرگرهای اشاره می کند که در فیلد توان آن یک عدد منفی، در اینجا -۱، جایگزین شده است. اکنون نمایش آرایه ای لیست مستلزم سه آرایه خطی است که ما آنها را COEF, EXP و LINK نامگذاری کرده ایم. یک چنین نمایشی در شکل ۳-۲ (ب) ظاهر شده است.



$$p(x) = 2x^8 - 5x^7 - 3x^2 + 4 \quad \text{شكل ٣٢}$$

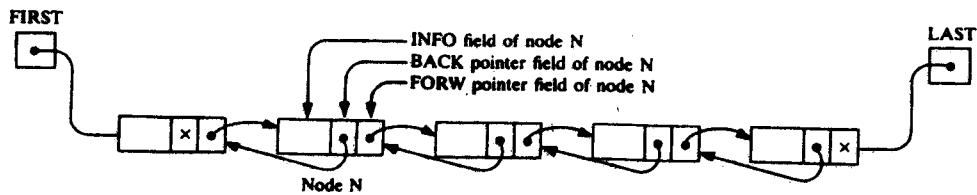
۵-۱۰ لیستهای دوطرفه

به لیستی که در بالا مورد بررسی قرار گرفت لیست یک طرفه می‌گویند. زیرا تنها از یک طرف می‌توان لیست را پیمایش کرد یعنی با شروع از **START** متغیر اشاره گر لیست که به گره اول یا سرگره اشاره می‌کند و با استفاده از فیلد اشاره گر بعدی لیست اشاره می‌کند می‌توان لیست را تنها در یک جهت پیمایش کرد. علاوه بر این، با معلوم بودن **LOC** مکان گره **N** در چنین لیستی، بلاfacله می‌توان به گره بعدی در لیست (با ارزیابی **LINK[LOC]**) دسترسی پیدا کرد. اما بدون پیمایش قسمتی از لیست نمی‌توان به گره قبلی دسترسی پیدا کرد. درحال خاص معنی آن این است که باید قسمتی از لیست قبل از **N** را برای حذف **N** از لیست پیمایش کرد.

این بخش ساختمان یک لیست جدید موسوم به لیست دوطرفه را معرفی می‌کند که می‌توان آن را در دو جهت پیمایش کرد: در جهت متدالو و معمولی به پیش‌رفتن، از ابتدای لیست تا انتهای آن یا درجهت عکس از انتهای لیست به ابتدای لیست. علاوه بر این، با معلوم بودن **LOC** مکان یک گره **N** در لیست، اکنون می‌توان بلاfacله هم به گره بعدی و هم به گره قبل از آن در لیست دسترسی پیدا کرد. در حالت خاص معنی آن این است که می‌توان بدون پیمایش هر قسمت از لیست **N** را از لیست حذف کرد. یک لیست دوطرفه یک مجموعه خطوط از عناصر داده‌ای بنام گره‌ها است که در آن هر گره **N** به سه

قسمت تقسیم می‌شود:

- (۱) یک فیلد اطلاعات **INFO** که شامل داده **N** است.
 - (۲) یک فیلد اشاره گر **FORW** که شامل مکان گره بعدی در لیست است.
 - (۳) یک فیلد اشاره گر **BACK** که شامل مکان گره قبل از آن در لیست است.
- علاوه بر این لیست نیازمند دو متغیر اشاره گر است: **FIRST** که به گره اول و **LAST** به گره آخر در لیست اشاره می‌کند. شکل ۵-۳۲ شامل یک نمودار از چنین لیستی است. ملاحظه می‌کنید که اشاره گر **FORW** در فیلد **FORW** از گره آخر لیست و همچنین در فیلد **BACK** از گره اول لیست است.



شکل ۵-۳۳. لیست دوطرفه

ملاحظه می‌کنید که با استفاده از متغیر FIRST و فیلد اشاره‌گر FORW، می‌توانیم یک لیست دوطرفه را مانند قبل، از اول تا آخر لیست پیمایش کنیم. از طرف دیگر، با استفاده از متغیر LAST و فیلد اشاره‌گر BACK نیز می‌توانیم این لیست را از آخر تا اول لیست پیمایش کنیم.

فرض کنید LOCA و LOCB به ترتیب مکانهای دو گره A و B در یک لیست دوطرفه باشند. آنگاه

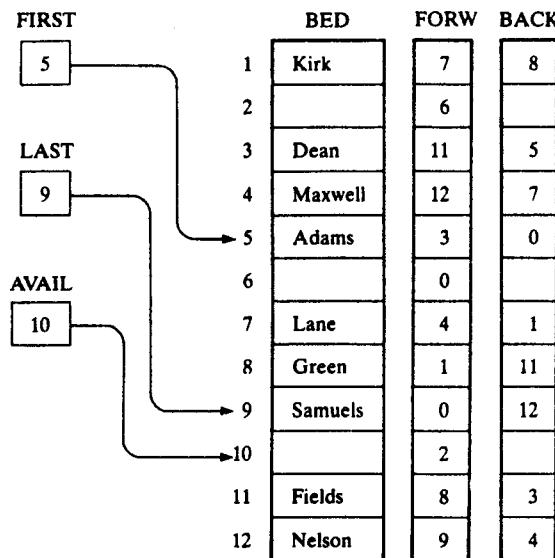
روشی که اشاره‌گرهای FORW و BACK تعریف می‌شوند متنهی به نتایج زیر می‌شود:

$$\text{BACK[LOCB]} = \text{LOCA} \quad \text{اگر و فقط اگر } \text{FORW[LOCA]} = \text{LOCB}$$

از طرف دیگر این بیان که B بعد از گره A است با این بیان که گره A قبل از گره B قراردارد معادل است. لیستهای دوطرفه را می‌توان به وسیله آرایه‌های خطی به جای یک آرایه اشاره‌گر LINK به دو آرایه بیان شد در حافظه نگهداری کرد با این تفاوت که اکنون به جای یک آرایه اشاره‌گر START به دو متغیر اشاره‌گر اشاره‌گر FORW و BACK نیاز داریم و به جای یک متغیر اشاره‌گر لیست LAST به دو متغیر اشاره‌گر لیست FIRST و LAST نیاز داریم. از طرف دیگر از آنجاکه عمل حذف و اضافه کردن گره‌ها تنها در ابتدای لیست AVAIL انجام می‌شود لیست AVAIL از حافظه موجود در آرایه‌ها نیز همانند لیست يکطرفة، با استفاده از FORW به عنوان فیلد اشاره‌گر نگهداری می‌شود.

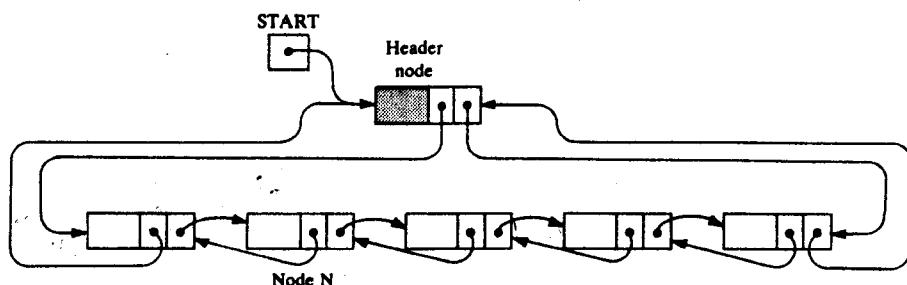
مثال ۵-۲۱

مجدداً داده‌های شکل ۵-۹ یعنی ۹ بیمار یک اطاق بیمارستان با ۱۲ تخت را در نظر بگیرید. شکل ۵-۴ چگونگی سازماندهی لیست الفایی بیماران را در لیست دوطرفه نشان می‌دهد. ملاحظه می‌کنید که مقادیر FIRST و فیلد اشاره‌گر FORW به ترتیب دارای همان مقدار START و آرایه LINK هستند. از این رو لیست را می‌توان همانند قبل به صورت الفایی پیمایش کرد. از طرف دیگر با استفاده از LAST و آرایه اشاره‌گر BACK، لیست را می‌توان به ترتیب الفایی عکس پیمایش کرد. یعنی LAST به Samuels و فیلد اشاره‌گر BACK از Nelson، فیلد اشاره‌گر BACK از Maxwell Nelson به Nelson و الی آخر اشاره می‌کند.



شکل ۳۴-۵

لیستهای دوطرفه دارای سرلیست
مزایا و ویژگیهای خوب یک لیست دوطرفه و یک لیست چرخشی دارای سرلیست در یک لیست چرخشی دوطرفه دارای سرلیست، به صورتی که در شکل ۳۵-۵ نشان داده شده، جمع شده است. این لیست چرخشی است زیرا دو گره انتهایی آن به سرگره اول لیست اشاره می‌کنند.

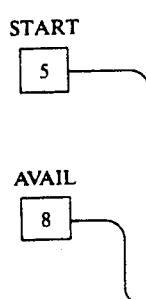


شکل ۳۵-۵ لیست چرخشی دوطرفه با سرلیست

ملاحظه می‌کنید چنین لیست دو طرفه‌ای نیازمند تنها یک متغیر اشاره‌گر لیست START است که به سرگره اشاره می‌کند. زیرا دو اشاره‌گر سرگره به دو انتهای لیست اشاره می‌کنند.

مثال ۵-۲۲

فایل پرستنی شکل ۵-۳۰ را درنظر بگیرید که به صورت یک لیست چرخشی با سرلیست سازماندهی می‌شود. داده‌ها را می‌توان در یک لیست چرخشی دارای سرلیست تنها با افزودن یک آرایه دیگر BACK که مکان گره‌های قبلی را به دست می‌دهد سازماندهی کرد. چنین ساختمانی در شکل ۵-۳۶ به تصویر درآمده است که LINK به صورت FORW نام‌گذاری مجدد شده است. بار دیگر لیست AVAIL تنها به صورت یک لیست یکطرفه نگهداری می‌شود.



	NAME	SSN	SEX	SALARY	FORW	BACK
1					0	
2	Davis	192-36-7282	Female	22 800	12	9
3	Kelly	165-64-3351	Male	19 000	7	14
4	Green	175-56-2251	Male	27 200	14	12
5					191 600	
6	Brown	178-52-1065	Female	14 700	9	5
7	Lewis	181-58-9939	Female	16 400	10	3
8					11	
9	Cohen	177-44-4557	Male	19 000	2	6
10	Rubin	135-46-6262	Female	15 500	5	7
11					13	
12	Evans	168-56-8113	Male	34 200	4	2
13					1	
14	Harris	208-56-1654	Female	22 800	3	4

شکل ۵-۳۶

عملیات بر روی لیستهای دوطرفه

فرض کنید LIST یک لیست دوطرفه در حافظه باشد. در این قسمت عملیات پیمایش، جستجو، حذف و اضافه کردن عنصر روی LIST شرح داده می‌شود.

پیمایش : فرض کنید بخواهیم لیست LIST را برای پردازش دقیقاً یکبار پیمایش کنیم. در آن صورت اگر LIST یک لیست دو طرفه معمولی باشد می‌توانیم از الگوریتم ۱-۵ استفاده کنیم یا اگر LIST یک سرلیست داشته باشد می‌توانیم از الگوریتم ۱۱-۵ استفاده کنیم. در اینجا هیچ مزیتی بین سازماندهی داده‌ها در لیست دو طرفه نسبت به لیست یکطرفه وجود ندارد.

جستجو کردن : فرض کنید اطلاعات ITEM، یک مقدار کلیدی داده شده است و بخواهیم LOC مکان اطلاعات ITEM را در لیست LIST تعیین کنیم. در آن صورت اگر LIST یک لیست دو طرفه معمولی باشد می‌توانیم از الگوریتم ۲-۵ استفاده کنیم یا اگر لیست LIST یک سرلیست داشته باشد می‌توانیم از الگوریتم ۱۲-۵ استفاده کنیم. در اینجا اگر این احتمال وجود داشته باشد که ITEM نزدیک انتهای لیست است در آن صورت مزیت اصلی آن این است که می‌توانیم جستجو برای ITEM را از انتهای لیست به ابتدای لیست انجام دهیم. برای مثال فرض کنید LIST لیستی از نامهای افراد باشد که به صورت الفابیک مرتب شده است. اگر ITEM = Smith آنگاه باید عمل جستجو را از انتهای لیست به طرف ابتدای لیست انجام دهیم اما اگر ITEM = Davis آنگاه باید جستجو در LIST از ابتدا لیست به طرف انتهای لیست انجام شود.

حذف کردن : فرض کنید LOC مکان گره N در لیست LIST داده شده است و فرض کنید بخواهیم N را از لیست حذف کنیم. فرض می‌شود که LIST یک لیست چرخشی دو طرفه با سرلیست است. توجه دارید که [FORW[LOC]] و BACK[LOC] به ترتیب مکان گره‌هایی هستند که قبل و بعد از گره N قرار دارند. بنابر این همانگونه که در شکل ۳۷-۵ نشان داده شده است N با تغییر جفت اشاره‌گرهای زیر از لیست حذف می‌شود:

BACK[FORW[LOC]] := BACK[LOC] و FORW[BACK[LOC]] := FORW[LOC]

آنگاه گره حذف شده N با دستورهای جایگزینی زیر به لیست AVAIL برگردانده می‌شود:

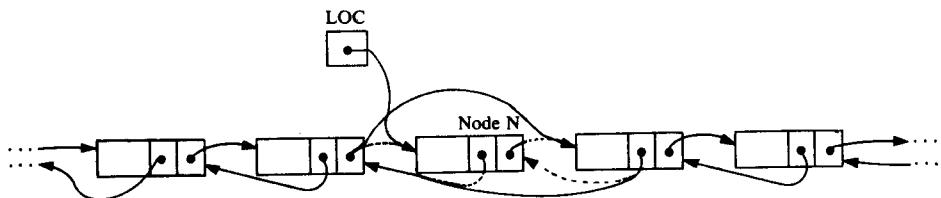
AVAIL := LOC و FORW[LOC] := AVAIL

بیان رسمی این الگوریتم به صورت زیر است:

Algorithm 5.15: DELTWL(INFO, FORW, BACK, START, AVAIL, LOC)

1. [Delete node.]
Set FORW[BACK[LOC]] := FORW[LOC] and
BACK[FORW[LOC]] := BACK[LOC].
2. [Return node to AVAIL list.]
Set FORW[LOC] := AVAIL and AVAIL := LOC.
3. Exit.

در اینجا یک مزیت عمده لیست دو طرفه را مشاهده می کنید. اگر داده ها به صورت یک لیست یک طرفه سازماندهی شوند آنگاه برای حذف N باید لیست یک طرفه را پیمایش کنیم تا مکان گره N قبل از N پیدا شود.

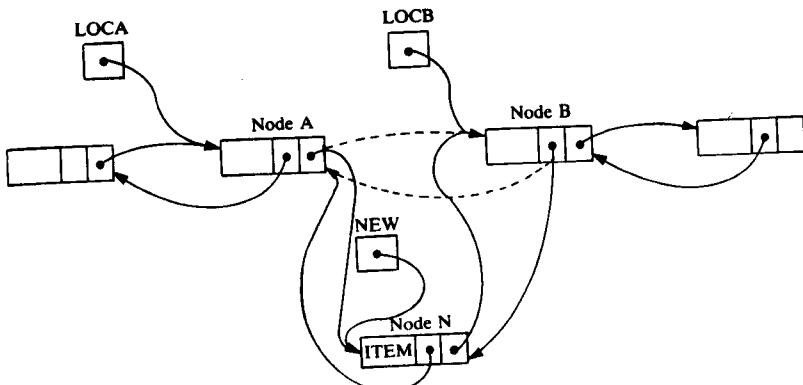
شکل ۳۷-۵ حذف گره N

اضافه کردن: فرض کنید مکانهای $LOCA$ و $LOCB$ دو گره مجاور A و B در لیست LIST داده شده است و فرض کنید بخواهیم اطلاعات معلوم ITEM را بین گره های A و B اضافه کنیم. همانگونه که با یک لیست یک طرفه عمل کردیم، نخست گره اول N را از لیست AVAIL حذف می کنیم، از متغیر NEW برای نگهداری مکان یا آدرس آن استفاده می کنیم و آنگاه داده ITEM را در گره N کپی می کنیم. به عبارت دیگر قرار می دهیم:

$$NEW := AVAIL, \quad AVAIL := FORW[AVAIL], \quad INFO[NEW] := ITEM$$

حال همانگونه که در شکل ۳۸-۵ نشان داده شده است، گره N با محتوای ITEM با تغییر چهار اشاره گر زیر در لیست اضافه می شود:

$$FORW[LOCA] := NEW, \quad FORW[NEW] := LOCB \\ BACK[LOCB] := NEW, \quad BACK[NEW] := LOCA$$

شکل ۳۸-۵ اضافه کردن گره N

بيان رسمي اين الگوريتم به صورت زير است :

Algorithm 5.16: INSTWL(INFO, FORW, BACK, START, AVAIL, LOCA, LOCB, ITEM)

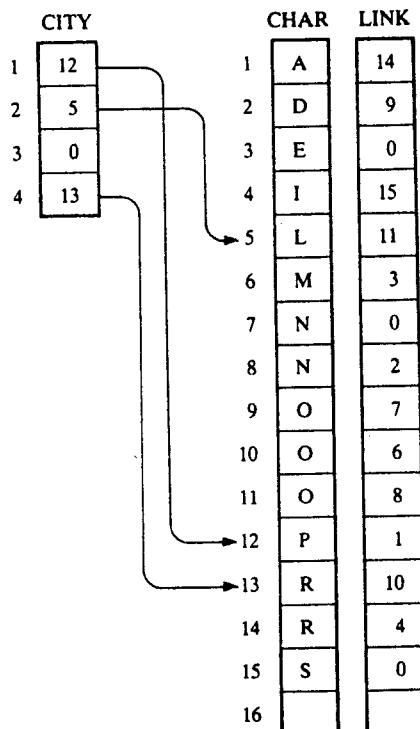
1. [OVERFLOW?] If AVAIL = NULL, then: Write: OVERFLOW; and Exit.
2. [Remove node from AVAIL list and copy new data into node.]
Set NEW := AVAIL, AVAIL := FORW[AVAIL], INFO[NEW] := ITEM.
3. [Insert node into list.]
Set FORW[LOCA] := NEW, FORW[NEW] := LOCB,
BACK[LOCB] := NEW, BACK[NEW] := LOCA.
4. Exit.

الگوريتم 5.16 فرض می‌کند که LIST شامل یک سرگره است. از این رو LOCB یا LOCA می‌توانند به سرگره اشاره کنند که در آن حالت N به عنوان گره اول یا گره آخر اضافه می‌شود. اگر LIST سرگره نداشته باشد آنگاه باید حالتی را درنظر بگیریم که LOCA = NULL و N به عنوان گره اول در لیست اضافه می‌شود و حالتی را درنظر بگیریم که LOCB = NULL و N به عنوان گره آخر در لیست اضافه می‌شود. توجه کنید : در حالت کلی ذخیره داده‌ها به صورت یک لیست دوطرفه، که نیازمند حافظه اضافی برای اشاره گرهای پیمایش از انتهای لیست و زمان اضافی برای تغییر در اشاره گرهای اضافه شده، است که در مقایسه با لیست یکطرفه قابل توجه نیست و هزینه زیادی دربر ندارد مگر آن که خواسته باشیم مکان گرهای را پیدا کنیم که قبل از گره معلوم N است مانند حالتی که هنگام عمل حذف در بالا انجام داده‌ایم.

مسأله‌های حل شده

لیستهای پیوندی

مسئله ۱ - ۵ : رشته‌های کاراکتری ذخیره شده در چهار لیست پیوندی شکل ۵-۳۹ را تعیین کنید.
حل : در اینجا اشاره گر چهار لیست در آرایه CITY دیده می‌شود. با [1] شروع می‌کنیم با تعقیب اشاره گرهای لیست را پیمایش می‌کنیم که درنتیجه آن رشته PARIS بدست می‌آید. با شروع از [2] و [3] پیمایش لیست رشته LONDON به دست می‌آید. چون NULL در [3] دیده می‌شود از این رو لیست سوم خالی است و آن را با Δ رشته تهی نمایش می‌دهند. با شروع از [4] و پیمایش لیست رشته ROME به دست می‌آید. به عبارت دیگر LONDON, PARIS, Δ , ROME چهار رشته مورد نظر هستند.



شکل ۵-۳۹

مسئله ۲-۵: لیست نامهای زیر (به ترتیب) در آرایه خطی INFO جایگزین می‌شوند.

یعنی **START** طوری جاگیر کنید که **INFO, LINK** تشکیل لیست الفبایی از نامها را بدهد.
حال: لیست الفبای نامها تند از:

Audrey, Barbara, Diana, Eileen, Helen, June, Karen, Mary, Nancy, Paula, Ruth, Sandra

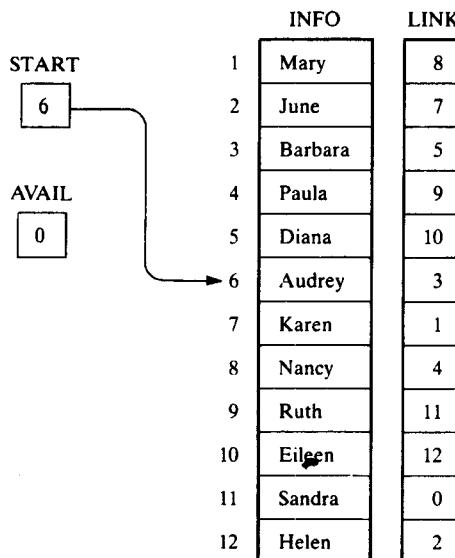
مقادیر START و LINK به صورت زیر بدهست می‌آیند:

(الف) **START =** Audrey **INFO[6] =** بنابراین در 6 جایگزین کند.

(ب) $\text{INFO[3]} = \text{Barbara}$ جایگزین کنید.

(ج) $\text{INFO[5]} = \text{Diana}$ جایگزین کنند.

(د) $\text{INFO}[10] = \text{Eileen}$ ، بنابراین در ۱۰ $\text{LINK}[5] = \text{LINK}[6]$ جایگزین کنید.
 و الى آخر، چون $\text{INFO}[11] = \text{Sandra}$ آخرین نام است $\text{LINK}[11] = \text{NULL}$ جایگزین کنید. شکل ۴۰-۵ ساختمان داده‌ای را نشان می‌دهد که در آن فرض شده است INFO برای تنها ۱۲ عنصر حافظه در اختیار دارد. قرار می‌دهیم $\text{AVAIL} = \text{NULL}$.



شکل ۴۰-۵

مسئله ۳-۵: فرض کنید LIST یک لیست پیوندی در حافظه باشد. یک زیربرنامه Procedure بتوسیید که:
 (الف) تعداد NUM دفعاتی را که ITEM داده شده در LIST ظاهر شده است پیدا کند.

(ب) تعداد NUM عناصر غیر صفر را در LIST پیدا کنید.

(ج) کلید معلوم K را به هر عنصر لیست LIST اضافه کنید.

حل: هر زیربرنامه Procedure برای پیمایش این لیست از الگوریتم ۵.۱ استفاده می‌کند.

Procedure P5.3A: 1. Set NUM := 0. [Initializes counter.]
 2. Call Algorithm 5.1, replacing the processing step by:
 If $\text{INFO}[\text{PTR}] = \text{ITEM}$, then: Set NUM := NUM + 1.
 3. Return
 (الف)

Procedure P5.3B: 1. Set NUM := 0. [Initializes counter.]
 2. Call Algorithm 5.1, replacing the processing step by:
 If $\text{INFO}[\text{PTR}] \neq 0$, then: Set NUM := NUM + 1.
 3. Return
 (ب)

Procedure P5.3C: 1. Call Algorithm 5.1, replacing the processing step by:

Set INFO[PTR] := INFO[PTR] + K.

2. Return.

(ج)

مسئله ۴-۵: لیست الفبایی بیماران شکل ۹-۵ را در نظر بگیرید. تغییرات موردنیاز در این ساختمان داده را تعیین کنید تا (الف) Walters به لیست اضافه شود و آنگاه (ب) Kirk از لیست حذف شود.

حل: (الف) چنانچه می‌کنید که Walters در تخت ۱۰ خوابانده می‌شود که اولین تخت خالی است و پس از Samuels به لیست اضافه می‌شود که آخرین بیمار لیست است. سه تغییر در فیلد اشاره‌گرها به صورت زیر است:

-۱ LINK[9] = ۱۰ [اکنون Walters به Samuels اشاره می‌کند].

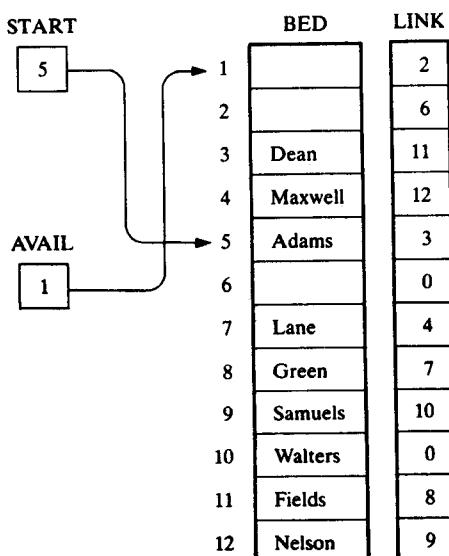
-۲ LINK[10] = ۰ [اکنون Walters آخرین بیمار در لیست است].

-۳ AVAIL = ۲ [اکنون AVAIL به تخت بعدی خالی اشاره می‌کند].

(ب) از آنجاکه Kirk مخصوص می‌شود، [BED[۱] اکنون خالی است. سه تغییر زیر در فیلد اشاره‌گرها باید داده شود:

$$\text{LINK[8]} = ۷ \quad \text{LINK[1]} = ۲ \quad \text{AVAIL} = ۱$$

بنابراین تغییر اول، Green، که در آغاز قبل از Kirk بود اکنون به Lane اشاره می‌کند که در آغاز بعد از Kirk قرار داشت. تغییر دوم و سوم تخت خالی جدید را به لیست AVAIL اضافه می‌کند. تأکید می‌کنیم که قبل از انجام عمل حذف باید گره [BED[۸] را پیدا کنیم که در آغاز به گره حذف شده [BED[۱] اشاره می‌کرد. شکل ۴-۵ ساختمان داده جدید را نشان می‌دهد.



شکل ۴-۵

مسئله ۵-۵: فرض کنید LIST در حافظه باشد. الگوریتمی بنویسید که گره آخر را از لیست LIST حذف کند.

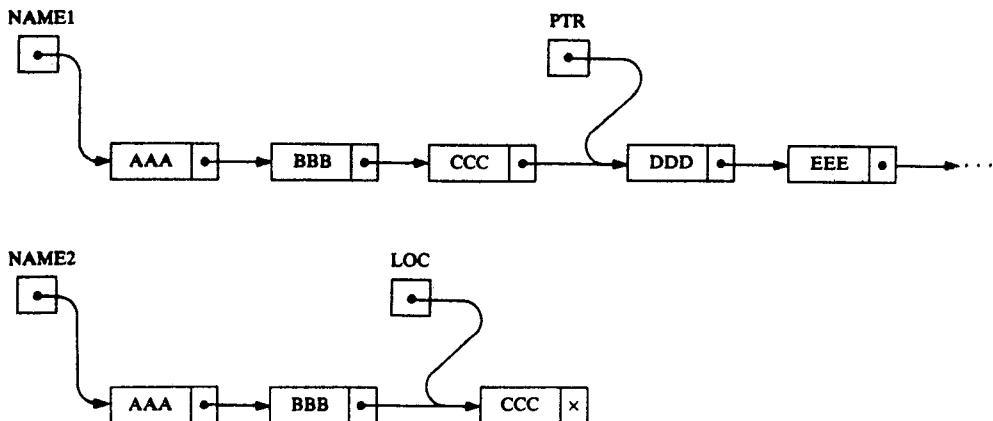
حل: گره آخر را تنها وقتی می‌توان حذف کرد که مکان عنصر قبل از گره آخر را بدانیم. بنابراین با استفاده از یک متغیر اشاره گر PTR لیست را پیمایش کنید و با استفاده از متغیر اشاره گر SAVE گره قبل از آن را نگهدارید. وقتی $PTR = \text{LINK[PTR]}$ به گره آخر اشاره می‌کند و در چنین حالتی SAVE به عنصر قبل از گره آخر اشاره می‌کند. حالتی را که LIST تنها یک گره دارد مستقلأً بررسی کنید چون SAVE تنها در صورتی قابل تعریف است که لیست ۲ یا چند عنصر داشته باشد. الگوریتم به صورت زیر است:

Algorithm P5.5: DELLST(INFO, LINK, START, AVAIL)

1. [List empty?] If START = NULL, then Write: UNDERFLOW, and Exit.
2. [List contains only one element?]
 - If $\text{LINK[START]} = \text{NULL}$, then:
 - (a) Set $\text{START} := \text{NULL}$. [Removes only node from list.]
 - (b) Set $\text{LINK[START]} := \text{AVAIL}$ and $\text{AVAIL} := \text{START}$. [Returns node to AVAIL list.]
 - (c) Exit.
 - [End of If structure.]
3. Set $\text{PTR} := \text{LINK[START]}$ and $\text{SAVE} := \text{START}$. [Initializes pointers.]
4. Repeat while $\text{LINK[PTR]} \neq \text{NULL}$. [Traverses list, seeking last node.]
 - Set $\text{SAVE} := \text{PTR}$ and $\text{PTR} := \text{LINK[PTR]}$. [Updates SAVE and PTR.]
 - [End of loop.]
5. Set $\text{LINK[SAVE]} := \text{LINK[PTR]}$. [Removes last node.]
6. Set $\text{LINK[PTR]} := \text{AVAIL}$ and $\text{AVAIL} := \text{PTR}$. [Returns node to AVAIL list.]
7. Exit.

مسئله ۶-۵: فرض کنید NAME1 یک لیست در حافظه است. الگوریتمی بنویسید که NAME1 را در لیست NAME2 کپی کند.

حل: نخست قرار دهید $NAME2 := \text{NULL}$ تا لیست خالی تشکیل شود. آنگاه با استفاده از یک متغیر اشاره گر PTR، NAME1 را پیمایش کنید و هنگام ملاقات گره آن $NAME1_{X\in E^{\text{SET}}}$ ، محتوای آن INFO[PTR] را در یک گره جدید کپی کنید که در آن صورت به انتهای NAME2 اضافه می‌شود. از LOC برای نگهداری گره آخر NAME2 در خلال پیمایش استفاده کنید. شکل ۴-۴۲، PTR و LOC را قبل از اضافه شدن گره چهارم به NAME2 نشان می‌دهد.



شکل ۵-۴۲

اضافه کردن گره اول به NAME2 باید جداگانه بررسی شود چون LOC هنگامی که دست کم یک گره دارد، تعریف نمی شود. این الگوریتم به صورت زیر است:

Algorithm P5.6: COPY(INFO, LINK, NAME1, NAME2, AVAIL)

This algorithm makes a copy of a list NAME1 using NAME2 as the list pointer variable of the new list.

1. Set NAME2 := NULL. [Forms empty list.]
2. [NAME1 empty?] If NAME1 = NULL, then: Exit.
3. [Insert first node of NAME1 into NAME2.]
Call INSLOC(INFO, LINK, NAME2, AVAIL, NULL, INFO[NAME1]) or:
 - (a) If AVAIL = NULL, then: Write: OVERFLOW, and Exit.
 - (b) Set NEW := AVAIL and AVAIL := LINK[AVAIL]. [Removes first node from AVAIL list.]
 - (c) Set INFO[NEW] := INFO[NAME1]. [Copies data into new node.]
 - (d) [Insert new node as first node in NAME2.]
Set LINK[NEW] := NAME2, and NAME2 := NEW.
4. [Initializes pointers PTR and LOC.]
Set PTR := LINK[NAME1] and LOC := NAME2.
5. Repeat Steps 6 and 7 while PTR ≠ NULL:
 - (a) Call INSLOC(INFO, LINK, NAME2, AVAIL, LOC, INFO[PTR]) or:
 - (i) If AVAIL = NULL, then: Write: OVERFLOW, and Exit.
 - (ii) Set NEW := AVAIL and AVAIL := LINK[AVAIL].
 - (iii) Set INFO[NEW] := INFO[PTR]. [Copies data into new node.]
 - (iv) [Insert new node into NAME2 after the node with location LOC.]
Set LINK[NEW] := LINK[LOC], and LINK[LOC] := NEW.
 - (b) Set PTR := LINK[PTR] and LOC := LINK[LOC]. [Updates PTR and LOC.]
6. [End of Step 5 loop.]
7. Exit.

لیستهای دارای سر لیست، لیستهای دو طرفه

مسئله ۷-۵: لیست‌های (چرخشی) دارای سر لیست از لیستهای یک طرفه شکل ۱۱-۵ بسازید.

حل: [1] TEST را به عنوان سرگره برای لیست ALG و [16] TEST را به عنوان سرگره برای لیست GEOM انتخاب کنید. آنگاه برای هر لیست:

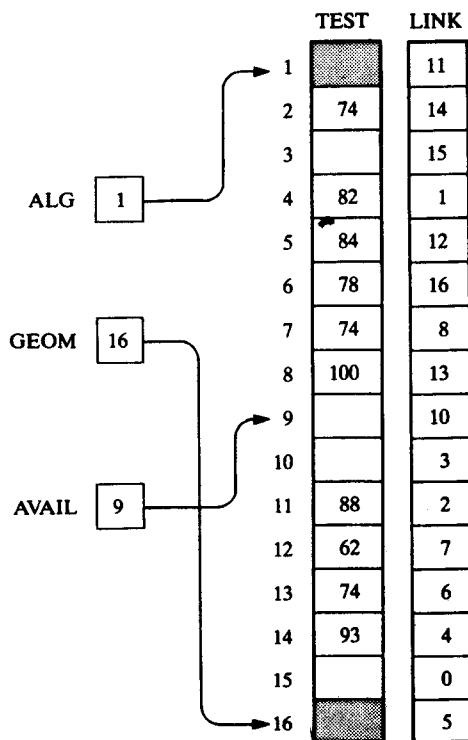
(الف) متغیر اشاره‌گر لیست را طوری تغییر دهید که به سرگره اشاره کند.

(ب) سرگره را طوری تغییر دهید که به گره اول لیست اشاره کند.

(ج) گره آخر را طوری تغییر دهید که از آخر به سرگره اشاره کند.

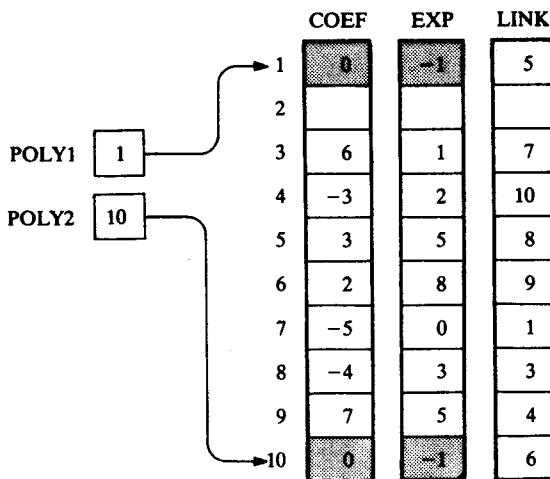
در پایان، لیست AVAIL را مجدداً سازماندهی کنید. شکل ۴۳-۵ ساختمان داده تازه شده را نشان

می‌دهد.



شکل ۴۳-۵

مسئله ۸-۵: چند جمله‌ای POLY1 و POLY2 ذخیره شده در شکل ۴۴-۵ را پیدا کنید.



شکل ۴۴-۵

حل: کار را با POLY1 شروع می‌کنیم. با تعقیب اشاره‌گرها، لیست را پیمایش می‌کنیم تا چند جمله‌ای به دست آید.

$$p_1(x) = 3x^5 - 4x^3 + 6x - 5$$

با شروع از **POLY2**، و تعقیب اشاره‌گرها، لیست را پیمایش می‌کنیم تا چند جمله‌ای

$$p_2(x) = 2x^8 + 7x^5 - 3x^2$$

به دست آید. در اینجا K و $\text{COEF}[K]$ به ترتیب حاوی ضریب و توان یک جمله چندجمله‌ای است. ملاحظه می‌کنید که در فیلد EXP سرگرهای با مقدار -1 -جایگزین شده است.

مسئله ۹ - ۵: چندجمله‌ای $p(x, y, z)$ را بر حسب سه متغیر y, x و z در نظر بگیرید، جملات چندجمله‌ای $p(x, y, z)$ به ترتیب حروف الفبا در کنار هم قرار می‌گیرند مگر آن که خلاف آن بیان شود. یعنی نخست جملات را بر حسب درجه‌های نزولی x مرتب می‌کنیم و چنانچه جملاتی دارای درجه x برابر باشند این جملات را بر حسب درجه‌های نزولی y مرتب می‌کنیم و چنانچه جملاتی با درجه x برابر باشند این

جملات را بر حسب درجه های نزولی z مرتب می کنیم. فرض کنید:

$$p(x, y, z) = 8x^2y^2z - 6yz^8 + 3x^3yz + 2xy^7z - 5x^2y^3 - 4xy^7z^3$$

(الف) چند جمله‌ای را طوری بازنویسی کنید که جملات آن مرتب شده باشند.

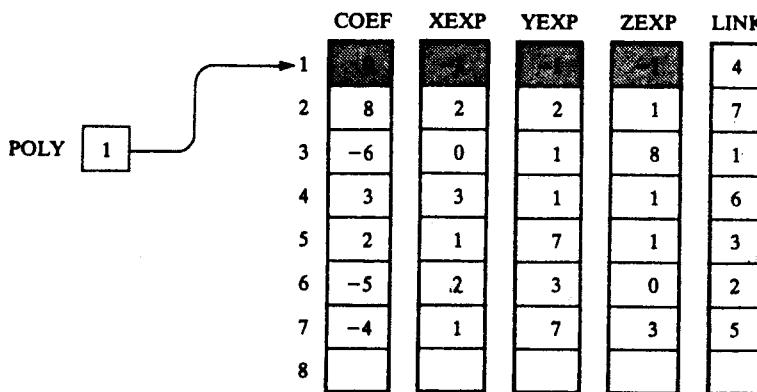
(ب) فرض کنید جملات با ترتیب گفته شده در مسأله در آرایه‌های خطی **COEF**، **XEXP** و **YEXP** با گرمه اول **HEAD** ذخیره شده‌اند. مقادیر را در **LINK** طوری جایگزین کنید که لیست پیوندی شامل دنباله جملات چندجمله‌ای با ترتیب ذکر شده باشد.

شامل دنباله جملات چندجمله‌ای با ترتیب ذکر شده باشد.

حل: (الف) توجه دارید که $3x^3yz$ قبل از همه می‌آید چون بالاترین درجه را نسبت به x دارد. توجه دارید که $8x^2y^3 - 5x^2y^2z$ - نسبت به x دارای درجه یکسان هستند اما $5x^2y^3 -$ قبل از $8x^2y^2z$ می‌آید چون درجه y آن بیشتر است. و الى آخره داریم:

$$p(x, y, z) = 3x^3yz - 5x^2y^3 + 8x^2y^2z - 4xy^7z^3 + 2xy^7z - 6yz^8$$

(ب) شکا ۴۵-۵ ساختمان داده موردنظر را نشان می‌دهد.



شکل ۴۵-۵

مسأله ۱۰-۵: در صورت وجود، مزایای یک لیست دو طرفه را نسبت به لیست یک طرفه در هر یک از عملیات‌های زیر بیان کنید.

(الف) با سماش، لست هر گه را بردازش کنند.

د) ابتدا محدوده آن گروه را حذف کند.

کتابی که تا زمان این مقاله ITEM با جستجو کننده

15-20-1 ITEM 2

(ج) دریک پیش مربوط نظر را در اینجا آورده است.

(۲) یک مرد قبل از زیرهای پا میگذرد - ۱۰۰

(ب) مکان گره قبلي موردنیاز است. لیست دو طرفه شامل این اطلاعات است در حالی که با یک لیست یکطرفه باید لیست را پیمایش کنیم.
 (ج) هیچ مزیتی ندارد.

(د) هیچ مزیتی ندارد مگر آن که بدانیم ITEM باید در انتهای لیست ظاهر شود که در آن حالت لیست از انتهای به ابتدای پیمایش می شود. برای مثال اگر عمل جستجو را برای پیدا کردن Walker در یک لیست الفبا بی انجام دهیم می توان سریعتر لیست را از انتهای به ابتدای پیمایش کرد.

(ه) مانند قسمت (ب) لیست دو طرفه کارایی بیشتری دارد.

(و) هیچ مزیتی ندارد.

توجه کنید: در حالت کلی، یک لیست دو طرفه مفیدتر از لیست یکطرفه نیست، مگر در شرایط خاص.
 مسئله ۱۱ - ۵: فرض کنید LIST یک لیست (چرخشی) دارای سر لیست در حافظه باشد. الگوریتمی بنویسید که گره آخر را از LIST حذف کند. این مسئله را با مسئله ۵-۵ مقایسه کنید.

حل: الگوریتم همان الگوریتم P5.5 است با این تفاوت که اکنون می توانیم حالت خاصی را که LIST تنها یک گره دارد حذف کنیم. به بیان دیگر، وقتی LIST خالی نیست بلا فاصله می توانیم SAVE را تعریف کنیم.

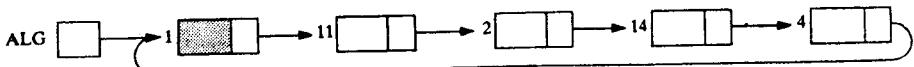
Algorithm P5.11: DELLSTH(INFO, LINK, START, AVAIL)

This algorithm deletes the last node from the header list.

1. [List empty?] If $LINK[START] = \text{NULL}$, then: Write: UNDERFLOW, and Exit.
2. Set $PTR := LINK[START]$ and $SAVE := START$. [Initializes pointers.]
3. Repeat while $LINK[PTR] \neq START$: [Traverses list seeking last node.]
 Set $SAVE := PTR$ and $PTR := LINK[PTR]$. [Updates SAVE and PTR.]
 [End of loop.]
4. Set $LINK[SAVE] := LINK[PTR]$. [Removes last node.]
5. Set $LINK[PTR] := AVAIL$ and $AVAIL := PTR$. [Returns node to AVAIL list.]
6. Exit.

مسئله ۱۲ - ۵: با استفاده از لیستهای یکطرفه دارای سر لیست شکل ۴-۳ لیستهای دو طرفه بسازید.

حل: لیست ALG را از ابتدای انتهای لیست پیمایش کنید تا نتیجه زیر بدست آید:



مانیاز مند اشاره گرهای انتهایی لیست هستیم. این اشاره گرهای گره به گره محاسبه می شوند. برای مثال آخرین گره با مکان $LOC = 4$ باید به گره قبل از گره آخر اشاره کند که دارای مکان $LOC = 14$ است. از این رو

$$\text{BACK}[4] = 14$$

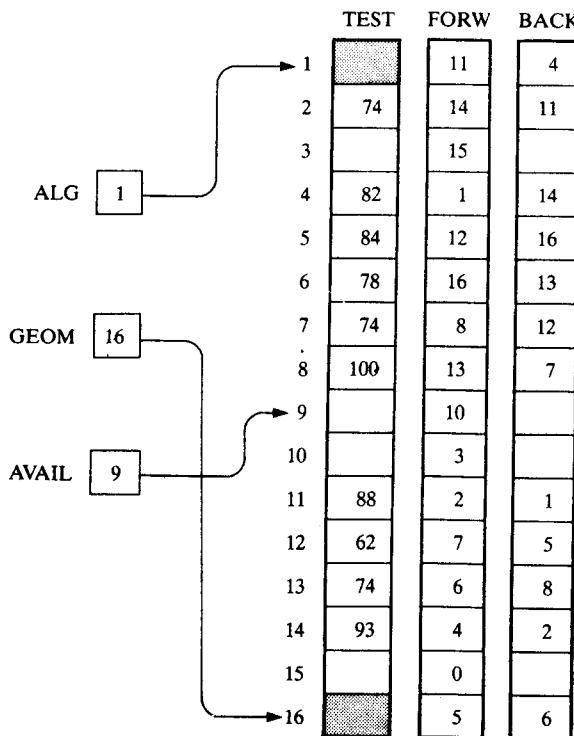
گره قبل از گره آخر با مکان (آدرس) $LOC = 14$ باید به گره قبل از آن با مکان $LOC = 2$ اشاره کند. از این رو

BACK[14] = 2

و الی آخر. سرگره با مکان آدرس **LOC = 1** باید به گره آخر با مکان **4** اشاره کند. از این‌رو

BACK[1] = 4

در مورد لیست **GEOM** روش مشابه‌ای انجام می‌شود. در شکل ۵-۴۶ لیستهای دو طرفه به تصویر درآمده است. توجه دارید که هیچ تفاوتی بین آرایه‌های **LINK** و **FORW** وجود ندارد. به بیان دیگر، تنها محاسبه آرایه **BACK** موردنیاز است.



شکل ۵-۴۶

مسائلهای تکمیلی

لیستهای پیوندی

مسئله ۱۳-۵: شکل ۵-۴۷ لیست بیماران پنج بیمارستان و شماره اتاقهای آنها را نشان می‌دهد.

(الف) مقادیر مربوط به **NLINK** و **NSTART** را به گونه‌ای پر کنید که تشکیل یک لیست الفبایی از نامها بدهد.

(ب) مقادیر مربوط به **RSTART** و **RLINK** را به گونه‌ای پر کنید که شماره اتاقها را به ترتیب ارائه دهد.

	NAME	ROOM	NLINK	RLINK
NSTART <input type="text"/>	1 Brown	650		
RSTART <input type="text"/>	2 Smith	422		
	3 Adams	704		
	4 Jones	462		
	5 Burns	632		

شکل ۵-۴۷

مسئله ۱۴-۵: شکل ۵-۴۸ یک لیست پیوندی را در حافظه نشان می‌دهد.

	INFO	LINK
START <input type="text"/> 4	1 A	2
AVAIL <input type="text"/> 3	2 B	8
	3	6
	4 C	7
	5 D	0
	6	0
	7 E	1
	8 F	5

شکل ۵-۴۸

(الف) دنباله کاراکترهای درون لیست را تعیین کنید.

(ب) فرض کنید **F** و آنگاه **C** از لیست حذف شده‌اند و بدنبال آن **G** به ابتدای لیست اضافه شده است.

ساختمان نهایی لیست را تعیین کنید.

(ج) فرض کنید **C** و آنگاه **F** از لیست حذف شده‌اند و آنگاه **G** به ابتدای لیست اضافه شده است.

ساختمان نهایی لیست را تعیین کنید.

(د) فرض کنید G به ابتدای لیست اضافه شده است و آنگاه F و بدنبال آن C از این ساختمان حذف شده است. ساختمان نهایی لیست را تعیین کنید.

مسئله ۱۵ - ۵: فرض کنید LIST یک لیست پیوندی در حافظه و شامل مقادیر عددی باشد. برای هر یک از حالت‌های زیر یک زیربرنامه Procedure بنویسید که:

(الف) MAX ماتریس مقادیر لیست LIST را پیدا کنید.

(ب) MEAN میانگین مقادیر لیست LIST را پیدا کنید.

(ج) PROD حاصلضرب عناصر لیست LIST را پیدا کنید.

مسئله ۱۶ - ۵: عدد صحیح K داده شده است. یک زیربرنامه Procedure بنویسید که عنصر K ام را از لیست پیوندی حذف کند.

مسئله ۱۷ - ۵: یک زیربرنامه Procedure بنویسید که اطلاعات معلوم ITEM را به انتهای لیست اضافه کند.

مسئله ۱۸ - ۵: یک زیربرنامه Procedure بنویسید که عنصر اول یک لیست را حذف کند و آن را بدون هیچ تغییری در مقدارهای INFO به انتهای لیست اضافه کند. تنها START و LINK قابل تغییر هستند.

مسئله ۱۹ - ۵: یک زیربرنامه Procedure SWAP(INFO, LINK, START, K) بنویسید که بدون هیچ تغییری در مقدارهای INFO جای عناصر K ام و K+1 ام لیست را عوض کند.

مسئله ۲۰ - ۵: یک زیربرنامه Procedure SORT(INFO, LINK, START) بنویسید که بدون هیچ تغییری در مقدارهای INFO لیست را مرتب کند.

راهنمایی: از زیربرنامه Procedure SWAP مسئله ۱۹ - ۵ و مرتب کردن حبابی استفاده کنید.

مسئله ۲۱ - ۵: فرض کنید AAA و BBB لیستهای پیوندی مرتب شده با عناصر متمایز باشند. این دو لیست در INFO و LINK نگهداری می‌شوند. یک زیربرنامه Procedure بنویسید که لیستها را بدون هیچ تغییری در مقادیر CCC تنها در یک لیست پیوندی INFO ترکیب کند.

مسئله‌های ۲۲ - ۵ تا ۲۴ - ۵ به رشتة‌های کاراکتری مربوط می‌شوند که به صورت لیستهای پیوندی ذخیره شده‌اند و در هر گره آن تنها یک کاراکتر وجود دارد و از همان آرایه‌های INFO و LINK استفاده می‌کنند.

مسئله ۲۲ - ۵: فرض کنید STRING یک رشتة کاراکتری در حافظه باشد.

(الف) یک زیربرنامه Procedure بنویسید که SUBSTRING(STRING, K, N) را چاپ کند یعنی زیررشته STRING را که از کاراکتر K ام شروع می‌شود و طول N دارد چاپ کند.

(ب) یک زیربرنامه Procedure بنویسید که یک رشتة جدید SUBKN در حافظه ایجاد کند که در آن

$$\text{SUBKN} = \text{SUBSTRING}(\text{STRING}, \text{K}, \text{N})$$

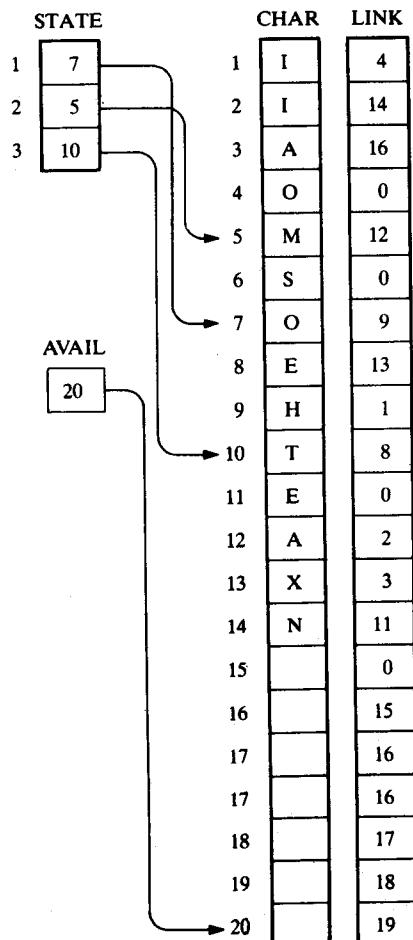
مسئله ۲۳ - ۵: فرض کنید STR1 و STR2 رشتة‌های کاراکتری در حافظه باشند. یک زیربرنامه

Procedure بنویسید که یک رشته جدید **STR3** ایجاد کند که از اتصال دو رشته **STR1** و **STR2** بوجود آمده است.

مسئله ۵-۲۴: فرض کنید **TEXT** و **PATTERN** رشته‌هایی در حافظه باشند. یک زیربرنامه **Procedure** بنویسید که مقدار **INDEX(TEXT, PATTERN)** یعنی مکانی را که در آن **PATTERN** برای اولین بار به صورت یک زیررشته **TEXT** ظاهر می‌شود پیدا کند.

لیستهای دارای سریلیست، لیستهای دو طرفه

مسئله ۵-۲۵: رشته‌های کاراکتری در سه لیست پیوندی طبق شکل ۵-۴۹ ذخیره شده‌اند.



شکل ۵-۴۹

(الف) سه رشته را پیدا کنید. (ب) لیستهای چرخشی دارای سرلیست از لیستهای یکطرفه با استفاده از **CHAR[18]** ، **CHAR[19]** و **CHAR[20]** به عنوان سرگرهای تشكیل دهید.

مسئله ۵-۲۶: چندجمله‌ای‌های ذخیره شده در سه لیست دارای سرلیست شکل ۵-۵۰ را تعیین کنید.

POLY	COEF	EXP	LINK
1	1	1	9
2	2	-1	2
3	3	-1	6
	4	3	7
	5	2	8
	6	3	5
	7	1	10
AVAIL	8	0	3
11	9	4	4
	10	0	1
	11		12
	12		13
	13		14
	:		:
	49		50
	50		0

شکل ۵-۵۰

مسئله ۵-۲۷: چندجمله‌ای زیر را در نظر بگیرید:

$$p(x, y, z) = 2xy^2z^3 + 3x^2yz^2 + 4xy^3z + 5x^2y^2 + 6y^3z + 7x^3z + 8xy^2z^5 + 9$$

(الف) این چندجمله‌ای را طوری بنویسید که جملات آن به ترتیب حروف الفبا باشند.

(ب) فرض کنید جملات این چندجمله‌ای به ترتیب نشان داده شده در اینجا در آرایه‌های مواردی **COEF** ، **XEXP** و **YEXP** که سرگرهای اولین گره است ذخیره شده‌اند. بنابراین به مازای $K = 2, 3, \dots, 9$ داریم. مقادیر را در آرایه **LINK** به گونه‌ای جایگزین کنید که لیست پیوندی شامل دنباله جملات با ترتیب ذکر شده باشد. مسئله ۹-۵ را ببینید.

مسئله ۵-۲۸ : یک زیربرنامه **HEAD(INFO, LINK, START, AVAIL)** بنویسید که از یک لیست یکطرفه معمولی یک لیست چرخشی دارای سرلیست بسازد.

مسئله ۵-۲۹ : مجدداً مسئله‌های ۱۶-۵ تا ۲۰-۵ را به جای یک لیست یکطرفه معمولی با استفاده از یک لیست چرخشی دارای سرلیست حل کنید. ملاحظه می‌کنید که الگوریتم اکنون خیلی ساده‌تر شده است.

مسئله ۵-۳۰ : فرض کنید **POLY1** و **POLY2** دو چندجمله‌ای یک متغیره باشند که با استفاده از همان آرایه‌های موازی **COEF**، **EXP** و **LINK** به صورت لیستهای چرخشی دارای سرلیست ذخیره شده‌اند.

Procedure یک زیربرنامه

ADD(COEF, EXP, LINK, POLY1, POLY2, AVAIL, SUMPOLY)

بنویسید که مجموع **SUMPOLY** دو چندجمله‌ای **POLY1** و **POLY2** را پیدا کند (که با استفاده از **COEF**، **EXP** و **LINK** نیز در حافظه ذخیره می‌شود).

مسئله ۵-۳۱ : برای چندجمله‌ای‌های **POLY1** و **POLY2** مسئله ۳۰-۵ یک زیربرنامه

MULT(COEF, EXP, LINK, POLY1, POLY2, AVAIL, PRODPOLY)

بنویسید که حاصلضرب **PRODPOLY** چندجمله‌ای‌های **POLY1** و **POLY2** را پیدا کند.

مسئله ۵-۳۲ : از لیستهای یکطرفه شکل ۴۹-۵ با استفاده از **CHAR[20]**، **CHAR[19]** و **CHAR[18]** به عنوان سرگرهای همانند مسئله ۲۵-۵، لیستهای چرخشی دوطرفه با سرلیست بسازید.

مسئله ۵-۳۳ : عدد صحیح **K** داده شده است. یک زیربرنامه **Procedure** به صورت

DELK(INFO, FORW, BACK, START, AVAIL, K)

بنویسید که عنصر **K** ام را از لیست چرخشی دوطرفه با سرلیست حذف کند.

مسئله ۵-۳۴ : فرض کنید **LIST(INFO, LINK, START, AVAIL)** لیست چرخشی یکطرفه با سرلیست در حافظه باشد. یک زیربرنامه **Procedure** به صورت

TWOWAY(INFO, LINK, BACK, START)

بنویسید که مقادیر را در آرایه خطی **BACK** جایگزین کند تا از لیست یکطرفه یک لیست دوطرفه بسازد.

برای مسئله‌های زیر برنامه بنویسید

مسئله‌های ۵-۳۵ تا ۴۰-۵ در ارتباط با ساختمندان داده شکل ۵-۵۱ هستند که از چهار لیست الفایی مربوط به موكلين و وکيل مدافعهای آنها تشکيل شده است.

مسئله ۵-۳۵ : برنامه‌ای بنویسید که عدد صحیح **K** را از ورودی بخواند و در خروجی لیست موكلين وکيل مدافع **K** را چاپ کند. برای هر **K** برنامه را آزمایش کنید.

مسئله ۵-۳۶ : برنامه‌ای بنویسید که نام و وکيل مدافع هر موكلى را که سنتش برابر **L** يا بزرگتر از **L** است

	LAWYER	POINT	CLIENT	AGE	LINK
1	Davis	4	Hall	35	16
2	Levine	12	Moss	28	13
3	Nelson	21	Ford	47	25
4	Rogers	8	Brown	54	22
			Ginn	38	14
			Pride	42	29
					26
			Berk	38	3
			White	45	0
10					28
11			Todd	25	0
12			Dixon	32	24
13			Newman	46	6
14			Harris	42	30
					7
16			Jackson	52	27
					23
18			Roberts	40	0
19					0
20			Eisen	32	1
21			Adams	48	5
22			Cohen	36	20
23					19
24			Fisher	33	18
25			Graves	42	11
26					10
27			Parker	50	9
28					17
29			Singer	45	0
30			Lewis	28	2

شكل ۵۱

چاپ کند. برنامه را با استفاده از (الف) $41 = L$ و (ب) $48 = L$ آزمایش کنید.

مسئله ۳۷-۵: برنامهای بنویسید که نام **LIL** یک وکیل مدافع را از ورودی بخواند و در خروجی لیست **Levine** وکلای مدافع موکلین را چاپ کند. برنامه را با استفاده از (الف) **Rogers** ، (ب) **Baker** ، (ج) **Azmaiesh** کنید.

مسئله ۳۸-۵: برنامهای بنویسید که **NAME** یک موکل را از ورودی بخواند و در خروجی نام موکل، سن و وکیل مدافع او را چاپ کند. برنامه را با استفاده از (الف) **Newman** ، (ب) **Ford** ، (ج) **Rivers** و (د) **Hall** آزمایش کنید.

مسئله ۳۹-۵: برنامهای بنویسید که **NAME** موکل را از ورودی بخواند و رکورد موکل را از این ساختمان داده حذف کند. برنامه را با استفاده از (الف) **Lewis** ، (ب) **Klein** و (ج) **Parker** آزمایش کنید.

مسئله ۴۰-۵: برنامهای بنویسید که رکورد یک موکل جدید را که از نام موکل، سن و وکیل مدافع او تشکیل شده است از ورودی بخواند و رکورد را در ساختمان داده اضافه کند. برنامه را با استفاده از (الف) **Olsen** و (ب) **Jones** ، 36 ، **Levine** آزمایش کنید.

مسئلهای ۴۱-۵ تا ۴۶-۵ در ارتباط با لیست الفایی رکورد کارمندان شکل ۳۰-۵ هستند که به صورت لیست چرخشی با سرلیست ذخیره شده‌اند.

مسئله ۴۱-۵: برنامهای بنویسید که در خروجی لیست الفایی رکورد تمام کارمندان را چاپ کند.

مسئله ۴۲-۵: برنامهای بنویسید که نام **NNN** یک کارمند را از ورودی بخواند و در خروجی رکورد کارمند را چاپ کند. برنامه را با استفاده از (الف) **Evans** ، (ب) **Smith** و (ج) **Lewis** آزمایش کنید.

مسئله ۴۳-۵: برنامهای بنویسید که شماره تأمین اجتماعی **SSS** یک کارمند را از ورودی بخواند و رکورد کارمند را چاپ کند. برنامه را با استفاده از (الف) **3351 - 64 - 165** ، (ب) **6262 - 46 - 136** و (ج) **5555 - 44 - 177** آزمایش کنید.

مسئله ۴۴-۵: برنامهای بنویسید که عدد صحیح **K** را از ورودی بخواند و نام هر کارمند مذکور را وقتی $K = 1$ است یا نام هر کارمند مؤنث را وقتی $K = 2$ است در خروجی چاپ کند. برنامه را با استفاده از (الف) **K = 2** ، (ب) **K = 5** و (ج) **K = 1** آزمایش کنید.

مسئله ۴۵-۵: برنامهای بنویسید که نام **NNN** یک کارمند را از ورودی بخواند و رکورد این کارمند را از ساختمان داده حذف کند. برنامه را با استفاده از (الف) **Davis** ، (ب) **Jones** و (ج) **Rubin** آزمایش کنید.

مسئله ۴۶-۵: برنامهای بنویسید که رکورديک کارمند جدید را از ورودی بخواند و رکورد را در فایل **Nelson** ، **Fletcher** ، 168 - 52 - 3388 ، **Female** ، 21 000 و (ب) **Male** ، 19 000 آضافه کند. برنامه را با استفاده از (الف) **175 - 32 - 2468** ، **19 000** آزمایش کنید.

توجه کنید: خاطرنشان می‌کنیم که هنگام اضافه کردن و حذف رکورد، رکورد سرگره را تازه کنید.

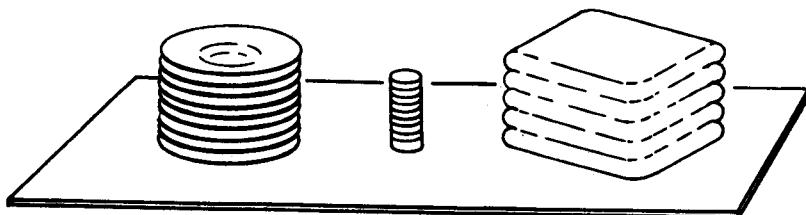
فصل ۶

پشته‌ها، صفحه‌ها، زیربرنامه‌های بازگشتی

۱- مقدمه

لیستها و آرایه‌های خطی که در فصل‌های قبل مورد بررسی قرار گرفته‌اند، اجازه می‌دادند که عناصر را در هر مکانی از لیست، ابتدای لیست، انتهای لیست یا وسط لیست حذف یا اضافه کنیم. در علم کامپیوتر اغلب وضعیتها بی پیش می‌آید که می‌خواهیم در عمل اضافه کردن و حذف عناصر لیست محدودیت‌هایی ایجاد کنیم طوری که این عملیات تنها در ابتدای انتها در ابتدای لیست انجام شود نه در وسط لیست. دو ساختمان داده‌ای که برای این منظور مفید هستند عبارتند از: پشته‌ها و صفحه‌ها.

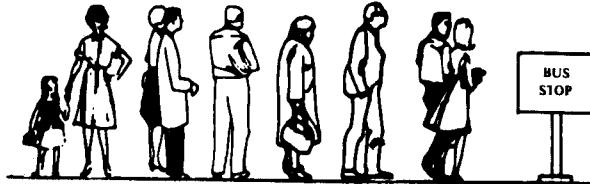
یک پشته، یک ساختمان خطی است که در آن عمل اضافه کردن یا حذف عنصر تنها در یک انتهای آن امکان‌پذیر است. شکل ۱-۶ سه مثال از این ساختمان داده را که در زندگی روزمره اتفاق می‌افتد نشان می‌دهد. یک پشته از بشقاب، یک پشته از سکه و یک پشته از حolle تا شده.



یک پشته از بشقاب یک پشته از سکه یک پشته از حolle تا شده

ملاحظه می‌کنید که یک قلم از عنصر را می‌توان تنها از بالای هر یک از این پشته‌ها حذف کرد یا به بالای آن اضافه کرد. بهویژه این که آخرین عنصر داده‌ای اضافه شده به یک پشته، اولین عنصر داده‌ای است که می‌توان از آن حذف کرد. به این ترتیب به پشته‌ها، لیستهای آخرین ورودی اولین خروجی است، نیز می‌گویند. نامهای دیگری برای پشته‌ها بکار برده می‌شود. که عبارتند از: "انبوه" و "لیستهای LIFO فشردنی". هرچند ممکن است پشته ساختمان داده بسیار محدودی بنظر رسد اما در علم کامپیوتر کاربرد بسیار زیادی دارد.

یک صفت، یک لیست خطی است که در آن هر عنصر داده‌ای را می‌توان تنها از یک انتهای آن اضافه کرد و عنصرهای داده‌ای را می‌توان تنها از انتهای دیگر آن حذف کرد. نام صفت احتمالاً از کاربرد روزمره این اصطلاح گرفته شده است. صفت مردمی را که در یک ایستگاه، به صورتی که در شکل ۲-۶ می‌بینید، منتظر اتوبوس هستند را در نظر بگیرید.



شکل ۲-۶. صفت انتظار اتوبوس

هر شخص جدید که وارد ایستگاه می‌شود در آخر صفت می‌ایستد و وقتی اتوبوس می‌رسد مردمانی که در جلوی صفت هستند اول، سوار اتوبوس می‌شوند. واضح است که اولین نفر داخل صفت اولین کسی است که صفت انتظار را ترک می‌کند. بنابراین به صفت‌ها لیستهای اولین ورودی اولین خروجی است، نیز می‌گویند. مثال دیگری از یک صفت، یک تعداد یا یک بسته از برنامه‌های است که در انتظار پردازش بسر می‌برند. فرض می‌شود که هیچ برنامه‌ای اولویت بالاتری نسبت به دیگری ندارد. مفهوم بازگشتی، از مفاهیم اساسی و بنیادی علم کامپیوتر است. این مبحث در این بخش معرفی می‌شود زیرا به وسیله ساختمان یک پشته می‌توان مفاهیم مربوط به مسایل بازگشتی را شبیه‌سازی کرد.

۲-۶ پشته‌ها

یک پشته، یک لیست از عناصر است که در آن هر عنصر را می‌توان تنها از یک انتهای موسوم به بالای

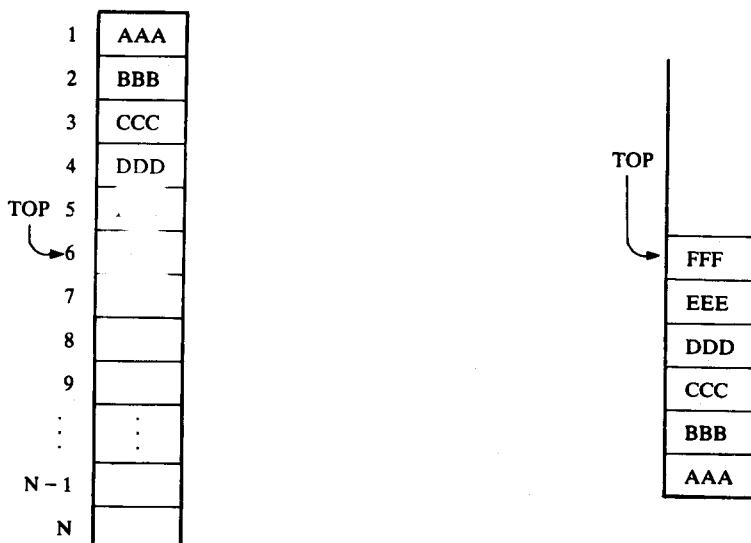
پشته حذف یا اضافه کرد، یعنی عناصر به ترتیب عکسی که وارد پشته می‌شوند از پشته حذف می‌شوند.
دو اصطلاح خاص، برای دو عمل اساسی، با پشته‌ها بکار می‌رود:
(الف) عمل **PUSH** که این اصطلاح برای اضافه کردن یک عنصر در پشته بکار می‌رود.
(ب) عمل **POP** که این اصطلاح برای حذف یک عنصر از پشته بکار می‌رود.
تأکید می‌کنیم که این اصطلاحات تنها هنگام کار با پشته‌ها بکار می‌روند و در هیچ ساختمان داده دیگری، مورد استفاده قرار نمی‌گیرد.

مثال ۱ - ۶

فرض کنید ۶ عنصر زیر به ترتیب در یک پشته خالی **PUSH** می‌شوند:

AAA, BBB, CCC, DDD, EEE, FFF

شکل ۳-۶ سه راه به تصویر کشیده شدن چنین پشته‌ای را نشان می‌دهد.



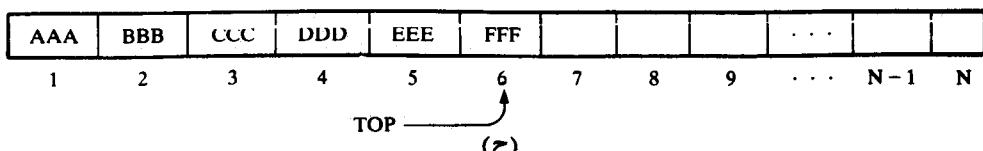
(ب)

(a)

(الف)

(b)

(ج)



شکل ۳-۶. نمودار پشته‌ها

جهت سهولت در نمادگذاری، اغلب پشته را به صورت زیر مشخص می‌کنیم:

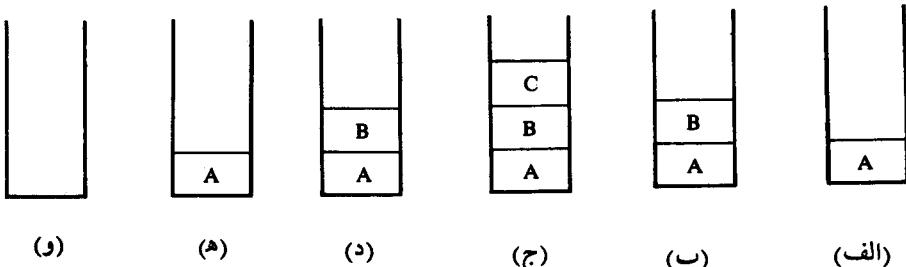
STACK: AAA, BBB, CCC, DDD, EEE, FFF

ضرورت این کار در آن است که سمت راست ترین عنصر، در بالای پشته قرار می‌گیرد. صرفنظر از روشی که یک پشته توصیف می‌شود، تأکید می‌کنیم که خاصیت اساسی آن، که همان عمل اضافه کردن و حذف عنصر است می‌تواند تنها در بالای پشته اتفاق بیفتد. معنی آن این است که قبل از حذف FFF نمی‌توان EEE را حذف کرد و قبل از حذف EEE و FFF نمی‌توان DDD را حذف کرد و الی آخر. درنتیجه عناصر را می‌توان از پشته، تنها به ترتیب عکسی که در پشته PUSH یا اضافه می‌شود، POP یا حذف کرد. بار دیگر لیست گرهای آزاد AVAIL را که در فصل ۵ بررسی کردیم در نظر بگیرید. یادآوری می‌کنیم که گرهای آزاد تنها از ابتدای لیست AVAIL حذف می‌شوند و گرهای جدید موجود تنها در ابتدای لیست AVAIL اضافه می‌شوند. به بیان دیگر لیست AVAIL به صورت یک پشته پیاده‌سازی می‌شود. این روش پیاده‌سازی لیست AVAIL به صورت یک پشته تنها با خاطر سادگی و سهولت آن نسبت به قسمت اصلی این ساختمان است. در قسمت زیر، وضعیت مهم و قابل توجهی را مورد بررسی قرار می‌دهیم که در آن پشته ابزار اساسی پردازش خود الگوریتم است.

تصمیم‌گیری‌های درجه دوم یا به تعویق افتاده

پشته‌ها اغلب برای بیان ترتیب مراحل پردازش‌هایی بکار می‌روند که در آن مراحل، یعنی پردازش باید تا برقراری و محقق شدن شرایط دیگر به تعویق بیفتد. به مثال زیر توجه کنید.

فرض کنید که هنگام پردازش پروژه A نیازمند آن باشیم که روی پروژه B کار کنیم که کامل شدن B مستلزم کامل شدن پروژه A است. آنگاه پوشه‌ای که شامل داده‌های پروژه A است را در پشته قرار می‌دهیم، این وضعیت در شکل ۴-۶ (الف) به تصویر کشیده شده است، همچنین شروع به پردازش B می‌کنیم. با وجود این فرض کنید با همان دلیل هنگام پردازش B منتهی به پردازش پروژه C می‌شویم. آنگاه همانند آنچه که در نمودار ۴-۶ (ب) به تصویر درآوردهیم B را در پشته بالای A قرار می‌دهیم و شروع به پردازش C می‌کنیم. علاوه براین فرض کنید هنگام پردازش C به همین ترتیب منتهی به پردازش D شویم. آنگاه C را در پشته بالای B قرار می‌هیم، این وضعیت در شکل ۴-۶(ج) به تصویر کشیده شده است و همچنین شروع به پردازش D می‌کنیم.



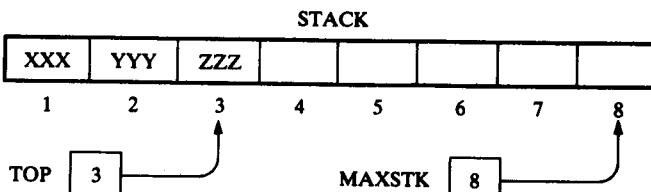
شکا ۶-۴

از طرف دیگر فرض کنید توانایی کامل کردن پردازش پروژه D را داریم. آنگاه تنها پروژه‌ای که می‌توانیم پردازش آن را ادامه دهیم پروژه C است که در بالای پشته است. از این رو پوشه پروژه C را از پشته حذف می‌کنیم، پشته به صورتی که در نمودار ۴-(d) به تصویر کشیده شده است باقی می‌ماند و پردازش C ادامه می‌یابد. به همین ترتیب پس از کامل شدن پردازش C، پوشه B را از پشته حذف می‌کنیم و پشته به صورتی که در نمودار ۴-(e) به تصویر کشیده شده است باقی می‌ماند و پردازش B ادامه می‌یابد. بالاخره پس از کامل شدن پردازش C، آخرین پوشه، A را از پشته حذف می‌کنیم، پشته خالی باقی مانده در نمودار ۴-(f) به تصویر کشیده شده است و پردازش پروژه اصلی ما A ادامه می‌یابد. ملاحظه می‌کنید که در هر مرحله از پردازش بالا، پشته بطور اتوماتیک ترتیبی را نگه می‌دارید که نیازمند کامل کردن پردازش است. یک مثال مهم از چنین پردازشی در علوم کامپیوتر در جایی است که در آن A یک برنامه اصلی است و B، C و D زیربرنامه‌هایی هستند که با ترتیب داده شده فراخوانده می‌شوند.

۳-۶ نمایش پسته‌ها با آرایه

پشته‌ها را می‌توان در کامپیوتر به صورت‌های مختلف، معمولاً به وسیله لیست یک‌طرفه یا آرایه خطی نمایش داد. هر یک از پشته‌های ما، به وسیله یک آرایه خطی **STACK**، یک متغیر اشاره‌گر **TOP**، که حاوی مکان عنصر بالای پشته و یک متغیر **MAXSTK** است که بیشترین تعداد عناصر قابل نگهداری توسط پشته را به دست می‌دهد، نمایش داده می‌شود. اگر منظور ما غیر از این باشد به صورت صریح با ضمنی بیان می‌کنیم. شرط **TOP = 0** یا **TOP = NULL** می‌بین آن است که پشته خالی است.

شکل ۵-۶ چنین نمایشی از پشته را، توسط آرایه نشان می‌دهد. جهت سهولت در نمادگذاری، آرایه را به جای صورت عمودی آن، به صورت افقی رسم کرده‌ایم.



شکل ۵-۶

از آنجاکه $TOP = 3$ ، پشته سه عنصر دارد، XXX و YYY و ZZZ ، چون $8 = MAXSTK$ ، جا برای عنصر در پشته وجود دارد.

عمل اضافه کردن (Push کردن) یک عنصر به درون یک پشته و عمل برداشتن یا حذف کردن (Pop کردن) یک عنصر از یک پشته را می‌توان به ترتیب با زیربرنامه‌های Procedure PUSH و POP پیاده‌سازی کرد. در اجرای زیربرنامه PUSH، نخست باید تحقیق کنیم که آیا جا برای عنصر جدید در پشته وجود دارد یا خیر، اگر جواب منفی بود آنگاه وضعیت موسوم به سرربزی Overflow را داریم. به طور مشابه، در اجرای زیربرنامه POP نخست باید تحقیق کنیم که آیا عنصری در پشته برای حذف وجود دارد یا خیر، اگر جواب منفی است آنگاه وضعیت موسوم به زیرربزی Underflow را داریم.

Procedure 6.1: PUSH(STACK, TOP, MAXSTK, ITEM)
This procedure pushes an ITEM onto a stack.

1. [Stack already filled?] If $TOP = MAXSTK$, then: Print: OVERFLOW, and Return.
2. Set $TOP := TOP + 1$. [Increases TOP by 1.]
3. Set $STACK[TOP] := ITEM$. [Inserts ITEM in new TOP position.]
4. Return.

Procedure 6.2: POP(STACK, TOP, ITEM)

This procedure deletes the top element of STACK and assigns it to the variable ITEM.

1. [Stack has an item to be removed?] If $TOP = 0$, then: Print: UNDERFLOW, and Return.
2. Set $ITEM := STACK[TOP]$. [Assigns TOP element to ITEM.]
3. Set $TOP := TOP - 1$. [Decreases TOP by 1.]
4. Return.

اغلب **TOP** و **MAXSTK** متغیرهای سراسری هستند از این‌دو زیربرنامه‌ها را می‌توان تنها با استفاده از **POP(STACK, ITEM)** و **PUSH(STACK, ITEM)**

به ترتیب فراخواند. خاطرنشان می‌کنیم که مقدار **TOP** قبل از اضافه‌شدن عنصر در **PUSH** تغییر می‌کند اما مقدار **TOP** بعد از حذف‌شدن عنصر در **POP** تغییر می‌کند.

مثال ۲ - ۶

(الف) پشته شکل ۵-۶ را در نظر بگیرید. عمل **PUSH(STACK, WWW)** را به صورت زیر شبیه‌سازی می‌کنیم:

1. Since $TOP = 3$, control is transferred to Step 2.
2. $TOP = 3 + 1 = 4$.
3. $STACK[TOP] = STACK[4] = WWW$.
4. Return.

توجه دارید که **WWW** اکنون عنصر بالای پشته است.

(ب) مجدداً پشته شکل ۵-۶ را در نظر بگیرید. این‌بار عمل **POP(STACK, ITEM)** را به صورت زیر شبیه‌سازی می‌کنیم:

1. Since $TOP = 3$, control is transferred to Step 2.
2. $ITEM = ZZZ$.
3. $TOP = 3 - 1 = 2$.
4. Return.

مالحظه می‌کنید که $STACK[TOP] = STACK[2] = YYY$ اکنون عنصر بالای پشته است.

به حداقل رساندن سرریزی

یک تفاوت اساسی بین زیرریزی و سرریزی در ارتباط با پشته‌ها نمایان می‌شود. زیرریزی به میزان زیادی به الگوریتم داده شده و داده ورودی بستگی دارد و از این‌دو برنامه‌نویس هیچ کنترل مستقیمی بر آن ندارد. از طرف دیگر، سرریزی بستگی به انتخاب برنامه‌نویس برای مقدار حافظه‌ای دارد که برای هر پشته ذخیره می‌کند، همچنین این انتخاب تعداد دفعات و قوع سرریزی را تحت الشعاع خود قرار می‌دهد. در حالت کلی، تعداد عناصر یک پشته با اضافه‌شدن یا کم‌شدن عناصر تغییر می‌کند. بنابراین، انتخاب ویژه مقدار حافظه برای یک پشته داده شده، مستلزم توازن بین زمان و حافظه است. بهویژه این که، در ابتداء ذخیره مقدار زیاد حافظه برای هر پشته تعداد دفعات و قوع سرریزی را کاهش می‌دهد. با وجود این که در اکثر کارها بمندرت از حافظه زیاد استفاده می‌شود، مصرف حافظه زیاد برای جلوگیری از

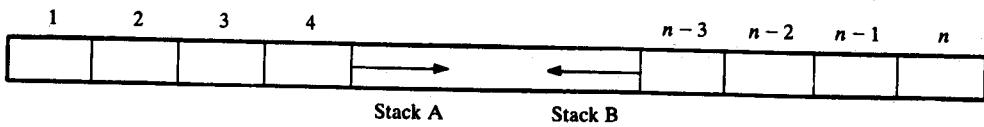
مسئله سرریزی پر هزینه خواهد بود و زمان مورد نیاز برای حل مسئله سرریزی، نظیر اضافه کردن حافظه به پشته می‌تواند پر هزینه‌تر از حافظه ذخیره شده باشد.

روشهای متعددی وجود دارد که نمایش آرایه‌های پشته‌ها را به گونه‌ای اصلاح می‌کند که مقدار فضای ذخیره شده برای بیش از یک پشته را می‌تواند با کارآیی بیشتری مورد استفاده قرار دهد. اغلب این روشها خارج از حدود این درس است. یک نمونه از چنین روشی در مثال زیر بیان شده است.

مثال ۳-۶

فرض کنید یک الگوریتم داده شده به دو پشته A و B احتیاج دارد. برای پشته A یک آرایه STACKA با n_1 عنصر و برای پشته B یک آرایه STACKB با n_2 عنصر می‌توان تعریف کرد. سرریزی وقتی اتفاق می‌افتد که یا پشته A شامل بیش از n_1 عنصر باشد یا پشته B بیش از n_2 عنصر داشته باشد.

فرض کنید بجای این که یک آرایه STACK با $n = n_1 + n_2$ عنصر برای پشته‌های A و B تعریف کنیم نظیر آنچه که در شکل ۳-۶ به تصویر درآمده است، [1] STACK[n] را به صورت پائین پشته A تعریف کنیم و به A اجازه دهیم به طرف راست رشد کند و [n] STACK[n] را به صورت پائین پشته B تعریف کنیم و به B اجازه دهیم به طرف چپ رشد کند. در این حالت، سرریزی تنها وقتی اتفاق می‌افتد که A و B بیش از $n = n_1 + n_2$ عنصر داشته باشند. این روش معمولاً تعداد دفعات وقوع سرریزی را کاهش می‌دهد حتی اگر ما تعداد کل فضای ذخیره شده برای دو پشته را افزایش ندهیم. در استفاده از این ساختمن داده عملیات PUSH و POP لازم است اصلاح شوند.



شکل ۳-۶

۴-۶ عبارتهای محاسباتی؛ نمادگذاری لهستانی

فرض کنید Q یک عبارت محاسباتی شامل ثابت‌ها و عملیات ریاضی باشد. این بخش الگوریتمی را ارائه می‌دهد که مقدار Q را با استفاده از نمادگذاری لهستانی معکوس یا نمادگذاری پسوندی پیدا می‌کند. ملاحظه خواهید کرد که پشته ابزار اساسی برای این الگوریتم است.

یادآور می‌شویم که عملیات دودویی در Q ممکن است دارای سطوح تقدم مختلف باشند. به عنوان مثال، فرض را بر سه سطح تقدم یا اولویت زیر برای پنج عمل دودویی متداول قرار می‌دهیم:

↑ بالاترین اولویت · توان ↑

بعد از بالاترین اولویت : ضرب (*) و تقسیم (/)

پائین‌ترین اولویت : جمع (+) و تفریق (-)

مالحظه می‌کنید که ما برای توان از نماد زبان **BASIC** استفاده می‌کنیم. جهت سهولت فرض می‌کنیم که **Q** شامل هیچ عمل یگانی نیست (نظیر علامت منفی در ابتدای عبارت **b**). علاوه بر این فرض می‌کنیم که در تمام عبارت بدون پرانتز، عملیات هم‌سطح از چپ به راست اجرا می‌شوند. این فرض استاندارد نیست چون برخی از زبانهای برنامه‌نویسی عمل توان رساندن را از راست به چپ اجرا می‌کنند.

مثال ۴ - ۶

فرض کنید بخواهیم عبارت محاسباتی بدون پرانتز زیر را ارزیابی کنیم :

$$2 \uparrow 3 + 5 * 2 \uparrow 2 - 12 / 6$$

نخست توان را ارزیابی می‌کنیم، نتیجه چنین است :

$$8 + 5 * 4 - 12 / 6$$

آنگاه ضرب و تقسیم را ارزیابی می‌کنیم که به دست می‌آید $2 - 20 + 8$ درنهایت جمع و تفریق را ارزیابی می‌کنیم که نتیجهٔ نهایی 26 است. ملاحظه می‌کنید که این عبارت سه بار پیمایش می‌شود که هر بار متناظر با یک سطح از اولویت عملیات است.

نمادگذاری لهستانی

در متداول‌ترین عملیات محاسباتی، عملگر بین دو عملوند قرار می‌گیرد. به عنوان مثال

$$A + B \quad C - D \quad E * F \quad G / H$$

این نمادگذاری، نمادگذاری میانوندی نامیده می‌شود. با این نمادگذاری، مابین دو عبارت

$$(A + B) * C \quad A + (B * C)$$

با استفاده از پرانتزگذاری‌ها یا برخی از قراردادهای اولویت عملگرها نظیر سطوح متداول اولویت‌ها که در بالا مورد بررسی قرار گرفت تمايز قائل هستیم. بنابراین، ترتیب عملگرها و عملوند‌ها در یک عبارت محاسباتی با توجه به ترتیبی که در آن عملیات اجرا می‌شوند به‌طور منحصر‌بفرد تعیین نمی‌شود.

نمادگذاری لهستانی : این نامگذاری که پس از ریاضیدان لهستانی یان لوکاسیویچ صورت گرفته است مربوط به نمادگذاری‌ای می‌شود که در آن، عملگر قبل از دو عملوند قرار می‌گیرد. برای مثال :

$$+ AB \quad - CD \quad * EF \quad / GH$$

ما مرحله به مرحله، عبارتهای میانوندی زیر را با استفاده از دو کروشه [+] به نماد لهستانی تبدیل می‌کنیم که مبین تبدیل قسمت به قسمت آن است:

$$\begin{aligned} (A + B) * C &= [+AB] * C = ++ABC \\ A + (B * C) &= A + [*BC] = +A*BC \\ (A + B) / (C - D) &= [+AB] / [-CD] = / +AB - CD \end{aligned}$$

خاصیت اساسی نمادگذاری لهستانی آن است، ترتیبی که در آن عملیات انجام می‌شوند به وسیله مکان عملگرها و عملوندهای عبارت به طور کامل تعیین می‌شود. بنابراین هنگام نوشتن عبارتها با نماد لهستانی به هیچ پرانتزی نیاز نیست.

نماد لهستانی معکوس در ارتباط با نمادگذاری مشابه‌ای است که در آن نماد عملگر پس از دو عملوند قرار می‌گیرد:

$$AB + \quad CD - \quad EF * \quad GH /$$

در اینجا نیز برای تعیین ترتیب عملیات هر عبارت محاسباتی نوشته شده با نماد لهستانی معکوس به هیچ پرانتزی نیاز نیست. این نمادگذاری اغلب نمادگذاری پسوندی نامیده می‌شود در حالی که نمادگذاری پیشوندی اصطلاحی است که برای نمادگذاری لهستانی بکار می‌رود که در پاراگراف قبل مورد بررسی قرار گرفت.

کامپیوٹر معمولاً عبارت محاسباتی نوشته شده به صورت نمادگذاری میانوندی را در دو مرحله ارزیابی می‌کند. نخست عبارت را به صورت نمادگذاری پسوندی تبدیل می‌کند و آنگاه عبارت پسوندی را ارزیابی می‌کند. در هر مرحله، پشته، ابزار اصلی‌ای است که برای انجام این کار مشخص مورد استفاده قرار می‌گیرد. ما این کاربرد پشته‌ها را به ترتیب عکس نشان می‌دهیم، یعنی نخست نشان می‌دهیم که چگونه از پشته‌ها برای ارزیابی عبارت پسوندی استفاده می‌شود و آنگاه نشان می‌دهیم که چگونه از پشته‌ها در تبدیل عبارتهای میانوندی به صورت عبارتهای پسوندی استفاده می‌شود.

ارزیابی یک عبارت پسوندی فرض کنید P یک عبارت محاسباتی نوشته شده با نماد پسوندی باشد. الگوریتم زیر که از یک **STACK** برای نگهداری عملوندها استفاده می‌کند P را ارزیابی می‌کند.

Algorithm 6.3: This algorithm finds the VALUE of an arithmetic expression P written in postfix notation.

1. Add a right parenthesis ")" at the end of P. [This acts as a sentinel.]
2. Scan P from left to right and repeat Steps 3 and 4 for each element of P until the sentinel ")" is encountered.
3. If an operand is encountered, put it on STACK.
4. If an operator \otimes is encountered, then:
 - (a) Remove the two top elements of STACK, where A is the top element and B is the next-to-top element.
 - (b) Evaluate $B \otimes A$.
 - (c) Place the result of (b) back on STACK.
 [End of If structure.]
- [End of Step 2 loop.]
5. Set VALUE equal to the top element on STACK.
6. Exit.

توجه دارید که هنگام اجرای مرحله 5 تنها یک عدد در STACK خواهد بود.

مثال ۶-۵

عبارت محاسباتی P ای زیر را که به صورت نماد پسوندی نوشته شده است درنظر بگیرید:

$$P: 5, 6, 2, +, *, 12, 4, /, -$$

از کامامها به این دلیل برای جدا کردن عناصر P استفاده کردند $5, 6, 2$ تا 562 به صورت عدد 562 تعبیر نشود.

Symbol Scanned	STACK
(1) 5	5
(2) 6	5, 6
(3) 2	5, 6, 2
(4) +	5, 8
(5) *	40
(6) 12	40, 12
(7) 4	40, 12, 4
(8) /	40, 3
(9) -	37
(10))	

شکل ۶-۷

عبارت میانوندی Q معادل آن به صورت زیر است :

$$Q: 5 * (6 + 2) - 12 / 4$$

توجه دارید که در عبارت میانوندی Q به پرانتز احتیاج داریم اما در عبارت پسوندی P هیچ احتیاجی به پرانتزگذاری نیست.

P را با شبیه‌سازی الگوریتم 6.3 ارزیابی می‌کنیم. نخست یک پرانتز بسته نگهبان در انتهای P اضافه می‌کنیم که به دست می‌آید:

$$P: \quad \begin{matrix} 5, & 6, & 2, & +, & *, & 12, & 4, & /, & -, &) \\ (1) & (2) & (3) & (4) & (5) & (6) & (7) & (8) & (9) & (10) \end{matrix}$$

جهت سهولت در مراجعه عناصر P ، از چپ به راست شماره‌گذاری شده‌اند. شکل ۷-۶ محتوای $STACK$ را به مخصوص جستجو و خواندن هر عنصر P نشان می‌دهد. عدد آخر در $STACK$ یعنی 37 که در $VALUE$ جایگزین شده است، هنگامی که نگهبان "(") را جستجو و می‌خواند مقدار P است.

تبديل عبارتهای میانوندی به عبارتهای پسوندی

فرض کنید Q یک عبارت محاسباتی باشد که با نماد میانوندی نوشته شده است. علاوه بر عملوندها و عملگرها، Q نیز می‌تواند شامل پرانتزهای چپ و راست باشد. فرض می‌کنیم که عملگرها در Q تنها شامل عملگرهای توان (\uparrow)، ضرب (*)، تضییم (/)، جمع (+) و تفریق (-) می‌باشد و مانند بالا سه سطح تقدم یا اولویت متدالول دارد. علاوه براین فرض می‌کنیم عملگرهای هم‌سطح، منجمله عملگرهای توان از چپ به راست اجرا می‌شوند مگر آن که با پرانتزگذاری خلاف آن بیان شود. این قرارداد استاندارد نیست چون عبارتها می‌توانند شامل عملگرهای یکانی باشند و برخی از زبانهای برنامه‌نویسی عمل توان رسانند را از راست به چپ اجرا می‌کنند. با وجود این، ما از این فرضها جهت ساده‌شدن الگوریتمها استفاده می‌کنیم.

الگوریتم زیر عبارت میانوندی Q را به عبارت پسوندی معادل آن P تبدیل می‌کند. این الگوریتم از یک پشته برای نگهداری موقت عملگرها و پرانتزهای چپ استفاده می‌کند. عبارت پسوندی P با استفاده از عملوندها از Q و عملگرهایی که از $STACK$ حذف می‌شود از چپ به راست ساخته می‌شوند. کار را با کردن پرانتز چپ به درون $STACK$ شروع می‌کنیم و در پایان Q یک پرانتز راست اضافه می‌کنیم. الگوریتم هنگامی کامل می‌شود که $STACK$ خالی باشد.

Algorithm 6.4: POLISH(Q, P)

Suppose Q is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression P.

1. Push "(" onto STACK, and add ")" to the end of Q.
2. Scan Q from left to right and repeat Steps 3 to 6 for each element of Q until the STACK is empty:
 3. If an operand is encountered, add it to P.
 4. If a left parenthesis is encountered, push it onto STACK.
 5. If an operator \otimes is encountered, then:
 - (a) Repeatedly pop from STACK and add to P each operator (on the top of STACK) which has the same precedence as or higher precedence than \otimes .
 - (b) Add \otimes to STACK.
 - [End of If structure.]
 6. If a right parenthesis is encountered, then:
 - (a) Repeatedly pop from STACK and add to P each operator (on the top of STACK) until a left parenthesis is encountered.
 - (b) Remove the left parenthesis. [Do not add the left parenthesis to P.]
 - [End of If structure.]
 - [End of Step 2 loop.]
7. Exit.

اصطلاحی که گاهی اوقات در مرحله ۵ مورد استفاده قرار می‌گیرد \otimes است که به سطح پائین خودش اشاره می‌کند.

مثال ۶-۶

عبارت محاسباتی میانوندی Q زیر را در نظر بگیرید:

$$Q: A + (B * C - (D / E \uparrow F) * G) * H$$

الگوریتم 6.4 را شبیه‌سازی می‌کنیم تا Q را به عبارت پسوندی معادل آن P تبدیل کند. نخست "(" را به داخل STACK Push می‌کنیم و آنگاه ")" را در انتهای Q اضافه می‌کنیم که حاصل می‌شود:

$$Q: A + (B * C - (D / E \uparrow F) * G) * H)$$

$$(1) (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13) (14) (15) (16) (17) (18) (19) (20)$$

عنصرهای Q اکنون جهت سهولت در مراجعته به آنها از چپ به راست شماره گذاری شده‌اند. شکل ۶-۸ وضعیت STACK و وضعیت رشته P را وقتی که هر عنصر Q جستجو و خوانده می‌شود نشان می‌دهد. ملاحظه می‌کنید که

(۱) هر عملوند فقط به P اضافه می‌شود و STACK تغییر نمی‌کند.

(۲) عملگر تفريقي (-) در ردیف ۷، * را از STACK به P ارسال می‌کند قبل از آنها (-) به داخل Push، STACK شود.

(۳) پرانتز راست در ردیف ۱۴، ↑ و آنگاه / را از STACK به P ارسال می‌کند و آنگاه پرانتز چپ را از بالای STACK حذف می‌کند.

(۴) پرانتز راست در ردیف ۲۰، * و آنگاه + را از STACK به P ارسال می‌کند و آنگاه پرانتز چپ را از بالای STACK حذف می‌کند.

پس از اجرای مرحله ۲۰ STACK خالی است و

P: A B C * D E F ↑ / G * - H * +

که عبارت پسوندی موردنیاز معادل Q است.

Symbol Scanned	STACK	Expression P
(1) A	(A
(2) +	(+	A
(3) ((+ (A
(4) B	(+ (A B
(5) *	(+ (*	A B
(6) C	(+ (*	A B C
(7) -	(+ (-	A B C *
(8) ((+ (- (A B C *
(9) D	(+ (- (A B C * D
(10) /	(+ (- (/	A B C * D
(11) E	(+ (- (/	A B C * D E
(12) ↑	(+ (- (/ ↑	A B C * D E
(13) F	(+ (- (/ ↑	A B C * D E F
(14))	(+ (-	A B C * D E F ↑ /
(15) *	(+ (- *	A B C * D E F ↑ /
(16) G	(+ (- *	A B C * D E F ↑ / G
(17))	(+	A B C * D E F ↑ / G * -
(18) *	(+ *	A B C * D E F ↑ / G * -
(19) H	(+ *	A B C * D E F ↑ / G * - H
(20))		A B C * D E F ↑ / G * - H * +

شکل ۸-۶

۶-۵ QUICKSORT، یک کاربرد از پشتنهای

فرض کنید A یک لیست با n عنصر داده‌ای باشد. منظور ما از "مرتب کردن A" عمل تجدید آرایش عناصر A است تا این عناصر با یک ترتیب منطقی کنار هم قرار گیرند. یعنی وقتی که A از داده‌های عددی تشکیل می‌شود به صورت عددی مرتب شده باشند و وقتی A از داده‌های کاراکتری تشکیل می‌شود به

صورت الفایی مرتب شده باشند. موضوع مرتب کردن، به همراه الگوریتم‌های مختلف آن بطور اساسی در فصل ۹ مورد بررسی قرار می‌گیرد. این بخش تنها یک الگوریتم مرتب کردن را ارائه می‌دهد که الگوریتم QuickSort نام دارد تا یک کاربرد از پشتدها را نشان می‌دهد.

الگوریتمی از نوع تقسیم و غلبه است. به بیان دیگر، مسئله مرتب کردن یک مجموعه به مسئله مرتب کردن دو مجموعه کوچکتر تبدیل می‌شود. ما این مرحله ساده‌شدن را به کمک یک مثال مشخص توضیح می‌دهیم.

فرض کنید A لیست ۱۲ عددی زیر باشد :

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

مرحله ساده‌شدن الگوریتم QuickSort مکان نهایی یکی از اعداد را پیدا می‌کند. در این مثال از اولین عدد یعنی ۴۴ استفاده می‌کنیم. این کار به صورت زیر انجام می‌شود. با آخرین عدد یعنی ۶۶ شروع می‌کنیم لیست را از راست به چپ پیمایش می‌کنیم هر عدد را با ۴۴ مقایسه می‌کنیم و کار را در نخستین عدد کوچکتر از ۴۴ متوقف می‌کنیم. این عدد ۲۲ است. جای ۴۴ و ۲۲ را عوض می‌کنیم، لیست زیر به دست می‌آید:

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

(ملاحظه می‌کنید که اعداد ۸۸ و ۶۶ که در طرف راست ۴۴ هستند بزرگتر از ۴۴ می‌باشند). با ۲۲ شروع می‌کنیم به دنبال آن لیست را در جهت مخالف، از چپ به راست پیمایش می‌کنیم. هر دو عدد را با ۴۴ مقایسه کرده و کار را در نخستین عدد بزرگتر از ۴۴ متوقف می‌کنیم. این عدد ۵۵ است. جای دو عدد ۴۴ و ۵۵ را عوض می‌کنیم، لیست زیر به دست می‌آید.

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

(ملاحظه می‌کنید که اعداد ۲۲، ۳۳ و ۱۱ در طرف چپ ۴۴ همگی کوچکتر از ۴۴ هستند) این بار با ۵۵ شروع می‌کنیم، اکنون لیست را درجهت اصلی، از راست به چپ پیمایش می‌کنیم این کار را تا آنجا ادامه می‌دهیم تا به اولین عدد کوچکتر از ۴۴ برسیم. این عدد ۴۰ است. جای دو عدد ۴۴ و ۴۰ را عوض می‌کنیم، درنتیجه آن، لیست زیر حاصل می‌شود:

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

(مجددآ اعداد طرف راست ۴۴ همگی بزرگتر از ۴۴ هستند). با ۴۰ شروع می‌کنیم، لیست را از چپ به راست پیمایش می‌کنیم. اولین عدد بزرگتر از ۴۴ عدد ۷۷ است. جای دو عدد ۴۴ و ۷۷ را عوض می‌کنیم، درنتیجه آن، لیست زیر حاصل می‌شود:

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

(این بار عدد طرف چپ 44 همگی کوچکتر از 44 هستند). با 77 شروع می‌کنیم. لیست را از راست به چپ برای جستجوی یک عدد کوچکتر از 44 پیماش می‌کنیم. قبل از ملاقات با 44 چنین عددی را ملاقات نمی‌کنیم. معنی آن این است که تمام اعداد پیماش شدند و با 44 مقایسه شدند. علاوه بر این، تمام اعداد کوچکتر از 44 که اکنون در طرف چپ 44 هستند تشکیل لیست کوچکی از اعداد می‌دهند و تمام اعداد بزرگتر از 44 که اکنون در طرف راست 44 هستند تشکیل لیست کوچکی از اعداد می‌دهند و دو وضعیت در زیر نشان داده شده است.

22, 33, 11, 40,	(44)	90, 77, 60, 99, 55, 88, 66
First sublist	Second sublist	

لیست طرف چپ

لیست طرف راست

بدین ترتیب 44 به صورت صحیحی در مکان نهایی اش قرار گرفته است و کار مرتب‌کردن لیست اصلی A اکنون به کار مرتب‌کردن هر یک از دو لیست طرف چپ و راست بالا ساده می‌شود. مرحله ساده‌شدن بالا بر روی هر یک از دو لیست اخیر که شامل 2 یا چند عنصر است تکرار می‌شود. از آنجاکه تنها می‌توانیم، یکی از دو زیرلیست را پردازش کنیم، باید توانایی نگهداری چند لیست کوچک را، برای پردازش آتی داشته باشیم. این کار با استفاده از دو پشته تحت نامهای LOWER و UPPER انجام می‌شود که بطور موقت چنین لیستهای کوچکی را نگه می‌دارد. به بیان دیگر آدرس‌های اولین و آخرین عنصر هر یک از این لیستهای کوچک که مقادیر کرانه‌ای یا مرزی آن نامیده می‌شود به درون پشته‌های LOWER و UPPER می‌شود و مرحله ساده‌سازی یک لیست کوچک تنها پس از حذف مقادیر مرزی آن از پشته‌ها، اعمال می‌شود. مثال زیر روش استفاده از پشته‌های LOWER و UPPER را روشن می‌سازد.

مثال ۷-۶

لیست A بالا با $n = 12$ عنصر را در نظر بگیرید. الگوریتم با PUSH کردن مقادیر مرزی 1 و 12 از A به داخل پشته‌ها شروع می‌شود که به دست می‌آید:

LOWER: 1 UPPER: 12

برای اعمال مرحله ساده‌سازی، نخست الگوریتم مقادیر 1 و 12 را از بالای پشته‌ها حذف می‌کند، باقی می‌ماند:

LOWER: (empty) UPPER: (empty)

و آنگاه مرحله ساده‌شدن را به همان صورتی که در بالا انجام شد بر لیست متناظر [A[12], A[2], A[1], ..., A[1]] قرار می‌گیرد. بنابراین الگوریتم مقادیر مرزی 1 و 4 از اعمال می‌کنیم. نهایتاً عنصر اول، یعنی 44 در [A[5]] قرار می‌گیرد.

لیست کوچک شده اول و مقادیر مرزی ۶ و ۱۲ از لیست کوچک شده دوم را به داخل پشتهدان Push می‌کند، نتیجه می‌شود:

LOWER: 1, 6 UPPER: 4, 12

مجدداً برای اعمال مرحله ساده شدن، الگوریتم مقادیر ۶ و ۱۲ را از بالای پشتهدان حذف می‌کند، نتیجه می‌شود:

LOWER: 1 UPPER: 4

و آنگاه مرحله ساده شدن را بر لیست کوچک متناظر آن [A[6], A[7], A[8], ..., A[12]]، اعمال می‌کنیم. مرحله ساده شدن، این لیست را مطابق شکل ۹-۶ تغییر می‌دهد.

A[6],	A[7],	A[8],	A[9],	A[10],	A[11],	A[12],
90,	77,	60,	99,	55,	88,	66
66,	77,	60,	99,	55,	88,	90
66,	77,	60,	90,	55,	88,	99
66,	77,	60,	88,	55,	90,	99

لیست کوچک اول لیست کوچک دوم

شکل ۹-۶

ملاحظه می‌کنید که لیست کوچک دوم تنها یک عنصر دارد. بنابراین، الگوریتم تنها مقادیر مرزی ۶ و ۱۰ لیست کوچک اول را به داخل پشتهدان Push می‌کند. نتیجه می‌دهد:

LOWER: 1, 6 UPPER: 4, 10

و الى آخر. الگوریتم وقتی پایان می‌یابد که پشتهدان‌ها حاوی هیچ لیست کوچک شده‌ای که پردازش نشده است توسط مرحله ساده شدن نباشد.

بیان رسمی الگوریتم QuickSort به صورت زیر است. جهت سهولت در نمادگذاری و ملاحظات آموختشی، الگوریتم به دو قسمت تقسیم می‌شود. قسمت اول زیربرنامه Procedure را ادامه می‌دهد که QUICK نام دارد و مرحله ساده سازی بالا را در الگوریتم انجام می‌دهد و قسمت دوم از QUICK برای مرتب کردن تمام لیست استفاده می‌کند.

ملاحظه می‌کنید که مرحله (iii) از (c) ضروری نیست و جهت تأکید در تقارن بین مرحله ۲ و مرحله ۳ اضافه شده است. در این Procedure فرض نشده است که عناصر A از هم متمایز هستند. در غیر اینصورت شرط $LOC \neq RIGHT$ در مرحله (a) و شرط $LOC \neq LEFT$ در مرحله (a) قابل حذف

خواهد بود.

Procedure 6.5: QUICK(A, N, BEG, END, LOC)

Here A is an array with N elements. Parameters BEG and END contain the boundary values of the sublist of A to which this procedure applies. LOC keeps track of the position of the first element A[BEG] of the sublist during the procedure. The local variables LEFT and RIGHT will contain the boundary values of the list of elements that have not been scanned.

1. [Initialize.] Set LEFT := BEG, RIGHT := END and LOC := BEG.
2. [Scan from right to left.]
 - (a) Repeat while $A[LOC] \leq A[RIGHT]$ and $LOC \neq RIGHT$:
 $RIGHT := RIGHT - 1$.
 [End of loop.]
 - (b) If $LOC = RIGHT$, then: Return.
 - (c) If $A[LOC] > A[RIGHT]$, then:
 - (i) [Interchange $A[LOC]$ and $A[RIGHT]$.]
 $TEMP := A[LOC]$, $A[LOC] := A[RIGHT]$,
 $A[RIGHT] := TEMP$.
 - (ii) Set $LOC := RIGHT$.
 - (iii) Go to Step 3.
 [End of If structure.]
3. [Scan from left to right.]
 - (a) Repeat while $A[LEFT] \leq A[LOC]$ and $LEFT \neq LOC$:
 $LEFT := LEFT + 1$.
 [End of loop.]
 - (b) If $LOC = LEFT$, then: Return.
 - (c) If $A[LEFT] > A[LOC]$, then
 - (i) [Interchange $A[LEFT]$ and $A[LOC]$.]
 $TEMP := A[LOC]$, $A[LOC] := A[LEFT]$,
 $A[LEFT] := TEMP$.
 - (ii) Set $LOC := LEFT$.
 - (iii) Go to Step 2.
 [End of If structure.]

الگوریتم قسمت دوم به صورت زیر است، همانگونه که در بالا ملاحظه کردید، **UPPER** و **LOWER** پشتۀای هستند که در آنها مقادیر مرزی لیستهای کوچک شده ذخیره می‌شود. طبق معمول از استفاده می‌کنیم $\text{NULL} = 0$.

Algorithm 6.6: (Quicksort) This algorithm sorts an array A with N elements.

1. [Initialize.] TOP := NULL.
2. [Push boundary values of A onto stacks when A has 2 or more elements.]
If $N > 1$, then: TOP := TOP + 1, LOWER[1] := 1, UPPER[1] := N.
3. Repeat Steps 4 to 7 while TOP ≠ NULL.
4. [Pop sublist from stacks.]
Set BEG := LOWER[TOP], END := UPPER[TOP],
TOP := TOP - 1.
5. Call QUICK(A, N, BEG, END, LOC). [Procedure 6.5.]
6. [Push left sublist onto stacks when it has 2 or more elements.]
If BEG < LOC - 1, then:
TOP := TOP + 1, LOWER[TOP] := BEG,
UPPER[TOP] = LOC - 1.
[End of If structure.]
7. [Push right sublist onto stacks when it has 2 or more elements.]
If LOC + 1 < END, then:
TOP := TOP + 1, LOWER[TOP] := LOC + 1,
UPPER[TOP] := END.
[End of If structure.]
[End of Step 3 loop.]
8. Exit.

پیچیدگی الگوریتم QuickSort

زمان اجرای یک الگوریتم مرتب کردن معمولاً با تعداد n^f دفعات مقایسه مورد نیاز برای مرتب کردن n عنصر اندازه گیری می‌شود. الگوریتم QuickSort که دارای تعداد ادی مقایسه است به شدت مورد مطالعه و بررسی قرار گرفته است. در حالت کلی، این الگوریتم در بدترین حالت زمان اجرایی از مرتبه 2^{n^2} دارد اما زمان اجرای حالت میانگین آن از مرتبه $n \log n$ است. آن در زیر ارائه شده است.

بدترین حالت وقتی اتفاق می‌افتد که لیست از قبل مرتب شده باشد. آنگاه نخستین عنصر به n مقایسه احتیاج دارد تا معلوم شود در مکان اول قرار می‌گیرد. علاوه بر این، لیست کوچک شده اول خالی خواهد بود اما لیست کوچک شده دوم $-1 - n$ عنصر دارد. بنابراین عنصر دوم به $1 - n$ مقایسه احتیاج دارد تا معلوم شود در مکان دوم قرار می‌گیرد و الی آخر. درنتیجه، مجموعاً تعداد

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

مقایسه انجام می‌شود. ملاحظه می‌کنید که این عدد برابر پیچیدگی الگوریتم مرتب کردن حبابی است (ر. ک. بخش ۶.۴).

پیچیدگی $f(n) = O(n \log n)$ حالت میانگین از این واقعیت ناشی می‌شود که به طور متوسط، هر مرحله ساده‌سازی در الگوریتم دو لیست کوچکتر تولید می‌کند. بنابراین:

- (۱) با ساده‌شدن لیست اولیه، ۱ عنصر در جای خود قرار می‌گیرد و دو لیست کوچکتر تولید می‌شود.
- (۲) با ساده‌شدن دو لیست، 2 عنصر در جای خود قرار می‌گیرند و چهار لیست کوچکتر تولید می‌شود.
- (۳) با ساده‌شدن چهار لیست، 4 عنصر در جای خود قرار می‌گیرند و هشت لیست کوچکتر تولید می‌شود.
- (۴) با ساده‌شدن دو لیست، 8 عنصر در جای خود قرار می‌گیرند و شانزده لیست کوچکتر تولید می‌شود.
- و الى آخر. ملاحظه می‌کنید که مرحله ساده‌شدن در K امین سطح مکان عنصر 2^{k-1} ام را پیدا می‌کند. از این رو تقریباً $\log n$ سطح ساده‌سازی وجود دارد. علاوه براین، هر سطح حداقل از n مقایسه استفاده می‌کند. بنابراین $O(n \log n) = f(n)$ درواقع تحلیل ریاضی و ملاحظات تجربی هردو نشان می‌دهند که
- $$f(n) \approx 1.4[n \log n]$$

تعداد انتظاری مقایسه‌ها برای الگوریتم QuickSort است.

۶- زیربرنامه‌های بازگشتی

بازگشتی یک مفهوم بسیار مهم در علم کامپیوتر است. بسیاری از الگوریتم‌ها را می‌توان با استفاده از مفهوم بازگشتی به صورت کاراتری بیان کرد. این بخش، این ابزار قادر تمند را معرفی می‌کند و بخش ۶-۸ چگونگی پیاده‌سازی بازگشتی را با استفاده از پشته‌ها نشان می‌دهد.

فرض کنید P یک زیربرنامه Procedure باشد که حاوی یک دستور Call است که خودش را صدا می‌کند یا حاوی یک دستور Call است که زیربرنامه دوم را فرا می‌خواند که نهایتاً نتیجه دستور Call این است که زیربرنامه Procedure اصلی را صدا می‌زند. در آن صورت به P یک زیربرنامه بازگشتی می‌گویند، طوری که این برنامه به تعداد مرحله معین و محدودی اجرا می‌شود و پس از آن اجرا ادامه نمی‌یابد. هر زیربرنامه بازگشتی باید دو خاصیت زیر را داشته باشد:

(۱) باید معیار معینی وجود داشته باشد که معیار پایه یا مبنای نام دارد و با توجه به آن معیار، زیربرنامه Procedure خودش را صدا نمی‌زند.

(۲) هرباری که زیربرنامه Procedure (به طور مستقیم یا غیرمستقیم) خودش را صدا می‌زند، باید به معیار پایه نزدیکتر شود.

یک زیربرنامه بازگشتی یا Recursive را با دو معیار بالا، خوش تعریف می‌گویند.

به طور مشابه، یک تابع را به صورت بازگشتی تعریف شده می‌گویند هرگاه تعریف تابع به خودش برگردد. مجدداً برای این که تعریف چرخشی نباشد، باید دارای دو خاصیت زیر باشد:

(۱) باید آرگومانهای مشخصی وجود داشته باشد که به آن مقادیر پایه می‌گویند که به ازای این مقادیر

تابع خودش را صدآنمی‌زند.

(۲) هر بار که تابع خودش را صدآنمی‌زند، آرگومان تابع باید به مقدار پایه نزدیکتر شود. یک تابع بازگشته را با این دو خاصیت نیز خوش تعریف می‌گویند. مثال‌های زیر در روشن‌شدن مفهوم بازگشته به شما کمک خواهد کرد:

تابع فاکتوریل!

حاصل ضرب اعداد صحیح مثبت از ۱ تا خود n ، $n!$ فاکتوریل نامیده می‌شود و معمولاً آن را با $n!$ نمایش می‌دهند. بنابراین

$$n! = 1 \cdot 2 \cdot 3 \cdots (n-2)(n-1)n$$

بنایه قرارداد $0! = 1$ تعریف می‌شود. بدین ترتیب تابع فاکتوریل برای تمام اعداد صحیح مثبت تعریف می‌شود. بنابراین داریم

$$\begin{array}{llll} 0! = 1 & 1! = 1 & 2! = 1 \cdot 2 = 2 & 3! = 1 \cdot 2 \cdot 3 = 6 \\ & & 5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120 & 4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24 \\ & & & 6! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 = 720 \end{array}$$

و الی آخر، ملاحظه می‌کنید که

$$6! = 6 \cdot 5! = 6 \cdot 120 = 720 \quad \text{و} \quad 5! = 5 \cdot 4! = 5 \cdot 24 = 120$$

یعنی برای هر عدد صحیح مثبت n تساوی زیر برقرار است:

$$n! = n \cdot (n-1)!$$

بنابراین تابع فاکتوریل را می‌توان به صورت زیر نیز تعریف کرد:

تعریف ۱ - ۶ : (تابع فاکتوریل)

(الف) اگر $0 = n$ ، آنگاه $1 = n!$.

(ب) اگر $0 < n$ ، آنگاه $n! = n \cdot (n-1)!$

ملاحظه می‌کنید که این تعریف $n!$ بازگشته است، چون وقتی از $(1-n)$ استفاده می‌کند به خودش مراجعه می‌کند. بنابراین (الف) مقدار $n!$ به صورت صریح داده می‌شود حتی وقتی $0 = n$ است بدین ترتیب ۰ مقدار پایه است و (ب) مقدار $n!$ به ازای n دلخواه بر حسب مقدار کوچکتر n تعریف می‌شود که به مقدار پایه ۰ نزدیک است. بنابراین، تعریف چرخشی نیست یا به عبارت دیگر، این تابع خوش تعریف است.

مثال ۶-۶

با استفاده از تعریف بازگشتی فاکتوریل، $4!$ را محاسبه کنید. این محاسبه نیازمند نه مرحله زیر است:

- (1) $4! = 4 \cdot 3!$
- (2) $3! = 3 \cdot 2!$
- (3) $2! = 2 \cdot 1!$
- (4) $1! = 1 \cdot 0!$
- (5) $0! = 1$
- (6) $1! = 1 \cdot 1 = 1$
- (7) $2! = 2 \cdot 1 = 2$
- (8) $3! = 3 \cdot 2 = 6$
- (9) $4! = 4 \cdot 6 = 24$

يعنى :

مرحله ۱ : در این مرحله $4!$ برحسب $3!$ تعریف می شود، از این رو تا هنگام ارزیابی $3!$ باید محاسبه $4!$ به تعویق بیفتند. این تعویق با نوشتن مرحله بعدی به صورت پله‌ای مشخص شده است.

مرحله ۲ : در اینجا $3!$ برحسب $2!$ تعریف می شود، از این رو تا هنگام ارزیابی $2!$ باید محاسبه $3!$ به تعویق بیفتند.

مرحله ۳ : این مرحله $2!$ را برحسب $1!$ تعریف می کند.

مرحله ۴ : این مرحله $1!$ را برحسب $0!$ تعریف می کند.

مرحله ۵ : این مرحله $0!$ را می تواند به صورت صریح ارزیابی کند چون 0 مقدار پایه تعریف بازگشتی است.

مرحله های ۶ تا ۹ . حال از آخر به اول بر می گردیم، برای محاسبه $1!$ از $0!$ استفاده می کنیم، از $1!$ برای محاسبه $2!$ ، از $2!$ برای محاسبه $3!$ و بالاخره از $3!$ برای محاسبه $4!$ استفاده می کنیم. این برگشت از آخر به اول با نوشتن مراحل محاسبات به صورت پله‌ای معکوس شده، بیان شده است.

مالحظه می کنید که ما به ترتیب عکس از محاسبات به تعویق افتاده اصلی به اول برگشتهیم. یادآور می شویم این نوع از پردازش به تعویق افتاده خود منتهی به استفاده از پشته ها می شود. بخش ۶-۶ را ببینید.

در زیر دو زیر برنامه **Procedure** ارائه شده است که هر یک از آنها n فاکتوریل را محاسبه می کند:

Procedure 6.7A: FACTORIAL(FACT, N)

This procedure calculates $N!$ and returns the value in the variable FACT.

1. If $N = 0$, then: Set FACT := 1, and Return.
2. Set FACT := 1. [Initializes FACT for loop.]
3. Repeat for $K = 1$ to N .
 - Set FACT := K * FACT.
 - [End of loop.]
4. Return.

Procedure 6.7B: FACTORIAL(FACT, N)

This procedure calculates $N!$ and returns the value in the variable FACT.

1. If $N = 0$, then: Set FACT := 1, and Return.
2. Call FACTORIAL(FACT, N - 1).
3. Set FACT := $N * FACT$.
4. Return.

ملاحظه می‌کنید که زیربرنامه اول با استفاده از پردازش حلقه‌های تکرار $N!$ را ارزیابی می‌کند. از طرف دیگر، زیربرنامه دوم یک زیربرنامه بازگشتی است چون دارای پردازشی است که خودش را صدا می‌کند. برخی از زبانهای برنامه‌نویسی، به ویژه **FORTRAN** استفاده از چنین زیربرنامه‌های بازگشتی را مجاز نمی‌دانند.

فرض کنید P یک زیربرنامه بازگشتی باشد. در طی اجرای یک الگوریتم یا یک برنامه که حاوی P است، به هر بار اجرای زیربرنامه P به صورت زیر یک شماره سطح نسبت می‌دهیم. در اجرای زیربرنامه اصلی P سطح 1 جایگزین می‌شود و هر بار که زیربرنامه P اجرا می‌شود، بخارت یک احضار بازگشتی، سطح آن 1 واحد بیشتر از سطح اجرایی است، که باعث احضار بازگشتی شده است. در مثال ۶-۸، مرحله ۱، سطح ۱ دارد. از این‌رو مرحله ۲ سطح ۲، مرحله ۳ سطح ۳، مرحله ۴ سطح ۴ و مرحله ۵ سطح ۵ دارد. از سوی دیگر، مرحله ۶ سطح ۶ دارد چون نتیجه یک بازگشت از سطح ۵ است. به بیان دیگر، مرحله ۶ و مرحله ۴ متعلق به همان سطح اجرا هستند. بطور مشابه، مرحله ۷ سطح ۷، مرحله ۸ سطح ۸ و مرحله نهایی، مرحله ۹، سطح اولیه ۱ دارد.

عمق بازگشتی یک زیربرنامه بازگشتی P با یک مجموعه معین از آرگومانها، بزرگترین شماره سطح P در طول اجرای آن است.

دنباله فیبوناچی

دنباله زیبا و معروف فیبوناچی که معمولاً با $F_0, F_1, F_2, F_3, \dots$ نمایش داده می‌شود به صورت زیر است:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \dots$$

یعنی $0 = F_0$ و $1 = F_1$ و هر جمله بعدی مجموع دو جمله قبلی است. برای مثال، دو جمله بعدی دنباله بالا به صورت زیر محاسبه می‌شود:

$$55 + 89 = 144 \quad 34 + 55 = 89$$

تعريف رسمی این تابع به صورت زیر است:

تعريف ۲-۶ : (دبالة فیبوناچی)

(الف) اگر $n = 0$ یا $n = 1$ ، آنگاه $F_n = n$.

(ب) اگر $n > 1$ ، آنگاه $F_n = F_{n-2} + F_{n-1}$.

این مثال دیگری از یک تعریف بازگشته است، چون وقتی از F_{n-2} و F_{n-1} استفاده می‌کند این تعریف به خودش برمی‌گردد. در اینجا در (الف) مقادیر پایه ۰ و ۱ هستند و در (ب) مقدار F_n بر حسب مقادیر کوچکتر از n تعریف می‌شود که نزدیک به مقادیر پایه هستند. بنابراین اینتابع خوش تعریف است.

زیربرنامه **Procedure** ای که جمله n ام دبالة فیبوناچی F_n را پیدا می‌کند به صورت زیر است :

Procedure 6.8: FIBONACCI(FIB, N)

این زیربرنامه F_N را محاسبه می‌کند و مقدار آن را در پارامتر اول **FIB** قرار می‌دهد و به برنامه احضارکننده تحويل می‌دهد :

1. If $N = 0$ or $N = 1$, then: Set **FIB** := N , and Return.
2. Call **FIBONACCI(FIBA, N - 2)**.
3. Call **FIBONACCI(FIBB, N - 1)**.
4. Set **FIB** := **FIBA** + **FIBB**.
5. Return.

این مثال دیگری از یک زیربرنامه بازگشته است، چون زیربرنامه **Procedure** خودش را احضار می‌کنند. درواقع امر، این زیربرنامه دوبار خودش را احضار می‌کند. متذکر می‌شویم (ر.ک. مسئله ۶-۱۶) که می‌توان یک زیربرنامه **Procedure** با روش تکرار برای محاسبه F_n نوشت که در آن از زیربرنامه بازگشته استفاده نشود.

الگوریتم‌های تجزیه یا الگوریتم‌های تقسیم و غلبه

مسئله **P** را که در ارتباط با مجموعه **S** است درنظر بگیرید. فرض کنید **A** الگوریتمی است که **S** را به مجموعه‌های کوچکتر افزار می‌کند به گونه‌ای که حل مسئله **P** برای **S**، به حل **P** برای یک یا چند مجموعه کوچکتر منتهی شود. آنگاه **A** یک الگوریتم تقسیم و غلبه نامیده می‌شود. دو مثال از الگوریتم‌های تقسیم و غلبه که قبلًا مورد بررسی قرار گرفت الگوریتمها **QuickSort** در بخش ۵-۶ و الگوریتم جستجوی دودویی در بخش ۷-۴ است، یادآوری می‌کنیم که الگوریتم **QuickSort** از یک مرحله ساده‌شدن برای تعیین مکان یک عنصر و مسئله مرتب‌کردن تمام مجموعه برای مسئله مرتب‌کردن مجموعه‌های کوچکتر استفاده می‌کند. الگوریتم جستجوی دودویی مجموعه

مرتب داده شده را به دو نیمه تقسیم می‌کند به گونه‌ای که مسئله جستجو برای یک عنصر در تمام مجموعه به مسئله جستجوی آن عنصر در یکی از دو نیمه منتهی می‌شود.

الگوریتم تقسیم و غلبه A را می‌توان به صورت یک زیربرنامه بازگشته مورد توجه قرار داد. به این دلیل که وقتی آن را بر مجموعه‌های کوچکتر اعمال می‌کنیم الگوریتم A خودش را احضار می‌کند. معیار پایه و اصلی «این الگوریتم‌ها معمولاً مجموعه‌های یک عنصری است. برای مثال در الگوریتم مرتب‌کردن، مجموعه یک عنصری بطور خودکار مرتب شده است و در الگوریتم جستجو، مجموعه یک عنصری تنها نیازمند یک مقایسه است.

تابع آکرمان Ackermann

تابع آکرمان یک تابع با دو آرگومان است که در هر یک از این آرگومانها می‌تواند هر عدد صحیح غیرمنفی: 0, 1, 2, ..., جایگزین شود.

این تابع به صورت زیر تعریف می‌شود:

تعریف ۳-۶: (تابع آکرمان)

$$(الف) \text{ اگر } 0 = \text{آنگاه } m = n + 1$$

$$(ب) \text{ اگر } 0 \neq m \text{ اما } 0 = \text{آنگاه } n = A(m - 1, 1)$$

$$(ج) \text{ اگر } 0 \neq m \neq 0 \text{ و } 0 = \text{آنگاه } n = A(m - 1, A(m, n - 1))$$

در تابع آکرمان یک بار بیشتر، تعریف بازگشته داریم چون تعریف قسمت‌های (ب) و (ج) به خودش رجوع می‌کند.

ملاحظه می‌کنید که $A(m, n)$ به صورت صریح تنها وقتی $m = 0$ است داده شده است. معیار پایه زوجهای مرتب زیر هستند:

$$(0, 0), (0, 1), (0, 2), (0, 3), \dots, (0, n), \dots$$

هرچند از تعریف روشن نمی‌شود اما مقدار هر $A(m, n)$ را می‌توان در نهایت بر حسب مقدار تابع و براساس یک یا چند زوج مرتب پایه بیان کرد.

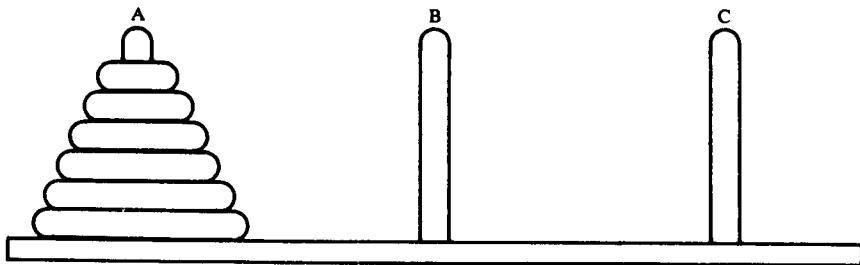
مقدار $A(1,3)$ در مسئله ۱۷-۶ محاسبه می‌شود. حتی این حالت ساده نیازمند ۱۵ مرحله است. در حالت کلی، تابع آکرمان آنقدر پیچیده است که هر مقدار دلخواه را نمی‌توان با آن محاسبه کرد اما امکان محاسبه مثالهای بدیهی و ساده وجود دارد. اهمیت تابع آکرمان از کاربرد آن در منطق ریاضی ناشی

می‌شود. در اینجا این تابع اساساً به این خاطر بیان شده است تا مثال دیگری از یک تابع بازگشته کلاسیک را رائمه دهد و نشان می‌دهیم که قسمت پازگشته این تعریف ممکن است پیچیده باشد.

۷-۶ برجهای هانوی

در بخش قبل مثالهایی چند از تعریفهای بازگشته و زیربرنامه‌های بازگشته ارائه دادیم. این بخش چگونگی استفاده از زیربرنامه بازگشته را به عنوان ابزاری جهت توسعه یک الگوریتم نشان می‌دهد که یک مسئله خاص را حل می‌کند. مسئله‌ای که برای این کار انتخاب شده است به مسئله برج‌های هانوی معروف است.

سه میله را در نظر بگیرید که با برچسب A، B، C مشخص شده‌اند و فرض کنید روی میله A تعداد n معین، n دیسک با اندازه‌های مختلف از بزرگ به کوچک قرار داده شده است. برای حالت $n = 6$ این وضعیت در شکل ۱۰-۶ نشان داده شده است.



شکل ۱۰-۶. وضعیت اولیه ایجاد برجهای هانوی با $n = 6$ دیسک

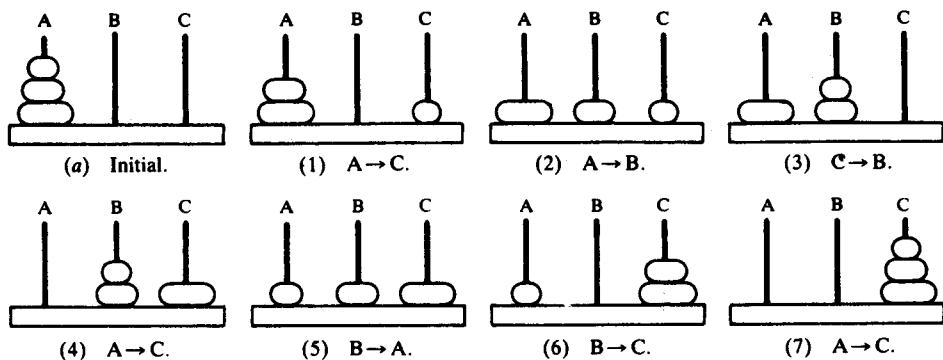
هدف بازی برج هانوی آن است که دیسکها را با استفاده از میله کمکی B، از میله A به میله C منتقل کیم، قوانین این بازی به صورت زیر است:

(الف) در هر بار تنها یک دیسک را می‌توان انتقال داد. به طور مشخص، تنها دیسک بالایی هر میله را می‌توان به هر میله دیگر منتقل کرد.

(ب) در هیچ زمانی نمی‌توان دیسک بزرگتر را روی دیسک کوچکتر قرار داد.

گاهی اوقات نمایش دستور "دیسک بالای میله X را به میله Y منتقل کنید" را به صورت $X \rightarrow Y$ می‌نویسیم که در آن X و Y می‌توانند هر یک از سه میله داده شده باشد.

در شکل ۱۱-۶ راه حل مسأله برجهای هانوی برای $n = 3$ دیسک ارائه شده است.



شکل ۱۱-۶

ملاحظه می‌کنید که حل مسأله برجهای هانوی در این حالت، از هفت انتقال یا جابجایی زیر تشکیل شده است:

$n = 3$: دیسک بالای میله A را به میله C منتقل کنید.

دیسک بالای میله A را به میله B منتقل کنید.

دیسک بالای میله C را به میله B منتقل کنید.

دیسک بالای میله A را به میله C منتقل کنید.

دیسک بالای میله B را به میله A منتقل کنید.

دیسک بالای میله B را به میله C منتقل کنید.

دیسک بالای میله A را به میله C منتقل کنید.

به بیان دیگر،

$$n = 3: \quad A \rightarrow C, \quad A \rightarrow B, \quad C \rightarrow B, \quad A \rightarrow C, \quad B \rightarrow A, \quad B \rightarrow C, \quad A \rightarrow C$$

برای روشن شدن مطلب، حل مسأله برجهای هانوی را برای $n = 1$ و $n = 2$ تیز می‌نویسیم:

$$n = 1: \quad A \rightarrow C$$

$$n = 2: \quad A \rightarrow B, \quad A \rightarrow C, \quad B \rightarrow C$$

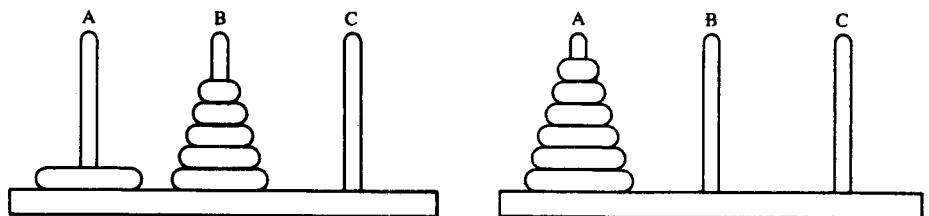
توجه دارید که در $n = 1$ تنها از یک جابجایی یا انتقال دیسک استفاده می‌کند و برای $n = 2$ از سه انتقال استفاده شده است.

به عرض پیدا کردن یک راه حل مجزا برای هر n دلخواه، ما از روش بازگشته برای ابداع یک راه حل کلی استفاده می‌کنیم. نخست ملاحظه می‌کنید که حل مسأله برجهای هانوی برای $n > 1$ دیسک می‌تواند

منجر به سه زیر مسئله داده شده در زیر می‌شود:

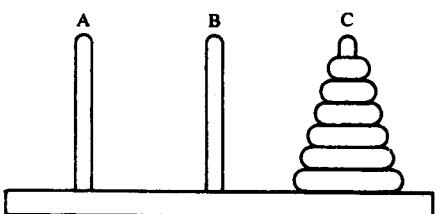
- (۱) تعداد $n - 1$ دیسک بالا را از میله A به میله B منتقل کنید.
- (۲) دیسک بالا از میله A را به میله C منتقل کنید: $A \rightarrow C$.
- (۳) تعداد $n - 1$ دیسک بالا را از میله B به میله C منتقل کنید.

به ازای $n = 6$ این مسئله‌های کاهش مرافق در شکل ۱۲-۶ نشان داده شده است یعنی نخست پنج دیسک بالای میله A را به میله B منتقل می‌کنیم، آنگاه دیسک بزرگ را از میله A به میله C منتقل می‌کنیم و بدنبال آن پنج دیسک بالای میله B را به میله C منتقل می‌کنیم:

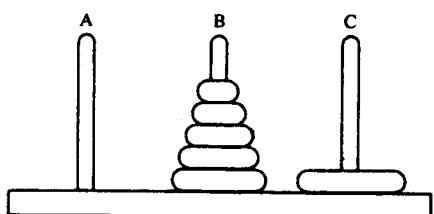


(ب) انتقال پنج دیسک بالای میله A به میله B

(الف) وضعیت اولیه:



(د) انتقال پنج دیسک بالای میله B به میله C



(ج) انتقال دیسک بالای میله A به میله C

شکل ۱۲-۶

اکنون وقت آن رسیده است که نماد اصلی را معرفی کنیم:

TOWER(N, BEG, AUX, END)

نماد بالا زیربرنامه **Procedure** ای را نشان می‌دهد که با استفاده از میله کمکی **AUX**، n دیسک بالای وضعیت اولیه میله **BEG** را به وضعیت نهایی میله **END** منتقل می‌کند. وقتی $1 = n$ است راه حل بدیهی زیر را داریم:

TOWER(1, BEG, AUX, END) تنها از دستور **BEG → END** تشکیل می‌شود. علاوه بر این همانگونه که در بالا مورد بررسی قرار گرفت وقتی $1 < n$ ، حل آن به حل سه زیرمسئله داده شده در زیر متفقی می‌شود:

TOWER($N - 1$, BEG, END, AUX) (۱)

TOWER(1, BEG, AUX, END) or BEG → END (۲)

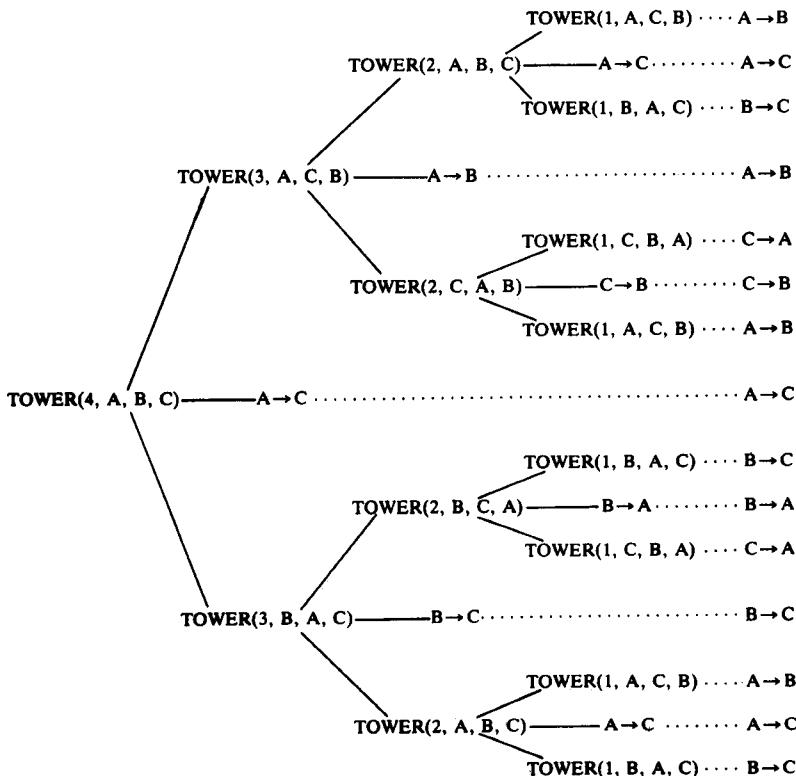
TOWER($N - 1$, AUX, BEG, END) (۳)

ملاحظه می‌کنید که هر یک از این سه زیرمسئله را می‌توان مستقیماً حل کرد یا اساساً همانند مسئله اصلی است با این تفاوت که در آن از تعداد دیسکهای کمتری استفاده شده است. بنابراین، پردازش این مسئله‌های کاهش مرحله، منتهی به یک راه حل بازگشتی برای مسئله برجهای هانوی می‌شود.

شکل ۱۳-۶ نمودار حل بازگشتی مسئله بالا برای

TOWER(4, A, B, C)

است.



شکل ۱۳-۶ حل بازگشتی مسئله برجهای هانوی برای $n = 4$ دیسک

ملاحظه می‌کنید که حل بازگشتی مسئله برجهای هانوی برای $n = 4$ دیسک شامل ۱۵ انتقال یا جابجایی زیر است:

$$\begin{array}{llllllll} A \rightarrow B & A \rightarrow C & B \rightarrow C & A \rightarrow B & C \rightarrow A & C \rightarrow B & A \rightarrow B & A \rightarrow C \\ B \rightarrow C & B \rightarrow A & C \rightarrow A & B \rightarrow C & A \rightarrow B & A \rightarrow C & B \rightarrow C & \end{array}$$

در حالت کلی، این حل بازگشتی برای n دیسک مستلزم $f(n) = 2^n - 1$ جابجایی یا انتقال است. بررسی خود را روی برجهای هانوی با زیربرنامه **Procedure** که به صورت رسمی نوشته شده است خلاصه می‌کنیم:

Procedure 6.9: TOWER(N, BEG, AUX, END)

This procedure gives a recursive solution to the Towers of Hanoi problem for N disks.

1. If $N = 1$, then:
 - (a) Write: $BEG \rightarrow END$.
 - (b) Return.

[End of If structure.]
2. [Move $N - 1$ disks from peg BEG to peg AUX .]
Call TOWER($N - 1$, BEG, END, AUX).
3. Write: $BEG \rightarrow END$.
4. [Move $N - 1$ disks from peg AUX to peg END .]
Call TOWER($N - 1$, AUX, BEG, END).
5. Return.

این زیربرنامه یک راه حل بازگشتی برای مسئله برجهای هانوی برای n دیسک ارائه می‌دهد. می‌توان این راه حل را به صورت یک الگوریتم تقسیم و غلبه مورد توجه قرار داد، چون راه حل مسئله برای n دیسک به راه حلی برای $1 - n$ و راه حلی برای $1 = n$ دیسک منجر می‌شود.

۶-۸ پیاده‌سازی زیربرنامه‌های بازگشتی به وسیله پشته‌ها

در بخش‌های قبل نشان داده‌ایم که چگونه برنامه‌های بازگشتی می‌توانند برای مسائل خاص یک ابزار مفید در توسعه الگوریتم‌ها باشد. در این بخش چگونگی استفاده از پشته‌ها در پیاده‌سازی زیربرنامه‌های بازگشتی نشان داده می‌شود. بهتر است ابتدا زیربرنامه‌ها را در حالت کلی مورد بحث و بررسی قرار دهیم. پادآوری می‌کنیم که یک زیربرنامه می‌تواند شامل همه پارامترها و همه متغیرهای محلی باشد. پارامترها، متغیرهایی هستند که مقادیر را از متغیرهای برنامه فراخواننده موسوم به آرگومانها دریافت می‌کنند و سپس مقادیر را به برنامه فراخواننده انتقال می‌دهند. علاوه بر پارامترها و متغیرهای محلی، زیربرنامه نیز باید آدرس بازگشت برنامه فراخواننده را نگهدارند. این آدرس بازگشتی اساسی و دارای

اهمیت است چون کنترل کار باید به مکان واقعی اش در برنامه فراخوانده منتقل شود. زمانی که اجرای زیربرنامه به پایان می‌رسد و کنترل کار به برنامه فراخوانده داده می‌شود به مقادیر متغیرهای محلی و آدرس بازگشته، دیگر نیازی نیست.

فرض کنید زیربرنامه ما یک زیربرنامه بازگشته است. آنگاه هر سطح اجرای زیربرنامه می‌تواند شامل مقادیر مختلف برای پارامترها و متغیرهای محلی و آدرس بازگشته باشد. علاوه براین، اگر برنامه بازگشته خودش را احضار کند، آنگاه این مقادیر جاری و گذرا، باید نگهداری شوند، چون هنگامی که برنامه مجدداً فعال می‌شود از آنها استفاده خواهد کرد.

فرض کنید یک برنامه‌نویس از یک زبان سطح بالا نظیر PASCAL استفاده می‌کند که اجازه استفاده از زیربرنامه‌های بازگشته را می‌دهد. آنگاه کامپیوتر از حافظه‌هایی برای نگهداری تمام مقادیر پارامترها، متغیرهای محلی و آدرس‌های بازگشته استفاده می‌کند. از طرف دیگر، اگر یک برنامه‌نویس از یک زبان برنامه‌نویسی سطح بالای دیگری نظیر FORTRAN استفاده کند که اجازه استفاده از زیربرنامه‌های بازگشته را نمی‌دهد آنگاه برنامه‌نویس باید تمهیداتی به عمل آورد تا زیربرنامه بازگشته را به یک زیربرنامه غیربازگشته تبدیل کند. این تمهیدات در زیر توضیح داده می‌شود.

تبدیل یک زیربرنامه بازگشته به یک زیربرنامه غیربازگشته

فرض کنید P یک زیربرنامه بازگشته باشد. فرض می‌کنیم P به عوض یک زیربرنامه تابعی، یک زیربرنامه Subroutine باشد. (بدون این که عمومیت مسئله از دست برود، زیرا زیربرنامه‌های تابع را می‌توان به سادگی به صورت زیربرنامه‌های Subroutine نوشت). علاوه بر این فرض می‌کنیم که فقط زیربرنامه P است که خود P را به صورت بازگشته فرا می‌خواند. بررسی بازگشته به صورت غیرمستقیم خارج از حدود مطالب درس ساختمنداده‌ها و این کتاب است.

در تبدیل زیربرنامه بازگشته P به یک زیربرنامه غیربازگشته به صورت زیر عمل می‌کنیم. قبل از هر چیز تعریفهای زیر را داریم:

(۱) برای هر پارامتر STPAR یک پشته PAR داریم.

(۲) برای هر متغیر محلی STVAR یک پشته VAR داریم.

(۳) برای نگهداری آدرس‌های بازگشته یک متغیر محلی ADD و یک پشته STADD داریم.

هر باری که یک احضار بازگشته به P وجود دارد، مقادیر جاری پارامترها و متغیرهای محلی برای پردازش آتی به داخل پشته‌های مربوطه Push می‌شوند و هر باری که یک بازگشت بازگشته به P وجود

دارد، مقادیر پارامترها و متغیرهای محلی برای اجرای جاری P از پشته‌ها برگردانده می‌شود. کار با آدرسهای بازگشته بسیار پیچیده است و به صورت زیر انجام می‌شود:

فرض کنید زیربرنامه P شامل یک فراخوانی بازگشته P در مرحله K باشد. آنگاه دو آدرس بازگشته منتظر با اجرای این مرحله K وجود دارد.

(۱) آدرس بازگشته جاری زیربرنامه P وجود دارد و هنگامی مورد استفاده قرار می‌گیرد که اجرای سطح جاری از اجرای P به پایان رسد.

(۲) آدرس بازگشته جدید $K+1$ وجود دارد که آدرس مرحله بعد از احضار P است و برای بازگشت به سطح جاری از اجرای زیربرنامه P مورد استفاده قرار می‌گیرد.

برخی از کتابها، اولین آدرس از این دو آدرس یعنی آدرس بازگشته جاری، را به داخل آدرس بازگشته پشته $Push$ ، $STADD$ می‌کنند در حالی که برخی دیگر، آدرس دوم یعنی آدرس بازگشته جدید $K+1$ را به داخل پشته $Push$ ، $STADD$ می‌کنند. ما روش اول را انتخاب می‌کنیم چون تبدیل P به یک زیربرنامه غیربازگشته ساده‌تر خواهد بود. به خصوص به این تعبیر که یک پشته خالی $STADD$ ، یک بازگشت به برنامه اصلی را بیان می‌کند که در ابتدای کار، زیربرنامه بازگشته P را احضار می‌کند. تبدیل دیگری که آدرس بازگشته جاری را به داخل پشته $Push$ می‌کند در مسئله ۲۰-۶ مورد بررسی قرار می‌گیرد.

الگوریتمی که زیربرنامه بازگشته P را به صورت یک زیربرنامه غیربازگشته تبدیل می‌کند به شرح زیر است: این الگوریتم از سه قسمت تشکیل شده است:

(۱) آماده‌سازی (۲) تبدیل فراخوانی بازگشته P در زیربرنامه P و (۳) تبدیل هر بازگشت $Return$ در زیربرنامه P .

(۱) "آماده‌سازی".

(الف) برای هر پارامتر $STPAR$ یک پشته PAR ، برای هر متغیر محلی $STVAR$ یک پشته VAR و برای نگهداری آدرسهای بازگشته یک متغیر محلی ADD و یک پشته $STADD$ تعریف کنید.

(ب) قرار دهید: $TOP := NULL$

(۲) تبدیل "مرحله K ، فراخوانی P "

(الف) مقادیر جاری پارامترها و متغیرهای محلی را به داخل پشته‌های مربوط $Push$ کنید و آدرس بازگشت جدید [مرحله] $K+1$ را به داخل $STADD$ ، PAR کنید.

(ب) با استفاده از مقادیر آرگومان جاری پارامترها را مجدداً مقداردهی کنید.

(ج) به مرحله ۱ بروید. [شروع زیربرنامه P].

(۳) تبدیل "مرحله J، بازگشت "Return

(الف) اگر STADD خالی است، آنگاه بازگشت کنید Return. [کنترل کار به برنامه اصلی بازگشت پیدا می‌کند].

(ب) مقادیر بالای پشتدها را برگردانید، یعنی پارامترها و متغیرهای محلی را برابر مقادیر بالای پشتدها قرار دهید و ADD را برابر مقدار بالای پشته STADD قرار دهید.

(ج) به مرحله ADD بروید.

ملاحظه می‌کنید که تبدیل "مرحله K، فراخوانی P" بستگی به مقدار K ندارد اما تبدیل "مرحله J، بازگشت Return" به مقدار J بستگی دارد. بنابراین تنها لازم است که دستور بازگشت Return تبدیل شود برای مثال، با استفاده از

Step L. Return

مانند بالا و آنگاه

Go To Step L.

را جایگزین هر دستور بازگشت Return دیگر کنید.

این کار تبدیل زیربرنامه Procedure را ساده می‌کند.

برجهای هانوی، صورت تجدیدنظر شده

بار دیگر مسأله برجهای هانوی را درنظر بگیرید. زیربرنامه ۹-۶ یک راه حل بازگشتی برای این مسأله با n دیسک است. ما این زیربرنامه را به یک حل غیربازگشتی تبدیل می‌کنیم. برای حفظ و نگهداری مراحل مشابه، دستور شروع را همانند مرحله ۰ با NULL := TOP: تنها دستور بازگشت Return را در مرحله ۵ همانند (۳) در صفحه قبل تبدیل می‌کنیم.

Procedure 6.10: TOWER(N, BEG, AUX, END)

This is a nonrecursive solution to the Towers of Hanoi problem for N disks which is obtained by translating the recursive solution. Stacks STN, STBEG, STAUX, STEND and STADD will correspond, respectively, to the variables N, BEG, AUX, END and ADD.

0. Set TOP := NULL.
1. If N = 1, then:
 - (a) Write: BEG → END.
 - (b) Go to Step 5.

[End of If structure.]
2. [Translation of “Call TOWER(N – 1, BEG, END, AUX).”]
 - (a) [Push current values and new return address onto stacks.]
 - (i) Set TOP := TOP + 1.
 - (ii) Set STN[TOP] := N, STBEG[TOP] := BEG,
STAUX[TOP] := AUX, STEND[TOP] := END,
STADD[TOP] := 3.
 - (b) [Reset parameters.]
Set N := N – 1, BEG := BEG, AUX := END, END := AUX.
 - (c) Go to Step 1.
3. Write: BEG → END.
4. [Translation of “Call TOWER(N – 1, AUX, BEG, END).”]
 - (a) [Push current values and new return address onto stacks.]
 - (i) Set TOP := TOP + 1.
 - (ii) Set STN[TOP] := N, STBEG[TOP] := BEG,
STAUX[TOP] := AUX, STEND[TOP] := END,
STADD[TOP] := 5.
 - (b) [Reset parameters.]
Set N := N – 1, BEG := AUX, AUX := BEG, END := END.
 - (c) Go to Step 1.
5. [Translation of “Return.”]
 - (a) If TOP := NULL, then: Return.
 - (b) [Restore top values on stacks.]
 - (i) Set N := STN[TOP], BEG := STBEG[TOP],
AUX := STAUX[TOP], STEND[TOP],
ADD := STADD[TOP].
 - (ii) Set TOP := TOP – 1.
 - (c) Go to Step ADD.

فرض کنید برنامه اصلی شامل دستور زیر است:

Call TOWER(3, A, B, C)

اجرای حل مسئله در زیربرنامه 6.10 را شبیه‌سازی می‌کنیم، تأکید می‌داریم که در سطح‌های مختلف اجرای زیربرنامه است. هر سطح اجرا با مرحله مقدار اولیه دادن شروع می‌شود که در پارامترها، مقادیر آرگومانها از دستور فراخواننده اولیه یا از فراخوان بازگشتی در مرحله (۲) یا مرحله (۴) جایگزین می‌شود. از این‌رو هر آدرس بازگشت جدید یا مرحله ۳ یا مرحله ۵ است. شکل ۶-۱۴ مراحله‌های مختلف پشت‌های را نشان می‌دهد.

STN:	3	3, 2	3	3, 2	3		3	3, 2	3	3, 2	3	
STBEG:	A	A, A	A	A, A	A		A	A, B	A	A, B	A	
STAUX:	B	B, C	B	B, C	B		B	B, A	B	B, A	B	
STEND:	C	C, B	C	C, B	C		C	C, C	C	C, C	C	
STADD:	3	3, 3	3	3, 5	3		5	5, 3	5	5, 5	5	
	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)	(k)	(l)

شکل ۱۴ - ۶ پشته‌ها برای TOWER(3, A, B, C)

(الف) (سطح ۱) فراخوانی اولیه CALL TOWER(3, A, B, C) مقادیر زیر را در پارامترها جایگزین می‌کند:

$$N := 3, \quad BEG := A, \quad AUX := B, \quad END := C$$

مرحله ۱. چون $N \neq 1$ ، کنترل به مرحله ۲ داده می‌شود.

مرحله ۲. این یک فراخوان بازگشته است. از این رو مقادیر جاری متغیرها و آدرس بازگشت جدید (در

مرحله ۳) به داخل پشته‌ها مانند شکل ۱۴ - ۶ (الف) Push می‌شوند.

(ب) (سطح ۲) مرحله ۲ فراخوان بازگشته [TOWER(N - 1, BEG, END, AUX)] مقادیر زیر را در پارامترها جایگزین می‌کند:

$$N := N - 1 = 2, \quad BEG := BEG = A, \quad AUX := END = C, \quad END := AUX = B$$

مرحله ۱. چون $N \neq 1$ کنترل به مرحله ۲ داده می‌شود.

مرحله ۲. این یک فراخوان بازگشته است. از این رو مقادیر جاری متغیرها و آدرس بازگشت جدید

(مرحله ۳) به داخل پشته‌ها مانند شکل ۱۴ - ۶ (ب) Push می‌شوند.

(ج) (مرحله ۳) مرحله ۲ فراخوان بازگشته [TOWER(N - 1, BEG, END, AUX)] مقادیر زیر را در پارامترها جایگزین می‌کند:

$$N := N - 1 = 1, \quad BEG := BEG = A, \quad AUX := END = B, \quad END := AUX = C$$

مرحله ۱. اکنون $N = 1$ عمل BEG → END انتقال زیر را پیاده‌سازی می‌کند:

$$A \longrightarrow C$$

اکنون کنترل به مرحله ۵ منتقل می‌شود. [برای بازگشت

مرحله ۵. پشته‌ها خالی نیستند، از این رو مقادیر بالای پشته‌ها حذف می‌شوند و به شکل ۱۴ - ۶ (ج)

درمی‌آید و جایگزینی‌های زیر صورت می‌گیرد:

$$N := 2, \quad BEG := A, \quad AUX := C, \quad END := B, \quad ADD := 3$$

کنترل به سطح ۲ قبلی در مرحله ADD منتقل می‌شود.

(د) (سطح ۲) فعال شدن در مرحله ۳ [ADD = ۳]

مرحله ۳. عمل BEG → END → انتقال زیر را پیاده‌سازی می‌کند:

$$A \longrightarrow B$$

مرحله ۴. این یک فرآخوان بازگشته است. از این‌رو مقادیر جاری متغیرها و آدرس بازگشت جدید (مرحله ۵) به داخل پشته‌ها مانند شکل ۱۴-۶ (د) Push می‌شوند. (ه) (سطح ۳) مرحله ۴ فرآخوان بازگشته است [TOWER(N - 1, AUX, BEG, END)] مقادیر زیر را در پaramترها جایگزین می‌کند:

$$N := N - 1 = 1, \quad BEG := AUX = C, \quad AUX := BEG = A, \quad END := END = B$$

مرحله ۱. اکنون $N = 1$ عمل BEG → END → انتقال زیر را پیاده‌سازی می‌کند:

$$C \longrightarrow B$$

آنگاه کنترل به مرحله ۵ منتقل می‌شود. [برای بازگشت Return]

مرحله ۵. پشته‌ها خالی نیستند، از این‌رو مقادیر بالای پشته‌ها حذف می‌شوند و به شکل ۱۴-۶ (ه) درمی‌آید و جایگزینی‌های زیر صورت می‌گیرد:

$$N := 2, \quad BEG := A, \quad AUX := C, \quad END := B, \quad ADD := 5$$

کنترل به سطح ۲ قبلی در مرحله ADD منتقل می‌شود.

(و) (سطح ۲) فعال شدن در مرحله ۵ [ADD = ۵]

مرحله ۵. پشته‌ها خالی نیستند، از این‌رو مقادیر بالای پشته‌ها حذف می‌شوند و به شکل ۱۴-۶ (و) درمی‌آید و جایگزینی‌های زیر صورت می‌گیرد:

$$N := 3, \quad BEG := A, \quad AUX := B, \quad END := C, \quad ADD := 3$$

کنترل به سطح ۱ قبلی در مرحله ADD منتقل می‌شود.

(ز) (سطح ۱) فعال شدن در مرحله ۳ [ADD = ۳]

مرحله ۳. عمل BEG → END → انتقال زیر را پیاده‌سازی می‌کند:

$$A \longrightarrow C$$

مرحله ۴. این یک فرآخوان بازگشته است. از این‌رو مقادیر جاری متغیرها و آدرس بازگشت جدید (مرحله ۵) به داخل پشته‌ها مانند شکل ۱۴-۶ (و) Push می‌شوند.

(ح) (سطح ۲) مرحله ۴ فرآخوان بازگشته است [TOWER(N - 1, AUX, BEG, END)] مقادیر زیر را در پaramترها جایگزین می‌کند:

$$N := N - 1 = 2, \quad BEG := AUX = B, \quad AUX := BEG = A, \quad END := END = C$$

مرحله ۱. چون $1 \neq N$ ، کنترل به مرحله ۲ منتقل می‌شود.

مرحله ۲ . این یک فراخوان بازگشته است. از این رو مقادیر جاری متغیرها و آدرس بازگشت جدید (مرحله ۳) به داخل پشته‌ها مانند شکل ۱۴-۶ (ج) Push می‌شوند.

(ط) (مرحله ۳) مرحله ۲ فراخوان بازگشت [TOWER(N - 1, BEG, END, AUX)] مقادیر زیر را در پارامترها جایگزین می‌کند:

$$N := N - 1 = 1, \quad BEG := BEG = B, \quad AUX := END = C, \quad END := AUX = A$$

مرحله ۱ . اکنون $N = 1$ عمل BEG → END انتقال زیر را پیاده‌سازی می‌کند :

$$B \longrightarrow A$$

آنگاه کنترل به مرحله ۵ داده می‌شود. [برای بازگشت Return]

مرحله ۵ . پشته‌ها خالی نیستند از این رو مقادیر بالای پشته‌ها حذف می‌شوند و به شکل ۱۴-۶ (ط) درمی‌آید و جایگزینی‌های زیر صورت می‌گیرد :

$$N := 2, \quad BEG := B, \quad AUX := A, \quad END := C, \quad ADD := 3$$

کنترل به سطح ۲ قبلی در مرحله ADD منتقل می‌شود.

(ی) (سطح ۲) [فعال شدن در مرحله ۳] ADD = 3

مرحله ۳ . عمل BEG → END انتقال زیر را پیاده‌سازی می‌کند :

$$B \longrightarrow C$$

مرحله ۴ . این یک فراخوان بازگشته است. از این رو مقادیر جاری متغیرها و آدرس بازگشت جدید (مرحله ۵) به داخل پشته‌ها مانند شکل ۱۴-۶ (ی) Push می‌شوند.

(ک) (سطح ۳) مرحله ۴ فراخوان بازگشته [TOWER(N - 1, AUX, BEG, END)] دریز زیر را در پارامترها جایگزین می‌کند :

$$N := N - 1 = 1, \quad BEG := AUX = C, \quad AUX := BEG = B, \quad END := END = C$$

مرحله ۱ . اکنون $N = 1$ عمل BEG → END انتقال زیر را پیاده‌سازی می‌کند :

$$A \longrightarrow C$$

آنگاه کنترل به مرحله ۵ داده می‌شود. [برای بازگشت Return]

مرحله ۵ . پشته‌ها خالی نیستند از این رو مقادیر بالای پشته‌ها حذف می‌شوند و به شکل ۱۴-۶ (ک) درمی‌آید و جایگزینی‌های زیر صورت می‌گیرد :

$$N := 2, \quad BEG := B, \quad AUX := A, \quad END := C, \quad ADD := 5$$

کنترل به سطح ۲ قبلی در مرحله ADD منتقل می‌شود.

(ل) (سطح ۲) [فعال شدن در مرحله ۵] ADD = 5

مرحله ۵ . پشته‌ها اکنون خالی نیستند. از این رو مقادیر بالای پشته‌ها حذف می‌شوند و به شکل ۱۶-۴ (ل) درمی‌آید و جایگزینی‌های زیر صورت می‌گیرد :

$$N := 3, \quad BEG := A, \quad AUX := B, \quad END := C, \quad ADD := 5$$

کنترل به سطح ۱ قبلی در مرحله ADD منتقل می‌شود.

(م) (سطح ۱) [فعال شدن در مرحله ۵ $\text{ADD} = 5$]

مرحله ۵. پشتنهای اکنون خالی هستند. بنابراین کنترل به برنامه اصلی اولیه که شامل دستور زیر است داده می‌شود:

Call TOWER(3, A, B, C)

مالحظه می‌کنید که خروجی شامل هفت انتقال زیر است:

$A \rightarrow C, A \rightarrow B, C \rightarrow B, A \rightarrow C, B \rightarrow A, B \rightarrow C, A \rightarrow C,$

این نتیجه با جواب شکل ۱۱-۶ سازگار است.

جمع‌بندی مطالب

مسأله برجهای هانوی، قدرت زیربرنامه‌های بازگشته را در حل مسائل الگوریتمی متعدد روشن می‌سازد. این بخش چگونگی پیاده‌سازی زیربرنامه‌های بازگشته را به وسیله پشتنهای نشان می‌دهد و این در حالی است که از یک زبان برنامه‌نویسی نظری **COBOL** یا **FORTRAN** یا **PASCAL** استفاده می‌کنیم که نوشتن برنامه‌های بازگشته در آنها مجاز نیست. در واقع، حتی وقتی که از یک زبان برنامه‌نویسی نظری استفاده می‌کنیم که از برنامه‌های بازگشته پشتیبانی می‌کند، برنامه‌نویس ممکن است بخواهد از راه حل غیربازگشته استفاده کند چون راه حل غیربازگشته هزینه‌کمتری نسبت به راه حل بازگشته دارد.

۹-۶ صفحه‌ها

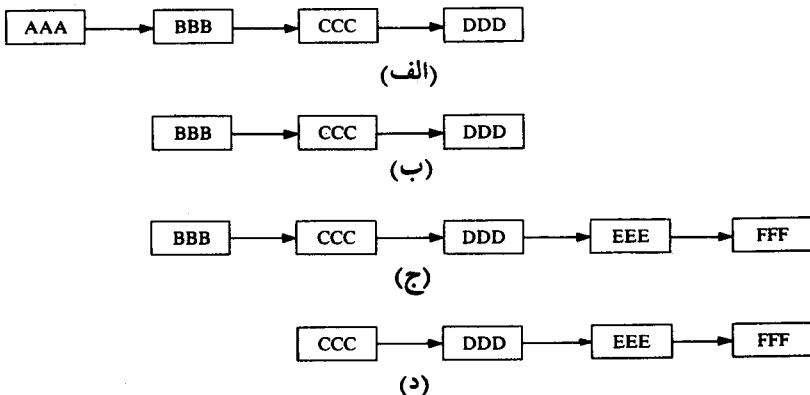
یک صفحه، لیست خطی از عناصر است که در آن عمل حذف تنها می‌تواند از یک انتهای آن موسوم به سر صفحه یا ابتدای صفحه front و عمل اضافه‌شدن تنها می‌تواند از انتهای دیگر آن موسوم تو صفحه یا انتهای آن rear صورت گیرد. اصطلاح ابتدای صفحه front و انتهای صفحه rear در توصیف یک لیست خطی تنها وقتی مورد استفاده قرار می‌گیرد که به عنوان یک صفحه پیاده‌سازی شود.

صفتها، لیستهای اولین و رودی اولین خروجی است، **FIFO** نیز نامیده می‌شود چون اولین عنصر داخل یک صفحه، اولین عنصری است که از صفحه خارج می‌شود یا آن را ترک می‌کند. به بیان دیگر، ترتیبی که در آن عناصرها وارد یک صفحه می‌شوند به همان ترتیبی است که عناصرها، صفحه را ترک می‌کنند. صفحهای در مقابل پشتنهای آخرین و رودی اولین خروجی است **LIFO** هستند. صفحهای در زندگی روزمره ما به وفور دیده می‌شود. اتومبیل‌هایی که در یک چهارراه در انتظار عبور هستند تشکیل یک صفحه می‌دهند که در آن، اولین اتومبیل وارد شده در صفحه انتظار، اولین اتومبیل است که صفحه را ترک می‌کند. انسانهایی که در یک بانک در یک خط در انتظار انجام کارشان هستند، تشکیل یک صفحه را می‌دهند که در آن اولین نفر داخل خط، اولین کسی است که در انتظار انجام کارش به سر می‌برد و پس از انجام، اولین کسی است که صفحه را ترک می‌کند و الى آخر. یک مثال کامل از صفحه در علم کامپیوتر، در سیستم اشتراک زمانی اتفاق می‌افتد، که در آن برنامه‌هایی که دارای اولویت یکسان

هستند تشکیل یک صف را می‌دهند و در حال انتظار اجرا بسر می‌برند. ساختمان دیگری که از صف استفاده می‌کند یک صف اولویت نام دارد که در بخش ۱۱-۶ مورد بحث و بررسی قرار خواهد گرفت.

مثال ۹-۶

شکل ۱۵-۶ (الف) نمودار یک صف با ۴ عنصر را نشان می‌دهد که در آن AAA عنصر ابتدای صف و DDD عنصر انتهای صف است. ملاحظه می‌کنید که عنصر ابتدا و انتهای صف نیز به ترتیب عنصر اول و آخر لیست هستند. فرض کنید یک عنصر از صف حذف شده است. آنگاه عنصر حذف شده باید AAA باشد. این عمل صف را در وضعیت شکل ۱۵-۶ (ب) قرار می‌دهد که در آن BBB اکنون عنصر اول صف است.



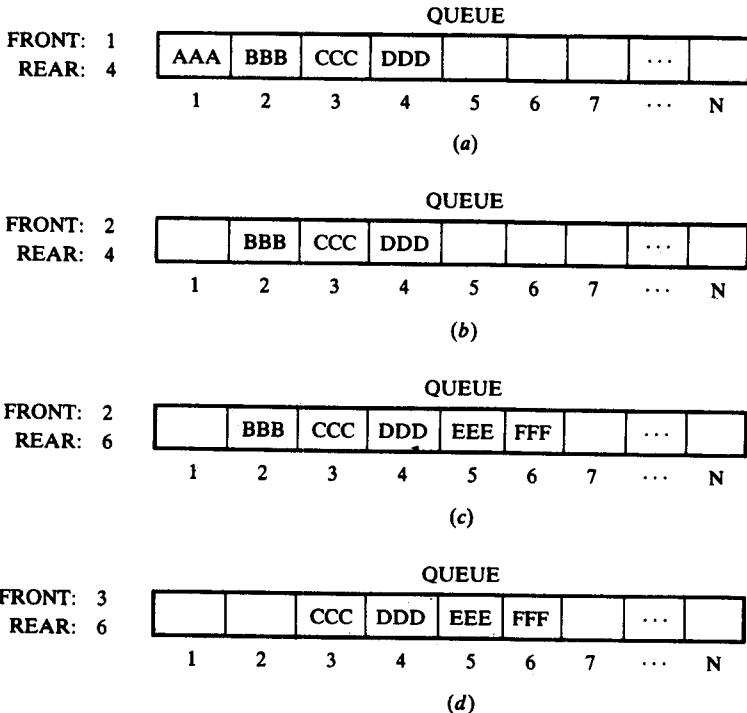
شکل ۱۵-۶

بدنبال آن، فرض کنید EEE به صف اضافه شده است و آنگاه FFF به صف اضافه می‌شود. در آن صورت این عناصر باید به انتهای صف اضافه شوند. این وضعیت در شکل ۱۵-۶ (ج) نشان داده شده است. توجه دارید که اکنون عنصر انتهای صف است. حال فرض کنید عنصر دیگری از صف حذف شده است. آنگاه این عنصر باید BBB باشد که با این عمل صف در وضعیت شکل ۱۵-۶ (د) قرار می‌گیرد و الى آخر. ملاحظه می‌کنید که در چنین ساختمان داده‌ای، FFF قبل از EEE حذف می‌شود چون این عنصر قبل از FFF در صف قرار گرفته است. بنابراین، EEE باید متظر بماند تا CCC و DDD حذف شوند.

نمایش صفحه

صفهای را می‌توان در کامپیوتر به صورتهای مختلف نمایش داد اما معمولاً آنها را با لیستهای یکطرفه

یا آرایه‌های خطی نمایش می‌دهند. هر یک از صفحه‌ای داخل کتاب توسط یک آرایه خطی **QUEUE** و دو متغیر اشاره‌گر **FRONT** که شامل مکان عنصر ابتدایی صفت است و **REAR** که شامل مکان عنصر انتهایی صفت است پیاده‌سازی می‌شود مگر آن که خلاف آن به صورت صریح یا ضمنی بیان شود. شرط مبین آن است که صفت **FRONT = NULL**.



شکل ۱۶-۶ نمایش یک صفت به وسیله آرایه

شکل ۱۶-۶ روشی را نشان می‌دهد که در آن آرایه شکل ۱۵-۶ با استفاده از آرایه **QUEUE** با **N** عنصر در حافظه ذخیره می‌شود. علاوه براین شکل ۱۶-۶ روشی را نشان می‌دهد که عناصر از صفت حذف می‌شوند و عناصر جدید به صفت اضافه می‌شوند. ملاحظه می‌کنید که هرگاه یک عنصر از صفت حذف شود، مقدار **FRONT** به اندازه ۱ افزایش می‌یابد و این حالت را می‌توان با دستور جایگزینی

FRONT := FRONT + 1

پیاده‌سازی کرد. به همین ترتیب، هرگاه یک عنصر به صفت اضافه شود، مقدار **REAR** به اندازه ۱ افزایش می‌یابد و این حالت را می‌توان با دستور جایگزینی

$$\text{REAR} := \text{REAR} + 1$$

پیاده‌سازی کرد.

معنی آن، این است که پس از N عمل اضافه کردن، عنصر انتهای صفت $\text{QUEUE}[N]$ را اشغال می‌کند یا به بیان دیگر، نهایتاً صفت آخرین قسمت آرایه را اشغال می‌کند. این وضعیت حتی وقتی خود صفت عنصر زیاد نداشته باشد اتفاق می‌افتد.

فرض کنید بخواهیم عنصر ITEM را زمانی که صفت آخرین قسمت آرایه را اشغال کرده است یعنی وقتی $\text{N} = \text{REAR}$ است به صفت اضافه کنیم. یک راه برای انجام این کار، آن است که تمام صفت را به ابتدای آرایه منتقل کنیم و براساس آن FRONT و REAR را تغییر دهیم و آنگاه ITEM را مطابق آنچه که در بالا انجام دادیم به صفت اضافه کنیم. این روش ممکن است پرهزینه و وقت‌گیر باشد. روشنی که ما اتخاذ کرده‌ایم بر این فرض استوار است که QUEUE چرخشی یا حلقوی است یعنی $\text{QUEUE}[1]$ پس از $\text{QUEUE}[N]$ در آرایه قرار گرفته است. با این فرض با جایگزینی ITEM در $\text{QUEUE}[1]$ عنصر ITEM را به صفت اضافه می‌کنیم. به طور مشخص، بجای افزایش REAR به $1 + N$ ، قرار می‌دهیم $\text{REAR} = 1$ و آنگاه جایگزین می‌کنیم:

$$\text{QUEUE}[\text{REAR}] := \text{ITEM}$$

به همین ترتیب، اگر $\text{FRONT} = N$ و یک عنصر از QUEUE حذف شود، بجای افزایش $1 = \text{FRONT}$ قرار می‌دهیم $1 + N$ بعضی از دانشجویان این عملیات را به صورت حساب باقیمانده‌ای مورد توجه قرار می‌دهند که در بخش ۲-۲ بررسی شده است.

فرض کنید صفت تنها از یک عنصر تشکیل شده است یعنی فرض کنید که

$$\text{FRONT} = \text{REAR} \neq \text{NULL}$$

و فرض کنید این عنصر حذف می‌شود. آنگاه جایگزینی‌های زیر را باریم:

$$\text{REAR} := \text{NULL} \quad \text{و} \quad \text{FRONT} := \text{NULL}$$

که بیانگر آن است صفت خالی است.

مثال ۱۰-۶

شکل ۱۷-۶ چگونگی پیاده‌سازی یک صفت را توسط یک آرایه چرخشی QUEUE با $5 = N$ خانه حافظه نشان می‌دهد، ملاحظه می‌کنید که صفت همیشه خانه‌های حافظه متولی را اشغال می‌کند بجز وقتی که، خانه‌هایی در ابتداء و انتهای آرایه را اشغال می‌کند. اگر صفت را به صورت یک آرایه چرخشی در نظر بگیریم معنی آن، این است که همچنان خانه‌های حافظه متولی را اشغال می‌کند. علاوه بر این، همانگونه که در شکل ۱۷-۶ (م) مشاهده می‌شود، صفت تنها وقتی خالی خواهد بود که

و یک عنصر حذف می‌شود. به همین دلیل **FRONT = REAR** در شکل **REAR** در شکل **NULL** است.

۱۷-۶ (م) جایگزین می‌شود:

		QUEUE										
		FRONT: 0 REAR: 0										
(a)	Initially, empty:	<table border="1" style="margin-left: auto; margin-right: auto;"><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr></table>						1	2	3	4	5
1	2	3	4	5								
(b)	A, B and then C inserted:	FRONT: 1 REAR: 3 <table border="1" style="margin-left: auto; margin-right: auto;"><tr><td>A</td><td>B</td><td>C</td><td></td><td></td></tr><tr><td></td><td></td><td></td><td>4</td><td>5</td></tr></table>	A	B	C						4	5
A	B	C										
			4	5								
(c)	A deleted:	FRONT: 2 REAR: 3 <table border="1" style="margin-left: auto; margin-right: auto;"><tr><td></td><td>B</td><td>C</td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td>4</td><td>5</td></tr></table>		B	C			1			4	5
	B	C										
1			4	5								
(d)	D and then E inserted:	FRONT: 2 REAR: 5 <table border="1" style="margin-left: auto; margin-right: auto;"><tr><td></td><td>B</td><td>C</td><td>D</td><td>E</td></tr><tr><td>1</td><td></td><td></td><td>4</td><td>5</td></tr></table>		B	C	D	E	1			4	5
	B	C	D	E								
1			4	5								
(e)	B and C deleted:	FRONT: 4 REAR: 5 <table border="1" style="margin-left: auto; margin-right: auto;"><tr><td></td><td></td><td></td><td>D</td><td>E</td></tr><tr><td>2</td><td>3</td><td></td><td>4</td><td>5</td></tr></table>				D	E	2	3		4	5
			D	E								
2	3		4	5								
(f)	F inserted:	FRONT: 4 REAR: 1 <table border="1" style="margin-left: auto; margin-right: auto;"><tr><td>F</td><td></td><td></td><td>D</td><td>E</td></tr><tr><td></td><td></td><td></td><td>2</td><td>3</td></tr></table>	F			D	E				2	3
F			D	E								
			2	3								
(g)	D deleted:	FRONT: 5 REAR: 1 <table border="1" style="margin-left: auto; margin-right: auto;"><tr><td>F</td><td></td><td></td><td></td><td>E</td></tr><tr><td></td><td></td><td></td><td></td><td>4</td></tr></table>	F				E					4
F				E								
				4								
(h)	G and then H inserted:	FRONT: 5 REAR: 3 <table border="1" style="margin-left: auto; margin-right: auto;"><tr><td>F</td><td>G</td><td>H</td><td></td><td>E</td></tr><tr><td></td><td></td><td></td><td></td><td>4</td></tr></table>	F	G	H		E					4
F	G	H		E								
				4								
(i)	E deleted:	FRONT: 1 REAR: 3 <table border="1" style="margin-left: auto; margin-right: auto;"><tr><td>F</td><td>G</td><td>H</td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td>4</td></tr></table>	F	G	H							4
F	G	H										
				4								
(j)	F deleted:	FRONT: 2 REAR: 3 <table border="1" style="margin-left: auto; margin-right: auto;"><tr><td></td><td>G</td><td>H</td><td></td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td>4</td></tr></table>		G	H			1				4
	G	H										
1				4								
(k)	K inserted:	FRONT: 2 REAR: 4 <table border="1" style="margin-left: auto; margin-right: auto;"><tr><td></td><td>G</td><td>H</td><td>K</td><td></td></tr><tr><td>1</td><td></td><td></td><td></td><td>5</td></tr></table>		G	H	K		1				5
	G	H	K									
1				5								
(l)	G and H deleted:	FRONT: 4 REAR: 4 <table border="1" style="margin-left: auto; margin-right: auto;"><tr><td></td><td></td><td></td><td>K</td><td></td></tr><tr><td>2</td><td>3</td><td></td><td></td><td>5</td></tr></table>				K		2	3			5
			K									
2	3			5								
(m)	K deleted, QUEUE empty:	FRONT: 0 REAR: 0 <table border="1" style="margin-left: auto; margin-right: auto;"><tr><td></td><td></td><td></td><td></td><td></td></tr><tr><td>3</td><td>4</td><td>5</td><td></td><td></td></tr></table>						3	4	5		
3	4	5										

اکنون وقت آن رسیده است تا زیربرنامه **QINSERT** (زیربرنامه ۶.۱۱) را به صورت رسمی بیان کنیم که داده **ITEM** را به صف اضافه می‌کند. از اولین کارهایی که ما در این زیربرنامه انجام می‌دهیم آزمایش و کنترل وضعیت سرریزی است یعنی آزمایش کنیم که آیا صف پر است یا خیر؟ بدنبال آن زیربرنامه **QDELETE** (زیربرنامه ۶.۱۲) را ارائه می‌دهیم که عنصر اول را از صف حذف می‌کند و آن را در متغیر **ITEM** جایگزین می‌کند. از اولین کارهایی که ما در زیربرنامه انجام می‌دهیم آزمایش و کنترل وضعیت زیرریزی است یعنی آزمایش کنیم که آیا صف خالی است یا خیر؟

Procedure 6.11: QINSERT(QUEUE, N, FRONT, REAR, ITEM)
This procedure inserts an element ITEM into a queue.

1. [Queue already filled?] If FRONT = 1 and REAR = N, or if FRONT = REAR + 1, then:
Write: OVERFLOW, and Return.
2. [Find new value of REAR.] If FRONT := NULL, then: [Queue initially empty.] Set FRONT := 1 and REAR := 1.
Else if REAR = N, then:
Set REAR := 1.
Else:
Set REAR := REAR + 1.
[End of If structure.]
3. Set QUEUE[REAR] := ITEM. [This inserts new element.]
4. Return.

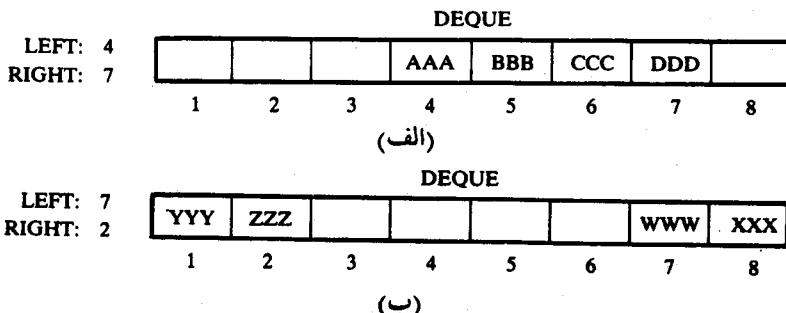
Procedure 6.12: QDELETE(QUEUE, N, FRONT, REAR, ITEM)
This procedure deletes an element from a queue and assigns it to the variable ITEM.

1. [Queue already empty?] If FRONT := NULL, then: Write: UNDERFLOW, and Return.
2. Set ITEM := QUEUE[FRONT].
3. [Find new value of FRONT.] If FRONT = REAR, then: [Queue has only one element to start.] Set FRONT := NULL and REAR := NULL.
Else if FRONT = N, then:
Set FRONT := 1.
Else:
Set FRONT := FRONT + 1.
[End of If structure.]
4. Return.

۱۰-۶ صفحهای دوسره یا DEQUE ها

یک صف دوسره یا یک Deque (به صورت دُک Deque یا دِکیو Dequeue تلفظ کنید) یک لیست خطی است که عناصر را می‌توان در آن از هر دو سر اضافه یا حذف کرد اما حذف یا اضافه کردن عنصر از وسط آن امکان‌پذیر نیست. اصطلاح Deque شکل اختصاری نام صف دوسره Double – Ended Queue است.

روش‌های مختلفی برای نمایش یک Deque در کامپیوتر وجود دارد. فرض می‌کنیم که Deque ما توسط یک آرایه چرخشی DEQUE با اشاره‌گرهای LEFT و RIGHT پیاده‌سازی می‌شود که به دوسر Deque اشاره می‌کند. مگر آن که خلاف آن به صورت صریح یا ضمنی بیان شود. فرض می‌کنیم که عناصر از سر سمت چپ تا سر سمت راست آرایه امتداد دارند. اصطلاح "چرخشی" یا "حلقوی" از این واقعیت گرفته شده است که فرض می‌کنیم عنصر DEQUE[1] پس از DEQUE[N] در آرایه می‌آید. شکل ۱۸-۶ دو Deque را نشان می‌دهد که هر یک با ۴ عنصر در یک آرایه $8 = N$ خانه حافظه‌ای پیاده‌سازی شده‌اند. شرط $LEFT = NULL$ برای این منظور بکار می‌رود تا نشان دهد Deque خالی است. دو نوع Deque وجود دارد که عبارتند از Deque با ورودی محدود شده و یک Deque با خروجی محدود شده که واسطه بین یک Deque و یک صف هستند. یک Deque با ورودی محدود شده یک Deque است که به ما اجازه می‌دهد عمل اضافه کردن را تنها از یک سر لیست انجام دهیم اما در این نوع Deque ها مجاز هستیم از هر دو سر لیست عناصرها را حذف کنیم و یک Deque با خروجی محدود شده یک Deque است که به ما اجازه می‌دهد عمل حذف را تنها از یک سر لیست انجام دهیم اما در این نوع Deque ها مجاز هستیم از هر دو سر لیست عناصرها را اضافه کنیم.



شکل ۱۸-۶

این زیربرنامه‌های Procedure که عمل اضافه کردن و حذف عناصر را در Deque ها انجام می‌دهند و انواع مختلف این‌گونه زیربرنامه‌ها به عنوان مسائل تکمیلی ارائه می‌شوند. همانند صفحه، یک وضع

پیچیده ممکن است اتفاق بیفتد (الف) وقتی که سرریزی روی می‌دهد یعنی وقتی یک عنصر به Deque اضافه می‌شود که از قبل پر است یا (ب) وقتی که زیرریزی روی می‌دهد یعنی وقتی یک عنصر از Deque حذف می‌شود که خالی است. زیربرنامه‌های Procedure باید این حالتهای ممکن را در نظر بگیرید.

۱۱-۶ صفحه‌های اولویت

یک صفحه اولویت، یک مجموعه از عناصر است به طوری که به هر عنصر آن اولویت یکسان داده می‌شود و ترتیبی که در آن عناصر حذف و پردازش می‌شوند از دو قاعدة زیر پیروی می‌کند:

- (۱) عنصری که دارای اولویت بیشتر است قبل از تمام عناصری که اولویت کمتر دارد پردازش می‌شود.
- (۲) دو عنصری که دارای اولویت یکسان هستند با توجه به ترتیبی که به صفحه اضافه شده‌اند پردازش می‌شوند. یک نمونه از صفحه اولویت، یک سیستم اشتراک زمانی است. در این سیستم برنامه‌هایی که دارای اولویت بالاتر هستند اول پردازش می‌شوند و برنامه‌هایی که دارای اولویت یکسان هستند تشکیل یک صفحه استاندارد می‌دهند.

روشهای مختلفی برای پیاده‌سازی صفحه اولویت در حافظه وجود دارد. ما به شرح دو روش در اینجا می‌پردازیم. در یکی از این روشها از یک لیست یکطرفه و در روش دیگر از چند صفحه استفاده می‌شود. واضح است که میزان ساده یا مشکل‌بودن، اضافه کردن یک یا چند عنصر به صفحه اولویت یا حذف از آن به نوع نمایشی که برای صفحه انتخاب می‌کنیم بستگی دارد.

نمایش یک صفحه اولویت به وسیله لیست یکطرفه

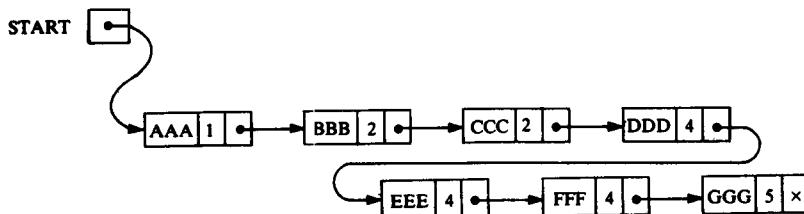
یک راه پیاده‌سازی یک صفحه اولویت در حافظه به وسیله لیست یکطرفه است که به شرح زیر است:
 (الف) هر گره در لیست شامل سه عنصر اطلاعاتی است که عبارتند از: یک فیلد اطلاعاتی INFO، یک عدد اولویت PRN و یک عدد پیوند LINK.

(ب) گره X قبل از گره Y در لیست است (۱) اگر اولویت X بیشتر از اولویت Y باشد یا (۲) اگر هر دو گره اولویت یکسان دارند آنگاه X قبل از Y به لیست اضافه شده باشد، به بیان دیگر ترتیب در لیست یکطرفه منتظر با ترتیب صفحه اولویت است.

اعداد اولویت بر پایه استنباط معمولی مورد استفاده قرار می‌گیرند، یعنی عناصری که عدد اولویت کوچکتری دارند، دارای اولویت بالاتری هستند.

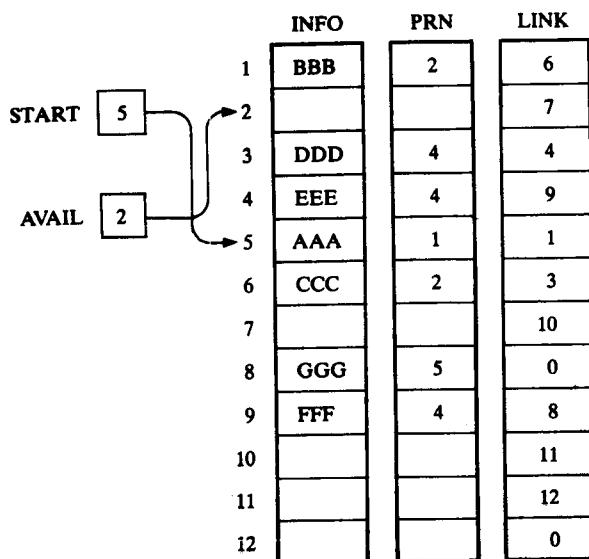
مثال ۱۱-۶

شکل ۱۹-۶ نمودار یک صفحه اولویت را با ۷ عنصر نشان می‌دهد.



شکل ۱۹-۶

نمودار در مورد این که BBB قبل یا بعد از DDD به لیست اضافه شده، چیزی به مانمی‌گوید. از طرف دیگر، از نمودار استنباط می‌شود که BBB قبل از CCC به لیست اضافه شده است، چون BBB و CCC دارای عدد اولویت یکسان هستند و BBB قبل از CCC در لیست ظاهر شده است. شکل ۲۰-۶ روشی را نشان می‌دهد که صفت اولویت با استفاده از آرایه‌های خطی INFO و PRN و LINK (بخش ۵-۲) ببینید. در حافظه ظاهر می‌شود.



شکل ۲۰-۶

خاصیت اصلی نمایش یک صف اولویت با استفاده از لیست یکطرفه، آن است که عنصر داخل صف که همواره در ابتدای لیست یکطرفه ظاهر می‌شوند باید اول پردازش شوند. بنابراین عمل حذف یا اضافه کردن عنصر در صف اولویت بسیار ساده است. شرح این الگوریتم به صورت زیر است:

Algorithm 6.13: This algorithm deletes and processes the first element in a priority queue which appears in memory as a one-way list.

1. Set ITEM := INFO[START]. [This saves the data in the first node.]
2. Delete first node from the list.
3. Process ITEM.
4. Exit.

این الگوریتم اولین عنصر را از یک صف اولویت، که در حافظه به صورت لیست یکطرفه ظاهر شده است. حذف می‌کند و آنرا پردازش می‌کند.

جزئیات این الگوریتم، به همراه احتمال وقوع زیرریزی به عنوان تمرین به دانشجو و اگذار می‌شود. اضافه کردن یک عنصر به صف اولویت بسیار پیچیده‌تر از حذف یک عنصر از صف است چون برای اضافه کردن یک عنصر به صف احتیاج است جای درست آن را در صف پیدا کنیم. شرح این الگوریتم به صورت زیر است:

Algorithm 6.14: This algorithm adds an ITEM with priority number N to a priority queue which is maintained in memory as a one-way list.

این الگوریتم عنصر ITEM با عدد اولویت N را در صف اولویت، که در حافظه به صورت لیست یکطرفه پیاده‌سازی شده است، اضافه می‌کند.
(الف) تا پیداشدن گره X که عدد اولویت آن بزرگتر از N است. لیست یکطرفه را پیمایش کنید ITEM را در جلوی گره X اضافه کنید.

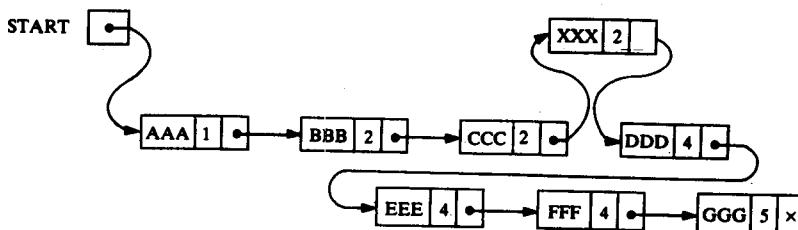
(ب) اگر چنین گره‌ای پیدا نشد، ITEM را به عنوان آخرین عنصر لیست اضافه کنید.
الگوریتم اضافه کردن بالا را می‌توان به صورت یک شیئی وزن داده شده که در لابلای عنصرها فرو می‌رود به تصویر درآورد تا به عنصری با وزن بیشتر برخورد کند.

جزئیات الگوریتم بالا به عنوان تمرین به دانشجو و اگذار می‌شود. مشکل اصلی الگوریتم، از این واقعیت ناشی شده است که ITEM قبل از گره X اضافه می‌شود. به بیان دیگر هنگام پیمایش لیست، باید آدرس گره‌ای را که قبل از گره پردازش شونده قرار دارد نگهداریم.

مثال ۶-۱۲

صف اولویت شکل ۶-۱۹ را در نظر بگیرید. فرض کنید عنصر XXX با عدد اولویت 2 را می‌خواهیم به صف اضافه کنیم. با مقایسه اعداد اولویت، لیست را پیمایش می‌کنیم. ملاحظه می‌کنید که

عنصر اول لیست است که عدد اولویت آن بزرگتر از عدد اولویت XXX است. از این‌رو به صورتی که در شکل ۶-۲۱ نشان داده شده است XXX در جلوی DDD در لیست اضافه می‌شود. ملاحظه می‌کنید که XXX پس از BBB و CCC ظاهر شده است که اولویت یکسان با XXX دارند. حال فرض کنید بخواهیم یک عنصر را از صف حذف کنیم. این عنصر AAA خواهد بود. که عنصر اول لیست است. با این فرض که هیچ عنصری به صف اضافه نمی‌شود، عنصر بعدی که حذف می‌شود BBB خواهد بود. آنگاه CCC و بدنبال آن XXX و الی آخر حذف خواهند شد.



شکل ۶-۲۱

نمایش یک صف اولویت با استفاده از آرایه

یک روش دیگر برای پیاده‌سازی یک صف اولویت در حافظه، استفاده از یک صف جداگانه برای هر سطح از اولویت (یا برای هر عدد اولویت) است. هر یک از این صفحات در آرایه چرخشی یا حلقوی مربوط به خود ظاهر می‌شود و باید جفت اشاره‌گر FRONT و REAR مختص خود را داشته باشند. درواقع چنانچه به هر صف مقدار مساوی اختصاص یابد، یک آرایه دو بعدی QUEUE را می‌توان بجای آرایه‌های خطی مورد استفاده قرار داد. شکل ۶-۲۲ این‌گونه نمایش صف اولویت را برای شکل ۶-۲۱ نشان می‌دهد.

	FRONT	REAR		1	2	3	4	5	6
1	2	2		AAA					
2	1	3		BBB	CCC	XXX			
3	0	0							
4	5	1		FFF			DDD	EEE	
5	4	4					GGG		

شکل ۶-۲۲

ملاحظه می‌کنید که $\text{FRONT}[K]$ و $\text{REAR}[K]$ به ترتیب شامل عناصر ابتدا و انتهای سطر K امصف هستند. سطري که عناصر در آن دارای عدد اولویت K هستند. در زیر شرحهایی از الگوریتم‌های حذف و اضافه کردن عناصر در یک صف اولویت ارائه شده است که مانند بالا به وسیله یک آرایه دو بعدی **QUEUE** پیاده‌سازی می‌شود. جزئیات این الگوریتم‌ها به عنوان تمرین به داشجو و آگذار می‌شود.

Algorithm 6.15: This algorithm deletes and processes the first element in a priority queue maintained by a two-dimensional array **QUEUE**.

1. [Find the first nonempty queue.]
Find the smallest K such that $\text{FRONT}[K] \neq \text{NULL}$.
2. Delete and process the front element in row K of **QUEUE**.
3. Exit.

این الگوریتم عنصر اول را از یک صف اولویت که به صورت آرایه دو بعدی **QUEUE** پیاده‌سازی شده است، حذف و آن را پردازش می‌کند.

Algorithm 6.16: This algorithm adds an ITEM with priority number M to a priority queue maintained by a two-dimensional array **QUEUE**.

1. Insert ITEM as the rear element in row M of **QUEUE**.
2. Exit.

این الگوریتم عنصر **ITEM** با عدد اولویت M را به یک صف اولویت که به صورت آرایه دو بعدی **QUEUE** پیاده‌سازی شده است اضافه می‌کند.

خلاصه مطالب فصل

هنگام انتخاب ساختمان داده‌های مختلف برای یک مسئله معین، بار دیگر توازن بین زمان و حافظه را ملاحظه کردید. نمایش یک صف اولویت به وسیله آرایه، از نظر زمان بسیار کاراتر از لیست یکطرفه است. علت آن هم این است که هنگام اضافه کردن یک عنصر به لیست یکطرفه، باید یک جستجوی خطی در لیست انجام داد. از طرف دیگر، نمایش صف اولویت به وسیله لیست یکطرفه می‌تواند از نظر مصرف حافظه بسیار کاراتر از نمایش آرایه‌ای صف اولویت باشد، چون هنگام استفاده از نمایش آرایه‌ای، وقتی تعداد عناصر یک سطح اولویت از ظرفیت آن سطح بزرگتر می‌شود سرربیزی اتفاق می‌افتد اما هنگام استفاده از لیست یکطرفه، سرربیزی تنها وقتی اتفاق می‌افتد که تعداد کل عناصر بزرگتر از ظرفیت کل می‌شود. یک روش دیگر، استفاده از لیست پیوندی برای هر سطح اولویت است.

مسأله‌های حل شده

پشته‌ها

مسئله ۱-۶: پشته کاراکترهای زیر را درنظر بگیرید که در آن $STACK$ به $N = 8$ خانه حافظه اختصاص داده می‌شود:

$STACK: A, C, D, F, K, _, _, _$

جهت سهولت در نمادگذاری از "—" برای نمایش خانه حافظه خالی استفاده می‌کنیم. وقتی عملیات زیر انجام می‌شود وضعیت پشته را بیان کنید:

POP(STACK, ITEM)	(۵)	POP(STACK, ITEM)	(الف)
PUSH(STACK, R)	(و)	POP(STACK, ITEM)	(ب)
PUSH(STACK, S)	(ز)	PUSH(STACK, L)	(ج)
POP(STACK, ITEM)	(ح)	PUSH(STACK, P)	(د)

حل: زیربرنامه POP همیشه عنصر بالای پشته را حذف می‌کند و زیربرنامه PUSH همیشه عنصر جدید به بالای پشته اضافه می‌کند. بنابراین:

STACK: A, C, D, L, <_, _, _, _	(۵)	STACK: A, C, D, F, <_, _, _, _	(الف)
STACK: A, C, D, L, R, <_, _, _	(و)	STACK: A, C, D, <_, _, _, _	(ب)
STACK: A, C, D, L, R, S, <_, _	(ز)	STACK: A, C, D, L, <_, _, _	(ج)
STACK: A, C, D, L, R, <_, _	(ح)	STACK: A, C, D, L, P, <_, _	(د)

مسئله ۲-۶: داده‌های مسئله ۱-۶ را درنظر بگیرید. (الف) چه وقت سرریزی اتفاق می‌افتد؟ (ب) چه وقت C قبل از D حذف خواهد شد؟

حل: (الف) چون به $STACK$ به $N = 8$ خانه حافظه اختصاص داده شده است، سرریزی وقتی اتفاق می‌افتد که $STACK$ حاوی 8 عنصر باشد و یک عمل PUSH برای اضافه کردن عنصر دیگر در $STACK$ داشته باشیم.

(ب) چون $STACK$ به صورت یک پشته پیاده‌سازی شده است، C هرگز قبل از D حذف نمی‌شود.

مسئله ۳-۶: پشته زیر را درنظر بگیرید که در آن به $STACK$ به $N = 6$ خانه حافظه اختصاص داده شده است.

$STACK: AAA, DDD, EEE, FFF, GGG, _$

وقتی عملیات زیر انجام می‌شود وضعیت پشته را شرح دهید:

PUSH(STACK, SSS), (د) PUSH(STACK, KKK), (الف)

POP(STACK, ITEM), (ب) POP(STACK, ITEM), (ب)

PUSH(STACK, TTT), (ج) PUSH(STACK, LLL), (ج)

حل: (الف) KKK به بالای پشته STACK اضافه می‌شود، درنتیجه:

STACK: AAA, DDD, EEE, FFF, GGG, KKK

(ب) عنصر بالا از STACK حذف می‌شود، درنتیجه:

STACK: AAA, DDD, EEE, FFF, GGG, —

(ج) LLL به بالای STACK اضافه می‌شود، درنتیجه:

STACK: AAA, DDD, EEE, FFF, GGG, LLL

(د) سریزی اتفاق می‌افتد چون STACK پر است و عنصر دیگر SSS به STACK اضافه می‌شود.

هیچ یک از عملیات دیگر تا وقتی مسأله سریزی حل نشود نمی‌توانند انجام شوند. عمل سریزی را مثلاً می‌توان با افزودن حافظه اضافی به STACK حل کرد.

مسأله ۴-۶: فرض کنید به STACK $n = 6$ خانه حافظه اختصاص داده شده است و در ابتدا STACK خالی است یا به بیان دیگر $TOP = 0$. خروجی قطعه برنامه زیر را تعیین کنید.

1. Set AA/ and BBB := 5.
2. Call PU STACK, AAA).
- Call PU STACK, 4).
- Call PUSH(STACK, BBB + 2).
- Call PUSH(STACK, 9).
- Call PUSH(STACK, AAA + BBB).
3. Repeat while $TOP \neq 0$:
 - Call POP(STACK, ITEM).
 - Write: ITEM.
- [End of loop.]
4. Return.

مرحله ۱ . قرار دهید $BBB = 5$ و $AAA = 2$.

مرحله ۲ . Push AAA + BBB = 7 را در پشته STACK را در پشته Push کنید. نتیجه می‌شود:

STACK: 2, 4, 7, 9, 7, —

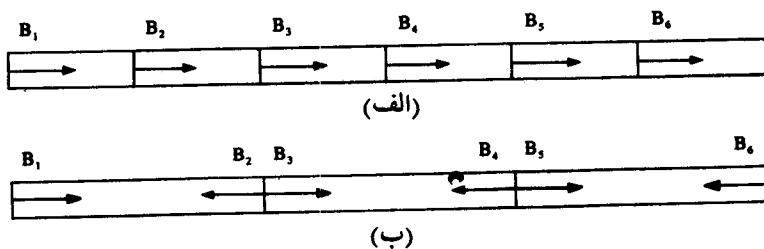
مرحله ۳ . عنصرهای STACK را تا وقتی خالی نشده است POP کنید و چاپ نمائید. چون عنصر بالا همیشه POP می‌شود، خروجی از دنباله زیر تشکیل می‌شود:

7, 9, 7, 4, 2

ملاحظه می‌کنید که دنباله بالا به ترتیب عکس عناصری هستند که به STACK اضافه شده‌اند.

مسأله ۵-۶: فرض کنید به $K = 6$ پشته فضای معلوم S که از N خانه حافظه هم‌جوار تشکیل شده، اختصاص داده شده است. روش‌های نگهداری پشته‌ها را در S شرح دهید.

حل: فرض کنید هیچ اطلاعاتی از قبل در دست نیست تا بیان کند یک پشته خیلی سریع تراز پشته دیگر رشد می‌کند. آنگاه می‌توان K / N خانه برای هر پشته در نظر گرفت این عمل در شکل ۶-۲۳ (الف) نشان داده شده است که در آن B_1, B_2, \dots, B_6 به ترتیب عناصر پائین پشته‌ها را نمایش می‌دهند. به عبارت دیگر می‌توان پشته‌ها را به دو قسمت تقسیم کرد و $K / 2N$ خانه حافظه برای هر جفت پشته به صورتی که در شکل ۶-۲۳ (ب) نشان داده شده است اختیار کرد. روش دوم می‌تواند تعداد دفعات وقوع سریزی را کاهش دهد.



شکل ۶-۲۳

نمادگذاری لهستانی

مسأله ۶-۶: با ملاحظه و بررسی و روش دستی هر عبارت میانوندی زیر را به عبارت پسوندی معادل آن تبدیل کنید:

$$(A + B \uparrow D) / (E - F) + G \quad (\text{ب}) \qquad (A - B) * (D/E) \quad (\text{الف})$$

$$A * (B + D) / E - F * (G + H/K) \quad (\text{ج})$$

حل: با استفاده از ترتیبی که با آن عملگرها اجرا می‌شوند، هر عملگر را از نماد میانوندی به پسوندی تبدیل می‌کنیم. از کروشه باز و بسته [] برای نمایش بخشی از تبدیل انجام شده در هر مرحله استفاده می‌کنیم.

$$(A - B) * (D/E) = [AB-] * [DE/] = AB - DE / * \quad (\text{الف})$$

$$\begin{aligned} (A + B \uparrow D) / (E - F) + G &= (A + [BD\uparrow]) / [EF-] + G = [ABD\uparrow+] / [EF-] + G \\ &= [ABD\uparrow + EF - /] + G = ABD\uparrow + EF - / G + \end{aligned} \quad (\text{ب})$$

$$\begin{aligned}
 A * (B + D) / E - F * (G + H / K) &= A * [BD+] / E - F * (G + [HK/]) \\
 &= [ABD+*] / E - F * [GHK/+] \\
 &= [ABD++*E/] - [FGHK/+*] \\
 &= ABD++*E/[FGHK/+*]
 \end{aligned} \tag{ج}$$

ملاحظه می‌کنید که تا وقتی عملوندها روی هم نیفتاده باشند ما در یک مرحله بیش از یک عملگر را به صورت پسوندی تبدیل کردیم.

مسئله ۷-۶: عبارت محاسباتی P زیر را که با نماد پسوندی نوشته شده است در نظر بگیرید:

$$P: 12, 7, 3, -, /, 2, 1, 5, +, *, +$$

(الف) با بررسی و روش دستی P را به عبارت میانوندی معادل آن تبدیل کنید.

(ب) عبارت میانوندی را ارزیابی کنید.

حل: (الف) عمل جستجو و خواندن را از چپ به راست انجام می‌دهیم، هر عملگر را از نماد پسوندی به میانوندی تبدیل می‌کنیم. از کروشه باز و بسته ۱ برای نمایش بخشی از تبدیل انجام شده در هر مرحله استفاده می‌کنیم.

$$\begin{aligned}
 P &= 12, [7 - 3], /, 2, 1, 5, +, *, + \\
 &= [12 / (7 - 3)], 2, 1, 5, +, *, + \\
 &= [12 / (7 - 3)], 2, [1 + 5], *, + \\
 &= [12 / (7 - 3)], [2 * (1 + 5)], + \\
 &= 12 / (7 - 3) + 2 * (1 + 5)
 \end{aligned}$$

(ب) با استفاده از عبارت میانوندی، به دست می‌آید:

$$P = 12 / (7 - 3) + 2 * (1 + 5) = 12 / 4 + 2 * 6 = 3 + 12 = 15$$

مسئله ۸-۶: عبارت پسوندی P مسئله ۷-۶ را در نظر بگیرید. با استفاده از الگوریتم 6.3، P را ارزیابی کنید.

حل: نخست یک پرانتز بسته نگهبان در انتهای P اضافه می‌کنیم به دست می‌آید:

$$P: 12, 7, 3, -, /, 2, 1, 5, +, *, +,)$$

عمل جستجو و خواندن P را از چپ به راست انجام می‌دهیم. اگر با یک مقدار ثابت روبرو شویم، آن را در پشتہ قرار می‌دهیم اما اگر با یک عملگر روبرو شویم دو مقدار ثابت بالای پشتہ را با آن عملگر ارزیابی می‌کنیم. شکل ۲۴-۶ محتوی STACK را به م Hassan جستجو و خوانده شدن هر عنصر P نشان می‌دهد.

Symbol	STACK
12	12
7	12, 7
3	12, 7, 3
-	12, 4
/	3
2	3, 2
1	3, 2, 1
5	3, 2, 1, 5
+	3, 2, 6
*	3, 12
+	15
)	15

شکل ۶-۲۴

عدد آخر یعنی 15 در پشتة STACK، وقتی پرانتز بسته نگهبان خوانده می‌شود مقدار عبارت P است. این مقدار با نتیجه مسأله ۶-۷ (ب) سازگار است.
مسأله ۶-۹: عبارت میانوندی زیر Q را درنظر بگیرید:

$$Q: ((A + B) * D) \uparrow (E - F)$$

با استفاده از الگوریتم ۶.۴، Q را به عبارت P پسوندی معادل آن تبدیل کنید.

حل: نخست یک پرانتز باز داخل پشتة STACK، Push می‌کنیم و آنگاه درانتهای Q یک پرانتز بسته اضافه می‌کنیم به دست می‌آید:

$$Q: (((A + B) * D)) \uparrow (E - F)$$

توجه دارید که اکنون Q شامل ۱۶ عنصر است. Q را از چپ به راست می‌خوانیم. یادآوری می‌کنیم که

(۱) اگر به یک مقدار ثابت برخورد کنیم آن را به P اضافه می‌کنیم؛

(۲) اگر به یک پرانتز باز برخورد کنیم آن را به داخل پشتة Push می‌کنیم (۳) اگر به یک عملگر برخورد کنیم آن را در سطح خودش پائین می‌بریم و (۴) اگر به یک پرانتز بسته برخورد کنیم آن را با اولین پرانتز باز فرو می‌بریم. شکل ۶-۲۵ تصویرهای STACK و رشتة P را به محض خوانده شدن هر عنصر Q نشان می‌دهد.

Symbol	STACK	Expression P
(((
((((
A	(((A
+	(((+	A
B	(((+	A B
)	((A B +
*	((*	A B +
D	((*	A B + D
)	(A B + D *
↑	(↑	A B + D *
((↑ (A B + D *
E	(↑ (A B + D * E
-	(↑ (-	A B + D * E
F	(↑ (-	A B + D * E F
)	(↑	A B + D * E F -
)		A B + D * E F - ↑

شکل ۶-۲۵

هنگامی که STACK خالی است، پرانتز بسته نهایی خوانده می‌شود و نتیجه چنین است :

$$P: A B + D * E F - \uparrow$$

که نماد پسوندی موردنظر معادل Q است.

مسئله ۱۰-۶: با بررسی و روش دستی، هر عبارت میانوندی زیر را به عبارت پیشوندی معادل آن تبدیل کنید.

$$(A - B) * (D / E) \quad (الف)$$

$$(A + B \uparrow D) / (E - F) + G \quad (ب)$$

آیا هیچ رابطه‌ای بین عبارتهای پیشوندی و عبارتهای پسوندی معادل آن که در مسئله ۶-۶ به دست آوردید وجود دارد؟

حل : با استفاده از ترتیبی که با آن عملگرها اجرا می‌شوند، هر عملگر را از نماد میانوندی به نماد پیشوندی تبدیل می‌کنیم.

$$(A - B) * (D / E) = [-AB] * [/DE] = * - A B / D E \quad (الف)$$

$$(A + B \uparrow D) / (E - F) + G = (A + [\uparrow BD]) / [-EF] + G \quad (ب)$$

$$\begin{aligned} &= [+A\uparrow BD] / [-EF] + G \\ &= [/+A\uparrow BD - EF] + G \\ &= + / + A \uparrow B D - E F G \end{aligned}$$

عبارت پیشوندی عکس عبارت پسوندی نیست. با وجود این، ترتیب عملوندهای D, B, A و E در قسمت (الف) و G, A, B, D, E, F در قسمت (ب) در هر سه عبارت پیشوندی، پسوندی و میانوندی یکسان است.

QUICKSORT

مسئله ۱۱-۶: فرض کنید S لیست زیر، از ۱۴ کاراکتر الفبایی تشکیل شده است:

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

فرض کنید کاراکترهای S به صورت الفبایی مرتب باشند. با استفاده از الگوریتم QuickSort مکان نهایی کاراکتر اول D را پیدا کنید.

حل: کار را با کاراکتر آخر S شروع می‌کنیم. لیست را از راست به چپ جستجو و می‌خوانیم تا کاراکتری را پیدا کنیم که از نظر الفبایی قبل از D قرار دارد. این کاراکتر C است. جای D و C را عوض می‌کنیم، لیست زیر بدست می‌آید:

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

کار را با C شروع می‌کنیم، لیست را به طرف D یعنی از چپ به راست می‌خوانیم تا کاراکتری را پیدا کنیم که از نظر الفبایی بعد از D قرار دارد. این کاراکتر T است. جای D و T را عوض می‌کنیم، لیست زیر بدست می‌آید:

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

کار را با T شروع می‌کنیم، لیست را به طرف D می‌خوانیم تا کاراکتری را پیدا کنیم که قبل از D قرار دارد. این کاراکتر A است. جای D و A را عوض می‌کنیم، لیست زیر به دست می‌آید:

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

کار را با A شروع می‌کنیم، لیست را به طرف D می‌خوانیم تا کاراکتری را پیدا کنیم که بعد از D قرار دارد. چنین حرفی وجود ندارد. معنی آن، این است که D در مکان نهایی اش است. علاوه بر این، حرфهای قبل از D تشکیل لیست کوچکی از تمام حرفهایی را می‌دهند که از نظر الفبایی قبل از D هستند همچنین حرفهای بعد از D تشکیل لیست کوچکی از تمام حرفهایی را می‌دهند که از نظر الفبایی بعد از D هستند، به صورت زیر:

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

Sublist

Sublist

اکنون مرتب کردن S به مرتب کردن هر یک از دو لیست کوچک تبدیل می‌شود.
مسئله ۱۲ - ۶: فرض کنید S از $5 = n$ حرف زیر تشکیل شده باشد:

(A) B C D (E)

C تعداد مقایسه‌هایی را تعیین کنید که برای مرتب کردن S با روش QuickSort لازم است. در صورت وجود، چه نتیجه‌کلی می‌توان از آن بدست آورد؟

حل: کار را با E شروع می‌کنیم برای این که بفهمیم حرف اول تاکنون در جای درستش قرار گرفته است یا n . به $4 = n - 1$ مقایسه احتیاج است. اکنون مرتب کردن S به مرتب کردن لیست کوچک زیر با $4 = n - 1$ حرف منتهی می‌شود.

A (B) C D (E)

کار را با E شروع می‌کنیم، برای اینکه بفهمیم حرف اول B در لیست کوچک از قبل در جای درستش قرار گرفته است یا نه، به $3 = n - 2$ مقایسه احتیاج است. اکنون مرتب کردن S به مرتب کردن لیست کوچک زیر با $3 = n - 2$ حرف منتهی می‌شود. به همین ترتیب برای این که بفهمیم حرف C در جای درستش قرار گرفته است به $2 = n - 3$ مقایسه احتیاج است و برای اینکه بفهمیم حرف D در جای درستش قرار گرفته است به $1 = n - 4$ مقایسه احتیاج است. از آنجاکه تنها یک حرف باقی می‌ماند لیست اکنون مرتب شده است. روی هم رفته تعداد:

$$\text{مقایسه } C = 4 + 3 + 2 + 1 = 10$$

داریم. به همین ترتیب با استفاده از QuickSort

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

مقایسه احتیاج است تا یک لیست n عنصری را وقتی از قبل مرتب است مرتب کند. نشان داده می‌شود که این حالت، بدترین حالت را برای QuickSort تشکیل می‌دهد.

مسئله ۱۳ - ۶: الگوریتم QuickSort را درنظر بگیرید. (الف) آیا می‌توان آرایه‌های LOWER و UPPER را به عوض پشته به صورت صف پیاده‌سازی کرد؟ چرا؟ (ب) برای الگوریتم Quicksort به چه مقدار حافظه اضافی موردنیاز است یا به عبارت دیگر، پیچیدگی الگوریتم از نظر حافظه چقدر است؟

حل: (الف) از آنجاکه اهمیت ندارد زیرمجموعه‌ها با چه ترتیبی مرتب می‌شوند، از این‌رو LOWER و UPPER را می‌توان به عوض پشته با صف یا حتی با صف دوسره یا Deque پیاده‌سازی کرد.

(ب) الگوریتم QuickSort یک الگوریتم "درجای" است یعنی اینکه عناصر به استثنای حالتی که جایشان عوض می‌شوند در جای خودشان باقی می‌مانند. اساساً حافظه اضافی برای پشتدها LOWER و UPPER موردنیاز است. در حالت میانگین، مقدار حافظه اضافی موردنیاز برای این الگوریتم متناسب با $\log n$ است که در آن n تعداد عناصر مرتب شده است.

زیربرنامه بازگشته

مسئله ۱۴ - ۶: فرض کنید a و b نمایش دو عدد صحیح مثبت باشند. فرض کنید تابع Q به شکل زیر به صورت بازگشته تعریف شده است:

$$Q(a, b) = \begin{cases} 0 & \text{if } a < b \\ Q(a - b, b) + 1 & \text{if } a \leq b \end{cases}$$

(الف) مقدار $Q(2, 3)$ و $Q(14, 3)$ را پیدا کنید.

(ب) این تابع چه عملی انجام می‌دهد؟ مقدار $Q(5861, 7)$ را پیدا کنید.

حل : (الف) $2 < 3$ $Q(2, 3) = 0$

$$\begin{aligned} Q(14, 3) &= Q(11, 3) + 1 \\ &= [Q(8, 3) + 1] + 1 = Q(8, 3) + 2 \\ &= [Q(5, 3) + 1] + 2 = Q(5, 3) + 3 \\ &= [Q(2, 3) + 1] + 3 = Q(2, 3) + 4 \\ &= 0 + 4 = 4 \end{aligned}$$

(ب) هر بار که b از a کم می‌شود مقادیر Q یک واحد افزایش می‌یابد. از این‌رو $Q(a, b)$ وقتی a بر b تقسیم می‌شود خارج قسمت را پیدا می‌کند. بنابراین

$$Q(5861, 7) = 837$$

مسئله ۱۵ - ۶: فرض کنید n یک عدد صحیح مثبت باشد. فرض کنید تابع بازگشته L به صورت زیر تعریف شده است.

$$L(n) = \begin{cases} 0 & \text{if } n = 1 \\ L(\lfloor n/2 \rfloor) + 1 & \text{if } n > 1 \end{cases}$$

در اینجا $\lfloor K \rfloor$ "کف floor" عدد K را نشان می‌دهد و بزرگترین عدد صحیحی است که بزرگتر از K نباشد. بخش ۲ - ۲ را ببینید.

(الف) مقدار $L(25)$ را بدست آورید.

(ب) این تابع چه عملی انجام می‌دهد؟

$$\begin{aligned}
 L(25) &= L(12) + 1 \\
 &= [L(6) + 1] + 1 = L(6) + 2 \\
 &= [L(3) + 1] + 2 = L(3) + 3 \\
 &= [L(1) + 1] + 3 = L(1) + 4 \\
 &= 0 + 4 = 4
 \end{aligned} \tag{الف}$$

(ب) هر بار که n بر 2 تقسیم می‌شود مقدار L یک واحد افزایش می‌یابد. از این‌رو L بزرگترین عدد صحیح است که

$$2^L \leq n$$

بنابراین تابع

$$L = \lfloor \log_2 n \rfloor$$

را به دست می‌آورد.

مسئله ۱۶-۶: فرض کنید اعداد فیبوناچی $89 = F_{11}$ و $144 = F_{12}$ داده شده‌اند.

(الف) برای محاسبه F_{16} آیا باید از روش بازگشتی استفاده کنیم یا روش تکرار؟ مقدار F_{16} را پیدا کنید.

(ب) یک زیربرنامه **Procedure** با روش تکرار بنویسید تا نخستین N عدد فیبوناچی $F[N], F[2], F[1], \dots, F[1]$ را به دست آورد که در آن $N > 2$ این زیربرنامه را با زیربرنامه بازگشتی 6.8 مقایسه کنید.

حل : (الف) به عوض استفاده از روش بازگشتی (که در آن عمل ارزیابی از بالا به پائین انجام می‌شود) بهتر است اعداد فیبوناچی با روش تکرار بدست آید (یعنی عمل ارزیابی از پائین به بالا انجام شود). یادآوری می‌کنیم که هر عدد فیبوناچی مجموع دو عدد فیبوناچی قبل از خودش است. با شروع از F_{11} و

F_{12} داریم :

$$F_{13} = 89 + 144 = 233, \quad F_{14} = 144 + 233 = 377, \quad F_{15} = 233 + 377 = 610$$

و درنتیجه :

$$F_{16} = 377 + 610 = 987$$

Procedure P6.16: FIBONACCI(F, N)

(ب)

این زیربرنامه نخستین N عدد فیبوناچی را پیدا می‌کند و آنها را در آرایه F قرار می‌دهد.

1. Set $F[1] := 1$ and $F[2] := 1$.
2. Repeat for $L = 3$ to N :
 - Set $F[L] := F[L - 1] + F[L - 2]$.
 - [End of loop.]
3. Return.

تأکید می‌کنیم که این زیربرنامه با روش تکرار کاراتر از زیربرنامه بازگشتی 6.8 است.

مسئله ۱۷-۶: با استفاده از تعریف تابع آکرمان (تعریف ۳-۶)، $A(1, 3)$ را بدست آورید.

حل : ما ۱۵ مرحله زیر را داریم :

- (1) $A(1, 3) = A(0, A(1, 2))$
- (2) $A(1, 2) = A(0, A(1, 1))$
- (3) $A(1, 1) = A(0, A(1, 0))$
- (4) $A(1, 0) = A(0, 1)$
- (5) $A(0, 1) = 1 + 1 = 2$
- (6) $A(1, 0) = 2$
- (7) $A(1, 1) = A(0, 2)$
- (8) $A(0, 2) = 2 + 1 = 3$
- (9) $A(1, 1) = 3$
- (10) $A(1, 2) = A(0, 3)$
- (11) $A(0, 3) = 3 + 1 = 4$
- (12) $A(1, 2) = 4$
- (13) $A(1, 3) = A(0, 4)$
- (14) $A(0, 4) = 4 + 1 = 5$
- (15) $A(1, 3) = 5$

حالت پلهای به طرف جلو بیان‌گر آن است که یک ارزیابی را به تعویق انداختیم و می‌خواهیم از تعریف استفاده کنیم و حالت پلهای به عقب بیان‌گر آن است که می‌خواهیم از انتها به ابتدا برگردیم و ملاحظه می‌کنید که از فرمول اول تعریف ۶.۳ در مرحله‌های ۵، ۸، ۱۱ و ۱۴ استفاده شده است و از فرمول دوم در مرحله ۴ و از فرمول سوم در مرحله‌های ۱، ۲ و ۳ استفاده شده است. در مراحل دیگر با جایگزینی‌ها به ابتدا و عقب برمی‌گردیم.

مسئله ۱۸ - ۶ : فرض کنید یک زیربرنامه بازگشته P تنها یک بار فراخوانی بازگشته دارد :

Step K. Call P.

دلیلی ارائه دهید که چرا به پشتة STADD (برای آدرس‌های بازگشته) احتیاج نیست.

حل : از آنجا که تنها یک فراخوانی بازگشته وجود دارد کنترل کار همیشه به مرحله $K+1$ برای یک بازگشت Return منتقل می‌شود، به استثنای بازگشت نهایی که به برنامه اصلی است. بنابراین به عوض نگهداری پشتة STADD (و متغیر محلی ADD) فقط به جای

(c) Go to Step K + 1

در تبدیل "Step J. Return" می‌نویسیم. (بخش ۶ را ببینید) :

(c) Go to Step ADD

مسئله ۱۹ - ۶ : راه حل مسئله برجهای هانوی را به گونه‌ای بازنویسی کنید که در آن به جای دو فراخوانی بازگشته تنها از یک فراخوانی استفاده شود.

حل : یک راه آن است که میله‌های A و B را به صورت متقارن درنظر بگیریم، یعنی مراحل زیر را بکار

بندیم:

تعداد ۱ - N دیسک را از A به B منتقل کنید و آنگاه C \rightarrow A را بکار بندید.

تعداد ۲ - N دیسک را از B به A منتقل کنید و آنگاه C \rightarrow B را بکار بندید.

تعداد ۳ - N دیسک را از A به B منتقل کنید و آنگاه C \rightarrow A را بکار بندید.

تعداد ۴ - N دیسک را از B به A منتقل کنید و آنگاه C \rightarrow B را بکار بندید.

و الى آخر. بنابراین می‌توانیم تنها یک فراخوانی بازگشته را تکرار کنیم که پس از هر تکرار جای BEG و AUX را عوض می‌کند به صورت زیر:

Procedure P6.19: TOWER(N, BEG, AUX, END)

1. If $N = 0$, then: Return.
2. Repeat Steps 3 to 5 for $K = N, N - 1, N - 2, \dots, 1$.
3. Call TOWER($K - 1$, BEG, END, AUX).
4. Write: BEG \rightarrow END.
5. [Interchange BEG and AUX.]
Set TEMP := BEG, BEG := AUX, AUX := TEMP.
[End of Step 2 loop.]
6. Return.

ملاحظه می‌کنید که ما در اینجا به جای $1 = N$ از $N = N$ به عنوان مقدار پایه برای بازگشت استفاده می‌کنیم.

مسئله ۲۰ - ۶ : پیاده‌سازی پشتنهای الگوریتم بخش ۸-۶ را برای تبدیل یک زیربرنامه بازگشته به یک برنامه غیربازگشته درنظر بگیرید. یادآوری می‌کنیم که در زمان فراخوانی بازگشته به جای آدرس بازگشت جاری، آدرس بازگشت جدید را به داخل پشتنه Push ، STADD می‌کنیم.

فرض کنید خواسته باشیم آدرس بازگشت جاری را به داخل پشتنه Push ، STADD کنیم. (در بسیاری از متنها و کتابها اینگونه عمل می‌کنند) چه تغییری لازم است در الگوریتم تبدیل داده شود؟

حل: تغییر اصلی عبارت است از آن که در زمان بازگشت به سطح اجرای قبلی، مقدار جاری ADD مکان بازگشت را تعیین می‌کند نه مقدار ADD را پس از POP شدن مقادیر پشتنه‌ها. بنابراین مقدار ADD باید با دستور $SAVE := ADD$ نگهداری شود، آنگاه مقادیر پشتنه‌ها POP می‌شوند و بدنبال آن کنترل به مرحله $SAVE$ می‌شود. تغییر دیگر آن است که باید از ابتدا دستور جایگزینی Main = ADD را داشته باشیم و وقتی $ADD = Main$ بـه برنامه فراخوانده اصلی بازگشت Return کنیم نه وقتی که پشتنه‌ها خالی هستند. الگوریتم به صورت رسمی به شرح زیر است:

(۱) "آماده‌سازی"

(الف) یک پشتنه STPAR برای هر پارامتر PAR ، یک پشتنه STVAR برای هر متغیر محلی VAR و یک متغیر محلی ADD و یک پشتنه STADD برای نگهداری آدرس‌های بازگشته تعریف کنید.

• ADD := Main و TOP := NULL (ب) فار دهید

"(٢) تدبیر، مرحله K، فاخوانی، P

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

(ب) با استفاده از مقادیر آرگومان جدید پارامترها را مقداردهی کنید و قرار دهید.

(ج) Go to Step 1 [پہلے مرحلے کی توجیہ کا آغاز]

"Return" محلة J، بازگشت

(الف) اگر $\text{ADD} = \text{Main}$ آنگاه **Return** [کنترل به برنامہ اصلی، منتقل، می شود.]

.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 پشته STADD قرار دهید.

Go to Step **SAVE** ↗

> . GO TO Step SAVE (3)

این الگوریتم تبدیل را با الگوریتم بخش ۸ مقایسه کنید.

صفها، صفحه‌ای دوسره Deque

مسئله ۲۱-۶: صفت کاراکترهای زیر را در نظر بگیرید که در آن QUEUE یک آرایه چرخشی یا حلقوی است که شش خانه حافظه به آن اختصاص داده است:

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

جهت سهولت در نمادگذاری از—"برای نمایش خانه حافظه خالی استفاده می‌کنیم. وضعیت صفات به محض انجام هر یک از عملیات زیر شرح دهید:

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

(ب) دو حرف حذف می شوند. (ج) صفت اضافه می شود.

(ز) حذف می شود: L, K, M به صفت اضافه می شوند. (ز) و حذف حذف می شود.

(ب) يكره حلف فيه شرعاً

¹⁰ مکالمہ فیضی، ۶۰

Journal of the American Statistical Association, Vol. 36, No. 213, March, 1941.

... queve A. S. B. E.

لایه را که REAR می‌گیرد، از پشت کشیده شود.

(ب) دو حرف A و C حذف می شوند، یاقع می ماند:

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

تو جہے دارید کہ FRONT دو واحد افزائش میں یابد۔

(ج) L , K و M به انتهایی صفت اضافه می‌شوند. از آنجا که K در آخرین خانه حافظه QUEUE قرار می‌گیرد L و M در دو خانه اول قرار می‌گیرند. درستیجه:

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

توجه دارید که REAR به اندازه 3 واحد افزایش می‌باید که حساب با باقیمانده 6 است.

$$\text{REAR} = 5 + 3 = 8 = 2 \pmod{6}$$

(د) دو حرف ابتدایی D و F حذف می شوند، باقی می ماند:

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

(۵) R به انتهای صفاتی می‌شود، درنتیجه:

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

(و) دو حرف ابتدایی K و L حذف می شوند، باقی می ماند:

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

توجه دارید که FRONT به اندازه دو واحد افزایش می‌یابد که حساب با باقیمانده ۶ است:

$$\text{FRONT} = 6 + 2 = 8 = 2 \pmod{6}$$

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

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

(ح) دو حرف ابتدایی M و R حذف می شود، باقی می ماند:

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

(ط) حرف ابتداء S حذف می شود. چون **FRONT = REAR** درنتیجه صفات خالی است. از این رو **NULL** را در **FRONT** و **REAR** جایگزین می کنیم. بنابراین

FRONT = 0, REAR = 0 QUEUE: —, —, —, —, —, —

(۵) چون $FRONT = \text{NULL}$ ، هیچ عمل حذفی نمی‌تواند صورت گیرد، زیربازی اتفاق می‌افتد.

مسئله ۲۲-۶: فرض کنید هر ساختمان داده در یک آرایه چرخشی با N خانه حافظه ذخیره می شود.

(الف) تعداد عناصر NUMB صفت را بر حسب FRONT و REAR تعیین کنید.

(ب) تعداد عناصر NUMB صف دوسره يا Deque را بحسب LEFT و RIGHT تعیین کنید.

(ج) چہ وقت آرایہ پر خواهد شد؟

حل: (الف) اگر $\text{FRONT} \leq \text{REAR}$ ، آنگاه $\text{NUMB} = \text{REAR} - \text{FRONT} + 1$. برای مثال صفت زیر را با $N = 12$ در نظر بگیرید :

$$\text{FRONT} = 3, \quad \text{REAR} = 9 \quad \text{QUEUE: } _, _, _, *, *, *, *, *, *, *, _, _, _$$

آنگاه، $\text{NUMB} = 9 - 3 + 1 = 7$ که در تصویر نشان داده شده است.

اگر $\text{FRONT} - \text{REAR} < \text{FRONT}$ تعداد خانهٔ خالی می‌باشد، بنابراین :

$$\text{NUMB} = N - (\text{FRONT} - \text{REAR} - 1) = N + \text{REAR} - \text{FRONT} + 1$$

برای مثال صفت زیر با $N = 12$ را در نظر بگیرید :

$$\text{FRONT} = 9, \quad \text{REAR} = 4 \quad \text{QUEUE: } *, *, *, *, _, _, _, _, *, *, *, *$$

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

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

$$\text{NUMB} = \text{REAR} - \text{FRONT} + 1 \pmod{N}$$

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

$$\text{NUMB} = \text{RIGHT} - \text{LEFT} + 1 \pmod{N}$$

(ج) زمانی که

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

با یک صفت آرایه پر است.

به همین ترتیب، وقتی

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

با یک صفت دوسره یا Deque آرایه پر است.

از هر یک از این شرایط نتیجه می‌شود $\text{NUMB} = N$.

مسئله ۶-۲۳ : صفت دوسره Deque کاراکترهای زیر را در نظر بگیرید که در آن DEQUE یک آرایه چرخشی یا حلقوی است و شش خانهٔ حافظه به آن اختصاص داده شده است.

$$\text{LEFT} = 2, \quad \text{RIGHT} = 4 \quad \text{DEQUE: } _, \text{A}, \text{C}, \text{D}, _, _$$

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

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

(ب) دو حرف از سمت راست حذف می‌شوند.

(ج) K و L در سمت چپ به صفت دوسره اضافه می‌شوند.

(د) یک حرف از سمت چپ حذف می‌شود.

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

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

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

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

$$\text{LEFT} = 2, \quad \text{RIGHT} = 5 \quad \text{DEQUE: } _, A, C, D, F, _$$

توجه دارید که RIGHT به اندازه واحد اضافه می‌شود.

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

$$\text{LEFT} = 2, \quad \text{RIGHT} = 3 \quad \text{DEQUE: } _, A, C, _, _, _$$

توجه دارید که RIGHT دو واحد افزایش می‌یابد.

(ج) K، L و M در سمت چپ اضافه می‌شوند. چون K اولین خانه حافظه و L آخرین خانه حافظه و M

یک خانه مانده به آخرین خانه حافظه است. درنتیجه

$$\text{LEFT} = 5, \quad \text{RIGHT} = 3 \quad \text{DEQUE: } K, A, C, _, M, L$$

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

$$\text{LEFT} = 2 - 3 = -1 = 5 \pmod{6}$$

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

$$\text{LEFT} = 6, \quad \text{RIGHT} = 3 \quad \text{DEQUE: } K, A, C, _, _, L$$

توجه دارید که LEFT به اندازه واحد افزایش می‌یابد.

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

$$\text{LEFT} = 5, \quad \text{RIGHT} = 3 \quad \text{DEQUE: } K, A, C, _, R, L$$

توجه دارید که LEFT به اندازه واحد افزایش می‌یابد.

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

$$\text{LEFT} = 5, \quad \text{RIGHT} = 4 \quad \text{DEQUE: } K, A, C, S, R, L$$

(ز) چون $1 + \text{LEFT} = \text{RIGHT}$ ، آرایه پر است و از این رو T نمی‌تواند به صفت دوسره اضافه شود،

سرریزی اتفاق می‌افتد.

مسأله ۶-۲۴ : یک صفت دوسره Deque را در نظر بگیرید که در یک آرایه چرخشی با N خانه حافظه

نگهداری می‌شود.

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

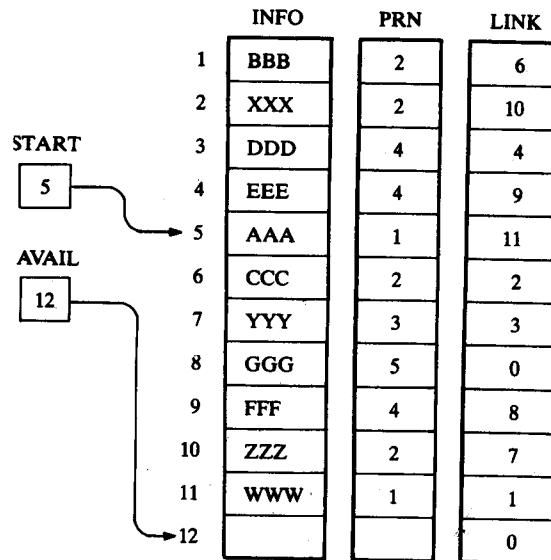
(ب) فرض کنید یک عنصر از صفت دوسره حذف می شود. چگونه تغییر می کنند؟
حل: (الف) اگر این عنصر در سمت چپ اضافه شود، آنگاه **LEFT** به اندازه واحد $(\text{mod } N)$ اضافه می شود. از طرف دیگر، اگر این عنصر در سمت راست اضافه شود، آنگاه **RIGHT** به اندازه واحد $(\text{mod } N)$ اضافه می شود.

(ب) اگر این عنصر از سمت چپ حذف شود، آنگاه **LEFT** به اندازه واحد $(\text{mod } N)$ افزایش می یابد و اگر این عنصر از سمت راست حذف شود، آنگاه **RIGHT** به اندازه واحد $(\text{mod } N)$ کاهش می یابد. در حالی که $\text{LEFT} = \text{RIGHT}$ قبل از عمل حذف (یعنی وقتی که صفت Deque تنها یک عنصر دارد)، آنگاه هم **RIGHT** و هم **LEFT** جایگزین می شود که بیانگر آن هستند صفت دوسره، خالی است.

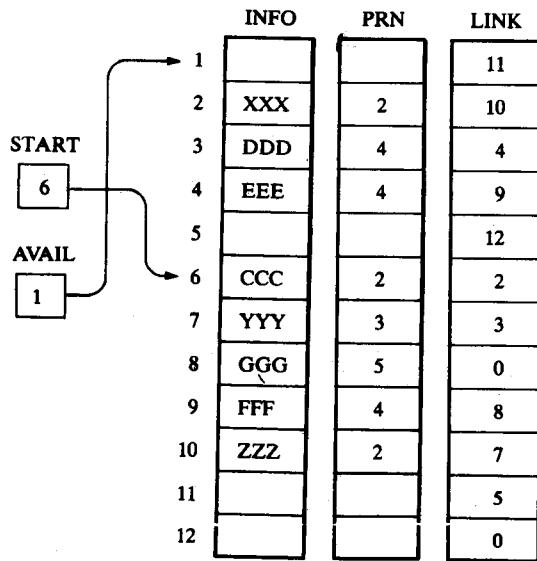
صفهای اولویت

مسئله ۲۵-۶: صفت اولویت شکل ۲۰-۶ را در نظر بگیرید که به صورت یک لیست یکطرفه نگهداری می شود. (الف) این ساختمان داده را پس از اضافه شدن **[2]**, **[XXX, 2]**, **[YYY, 3]** و **[WWW, 1]** به صفت توصیف کنید. (ب) اگر پس از عملیات اضافه شدن 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 می کنیم. این عنصر **DDD** است. از این رو **XXX** را قبل از **DDD** (بعد از **CCC**) در خانه حافظه خالی اول **[INFO[2]]** اضافه می کنیم. آنگاه لیست را پیمایش کرده اولین عنصری را که عدد اولویت آن بزرگتر از عدد اولویت **YYY** است پیدا می کنیم. این بار نیز این عنصر، **DDD** است. از این رو **YYY** را قبل از **DDD** (بعد از **XXX**) در خانه خالی بعدی، **[INFO[7]]** اضافه می کنیم. آنگاه لیست را پیمایش کرده اولین عنصری را که عدد اولویت آن بزرگتر از عدد اولویت **ZZZ** است پیدا می کنیم. این عنصر، **YYY** است. از این رو **ZZZ** را قبل از **YYY** (بعد از **XXX**) در خانه حافظه بعدی، **[INFO[10]]** اضافه می کنیم. بالاخره، لیست را پیمایش کرده اولین عنصری را که عدد اولویت آن بزرگتر از عدد اولویت **WWW** است پیدا می کنیم. این عنصر، **BBB** است. از این رو **WWW** را قبل از **BBB** (بعد از **AAA**) در خانه حافظه خالی بعدی، **[INFO[11]]** اضافه می کنیم. نهایتاً ساختمان داده شکل ۲۰-۶ (الف) نتیجه می شود.



(الف)



(ب)

(ب) سه عنصر اول از لیست یکطرفه حذف می‌شوند. به طور مشخص، نخست AAA حذف می‌شود و خانه حافظه آن INFO[5] به لیست AVAIL اضافه می‌شود. آنگاه WWW حذف می‌شود و خانه حافظه آن INFO[11] به لیست AVAIL اضافه می‌شود. بالاخره BBB حذف می‌شود و خانه حافظه آن INFO[1] به لیست AVAIL اضافه می‌شود. نهایتاً ساختمان داده شکل ۲۶-۶ (ب) نتیجه می‌شود.

توجه کنید: ملاحظه می‌کنید که START و AVAIL با توجه به عملیات مسأله، تغییر می‌کنند.
مسأله ۶-۶: صفت اولویت شکل ۲۶-۶ را در نظر بگیرید که در آرایه دو بعدی QUEUE نگهداری می‌شود. (الف) وضعیت این ساختمان داده را پس از اضافه شدن، (RRR, 3)، (SSS, 4)، (TTT, 1)، (UUU, 4) و (VVV, 2) به صفت توضیح دهید. (ب) اگر پس از عملیات اضافه شدن قبلی، سه عنصر از این ساختمان داده حذف شوند وضعیت صفت را توضیح دهید.

حل: (الف) هر عنصر را در ردیف اولویت مرتبه اش 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 داده شکل ۲۷-۶ (الف) مرسیم. همانگونه که قبل از این مذکور شدیم، عملیات اضافه کردن در نمایش آرایه‌ای معمولاً ساده‌تر از عملیات اضافه کردن در نمایش لیست یکطرفه است.

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

(الف)

		QUEUE							
		FRONT	REAR	1	2	3	4	5	6
1	0		0		CCC	XXX	VVV		
2	2		4						
3	1		1		RRR				
4	5		3		FFF	SSS	UUU		DDD EEE
5	4		4					GGG	

(ب)

شکل ۲۷-۶

(ب) نخست عناصر ردیف ۱ را که دارای بیشترین الوبت هستند حذف می‌کنیم. چون در ردیف ۱ تنها دو عنصر، AAA و TTT وجود دارد، آنگاه عنصر ابتدای ردیف ۲، BBB، نیز باید حذف شود. نهایتاً این عملیات به ساختمان داده شکل ۶-۲۷ (ب) منتهی می‌شود.

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

مسأله‌های تكمیلی

پشته‌ها

مسأله ۶-۲۷: پشته زیر متشكل از نام شهرها را درنظر بگیرید:

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

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

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

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

مسأله ۶-۲۸: پشته زیر را درنظر بگیرید که در آن به STACK، N = 4 خانه حافظه اختصاص داده شده است.

STACK: AAA, BBB, _____.

وضعیت پشته را به محض انجام عملیات زیر توضیح دهید.

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

مسأله ۶-۲۹: پشته زیر متشكل از اعداد صحیح را در حافظه درنظر بگیرید که در آن به STACK، N = 6، خانه حافظه اختصاص داده شده است.

TOP = 3 STACK: 5, 2, 3, → → →

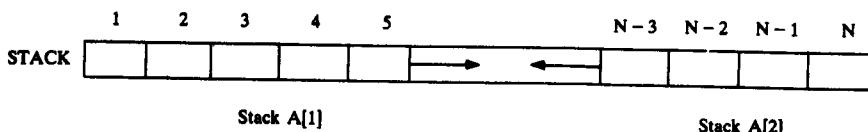
خروجی قطعه برنامه زیر را مشخص کنید.

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

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

(الف) یک زیربرنامه PUSH(STACK, N, TOP, ITEM, K) را به داخل پشته A[K] Push کند.
(ب) یک زیربرنامه POP(STACK, TOP, ITEM, K) بنویسید تا عنصر بالای پشته A[K] را حذف کند و

این عنصر را در متغیر ITEM جایگزین کند.



شکل ۲۸-۶

عبارت‌های محاسباتی، عبارتهاي لهستانی

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

$$(A + B)/D \uparrow ((E - F) * G) \quad (ب) \quad (الف) (A - B)/((D + E) * F)$$

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

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

$$5 + 3 \uparrow 2 - 8 / 4 * 3 + 6 \quad (الف)$$

$$6 + 2 \uparrow 3 + 9 / 3 - 4 * 5 \quad (ب)$$

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

$$E: 6 + 2 \uparrow 3 \uparrow 2 - 4 * 5$$

با فرض‌های زیر E را ارزیابی کنید: (الف) همانگونه که در عبارتهای بالا داشتید عمل توان رسانی از چپ به راست اجرا شود. (ب) عمل توان رسانی از راست به چپ اجرا شود.
مسئله ۳۵-۶: هر یک از عبارتهای پسوندی زیر را در نظر بگیرید:

$$\begin{aligned} P_1: & \quad 5, 3, +, 2, *, 6, 9, 7, -, /, - \\ P_2: & \quad 3, 5, +, 6, 4, -, *, 4, 1, -, 2, \uparrow, + \\ P_3: & \quad 3, 1, +, 2, \uparrow, 7, 4, -, 2, *, +, 5, - \end{aligned}$$

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

مسئله ۳۶-۶: هر عبارت پسوندی مسئله ۳۵-۶ را با استفاده از الگوریتم 6.3 ارزیابی کنید.

مسئله ۳۷-۶: با استفاده از الگوریتم 6.4 هر عبارت میانوندی را به عبارت پسوندی معادل آن تبدیل کنید.

$$((A + B)/D) \uparrow ((E - F) * G) \quad (ب)$$

$$(A - B)/((D + E) * F) \quad (الف)$$

این مسئله را با مسئله ۳۱-۶ مقایسه کنید.

زیربرنامه‌های بازگشتی

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

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

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

مسئله ۳۹-۶: فرض کنید A و B دو عدد صحیح نامنفی باشند. فرض کنید تابع GCD به صورت بازگشتی زیر تعريف 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) $GCD(540, 168)$ و $GCD(6, 15)$, $GCD(20, 28)$ را محاسبه کنید. (ب) این تابع چه عملی انجام می‌دهد.

مسئله ۴۰-۶: فرض کنید N یک عدد صحیح است و $H(N)$ به صورت بازگشتی زیر تعريف شده است:

$$H(N) = \begin{cases} 3 * N & \text{if } N < 5 \\ H(\frac{N-3}{2}) + 7 & \text{otherwise} \end{cases}$$

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

مسأله ۴۱-۶: با استفاده از تعریف ۶.۳ برای تابع آکرمان، $A(2, 2)$ را تعیین کنید.

مسأله ۴۲-۶: 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) $F(1, 5)$, $F(4, 2)$, $F(1, 4)$ و $F(2, 4)$ را پیدا کنید. (ب) چه وقت $F(M, N)$ قابل تعریف نیست؟

مسأله ۴۳-۶: فرض کنید A یک آرایه N عنصری از اعداد صحیح باشد. فرض کنید X یک تابع صحیح تعريف 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 (5) را پیدا کنید:

$$N = 3, \quad A: 2, 7, -4 \quad (ب) \quad N = 8, \quad A: 3, 7, -2, 5, 6, -4, 2, 7 \quad (\text{الف})$$

این تابع چه عملی انجام می‌دهد؟

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

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

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

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

صفهای صفحه‌ای دوسره Deque

مسأله ۴۶-۶: صف زیر را درنظر بگیرید که در آن به QUEUE شش خانه حافظه اختصاص داده شده است:

$\text{FRONT} = 2$, $\text{REAR} = 5$

QUEUE: _____, London, Berlin, Rome, Paris, _____

به محض 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.

مسئله ۴۷ - ۶: صف دوسره Deque زیر را در نظر بگیرید که در آن به DEQUE، ۶ خانه حافظه اختصاص داده شده است.

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

به محض انجام عملیات زیر وضع صف و وضع LEFT و RIGHT را توضیح دهید.

(الف) Athens در سمت چپ اضافه می‌شود. (ه) دو شهر از سمت راست حذف می‌شوند.

(ب) دو شهر از سمت حذف می‌شوند. (و) یک شهر از سمت چپ حذف می‌شود.

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

(د) Moscow در سمت راست اضافه می‌شود.

مسئله ۴۸ - ۶: فرض کنید یک صف در آرایه چرخشی یا حلقوی QUEUE با $N = 12$ خانه حافظه نگهداری می‌شود. تعداد عناصر صف QUEUE را در هر یک از حالات زیر محاسبه کنید.

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

مسئله ۴۹ - ۶: صف اولویت شکل ۲۶ - ۶ (ب) را در نظر 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 حذف کردن، عناصرهای (3, RRR, 3), (SSS, 1), (TTT, 3) و (UUU, 2) به صف اضافه شوند وضعیت این ساختمان داده را توضیح دهید.

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

مسئله ۵۰ - ۶: صف اولویت شکل ۲۷ - ۶ (ب) را در نظر بگیرید که در آرایه دو بعدی QUEUE نگهداری می‌شود.

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

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

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

برای مسائلهای زیر برنامه بتویسید

مسئله ۵۱-۶: **QuickSort**: را به یک زیربرنامه **QUICK(A, N)** تبدیل کنید که آرایه N عنصری A را مرتب می‌کند. برنامه را با استفاده از داده‌های زیر آزمایش کنید:

(الف) ۴۴, ۳۳, ۱۱, ۵۵, ۷۷, ۹۹, ۴۰, ۶۸, ۲۲, ۸۸, ۶۶

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

مسئله ۵۲-۶: برنامه‌ای بتویسید تا راه حل مسئله برج‌های هانوی را برای n دیسک به دست دهد. برنامه را با استفاده از (الف) $3 = \text{د} + (\text{ب}) 4 = \text{آزمایش}$ کنید.

مسئله ۵۳-۶: الگوریتم ۶۴ را به یک زیربرنامه **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 القابی مفرد است و از نمادهای متداول بولی جمع (+)، تفرقه (-)، ضرب (*) و تقسیم (/) استفاده کنید اما بولی توان از نماد \oplus یا \otimes استفاده کنید (یعنی از زبانهای بر تام‌نویسی نماد \oplus را برای توان قبول نمی‌کنند) برنامه را با استفاده از

(الف) $A + (B * C - (D / E * F) * G) * H$ (ب) $((A + B) * D) \$ (E - F)$
آزمایش کنید

مسئله ۵۴-۶: فرض کنید یک صف لولیت مطابق شکل ۲۰-۶ به صورت لیست یک‌طرفه نگهداری می‌شود.

(الف) یک زیربرنامه **Procedure** بتویسید:

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

تا عصر **ITEM** با عدد لولیت **N** را به صف اخراج کند الگوریتم ۶۴ را بینید.

(ب) یک زیربرنامه **Procedure** بتویسید:

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

تا یک عنصر را از صف حذف کند و آن را در متغیر **ITEM** جایگزین کند الگوریتم ۶۴ را بینید.

لين زيربرنامه داده‌های **Procedure** را با استفاده از داده‌های مسئله ۵۴-۶ آزمایش کنید.

مسئله ۵۵-۶: فرض کنید یک صف الولیت مطابق شکل ۲۲-۶ به صورت آرایه دو بعدی نگهداری می‌شود.

(الف) یک زیربرنامه **Procedure** بتویسید:

INSPOA(QUEUE, FRONT, REAR, ITEM, N)

تا عنصر **ITEM** با عدد الولویت **N** را به صف اضافه کند. الگوریتم ۶.۱۶ را بینید.

((ب)) یک زیربرنامه **Procedure** بتوانید:

DELPOL(QUEUE, FRONT, REAR, ITEM)

تا یک عنصر را از صف حذف کند و آن را در متغیر **ITEM** جایگزین کند. الگوریتم ۶.۱۵ را بینید.
این زیربرنامه از **Procedure** ریلا استفاده از «الفهای مساله» ۶-۲۶ آزمایش کنید. فرض کنید
تعداد **ROW** سطر و تعداد **COL** ستون دارد که در آن **ROW** و **COL** متغیرهای سراسری هستند.

فصل ۷

درختها

۱- مقدمه

تا اینجا، انواع مختلف ساختمان داده‌های خطی از قبل رشته‌ها، آرایه‌ها، لیستهای پشتیها و صفحهای مورد مطالعه و بررسی کامل قرار گرفته است. فصل حاضر یک ساختمان داده غیرخطی موسوم به درخت را تعریف می‌کند. این ساختمان اساساً برای نمایش داده‌هایی که شامل رابطه سلسله مراتبی بین عناصر مانند رکوردها، درختهای خانوادگی و جدول‌های فهرست مطالب کتاب است بکار می‌رود.

نخست یک نوع خاص از درخت موسوم به درخت دودویی را بررسی می‌کنیم که می‌توان آن را به سادگی 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.

۲- درختهای دودویی

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

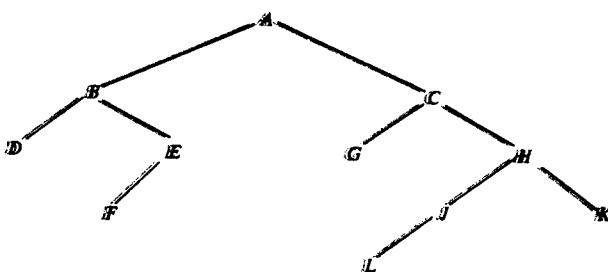
(الف) T خالی است که به آن درخت پوچ یا درخت تهی می‌گویند یا

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

اگر T شامل ریشه R باشد، آنگاه دو درخت T_1 و T_2 به ترتیب زیردرختهای چپ و راست R نامی‌باشد. اگر T_1 غیرتھی باشد، آنگاه ریشه آن را گریب‌طبعی چپ R می‌گویند به همین ترتیب اگر T_2 غیرتھی باشد آنگاه ریشه آن را گریب‌طبعی راست R می‌گویند.

اگر اوقات یک درخت دودویی را به وسیله یک نمودار نمایش می‌دهند، بطور مشخص نمودار

شکل ۱۱-۷ زیر یک درخت دودویی را نشان می‌دهد.



شکل ۱۱-۷

(ii) از T گره تشکیل شده است که با حروف A تا I یا چنانچه شده است.

(iii) ریشه T گره A است که در بالای نمودار است. (iii) یک خط از گره N که به طرف پائین و متمایل به چپ است میین گریب‌طبعی چپ N است و خط دیگر از گره N که به طرف پائین و متمایل به راست است میین گریب‌طبعی راست N است. ملاحظه می‌کنید که:

(الف) B یک گریب‌طبعی چپ و C یک گریب‌طبعی راست گره A است.

(ب) زیردرخت چپ ریشه A مشکل از گرهای B ، D ، E و F است و زیردرخت راست A مشکل از گرهای C ، G ، H ، J و K است.

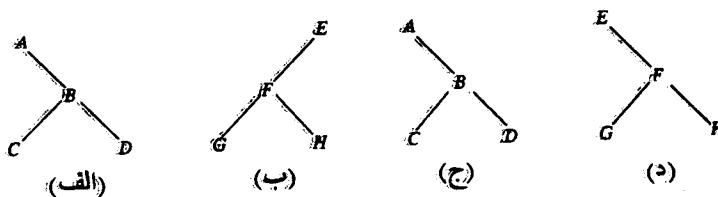
هر گره N در یک درخت دودویی T دارای ۰، ۱ یا ۲ گریب‌طبعی است. گرمهای A ، B و C و دو گریب‌طبعی دارند و گرمهای E و I همچنان یک گریب‌طبعی دارند و گرمهای D ، F ، G ، H و K هیچ گریب‌طبعی ندارند. گرمهایی که هیچ گریب‌طبعی ندارند گرمایی انتہائی نام دارند.

تعریف بالا از درخت دودویی T یازگشتی است چون T بر حسب زیردرختهای دودویی T_1 و T_2 تعریف می‌شود. به عویض معنی آن این است که هر گره N از T ، یک زیردرخت چپ و راست دارد. علاوه بر این، اگر N یک گره انتہائی 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 داری یک شکل باشند، درختها را کپی هم گویند اگر این درختها متشابه یوشه و محتوای گرمایی متناظر آنها یکسان باشد.

مثال ۱-۷

چهار درخت دودویی شکل ۱-۷ را درنظر بگیرید. سه درخت (الف)، (ج) و (د) مشابه هستند. بعویظه، درختهای (الف) و (ج) کپی هم هستند چون دارای دامنهای مساوی و یکسان در گرمایی متناظر هستند، درخت (ب) نه مشابه درخت (د) است نه کپی آن، زیرا ماده درخت دودویی بین یک گرمهبدی چپ و یک گرمهبدی راست حتی وقتی تنها یک گرمهبدی وجود داشته باشد تفاوت و تمایز فائق هستیم.



شکل ۱-۷

مثال ۲-۷ عبارتهای جیری

یک عبارت جیری E نظیر عبارت زیر را که تنها حلوی عملیات دوتایی است، درنظر بگیرید:

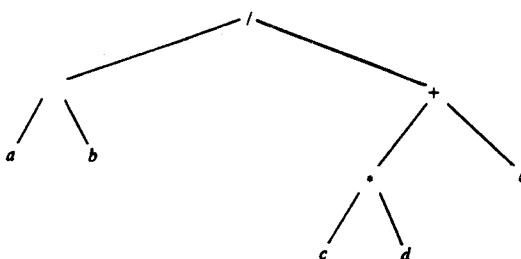
$$E = (a - b) / ((c * d) + e)$$

برای توان به وسیله درخت دودویی T به صورتی که در شکل ۱-۸ آمده است نمایش داد. به عبارت دیگر هو متغیر یا ثابت در E به صورت گرمه "لایخانی" در T ظاهر شده است. که زیر درختهایی چپ و راست آن متناظر با عملوندهای عملگر هستند. برای مثال:

(الف) در آن عبارت E ، عملوندهای $+ \cdot - / *$ عبارتند از a ، b ، c و d .

(ب) در درخت T ، زیر درختهای گرمه $+ \cdot - / *$ متناظر با زیر عبارتهای a ، b ، c و d هستند.

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

شکل ۳ $E = (a - b) / ((c * d) + e)$.

چند اصطلاح

اصطلاحاتی که روابط خانوادگی را توصیف می‌کنند اغلب برای توصیف روابط بین گره‌ها در یک درخت T نیز بکار گرفته می‌شوند. به طور مشخص، فرض کنید N یک گره درخت T با گره بعدی چپ S_1 و گره بعدی راست S_2 باشد. آنگاه N را پدر S_1 می‌گویند. به طور مشابه S_1 بجهة (پسر) چپ N و S_2 را بجهة (پسر) راست N می‌گویند. علاوه براین S_1 و S_2 را هم ردیف یا براذرگویند. هر گره N در یک درخت دودویی T با استثنای ریشه، یک پدر منحصر بفرد دارند که ریشه قبلی N می‌باشد.

اصطلاحات نسل و جدّ دارای معنی متداول خودشان هستند. به عبارت دیگر گره L یک نسل گره N است و N یک جد گره L است اگر یک دنباله از بجهه‌ها از N به L وجود داشته باشد. به عویژه، بسته به اینکه L به زیردرخت چپ تعلق داشته باشد یا زیردرخت راست. L یک نسل چپ یا راست گره N است.

اصطلاحات متداول در نظریه گراف و علم باگبانی نیز، با یک درخت دودویی T بکار می‌رود. بطور مشخص، خطی که از گره N درخت T به یک گره بعدی رسم می‌شود یک یال و دنباله‌ای از یالهای متوالی یک مسیر نامیده می‌شود. یک گره 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 ریشه R درخت T عدد سطح ۰ نسبت داده می‌شود و به هر گره 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 حداقل تعداد گره‌های یک شاخه T است، ثابت می‌شود که عمق یک درخت

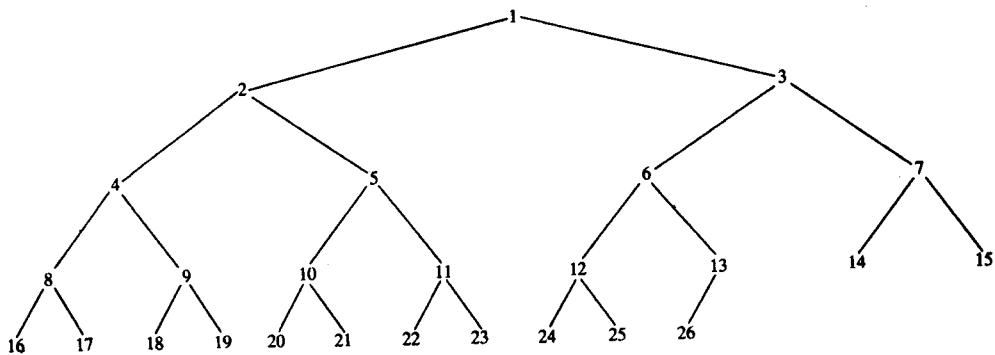
یک واحد بیشتر از بزرگترین عدد سطح درخت T است. درخت T در شکل ۱-۷ عمقدار دارد.

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 منتظر آنها یکی باشد.

درختهای دودویی کامل

درخت دودویی دلخواه T را درنظر بگیرید. هر گره 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 سطوح‌های آن بجز احتمالاً آخرین سطح، حداکثر تعداد گره 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 ۴-۷ رسم شده است.



شکل ۴-۷ درخت کامل T_{26}

گره‌های درخت دودویی کامل T_{26} شکل ۴-۷ با هدف مشخص، با اعداد صحیح ۱، ۲، ۳، ...، ۲۶ از چپ به راست، نسل به نسل شماره گذاری شده است. با این نحوه شماره گذاری، می‌توان به سادگی، بچه‌ها و پدر هر گره K را در یک درخت کامل T_n بدست آورد. بطور مشخص بچه‌های چپ و راست گره K به ترتیب $K * 2 + 1$ و $2 * K + 1$ هستند و پدر آن گره $\lceil k/2 \rceil$ است. برای مثال بچه گره ۹ گره‌های ۱۸ و ۱۹ هستند و پدرشان گره $= 4 = \lceil 9/2 \rceil$ است. عمق d_n درخت کامل T_n با n گره از رابطه

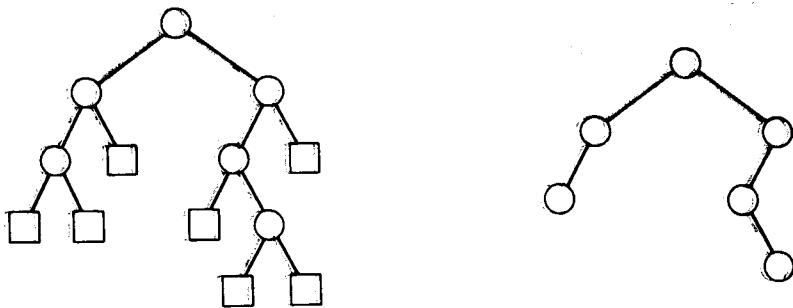
$$D_n = \lfloor \log_2 n + 1 \rfloor$$

بدست می‌آید. عدد بالا، عدد نسبتاً کوچکی است، برای مثال، اگر درخت کامل T_n دارای ۱۰۰۰۰۰۰ گره باشد، آنگاه عمق آن $21 = D_n$ است.

درختهای دودویی گسترش یافته: ۲- درخت

یک درخت دودویی T را یک ۲-درخت یا یک درخت دودویی گسترش یافته گویند اگر هر گره N ، ۰ یا ۲ بچه داشته باشد. در چنین حالتی، گرهایی که دو بچه دارند گرهای داخلی و گرهایی که ۰ بچه دارند گرهای خارجی نامیده می‌شوند. گاهی اوقات گره‌ها را در نمودار به صورت زیر از هم تمیز می‌دهند: برای نمایش گرهای داخلی از دایره و برای نمایش گرهای خارجی از مربع استفاده می‌کنند.

اصطلاح "درخت دودویی کامل" از عمل زیرمتناهی گرفته است. یک درخت دودویی دلخواه T نظیر درخت شکل ۵-(الف) را در نظر بگیرید. آنگاه T را می‌توان با جانشین‌سازی هر زیردرخت خالی توسط یک گره جدید، مانند نمودار شکل ۵-(ب) به یک ۲-درخت تبدیل کرد. ملاحظه می‌کنید که درخت جدید واقعاً یک ۲-درخت است. علاوه بر این، گرهای درخت اصلی T اکنون گرهای داخلی در درخت گسترش یافته هستند و گرهای خارجی در درخت گسترش یافته هستند.



(ب) ۲-درخت گسترش یافته

(الف) درخت دودویی T شکل ۵-۷ تبدیل یک درخت دودویی T به یک ۲-درخت

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

۳- 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 معموله نمایش پیوندی درخت T است و مشابه روش لیست‌های

پیونلای است که در حافظه تمایش داده می‌شود. روش دوم که تنها از یک آرایه استفاده می‌کند نمایش ترتیبی Sequential درخت T است. اصلی ترین موضوع در هر تمایش درخت T ، آن است که به ریشه R درخت T دسترسی مستقیم داشته باشیم و یا معلوم بودن هر گره N از T ، باید بتوانیم هر یک گره N دسترسی پلاکرد.

تمایش پیونلای درختهای دودویی

درخت دودویی T را در نظر بگیرید. درخت T در حافظه بوسیله یک تمایش پیونلای نگهداری می‌شود که از سه آرایه موازنی 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.

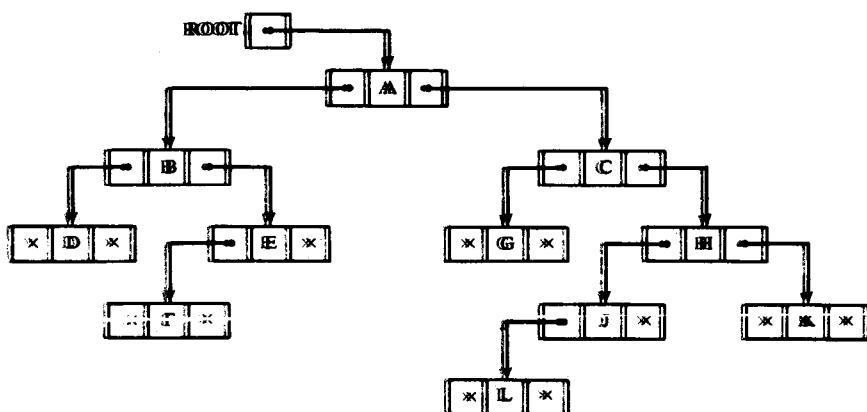
قبل از هر چیز، متذکر می‌شویم که هر گره N متناظر با یک مکان K است به گونه‌ای که:

((۱)) حلوی داده گره N است.

((۲)) $\text{LEFT}[K]$ حلوی مکان یقه چپ گره N است.

((۳)) $\text{RIGHT}[K]$ حلوی مکان بچه راست گره N است.

علاوه بر این، ROOT حلوی مکان ریشه R درخت T است. اگر هر زیردرخت $-$ باشد، آنگاه اشاره‌گر متناظر آن دارای مقدار پوج NULL خواهد بود. اگر خود درخت T خالی باشد، آنگاه NULL خواهد بود.



شکل ۶-۷

	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 ۷-۷

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 گره N یک رکورد ذخیره شود. به بیان دیگر، INFO واقعاً ممکن است یک آرایه 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 اشاره گر AVAIL را می‌دهند که در رابطه با لیستهای پیوندی در فصل ۵ مورد بحث و بررسی قرار گرفت. معمولاً فرض می‌کنند آرایه LEFT دارای اشاره گرهایی برای لیست AVAIL است.

تذکر ۳: هر آدرس غیرمجازی که ممکن است برای اشاره گر پوج **NULL** انتخاب شود با **NULL** نمایش داده می‌شود. در کاربردهای عملی، برای **NULL** از ۰ یا یک عدد منفی استفاده می‌شود.

مثال ۷-۳

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

مثال ۷-۴

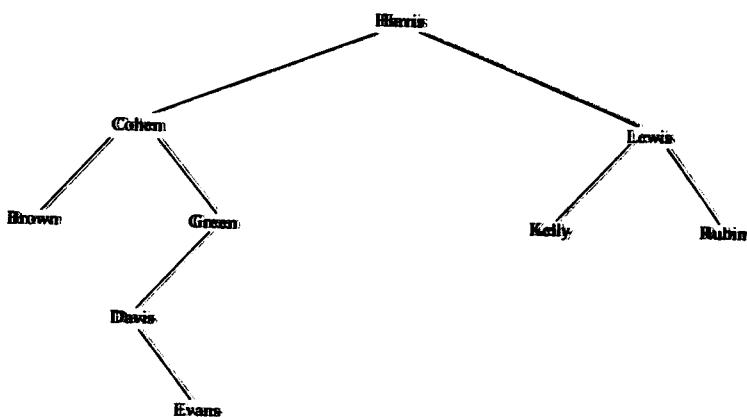
فرض کنید فایل پرستنی یک شرکت کوچک با 9 کارمند از فیلد‌های زیر تشکیل شده است:

Name, Social Security Number, Sex, Monthly Salary

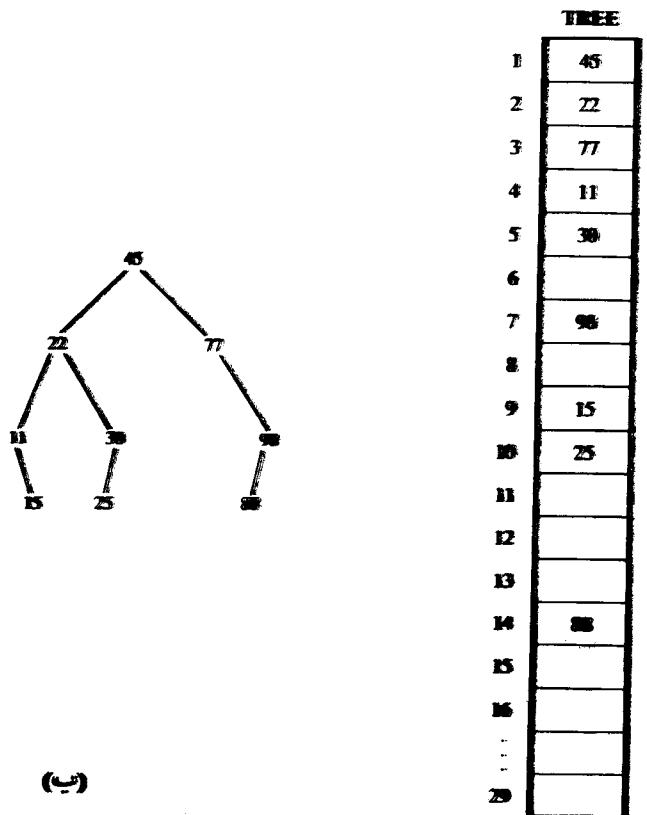
شکل ۷-۸ چگونگی نگهداری فایل را در حافظه به صورت یک درخت دودویی نشان می‌دهد. این ساختمان داده را با شکل ۱-۵ مقایسه کنید که در آن دقیقاً همین داده‌ها به صورت یک لیست یکطرفه سازماندهی شده است.

	NAME	SSN	SEX	SALARY	LEFT	RIGHT
1					0	
2	Davis	192-38-7282	Female	22 800	0	12
3	Kelly	165-64-3351	Male	19 000	0	0
4	Green	175-56-2251	Male	27 200	2	0
5					1	
6	Brown	178-52-1065	Female	14 700	0	0
7	Lewis	181-58-9939	Female	16 400	3	10
8					11	
9	Cohen	177-44-4557	Male	19 000	6	4
10	Rubin	135-46-6262	Female	15 500	0	0
11					13	
12	Evans	168-56-8113	Male	34 200	0	0
13					5	
14	Harris	208-56-1654	Female	22 800	9	7

شکل ۷-۸



شكل ٧-٩



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

(الف) مقدار $14 = \text{ROOT}$ بیانگر آن است که **Harris** ریشه درخت است.

(ب) $9 = \text{LEFT}[14]$ بیانگر آن است که **Cullen** بجهة راست **Harris** و $7 = \text{RIGHT}[14]$ بیانگر آن است که **Lewis** بجهة راست **Harris** است.

برای هر گره جدید داخل نمودار، با تکرار مرحله (ب) شکل ۹-۷ بدست می‌آید.

نمایش ترتیبی درختهای دودویی

فرض کنید **T** یک درخت دودویی باشد که کامل یا تقریباً کامل است. آنگاه روش کارایی برای تگهداری **T** در حافظه وجود دارد که نمایش ترتیبی **T** تمام دارد. این نمایش تنها از یک آرایه خطی TREE به صورت زیر لستله می‌کند.

(الف) ریشه **T** در **TREE[1]** ذخیره می‌شود.

(ب) اگر گره **N** **TREE[K]** را شکال کند آنگه بجهة چپ آن در **TREE[2 * K]** و بجهة راست آن در **TREE[2 * K + 1]** ذخیره می‌شود مجلدات در **NULL** برای بیان یک زیردرخت خالی مورد استفاده قرار می‌گیرد. بصورت $\text{NULL} = \text{TREE}[1]$ بیانگر آن است که درخت خالی است.

نمایش ترتیبی درخت دودویی **T** در شکل ۷-۱۰ (الف) در شکل ۷-۱۰ (ب) شان داشته است. ملاحظه می‌کنید که حتی اگر **T** تها ۹ گره داشته باشد در آرایه **TREE** به ۱۴ خانه احتیاج است. در واقع، اگر دودویی پیچ برای گرمهای تنهایی اختیار شود آنگه واقعاً برای گره بعدی **TREE[15]** به احتیاج خواهیم داشت که بیان کلی آن، نمایش ترتیبی معمولاً از کارلی لادم بتوحداریت با تقریباً 1.44^k عنصر خواهد بود. بر طبق آن، این نمایش ترتیبی معمولاً از کارلی لادم بتوحداریت حملنگوته که در بالا بیان شده است، درخت دودویی **T** کامل یا تقریباً کامل است. برای مثال در شکل ۷-۱۰ دارای ۱۱ گره و عمق ۵ است معنی آن، این است که تیزامتد یک آرایه با تقریباً $= 65$ عنصر است.

۴-۷ نمایش درختهای دودویی

سه روش استاندارد برای نمایش یک درخت دودویی **T** یا ریشه **R** وجود دارد این سه الگوریتم که PostOrder ، InOrder و PreOrder تعلیمه می‌شوند به شرح زیر می‌باشند:

روش **PreOrder**

(۱) ریشه 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) ریشه 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) ریشه R را پردازش کنید.

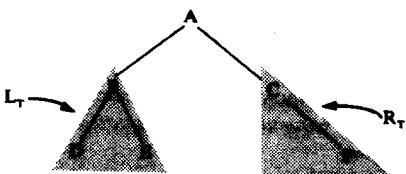
ملاحظه می‌کنید هر الگوریتم دارای همین سه مرحله است و زیردرخت چپ R همیشه قبل از زیردرخت راست پیمایش می‌شود. تفاوت بین این سه الگوریتم، در زمانی است که در آن ریشه R پردازش می‌شود. بطور مشخص، در الگوریتم دارای "Pre" ، ریشه R قبل از پیمایش زیر درخت‌ها پردازش می‌شود. در الگوریتم دارای "In" ، ریشه R مابین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

گاهی اوقات این سه الگوریتم را به ترتیب پیمایش گره - چپ - راست (NLR) پیمایش چپ - گره - راست (LNR) و پیمایش چپ - راست - 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 ۷-۵

درخت دودویی شکل ۱۱-۷ را درنظر بگیرید.



شکل ۱۱-۷

ملاحظه می‌کنید که A ریشه این درخت 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.

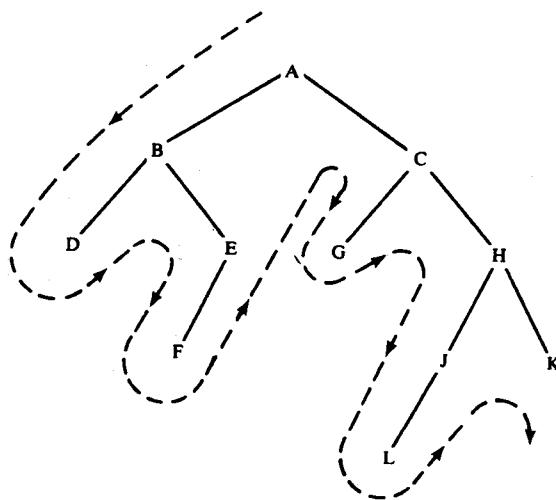
(الف) پیمایش PreOrder درخت T، A را پردازش می‌کند، L_T را پیمایش می‌کند و R_T را پیمایش می‌کند. با وجود این، پیمایش PreOrder زیردرخت L_T ریشه B را پردازش می‌کند و آنگاه D و E را پردازش می‌کند و پیمایش PreOrder زیردرخت R_T ریشه C را پردازش می‌کند و آنگاه F را پردازش می‌کند. از این رو ABDECF پیمایش PreOrder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

(ب) پیمایش InOrder زیردرخت T را پردازش می‌کند، A را پردازش می‌کند و R_T را پیمایش می‌کند. با وجود این، پیمایش InOrder از زیردرخت T، L_T ، D، E و آنگاه E را پردازش می‌کنند و پیمایش InOrder زیردرخت R_T ، C و آنگاه F را پردازش می‌کند. از این رو 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 ۶ - ۷

درخت دودویی شکل ۷-۱۲ را در نظر بگیرید.



شکل ۷-۱۲

پیمایش PreOrder درخت T ، $ABDEFCGHJLK$ است. این ترتیب، همان ترتیبی است که با جستجو و خواندن درخت از چپ، به صورتی که به وسیله مسیر شکل ۱۲-۷ مشخص شده است به دست می‌آید یعنی عمل پیمایش را به طرف پائین در سمت چپ ترین شاخه انجام می‌دهیم تا گردد انتهایی را ملاقات کنیم، آنگاه به عقب بررسی گردیم تا به شاخه بعدی بررسیم و الی آخر. در پیمایش PreOrder سمت راست ترین گره انتهایی، گره K ، آخرین گره‌ای است که خوانده می‌شود. ملاحظه می‌کنید که زیردرخت چپ ریشه A قبل از زیردرخت راست پیمایش می‌شود و هر دو بعد از A پیمایش می‌شوند. این مطلب برای هر گره دیگر زیردرختها برقرار است که خاصیت عمده یک پیمایش PreOrder است.

خواننده با بررسی می‌تواند تحقیق کند که دو روش دیگر پیمایش درخت دودویی شکل ۱۲-۷ 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 ، E ، G ، C ، L ، J ، H و K با همان ترتیب، در تمام سه پیمایش از چپ به راست پیمایش می‌شوند. تأکید می‌کنیم که این مطلب برای هر درخت دودویی T برقرار است.

۷-۷ مثال

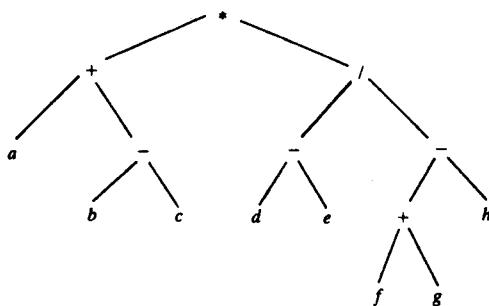
فرض کنید E نمایش عبارت جبری زیر باشد:

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

درخت دودویی T مربوط به آن در شکل ۱۲-۷ 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 - / *

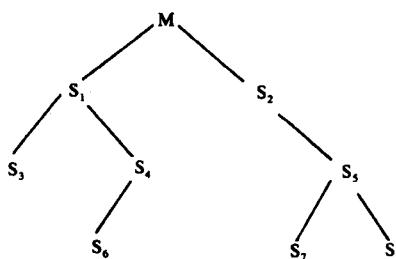
علاوه بر این دانشجو می‌تواند تحقیق کند که این ترتیبها دقیقاً متناظر با نماد پیشوندی و نماد لهستانی پسوندی E است که در بخش ۴-۶ مورد بحث و بررسی قرار گرفت. تأکید می‌کنیم که برای هر عبارت جبری E این ترتیب برقرار است.



شکل - ۱۳

۷-۸ مثال

درخت دودویی T شکل ۷-۱۴ را درنظر بگیرید :



شکل - ۱۴

دانشجو می تواند 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 گره M پردازش می شود. برای مثال، S_6 قبل از S_4 ، S_6 و S_4 قبل از S_1 پردازش می شود. بطور مشابه S_8 و S_7 قبل از S_5 و S_8 و S_7 قبل از S_2 پردازش می شوند. علاوه براین، تمام گره های S_1, S_2, \dots, S_8 قبل از ریشه M پردازش می شوند.

توجه کنید: اگر درختی مانند دو مثال بالا تعداد نسبتاً کمی گره داشته باشد، دانشجو می تواند با بررسی سه پیمایش مختلف درخت دودویی T را پیاده سازی کند. هرگاه درخت T صدها یا هزارها گره داشته باشد پیاده سازی با بررسی، ممکن است امکان پذیر نباشد، به بیان دیگر، نیازمند روش منظم و اصولی

برای پیمایش‌های هستیم که به صورت بازگشتی تعریف می‌شوند. پشته یک ساختمان طبیعی برای پیاده‌سازی به این صورت می‌باشد. بحث و بررسی الگوریتم‌های پشته‌ای برای این منظور، در بخش بعد گنجانده شده است.

۷-۵ الگوریتم‌های پیمایش به کمک پشته‌ها

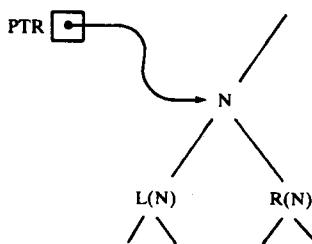
فرض کنید درخت دودویی T به وسیله نمایش پیوندی در حافظه نگهداری می‌شود:

TREE(INFO, LEFT, RIGHT, ROOT)

این بخش پیاده‌سازی سه پیمایش استاندارد T را توضیح می‌دهد که در بخش گذشته به وسیله زیربرنامه‌های غیربازگشتی و با استفاده از پشته‌ها به صورت بازگشتی تعریف شده بود. سه پیمایش رابه صورت مستقل مورد بحث و بررسی قرار می‌دهیم.

پیمایش PreOrder

الگوریتم پیمایش PreOrder از یک متغیر (اشارة گر) PTR استفاده می‌کند که حاوی مکان گره N ای است که در حال حاضر جستجو و خوانده شد. این وضعیت در شکل ۷-۱۵ به تصویر درآمده است که در آن $L(N)$ بجهة چپ گره N و $R(N)$ بجهة راست گره N را نشان می‌دهد. الگوریتم از آرایه STACK نیز استفاده می‌کند، که آدرس گره‌ها را برای پردازش بعدی نگه می‌دارد.



شکل ۷-۱۵

الگوریتم: در آغاز کار N را به داخل $STACK$ push کنید و بدنبال آن قرار دهید $PTR := ROOT$ با وقته که $PTR = NULL$ یا معادل آن $PTR \neq NULL$ است.

(الف) کار پردازش را در سمت چپ‌ترین میسر از ریشه PTR به طرف پائین انجام می‌دهیم، هر گره N در طول مسیر را پردازش کرده و بجهة راست $R(N)$ را در صورت وجود، به داخل $STACK$ Push کنیم. پس از آن که برای یک گره N هیچ چپ $L(N)$ پردازش نشده موجود نباشد پیمایش به پایان می‌رسد.

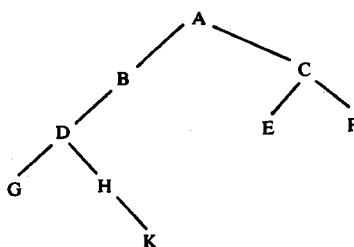
بنابراین PTR با استفاده از دستور جایگزینی $\text{LEFT}[\text{PTR}] := \text{LEFT}[\text{PTR}]$ نازه می‌شود و وقتی $\text{LEFT}[\text{PTR}] = \text{NULL}$ است پیمایش متوقف می‌شود.

(ب) [بازگشت به عقب BackTracking]. POP کنید و در PTR عنصر بالای پشته STACK را جایگزین کنید. اگر $\text{PTR} \neq \text{NULL}$ ، آنگاه به مرحله (الف) برگردید، در غیر اینصورت کار به پایان می‌رسد Exit . یادآور می‌شویم که عنصر اولیه NULL در پشتة STACK به صورت یک نگهبان مورد استفاده قرار گرفته است.

این الگوریتم را در مثال بعد شبیه‌سازی می‌کنیم. اگرچه مثال با خود گره‌ها کار می‌کند اما در کابرد های عملی، مکان گره‌ها در PTR جایگزین می‌شود و به داخل پشتة STACK Push می‌شود.

مثال ۷-۹

درخت دودویی T شکل ۷-۱۶ را در نظر بگیرید.



شکل ۷-۱۶

الگوریتم بالا را با درخت T شبیه‌سازی می‌کنیم، محتوای STACK را در هر مرحله نشان می‌دهیم.

۱- NULL را در آغاز به داخل STACK Push کنید. $\text{STACK} : \emptyset$.

آنگاه قرار دهید $\text{PTR} := \text{A}$ ، که ریشه درخت T است.

۲- از سمت چپ ترین مسیر ریشه $\text{PTR} = \text{A}$ به طرف پائین و به شرح زیر پیش بروید.

(i) A را پردازش کنید و بجهة سمت راست آن، C را به داخل پشتة STACK Push کنید. C

(ii) B را پردازش کنید. هیچ بجهة راست وجود ندارد.

(iii) D را پردازش کنید و بجهة راست آن، H را به داخل پشتة STACK Push کنید.

$\text{STACK}: \emptyset, \text{C}, \text{H}$

(iv) G را پردازش کنید. هیچ بجهة راست وجود ندارد.

از آنجاکه G بچه چپ ندارد، هیچ گره دیگری پردازش نمی‌شود.

- ۳- [بازگشت به عقب PTR := H]. BackTracking عنصر بالا H را از پشته STACK کنید و قرار دهید
درنتیجه:

STACK: \emptyset, C

چون $PTR \neq NULL$ ، به مرحله (الف) الگوریتم برگردید.

- ۴- از سمت چپ ترین مسیر ریشه $H = PTR$ به طرف پائین و به شرح زیر پیش بروید.
(v) H را پردازش کنید و بچه راست آن K ، را به داخل پشته STACK Push کنید: از آنجاکه H بچه چپ
ندارد، هیچ گره پردازش نشده دیگری وجود ندارد،

STACK: \emptyset, C, K

- ۵- [بازگشت به عقب PTR := K]. BackTracking K را از پشته STACK کنید و قرار دهید
درنتیجه:

STACK: \emptyset, C

چون $PTR \neq NULL$ ، به مرحله (الف) الگوریتم برگردید.

- ۶- از سمت چپ ترین مسیر ریشه $K = PTR$ به طرف پائین و به شرح زیر پیش بروید.
(vi) K را پردازش کنید. بچه راست وجود ندارد.
از آنجاکه K بچه چپ ندارد، هیچ گره پردازش نشده دیگری وجود ندارد.

- ۷- [بازگشت به عقب PTR := C]. BackTracking C را از پشته STACK کنید و قرار دهید
درنتیجه:

STACK: \emptyset

چون $PTR \neq NULL$ به مرحله (الف) الگوریتم برگردید.

- ۸- از سمت چپ ترین مسیر ریشه $C = PTR$ به طرف پائین و به شرح زیر پیش بروید:
(vii) C را پردازش کنید و بچه راست آن F را به داخل پشته STACK Push کنید:

STACK: \emptyset, F

را پردازش کنید. بچه راست وجود ندارد.

- ۹- [بازگشت به عقب PTR := F]. BackTracking F را از پشته STACK کنید و قرار دهید
درنتیجه:

STACK: \emptyset

قرار دهید $PTR \neq NULL$ به مرحله (الف) الگوریتم برگردید.

- ۱۰- از سمت چپ ترین مسیر ریشه $F = PTR$ به طرف پائین و به شرح زیر پیش بروید:
(ix) F را پردازش کنید. بچه راست وجود ندارد.

از آنجاکه F بچه چپ ندارد، هیچ گره پردازش نشده دیگری وجود ندارد.

۱۱ - [بازگشت به عقب]. BackTracking عنصر بالا **NULL** را از پشت **STACK** pop کنید و قرار دهید. چون **PTR = NULL** الگوریتم به پایان رسیده است.

همانگونه که از مرحله‌های ۲، ۴، ۶، ۸، ۱۰ نتیجه می‌شود. گره‌های پردازش شده به ترتیب هستند. این همان پیمایش PreOrder مورد نظر درخت T است.

نمایش رسمی الگوریتم پیمایش preOrder به صورت زیر است:

Algorithm 7.1: PREORD(INFO, LEFT, RIGHT, ROOT)

A binary tree T is in memory. The algorithm does a preorder traversal of T, applying an operation PROCESS to each of its nodes. An array STACK is used to temporarily hold the addresses of nodes.

1. [Initially push NULL onto STACK, and initialize PTR.]
Set TOP := 1, STACK[1] := NULL and PTR := ROOT.
2. Repeat Steps 3 to 5 while PTR ≠ NULL:
 3. Apply PROCESS to INFO[PTR].
 4. [Right child?] If RIGHT[PTR] ≠ NULL, then: [Push on STACK.]
Set TOP := TOP + 1, and STACK[TOP] := RIGHT[PTR].
[End of If structure.]
 5. [Left child?] If LEFT[PTR] ≠ NULL, then:
Set PTR := LEFT[PTR].
Else: [Pop from STACK.]
Set PTR := STACK[TOP] and TOP := TOP - 1.
[End of If structure.]
[End of Step 2 loop.]
 6. Exit.

InOrder پیمایش

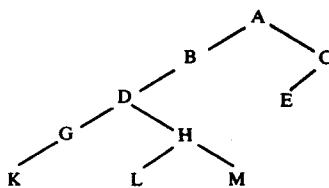
الگوریتم پیمایش InOrder نیز از متغیر اشاره گر PTR استفاده می‌کند که حاوی مکان گره N ای است که به تازگی جستجو و خوانده شد و نیز از یک آرایه STACK استفاده می‌کند که آدرس گره‌ها را برای پردازش بعدی نگه می‌دارد. درواقع، با این الگوریتم یک گره تنها وقتی پردازش می‌شود که از STACK pop شود. الگوریتم: در آغاز کار NULL را به داخل STACK (به عنوان نگهبان) Push کنید و آنگاه قرار دهید **PTR = ROOT**. آنگاه مراحل زیر را تکرار کنید تا وقتی که NULL از STACK pop شود.

(الف) از سمت چپ ترین مسیر ریشه PTR به طرف پائین پیش بروید، هر گره N را به داخل STACK push کنید و این کار را هنگامی متوقف کنید تا گره N هیچ بچه چپ جهت Push شدن در STACK نداشته باشد.

(ب) [بازگشت به عقب pop] BackTracking کنید و گره‌های STACK را پردازش کنید. اگر NULL ، $\text{PTR} := \text{RIGHT}[\text{PTR}]$ باشد آنگاه از الگوریتم خارج شوید Exit. اگر بخواهید گره N با یک بجهه راست $R(N)$ را پردازش کنید قرار دهید $\text{PTR} = R(N)$ (با دستور جایگزینی) و به مرحله (الف) برگردید. تأکید می‌کنیم که گره N تنها وقتی پردازش می‌شود که از pop شده باشد.

مثال ۷-۱۰

درخت دودویی شکل ۷-۱۷ را درنظر بگیرید.



شکل ۷-۱۷

الگوریتم بالا را با T شبیه‌سازی می‌کنیم و محتوای STACK را نشان می‌دهیم.

۱- در آغاز کار NULL را به داخل STACK، Push کنید:

STACK: \emptyset

آنگاه قرار دهید $\text{PTR} := \text{A}$ که ریشه درخت T است.

۲- از سمت چپ ترین مسیر ریشه $\text{PTR} = \text{A}$ به طرف پائین پیش بروید، گره‌های G , D , B , A و K را به داخل STACK، Push کنید:

STACK: $\emptyset, \text{A}, \text{B}, \text{D}, \text{G}, \text{K}$

از آنجاکه K بجهه چپ ندارد، هیچ گره دیگری به داخل STACK، Push نمی‌شود.

۳- [بازگشت به عقب pop] BackTracking گره‌های K , G و D شده پردازش می‌شوند. درنتیجه:

STACK: $\emptyset, \text{A}, \text{B}$

چون D بجهه راست ندارد، پردازش را در D متوقف می‌کنیم. آنگاه قرار دهید $\text{PTR} := \text{H}$ که بجهه راست D است.

۴- از سمت حست راست H مسیر ریشه H به طرف پائین پیش بروید و گره‌های H و L را به داخل

Push ، STACK کنید.

از آنجاکه L بجهه چپ ندارد، هیچ گره دیگری وجود ندارد که به داخل Push ، STACK نشده باشد.

STACK: ϕ, A, B, H, L

۵. [بازگشت به عقب BackTracking]. گره‌های L و H pop شده و پردازش می‌شود. درنتیجه:

STACK: ϕ, A, B

از آنجاکه H بجهه راست دارد، پردازش را در H متوقف می‌کنیم. آنگاه قرار دهید $M := PTR$ که بجهه راست H است.

۶- از سمت چپ ترین مسیر ریشه $M = PTR$ به طرف پائین پیش بروید، گره M را به داخل STACK ، Push کنید.

STACK: ϕ, A, B, M

چون M بجهه چپ ندارد، هیچ گره دیگری به داخل Push ، STACK نمی‌شود.

۷- [بازگشت به عقب BackTracking]. گره‌های M و A pop شده پردازش می‌شوند. درنتیجه:

STACK: ϕ

چون A بجهه راست دارد، هیچ عنصر دیگری از STACK ، pop نمی‌شود. قرار دهید $C := PTR$ ، که بجهه راست A است.

۸- از سمت چپ ترین مسیر ریشه $C = PTR$ به طرف پائین پیش بروید، گره‌های C و E را داخل Push ، STACK کنید.

STACK: ϕ, C, E

۹- [بازگشت به عقب BackTracking]. گره E ، pop شده پردازش می‌شود. چون E هیچ بجهه راست ندارد، گره C ، pop شده و پردازش می‌شود. از آنجاکه C بجهه راست ندارد، عنصر بعدی یعنی NULL ، از STACK ، pop می‌شود.

الگوریتم اکنون به پایان رسیده است، چون NULL از STACK ، pop شده است. همانگونه که از مرحله‌های ۳ ، ۵ ، ۷ و ۹ ملاحظه می‌شود گره‌ها با ترتیب C, E, A, B, M, H, L, D, G, K پردازش می‌شوند. این همان پیمایش InOrder موردنظر درخت دودویی T است.

نمایش رسمی الگوریتم پیمایش InOrder به صورت زیر است:

Algorithm 7.2: INORD(INFO, LEFT, RIGHT, ROOT)

A binary tree is in memory. This algorithm does an inorder traversal of T, applying an operation PROCESS to each of its nodes. An array STACK is used to temporarily hold the addresses of nodes.

1. [Push NULL onto STACK and initialize PTR.]
Set TOP := 1, STACK[1] := NULL and PTR := ROOT.
2. Repeat while PTR ≠ NULL: [Pushes left-most path onto STACK.]
 - (a) Set TOP := TOP + 1 and STACK[TOP] := PTR. [Saves node.]
 - (b) Set PTR := LEFT[PTR]. [Updates PTR.]
 [End of loop.]
3. Set PTR := STACK[TOP] and TOP := TOP - 1. [Pops node from STACK.]
4. Repeat Steps 5 to 7 while PTR ≠ NULL: [Backtracking.]
 5. Apply PROCESS to INFO[PTR].
 6. [Right child?] If RIGHT[PTR] ≠ NULL, then:
 - (a) Set PTR := RIGHT[PTR].
 - (b) Go to Step 3.
 [End of If structure.]
 7. Set PTR := STACK[TOP] and TOP := TOP - 1. [Pops node.]
 [End of Step 4 loop.]
8. Exit.

PostOrder پیمایش

الگوریتم پیمایش PostOrder اندکی پیچیده‌تر از دو الگوریتم قبلی است، زیرا در اینجا لازم است گره N را در دو وضعیت مختلف ذخیره کنیم. مابین این دو حالت از طریق Push کردن N یا N – به داخل STACK تمايز قائل می‌شویم، در کاربردهای عملی، مکان N داخل STACK، مکان N شود، از این رو N – دارای معنی واضحی است. مجدداً از متغیر (اشارة گر) PTR که حاوی مکان گره N است استفاده می‌شود که اخیراً خوانده شده است، این وضعیت در شکل ۱۵-۷ نشان داده شده است.

الگوریتم : در آغاز کار NULL را به داخل STACK به عنوان نگهبان Push کنید و آنگاه قرار دهید := PTR بدنیال آن مراحل زیر را تکرار کنید تا NULL از STACK pop شود.

(الف) از سمت چپ ترین مسیر ریشه PTR به طرف پائین پیش بروید. در هر گره N از مسیر، N را به داخل STACK، STACK Push کنید و اگر N بجهة راست R(N) دارد، R(N) – را به داخل STACK، STACK Push کنید.

(ب) [BackTracking] بازگشت به عقب pop کنید و گره‌های مثبت داخل STACK را پردازش کنید. اگر بعضی گرهات NULL شد، آنگاه از الگوریتم خارج شوید Exit. اگر یک گره منفی pop شد، یعنی اگر به ازای تأکید می‌کنیم که گره N تنها وقتی پردازش می‌شود که از STACK pop شود و مثبت باشد.

مثال ۷-۱۱

بار دیگر درخت دودویی شکل ۷-۱۷ را در نظر بگیرید. الگوریتم بالا را با T شبیه سازی می کنیم و محتوای $STACK$ را نشان می دهیم.

۱- در آغاز $NULL$ را به داخل $STACK$ کنید و فوار دهد $A := PTR$ که ریشه T است.

$STACK: \emptyset$

۲- از سمت چپ ترین مسیر ریشه $A = PTR$ به طرف پائین پیش بروید، گره های G, D, B, A و K را به داخل $STACK$ کنید. علاوه بر این، چون A بجهة راست C دارید، $-C$ را پس از A اما قبل از B به داخل $STACK$ کنید و چون D بجهة راست H دارد، $-H$ را پس از D اما قبل از G به داخل $STACK$ کنید. درنتیجه $Push$

$STACK: \emptyset, A, - C, B, D, - H, G, K$

۳- [بازگشت به عقب pop کنید و K را پردازش کنید همچنین pop کنید و G پردازش کنید. چون H - منفی است، تنها $-H$ ، pop می شود. درنتیجه :

$STACK: \emptyset, A, - C, B, D$

اکنون $PTR = -H$. مجدداً قرار دهید $H = PTR$ و به مرحله (الف) برگردید.

۴- از سمت چپ ترین مسیر ریشه $H = PTR$ به طرف پائین پیش بروید، نخست H را به داخل $STACK$ کنید. چون H بجهة راست M دارد، $-M$ را پس از H به داخل $STACK$ کنید. بالاخره، L را به داخل $STACK$ کنید. درنتیجه :

$STACK: \emptyset, A, - C, B, D, H, - M, L$

۵- [بازگشت به عقب pop کنید و L را پردازش کنید، اما فقط $-M$ را pop کنید. درنتیجه :

$STACK: \emptyset, A, - C, B, D, H$

حال M . قرار دهید $M = PTR$ و به مرحله (الف) برگردید.

۶- از سمت چپ ترین مسیر ریشه $M = PTR$ به طرف پائین پیش بروید. اکنون، تنها M را به داخل $STACK$ کنید. درنتیجه :

$STACK: \emptyset, A, - C, B, D, H, M$

۷- [بازگشت به عقب pop کنید و D, H, M و B را پردازش کنید اما فقط $C -$ را pop کنید. درنتیجه :

$STACK: \emptyset, A$

حال دهید $PTR = -C$. قرار دهید $C = PTR$ و به مرحله (الف) برگردید.

از سمت چپ‌ترین مسیر ریشه $C = PTR$ به طرف پائین پیش بروید. نخست C به داخل STACK، $STACK = E, A, C, \emptyset$ می‌شود و آنگاه Push درنتیجه :

$STACK: \emptyset, A, C, E$

بازگشت به عقب [BackTracking] pop کنید و A را پردازش کنید. وقتی $NULL$ ، C و E را پردازش کنید. خالی است و الگوریتم کامل می‌شود.

همانگونه که از مراحل ۳، ۵، ۷ و ۹ ملاحظه می‌شود، گره‌ها با ترتیب $K, G, L, M, H, D, B, C, E$ با ترتیب T پردازش می‌شوند. این همان پیمایش PostOrder مورد نظر درخت دودویی است.

نمایش رسمی الگوریتم پیمایش Postorder به شرح زیر است:

Algorithm 7.3: POSTORD(INFO, LEFT, RIGHT, ROOT)

A binary tree T is in memory. This algorithm does a postorder traversal of T , applying an operation PROCESS to each of its nodes. An array STACK is used to temporarily hold the addresses of nodes.

1. [Push NULL onto STACK and initialize PTR.]
Set TOP := 1, STACK[1] := NULL and PTR := ROOT.
2. [Push left-most path onto STACK.]
Repeat Steps 3 to 5 while PTR ≠ NULL:
 3. Set TOP := TOP + 1 and STACK[TOP] := PTR.
[Pushes PTR on STACK.]
 4. If RIGHT[PTR] ≠ NULL, then: [Push on STACK.]
Set TOP := TOP + 1 and STACK[TOP] := -RIGHT[PTR].
[End of If structure.]
 5. Set PTR := LEFT[PTR]. [Updates pointer PTR.]
[End of Step 2 loop.]
 6. Set PTR := STACK[TOP] and TOP := TOP - 1.
[Pops node from STACK.]
 7. Repeat while PTR > 0:
 - (a) Apply PROCESS to INFO[PTR].
 - (b) Set PTR := STACK[TOP] and TOP := TOP - 1.
[Pops node from STACK.]
[End of loop.]
 8. If PTR < 0, then:
 - (a) Set PTR := -PTR.
 - (b) Go to Step 2.
[End of If structure.]
 9. Exit.

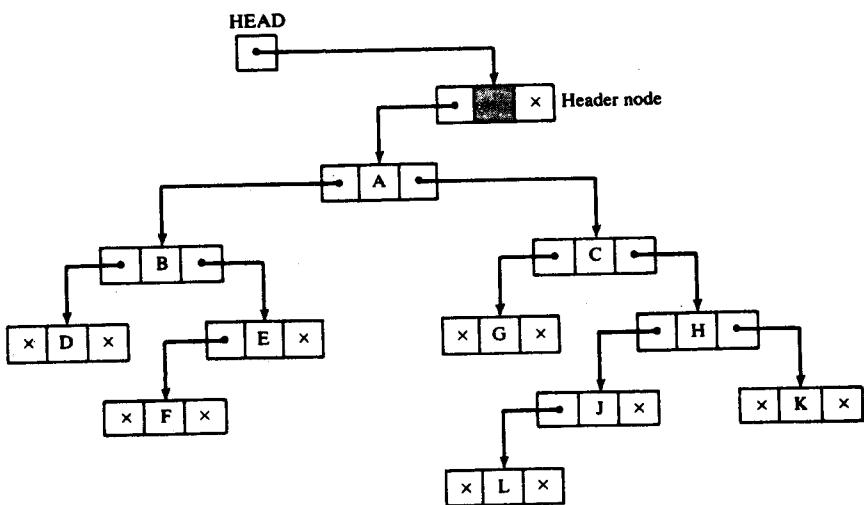
۶- سرگرهای نخکشی شده

درخت دودویی T را درنظر بگیرید. اغلب انواع مختلف از نمایش پیوندی درخت T مورد استفاده

قرار می‌گیرد. زیرا برخی از عملیات اصلاح شده روی T ، به منظور پیاده‌سازی ساده‌تر می‌باشند. برخی از این انواع نمایش که مشابه لیستهای دارای سرگره و لیستهای پیوندی چرخشی یا حلقوی است در این بخش مورد بحث و بررسی قرار می‌گیرد.

سرگرهای

درخت دودویی T را درنظر بگیرید که به وسیله یک نمایش پیوندی در حافظه نگهداری می‌شود. گاهی اوقات گره اضافی خاصی موسوم به سرگره به ابتدای T اضافه می‌شود. هنگامی که این گره اضافی مورد استفاده قرار می‌گیرد، سه متغیر اشاره‌گر که آنها را **HEAD** (به جای **ROOT**) می‌نامیم و به سرگره اشاره می‌کند همچنین اشاره‌گر چپ سرگره که به ریشه T اشاره می‌کند. شکل ۷-۱۸ تصویر درخت دودویی شکل ۱-۷ را نشان می‌دهد که از نمایش پیوندی با یک سرگره استفاده می‌کند، آن را با شکل ۶-۶ مقایسه کنید.



شکل ۷-۱۸

فرض کنید درخت دودویی T خالی است. آنگاه T همچنان حاوی یک سرگره است، اما اشاره‌گر چپ سرگره حاوی مقدار پرج **NULL** است. بدین ترتیب شرط

$$\text{LEFT}[\text{HEAD}] = \text{NULL}$$

بیان می‌کند که درخت خالی است.

روش دیگر نمایش بالا برای یک درخت دودویی T ، از سرگره به عنوان نگهبان استفاده می‌کند. به بیان دیگر، اگر یک گره زیردرخت خالی داشته باشد، آنگاه فیلد اشاره‌گر زیر درخت حاوی آدرس سرگره به عوض مقدار پوج **NULL** است. برطبق آن، هیچ اشاره‌گری آدرس غیرمجاز ندارد و شرط

$$\text{LEFT}[\text{HEAD}] = \text{HEAD}$$

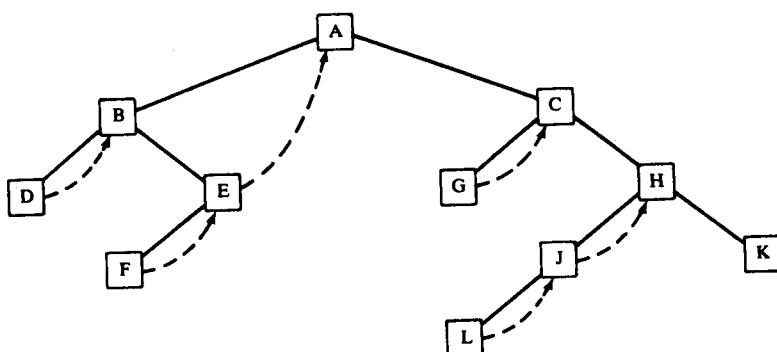
بیان می‌کند که زیردرخت خالی است.

نخ‌کشی‌ها، نخ‌کشی **InOrder**

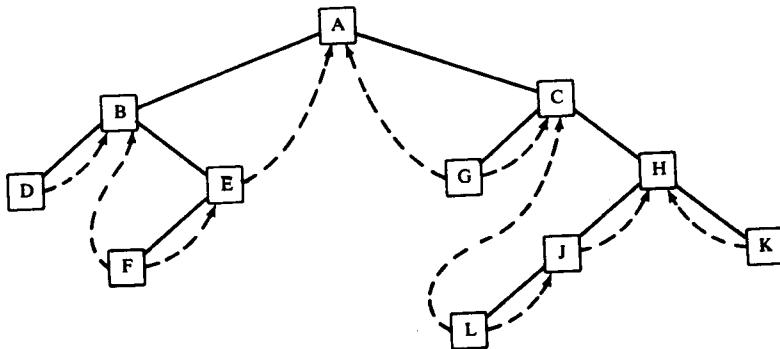
بار دیگر نمایش پیوندی درخت دودویی T را درنظر بگیرید. تقریباً نصف ورودیهای فیلدهای اشاره‌گر **LEFT** و **RIGHT** شامل عناصر پوج **NULL** است. این فضای قراردادن نوع دیگری از اطلاعات به جای ورودیهای پوج می‌تواند به شکل کارتاری مورد استفاده قرار گیرد. بطور مشخص ما اشاره‌گرهای خاصی را جانشین ورودیهای پوج می‌کنیم که به گره‌های بالاتر درخت اشاره می‌کند. این اشاره‌گرهای خاص را نخ‌کشی‌ها و درخت دودویی با این اشاره‌گرهای را درخت‌های نخ‌کشی شده می‌گویند.
نخ‌کشی‌ها در یک درخت نخ‌کشی شده، باید به طریقی از اشاره‌گرهای معمولی تمیز داده شوند. در نمودار یک درخت نخ‌کشی شده، نخ‌کشی‌ها را معمولاً با خط‌چینها نمایش می‌دهند. در حافظه کامپیوتر یک فیلد **TAG**، یک بیتی اضافی را می‌توان برای تمایز نخ‌کشی از اشاره‌گرهای معمولی، مورد استفاده قرار داد. یا به بیان دیگر، نخ‌کشی‌ها را وقتی اشاره‌گرهای معمولی با اعداد صحیح مثبت نمایش داده می‌شوند می‌توان با اعداد صحیح منفی نشان داد.

برای نخ‌کشی یک درخت دودویی راههای متعدد وجود دارد اما هر نخ‌کشی متناظر با یک پیمایش خاص T است. علاوه بر این می‌توان نخ‌کشی یکطرفه یا نخ‌کشی دوطرفه را انتخاب کرد. نخ‌کشی ما متناظر با پیمایش **InOrder** درخت T است مگر آن که خلاف آن بیان شود. بنابراین در نخ‌کشی یکطرفه T ، یک نخ‌کشی در فیلد **RIGHT** راست یک گره ظاهر می‌شود و به گره بعدی در پیمایش **InOrder** اشاره خواهد کرد و در نخ‌کشی دوطرفه T ، یک نخ‌کشی نیز در فیلد **LEFT** یک گره ظاهر می‌شود و به گره قبلی در پیمایش **InOrder** اشاره خواهد کرد. علاوه بر این وقتی T سرگره ندارد اشاره‌گر چپ‌گره اول و اشاره‌گر راست‌گره آخر (در پیمایش **InOrder** درخت T) حاوی مقدار پوج **NULL** است، اما وقتی T یک سرگره دارد به سرگره اشاره می‌کند.

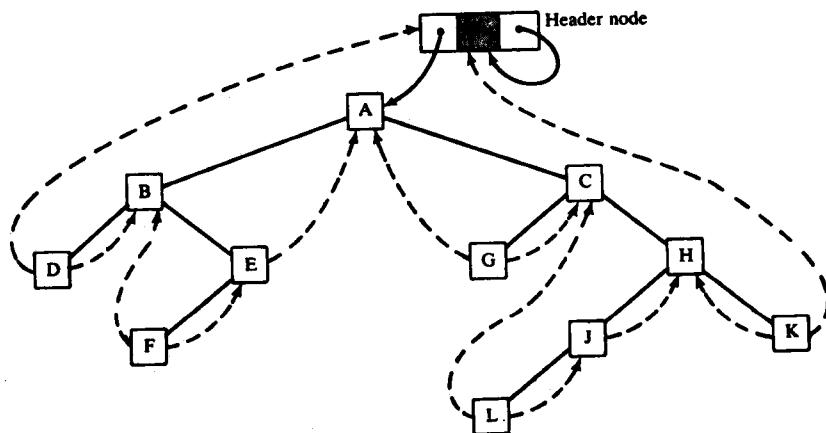
مشابه نخ‌کشی یکطرفه یک درخت دودویی T ، نخ‌کشی‌ای وجود دارد که متناظر با پیمایش **PreOrder** درخت T است. مسأله ۱۳-۷ را بینید. از طرف دیگر، هیچ نخ‌کشی‌ای وجود ندارد که متناظر با پیمایش **PostOrder** درخت T باشد.



(الف) نخ کشی InOrder یک طرفه



(ب) نخ کشی InOrder دو طرفه



(ج) نخ کشی دو طرفه با سرگره

مثال ۷-۱۲

درخت دودویی T شکل ۱-۷ را درنظر بگیرید.

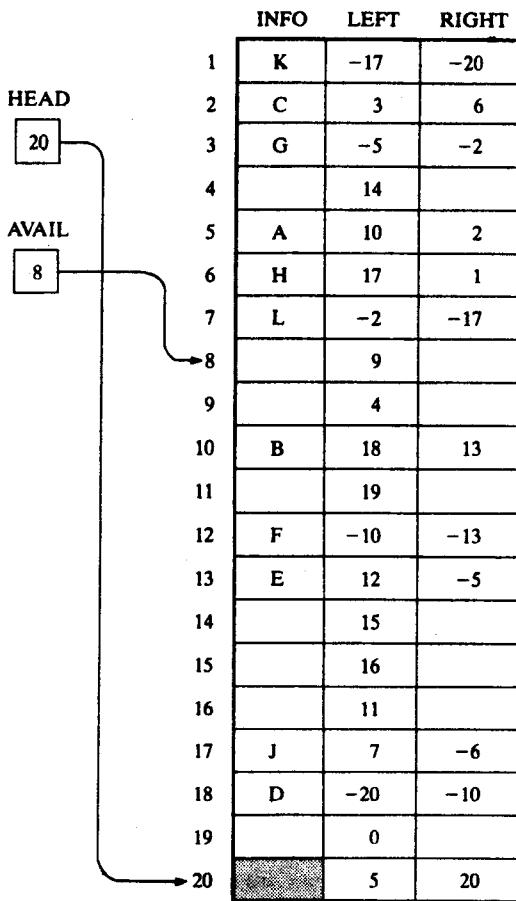
(الف) نخ کشی InOrder یک طرفه T در شکل ۱۹-۷ (الف) رسم شده است. از آنجا که در پیمایش InOrder درخت T، به A می‌توان پس از E دسترسی پیدا کرد، یک نخ کشی از گره E به گره A وجود دارد. ملاحظه می‌کنید یک نخ با استثنای گره K جانشین هر اشاره‌گر راست پوچ شده است که گره آخر در پیمایش InOrder درخت T است.

(ب) نخ کشی Inorder دوطرفه T در شکل ۱۹-۷ (ب) رسم شده است. از آنجا که در پیمایش InOrder درخت T، به L می‌توان پس از C دسترسی پیدا کرد، یک نخ کشی چپ از گره L به گره C وجود دارد. ملاحظه می‌کنید که یک نخ با استثنای گره D جانشین هر اشاره‌گر چپ پوچ شده است که تخلیه گره در پیمایش InOrder درخت T است. تمام نخ کشی‌های راست همانند شکل ۱۹-۷ (الف) هستند.

(ج) نخ کشی InOrder دوطرفه درخت T وقتی T یک سرگره دارد در شکل ۱۹-۷ (ج) رسم شده است. در اینجا نخ کشی چپ D و نخ کشی راست K به سرگره اشاره می‌کنند. نخ کشی‌های دیگر به همان صورتی است که در شکل ۱۹-۷ (ب) ارائه شده است.

(د) شکل ۷-۷ چگونگی نگهداری T را در حافظه با استفاده از یک نمایش پیوندی نشان می‌دهد. شکل ۲۰-۷ نشان می‌دهد که چگونه این نمایش باید اصلاح شود تا T با استفاده از INFO[20] به عنوان یک سرگره یک درخت نخ کشی شده InOrder دوطرفه باشد.

توجه می‌کنید که $10 = \text{LEFT}[12]$ ، به بیان دیگر یک نخ کشی چپ از گره F به گره B وجود دارد. بطور مشابه، $6 = \text{RIGHT}[17]$ به این معنی است که یک نخ کشی راست از گره J به گره H وجود دارد. بالاخره ملاحظه می‌کنید که $20 = \text{RIGHT}[20]$ به عبارت دیگر یک اشاره‌گر راست معمولی از سرگره به خودش وجود دارد. اگر T خالی باشد، آنگاه قرار دهید $20 = \text{LEFT}[20]$ و به این معنی است که یک نخ کشی چپ از سرگره به خودش وجود دارد.



شکل ۷-۲۰

۷-۷ درختهای جستجوی دودویی

این بخش یکی از مهم‌ترین ساختمندان داده علم کامپیوتر یعنی یک درخت جستجوی دودویی را مورد بحث و بررسی قرار می‌دهد. این ساختمنان به ما امکان می‌دهد تا یک عنصر را جستجو کنیم و آن را با زمان اجرای میانگین $O(f(n)) = O(\log_2 n)$ پیدا کنیم. علاوه بر این بمسادگی می‌توان عنصر را در این ساختمنان داده اضافه کرد یا از آن حذف کرد. این ساختمنان داده در مقابل ساختمنان‌های زیر قرار دارد:

(الف) آرایه مرتب شده خطی. در اینجا می‌توان یک عنصر را جستجو کرد و آن را با زمان اجرای میانگین

$f(n) = O(\log_2 n)$ پیدا کرد اما اضافه کردن و حذف عنصر پرهزینه است.

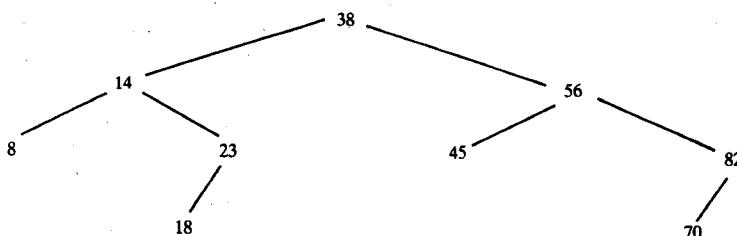
(ب) لیست پیوندی. در اینجا به مسادگی می‌توان عناصر را اضافه یا حذف کرد، اما در این روش جستجوی عنصر و پیداکردن آن پرهزینه است، چون باید از جستجوی خطی با زمان اجرای $f(n) = O(n)$ استفاده کرد.

هرچند هرگره در یک درخت جستجوی دودویی می‌تواند شامل تمام رکوردها باشد اما تعریف درخت دودویی بستگی به فیلد داده شده دارد که مقادیر آن متمایز هستند و می‌توانند مرتب شده باشد. فرض کنید T یک درخت دودویی باشد. آنگاه T یک درخت جستجوی دودویی (یا درخت

مرتب شده دودویی) نامیده می‌شود اگر هر گره N درخت T دارای خاصیت زیر باشد: مقادار در N بزرگتر از هر مقدار در زیردرخت چپ N و کوچکتر از هر مقدار در زیردرخت راست N است. به آسانی ملاحظه می‌شود که این خاصیت تضمین می‌کند که پیمایش InOrder درخت T باعث می‌شود لیست عناصر T مرتب شده باشند.

مثال ۷-۱۳

درخت دودویی T شکل ۷-۲۱ را درنظر بگیرید.



شکل ۷-۲۱

یک درخت جستجوی دودویی است، یعنی هر گره N در T از هر عدد زیردرخت چپ آن بزرگتر و از هر عدد زیردرخت راست آن کوچکتر است. فرض کنید عدد 35 جانشین عدد 23 شده است آنگاه T همچنان یک درخت جستجوی دودویی خواهد بود. ازطرف دیگر اگر عدد 40 را جایگزین عدد 23 کنیم T یک درخت جستجوی دودویی نخواهد بود، چون در زیردرخت چپ عدد 38 بزرگتر از 40 نیست.

(ب) فایل شکل ۷-۸ را درنظر بگیرید. همانگونه که در شکل ۹-۷ گفته شد، این فایل نسبت به کلید NAME یک درخت جستجوی دودویی است. ازطرف دیگر این فایل نسبت به کلید شماره تأمین اجتماعی SSN یک درخت جستجوی دودویی نیست. این وضعیت مشابه یک آرایه از رکوردها است که

نسبت به یک کلید مرتب شده اما نسبت به هیچ کلید دیگر مرتب شده نباشد. تعریف یک درخت جستجوی دودویی داده شده در این بخش براین فرض استوار است که مقادیر تمام گره‌ها متمایز هستند. تعریف مشابه‌ای از یک درخت جستجوی دودویی وجود دارد که اجازه مساوی بودن دو مقدار را به ما می‌دهد به بیان دیگر، هر گره N در آن دارای خاصیت زیر است: مقدار در N بزرگتر از هر مقدار در زیردرخت چپ N و کوچکتر یا مساوی هر مقدار در زیردرخت راست N است. هنگام استفاده از این تعریف، عملیات بخش بعد باید براساس آن تغییر کند.

۷-۸ جستجو و واردکردن یک عنصر در درختهای جستجوی دودویی

فرض کنید T یک درخت جستجوی دودویی باشد. این بخش عملیات اصلی و پایه جستجو و واردکردن یک عنصر را در T توضیح می‌دهد. درواقع، جستجوکردن و واردکردن یک عنصر، تنها با یک الگوریتم جستجو و واردکردن انجام می‌شود. عمل حذف عنصر در بخش بعد بررسی می‌شود. پیمايش درخت T دقیقاً مانند پیمايش هر درخت دودویی است، این مبحث در بخش ۷-۴ گنجانده شده است. فرض کنید عنصر اطلاعاتی $ITEM$ داده شده است. الگوریتم زیر مکان $ITEM$ را در درخت جستجوی دودویی پیدا می‌کند یا $ITEM$ را به عنوان یک گره جدید در مکان مربوطه اش در درخت اضافه می‌کند.

(الف) $ITEM$ را با N گره ریشه درخت مقایسه کنید.

(i) اگر $ITEM < N$ ، به طرف بجهة چپ N پیش بروید.

(ii) اگر $ITEM > N$ ، به طرف بجهة راست N پیش بروید.

(ب) مرحله (الف) را تکرار کنید تا یکی از حالت‌های زیر اتفاق بیفتد:

(i) گره N را وقتی $ITEM = N$ است ملاقات کنید. در این حالت جستجو موفق است.

(ii) یک زیردرخت خالی را ملاقات کنید که بیان می‌کند جستجو موفق نیست و $ITEM$ را به جای زیردرخت خالی اضافه کنید.

به بیان دیگر، از ریشه R در درخت T به طرف پائین پیش بروید تا $ITEM$ در T پیدا شود یا $ITEM$ را به عنوان گره انتهايی در T اضافه کنید.

مثال ۷-۱۴

(الف) درخت جستجوی دودویی T شکل ۷-۲۱ را در نظر بگیرید. فرض کنید $ITEM = 20$ داده شده است. با شبیه‌سازی الگوریتم بالا به مرحله‌های زیر می‌رسیم:
 $ITEM = 20$ را با ریشه درخت T یعنی 38 مقایسه کنید. چون $38 < 20$ به طرف بجهة چپ 38 که 14

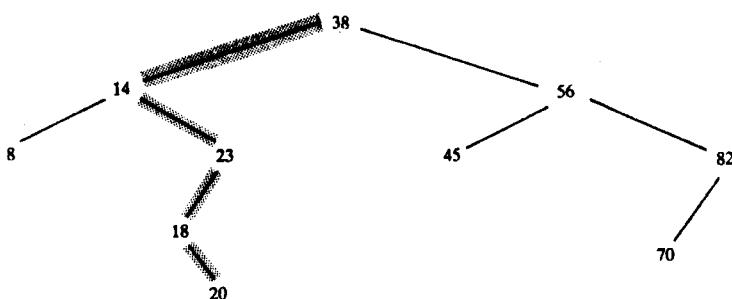
است پیش بروید.

۲ - $ITEM = 20$ را با 14 مقایسه کنید. چون $14 < 20$ به طرف بجهه راست 14 که 23 است پیش بروید.

۳ - $ITEM = 20$ را با 23 مقایسه کنید. چون $23 > 20$ به طرف بجهه چپ 23 که 18 است پیش بروید.

۴ - $ITEM = 20$ را با 18 مقایسه کنید. چون $18 < 20$ و 18 بجهه راست ندارد. 20 را به عنوان بجهه راست 18 اضافه کنید.

شکل ۷-۲۲ درخت جدیدی را نشان می‌دهد که عنصر $20 = ITEM$ به آن اضافه شده است.



شکل ۷-۲۲ $ITEM = 20$ اضافه شده است.

بالهای هاشورخورده و سایه‌دار، بیان می‌کند که این مسیر هنگام اجرای الگوریتم تا پائین طی می‌شود.

(ب) درخت جستجوی دودویی T شکل ۹-۷ را در نظر بگیرید. فرض کنید $ITEM = Davis$ داده شده است. با شبیه‌سازی الگوریتم بالا، مراحل زیر حاصل می‌شود:

۱ - $ITEM = Davis$ را با ریشه درخت، $Harris$ مقایسه کنید. چون $Davis < Harris$ ، به طرف بجهه چپ $Harris$ که $Cohen$ است پیش بروید.

۲ - $ITEM = Davis$ را با $Cohen$ مقایسه کنید. چون $Davis > Cohen$ ، به طرف بجهه راست $Cohen$ که $Green$ است پیش بروید.

۳ - $ITEM = Davis$ را با $Green$ مقایسه کنید. چون $Davis < Green$ ، به طرف بجهه چپ $Green$ که در $Davis$ است پیش بروید.

۴ - $ITEM = Davis$ را با بجهه چپ $Davis$ مقایسه کنید. از این‌رو مکان $Davis$ در درخت پیدا شده است.

مثال ۷-۱۵

فرض کنید شش عدد زیر به ترتیب در یک درخت جستجوی دودویی خالی اضافه شده است :

40, 60, 50, 33, 55, 11

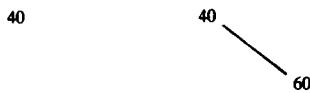
شکل ۷-۲۳ ۷ شش مرحله از درخت را نشان می‌دهد. تأکید می‌کنیم که اگر شش عدد داده شده با ترتیب مختلف داده شده باشد، آنگاه درخت‌ها ممکن است با هم فرق کنند و عمق مختلف داشته باشند. نمایش رسمی الگوریتم جستجو و اضافه کردن از زیربرنامه **Procedure** زیر استفاده می‌کند، که مکان یک ITEM داده شده و پدر آن را پیدا می‌کند. زیربرنامه با استفاده از اشاره گر PTR و اشاره گر SAVE برای گره پدر به طرف پائین درخت را پیمایش می‌کند. این زیربرنامه در بخش بعد هنگام حذف عنصرها مورد استفاده قرار خواهد گرفت:

Procedure 7.4: FIND(INFO, LEFT, RIGHT, ROOT, ITEM, LOC, PAR)

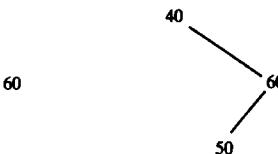
A binary search tree T is in memory and an ITEM of information is given. This procedure finds the location LOC of ITEM in T and also the location PAR of the parent of ITEM. There are three special cases:

- (i) LOC = NULL and PAR = NULL will indicate that the tree is empty.
 - (ii) LOC ≠ NULL and PAR = NULL will indicate that ITEM is the root of T.
 - (iii) LOC = NULL and PAR ≠ NULL will indicate that ITEM is not in T and can be added to T as a child of the node N with location PAR.
1. [Tree empty?]
If ROOT = NULL, then: Set LOC := NULL and PAR := NULL, and Return.
 2. [ITEM at root?]
If ITEM = INFO[ROOT], then: Set LOC := ROOT and PAR := NULL, and Return.
 3. [Initialize pointers PTR and SAVE.]
If ITEM < INFO[ROOT], then:
 Set PTR := LEFT[ROOT] and SAVE := ROOT.
Else:
 Set PTR := RIGHT[ROOT] and SAVE := ROOT.
[End of If structure.]
 4. Repeat Steps 5 and 6 while PTR ≠ NULL:
 5. [ITEM found?]
 If ITEM = INFO[PTR], then: Set LOC := PTR and PAR := SAVE, and Return.
 6. If ITEM < INFO[PTR], then:
 Set SAVE := PTR and PTR := LEFT[PTR].
 Else:
 Set SAVE := PTR and PTR := RIGHT[PTR].
 [End of If structure.]
 7. [End of Step 4 loop.]
8. [Search unsuccessful.] Set LOC := NULL and PAR := SAVE.
 8. Exit.

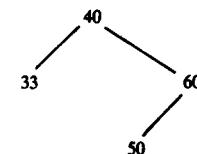
در مرحله ۶ ملاحظه می‌کنید که بسته به این که **ITEM > INFO[PTR]** یا **ITEM < INFO[PTR]** به طرف بجهه چپ یا بجهه راست حرکت می‌کنیم.



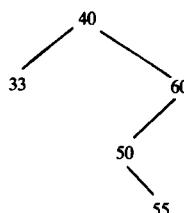
(1) ITEM = 40.



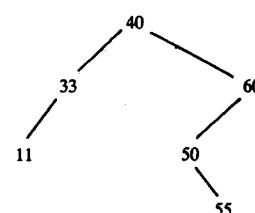
(2) ITEM = 60.



(3) ITEM = 50.



(5) ITEM = 55.



(6) ITEM = 11.

شكل ۷-۲۳

بيان رسمي الگوریتم جستجو و اضافه کردن به شرح زیر است :

Algorithm 7.5: INSBST(INFO, LEFT, RIGHT, ROOT, AVAIL, ITEM, LOC)

A binary search tree T is in memory and an ITEM of information is given. This algorithm finds the location LOC of ITEM in T or adds ITEM as a new node in T at location LOC.

1. Call FIND(INFO, LEFT, RIGHT, ROOT, ITEM, LOC, PAR).
[Procedure 7.4.]
2. If LOC ≠ NULL, then Exit.
3. [Copy ITEM into new node in AVAIL list.]
 - (a) If AVAIL = NULL, then: Write: OVERFLOW, and Exit.
 - (b) Set NEW := AVAIL, AVAIL := LEFT[AVAIL] and INFO[NEW] := ITEM.
 - (c) Set LOC := NEW, LEFT[NEW] := NULL and RIGHT[NEW] := NULL.
4. [Add ITEM to tree.]
 - If PAR = NULL, then:
Set ROOT := NEW.
 - Else if ITEM < INFO[PAR], then:
Set LEFT[PAR] := NEW.
 - Else:
Set RIGHT[PAR] := NEW.

[End of If structure.]
5. Exit.

در مرحله ۴ ملاحظه می‌کنید که سه حالت ممکن وجود دارد: (۱) درخت خالی باشد (۲) ITEM به عنوان یک بجهه چپ اضافه شود و (۳) ITEM به عنوان یک بجهه راست اضافه شود.

پیچیدگی الگوریتم جستجوی عنصر

فرض کنید در یک درخت جستجوی دودویی T می‌خواهیم یک عنصر اطلاعاتی را جستجو کنیم. ملاحظه می‌کنید که تعداد مقایسه‌ها محدود به عمق درخت می‌شود. این موضوع از این واقعیت ناشی می‌شود که ما از یک مسیر درخت به طرف پائین پیش می‌رویم. بنابراین، زمان اجرای جستجو متناسب با عمق درخت است.

فرض کنید n عنصر اطلاعاتی A_1, A_2, \dots, A_N داده شده است و فرض کنید عناصر به ترتیب در یک درخت جستجوی دودویی T اضافه می‌شوند. یادآوری می‌کنیم که برای n عنصر تعداد $n!$ جایگشت وجود دارد (بخش ۲-۲). هر یک از چنین جایگشتی باعث به وجود آمدن درخت مربوط به خود می‌شود. می‌توان نشان داد که عمق میانگین $n!$ درخت تقریباً برابر $\log_2 n$ است که در آن $c = 1.4$. بنابراین، زمان اجرای میانگین $f(n)$ جستجو یک عنصر در درخت دودویی T با n عنصر متناسب با $f(n) = O(\log_2 n)$ است یعنی $\log_2 n$

کاربرد درختهای جستجوی دودویی

مجموعه‌ای از n عنصر اطلاعاتی A_1, A_2, \dots, A_N را درنظر بگیرید. فرض کنید بخواهیم تمام عناصر تکراری را که در این مجموعه وجود دارند پیدا کرده آنها را حذف کنیم. یک راه ساده برای این منظور به شرح زیر است:

الگوریتم A : عناصر را از A_1 تا A_N یعنی از چپ به راست بخوانید.

(الف) برای هر عنصر A_K ، A_K را با A_1, A_2, \dots, A_{K-1} مقایسه کنید یعنی A_K را با عناصری که قبل از A_K هستند مقایسه کنید.

(ب) اگر A_K در بین A_1, A_2, \dots, A_{K-1} وجود داشت، آنگاه A_K را حذف کنید.

پس از آن که تمام عناصر خوانده شده و مورد بررسی قرار گرفت، آنگاه در این مجموعه عناصر تکراری نخواهد بود.

مثال ۷-۱۶

فرض کنید الگوریتم A بر لیست ۱۵ عددی زیر بکار گرفته شد:

ملاحظه می‌کنید که چهار عدد اول یعنی 14، 10، 17 و 12 حذف نمی‌شوند. بنابراین:

$$A_5 = A_2 = 10$$

$$A_8 = A_4 = 12$$

$$A_{11} = A_7 = 20$$

$$A_{14} = A_6 = 11$$

هنگامی که اجرای الگوریتم A به پایان می‌رسد، ۱۱ عدد

14, 10, 17, 12, 11, 20, 18, 25, 8, 22, 23

که همگی متمایز هستند باقی می‌ماند.

حال پیجیدگی زمانی الگوریتم A را درنظر بگیرید که به وسیله تعداد مقایسه‌ها تعیین می‌شود. قبل از همه، فرض می‌کنیم که n تعداد عناصر دوتایی تکراری در مقایسه با n تعداد عناصر اطلاعاتی بسیار کوچک است. ملاحظه می‌کنید مرحله‌ای که شامل A_K است به طور تقریبی به $K-1$ مقایسه احتیاج دارد چون A_k با عنصرهای A_{K-1}, \dots, A_2, A_1 (کمتر ممکن است تابحال حذف شده باشد) مقایسه می‌شود. بنابراین، $f(n)$ تعداد مقایسه‌ها مورد نیاز در الگوریتم A تقریباً برابر

$$0 + 1 + 2 + 3 + \dots + (n - 2) + (n - 1) = \frac{(n - 1)n}{2} = O(n^2)$$

است. برای مثال، برای $n = 1000$ عنصر، الگوریتم A تقریباً به 500 000 مقایسه احتیاج دارد. به بیان دیگر، زمان اجرای الگوریتم A متناسب با n^2 است.

با استفاده از یک درخت جستجوی دودویی می‌توان الگوریتم دیگر نوشت که عناصر دوتایی تکراری را از یک مجموعه n عنصری A_1, A_2, \dots, A_N پیدا کند.

الگوریتم B: با استفاده از عناصر A_1, A_2, \dots, A_N یک درخت جستجوی دودویی بسازید. هنگام ساختن درخت، در صورتی که مقدار A_k قبل از درخت ظاهر شده باشد A_k را از لیست حذف کنید.

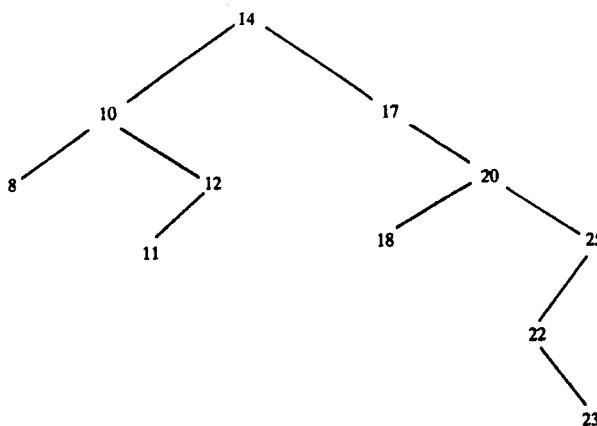
مزیت اصلی الگوریتم B آن است که هر عنصر A_k تنها با عناصرهای یک شاخه درخت مقایسه می‌شود. می‌توان نشان داد که طول میانگین چنین شاخه‌ای تقریباً برابر $\log_2 K$ است که در آن $c = 1.4$. بنابراین $f(n)$ تعداد کل مقایسه‌های مورد نیاز در الگوریتم B تقریباً به $n \log_2 n$ مقایسه نیاز دارد. برای مثال، برای $n = 1000$ ، الگوریتم B مستلزم تقریباً 10000 مقایسه است که در الگوریتم A، تعداد مقایسه‌ها برابر 500000 است. متذکر می‌شویم که در بدترین حالت، تعداد مقایسه‌های الگوریتم B، برابر تعداد مقایسه‌های الگوریتم A است.

مثال ۷-۱۷

مجدداً لیست ۱۵ عددی زیر را درنظر بگیرید:

14, 10, 17, 12, 10, 11, 20, 12, 18, 25, 20, 8, 22, 11, 23

با اعمال الگوریتم B بر این لیست عددی، درخت شکل ۷-۲۴ به دست می‌آید:



شکل ۷-۲۴

تعداد دقیق مقایسه‌ها برابر است با:

$$0 + 1 + 1 + 2 + 2 + 3 + 2 + 3 + 3 + 3 + 2 + 4 + 4 + 5 = 38$$

از طرف دیگر الگوریتم A نیازمند

$$0 + 1 + 2 + 3 + 2 + 4 + 5 + 4 + 6 + 7 + 6 + 8 + 9 + 5 + 10 = 27$$

مقایسه است.

۷-۹ حذف یک عنصر از یک درخت جستجوی دودویی

فرض کنید T یک درخت جستجوی دودویی است و عنصر اطلاعاتی ITEM داده شده است. این بخش الگوریتمی را ارائه می‌دهد که عنصر ITEM را از درخت T حذف می‌کند.

الگوریتم حذف در وهله اول از زیربرنامه 7.4 استفاده می‌کند تا مکان گره N را که حاوی عنصر ITEM است و همچنین مکان گره پدر P(N) را پیدا می‌کند. روشی که با آن N از درخت حذف می‌شود در درجه اول بستگی به تعداد بچه‌های N دارد. سه حالت وجود دارد:

حالات ۱: N بچه‌ای ندارد. آنگاه تنها با جایگزین شدن مکان N در گره پدر P(N) به وسیله اشاره گر پوج null گره N از درخت حذف می‌شود.

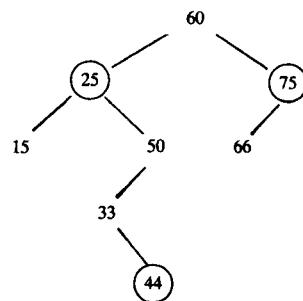
حالات ۲: N دقیقاً یک بچه دارد. آنگاه تنها با جایگزین شدن مکان N در P(N) به وسیله مکان تنها بچه N، گره N از درخت حذف می‌شود.

حالت ۳: N دو بجهه دارد. فرض کنید $S(N)$ نمایش ریشه بعدی پیمایش InOrder، گره N باشد. دانشجو می‌تواند تحقیق کند که $S(N)$ بجهه چپ ندارد. آنگاه نخست با حذف $S(N)$ از T (با استفاده از حالت ۱ یا حالت ۲) و سپس با جانشینی کردن گره $S(N)$ به جای گره N در درخت T ، گره N از T حذف می‌شود. ملاحظه می‌کنید که حالت سوم پیچیده‌تر از دو حالت اول است. در تمام سه حالت بالا، فضای حافظه گره حذف شده N به لیست AVAIL برگردانده می‌شود.

مثال ۷-۱۸

درخت جستجوی دودویی شکل ۷-۲۵ (الف) را در نظر بگیرید. فرض کنید T به صورت شکل ۷-۲۵ (ب) در حافظه قرار می‌گیرد.

	INFO	LEFT	RIGHT
ROOT	1	33	0
AVAIL	2	25	8
3	60	2	7
4	66	0	0
5	6		
6	0		
7	75	4	0
8	15	0	0
9	44	0	0
10	50	1	0



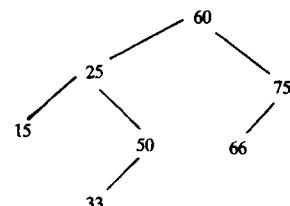
(ب) نمایش پیوندی

(الف) قبل از حذف

شکل ۷-۲۵

(الف) فرض کنید گره ۴۴ از درخت T شکل ۷-۲۵ حذف شده است. توجه دارید که گره ۴۴ بجهه‌ای ندارد. شکل ۷-۲۶ (الف) این درخت را پس از حذف ۴۴ نشان می‌دهد و شکل ۷-۲۶ (ب) نمایش پیوندی آن را در حافظه نشان می‌دهد.

	INFO	LEFT	RIGHT
ROOT	1	33	0
AVAIL	3	25	8
9	2	10	
60	6		
66	0	0	
5			
6	0		
75	4	0	
15	0	0	
9	5		
50	1	0	
10			



(ب) نمایش پیوندی

(الف) گره 44 حذف می شود.

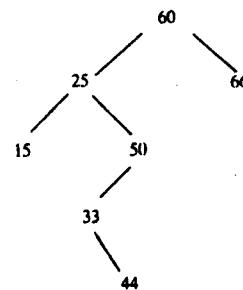
شکل ۷-۲۶

عمل حذف تنها با جایگزینی **NULL** در گره پدر یعنی 33 انجام شده است. مکانهای هاشورخورده بیانگر این تغییرات است.

(ب) فرض کنید به جای گره 44، گره 75 از درخت T شکل ۷-۲۵ حذف می شود. توجه دارید که گره 75 تنها یک بچه دارد. شکل ۷-۲۷ (الف) این درخت را پس از حذف گره 75 نشان می دهد و شکل ۷-۲۷ (ب) نمایش پیوندی آن را نشان می دهد. عمل حذف تنها با تغییر اشاره گر راست گره پدر 60 انجام می شود که در آغاز به 75 اشاره می کرد، درحالی که اکنون به گره 66 اشاره می کند که تنها بچه گره 75 است. مکانهای هاشورخورده بیانگر این تغییرات است.

(ج) فرض کنید به جای گره 44 یا گره 75، گره 25 از درخت T شکل ۷-۲۵ حذف می شود. توجه دارید که گره 25 دو بچه دارد. علاوه بر این ملاحظه می کنید که گره 33 ریشه بعدی پیمایش **InOrder** گره 25 است. شکل ۷-۲۸ (الف) این درخت را پس از حذف 25 نشان می دهد و شکل ۷-۲۸ (ب) نمایش پیوندی آن را نشان می دهد. عمل حذف نخست با حذف 33 از درخت و بدنبال آن با جانشینی گره 33 به جای گره 25 انجام می شود. تأکید می کنیم که جانشینی گره 33 به جای 25، در حافظه تنها با تغییر اشاره گرها انجام می شود نه با جابجایی محتواهای یک گره از یک مکان به مکان دیگر. بدین ترتیب 33 همچنان مقدار INFO[1] است.

	INFO	LEFT	RIGHT	
ROOT	1	33	0	9
AVAIL	2	25	8	10
3	60	2		
4	66	0	0	
5		6		
6		0		
7		15		
8	15	0	0	
9	44	0	0	
10	50	1	0	



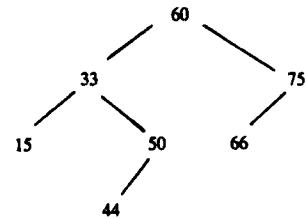
(الف) گره 75 حذف می‌شود.

شکل ۷-۲۷

الگوریتم حذف ما، بر حسب زیربرنامه‌های Procedure 7.6 و Procedure 7.7 به شرح زیر بیان می‌شود. اولین زیربرنامه به حالتهای ۱ و ۲ مربوط می‌شود که در آن گره حذف شده N دو بچه ندارد و زیربرنامه دوم به حالت ۳ مربوط می‌شود که در آن N دو بچه دارد. حالتهای کوچک متعددی وجود دارد که منعکس‌کننده این واقعیت است که N می‌تواند "بچه چپ" بچه راست یا ریشه باشد. همچنین در حالت ۲، N می‌تواند یک بچه چپ یا یک بچه راست داشته باشد.

زیربرنامه 7.7 حالتی را مورد بررسی قرار می‌دهد که گره حذف شده N دو بچه دارد. مذکور می‌شویم که ریشه بعدی پیمایش InOrder را می‌توان با جابجایی بچه راست N بدست آورد و آنگاه به طور مکرر به طرف چپ جابجا می‌کنیم تا گره‌ای با زیردرخت چپ خالی ملاقات شود.

	INFO	LEFT	RIGHT
ROOT	33		10
AVAIL	60		7
3			
2			
4	66	0	0
5		6	
6		0	
7	75	4	0
8	15	0	0
9	44	0	0
10	50		0



(ب) نمايش یوندی

(الف) گرہ 25 حذف می شود۔

٢٨- شکل

Procedure 7.6: CASEA(INFO, LEFT, RIGHT, ROOT, LOC, PAR)

This procedure deletes the node N at location LOC, where N does not have two children. The pointer PAR gives the location of the parent of N, or else PAR = NULL indicates that N is the root node. The pointer CHILD gives the location of the only child of N, or else CHILD = NULL indicates N has no children.

1. [Initializes CHILD.]
If LEFT[LOC] = NULL and RIGHT[LOC] = NULL, then:
 Set CHILD := NULL.
Else if LEFT[LOC] ≠ NULL, then:
 Set CHILD := LEFT[LOC].
Else
 Set CHILD := RIGHT[LOC].
[End of If structure.]
 2. If PAR ≠ NULL, then:
 If LOC = LEFT[PAR], then:
 Set LEFT[PAR] := CHILD.
 Else:
 Set RIGHT[PAR] := CHILD.
 [End of If structure.]
Else:
 Set ROOT := CHILD.
[End of If structure.]
 3. Return.

Procedure 7.7: CASEB(INFO, LEFT, RIGHT, ROOT, LOC, PAR)

This procedure will delete the node N at location LOC, where N has two children. The pointer PAR gives the location of the parent of N, or else PAR = NULL indicates that N is the root node. The pointer SUC gives the location of the inorder successor of N, and PARSUC gives the location of the parent of the inorder successor.

1. [Find SUC and PARSUC.]
 - (a) Set PTR := RIGHT[LOC] and SAVE := LOC.
 - (b) Repeat while LEFT[PTR] ≠ NULL:
 Set SAVE := PTR and PTR := LEFT[PTR].
 [End of loop.]
 - (c) Set SUC := PTR and PARSUC := SAVE.
2. [Delete inorder successor, using Procedure 7.6.]
Call CASEA(INFO, LEFT, RIGHT, ROOT, SUC, PARSUC).
3. [Replace node N by its inorder successor.]
 - (a) If PAR ≠ NULL, then:
 - If LOC = LEFT[PAR], then:
 Set LEFT[PAR] := SUC.
 - Else:
 Set RIGHT[PAR] := SUC.
 [End of If structure.]
 - Else:
 Set ROOT := SUC.
 [End of If structure.]
 - (b) Set LEFT[SUC] := LEFT[LOC] and
RIGHT[SUC] := RIGHT[LOC].
4. Return.

اکنون می‌توانیم با استفاده از زیربرنامه‌های پایه و اصلی 7.6 و 7.7 به عنوان آجرهای ساختمانی و سنگ بنا، الگوریتم حذف را به صورت رسمی بیان کنیم:

Algorithm 7.8: DEL(INFO, LEFT, RIGHT, ROOT, AVAIL, ITEM)

A binary search tree T is in memory, and an ITEM of information is given. This algorithm deletes ITEM from the tree.

1. [Find the locations of ITEM and its parent, using Procedure 7.4.]
Call FIND(INFO, LEFT, RIGHT, ROOT, ITEM, LOC, PAR).
2. [ITEM in tree?]
 - If LOC = NULL, then: Write: ITEM not in tree, and Exit.
 - [Delete node containing ITEM.]
If RIGHT[LOC] ≠ NULL and LEFT[LOC] ≠ NULL, then:
 Call CASEB(INFO, LEFT, RIGHT, ROOT, LOC, PAR).
Else:
 Call CASEA(INFO, LEFT, RIGHT, ROOT, LOC, PAR).
 [End of If structure.]
3. [Return deleted node to the AVAIL list.]
Set LEFT[LOC] := AVAIL and AVAIL := LOC.
4. Exit.

HeapSort ، Heap ۷-۱۰

این بخش ساختمان درخت دیگری را مورد بحث و بررسی قرار می‌دهد که **Heap** نامیده می‌شود. **Heap** در یک الگوریتم مرتب‌کردن جالب و زیبا موسوم به **HeapSort** بکار برده می‌شود. اگرچه روش‌های مرتب‌کردن به طور اساسی در فصل ۹ بررسی می‌شود اما در این بخش الگوریتم **HeapSort** ارائه می‌شود و پیچیدگی آن با الگوریتم مرتب‌کردن حبابی و الگوریتم **QuickSort** که به ترتیب در فصل‌های ۴ و ۶ توضیح داده شدند مقایسه می‌شود.

فرض کنید **H** یک درخت دودویی کامل با n عنصر باشد. فرض می‌کنیم که **H** در حافظه به وسیله آرایه خطی **TREE** و با استفاده از نمایش ترتیبی **H** نگهداری می‌شود نه با نمایش پیوندی، مگر آن که خلاف آن بیان شود آنگاه **H** یک **MaxHeap** یا یک **MinHeap** نامیده می‌شود مشروط بر این که هر گره **N** از **H** دارای خاصیت زیر باشد:

مقدار در **N** بزرگتر یا مساوی با مقدار در هر بچه **N** است. بنابراین مقدار در **N** بزرگتر یا مساوی با مقدار در هر نسل **N** است. یک **MinHeap** به صورت مشابه تعریف می‌شود: مقدار در **N** کوچکتر یا مساوی با مقدار در هر بچه **N** است.

مثال ۷-۱۹

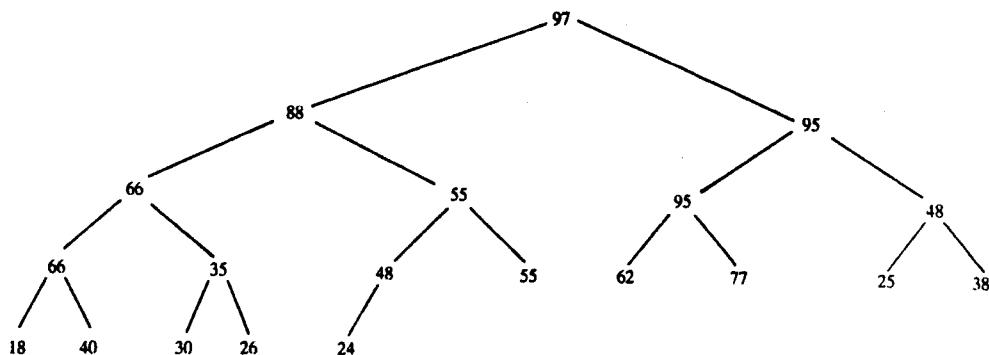
درخت کامل **H** شکل ۷-۲۹ (الف) را در نظر بگیرید. ملاحظه می‌کنید که **H** یک **Heap** است. در حالت خاص، بزرگترین عنصر در **H** در "بالای" **Heap** یعنی در ریشه درخت است. شکل ۷-۲۹ (ب) نمایش ترتیبی **H** را به وسیله آرایه **TREE** نشان می‌دهد. به بیان دیگر، **TREE[1]** ریشه درخت **H** همچنین بچه چپ و راست گره **TREE[K]** به ترتیب **[2K]** و **[2K + 1]** هستند. در حالت خاص پدر هر گره غیر ریشه **J** گره **[J/2]** است که در آن منظور از $2 \div J$ تقسیم صحیح است. ملاحظه می‌کنید که گره‌های هم‌سطح در **H** یکی پس از دیگری در آرایه **TREE** ظاهر می‌شود.

اضافه کردن یک عنصر در **Heap**

فرض کنید **H** یک **Heap** با n عنصر باشد همچنین فرض کنید یک عنصر اطلاعاتی **ITEM** داده شده است. عنصر **ITEM** را به داخل **Heap** به صورت زیر اضافه می‌کنیم:

- (۱) نخست عنصر **ITEM** را به انتهای **H** طوری اضافه می‌کنیم که **H** همچنان یک درخت کامل باشد، اما الزاماً یک **Heap** نیست.
- (۲) آنگاه اجازه دهید **ITEM** به جای مربوطه‌اش در **H** طوری بالا رود که **H** نهایتاً یک **Heap** باشد.

قبل از آن که زیربرنامه **Procedure** را بیان کنیم چگونگی کار این زیربرنامه را توضیح می‌دهیم.



(الف) Heap

TREE

97	88	95	66	55	95	48	66	35	48	55	62	77	25	38	18	40	30	26	24
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

(ب) نمایش ترتیبی Sequential

شکل ۷-۲۹

مثال ۷-۲۰

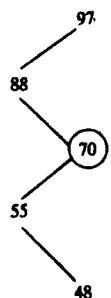
Heap II شکل ۷-۲۹ را در نظر بگیرید. فرض کنید بخواهیم $ITEM = 70$ را به H اضافه کنیم. نخست 70 را به عنوان عنصر بعدی در درخت کامل اضافه می‌کنیم، یعنی قرار می‌دهیم $TREE[21] = 70$ آنگاه $TREE[10] = 70$ پس از 70 راست است. مسیر از 70 تا ریشه H در شکل ۷-۳۰ (الف) به تصویر درآمده است. اکنون مکان مربوط به 70 را در Heap II به شرح زیر پیدا می‌کنیم:

(الف) 70 را با پدرس، 48 مقایسه کنید. از آنجاکه 70 بزرگتر از 48 است، جای 70 و 48 را عوض کنید، مسیر اکنون به شکل ۷-۳۰ (ب) درخواهد آمد.

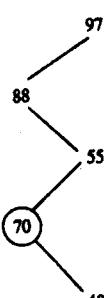
(ب) 70 را با پدر جدیدش، 55 مقایسه کنید از آنجاکه 70 بزرگتر از 55 است، جای 70 و 55 را عوض کنید، مسیر اکنون به شکل ۷-۳۰ (ج) درخواهد آمد.

(ج) 70 را با پدر جدیدش، 88 مقایسه کنید. از آنجاکه 70 بزرگتر از 88 نیست، $ITEM = 70$ به مکان مربوطه اش در H رسیده است.

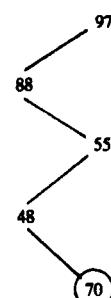
شکل ۷-۳۰ (د) درخت نهایی را نشان می‌دهد. خط‌چین بیانگر آن است که یک جابجایی صورت گرفته است.



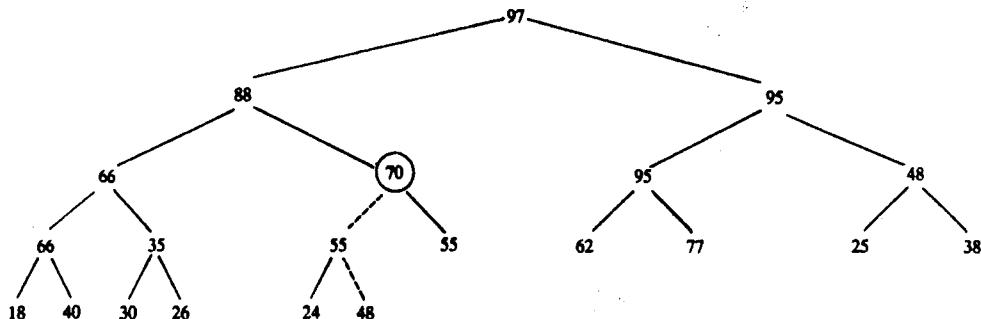
(ج)



(ب)



(الف)



(3)

شكل ٣٠ ITEM = ٧٠ V اضافه مي شود.

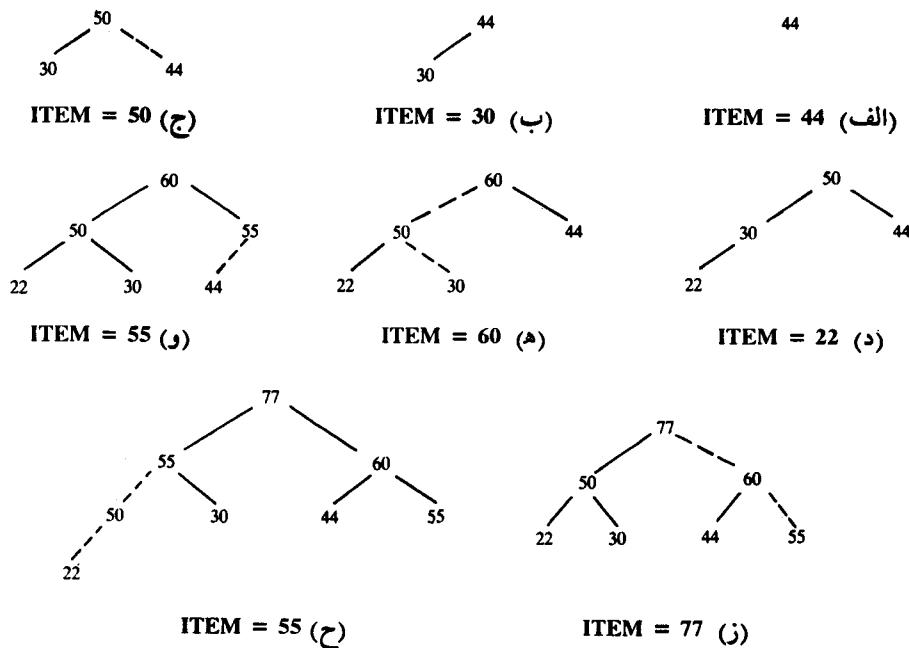
توجه کنید: می‌توان تحقیق کرد که زیربرنامه **Procedure** بالا در نهایت همیشه متوجه به یک درخت **Heap** می‌شود. به عبارت دیگر هیچ چیز دیگری اتفاق نمی‌افتد. این مطلب به سادگی ثابت می‌شود و ما اثبات آن را به عنوان تمرین به دانشجو و اگذار می‌کنیم.

مثال ۷-۲۱

فرض کنید بخواهیم یک Heap H از لیست عددی زیر بسازیم:

این کار را می‌توان با اضافه کردن هشت عدد یکی پس از دیگری در **Heap H** خالی با استفاده از زیر برنامه **Procedure** بالا اضافه کرد. شکل از ۷-۲۱ (الف) تا (ح) تصویرهای مربوطه **Heap** را پس از اضافه شدن هر یک از هشت عنصر نشان می‌دهد. مجدداً خط چین بیانگر آن است که در طی اضافه کردن عنصر

اطلاعاتی داده شده ITEM یک جابجایی صورت گرفته است.



شکل ۷-۳۱. ساختمان یک Heap

بيان رسمي زیربرنامه Procedure برای اضافه کردن یک عنصر اطلاعاتی به صورت زیر است:

Procedure 7.9: INSHEAP(TREE, N, ITEM)

A heap H with N elements is stored in the array TREE, and an ITEM of information is given. This procedure inserts ITEM as a new element of H. PTR gives the location of ITEM as it rises in the tree, and PAR denotes the location of the parent of ITEM.

1. [Add new node to H and initialize PTR.]
Set N := N + 1 and PTR := N.
2. [Find location to insert ITEM.]
Repeat Steps 3 to 6 while PTR < 1.
 3. Set PAR := [PTR/2]. [Location of parent node.]
 4. If ITEM ≤ TREE[PAR], then:
Set TREE[PTR] := ITEM, and Return.
[End of If structure.]
 5. Set TREE[PTR] := TREE[PAR]. [Moves node down.]
 6. Set PTR := PAR. [Updates PTR.]
[End of Step 2 loop.]
7. [Assign ITEM as the root of H.]
Set TREE[1] := ITEM.
8. Return.

ملاحظه می‌کنید که تا وقتی مکان مناسبی برای ITEM پیدا نشود TREE در یک عنصر آرایه TREE[1] حاویگزین می‌شود. مرحله ۷ حالت خاصی را که ITEM تا ریشه [1] بالا می‌رود درنظر می‌گیرد. فرض کنید آرایه A با N عنصر داده شده است. با استفاده مکرر از زیربرنامه ۷ با آرایه A، یعنی با اجرای

Call INSHEAP(A, J, A[J + 1])

به ازای $J = 1, 2, \dots, N - 1$ ، می‌توانیم یک Heap H، از آرایه A بسازیم.

حذف ریشه یک Heap

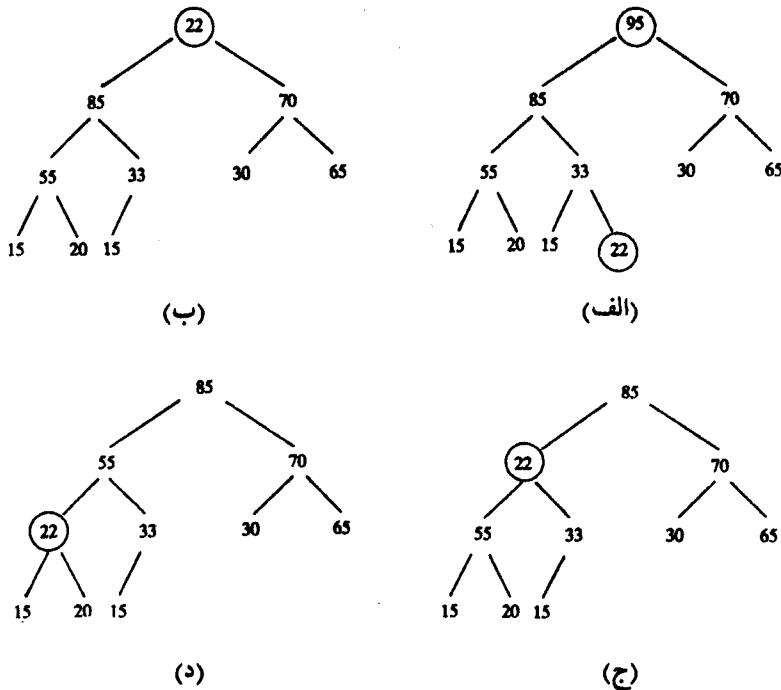
فرض کنید H یک Heap با N عنصر باشد و بخواهیم ریشه R این H را حذف کنیم. این کار به صورت زیر انجام می‌شود:

- (۱) ریشه R را در متغیر ITEM جایگزین کنید.
- (۲) گره آخر L از H را جانشین گره حذف شده R کنید طوری که H همچنان یک درخت کامل باشد، اما الزاماً یک Heap نباشد.
- (۳) (بازسازی Heap) فرض کنید L به مکان مربوطه اش طوری پائین رود که نهایتاً یک Heap باشد.

مثال ۷-۲۲

شکل ۷-۳۲ (الف) را درنظر بگیرید که در آن $95 = R$ ریشه و $22 = L$ آخرین گره درخت است. مرحله ۱ زیربرنامه بالا، $95 = R$ را حذف می‌کند و در مرحله ۲، $22 = L$ جانشین $95 = R$ می‌شود. با این کار درخت کامل شکل ۷-۳۲ (ب) نتیجه می‌شود که Heap نیست. با وجود این ملاحظه می‌کنید که هر زیردرخت چپ و هم زیردرخت راست ۲۲ همچنان Heap هستند. با بکارگیری مرحله ۳، مکان مناسب ۲۲ در Heap به صورت زیر پیدا می‌شود:

- (الف) ۲۲ را با دو بچه اش، ۸۵ و ۷۰ مقایسه کنید. چون ۲۲ کوچکتر از بچه بزرگتر، ۸۵ است جای ۲۲ و ۸۵ را عوض کنید تا درخت به صورت شکل ۷-۳۲ (ج) درآید.
- (ب) ۲۲ را با دو بچه جدیدش، ۵۵ و ۳۳ مقایسه کنید. چون ۲۲ کوچکتر از بچه بزرگتر، ۵۵ است جای ۲۲ و ۵۵ را عوض کنید تا درخت به شکل ۷-۳۲ (د) درآید.
- (ج) ۲۲ را با بچه جدیدش ۱۵ و ۲۰ مقایسه کنید. چون ۲۲ بزرگتر از هر دو بچه است. گره ۲۲ به مکان مربوطه اش در H افتاده است.



شکل ۷-۳۲

بدین ترتیب، شکل ۷-۳۲ (د) **Heap H** خواسته شده بدون ریشه اولیه R می‌باشد.
 توجه کنید: همانگونه که در قسمت اضافه کردن یک عنصر در **Heap** بیان شد، می‌توان تحقیق کرد که زیربرنامه **Procedure** بالا درنهایت همیشه متنه به یک درخت **Heap** می‌شود. بار دیگر اثبات آن را به دانشجو واگذار می‌کنیم. علاوه بر این مذکور می‌شویم که ممکن است تا وقتی گره L به پائین درخت نرسیده است یا تا وقتی L بچهای ندارد مرحله ۳ زیربرنامه به پایان نرسد.
 بیان رسمی این زیربرنامه به صورت زیر است:

Procedure 7.10: DELHEAP(TREE, N, ITEM)

A heap H with N elements is stored in the array TREE. This procedure assigns the root TREE[1] of H to the variable ITEM and then reheaps the remaining elements. The variable LAST saves the value of the original last node of H. The pointers PTR, LEFT and RIGHT give the locations of LAST and its left and right children as LAST sinks in the tree.

1. Set ITEM := TREE[1]. [Removes root of H.]
2. Set LAST := TREE[N] and N := N - 1. [Removes last node of H.]
3. Set PTR := 1, LEFT := 2 and RIGHT := 3. [Initializes pointers.]
4. Repeat Steps 5 to 7 while RIGHT ≤ N:
5. If LAST ≥ TREE[LEFT] and LAST ≥ TREE[RIGHT], then:
 Set TREE[PTR] := LAST and Return.
 [End of If structure.]
6. IF TREE[RIGHT] ≤ TREE[LEFT], then:
 Set TREE[PTR] := TREE[LEFT] and PTR := LEFT.
 Else:
 Set TREE[PTR] := TREE[RIGHT] and PTR := RIGHT.
 [End of If structure.]
7. Set LEFT := 2 * PTR and RIGHT := LEFT + 1.
 [End of Step 4 loop.]
8. If LEFT = N and if LAST < TREE[LEFT], then: Set PTR := LEFT.
9. Set TREE[PTR] := LAST.
10. Return.

تا وقتی که LAST یک بچه راست دارد حلقه مرحله 4 تکرار می شود. مرحله 8 حالت خاصی را در نظر می گیرد که در آن LAST یک بچه راست ندارد اما بچه چپ دارد (که باید آخرین گره H باشد). دلیل استفاده از دو دستور IF در مرحله 8 آن است که تا وقتی $TREE[LEFT] > N$ ممکن است تعریف شود.

كاربرد Heap در مرتب کردن اطلاعات

فرض کنید آرایه A با N عنصر داده شده است. الگوریتم HeapSort که A را مرتب می کند از دو مرحله

زیر تشکیل می شود :

مرحله A : یک H از عناصر آرایه A بسازید.

مرحله B : عنصر ریشه H را بطور مکرر حذف کنید.

از آنجاکه ریشه H همواره بزرگترین گره H است، مرحله B، عناصرهای آرایه A را به ترتیب نزولی حذف می کند. بیان رسمی این الگوریتم که از زیر برنامه های 7.9 و 7.10 استفاده می کند به شرح زیر است :

Algorithm 7.11: HEAPSORT(A, N)

An array A with N elements is given. This algorithm sorts the elements of A.

1. [Build a heap H, using Procedure 7.9.]
Repeat for J = 1 to N - 1:
 Call INSHEAP(A, J, A[J + 1]).
[End of loop.]
2. [Sort A by repeatedly deleting the root of H, using Procedure 7.10.]
Repeat while N > 1:
 (a) Call DELHEAP(A, N, ITEM).
 (b) Set A[N + 1] := ITEM.
[End of Loop.]
3. Exit.

هدف مرحله (b) ذخیره فضای حافظه است. به بیان دیگر، می‌توان از یک آرایه دیگر B برای نگهداری عنصرهای مرتب شده A استفاده کرد و به جای مرحله (b) چنین نوشت:

Set B[N + 1] := ITEM

بنابراین، دانشجو می‌تواند تحقیق کند که مرحله (b) داده شده، در الگوریتم دخالت نمی‌کند چون Heap H به A[N+1] تعلق ندارد.

پیچیدگی HeapSort

فرض کنید الگوریتم HeapSort بر آرایه A با n عنصر اعمال شده است. این الگوریتم دو مرحله دارد همچنین پیچیدگی هر مرحله بطور مستقل تجزیه و تحلیل می‌شود.

مرحله A: فرض کنید H یک Heap ملاحظه می‌کنید که تعداد مقایسه‌های موردنبیاز برای پیدا کردن مکان مربوط به عنصر جدید ITEM در H نمی‌تواند بزرگتر از عمق H باشد. از آنجا که H یک درخت کامل است عمق آن به $\log_2 m$ محدود می‌شود که در آن m تعداد عناصر H است. بنابراین، تعداد کل مقایسه‌ها برای اضافه کردن n عنصر A در H به عبارت زیر محدود می‌شود:

$$g(n) \leq n \log_2 n$$

درنتیجه، زمان اجرای مرحله A در Heapsort با $n \log_2 n$ متناسب است.

مرحله B: فرض کنید H یک درخت کامل با m عنصر باشد همچنین فرض کنید زیردرختهای چپ و راست H خود Heap باشند و L ریشه H باشد. ملاحظه می‌کنید Heap سازی مجدد از 4 مقایسه استفاده می‌کند تا گره L یک مرحله به طرف پائین درخت H منتقل شود. از آنجا که عمق H بزرگتر از $\log_2 m$ نیست، از این رو Heap سازی مجدد حداقل از $4 \log_2 m$ مقایسه برای پیدا کردن مکان مناسب L در درخت H استفاده می‌کند. به این معنی که تعداد کل $h(n)$ مقایسه‌ها برای حذف n عنصر A از H که نیازمند n بار

سازی Heap مجدد است به صورت زیر محدود می‌شود:

$$h(n) \leq 4n \log_2 n$$

بنابراین، زمان اجرای مرحله B در Heapsort نیز متناسب با $n \log_2 n$ است.

از آنجاکه هر مرحله از نظر زمانی متناسب با $n \log_2 n$ است، زمان اجرای مرتب کردن n عنصر آرایه A با استفاده از Heapsort متناسب با n است یعنی $f(n) = O(n \log_2 n)$. ملاحظه می‌کنید که این رابطه پیچیدگی بدترین حالت الگوریتم HeapSort را به دست می‌دهد. این الگوریتم در مقابل دو الگوریتم مرتب کردن زیر قرار دارد که قبلًا مورد مطالعه قرار گرفت:

(۱) مرتب کردن حبابی (بخش ۶-۴). زمان اجرای مرتب کردن حبابی $O(n^2)$ است.

(۲) QuickSort (بخش ۶-۵)، زمان اجرای میانگین QuickSort برابر $O(n \ log_2 n)$ است، که برابر زمان اجرای HeapSort است، اما زمان اجرای بدترین حالت QuickSort برابر $O(n^2)$ است که برابر زمان اجرای مرتب کردن حبابی است.

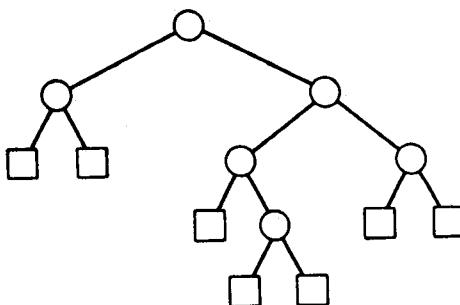
دیگر الگوریتمهای مرتب کردن در فصل ۹ مورد مطالعه قرار می‌گیرد.

۱۱-۷ طول مسیر، الگوریتم هافمن

یادآوری می‌کنیم که یک درخت دودویی گسترش یافته یا ۲-درخت، یک درخت دودویی T است که در آن هر گره ۰ یا ۲ بچه دارند. گرهایی که ۰ بچه دارند گرهای خارجی نام دارند. همچنین گرهایی که ۲ بچه دارند گرهای داخلی نام دارند، شکل ۷-۳۳. ۲-درخت را نشان می‌دهد که در آن گرهای داخلی با دایره و گرهای خارجی با مربع نشان داده است در هر ۲-درخت، تعداد N_E گرهای خارجی ۱ واحد بیشتر از N_I گرهای داخلی است. یعنی:

$$N_E = N_I + 1$$

برای مثال، برای ۲-درخت شکل ۷-۳۳ داریم $N_E = 6$ و $N_I = 5$.



شکل ۷-۳۳

اغلب یک الگوریتم را می‌توان با ۲- درخت T نمایش داد که در آن گره‌های داخلی آزمایشها و گره‌های خارجی، عمل‌ها Actions را نمایش می‌دهند. بنابراین، زمان اجرای الگوریتم می‌تواند به طول مسیرهای داخل درخت بستگی داشته باشد. با توجه به این مطلب، طول مسیر خارجی L_E یک ۲- درخت T به صورت مجموع طولهای تمام مسیرهایی تعریف می‌شود که حاصل جمع طول تمام مسیرها از ریشه R درخت T تا یک گره خارجی است. طول مسیر داخلی L_I درخت T بطور مشابه تعریف می‌شود که در آن به جای گره‌های خارجی، از گره‌های داخلی استفاده می‌شود. برای درخت شکل

۷-۳۳ داریم :

$$L_I = 0 + 1 + 1 + 2 + 3 + 2 = 9 \quad L_E = 2 + 2 + 3 + 4 + 4 + 3 + 3 = 21$$

ملاحظه می‌کنید که

$$L_I + 2n = 9 + 2.6 = 9 + 12 = L_E$$

که در آن $n = 6$ تعداد گره‌های داخلی است درواقع، فرمول

$$L_E = L_I + 2n$$

برای هر ۲- درخت با n گره داخلی برقرار است.

فرض کنید T یک ۲- درخت با n گره خارجی است و همچنین فرض کنید به هر گره خارجی یک وزن (غیرمنفی) نسبت داده شده است. طول مسیر وزن داده شده (خارجی) درخت T بنایه تعریف مجموع طولهای مسیر وزن داده شده است یعنی :

$$P = W_1 L_1 + W_2 L_2 + \dots + W_n L_n$$

که در آن W_i و L_i به ترتیب وزن و طول مسیر یک گره خارجی i است.

حال مجموعه تمام ۲- درختهایی را درنظر بگیرید که n گره خارجی دارند. واضح است که درخت کامل در میان آنها طول مسیر خارجی حداقل (Minimal) L_E دارد. از طرف دیگر فرض کنید به هر درخت همان n وزن مربوط به گره‌های خارجی اشن داده شده است. آنگاه روشن نیست که درخت یک طول مسیر وزن داده شده حداقل P را به دست می‌دهد یا خیر؟

مثال ۷-۲۳

شکل ۷-۳۴ سه ۲- درخت T_1 , T_2 و T_3 را نشان می‌دهد که هر یک از این درختها گره‌های خارجی با

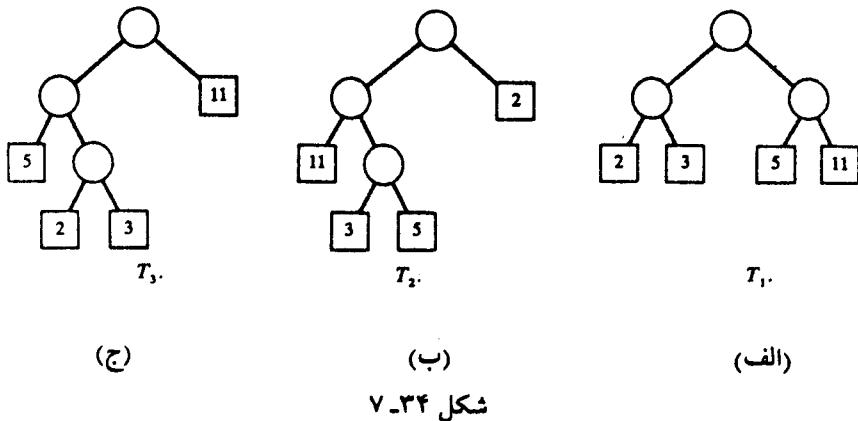
وزن ۲، ۳، ۵ و ۱۱ دارند. طولهای مسیر وزن داده شده سه درخت به قرار زیرند :

$$P_1 = 2.2 + 3.2 + 5.2 + 11.2 = 42$$

$$P_2 = 2.1 + 3.3 + 5.3 + 11.2 = 48$$

$$P_3 = 2.3 + 3.3 + 5.2 + 11.1 = 36$$

کمیتهای P_1 و P_3 بیان می‌کنند که درخت کامل لزومی ندارد یک طول حداقل P را به دست دهد و کمیتهای P_2 و P_3 بیان می‌کنند که درختهای مشابه لزومی ندارند همان طول‌ها را به دست دهنند.



مسائله‌ای که ما در حالت کلی مایل به حل آن هستیم به شرح زیر است. فرض کنید یک لیست با « وزن، داده شده است :

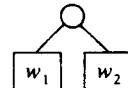
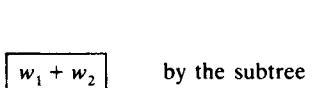
$$w_1, w_2, \dots, w_n$$

در میان تمام ۲ - درختهای دارای «گره خارجی و «وزن داده شده، مطلوب است تعیین یک درخت T با حداقل طول مسیر وزن داده شده. بندرت اتفاق می‌افتد چنین درخت T ای یکتا و منحصر بفرد باشد. هافمن الگوریتمی برای پیدا کردن چنین درخت T ای ارائه داد که ما اکنون آن را بیان می‌کنیم. ملاحظه می‌کنید که الگوریتم هافمن به صورت بازگشتی بر حسب تعداد وزنهای و این که جواب برای یک وزن منحصر به یک درخت با یک گره است، تعریف می‌شود. از طرف دیگر در عمل، ما از یک الگوریتم با حالت تکرار و غیر بازگشتی معادل الگوریتم هافمن برای ساختن درخت از پائین به بالا (Bottom – Up) به عوض ساخته شدن درخت از بالا به پائین (Top – Down) استفاده می‌کنیم.

Huffman's Algorithm: Suppose w_1 and w_2 are two minimum weights among the n given weights w_1, w_2, \dots, w_n . Find a tree T' which gives a solution for the $n - 1$ weights

$$w_1 + w_2, w_3, w_4, \dots, w_n$$

Then, in the tree T' , replace the external node



The new 2-tree T is the desired solution.

مثال ۷-۲۴

فرض کنید A, B, C, D, E, F, G و H ، ۸ عنصر اطلاعاتی هستند همچنین فرض کنید به این عناصر وزنهای زیر نسبت داده شده است :

عنصر اطلاعاتی	A	B	C	D	E	F	G	H
وزن	22	5	11	19	2	11	25	5

شکل ۷-۳۵ (الف) تا (ح) چگونگی ساخته شدن درخت T را با حداقل طول مسیر وزن داده شده با استفاده از اطلاعات بالا و الگوریتم هافمن می‌دهد. ما هر مرحله را بطور مستقل توضیح می‌دهیم:
 (الف) در اینجا هر عنصر اطلاعاتی به زیردرخت خودش تعلق دارد. دو زیردرخت با کوچکترین ترکیب وزنی ممکن، یکی به وزن 2 و دیگری به وزن 5 سایه‌دار شده است.

(ب) در اینجا زیردرخت‌هایی که در شکل ۷-۳۵ (الف) سایه‌دار شده‌اند به هم متصل می‌شوند تا تشکیل یک زیردرخت به وزن 7 را بدهند. مجدداً، دو زیردرخت جاری با کمترین وزن سایه‌دار شده‌اند.
 (ج) تا (ز) هر مرحله دو زیردرخت را به هم متصل می‌کند که کوچکترین وزن موجود را دارد (همیشه زیردرخت‌هایی هستند که در نمودار قبلی سایه‌دار شده‌اند) و مجدداً دو زیردرخت حاصل با کمترین وزن، سایه‌دار شده‌اند.

(ح) درخت نهایی مطلوب T وقتی ساخته می‌شود که دو زیردرخت با قیمانده به هم متصل شده باشد.

پیاده‌سازی الگوریتم هافمن در کامپیوتر

مجدداً اطلاعات مثال ۷-۲۴ را درنظر بگیرید. فرض کنید خواسته باشیم الگوریتم هافمن را با استفاده از کامپیوتر پیاده‌سازی کنیم. قبل از هرچیز، نیازمند یک آرایه اضافی WT برای نگهداری وزن

گره‌ها هستیم یعنی درخت ما توسط چهار آرایه موازی **INFO**، **WT**، **LEFT** و **RIGHT** نگهداری می‌شود. شکل ۷-۳۶ (الف) چگونگی ذخیره‌شدن اطلاعات داده شده را در آغاز در کامپیوتر نشان می‌دهد. ملاحظه می‌کنید که جای کافی برای گره‌های اضافی وجود دارد. دیده می‌شود که برای گره‌های اولیه اشاره‌گرهای چپ و راست **N**ULL ظاهر شده است زیرا این گره‌ها در درخت نهایی گره‌های انتهایی هستند.

در طی اجرای الگوریتم باید بتوانیم تمام زیردرختهای مختلف را نگهداریم، علاوه بر این باید قادر باشیم زیردرختهایی که حداقل وزن را دارند پیدا کنیم. این کار با نگهداری یک **MinHeap** کمکی انجام می‌شود که هر گره آن شامل وزن و مکان ریشه زیردرخت جاری است. **MinHeap** اولیه در شکل ۷-۳۶ (ب) نشان داده شده است. از آنجا که می‌خواهیم گره با کمترین وزن در بالای **Heap** قرار گیرد به جای **MinHeap** از **MaxHeap** مستفاده شده است.

مرحله اول در ساختن **T** درخت هافمن موردنظر شامل جزء مراحل زیر است :

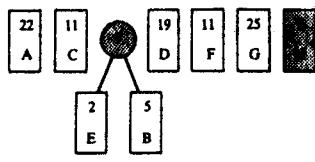
(i) گره **5** و گره **2** = **N₁** را از **Heap** حذف کنید. هر باری که یک گره حذف می‌شود، باید بازسازی **Heap** شود.

(ii) از **N₁** و **N₂** و اولین فضای موجود **9** = **AVAIL** برای اضافه کردن گره جدید به صورت زیر استفاده کنید:

$$\text{WT}[9] = 2 + 5 = 7 \quad \text{LEFT}[9] = 5 \quad \text{RIGHT}[9] = 2$$

بدین ترتیب **N₁** بجهه چپ گره جدید و **N₂** بجهه راست گره جدید است.

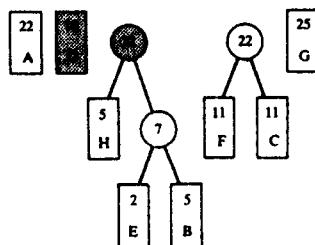
(iii) وزن و مکان گره جدید یعنی **[9, 7]** را به **Heap** اضافه کنید.



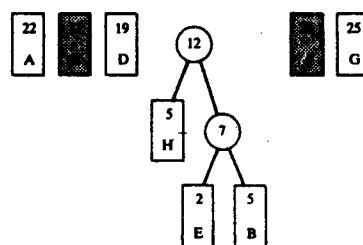
(ب)



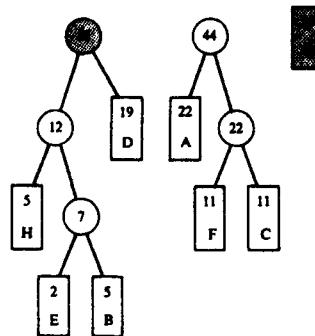
(الف)



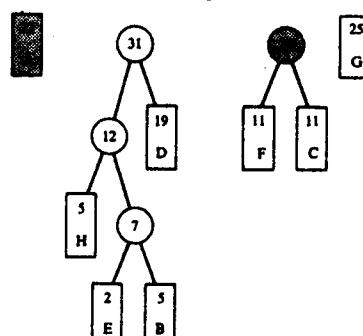
(ج)



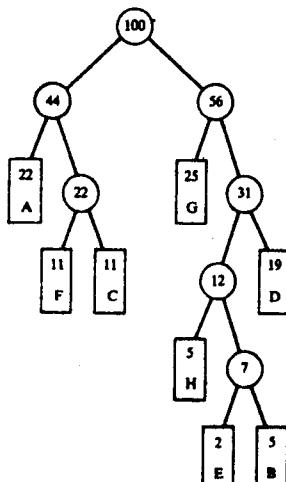
(ز)



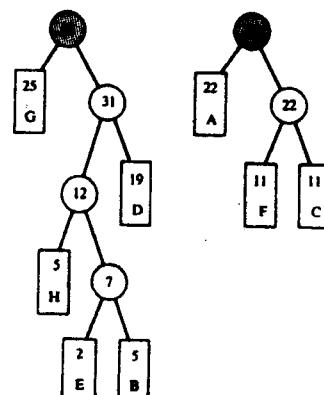
(س)



(د)



(هـ)



(حـ)

	INFO	WT	LEFT	RIGHT
1	A	22	0	0
2	B	5	0	0
3	C	11	0	0
4	D	19	0	0
5	E	2	0	X
6	F	11	0	0
7	G	25	0	0
8	H	5	0	Δ Δ Δ Δ *
9				
10		12	8	9
11		22	6	3
12		31	10	4
13		44	1	11
14		56	7	12
15		100	13	14
16				

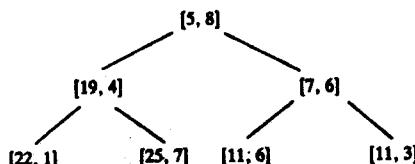
ROOT = 15, AVAIL = 16

	INFO	WT	LEFT	RIGHT
1	A	22	0	0
2	B	5	0	0
3	C	11	0	0
4	D	19	0	0
5	E	2	0	0
6	F	11	0	0
7	G	25	0	0
8	H	5	0	0
9				10
10				11
11				12
12				13
13				14
14				15
15			Q	16
16			Q	0

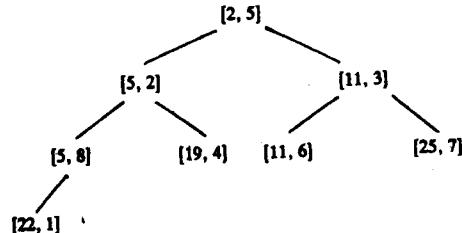
AVAIL = 9

(ج)

(الف)



(د)



(ب)

شكل ٣٦ - پیاده‌سازی الگوریتم هافمن

منطقه سایه زده شده در شکل ۷-۳۶ (ج) گره‌های جدید را نشان می‌دهد همچنین شکل ۷-۳۶ (د) جدید را نشان می‌دهد که یک عنصر کمتر از Heap شکل ۷-۳۶ (ب) دارد.

مرحله بالا را آنچه ادامه دهد که Heap خالی شود. بدین ترتیب درخت موردنظر در شکل ۷-۳۶ (ج) به دست می‌آید. باید قرار دهیم $\text{ROOT} = 15$ ، چون این مکان آخرین گره اضافه شده به درخت است.

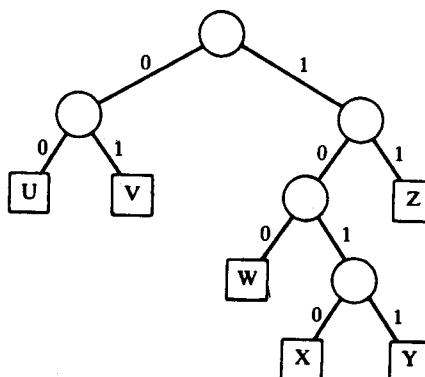
کاربرد الگوریتم هافمن در کدگذاری

فرض کنید خواسته باشیم مجموعه‌ای از n عنصر اطلاعاتی از A_1, A_2, \dots, A_N به وسیله رشته‌هایی از بیت‌ها به صورت کد درآوریم. یک راه برای این منظور آن است که هر عنصر اطلاعاتی را به وسیله یک رشته r بیتی کدگذاری کنیم که در آن

$$2^{r-1} < n \leq 2^r$$

برای مثال، یک مجموعه با ۴۸ کاراکتر را اغلب با استفاده از رشته‌های ۶ بیتی در حافظه کدگذاری می‌کنند. برای این مجموعه نمی‌توان از رشته‌های ۵ بیتی استفاده کرد چون $2^5 < 48 \leq 2^6$.

فرض کنید عنصرهای اطلاعاتی با احتمال مساوی اتفاق نمی‌افتد. آنگاه فضای حافظه را می‌توان با استفاده از رشته‌های با طول متغیر حفظ کرد که در آن عناصری که اغلب اوقات ظاهر می‌شوند در رشته‌های کوتاه‌تر جایگزین می‌شوند و عناصری که بسیار نمایند در رشته‌های بزرگتر جایگزین می‌شوند. این بخش با استفاده از رشته‌هایی با طول متغیر یک روش کدگذاری را توضیح می‌دهد که بر پایه T درخت هافمن برای عناصر اطلاعاتی وزن داده شده استوار است.



شکل ۷-۳۷

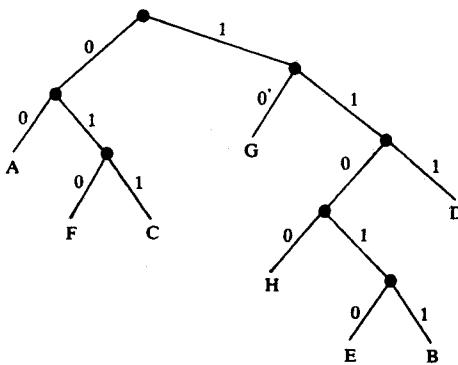
T درخت دودویی گسترش یافته شکل ۷-۳۷ را درنظر بگیریم که گره‌های خارجی آن عناصر U, V, Y, X, W, Z هستند. ملاحظه می‌کنید که هر یال از گره داخلی تا بجهة چپ با بیت ۰ و هر یال از گره داخلی به بجهة راست با بیت ۱ شماره‌گذاری می‌شود. کد هافمن در هر گره خارجی دنباله بیتها از ریشه تا آن گره را جایگزین می‌کند. بدین ترتیب درخت T در شکل ۷-۳۷ گُدد زیر را برای گره‌های خارجی معین می‌کند:

U: 00 V: 01 W: 100 X: 1010 Y: 1011 Z: 11

این کد خاصیت "پیشوندی" دارد یعنی کد هر عنصر اطلاعاتی زیررشتة اولیه کد هر عنصر اطلاعاتی دیگر نیست. به بیان دیگر هیچ ابهامی نمی‌تواند در هنگام از کد درآوردن هر پیغام با استفاده از کد هافمن وجود داشته باشد.

بار دیگر 8 عنصر اطلاعاتی A, B, C, D, E, F, G, H را درنظر بگیرید. فرض کنید وزنها نمایش درصد احتمالاتی باشد که عنصرها ظاهر می‌شوند. آنگاه درخت T با حداقل طول مسیر وزن داده شده ساخته شده در شکل ۷-۳۸ که با شماره‌گذاری‌های بیتی در شکل ۷-۳۸ نشان داده است، یک کدگذاری مؤثر و کارا از عناصر اطلاعاتی را نتیجه می‌دهد. دانشجو می‌تواند تحقیق کند که درخت T کد زیر را بدست می‌دهد:

A: 00	B: 11011	C: 011	D: 111
E: 11010	F: 010	G: 10	H: 1100



شکل ۷-۳۸

۱۲ - درختهای عمومی

یک درخت عمومی (که گاهی اوقات یک درخت نامیده می‌شود) بسته تعریف یک مجموعه متناهی و غیرتنهی T از عناصر موسوم به گره‌ها است، به طوری که:

(۱) یک عنصر متمایز R موسوم به ریشه R داشته باشد.

(۲) عناصرهای باقیمانده T تشکیل یک مجموعه مرتب از صفر یا چند درخت جدا از هم T_1, T_2, \dots, T_m را بدیند.

درختهای T_1, T_2, \dots, T_m زیردرختهای ریشه R نامیده می‌شوند و ریشه‌های T_1, T_2, \dots, T_m گره‌های بعدی R نامیده می‌شوند.

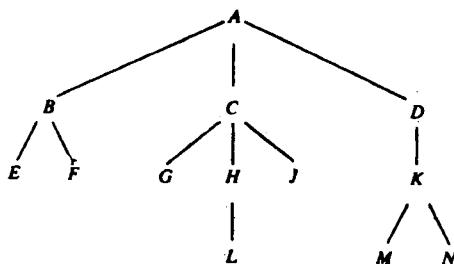
اصطلاحات مربوط به رابطه‌های خانوادگی، نظریه گراف و علم باگبانی به همان صورتی که در درختهای دودویی بکار رفت برای درختهای عمومی نیز مورد استفاده قرار می‌گیرد. بهویژه، اگر یک گره N با گره‌های بعدی S_1, S_2, \dots, S_m باشد آنگاه N پدر S_1 ‌ها و S_2 ‌ها بچه‌های N و S_3 ‌ها همردیف یا برادرهای یکدیگر نامیده می‌شوند.

اصطلاح "درخت" با اندکی تغییر در معنی، در بسیاری از شاخه‌های مختلف ریاضی و علم کامپیوتر ظاهر می‌شود. در اینجا فرض می‌کنیم که T درخت عمومی، ریشه‌دار است یعنی T یک گره متمایز R موسوم به ریشه T دارد و T مرتب است یعنی بچه‌های هر گره N درخت T یک ترتیب مشخص دارد. این دو خاصیت همیشه برای تعریف یک درخت لازم نیستند.

مثال ۷-۲۵

شکل ۷-۳۹ یک درخت عمومی T را با ۳ گرمیشان می‌دهد،

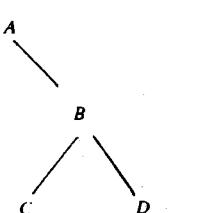
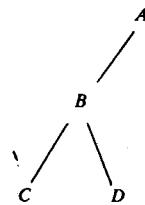
$A, B, C, D, E, F, G, H, J, K, L, M, N$



شکل ۷-۳۹

ریشه درخت T گره بالای نمودار است و بچه‌های یک گره از چپ به راست مرتب هستند، مگر آن که خلاف آن بیان شود. بنابراین، A ریشه T و B سه بچه دارد، بچه اول B ، بچه دوم C و بچه سوم D است. ملاحظه می‌کنید که:

- (الف) گره C سه بچه دارد.
- (ب) هر یک از گره‌های B و K دو بچه دارند.
- (ج) هر یک از گره‌های D و H تنها یک بچه دارند.
- (د) گره‌های E، F، G، L، J، M و N هیچ بچه‌ای ندارند.
- به دسته گره‌های آخر که هیچ بچه‌ای ندارند گره‌های انتهایی می‌گویند.
- یک درخت دودویی^۱ حالت خاصی از درخت عمومی T نیست: درختهای دودویی و درختهای عمومی به دو موضوع مختلف تعلق دارند. دو اختلاف اساسی آنها عبارت است از:
- (۱) یک درخت دودویی T می‌تواند خالی باشد اما یک درخت عمومی نمی‌تواند خالی باشد.
 - (۲) فرض کنید گره N تنها یک بچه دارد. آنگاه این بچه در یک درخت دودویی T با عنوان بچه چپ و بچه راست از هم متمایز می‌شوند، اما در یک درخت عمومی T هیچ‌گونه تمایزی بین آنها وجود ندارد. اختلاف دوم به وسیله درختهای T₁ و T₂ در شکل ۷-۴۰ توضیح داده می‌شود. به طور مشخص، به عنوان درختهای دودویی، T₁ و T₂ درختهای متمایز هستند چون B بچه چپ A در درخت T₁ است اما در درخت T₂ بچه راست A است. از طرف دیگر، به عنوان درختهای عمومی هیچ‌گونه تفاوتی بین درختهای T₂ و T₁ وجود ندارد.

Tree T₂Tree t₁

شکل ۷-۴۰

یک جنگل بنابراین تعریف مجموعه‌ای مرتب از صفر یا چند درخت متمایز است. بهوضوح، اگر ریشه R از درخت عمومی T حذف شود، آنگاه جنگل F به دست می‌آید که از زیردرختهای R (که می‌تواند خالی باشد) تشکیل می‌شود. بالعکس، اگر F یک جنگل باشد، آنگاه می‌توان گره R را به F اضافه کرد تا تشکیل یک درخت عمومی T را بدهد که در آن R ریشه T و زیردرختهای R مشتمل از درختهای اصلی در F است.

نمایش درختهای عمومی در کامپیوتر

فرض کنید T یک درخت عمومی باشد، T در حافظه بهوسیله یک نمایش پیوندی نگهداری می‌شود که از سه آرایه موازی **INFO**، **CHILD** و **SIBL** (یا **DOWN**) و (یا **HORZ**) و یک متغیر اشاره‌گر **ROOT** به شرح زیر استفاده می‌کند مگر آن که خلاف آن به صورت صریح یا ضمنی بیان گردد: قبل از همه، هر گره N از درخت T متناظر با یک مکان K است به طوری که:

- (۱) **INFO[K]** شامل اطلاعات گره N است.

- (۲) **CHILD[K]** شامل مکان بچه اول N است. شرط $\text{CHILD}[K] = \text{NULL}$ بیان می‌کند که N بچه‌ای ندارد.

- (۳) **SIBL[K]** شامل مکان همردیف یا برادر بعدی گره N است. شرط $\text{SIBL}[K] = \text{NULL}$ بیان می‌کند که N بچه آخر پدرش است.

علاوه براین، **ROOT** شامل مکان ریشه R درخت T است. اگرچه این نمایش می‌تواند مصنوعی به نظر رسد، اما دارای این مزیت مهم است که هر گره N از درخت T ، بدون درنظر گرفتن تعداد بچه‌های N ، دقیقاً شامل سه فیلد خواهد بود.

نمایش بالا را می‌توان به سادگی برای نمایش یک جنگل مشکل از درختهای T_{m,\dots,T_2,T_1} تعمیم داد مشروط بر این که ریشه این درختها همردیف یا برادر باشند. درچنین حالی، **ROOT** شامل مکان ریشه R_1 از درخت اول T_1 است، یا وقتی F خالی است **ROOT** برابر **NULL** است.

مثال ۷-۲۶

T درخت عمومی شکل ۷-۳۹ را درنظر بگیرید. فرض کنید همانند شکل ۷-۴۱ (الف) اطلاعات گره‌های T در آرایه **INFO** ذخیره شده‌اند. رابطه‌های ساختاری درخت T با جایگزینی مقادیر در اشاره‌گر **ROOT** و آرایه **CHILD** و **SIBL** به صورت زیر بدست می‌آید:

(الف) چون ریشه A از درخت T در **INFO[2]** ذخیره می‌شود قرار دهید $2 := \text{ROOT}$.

(ب) چون بچه اول گره B است که در **INFO[3]** ذخیره می‌شود قرار دهید $3 := \text{CHIL[2]}$. چون گره A همردیفی ندارد، قرار دهید $\text{SIBL[2]} := \text{NULL}$.

(ج) چون بچه اول گره E است، که در **INFO[15]** ذخیره می‌شود قرار دهید $15 := \text{CHILD[3]}$. چون گره C همردیف بعدی B است و C در **INFO[4]** ذخیره می‌شود قرار دهید $4 := \text{SIBL[3]}$.

والی آخر: شکل ۷-۴۱ (ب) مقادیر نهایی **CHILD** و **SIBL** را بدست می‌دهد. ملاحظه می‌کنید که لیست گره‌های خالی **AVAIL** در آرایه اول **CHILD** نگهداری می‌شود که در آن $1 = \text{AVAIL}$.

	INFO	CHILD	SIBL
1		5	
2	A	3	0
3	B	15	4
4	C	6	16
5		13	
6	G	0	7
7	H	11	8
8	J	0	0
9	N	0	0
10	M	0	9
11	L	0	0
12	K	10	0
13		0	
14	F	0	0
15	E	0	14
16	D	12	0

ROOT = 2, AVAIL = 13

(ب)

(الف)

شکل ۷-۴۱

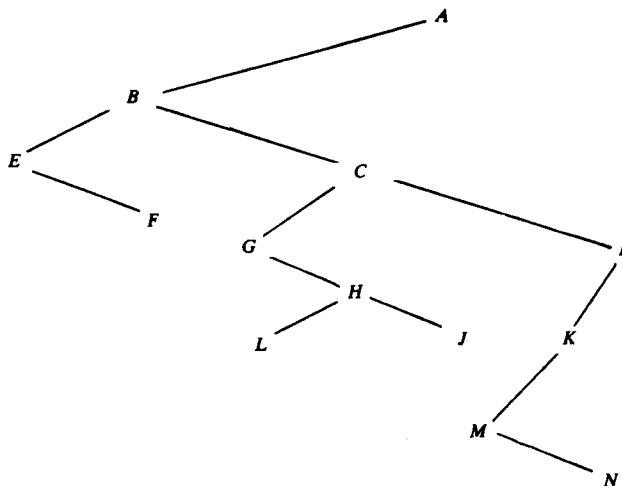
تناوله بین درختهای عمومی و درختهای دودویی

فرض کنید T' یک درخت عمومی باشد. آنگاه می‌توان یک درخت دودویی T' منحصر بفرد و یکتا به شرح زیر به T' نسبت داد. قبل از هرچیز، باید گفت که گره‌های درخت دودویی T' همان گره‌های درخت عمومی T هستند و ریشه T' ریشه T خواهد بود. فرض کنید N یک گره دلخواه از درخت دودویی T' باشد. آنگاه بچه چپ N در T' بچه اول گره N در درخت عمومی T خواهد بود و بچه راست N در T' هم‌دیف بعدی N در درخت عمومی خواهد بود.

مثال ۷-۲۸

درخت عمومی شکل ۷-۳۹ را درنظر بگیرید. دانشجو می‌تواند تحقیق کند که T' درخت دودویی

شکل ۴۲-۷ متناظر با T' درخت عمومی است. ملاحظه می‌کنید که با دوران تصویر T' شکل ۴۲-۷ در جهت مثلثاتی تا هنگامی که بالهای مربوط به بجهة راست موازی گردند تصویری به دست می‌آید که در آن گره‌ها، همان مکان نسبی گره‌های شکل ۴۲-۷ را اشغال می‌کنند.



شکل ۴۲-۷ درخت دودویی T'

نمایش درخت عمومی T در کامپیوتر و نمایش پیوندی T' درخت دودویی متناظر با آن دقیقاً یکی هستند بجز اینکه نامهای آرایه‌های SIBL و CHILD برای درخت عمومی T متناظر با نامهای آرایه‌های LEFT و RIGHT برای درخت دودویی T' هستند. اهمیت این تناظر در آن است که الگوریتم‌های معینی که بر درختهای دودویی اعمال می‌شوند نظیر الگوریتم‌های پیمایش را اکنون می‌توان بر درختهای عمومی اعمال کرد.

مسائلهای حل شده

درختهای دودویی

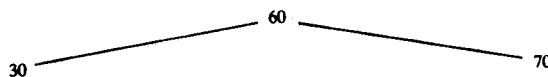
مسئله ۷-۷: فرض کنید درخت دودویی T به صورت شکل ۴۳-۷ در حافظه ذخیره شده است. نمودار درخت T را رسم کنید.

حل: درخت T از ریشه‌اش R به طرف پائین به صورت زیر رسم می‌شود.

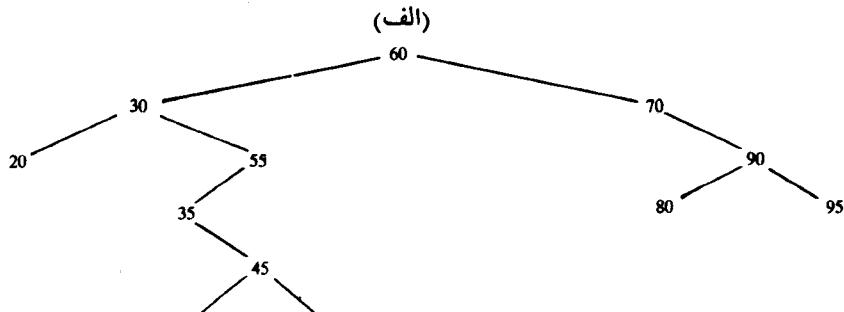
(الف) ریشه R از مقدار اشاره گر ROOT به دست می‌آید. توجه دارید که $\text{INFO}[5] = \text{ROOT} = 5$ از این‌رو $\text{INFO}[5] = 60$ ریشه R درخت T است.

	INFO	LEFT	RIGHT
ROOT	1	20	0
	2	30	1
AVAIL	3	40	0
	4	50	0
	5	60	2
	6	70	0
	7	80	0
	8	90	7
	9		10
	10		0
	11	35	0
	12	45	3
	13	55	11
	14	95	0

شکل ۷-۴۳



(الف)



(ب)

شکل ۷-۴۴

(ب) بچه چپ R از فیلد اشاره گر چپ R بدست می‌آید. توجه دارید که $2 = \text{LEFT}[5]$ از این رو $\text{INFO}[2] = 30$ بچه چپ R است.

(ج) بچه راست R از فیلد اشاره گر R بدست می‌آید. توجه دارید که $6 = \text{RIGHT}[5]$ از این رو $\text{INFO}[6] = 70$ بچه راست R است.

اکنون می‌توانیم قسمت بالای درخت را به صورت شکل ۴۴-۷ (الف) رسم کنیم. با تکرار پردازش بالا با هر گره جدید، نهایتاً T درخت مطلوب در شکل ۴۴-۷ (ب) به دست می‌آید.

مسئله ۲-۷: درخت دودویی T دارای 9 گره است. پیمایش‌های PreOrder و InOrder درخت T دنباله گره‌های زیر را بدست می‌دهد:

Inorder:	E A C K F H D B G
Preorder:	F A E K C D H G B

درخت T را رسم کنید.

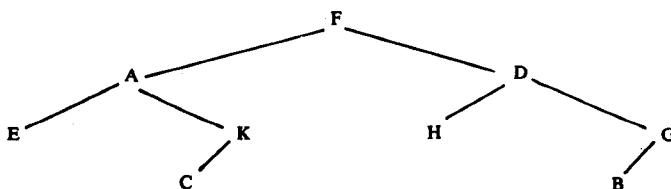
حل: درخت T از ریشه‌اش به طرف پائین به شرح زیر رسم می‌شود.

(الف) ریشه T با انتخاب گره اول در PreOrder اش به دست می‌آید. بدین ترتیب F ریشه T است.

(ب) بچه چپ گره F به صورت زیر به دست می‌آید. نخست برای پیدا کردن گره‌های زیر درخت چپ T_1 از F از پیمایش InOrder استفاده کنید. بدین ترتیب T_1 شامل گره‌های C, A, E و K است. آنگاه بچه چپ با انتخاب گره اول در پیمایش PreOrder زیر درخت T_1 (که در درخت T ظاهر شده است) بچه چپ F به دست می‌آید. بنابراین A پسر چپ F است.

(ج) به همین ترتیب، T_2 زیر درخت راست از گره‌های H, B, D و G تشکیل شده است همچنین D ریشه T_2 یعنی D بچه راست F است.

با تکرار پردازش بالا با هر گره جدید، نهایتاً درخت موردنظر در شکل ۴۵-۷ بدست می‌آید.



شکل ۴۵-۷

مسئله ۳-۷: عبارت جبری $(2x + y)(5a - b)^3 = E$ را درنظر بگیرید:

(الف) درخت T ای رسم کنید که متناظر با عبارت E باشد.

(ب) دامنه فعالیت عملگر توان را تعیین کنید یعنی زیردرختی را پیدا کنید که ریشه آن عملگر توان است.

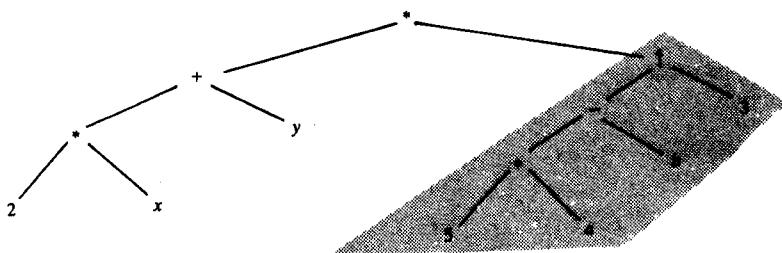
(ج) عبارت لهستانی پیشوندی P را که معادل E است پیدا کنید همچنین نمایش PreOrder T را تعیین کنید.

حل : (الف) با استفاده از پیکان \uparrow برای توان و ستاره * برای ضرب، درخت نشان داده شده در شکل ۷-۴۶ به دست می آید.

(ب) دامنه فعالیت \uparrow درخت سایه زده شده شکل ۷-۴۶ است، که متناظر با عبارت^۳ (5a - b) است.

(ج) تفاوتی بین عبارت لهستانی پیشوندی P و نمایش PreOrder درخت T وجود ندارد. درخت T را مانند شکل ۷-۷ از چپ جستجو و می خوانیم به دست می آید :

$$* + * 2 * x y \uparrow - * 5 a b 3$$



شکل ۷-۴۶

مسأله ۴-۷ : فرض کنید T یک درخت دودویی در حافظه باشد. یک زیربرنامه بازگشتی بنویسید که تعداد گرهای درخت T را پیدا کند.

حل : NUM تعداد گرهای در T یک واحد بیشتر از NUML تعداد گرهای زیردرخت چپ T به اضافه NUMR تعداد گرهای زیردرخت راست T است. بنابراین

Procedure P7.4: COUNT(LEFT, RIGHT, ROOT, NUM)
This procedure finds the number NUM of nodes in a binary tree T in memory.

1. If ROOT = NULL, then: Set NUM := 0, and Return.
2. Call COUNT(LEFT, RIGHT, LEFT[ROOT], NUML).
3. Call COUNT(LEFT, RIGHT, RIGHT[ROOT], NUMR).
4. Set NUM := NUML + NUMR + 1.
5. Return.

مالحظه می کنید که آرایه INFO هیچ نقشی در این زیربرنامه Procedure ایفا نمی کند.

مسأله ۷-۵ : فرض کنید T یک درخت دودویی در حافظه باشد. یک زیربرنامه بازگشتی بنویسید تا عمق درخت T را پیدا کند.

حل : عمق DEP درخت T یک واحد بیشتر از حداقل عمق های زیردرختهای چپ و راست T است.

بنابراین:

Procedure P7.5: DEPTH(LEFT, RIGHT, ROOT, DEP)

This procedure finds the depth DEP of a binary tree T in memory.

1. If ROOT = NULL, then: Set DEP := 0, and Return.
2. Call DEPTH(LEFT, RIGHT, LEFT[ROOT], DEPL).
3. Call DEPTH(LEFT, RIGHT, RIGHT[ROOT], DEPR).
4. If DEPL \geq DEPR, then:
 Set DEP := DEPL + 1.
Else:
 Set DEP := DEPR + 1.
[End of If structure.]
5. Return.

ملاحظه می‌کنید که آرایه INFO هیچ نقشی در این زیربرنامه Procedure ایفا نمی‌کند.

مسئله ۶-۷: تمام درختهای غیرمتشابه ممکن T را رسم کنید که در آن :

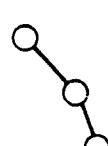
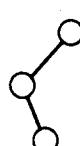
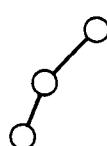
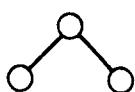
(الف) T یک درخت دودویی با ۳ گره است.

(ب) T یک ۲-درخت با ۴ گره خارجی است.

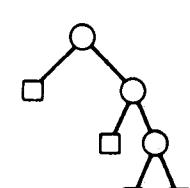
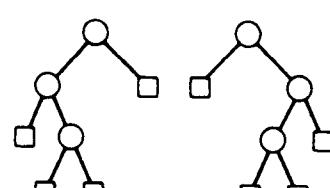
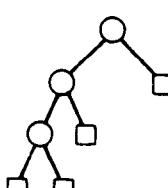
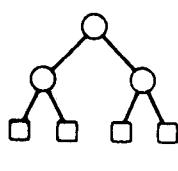
حل : (الف) ۵ درخت از این نوع وجود دارد که در شکل ۷-۴۷ (الف) رسم شده‌اند.

(ب) هر ۲-درخت با ۴ گره خارجی به وسیله یک درخت دودویی با ۳ گره یعنی به وسیله درخت

قسمت (الف) تعیین می‌شود. بدین ترتیب پنج درخت از این نوع وجود دارد که در شکل ۷-۴۷ (ب) رسم شده‌اند.



(الف) درختهای دودویی با ۳ گره



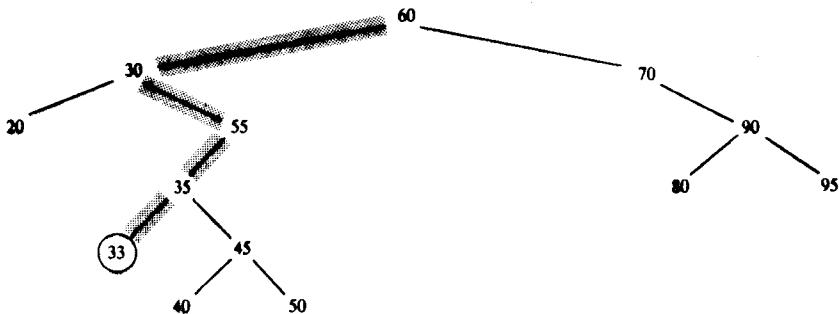
(ب) درخت دودویی گسترش یافته با ۴ گره خارجی

درختهای جستجوی دودویی، Heap ها

مسأله ۷-۷: درخت جستجوی دودویی T شکل ۷-۴۴ (ب) را در نظر بگیرید که مطابق شکل ۷-۴۳ است. حافظه ذخیره می‌شود. فرض کنید $ITEM = 33$ به درخت T اضافه شده است.

(الف) درخت جدید T را پیدا کنید. (ب) چه تغییری در شکل ۷-۴۳ وجود می‌آید؟

حل: (الف) $ITEM = 33$ را با ریشه یعنی 60 مقایسه کنید. چون $60 > 33$ ، آن را به طرف بجهة چپ، 30 منتقل کنید. از آنجا که $30 < 33$ ، آن را به طرف بجهة راست، 55 منتقل کنید. چون $33 < 55$ ، آن را به طرف بجهة چپ، 35 منتقل کنید. اکنون $35 < 33$ ، اما 35 بجهة چپی ندارد. از این‌رو $ITEM = 33$ را به عنوان بجهة چپ گره 35 اضافه کنید تا درخت شکل ۷-۴۸ به دست آید. يالهای سایه زده شده بیان می‌کند که در طی اجرای الگوریتم اضافه کردن، درخت مسیر را به طرف پائین طی می‌کنیم.



شکل ۷-۴۸

(ب) نخست، $ITEM = 33$ در گره موجود اول جایگزین می‌شود. چون $9 = AVAIL$ ، قرار دهید $INFO[9] := 33$ و قرار دهید $0 = LEFT[9]$ و $0 = RIGHT[9]$ علاوه بر این قرار دهید $10 = AVAIL$ که گره موجود بعدی است. بالاخره، قرار دهید $9 = LEFT[11]$ طوری که $ITEM = 33$ باشد. شکل ۷-۴۹ درخت تازه شده T را در حافظه نشانو می‌دهد. سایه زدگی به انگریزه تغییرات در نمودار اصلی است.

	INFO	LEFT	RIGHT
ROOT	1 20	0	0
AVAIL	2 30	1	13
5	3 40	0	0
10	4 50	0	0
	5 60	2	6
	6 70	0	8
	7 80	0	0
	8 90	7	14
	9 100	0	0
	10	0	
	11 35	0	12
	12 45	3	4
	13 55	11	0
	14 95	0	0

شکل ۷-۴۹

مسئله ۷: فرض کنید لیست حروف زیر به ترتیب در یک درخت جستجوی دودویی خالی اضافه شده است:

J, R, D, G, T, E, M, H, P, A, F, Q

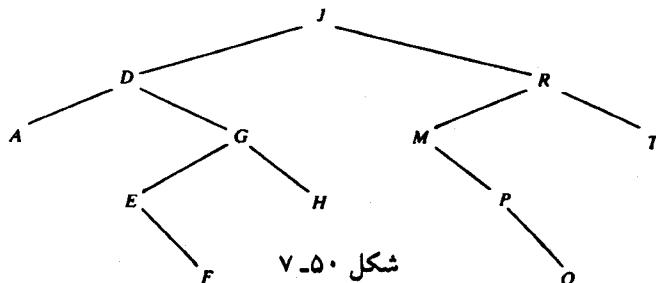
(الف) درخت نهایی T و (ب) پیمایش InOrder درخت T را پیدا کنید.

حل: (الف) گره‌ها را یکی پس از دیگری اضافه کنید تا درخت شکل ۷-۵۰ به دست آید.

(ب) پیمایش InOrder درخت T به صورت زیر است:

A, D, E, F, G, H, J, M, P, Q, R, T

ملاحظه می‌کنید که عبارت بالا، لیست الفبایی حروف است.



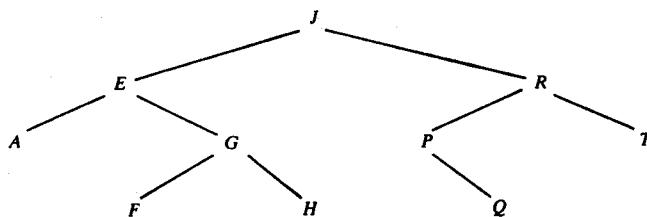
شکل ۷-۵۰

مسئله ۷-۹: درخت جستجوی دودویی T در شکل ۷-۵۰ را درنظر بگیرید. وضعیت درخت را پس از

(الف) حذف گره M (ب) همچنین حذف گره D شرح دهد.

حل: (الف) گره M تنها یک بچه P دارد. از این رو با حذف M ، P بچه چپ R به جای M می‌نشیند.

(ب) گره D دو بچه دارد. گره بعدی پیماش $InOrder$ مربوط به D را پیدا کنید که گره E است. نخست E را از درخت حذف کنید و آنگاه D را جانشین E کنید.



شکل ۷-۵۱

مسئله ۱۰-۷: فرض کنید n عنصری اطلاعاتی A_1, A_2, \dots, A_N ازقبل مرتب شده هستند یعنی

$$A_1 < A_2 < \dots < A_N$$

(الف) با فرض این که عناصر به ترتیب در درخت جستجوی دودویی خالی اضافه می‌شوند و وضعیت درخت نهایی T را توضیح دهد.

(ب) عمق D درخت T را تعیین کنید.

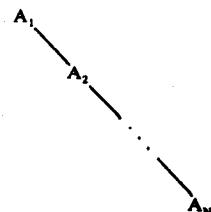
(ج) به ازای (i) $n = 50$ (ii) $n = 100$ (iii) $n = 500$ (iv) $n = 1000$ یا n $Average\ Depth$ AD را با عمق میانگین یک درخت جستجوی دودویی با n گره مقایسه کنید.

حل: (الف) درخت از یک شاخه تشکیل شده است که در طرف راست همانند شکل ۷-۵۲ می‌باشد.

(ب) چون T یک شاخه دارد که دارای تمام n گره است، درنتیجه $n = D$.

(ج) می‌دانیم که $AD = c \log_2 n$ ، که در آن $c \approx 1.4$ از این رو

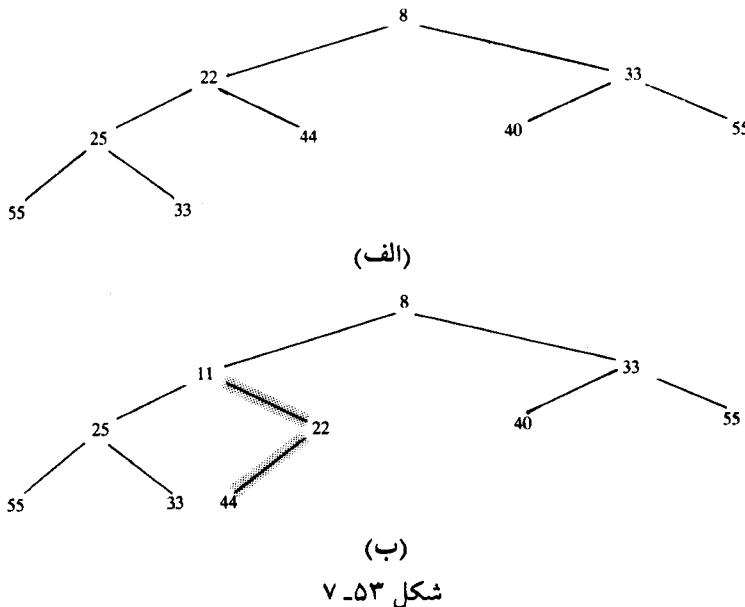
$$D(50) = 50, AD(50) \approx 9; D(100) = 100, AD(100) \approx 10; D(500) = 500, AD(500) \approx 12$$



شکل ۷-۵۲

مسئله ۱۱-۷ : $H = \text{MinHeap}$ شکل ۵۳-۷ (الف) را در نظر بگیرید. (یک H است چون عناصرهای کوچکتر به عوض عناصرهای بزرگتر، بالای Heap هستند). وضعیت Heap را پس از اضافه شدن $ITEM = 11$ در H توضیح دهید.

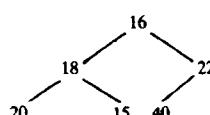
حل : نخست $ITEM$ را به عنوان گره بعدی یعنی به عنوان بچه چپ گره ۴۴ در درخت کامل اضافه کنید. آنگاه بطور مکرر $ITEM$ را با پدرش مقایسه کنید. چون $11 < 44$ ، جای ۱۱ و ۴۴ را عوض کنید. از آنجاکه $11 < 22$ ، جای ۱۱ و ۲۲ را عوض کنید. چون $8 < 11$ ، $11 < 11$ مکان مربوطه اش را در Heap پیدا کرده است. شکل ۵۳-۷ (ب) تازه شده H را نشان می‌دهد. يالهای سایه‌زده شده بیانگر مسیر $ITEM$ به طرف بالای درخت است.



مسئله ۱۲-۷ : درخت کامل T با $N = 6$ گره شکل ۵۴-۷ را در نظر بگیرید. فرض کنید یک Heap از T با استفاده از

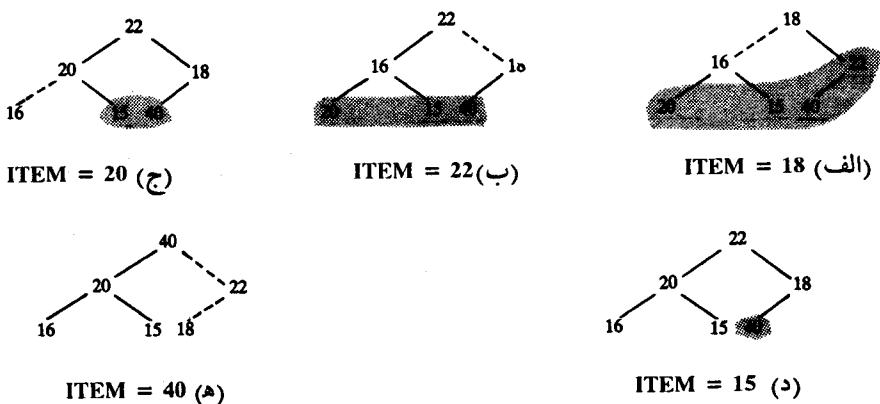
Call INSHEAP($A, J, A[J + 1]$)

به ازای $1, 2, \dots, N - 1$ ساخته ایم. (در اینجا T به صورت ترتیبی در آرایه A ذخیره می‌شود). مراحل مختلف را توضیح دهید.



- شکل ۷-۵۵ ۷ مراحل مختلف را نشان می‌دهد. ما هر مرحله را بطور مستقل توضیح می‌دهیم.
- (الف) $J = 1$ و $ITEM = A[2] = 18$. چون $18 > 16$ ، جای ۱۸ و ۱۶ را عوض کنید.
- (ب) $J = 2$ و $ITEM = A[3] = 22$. چون $18 > 22$ ، جای ۲۲ و ۱۸ را عوض کنید.
- (ج) $J = 3$ و $ITEM = A[4] = 20$. چون $16 > 20$ اما $22 < 20$ ، تنها جای ۲۰ و ۱۶ را عوض کنید.
- (د) $J = 4$ و $ITEM = A[5] = 15$. چون $20 < 15$ ، هیچ جابجایی صورت نمی‌گیرد.
- (ه) $J = 5$ و $ITEM = A[6] = 40$. چون $18 > 40$ و $22 > 40$ ، نخست جای ۴۰ و ۱۸ و سپس جای ۴۰ و ۲۲ را عوض کنید.

اکنون درخت یک Heap است. بالهای خط چین شده بیان می‌کنند که یک جابجایی صورت گرفته است منطقه سایه‌دار نشده بیان‌گر قسمتی از درخت می‌باشد که Heap می‌سازد. ملاحظه می‌کنید که Heap از بالا به پائین ایجاد می‌شود (هرچند عناصر، به صورت انفرادی، به بالای درخت هجرت می‌کنند!).



شکل ۷-۵۵

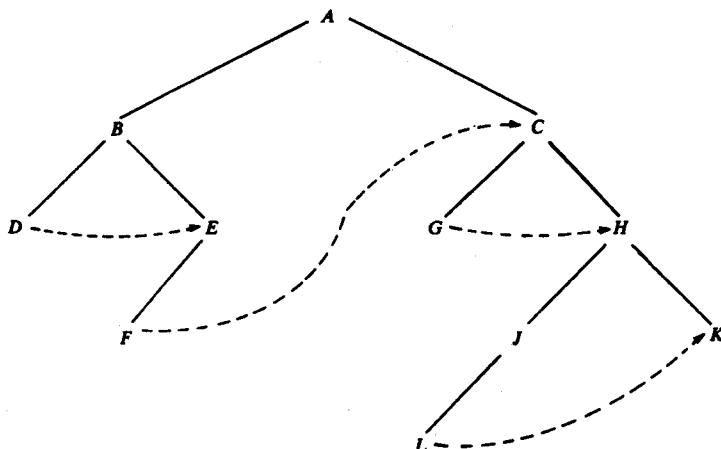
مسئله‌های گوناگون

مسئله ۱۳-۷: درخت دودویی T شکل ۱-۷ را درنظر بگیرید (الف) نخ‌کشی PreOrder یکطرفه T را تعیین کنید. (ب) نخ‌کشی PreOrder دوطرفه T را تعیین کنید.

حل: (الف) به جای نخ‌کشی ای که به گره بعدی N در پیمایش PreOrder درخت T اشاره می‌کند زیر درخت خالی راست گره انتهایی N را قرار دهید. بدین ترتیب، یک نخ‌کشی از D به E وجود دارد چون در پیمایش PreOrder درخت T، پس از D ملاقات می‌شود. به همین ترتیب، یک نخ‌کشی از F به C، از

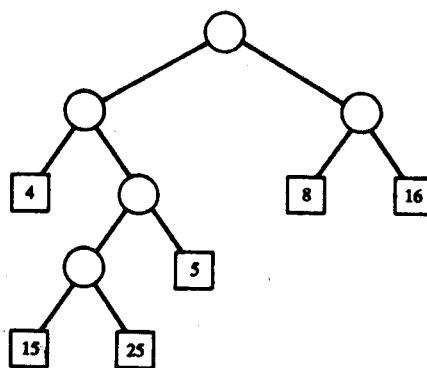
G به **H** و از **L** به **K** وجود دارد. درخت نخ‌کشی شده در شکل ۵۶-۷ نشان داده شده است. گره انتهایی **K** هیچ نخ‌کشی ندارد چون آخرین گره پیمایش PreOrder درخت **T** است. از طرف دیگر، اگر **T** یک سرگره **Z** داشته باشد، آنگاه یک نخ‌کشی از **K** به سرگره **Z** وجود خواهد داشت.

(ب) هیچ نخ‌کشی PreOrder دو طرفه برای درخت **T** وجود ندارد که مشابه نخ‌کشی InOrder دو طرفه **T** باشد.



شکل ۵۶-۷ درخت Preorder نخ‌کشی شده

مسأله ۱۴-۷: درخت **T** وزن داده شده شکل ۵۷-۷ را درنظر بگیرید.



شکل ۵۷-۷

P طول مسیر وزن داده شده درخت T را به دست آورید.

حل: هر وزن W_i را در طول L_i مسیر از ریشه T تا گرهای که دارای وزن است ضرب کنید و آنگاه تمام این حاصلضربها را با هم جمع کنید تا P بدست آید. بنابراین:

$$P = 4.2 + 15.4 + 25.4 + 5.3 + 8.2 + 16.2 = 8 + 60 + 100 + 15 + 16 + 32 = 231$$

مسئله ۱۵-۷: فرض کنید شش وزن 4، 15، 25، 5، 8، 16 داده شده است. یک ۲-درخت با وزن‌های داده شده و یک طول مسیر وزن داده شده حداقل P پیدا کنید. T را با درخت شکل ۷-۵۷ مقایسه کنید.

حل: از الگوریتم هافمن استفاده کنید، یعنی بطور مکرر دو زیردرخت با حداقل وزنها را در یک زیردرخت به شرح زیر ترکیب کنید:

(الف) 4، 15، 25، 5، 8، 16

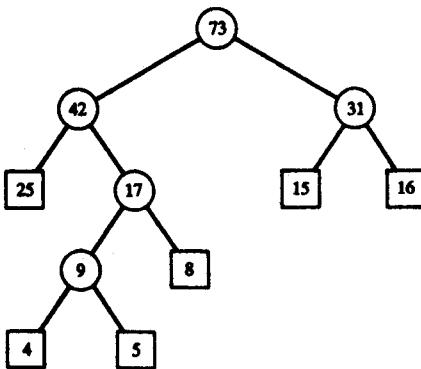
(ب) 15، 25، 9، 8، 16

(ج) 15، 25، 17، 16

(د) 25، 17، 31

(ه) 42، 31

(و) 73



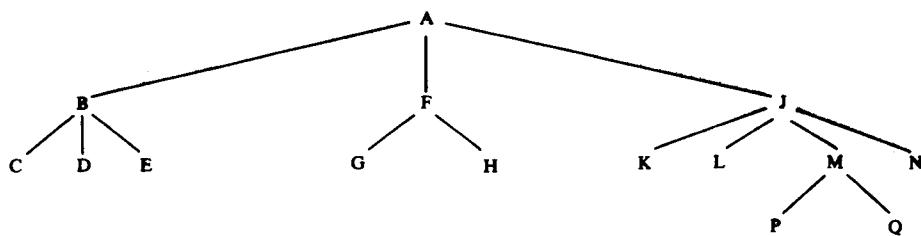
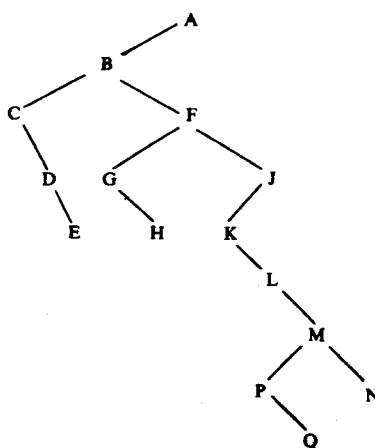
شکل ۷-۵۸

عدد داخل دایره بیانگر ریشه زیردرخت جدید در این فرایند است. زیردرخت T از مرحله (و) به بالا رسم شده است که حاصل آن شکل ۷-۵۸ است. P با این درخت به صورت زیر محاسبه می‌شود:

$$P = 25.2 + 4.4 + 5.4 + 8.3 + 15.2 + 16.2 = 50 + 16 + 20 + 24 + 30 + 32 = 172$$

طول مسیر وزن داده شده درخت شکل ۷-۵۷، 231 است.

مسأله ۱۶-۷: درخت عمومی T شکل ۵۹-۷ (الف) را درنظر بگیرید درخت دودویی T' متناظر با آن را پیدا کنید.

(الف) درخت عمومی T (ب) درخت دودویی T'

شکل ۵۹-۷

حل: گره‌های T' همان گره‌های درخت عمومی T خواهند بود همچنین در حالت خاص ریشه T' همان ریشه درخت T است. هلاوه بر این، اگر N یک گره در درخت دودویی T' باشد آنگاه بجهة چپ آن، بجهة اول N در T و بجهة راست آن، همردیف بعدی N در T است. T' را از ریشه به طرف پائین می‌سازیم، درخت شکل ۵۹-۷ (ب) حاصل می‌شود.

مسأله ۱۷-۷: فرض کنید T یک درخت عمومی با ریشه R و زیردرختهای T_1, T_2, \dots, T_M باشد. پیمایش $PostOrder$ و $PreOrder$ درخت T به صورت زیر تعریف می‌شوند:

(۱) $PostOrder$: ریشه R را پردازش کنید.

(۲) زیردرختهای T_1, T_2, \dots, T_M را با روش PreOrder پیمایش کنید.

: (۱) زیردرختهای T_1, T_2, \dots, T_M را با روش PostOrder پیمایش کنید.

(۲) ریشه R را پردازش کنید.

فرض کنید T درخت عمومی شکل ۵۹ (الف) باشد. (الف) T را با روش PreOrder پیمایش کنید.

(ب) T را با روش PostOrder پیمایش کنید.

حل: توجه دارید که R ریشه A دارد و زیردرختها T_1, T_2 و T_3 هستند به گونه‌ای که:

T_1 از گره‌های B, C, D تشکیل شده است.

T_2 از گره‌های F, G, H تشکیل شده است.

T_3 از گره‌های J, K, L, M, N, P, Q تشکیل شده است.

(الف) پیمایش PreOrder درخت T از مرحله‌های زیر تشکیل می‌شود:

(i) Rیشه A را پردازش کنید.

(ii) T را با روش PreOrder پیمایش کنید: گره‌های B, C, D, E را پردازش کنید.

(iii) T را با روش PreOrder پیمایش کنید: گره‌های F, G, H را پردازش کنید.

(iv) T را با روش PreOrder پیمایش کنید: گره‌های J, K, L, M, N, P, Q را پردازش کنید.

به بیان دیگر، پیمایش Preorder درخت T به شرح زیر است:

A, B, C, D, E, F, G, H, J, K, L, M, P, Q, N

(ب) پیمایش PostOrdr از مرحله‌های زیر تشکیل می‌شود:

(i) T را با روش PostOrder پیمایش کنید: گره‌های C, D, E و B را پردازش کنید.

(ii) T را با روش PostOrder پیمایش کنید: گره‌های G, H و F را پردازش کنید.

(iii) T را با روش PostOrder پیمایش کنید: گره‌های K, L, M, N و J را پردازش کنید.

(iv) Rیشه A را پردازش کنید.

به بیان دیگر، پیمایش Postorder درخت T به شرح زیر است:

C, D, E, B, G, H, F, K, L, P, Q, M, N, J, A

مسئله ۷-۱۸: درخت دودویی T شکل ۵۹-(ب) را در نظر بگیرید. پیمایش‌های InOrder، PreOrder و PostOrder درخت' T را تعیین کنید. آنها را با پیمایش‌های PreOrder و PostOrder به دست آمده در

مسئله ۷-۱۷ درخت عمومی T در شکل ۵۹-(الف) مقایسه کنید.

حل: با استفاده از الگوریتم‌های پیمایش درخت دودویی بخش ۴-۷، پیمایش‌های زیر از 'T' به دست می‌آید:

Preorder: A, B, C, D, E, F, G, H, J, K, L, M, P, Q, N

Inorder: C, D, E, B, G, H, F, K, L, P, Q, M, N, J, A

Postorder: E, D, C, H, G, Q, P, N, M, L, K, J, F, B, A

ملاحظه می‌کنید که پیمایش PreOrder درخت دودویی T' همان پیمایش PreOrder درخت عمومی T است، همچنین پیمایش InOrder درخت دودویی T' همان پیمایش PostOrder درخت عمومی T است. هیچ پیمایش طبیعی و درخت عمومی برای T وجود ندارد که متناظر با پیمایش PostOrder درخت دودویی T' مربوطه باشد.

مسائلهای تكميلی

درختهای دودویی

مسئله ۱۹-۷: درخت T در شکل ۶۰-۷ (الف) را درنظر بگیرید:

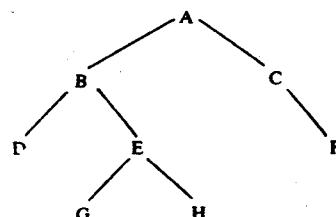
(الف) مقادیر LEFT و RIGHT شکل ۶۰-۷ (ب) را طوری پرکنید که T در حافظه ذخیره خواهد شد.

(ب) مطلوب است تعیین (i) عمق درخت T (ii) تعداد زیردرختهای خالی یا پرچ و (iii) نسل‌های گره B .

مسئله ۲۰-۷: لیست گره‌های درخت T در شکل ۶۰-۷ (الف) را با روش (۱) PreOrder و InOrder (۲) و PostOrder (۳) بنویسید.

	INFO	LEFT	RIGHT
ROOT	A		
AVAIL	5		
1	A		
2	C		
3	D		
4	G		
5		6	
6		0	
7	H		
8	F		
9	E		
10	B		

(ب)



(الف)

شکل ۶۰-۷

مسئله ۲۱-۷: نمودار درخت T در شکل ۱-۶۱-۷ را رسم کنید.

	INFO	LEFT	RIGHT
ROOT 14	H	4	11
	R	0	0
		17	
	P	0	0
	B	18	7
AVAIL 8		3	
	E	1	0
		6	
	C	0	10
	F	15	16
	Q	0	12
	S	0	0
		0	
	A	5	9
	K	2	0
	L	0	0
		13	
	D	0	0

شکل ۶۱

مسئله ۲۲-۷: فرض کنید دنباله زیرلیست گره‌های یک درخت دودویی T به ترتیب در پیمایش باشند: InOrder و PreOrder

Preorder: G, B, Q, A, C, K, F, P, D, E, R, H

Inorder: Q, B, K, C, F, A, G, P, E, D, H, R

نمودار درخت رارسم کنید.

مسئله ۲۳-۷: فرض کنید T یک درخت دودویی در حافظه باشد و عنصر اطلاعاتی ITEM داده شده است.

(الف) زیربرنامه‌ای بنویسید که مکان LOC عنصر ITEM را در درخت T پیدا کند. فرض می‌شود عنصرهای درخت T متمایز هستند.

(ب) زیربرنامه‌ای بنویسید که مکان LOC عنصر ITEM و مکان پدر PAR عنصر ITEM را در درخت T پیدا کند.

(ج) زیربرنامه‌ای بنویسید که NUM تعداد دفاتر ظاهرشدن عنصر ITEM را در درخت T پیدا کند. فرض می‌شود عنصرهای T الزاماً متمایز نیستند.
توجه کنید: T الزاماً یک درخت جستجوی دودویی نیست.

مسأله ۲۴-۷: فرض کنید T یک درخت دودویی در حافظه باشد. یک زیربرنامه غیربازگشتی برای هر یک از موارد زیر بنویسید:

(الف) تعداد گره‌های درخت T را پیدا کنید.

(ب) عمق درخت T را پیدا کنید.

(ج) تعداد گره‌های انتهایی درخت T را پیدا کنید.

مسأله ۲۵-۷: فرض کنید T یک درخت دودویی در حافظه باشد. یک زیربرنامه Procedure بنویسید که تمام گره‌های انتهایی درخت T را حذف کند.

مسأله ۲۶-۷: فرض کنید ROOTA به درخت دودویی T_1 در حافظه اشاره می‌کند. یک زیربرنامه Procedure بنویسید که با استفاده از اشاره گر ROOTB، T_1 را در T_2 کپی کند.

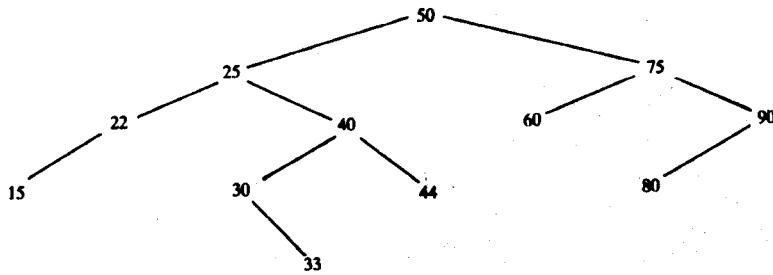
درخت‌های جستجوی دودویی

مسأله ۲۷-۷: فرض کنید هشت عدد زیر به ترتیب به یک درخت جستجوی دودویی خالی T اضافه شده‌اند.

50, 33, 44, 22, 77, 35, 60, 40

درخت را رسم کنید.

مسأله ۲۸-۷: درخت جستجوی دودویی T شکل ۷-۶۲ را در نظر بگیرید:



شکل ۷-۶۲

اگر هر یک از عملیات زیر بر درخت اصلی T اعمال شود، درخت T را رسم کنید. به عبارت دیگر، عملیات به طور مستقل اعمال می‌شوند نه به طور متوالی.

(الف) گره 20 به درخت T اضافه می‌شود. (د) گره 22 از درخت T حذف می‌شود.

(ب) گره 15 به درخت T اضافه می‌شود. (ه) گره 25 از درخت T حذف می‌شود.

(ج) گره 88 به درخت T اضافه می‌شود. (و) گره 75 از درخت T حذف می‌شود.

مسأله ۲۹-۷: درخت جستجوی دودویی T در شکل ۶۲-۷ را درنظر بگیرید. اگر شش عمل مسأله

۲۸-۷ یکی پس از دیگری (نه به طور مستقل) بر درخت T اعمال شود درخت نهایی T را رسم کنید.

مسأله ۳۰-۷: درخت جستجوی دودویی شکل ۶۳-۷ را رسم کنید.

	INFO	LEFT	RIGHT
ROOT 4	Jones	7	0
AVAIL 3	Fox	11	1
		8	
	Murphy	2	15
		13	
	Thomas	0	0
	Green	0	0
		9	
		10	
		5	
	Conroy	0	0
	Parker	0	0
		14	
		0	
	Rosen	12	6

شکل ۶۳-۷

مسأله ۳۱-۷: درخت جستجوی دودویی T شکل ۶۳-۷ را درنظر بگیرید. اگر هر یک از عملیات زیر به طور مستقل (نه به طور متوالی) بر درخت T اعمال شود تغییراتی را که در 'INFO'، 'LEFT' و 'RIGHT' بوجود می‌آید توضیح دهید.

(د) از T حذف می‌شود. (الف) Davis به T اضافه می‌شود.

(ه) از T حذف می‌شود. (ب) Harris به T اضافه می‌شود.

(و) از T حذف می‌شود. (ج) Smith به T اضافه می‌شود.

مسأله ۳۲-۷: درخت جستجوی دودویی T شکل ۶۳-۷ را درنظر بگیرید. اگر شش عمل مسأله ۳۱-۷ یکی پس از دیگری (نه بطور مستقل) بر T اعمال شود تغییرات نهایی در **INFO**, **LEFT**, **RIGHT**, **AVAIL** و **ROOT** را توضیح دهید.

مسأله‌های متفرقه

مسأله ۳۳-۷: درخت دودویی T در شکل ۶۰-۷ (الف) را درنظر بگیرید.

(الف) نخ‌کشی **InOrder** یکطرفه T را رسم کنید.

(ب) نخ‌کشی **PreOrder** یکطرفه T را رسم کنید.

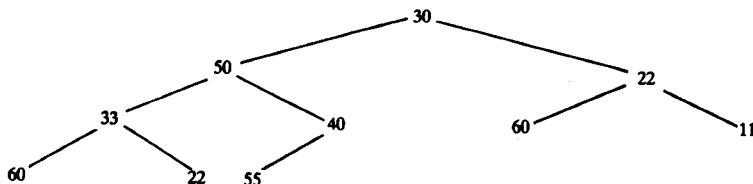
(ج) نخ‌کشی **InOrder** دوطرفه T را رسم کنید.

در هر حالت بالا، چگونگی ذخیره درخت نخ‌کشی شده را در حافظه با استفاده از اطلاعات شکل ۶۰-۷ نشان دهید.

مسأله ۳۴-۷: درخت کامل T با $N = 10$ گره در شکل ۶۴-۷ را درنظر بگیرید. فرض کنید يك از درخت کامل T با اعمال **MaxHeap**

Call **INSHEAP(A, J, A[J + 1])**

برای $1 \leq J \leq N - 1$, $A[J] = 1, 2, \dots, N$ ساخته می‌شود. فرض کنید T بطور متواالی در آرایه A ذخیره می‌شود. **MaxHeap** نهایی را پیدا کنید.



شکل ۶۴-۷

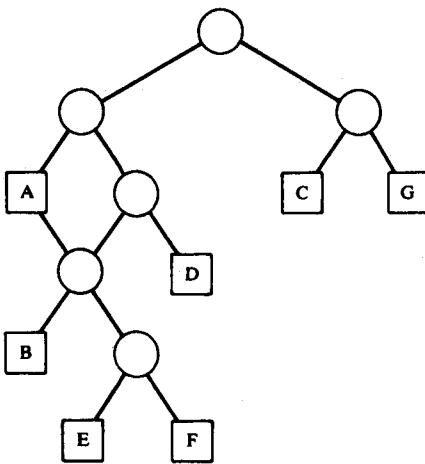
مسأله ۳۵-۷: مسأله ۳۴-۷ را برای درخت T در شکل ۶۴-۷ حل کنید با این تفاوت که اکنون بدون درنظر گرفتن T یک **MinHeap** بجای **MaxHeap** بسازید.

مسأله ۳۶-۷: ۲-درخت متناظر با هر یک از عبارتهای جبری زیر را رسم کنید:

$$E_1 = (a - 3b)(2x - y)^3 \quad (\text{الف})$$

$$E_2 = (2a + 5b)^3(x - 7y)^4 \quad (\text{ب})$$

مسئله ۳۷-۲: درخت شکل ۶۵-۷ را در نظر بگیرید.



شکل ۶۵-۷

مطلوب است تعیین کد هافمن هفت حرفی، که در درخت T معین شده است.

مسئله ۳۸-۷: فرض کنید به ۷ عنصر اطلاعاتی A, B, ..., G وزنهای زیر نسبت داده شده است :

(A, 13), (B, 2), (C, 19), (D, 23), (E, 29), (F, 5), (G, 9)

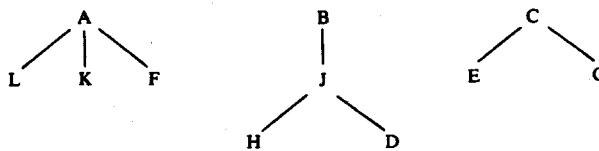
P طول مسیر وزن داده شده درخت شکل ۶۵-۷ را بدست آورید.

مسئله ۳۹-۷: با استفاده از اطلاعات مسئله ۳۸-۷، یک ۲-درخت با حداقل طول مسیر وزن داده شده P پیدا کنید. برای ۷ حرف، با استفاده از این درخت جدید کد هافمن تعیین کنید.

مسئله ۴۰-۷: جنگل F شکل ۶۶-۷ را در نظر بگیرید که از سه درخت و به ترتیب با ریشه‌های A، B و C تشکیل شده است.

(الف) درخت دودویی F متناظر با جنگل F را پیدا کنید.

(ب) مقادیر ROOT و CHILD و SIB شکل ۶۷-۷ را به گونه‌ای پر کنید که F در حافظه ذخیره شود.



شکل ۷-۶۶ جنگل F

ROOT	INFO	CHILD	SIB
1	A		
2	C		
3	E		
4	G		
5	J		
6	L		
7			
8	K		
9	H		
10	F		
11	D		
12	B		

شکل ۷-۶۷

مسئله ۴۱-۷: فرض کنید T یک درخت کامل با n گره و عمق D باشد. ثابت کنید:

$$\text{(الف) } 1 \leq 2^D - 1 < n \leq 2^{D-1}$$

راهنمایی: به ازای $x = 2$ از اتحاد زیر استفاده کنید:

$$1 + x + x^2 + x^3 + \dots + x^n = \frac{x^{n+1} - 1}{x - 1}$$

مسئله ۴۲-۷: فرض کنید T یک درخت دودویی گسترش یافته باشد. ثابت کنید:

$$\text{(الف) } N_E = N_I + 1$$

(ب) L_E که در آن L_E طول مسیر خارجی و L_I طول مسیر داخلی و n تعداد گره‌های داخلی است.

برای مسئله‌های زیر برنامه بنویسید

مسئله‌های ۴۳-۴۷ تا ۷-۷ به درخت T در شکل ۱-۷ مربوط هستند که در حافظه به صورت شکل

۷-۶۸ ذخیره می‌شود.

	INFO	LEFT	RIGHT
ROOTA	1	K	0
	2	C	3
AVAIL	3	G	0
	4		14
	5	A	10
	6	H	17
	7	L	0
	8		9
	9		4
	10	B	18
	11		19
	12	F	0
	13	E	12
	14		15
	15		16
	16		11
	17	J	7
	18	D	0
	19		20
	20		21
	21		22
	22		23
	23		24
	24		0

شکل ۷-۶۸

مسئله ۴۳-۷: برنامه‌ای بنویسید که گره‌های شکل ۶۸-۷ را با روش (الف) PreOrder و (ب) InOrder و (ج) PostOrder چاپ کند.

مسئله ۴۴-۷: برنامه‌ای بنویسید که گره‌های انتهایی T را با روش (الف) PreOrder و (ب) InOrder و (ج) PostOrder چاپ کند.

توجه کنید: هر سه لیست باید عین! هم باشند.

مسئله ۴۵-۷: برنامه‌ای بنویسید که با استفاده از **ROOTB** به عنوان یک اشاره‌گر یک کپی از T بنام 'T' بسازد. با چاپ گره‌های 'T' با روش PreOrder و InOrder و مقایسه آن با لیستهایی که در مسئله ۴۳-۷ بدست آمد برنامه را آزمایش کنید.

مسئله ۴۶-۷: **HeapSort** را به صورت زیر برنامه **HEAPSORT(A, N)** بنویسید که آرایه A با N عنصر را مرتب کند. برنامه را با استفاده از داده‌های زیر آزمایش کنید:

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

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

مسئله‌های ۴۷-۷ تا ۵۲-۷ به لیست رکوردهای کارمندان مربوط می‌شوند که به صورت شکل ۸-۷ یا به صورت شکل ۶۹-۷ ذخیره می‌شوند.

	NAME	SSN	SEX	SALARY	LEFT	RIGHT
HEAD	1				0	
AVAIL	5	Davis	Female	22 800	5	12
8	Kelly	192-38-7282	Male	19 000	5	5
5	Green	165-64-3351	Male	27 200	2	5
6	Brown	175-56-2251		191 600	14	5
7	Lewis	178-52-1065	Female	14 700	5	5
8		181-58-9939	Female	16 400	3	10
9	Cohen		Male	19 000	11	4
10	Rubin	177-44-4557	Female	15 500	6	5
11		135-46-6262			5	5
12	Evans		Male	34 200	13	
13		168-56-8113			5	5
14	Harris	208-56-1654	Female	22 800	1	7

شکل ۶۹-۷

هر یک از آنها یک درخت جستجوی دودویی نسبت به کلید NAME است اما شکل ۶۹-۷ از یک سر گره استفاده می‌کند که به عنوان یک نگهبان نیز عمل می‌کند. این مسئله را با مسائلهای ۵-۴۶ تا ۵-۴۱ فصل ۵ مقایسه کنید.

مسئله ۴۷-۷: برنامه‌ای بنویسید که لیست رکوردهای کارمندان را به ترتیب الفبایی چاپ کند.
راهنمایی: رکوردها را با روش InOrder چاپ کنید.

مسئله ۴۸-۷: برنامه‌ای بنویسید که نام NNN یک کارمند را بخواند و رکورد کارمند را چاپ کند. برنامه را با استفاده از (الف) Smith ، (ب) Evans و (ج) Lewis آزمایش کنید.

مسئله ۴۹-۷: برنامه‌ای بنویسید که شماره تأمین اجتماعی SSS یک کارمند را از ورودی بخواند و رکورد کارمند را چاپ کند. برنامه را با استفاده از (الف) 3351 - 64 - 165 ، (ب) 6262 - 46 - 135 و (ج) 5555 - 44 - 177 آزمایش کنید.

مسئله ۵۰-۷: برنامه‌ای بنویسید که عدد صحیح K را از ورودی بخواند و نام هر کارمند مرد را وقتی که K = 1 یا نام هر کارمند زن را وقتی $2 = K$ است چاپ کند. برنامه را با استفاده از (الف) 2 = K ، (ب) 5 = K و (ج) 1 = K آزمایش کنید.

مسئله ۵۱-۷: برنامه‌ای بنویسید که نام NNN یک کارمند را بخواند و رکورد کارمند را از این ساختمان حذف کند. برنامه را با استفاده از (الف) Davis ، (ب) Jones و (ج) Rubin آزمایش کنید.

مسئله ۵۲-۷: برنامه‌ای بنویسید که رکورد یک کارمند جدید را بخواند و رکورد را در فایل اضافه کند.
برنامه را با استفاده از :

(الف) Fletcher; 168 - 52 - 3388; Female; 21 000

(ب) Nelson; 175 - 32 - 2468; Male; 19 000

اجرا کنید.

فصل ۸

گرافها و کاربردهای آن

۱- مقدمه

این فصل یکی دیگر از ساختمان داده غیرخطی، موسوم به گراف را مورد بررسی قرار می‌دهد. همانند روشی که در مورد دیگر ساختمان داده‌ها بکار برده‌ایم، در مورد گرافها نیز ابتدا روش نمایش آنها را در حافظه توضیح می‌دهیم و سپس عملیات و الگوریتم‌های متعدد روی آنها را بررسی می‌کنیم. به خصوص، جستجوی عرضی یا ردیفی و جستجوی عمقی گرافها را مورد بحث و بررسی قرار می‌دهیم. کاربردهای متعدد گرافها به همراه روش‌های مرتب‌سازی موضعی نیز، دیگر مطالب این فصل را تشکیل می‌دهد.

۲- چند اصطلاح نظریه گراف

در این بخش بطور خلاصه برخی از اصطلاحات اصلی نظریه گراف ارائه می‌شود. متأسفانه باید گفت که هیچ اصطلاح استانداردی در نظریه گراف وجود ندارد. از این‌رو به دانشجویان اطلاع می‌دهیم که تعریف‌های ارائه شده در این کتاب ممکن است با تعریف‌هایی که در دیگر کتب ساختمان داده‌ها و نظریه گراف ارائه می‌شود اندکی متفاوت باشد.

گرافها و گرافهای چندگانه

یک گراف G از دو مجموعه زیر تشکیل می‌شود:

(۱) V مجموعه‌ای از عناصر که گره‌ها یا نقاط یا رأس‌ها نامیده می‌شود.

(۲) E مجموعه‌ای از یال‌ها طوری که هر یال e در E به وسیله یک جفت منحصر بفرد (نامرتب)

$[u, v]$ از گره‌ها در V مشخص می‌شود و آن را با $e = [u, v] =$ نشان می‌دهند.

گاهی اوقات قسمت‌هایی از یک گراف را به صورت $(V, E) = G$ بیان می‌کنند.

فرض کنید $e = [u, v]$ ، آنگاه گره‌های u و v نقاط پایانی e نامیده می‌شوند همچنین به u و v گره‌های مجاور یا همسایه می‌گویند. درجه گره u که به صورت $\deg(u)$ نوشته می‌شود تعداد بالهایی است که شامل u می‌شوند. اگر $\deg(u) = 0$ یعنی اگر u به هیچ یالی تعلق نداشته باشد آنگاه u را گره منفرد می‌نامند.

یک مسیر P با طول n از یک گره u به گره v به صورت دنباله‌ای از $n+1$ گره تعریف می‌شود.

$$P = (v_0, v_1, v_2, \dots, v_n)$$

طوری که $v_0 = u$ ؛ v_1 به ازای $i = 1, 2, \dots, n$ $v_i = v$ مجاور v_{i-1} است. مسیر P را بسته گویند

اگر $v_0 = v_n$. مسیر P را ساده گویند اگر تمام گره‌های آن متمایز باشند با این تفاوت که v_0 ممکن است با v_n یکی باشد یعنی P ساده است اگر گره‌های $v_0, v_1, \dots, v_{n-1}, v_n$ متمایز و نیز گره‌های v_1, v_2, \dots, v_n متمایز باشند.

یک دور یا حلقه مسیر ساده بسته‌ای به طول ۳ یا بیشتر است. یک دور به طول K را K دور یا حلقه گویند. گراف G را همبند گویند اگر بین هر دو گره آن مسیری وجود داشته باشد.

در مسئله ۸-۸ نشان خواهیم داد که اگر از گره u به گره v یک مسیر وجود داشته باشد آنگاه پس از حذف بالهای غیرضروری، می‌توان یک مسیر ساده Q از u به v بدست آورد. بنابراین می‌توان قضیه زیر را بیان کرد.

قضیه ۸-۸: گراف G همبند است اگر و فقط اگر یک مسیر ساده بین هر دو گره G وجود داشته باشد.
 گراف G را کامل گویند اگر هر گره u در G مجاور هر گره دیگر v در G باشد، واضح است که چنین گرافی همبند است. گراف کامل با n گره، $\binom{n}{2} = n(n-1)/2$ یال دارد. گراف کامل T بدون هیچ دور یا حلقه را یک گراف درختی یا درخت آزاد یا تنها یک درخت می‌گویند. در حالت خاص معنی آن، این است که یک مسیر ساده منحصر بفرد بین هر دو گره u و v در T (مسئله ۸-۸) وجود دارد. علاوه بر این، اگر T یک درخت متناهی با m گره باشد آنگاه T دارای $m-1$ یال است (مسئله ۸-۲۰).
 گراف G را شماره دار یا برچسب دار گویند اگر اطلاعاتی به بالهای آن نسبت داده شود. به ویژه، گراف

G را وزن داده شده گویند اگر به هر یال e در G یک مقدار عددی غیر منفی (e) که وزن یا طول e نامیده می شود نسبت داده شود. در چنین حالتی به هر مسیر P در G یک وزن یا طول نسبت داده می شود که وزن یا طول مسیر P ، مجموع وزن یالهایی است که در این مسیر قرار دارند. اگر درمورد وزن ها هیچ اطلاع دیگری داده نشود می توان فرض کرد که گراف G وزن داده شده است یعنی به هر یال e از گراف G وزن $1 = e$ نسبت داده شده است.

تعریف گراف را می توان با مجاز دانستن تعریف های زیر تعمیم داد:

(۱) یالهای چندگانه . یالهای متمایز e و e' را یالهای چندگانه گویند اگر نقاط پایانی یکسانی را به هم وصل کنند یعنی اگر $[u, v] = e$ و $[u, v] = e'$.

(۲) حلقه ها . یال e را حلقه گویند اگر نقاط انتهایی یکسانی داشته باشد یعنی اگر $[u, u] = e$ چنین تعمیمی M را گراف چندگانه یا چند گراف گویند. به بیان دیگر، تعریف یک گراف معمولاً یالهای چندگانه یا حلقه ها را مجاز نمی داند.

یک گراف چندگانه M را متناهی گویند اگر تعداد گره ها و تعداد یالهای آن متناهی باشد. ملاحظه می کنید که یک گراف G با تعداد گره های متناهی باید خود به خود تعداد یالهای متناهی داشته باشد و از این رو باید متناهی باشد اما این مطلب الزاماً در مورد یک گراف چندگانه M برقرار نیست چون M ممکن است یالهای چندگانه داشته باشد. در این کتاب گرافها و گرافهای چندگانه متناهی خواهد بود مگر آن که خلاف آن بیان شود.

مثال ۸ - ۱

(الف) شکل ۸ - ۱ (الف)، نمودار یک گراف همبند با ۵ گره A, B, C, D و E و ۷ یال

$$[A, B], \quad [B, C], \quad [C, D], \quad [D, E], \quad [A, E], \quad [C, E] \quad [A, C]$$

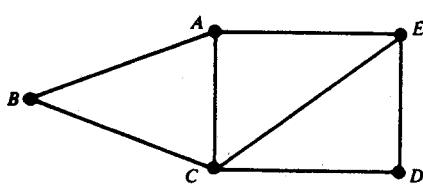
می باشد از B به E دو مسیر ساده به طول ۲ وجود دارد: (B, C, E) و (B, A, E) . از B به D تنها یک مسیر ساده به طول ۲ وجود دارد: (B, C, D) . یادآوری می کنیم که (B, A, D) مسیر نیست چون $[A, D]$ یال نیست. در این گراف دو ۴ دور یا ۴ حلقه وجود دارد:

$$[A, C, D, E, A] \quad \text{و} \quad [A, B, C, E, A]$$

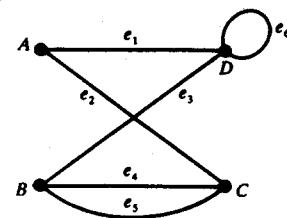
توجه دارید که $\deg(A) = 3$ ، چون A به ۳ یال تعلق دارد. بطور مشابه، $\deg(D) = 2$ و $\deg(C) = 4$

(ب) شکل ۸ - ۱ (ب) یک گراف نیست اما یک گراف چندگانه است چون یالهای چندگانه $[B, C] = e_4$ و $[B, C] = e_5$ دارد و یک حلقه $[D, D] = e_6$ دارد. تعریف یک گراف معمولاً یالهای چندگانه یا حلقه ها را مجاز نمی داند.

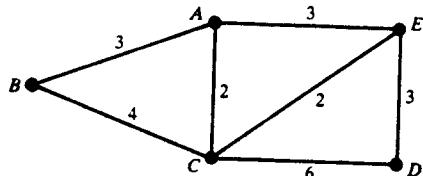
(ج) شکل ۱-۸ (ج) یک گراف درختی با $m = 6$ گره است و درنتیجه، $5 = m - 1 = 5$ یال دارد. دانشجو می‌تواند تحقیق کند که بین هر دو گره یک گراف درختی تنها یک مسیر ساده وجود دارد.



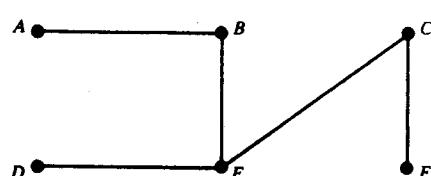
(ب)



(الف)



(د) گراف وزن داده شده



(ج)

شکل ۱-۸

(د) شکل ۱-۸ (د) همان گراف شکل ۱-۸ (الف) است با این تفاوت که اکنون گراف وزن داده شده، است. ملاحظه می‌کنید که $P_1 = (B, A, E, D)$ و $P_2 = (B, C, D)$ هر دو مسیرهایی از گره B به گره D هستند. اگرچه P_2 شامل یالهای بیشتری از P_1 است، اما وزن ۹ = $w(P_2) - w(P_1) = 10$ است.

گرافهای جهت دار

گراف جهت دار G که یک دایگراف DiGraph یا گراف نیز نامیده می‌شود همان گراف چندگانه است با این تفاوت که به هر یال e در G یک جهت نسبت داده می‌شود، به بیان دیگر، هر یال e به عوض زوج نامرتب $[u, v]$ ؛ یک زوج مرتب (u, v) از گره‌ها در G مشخص می‌شود.

فرض کنید G یک گراف جهت دار با یال جهت داده شده $(u, v) = e$ باشد. آنگاه e نیز یک کمان نامیده می‌شود. علاوه بر این، اصطلاحات زیر مورد استفاده قرار می‌گیرد:

(۱) e با u شروع می‌شود و با v پایان می‌یابد.

(۲) u مبدأ یا نقطه اولیه e و v مقصد یا نقطه انتهایی e نامیده می‌شود.

(۳) \forall گره قبلي \neq و \forall گره بعدی يا همسایه \neq نامیده می شود.

(۴) \forall مجاور \neq و همچنین \forall مجاور \neq است.

درجه خروجی گره \neq در G که به صورت $\text{outdeg}(\neq)$ ، نوشته می شود تعداد يال هایی است که با \neq شروع می شود. به همین ترتیب، درجه ورودی \neq که به صورت $\text{indeg}(\neq)$ نوشته می شود تعداد يال هایی است که با \neq به پایان می رستند. گره \neq یک گره منبع نامیده می شود اگر درجه خروجی مشبت داشته باشد و درجه ورودی اش صفر باشد. به همین ترتیب، گره \neq یک چاه نامیده می شود اگر درجه خروجی صفر و درجه ورودی مشبت داشته باشد.

مفهوم مسیر، مسیر ساده و دور یا حلقه از گرافهای بدون جهت، به گرافهای جهت دار قابل انتقال است با این تفاوت که اکنون جهت هر يال در یک مسیر (دور) باید با جهت مسیر (دور) سازگار باشد. گره \neq از گره \neq قابل دسترس نامیده می شود اگر از \neq به \neq یک مسیر (جهت دار) وجود داشته باشد.

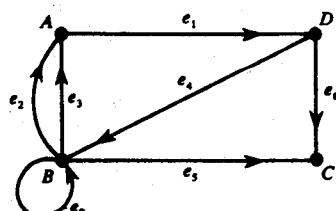
گراف جهت دار G را همبند یا همبند قوی گویند اگر برای هر زوج \neq ، \forall گره ها در G هم یک مسیر از \neq به \neq و هم یک مسیر از \neq به \neq وجود داشته باشد. از طرف دیگر، گراف G را همبند یکطرفه گویند اگر برای هر زوج \neq ، \forall گره ها در G ، یک مسیر از \neq به \neq یا یک مسیر از \neq به \neq وجود داشته باشد.

مثال ۸-۲

شکل ۸-۲ گراف جهت دار G با ۴ گره و ۷ يال (جهت دار) را نشان می دهد. يال های e_2 و e_3 را موازی گویند چون هر یک از این يالها با B شروع می شوند و با A به پایان می رستند. يال e_7 یک حلقه نامیده می شود چون نقطه شروع و پایان آن تنها یک نقطه B است.

دنیاله $P_1 = (D, C, B, A)$ مسیر نیست چون (C, B) یک يال نیست یعنی جهت يال $(C, B) = (B, C)$ با جهت مسیر P_1 سازگار نیست. از طرف دیگر، $P_2 = (D, B, A)$ یک مسیر از D به A است چون (B, A) دو يال هستند. بنابراین A از D قابل دسترس است. از C تا هر گره دیگر هیچ مسیری وجود ندارد، بنابراین G همبند قوی نیست. با وجود این، G همبند یکطرفه نامیده می شود. توجه دارید که $\text{outdeg}(C) = 2$ و $\text{indeg}(C) = 0$ اما $\text{outdeg}(D) = 1$ و $\text{indeg}(D) = 1$.

گره ای در G منبع نیست.



شکل ۸-۲

فرض کنید T گراف درختی غیرتهی باشد. فرض کنید گره دلخواه R در T انتخاب شده است. آنگاه T با گره خاص R ، یک درخت ریشه‌دار نامیده می‌شود و به R ریشه درخت می‌گویند. یادآور می‌شویم که یک مسیر ساده یکتا از ریشه R به هر گره دیگر در درخت وجود دارد. این امر یک جهت به یالهای درخت T می‌دهد، بنابراین درخت ریشه‌دار T را می‌توان به صورت یک گراف جهت‌دار مورد توجه قرار داد. علاوه بر این، فرض کنید گره‌های بعدی هر گره v در T نیز مرتب هستند. آنگاه T را یک درخت ریشه‌دار مرتب گویند. درختهای ریشه‌دار مرتب نکات قابل توجه بیشتری نسبت به درختهای عمومی که در فصل ۷ مورد بررسی قرار گرفتندارند.

گراف جهت‌دار G را ساده گویند هرگاه G هیچ یال موازی نداشته باشد. گراف ساده G می‌تواند حلقه داشته باشد، اما در یک گره معین نمی‌تواند بیش از یک حلقه داشته باشد. یک گراف بدون جهت G را می‌توان به صورت یک گراف جهت‌دار ساده مورد توجه قرار داد، مشروط بر این که هر یال $[u, v]$ در G دو یال جهت‌دار (v, u) و (u, v) را نمایش دهد. ملاحظه می‌کنید که ما از نماد $[v, u]$ استفاده می‌کنیم تا یک زوج نامرتب را نمایش دهیم و نماد (v, u) برای نمایش یک زوج مرتب بکار می‌رود. توجه کنید: موضوع اصلی مطالب این فصل را گراف‌های جهت‌دار ساده تشکیل می‌دهد. بنابراین، منظور از اصطلاح "گراف"، گراف جهت‌دار ساده است و منظور از اصطلاح "یال"، یال جهت‌دار است مگر آن که خلاف آن به صورت صریح یا ضمنی بیان شود.

۳-۸ نمایش ترتیبی گرافها، ماتریس مجاورت، ماتریس مسیر

دو روش استاندارد برای نگهداری گراف G در حافظه کامپیوتر وجود دارد. یک روش که نمایش ترتیبی G نامیده می‌شود به وسیله ماتریس مجاورت آن A انجام می‌شود. روش دیگر که نمایش پیوندی G نامیده می‌شود، به وسیله لیستهای پیوندی از گره‌های مجاور انجام می‌شود. این بخش نمایش اول را دربر دارد که چگونگی استفاده از ماتریس مجاورت A را نشان می‌دهد و به کمک آن می‌توان به سادگی به بسیاری از سوالات در رابطه با G جواب داد. نمایش پیوندی گراف G در بخش ۵-۸ گنجانده شده است. صرفنظر از روشی که برای نگهداری گراف G در حافظه کامپیوتر بکار می‌رود، معمولاً گراف G را با استفاده از تعریف رسمی آن که مجموعه‌ای از گره‌ها و مجموعه‌ای از یالها است به کامپیوتر وارد می‌کنند.

ماتریس مجاورت یا همسایگی

فرض کنید G یک گراف جهت‌دار ساده با m گره باشد، همچنین فرض کنید گره‌های G مرتب شده‌اند و $\{v_1, v_2, \dots, v_m\}$ نامیده می‌شوند. آنگاه ماتریس مجاورت (همسایگی) $A_{(m \times m)}$ گراف G ماتریس

ای است که به صورت زیر تعریف می‌شود:

اگر v_i مجاور v_j باشد یعنی اگر v_i با v_j وجود داشته باشد.
 $a_{ij} = \begin{cases} 1 & \text{در غیر این صورت} \\ 0 & \end{cases}$

به ماتریس A که درایه‌های آن تنها از 0 و 1 تشکیل می‌شود ماتریس بولین Boolean می‌گویند.

در ماتریس مجاورت A از گراف G ، ترتیب گره‌های G حائز اهمیت است یعنی ترتیب مختلف گره‌ها ممکن است منتهی به ماتریس مجاورت مختلف شود. با وجود این، ماتریسهای حاصل از دو ترتیب مختلف ارتباط بسیار نزدیکی با هم دارند، به این معنی که تنها با جابجایی سطرها و ستونها از یکی از ماتریسهای می‌توان ماتریس دیگر را بدست آورد. قرارداد می‌کنیم که گره‌های گراف G ، ترتیب ثابت و مشخص دارند مگر آن که خلاف آن بیان شود.

فرض کنید G یک گراف بدون جهت باشد. آنگاه ماتریس مجاورت A از گراف G یک ماتریس متقارن خواهد بود یعنی در این ماتریس بعازای تمام i و j ، $a_{ij} = a_{ji}$. این مطلب از این واقعیت ناشی می‌شود که هر یال بدون جهت $[v_i, v_j]$ متناظر با دو یال جهت دار (v_i, v_j) و (v_j, v_i) است.

نمایش ماتریسی بالا از یک گراف را می‌توان به گرافهای چندگانه تعمیم داد. بطور مشخص، اگر G یک گراف چندگانه باشد آنگاه ماتریس مجاورت G ماتریس $m \times m$ ای مانند $(a_{ij}) = A$ است که با قراردادن تعداد یالها از v_i به v_j به جای a_{ij} تعریف می‌شود.

مثال ۳-۸

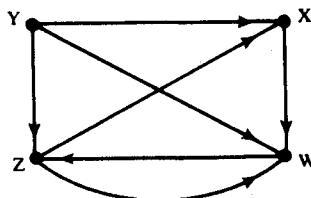
گراف G در شکل ۳-۸ را در نظر بگیرید. فرض کنید که گره‌ها در آرایه خطی DATA به صورت زیر در حافظه ذخیره شده‌اند.

DATA: X, Y, Z, W

آنگاه فرض می‌کنیم که ترتیب گره‌ها در G به قرار زیرند $v_4 = X$, $v_2 = Y$, $v_3 = Z$ و $v_1 = W$ و A ماتریس مجاورت گراف G چنین است :

$$A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

توجه دارید تعداد ۱ ها در A برابر تعداد یالهای گراف G است.



شکل ۳

توان‌های A , A^2 , A^3 ... از ماتریس مجاورت A گراف G را در نظر بگیرید. فرض کنید

$$a_{k(i,j)} = A^k$$

ملاحظه می‌کنید که $a_1(i,j) = a_1(j,i)$ تعداد مسیرهای به طول ۱ از گره i به گره j را به فست می‌دهد. می‌توان نشان داد که $a_2(j,i) = a_2(i,j)$ تعداد مسیرهای به طول ۲ از i به j است. در واقع در مسئله ۱۹-۸ نتیجه کلی زیر را ثابت می‌کنیم:

قضیه ۲-۸: فرض کنید A ماتریس مجاورت گراف G باشد. آنگاه $(j,i) \in a_k$ درایه $a_{k(i,j)}$ تابع ماتریس A به صورت مسیرهای از i به j را به دست می‌دهد که طول K دارد.

بار دیگر گراف G شکل ۳-۸ را در نظر بگیرید که ماتریس مجاورت A ای آن در مثال ۳-۸ داده شده است. توان‌های A^2 , A^3 و A^4 ماتریس A به شرح زیر است:

$$A^2 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 2 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} \quad A^3 = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 2 & 2 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \quad A^4 = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 2 & 0 & 2 & 3 \\ 1 & 0 & 1 & 2 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

بنابراین در حالت خاص، یک مسیر از v_4 به v_1 با طول ۲ وجود دارد، دو مسیر از v_2 به v_3 با طول ۳ وجود دارد و سه مسیر از v_2 به v_4 با طول ۴ طول وجود دارد. در اینجا $v_3 = Z$, $v_2 = Y$, $v_3 = X$, $v_2 = Y$, $v_4 = W$ و $v_1 = X$.

فرض کنید اکنون ماتریس B_r را به صورت زیر تعریف کرده‌ایم:

$$B_r = A + A^2 + A^3 + \dots + A^r$$

آنگاه درایه z_{ij} ام ماتریس B_r تعداد مسیرهای به طول r یا کمتر از r را از گره i به j محاسبه می‌کند.

ماتریس مسیر

فرض کنید G یک گراف جهت دار ساده با گره v_1, v_2, \dots, v_m باشد. ماتریس مسیر یا ماتریس دسترسی گراف G ماتریس m مرتبه (P_i) است که به صورت زیر تعریف می‌شود:

اگر یک مسیر از v_i به v_j وجود داشته باشد
 $P_{ij} = \begin{cases} 1 & \text{در غیر اینصورت} \\ 0 & \end{cases}$

فرض کنید یک مسیر از v_i به v_j وجود دارد. آنگاه باید یک مسیر از v_i به v_j وجود داشته باشد که $v_i \neq v_j$ یا باید یک دور از v_i به v_j وجود داشته باشد که $v_i = v_j$. چون G تنها گره دارد. چنین مسیر ساده‌ای باید طول ۱ - m یا کمتر داشته باشد یا چنین دوری باید طول m یا کمتر داشته باشد. بدین‌دیگر یک درایه لا غیر صفر در ماتریس B_m وجود داشته باشد که در پایان بخش قبل تعریف شده است. بنابراین، بین ماتریس مسیر P و ماتریس مجاورت A رابطه زیر برقرار است:

قضیه ۳-۸: فرض کنید A ماتریس مجاورت و $(p_{ij}) = P$ ماتریس مسیر دایگراف G باشد. آنگاه $1 = p_{ij}$ اگر و تنها اگر یک عدد غیر صفر در درایه l_{ij} از ماتریس وجود داشته باشد:

$$B_m = A + A^2 + A^3 + \dots + A^m$$

گراف G را با $m = 4$ گره در شکل ۳-۸ در نظر بگیرید. با جمع ماتریسهای A و A^2 , A^3 , A^4 , ماتریس B_4 زیر بدست می‌آید و اگر به جای درایه‌های غیر صفر B_4 عدد ۱ را قرار دهیم ماتریس مسیر P گراف G بدست می‌آید:

$$B_4 = \begin{pmatrix} 1 & 0 & 2 & 3 \\ 5 & 0 & 6 & 8 \\ 3 & 0 & 3 & 5 \\ 2 & 0 & 3 & 3 \end{pmatrix} \quad \text{و} \quad P = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

با مطالعه و بررسی ماتریس P , ملاحظه می‌کنید که به گره v_2 از طریق هیچ گره دیگری نمی‌توان دسترسی پیدا کرد.

یادآوری می‌کنیم که گراف جهت دار G همبند قوی است اگر برای هر زوج گره v_i و v_j در G , هم یک مسیر از v_i به v_j و هم یک مسیر از v_j به v_i وجود داشته باشد. بنابراین G همبند قوی است اگر و فقط اگر ماتریس مسیر P از درایه غیر صفر داشته باشد. بدین ترتیب گراف G در شکل ۳-۸ متصصل قوی نیست. بستگی تراکذی گراف G بناهه تعریف گراف G' است به طوری که G' دارای همان گره‌های گراف G بوده و یک یال (v_i, v_j) در G' وجود داشته باشد هرگاه در G یک مسیر از v_i به v_j موجود باشد. بنابراین ماتریس مسیر P از گراف G دقیقاً ماتریس مجاورت بستگی تراکذی آن G' است. علاوه بر این گراف G همبند قوی است اگر و فقط بستگی تراکذی آن یک گراف کامل باشد.

توجه کنید: ماتریس مجاورت A و ماتریس مسیر P از گراف G را می‌توان به صورت ماتریسهای منطقی یا بولین مورد بررسی قرار داد که در آن ۰ نمایشگر False و ۱ نمایشگر True است. بنابراین، عملیات منطقی \wedge (AND) و \vee (OR) را می‌توان بر درایه‌های ماتریس A و P اعمال کرد. مقادیر \wedge و \vee در شکل

۴-۸. ظاهر شده است. از این عملیات در بخش بعد استفاده خواهد شد.

v	0	1
0	0	1
1	1	1

OR.

Λ	0	1
0	0	0
1	0	1

AND.

(ب)

(الف)

شکل ۴-۸

۴-۸. الگوریتم وارشال، کوتاهترین مسیر

فرض کنید G یک گراف جهت دار با m گره v_1, v_2, \dots, v_m باشد. فرض کنید بخواهیم ماتریس مسیر P از گراف G را پیدا کنیم. برای این منظور، وارشال الگوریتمی ارائه داد که بسیار کاربر و مؤثرتر از محاسبه توانهای مختلف ماتریس مجاورت A و استفاده از قضیه ۳-۸ است. الگوریتم وارشال در این بخش توضیح داده می‌شود و وقتی گراف G وزن داده شده باشد از الگوریتم مشابهی برای تعیین کوتاهترین مسیر در G استفاده می‌شود.

نخست ماتریسهای بولین m مربعی P_0, P_1, \dots, P_m را به صورت زیر تعریف می‌کنیم. فرض کنید $P_k[i, j]$ نمایش درایه i ام ماتریس P_k باشد. بدین ترتیب تعریف می‌کنیم:

$$P_k[i, j] = \begin{cases} 1 & \text{اگر یک مسیر ساده از } v_i \text{ به } v_j \text{ وجود داشته باشد که از هیچ گره} \\ & \text{دیگر بجز احتمالاً } v_1, v_2, \dots, v_k \text{ استفاده نمی‌کند.} \\ 0 & \text{در غیر اینصورت} \end{cases}$$

به عبارت دیگر:

اگر یک یال از v_i به v_j وجود داشته باشد.

اگر یک مسیر ساده از v_i به v_j وجود داشته باشد که از هیچ گره دیگر بجز احتمالاً v_1 استفاده نمی‌کند.

اگر یک مسیر ساده از v_i به v_j وجود داشته باشد که از هیچ گره دیگر بجز احتمالاً v_1 و v_2 استفاده نمی‌کند.

نخست ملاحظه کنید که ماتریس $A = P_0$ ، ماتریس مجاورت G است. علاوه بر این، چون G تنها گره دارد، ماتریس آخر $P_m = P_k$ ماتریس مسیر G است. وارشال ملاحظه کرد که می‌تواند اتفاق بیفتد که یکی از دو حالت زیر روی دهد:

- (۱) یک مسیر ساده از v_i به v_j وجود دارد که در آن از هیچ گره دیگر بجز احتمالاً $v_1, v_2, \dots, v_{k-1}, v_k$ استفاده نمی‌کند. از این رو

$$P_{k+1}[i, j] = 1$$

- (۲) یک مسیر ساده از v_i به v_k و یک مسیر ساده از v_k به v_j وجود دارد که در آن از هیچ گره دیگر بجز احتمالاً $v_1, v_2, \dots, v_{k-1}, v_k$ استفاده نمی‌کند. از این رو

$$P_{k+1}[k, j] = 1 \quad P_{k+1}[i, k] = 1$$

این دو حالت به ترتیب در شکل ۸-(الف) و (ب) به تصویر درآمده است که در آن

→ → → → →

نمایش قسمتی از مسیر ساده است که در آن از هیچ گره دیگر بجز احتمالاً $v_1, v_2, \dots, v_{k-1}, v_k$ استفاده نمی‌کند.

$v_i \rightarrow \dots \rightarrow v_k \rightarrow \dots \rightarrow v_j$

(ب)

$v_i \rightarrow \dots \rightarrow v_j$

(الف)

شکل ۸-۵

بنابراین، عناصر ماتریس P_k را می‌توان از

$$P_k[i, j] = P_{k+1}[i, j] \vee (P_{k+1}[i, k] \wedge P_{k+1}[k, j])$$

به دست آورد که در آن از عملگرهای منطقی (AND و OR) استفاده می‌کنیم. به بیان دیگر، هر درایه ماتریس P_k را می‌توان تنها با نگاه سریع به سه درایه ماتریس P_0 به دست آورد. الگوریتم وارشال به شرح زیر است:

Algorithm 8.1: (Warshall's Algorithm) A directed graph G with M nodes is maintained in memory by its adjacency matrix A . This algorithm finds the (Boolean) path matrix P of the graph G .

1. Repeat for $I, J = 1, 2, \dots, M$: [Initializes P .]
 - If $A[I, J] = 0$, then: Set $P[I, J] := 0$;
 - Else: Set $P[I, J] := 1$.

[End of loop.]
2. Repeat Steps 3 and 4 for $K = 1, 2, \dots, M$: [Updates P .]
3. Repeat Step 4 for $I = 1, 2, \dots, M$:
4. Repeat for $J = 1, 2, \dots, M$:
 - Set $P[I, J] := P[I, J] \vee (P[I, K] \wedge P[K, J])$.

[End of loop.]

[End of Step 3 loop.]

[End of Step 2 loop.]
5. Exit.

الگوریتم کوتاهترین مسیر

فرض کنید G یک گراف جهت دار با m گره v_1, v_2, \dots, v_m ، فرض کنید G وزن داده شده باشد. به عبارت دیگر فرض کنید به هر یال e از G یک عدد غیر منفی ($w(e)$) نسبت داده می‌شود که وزن یا طول یال e نامیده می‌شود. آنگاه G را می‌توان به وسیله ماتریس وزن آن ($w = W$) که به صورت زیر تعریف می‌شود در حافظه نگهداری کرد.

$$w_{ij} = \begin{cases} w(e) & \text{اگر } v_i \text{ به } v_j \text{ یک یال } e \text{ وجود داشته باشد.} \\ 0 & \text{اگر هیچ یالی از } v_i \text{ به } v_j \text{ وجود نداشته باشد.} \end{cases}$$

ماتریس مسیر P به ما می‌گوید آیا مسیرهایی بین گره‌ها وجود دارد یا خیر. اکنون مایلیم ماتریس Q را پیدا کنیم که به ما طول‌های کوتاهترین مسیرها را بین گره‌ها می‌دهد یا به بیان دیگر ماتریس ($q_{ij} = Q$) می‌دهد که در آن

$$q_{ij} = \text{طول کوتاهترین مسیر از } v_i \text{ به } v_j$$

بعد تغییری در الگوریتم وارشاو ایجاد می‌کنیم که ماتریس Q را پیدا می‌کند.

در اینجا دنباله‌ای از ماتریسهای $Q_0, Q_1, Q_2, \dots, Q_m$ را تعریف می‌کنیم (مشابه ماتریسهای P_0, P_1, \dots, P_m) که درایه‌های آن به صورت زیر تعریف می‌شود:

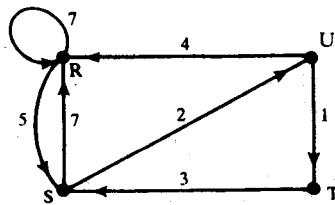
$$Q_K[i, j] = \text{کوچکترین طول مسیر قبل از } v_i \text{ به } v_j \text{ یا مجموع طول‌های}$$

مسیرهای قبلی از v_k به v_k و از v_k به v_j

به بیان دقیق‌تر:

$$Q_k[i, j] = \min(Q_{k-1}[i, j], Q_k[i, k] + Q_k[k, j])$$

ماتریس اولیه Q_0 ماتریس وزن W است با این تفاوت که در آن به جای بینهایت یا ∞ (یا هر عدد بسیار بزرگ) عدد صفر جایگزین می‌شود. ماتریس نهایی Q_m ماتریس موردنظر Q خواهد بود.



شکل ۸-۶

مثال ۴

گراف وزن داده شده G در شکل ۸-۶ را در نظر بگیرید. فرض کنید $v_1 = R, v_2 = S, v_3 = T$

۷۴ . آنگاه W ماتریس وزن G به صورت زیر است :

$$W = \begin{pmatrix} 7 & 5 & 0 & 0 \\ 7 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 1 & 0 \end{pmatrix}$$

با اعمال تغییر در الگوریتم وارشاو، ماتریسهای Q_0 , Q_1 , Q_2 , Q_3 و Q_4 به دست می‌آید. در طرف راست هر ماتریس Q_k ، ماتریس مسیرها را نشان می‌دهیم که متناظر با طول‌های ماتریس Q_k است.

$$Q_0 = \begin{pmatrix} 7 & 5 & \infty & \infty \\ 7 & \infty & \infty & 2 \\ \infty & 3 & \infty & \infty \\ 4 & \infty & 1 & \infty \end{pmatrix} \quad \begin{array}{l|lll} RR & RS & - & - \\ SR & - & - & SU \\ - & TS & - & - \\ UR & - & UT & - \end{array}$$

$$Q_1 = \begin{pmatrix} 7 & 5 & \infty & \infty \\ 7 & 12 & \infty & 2 \\ \infty & 3 & \infty & \infty \\ 4 & 9 & 1 & \infty \end{pmatrix} \quad \begin{array}{l|lll} RR & RS & - & - \\ SR & SRS & - & SU \\ - & TS & - & - \\ UR & URS & UT & - \end{array}$$

$$Q_2 = \begin{pmatrix} 7 & 5 & \textcircled{8} & 7 \\ 7 & 12 & \infty & 2 \\ 10 & 3 & \infty & 5 \\ 4 & 9 & 1 & 11 \end{pmatrix} \quad \begin{array}{l|lll} RR & RS & - & RSU \\ SR & SRS & - & SU \\ TSR & TS & - & TSU \\ UR & URS & UT & URS \end{array}$$

$$Q_3 = \begin{pmatrix} 7 & 5 & \infty & 7 \\ 7 & 12 & \infty & 2 \\ 10 & 3 & \infty & 5 \\ 4 & 9 & 1 & 6 \end{pmatrix} \quad \begin{array}{l|lll} RR & RS & - & RSU \\ SR & SRS & - & SU \\ TSR & TS & - & TSU \\ UR & UTS & UT & UTSU \end{array}$$

$$Q_4 = \begin{pmatrix} 7 & 5 & 8 & 7 \\ 7 & 11 & 3 & 2 \\ \textcircled{9} & 3 & 6 & 5 \\ 4 & 4 & 1 & 6 \end{pmatrix} \quad \begin{array}{l|lll} RR & RS & RSUT & RSU \\ SR & SURS & SUT & SU \\ TSUR & TS & TSUT & TSU \\ UR & UTS & UT & UTSU \end{array}$$

توضیح می‌دهیم که چگونه درایه‌های داخل دایره به دست آمده‌اند :

$$Q_1[4, 2] = \text{MIN}(Q_0[4, 2], Q_0[4, 1] + Q_0[1, 2]) = \text{MIN}(\infty, 4 + 5) = 9$$

$$Q_2[1, 3] = \text{MIN}(Q_1[1, 3], Q_1[1, 2] + Q_1[2, 3]) = \text{MIN}(\infty, 5 + \infty) = \infty$$

$$Q_3[4, 2] = \text{MIN}(Q_2[4, 2], Q_2[4, 3] + Q_2[3, 2]) = \text{MIN}(9, 3 + 1) = 4$$

$$Q_4[3, 1] = \text{MIN}(Q_3[3, 1], Q_3[3, 4] + Q_3[4, 1]) = \text{MIN}(10, 5 + 4) = 9$$

بیان رسمی الگوریتم به شرح زیر است :

Algorithm 8.2: (Shortest-Path Algorithm) A weighted graph G with M nodes is maintained in memory by its weight matrix W . This algorithm finds a matrix Q such that $Q[I, J]$ is the length of a shortest path from node V_i to node V_j . INFINITY is a very large number, and MIN is the minimum value function.

1. Repeat for $I, J = 1, 2, \dots, M$: [Initializes Q .]
 - W[I, J] = 0, then: Set $Q[I, J] := \text{INFINITY}$;
 - Else: Set $Q[I, J] := W[I, J]$.

[End of loop.]
2. Repeat Steps 3 and 4 for $K = 1, 2, \dots, M$: [Updates Q .]
3. Repeat Step 4 for $I = 1, 2, \dots, M$:
4. Repeat for $J = 1, 2, \dots, M$:
 - Set $Q[I, J] := \text{MIN}(Q[I, J], Q[I, K] + Q[K, J])$.

[End of loop.]

[End of Step 3 loop.]

[End of Step 2 loop.]
5. Exit.

به شباهت بین الگوریتم 8.1 و الگوریتم 8.2 توجه کنید.

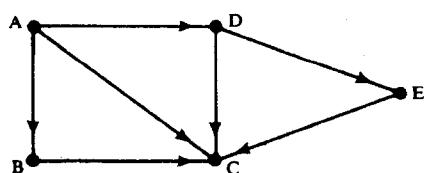
الگوریتم 8.2 را نیز می‌توان برای گراف G ای بکار گرفت که بدون وزن است و به هر یال e در G تنها وزن $w(e)$ نسبت داده شده است.

۸-۵ نمایش یک گراف با استفاده از لیست پیوندی

فرض کنید G یک گراف جهت دار با m گره باشد. نمایش ترتیبی G در حافظه یعنی نمایش G به صورت ماتریس مجاورت A آن، دارای چند اشکال عمده است. قبل از هر چیز، باید بگوئیم که اضافه و حذف گره‌ها با این نمایش در G مشکل است چون به علت قابل تغییر بودن اندازه A ، گره‌ها را الزاماً باید از نو مرتب کرد، از این‌رو در ماتریس A تغییرات بسیار زیادی انجام می‌شود. علاوه بر این اگر تعداد یال‌ها $O(m \log_2 m)$ باشد، آنگاه ماتریس A ، خلوت یا تُنک خواهد بود چون دارای صفرهای بسیار زیادی است. از این‌رو مقدار زیادی از حافظه به هدر می‌رود. بنابراین، G را معمولاً در حافظه به صورت نمایش پیوندی نمایش می‌دهد که ساختار مجاورتی نیز نامیده می‌شود که در بخش حاضر توضیح داده می‌شود. گراف G شکل ۸-۷ (الف) را در نظر بگیرید:

Node	Adjacency List
A	B, C, D
B	C
C	
D	C, E
E	C

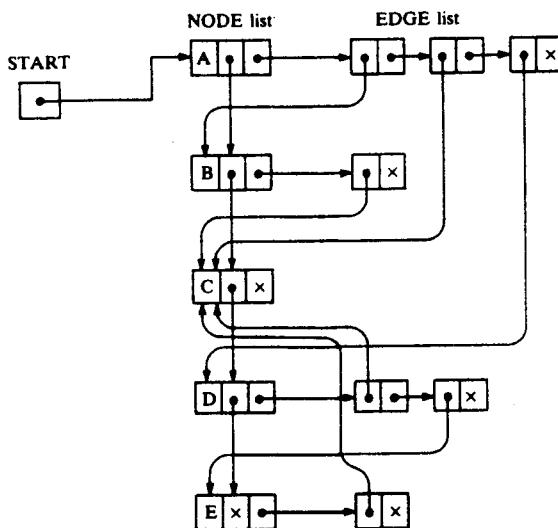
(ب) لیست‌های مجاورت G



(الف) گراف G

شکل ۸-۷

جدول شکل ۸-۷ (ب) هر گره داخل G را بعد از لیست مجاورتی آن نشان می‌دهد که لیست گره‌های مجاور آن است و گره‌های بعدی یا گره‌های همسایه نیز نامیده می‌شود. شکل ۸-۸ نمودار نمایش پیوندی G را در حافظه نشان می‌دهد.



شکل ۸-۸

روشن است که نمایش پیوندی از دو لیست (فایل) یعنی لیست گره NODE و لیست یال EDGE به شرح زیر تشکیل شده است :

(الف) لیست گره : هر عنصر در لیست NODE متناظر با یک گره در گراف G است و یک رکورد به شکل زیر است :

NODE	NEXT	ADJ	
------	------	-----	--

در اینجا NODE نام یا مقدار کلیدی گره است و NEXT اشاره گر به گره بعدی در لیست NODE و ADJ اشاره گر به عنصر اول در لیست مجاورتی گره است که در لیست EDGE نگهداری می‌شود. منطقه سایزده شده بیان می‌کند که اطلاعات دیگر در رکورد نظیر درجه ورودی INDEG گره، درجه خروجی OUTDEG گره، در طی اجرای الگوریتم وغیره می‌تواند وجود داشته باشد. (به بیان دیگر می‌توان فرض کرد که NODE آرایه‌ای از رکوردها شامل فیلد‌هایی نظیر NAME ، OUTDEG ، INDEG ، STATUS ، ... است). خود گره‌ها که در شکل ۸-۸ رسم شده است به صورت یک لیست پیوندی

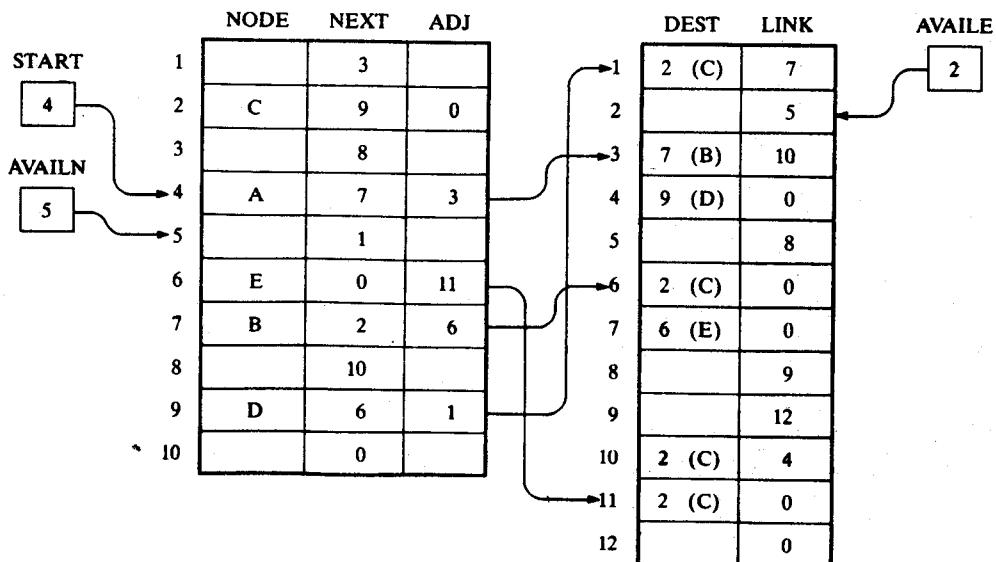
سازماندهی خواهد شد و از این رو یک متغیر اشاره‌گر START برای شروع لیست و یک متغیر اشاره‌گر AVAILN برای لیست حافظه موجود، خواهد داشت. گاهی اوقات بسته به کاربرد، گره‌ها را می‌توان به صورت آرایه مرتب شده یا یک درخت جستجوی دودویی بجای یک لیست پیوندی سازماندهی کرد.

(ب) لیست یال . هر عنصر در لیست EDGE متناظر با یال G است و یک رکورد به صورت زیر است :

DEST	LINK
------	------

فیلد DEST به مکان لیست NODE گرده مقصود یا گره انتهایی یال اشاره می‌کند. فیلد LINK یالهایی با گره اولیه یکسان یعنی گره‌هایی با لیست مجاورتی یکسان را به هم پیوند می‌دهند. منطقه سایه‌زده شده بیان می‌کند که اطلاعات دیگری در رکورد متناظر با یال است نظیر فیلد EDGE که شامل اطلاعات یال شماره‌دار است و این در حالتی است که G یک گراف دارای شماره باشد، یک فیلد WEIGHT که شامل وزن یال است و این در حالتی است که G یک گراف وزن داده شده باشد و الى آخر، علاوه براین برای لیست فضای حافظه موجود در لیست AVAILE به یک متغیر اشاره گر EDGE نیاز داریم.

شکل ۸-۹ چگونگی ظاهرشدن گراف G شکل ۸-۷ (الف) را در حافظه نشان می‌دهد. انتخاب ۱۰ مکان برای لیست NODE و ۱۲ امکان برای لیست EDGE اختیاری و دلخواه است.



شکل ۸-۹

نمایش پیوندی گراف G که مورد بحث و بررسی قرار می‌گیرد را می‌توان با

GRAPH(NODE, NEXT, ADJ, START, AVAILN, DEST, LINK, AVAILE)

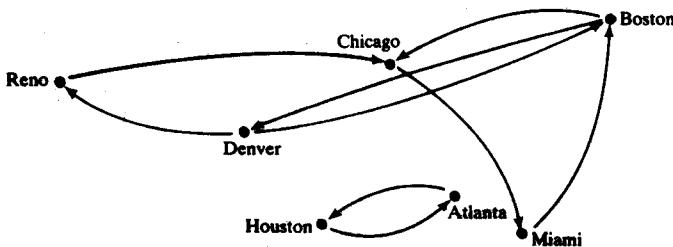
نمایش داد. علاوه بر این وقتی G یک گراف وزن داده شده است، این نمایش می‌تواند شامل آرایه

WEIGHT باشد یا وقتی G یک گراف دارای شماره است می‌تواند شامل آرایه EDGE باشد.

مثال ۸-۵

فرض کنید شرکت هوایپمایی Friendly Airlines در یک روز، نه پرواز به شرح زیر دارد:

103	Atlanta to Houston	203	Boston to Denver	305	Chicago to Miami
106	Houston to Atlanta	204	Denver to Boston	308	Miami to Boston
201	Boston to Chicago	301	Denver to Reno	402	Reno to Chicago



شکل ۸-۱۰

واضح است که اطلاعات را می‌توان به صورت کارایی در یک فایل ذخیره کرد که در آن فایل هر رکورد از سه فیلد زیر تشکیل شده است:

Flight Number,	City of Origin,	City of Destination
----------------	-----------------	---------------------

با وجود این، چنین نمایشی به سادگی، به سوالات طبیعی زیر جواب نمی‌دهد:

(الف) آیا از شهر X به شهر Y یک پرواز مستقیم وجود دارد؟

(ب) آیا یک پرواز با تعداد توقف‌های ممکن از شهر X به شهر Y می‌تواند انجام شود؟

(ج) مستقیم‌ترین مسیر کدام مسیر است یعنی مسیری که کمترین تعداد توقف ممکن از شهر X به شهر Y را دارد کدام است؟

برای دادن پاسخ‌های گویاگزین به این سوالات بهتر است اطلاعات به صورت یک گراف G با شهرها به عنوان گره‌ها و پروازها به عنوان یال‌ها سازماندهی شوند. شکل ۸-۱۰ بالا تصویری از گراف G است.

شکل ۸-۱۱ چگونگی ظاهر شدن گراف G را در حافظه با استفاده از نمایش پیوندی نشان می‌دهد.

NODE list			EDGE list			
CITY	NEXT	ADJ	NUMBER	ORIG	DEST	LINK
1	0		1	2	4	0
2	Atlanta	12	1	2	106	2
3	Chicago	11	7	3	201	12
4	Houston	7	2	4	203	11
5		6		5	204	11
6		8		6	301	10
7	Miami	10	8	7	305	7
8		9		8	308	12
9		1		9	402	3
10	Reno	0	9	10		
11	Denver	4	5	11		
12	Boston	3	3	12		0

START = 2, AVAILN = 5

AVAILE = 10

شکل ۸-۱۱

مذکور می‌شویم که G یک گراف شماره‌دار است نه یک گراف وزن داده شده، چون شماره پرواز تنها جهت شناسایی است. حتی اگر اطلاعات به صورت یک گراف سازماندهی شده باشند، با وجود این همچنان نیازمند یک نوع الگوریتم برای پاسخ به سوالات (ب) و (ج) هستیم. چنین الگوریتم‌هایی دیرتر در همین فصل مورد توجه قرار می‌گیرد.

۶-۸ عملیات بر روی گراف‌ها

فرض کنید گراف G به وسیله نمایش پیوندی در حافظه نگهداری می‌شود:

GRAPH(NODE, NEXT, ADJ, START, AVAILN, DEST, LINK, AVAILE)

که در بخش قبل مورد بحث و بررسی قرار گرفت. این بخش عملیات جستجو، اضافه کردن و حذف گره‌ها و یالهای گراف G را شرح می‌دهد. عمل پیمایش در بخش بعد بررسی می‌شود.

عملیات این بخش از بعضی از زیر برنامه‌های **Procedure** فصل ۵ در رابطه با لیستهای پیوندی استفاده می‌کند. جهت کامل شدن بحث، این زیر برنامه‌ها در زیر با اندکی تغییر روش نسبت به فصل ۵

بیان می‌شود. طبیعی است که اگر به جای یک لیست پیوندی از لیست پیوندی چرخشی یا حلقوی، یا یک درخت جستجوی دودویی استفاده شود، آنگاه از زیربرنامه‌های مشابهی باید استفاده کرد.

زیربرنامه 8.3 (با نام اولیه **Algorithm 5.2**) مکان LOC عنصر ITEM را در یک لیست پیوندی پیدا می‌کند.

Procedure 8.3: FIND(INFO, LINK START, ITEM, LOC) [Algorithm 5.2]
Finds the location LOC of the first node containing ITEM, or sets LOC := NULL.

1. Set PTR := START.
2. Repeat while PTR ≠ NULL:
 - If ITEM = INFO[PTR], then: Set LOC := PTR, and Return.
 - Else: Set PTR := LINK[PTR].

[End of loop.]
3. Set LOC := NULL, and Return.

زیربرنامه 8.4 (با نام اولیه **Algorithm 5.9** و **Procedure 5.10**) عنصر داده شده ITEM را از یک لیست پیوندی حذف می‌کند. در اینجا از یک متغیر منطقی FLAG استفاده می‌کنیم تا بینیم آیا ITEM از آغاز در لیست پیوندی وجود دارد یا خیر.

Procedure 8.4: DELETE(INFO, LINK, START, AVAIL, ITEM, FLAG) [Algorithm 5.10]
Deletes the first node in the list containing ITEM, or sets FLAG := FALSE when ITEM does not appear in the list.

1. [List empty?] If START = NULL, then: Set FLAG := FALSE, and Return.
2. [ITEM in first node?] If INFO[START] = ITEM, then:
 - Set PTR := START, START := LINK[START],
 - LINK[PTR] := AVAIL, AVAIL := PTR,
 - FLAG := TRUE, and Return.

[End of If structure.]
3. Set PTR := LINK[START] and SAVE := START. [Initializes pointers.]
4. Repeat Steps 5 and 6 while PTR ≠ NULL:
 5. If INFO[PTR] = ITEM, then:
 - Set LINK[SAVE] := LINK[PTR], LINK[PTR] := AVAIL,
 - AVAIL := PTR, FLAG := TRUE, and Return.

[End of If structure.]
 6. Set SAVE := PTR and PTR := LINK[PTR]. [Updates pointers]

[End of Step 4 loop.]
7. Set FLAG := FALSE, and Return.

جستجوی گره در یک گراف

فرض کنید بخواهیم مکان LOC یک گره N را در گراف G پیدا کنیم. این کار را می‌توان با استفاده از

به شرح زیر انجام داد:

Call FIND(NODE, NEXT, START, N, LOC)

یعنی این دستور Call لیست NODE را برای پیدا کردن گره N جستجو می‌کند. از طرف دیگر، می‌خواهیم مکان LOC یا (A, B) را در گراف G پیدا کنیم. نخست باید مکان LOCA گره A و مکان LOCB گره B را در لیست NODE پیدا کنیم. آنگاه باید در لیست گره‌های بعدی A مکان Procedure 8.5 LOCB پیدا کنیم که اشاره‌گر لیست [ADJ[LOCA]] دارد. این کار با زیربرنامه LOC پیاده‌سازی می‌شود که علاوه بر این تحقیق می‌کند آیا A و B گره‌هایی در گراف G هستند یا خیر. ملاحظه می‌کنید که LOC مکان LOCB را در لیست EDGE به دست می‌دهد.

اضافه کردن گره در یک گراف

فرض کنید بخواهیم گره N را به گراف G اضافه کنیم. توجه دارید که N در [NODE[AVAILN]] که اولین گره آزاد است جایگزین می‌شود. علاوه بر این چون N یک گره منفرد خواهد بود علاوه بر این باید قرار دهیم **ADJ[NODE] := NULL**. این کار را با استفاده از یک متغیر منطقی FLAG برای بیان حالت سرریزی OverFlow انجام می‌دهد. واضح است که اگر لیست NODE به صورت یک لیست مرتب شده نگهداری شود یا به صورت یک درخت جستجوی دودویی باشد، باید زیربرنامه 8.6 اصلاح شود.

Procedure 8.5: FINDEdge(NODE, NEXT, ADJ, START, DEST, LINK, A, B, LOC)
This procedure finds the location LOC of an edge (A, B) in the graph G, or sets LOC := NULL.

1. Call FIND(NODE, NEXT, START, A, LOCA).
2. CALL FIND(NODE, NEXT, START, B, LOCB).
3. If LOCA = NULL or LOCB = NULL, then: Set LOC := NULL.
Else: Call FIND(DEST, LINK, ADJ[LOCA], LOCB, LOC).
4. Return.

Procedure 8.6: INSNODE(NODE, NEXT, ADJ, START, AVAILN, N, FLAG)
This procedure inserts the node N in the graph G.

1. [OVERFLOW?] If AVAILN = NULL, then: Set FLAG := FALSE, and Return.
2. Set ADJ[AVAILN] := NULL.
3. [Removes node from AVAILN list.]
Set NEW := AVAILN and AVAILN := NEXT[AVAILN].
4. [Inserts node N in the NODE list.]
Set NODE[NEW] := N, NEXT[NEW] := START and START := NEW.
5. Set FLAG := TRUE, and Return.

فرض کنید بخواهیم یال (A, B) را به گراف G اضافه کنیم. این زیربرنامه فرض می‌کند که هم A و هم B قبلاً گرهایی در G هستند. این زیربرنامه نخست $LOCA$ مکان A و $LOCB$ مکان B را در لیست گره پیدا می‌کند. آنگاه (A, B) به عنوان یک یال G با اضافه کردن $LOCB$ در لیست گرهای بعدی A اضافه می‌شود، که اشاره گر لیست $ADJ[LOCA]$ دارد. بار دیگر یک متغیر منطقی $FLAG$ برای بیان حالت سرریزی OverFlow مورد استفاده قرار می‌گیرد. این زیربرنامه به شرح زیر است:

Procedure 8.7: INSEdge(NODE, NEXT, ADJ, START, DEST, LINK, AVAILE, A, B, FLAG)

This procedure inserts the edge (A, B) in the graph G .

1. Call FIND(NODE, NEXT, START, A, LOCA).
2. Call FIND(NODE, NEXT, START, B, LOCB).
3. [OVERFLOW?] If AVAILE = NULL, then: Set FLAG := FALSE, and Return.
4. [Remove node from AVAILE list.] Set NEW := AVAILE and AVAILE := LINK[AVAILE].
5. [Insert LOCB in list of successors of A.] Set DEST[NEW] := LOCB, LINK[NEW] := ADJ[LOCA] and ADJ[LOCA] := NEW.
6. Set FLAG := TRUE, and Return.

اگر A یا B یک گره در گراف G نباشد با استفاده از Procedure 8.6 باید زیربرنامه اصلاح شود.

حذف کردن یک گره از یک گراف

فرض کنید بخواهیم یال (A, B) را از گراف G حذف کنیم. این زیربرنامه فرض می‌کند که هم A و هم B گرهایی در گراف G هستند. بار دیگر باید نخست $LOCA$ مکان A و $LOCB$ مکان B را در لیست گره پیدا کنیم. آنگاه تنها $LOCB$ را از لیست گرهای بعدی A حذف می‌کنیم که اشاره گر لیست $ADJ[LOCA]$ دارد. متغیر منطقی $FLAG$ برای بیان این مطلب که هیچ یالی در گراف G وجود ندارد مورد استفاده قرار می‌گیرد. این زیربرنامه به شرح زیر است:

Procedure 8.8: DELEDGE(NODE, NEXT, ADJ, START, DEST, LINK, AVAILE, A, B, FLAG)

This procedure deletes the edge (A, B) from the graph G .

1. Call FIND(NODE, NEXT, START, A, LOCA). [Locates node A.]
2. Call FIND(NODE, NEXT, START, B, LOCB). [Locates node B.]
3. Call DELETE(DEST, LINK, ADJ[LOCA], AVAILE, LOCB, FLAG). [Uses Procedure 8.4.]
4. Return.

فرض کنید بخواهیم گره N را از گراف G حذف کنیم. این عمل بسیار پیچیده‌تر از عملیات جستجو و اضافه کردن و حذف یک یال است، چون باید تمام یال‌هایی که شامل N است نیز حذف شود. توجه کنید که این یال‌ها دو حالت مختلف دارند، یال‌هایی که با N شروع می‌شود و یال‌هایی که با N به پایان می‌رسند.

بنابراین، زیر برنامه اساساً شامل چهار مرحلهٔ زیر است:

(۱) مکان LOC گره N در G را پیدا کند.

(۲) تمام یال‌هایی را که با N به پایان می‌رسند حذف کند یعنی LOC را از لیست گره‌های بعدی هر گره M در G حذف کند. این مرحله نیازمند پیمایش لیست گره G است.

(۳) تمام یال‌هایی را که با N شروع می‌شوند حذف کند. این کار با پیدا کردن مکان BEG اولین گره بعدی و مکان END گره بعدی N انجام می‌شود و آنگاه لیست گره بعدی N را به لیست AVAILE آزاد اضافه کند.

(۴) خود N را از لیست NODE حذف کند.

این زیر برنامه به شرح زیر است:

Procedure 8.9: DELNODE(NODE, NEXT, ADJ, START, AVAILN, DEST, LINK, AVAILE, N, FLAG)

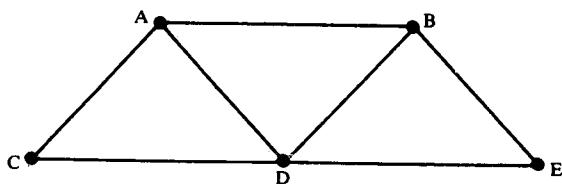
This procedure deletes the node N from the graph G.

1. Call FIND(NODE, NEXT, START, N, LOC). [Locates node N.]
2. If LOC = NULL, then: Set FLAG := FALSE, and Return.
3. [Delete edges ending at N.]
 - (a) Set PTR := START.
 - (b) Repeat while PTR ≠ NULL:
 - (i) Call DELETE(DEST, LINK, ADJ[PTR], AVAILE, LOC, FLAG).
 - (ii) Set PTR := NEXT[PTR].
- [End of loop.]
4. [Successor list empty?] If ADJ[LOC] = NULL, then: Go to Step 7.
5. [Find the first and last successor of N.]
 - (a) Set BEG := ADJ[LOC], END := ADJ[LOC] and PTR := LINK[END].
 - (b) Repeat while PTR ≠ NULL:
 - Set END := PTR and PTR := LINK[PTR].
- [End of loop.]
6. [Add successor list of N to AVAILE list.]
Set LINK[END] := AVAILE and AVAILE := BEG.
7. [Delete N using Procedure 8.4.]
Call DELETE(NODE, NEXT, START, AVAILN, N, FLAG).
8. Return.

مثال ۸-۶

گراف (بدون جهت) G شکل ۸-۱۲ (الف) را درنظر بگیرید که لیست‌های مجاورتی آن در شکل ۸-۱۲ (ب) نشان داده شده است.

Adjacency Lists	
A:	B, C, D
B:	A, D, E
C:	A, D
D:	A, B, C, E
E:	B, D



(ب)

(الف)

شکل ۸-۱۲

مالحظه می‌کنید که G دارای ۱۴ یال جهت دار است چون ۷ یال جهت دار نیستند. فرض کنید G مانند شکل ۸-۱۳ (الف) در حافظه نگهداری می‌شود. علاوه براین فرض کنید گره B با استفاده از Procedure 8.9 از G حذف می‌شود. مراحل زیر بدست می‌آید:

مرحله ۱. $LOC = 2$ ، مکان B در لیست گره را پیدا می‌کند.

مرحله ۲. $LOC = 2$ ، را از لیست یال یعنی از هر لیست گره‌های بعدی حذف می‌کند.

مرحله ۴. $LOC = 4$ و $END = 6$ اولین و آخرین گره بعدی B را پیدا می‌کند.

مرحله ۶. لیست گره‌های بعدی را از لیست یال حذف می‌کند.

مرحله ۷. گره B را از لیست گره حذف می‌کند.

مرحله ۸. بازگشت

در شکل ۸-۱۳ (الف) عنصرهای حذف شده با دایره مشخص شده‌اند. شکل ۸-۱۳ (ب) G را پس از حذف گره B (و یال‌های آن) در حافظه نشان می‌دهد.

	NODE	NEXT	ADJ
1	A	3	2
2		6	
3	C	4	7
4	D	5	9
5	E	0	14
6		7	
7		8	
8		0	

START = 1

AVAILN = 2

	DEST	LINK
1		15
2	3	3
3	4	0
4		5
5		6
6		13
7	1	8
8	4	0
9	1	11
10		1
11	3	12
12	5	0
13		10
14	4	0
15		16
16		0

AVAILE = 4

(ب) بعد از حذف B

	NODE	NEXT	ADJ
1	A	2	1
2	(B)	3	4
3	C	4	7
4	D	5	9
5	E	0	13
6		7	
7		8	
8		0	

START = 1

AVAILN = 6

	DEST	LINK
1	(2)	2
2	3	3
3	4	0
4	(1)	5
5	(4)	6
6	(5)	0
7	1	8
8	4	0
9	1	10
10	(2)	11
11	3	12
12	5	0
13	(2)	14
14	4	0
15		16
16		0

AVAILE = 16

(الف) قبل از حذف

۷-۸ پیمایش یک گراف

بسیاری از الگوریتم‌های مربوط به گراف، نیازمند آن هستند تا به صورت منظم و اصولی گره‌ها و یال‌های گراف G را مورد بررسی و پردازش قرار دهند. دو روش استاندارد برای این منظور وجود دارد. یکی از این روشها جستجوی عرضی یا ردیفی و روش دیگر جستجوی عمقی نام دارد. جستجوی عرضی از یک صفت به عنوان یک ساختمان کمکی جهت نگهداری گره‌ها در پردازش بعدی استفاده می‌کند و به طور مشابه جستجوی عمقی از یک پشته استفاده می‌کند.

در خلال اجرای الگوریتم‌ها، هر گره N از گراف G دارای یکی از سه حالت زیر موسوم به وضعیت N خواهد بود:

STATUS = 1: (حالت آماده) حالت اولیه گره N

STATUS = 2: (حالت انتظار) گره N داخل صفت یا پشته در انتظار پردازش بسر می‌برد.

STATUS = 3: (حالت پردازش شده) گره N پردازش شده است.

اکنون دو روش جستجو را به طور مستقل مورد بحث و بررسی قرار می‌دهیم.

جستجوی عرضی

ایده‌کلی که در پشت جستجوی عرضی وجود دارد آن است که کار را با گره آغازش به شرح زیر شروع می‌کنیم. نخست گره آغازین A را ملاقات می‌کنیم. آنگاه تمام همسایه‌ها یا گره‌های مجاور A را ملاقات می‌کنیم. بدنبال آن تمام همسایه‌های (گره‌های مجاور) A را ملاقات می‌کنیم و الى آخر. طبیعی است که لازم است همسایه‌های یک گره را تعقیب کنیم و لازم است اطمینان داشته باشیم که هیچ گره‌ای بیشتر از یکبار پردازش نشود. این کار با استفاده از یک صفت جهت نگهداشت گره‌هایی که درانتظار پردازش بسر می‌برند و با استفاده از فیلد STATUS که وضعیت جاری هر گره را به ما اطلاع می‌دهد انجام می‌شود. الگوریتم به شرح زیر است:

الگوریتم A: این الگوریتم جستجوی عرضی را با شروع از گره آغازین A روی یک گراف G اجرا می‌کند.

۱- تمام گره‌هایی را که در حالت آماده $STATUS = 1$ هستند مقدار اولیه می‌دهد.

۲- گره آغازین A را در QUEUE قرار دهید و وضعیت آن را به حالت انتظار $STATUS = 2$ تغییر دهید.

۳- مرحله‌های ۴ و ۵ را تا وقتی QUEUE خالی نشده تکرار کنید.

۴- گره ابتدای QUEUE را حذف کنید. N را پردازش کنید و وضعیت N را به حالت پردازش شده $STATUS = 3$ تغییر دهید.

۵- تمام همسایه‌های N را به انتهای QUEUE اضافه کنید که در حالت آماده هستند $STATUS = 1$ و

وضعیت آنها را به حالت انتظار $STATUS = 2$ تغییر دهید.

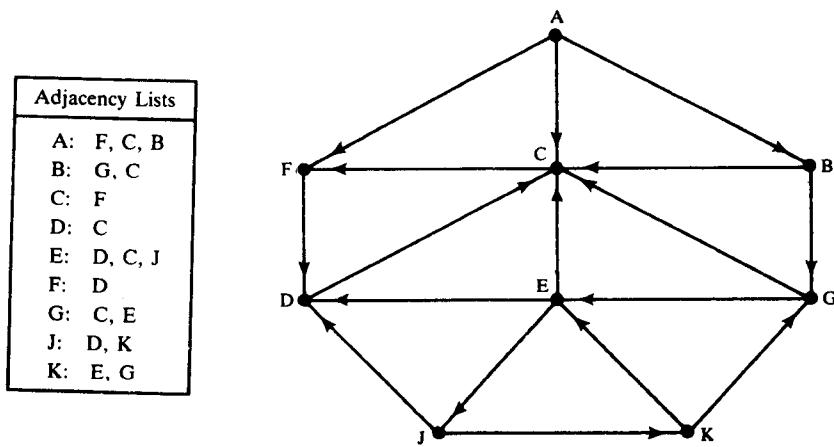
[پایان حلقه مرحله ۱۳]

۶. پایان . Exit

الگوریتم بالا تنها آن دسته از گره‌ها را پردازش می‌کند که از گره آغازین A قابل دسترسی هستند. فرض کنید بخواهیم تمام گره‌های گراف G را ملاقات کنیم. آنگاه الگوریتم باید طوری اصلاح شود تا کار را مجددًا با گره دیگری شروع کند که ما آن را B می‌نامیم یعنی همچنان در حالت آماده است. این گره B را می‌توان با پیمایش لیست گره‌ها به دست آورد.

مثال ۸.۷

گراف G شکل ۸-۱۴ (الف) را در نظر بگیرید. لیست‌های مجاورتی گره‌ها در شکل ۸-۱۴ (ب) نشان داده شده است.



(ب)

شکل ۸-۱۴

(الف)

فرض کنید G نمایش پروازهای روزانه بین شهرها یک شرکت هوایپمایی باشد و بخواهیم از شهر A به شهر J با حداقل تعداد توقف پرواز کنیم. به بیان دیگر کوتاهترین مسیر از A به J را پیدا کنیم که در آن هر یال طول 1 دارد.

کوتاهترین مسیر P را می‌توان با استفاده از جستجوی عرضی با شروع از شهر A و به پایان رسیدن در

J هنگام ملاقات آن پیدا کرد. در طی اجرای جستجو، ابتدای هر یال را با استفاده از آرایه **ORIG** به همراه آرایه **QUEUE** نگه می‌داریم. مراحل جستجو به شرح زیر است:
 (الف) در آغاز، A را به **QUEUE** اضافه کنید و NULL را به **ORIG** به صورت زیر اضافه کنید:

FRONT = 1	QUEUE: A
REAR = 1	ORIG: Ø

(ب) عنصر ابتدا A را از **QUEUE** با قراردادن دستور جایگزینی $FRONT := FRONT + 1$ حذف کنید و به **QUEUE** همسایه‌های A را به صورت زیر اضافه کنید:

FRONT = 2	QUEUE: A, F, C, B
REAR = 4	ORIG: Ø, A, A, A

توجه دارید که A ابتدای هر سه یال به **ORIG** اضافه می‌شود.

(ج) عنصر ابتدا F را با قراردادن دستور جایگزینی $FRONT := FRONT + 1$ از **QUEUE** حذف کنید و به **QUEUE** همسایه‌های F را به صورت زیر اضافه کنید:

FRONT = 3	QUEUE: A, F, C, B, D
REAR = 5	ORIG: Ø, A, A, A, F

(د) عنصر ابتدا C را از **QUEUE** حذف کنید همچنین به **QUEUE** همسایه‌های C را به صورت زیر اضافه کنید (که در حالت آماده هستند):

FRONT = 4	QUEUE: A, F, C, B, D
REAR = 5	ORIG: Ø, A, A, A, F

توجه دارید که همسایه F از C به **QUEUE** اضافه نمی‌شود چون F در حالت آماده نیست (زیرا F قبل از **QUEUE** اضافه شده است).

(ه) عنصر ابتدا B را از **QUEUE** حذف کنید همچنین به **QUEUE** همسایه‌های B را به صورت زیر اضافه کنید (همسایه‌هایی که در حالت آماده هستند):

FRONT = 5	QUEUE: A, F, C, B, D, G
REAR = 6	ORIG: Ø, A, A, A, F, B

توجه دارید که تنها G به **QUEUE** اضافه می‌شود چون همسایه دیگر، C در حالت آماده نیست.

(و) عنصر ابتدا D را از **QUEUE** حذف کنید همچنین همسایه‌های D را که در حالت آماده هستند به **QUEUE** اضافه کنید:

FRONT = 6	QUEUE: A, F, C, B, D, G
REAR = 6	ORIG: Ø, A, A, A, F, B

(ز) عنصر ابتدا G را از QUEUE حذف کنید و همسایه های G را که در حالت آماده هستند به صورت زیر به QUEUE اضافه کنید :

FRONT = 7	QUEUE: A, F, C, B, D, G, E
REAR = 7	ORIG: \emptyset , A, A, A, F, B, G

(ح) عنصر ابتدا E را از QUEUE حذف کنید و همسایه های E را که در حالت آماده هستند به صورت زیر به QUEUE اضافه کنید :

FRONT = 8	QUEUE: A, F, C, B, D, G, E, J
REAR = 8	ORIG: \emptyset , A, A, A, F, B, G, E

به محض اضافه شدن J به QUEUE کار را متوقف می کنیم چون J مقصد نهایی است. اکنون با استفاده از آرایه ORIG برای پیدا کردن مسیر P از J به عقب برمی گردیم. بدین ترتیب

$J \leftarrow E \leftarrow G \leftarrow B \leftarrow A$

که مسیر P ای مورد نظر است.

جستجوی عمقی

ایده کلی که در پشت جستجوی عمقی وجود دارد آن است که کار را با گره آغازین A به شرح زیر شروع می کنیم. نخست گره آغازین A را ملاقات می کنیم. آنگاه هر گره N را که در مسیر P قرار دارد و با شروع می شود را ملاقات می کنیم یعنی یک همسایه A را پردازش می کنیم. آنگاه همسایه همسایه A و الی آخر را پردازش می کنیم. پس از رسیدن به نقطه پایان P یعنی پایان مسیر P به طرف P برمی گردیم تا بتوانیم در طول مسیر دیگر' P پردازش را ادامه دهیم، و الی آخر. این الگوریتم مشابه پیمایش InOrder است که درخت دودویی است، همچنین این الگوریتم مشابه راهی است که شخص ممکن است هنگام عبور از راه پر پیچ و خم طی کند. الگوریتم خیلی شبیه جستجوی عرضی است با این تفاوت که اکنون ما به جای صفت از یک پشته استفاده می کنیم. مجدداً، از فیلد STATUS استفاده می کنیم که به ما وضعیت جاری یک گره را اعلام می کند. الگوریتم به صورت زیر است :

الگوریتم B: این الگوریتم جستجوی عمقی را با شروع از گره آغازین A روی یک گراف G اجرا می کند.

۱- تمام گره هایی را که در حالت آماده 1 = STATUS هستند مقدار اولیه می دهد.

۲- گره آغازین A را در Push STACK کنید و وضعیت آن را به حالت انتظار 2 = STATUS تغییر دهید.

۳- مرحله های ۴ و ۵ را تا وقتی STACK خالی نشده تکرار کنید.

۴- گره بالای STACK را pop کنید. N را پردازش کنید و وضعیت آن را به حالت پردازش

5 = STATUS تغییر دهید.

۵- تمام همسایه‌های N را به داخل **Push STACK** کنید که همچنان در حالت آماده $1 = STATUS$ هستند.

وضعیت آنها را به حالت انتظار $2 = STATUS$ تغییر دهید.

[پایان حلقه مرحله ۳]

۶- پایان Exit.

بار دیگر، الگوریتم بالا تنها آن دسته از گره‌ها را پردازش می‌کند که از گره آغازین A قابل دسترسی هستند. فرض کنید بخواهیم تمام گره‌های گراف G را ملاقات کنیم. آنگاه الگوریتم باید طوری اصلاح شود تا کار را مجدداً با گره دیگری شروع کند که ما آن را B می‌نامیم یعنی همچنان در حالت آماده است. این گره B را می‌توان با پیمایش لیست گره‌ها بدست آورد.

مثال ۸-۸

گراف G شکل ۱۴-۸ (الف) را در نظر بگیرید. فرض کنید بخواهیم تمام گره‌های قابل دسترسی از گره J (و خود J) را پیدا کرده چاپ کنیم. یک راه برای این منظور استفاده از جستجوی عمقی G و شروع از گره J است. مراحل جستجو به شرح زیر است:

(الف) در آغاز، J را به داخل پشته به صورت زیر **Push** کنید:

STACK: J

(ب) عنصر بالا J را **pop** کنید و چاپ کنید همچنین آنگاه تمام همسایه‌های J را که در حالت آماده هستند به داخل پشته به صورت زیر **Push** کنید:

Print J STACK: D, K

(ج) عنصر بالا K را **pop** کنید و چاپ کنید همچنین تمام همسایه‌های K را که در حالت آماده هستند به داخل پشته به صورت زیر **Push** کنید:

Print K STACK: D, E, G

(د) عنصر بالا G را **pop** کنید و چاپ کنید همچنین آنگاه تمام همسایه‌های G را که در حالت آماده هستند به داخل پشته به صورت زیر **Push** کنید:

Print G STACK: D, E, C

توجه دارید که تنها C به داخل پشته **Push** می‌شود چون همسایه دیگر، E ، در حالت آماده نیست زیرا E قبل از داشتن **Push** شده است.

(ه) عنصر بالا C را **pop** کنید و چاپ کنید همچنین آنگاه تمام همسایه‌های C را که در حالت آماده هستند

به داخل پشته به صورت زیر Push کنید:

Print C STACK: D, E, F

(و) عنصر بالا F را pop کنید و چاپ کنید و آنگاه تمام همسایه‌های F را که در حالت آماده هستند به داخل پشته به صورت زیر Push کنید:

Print F STACK: D, E

توجه دارید که D تنها همسایه F به داخل پشته Push نمی‌شود چون D در حالت آماده نیست. زیرا D قبل از F در حالت آماده نیست.

(ز) عنصر بالا E را pop کنید و چاپ کنید همچنین تمام همسایه‌های E را که در حالت آماده هستند به داخل پشته به صورت زیر Push کنید.

Print E STACK: D

توجه دارید که هیچیک از سه همسایه E در حالت آماده نیستند.

(ح) عنصر بالا D را pop کنید و چاپ کنید همچنین تمام همسایه‌های D را که در حالت آماده هستند به داخل پشته به صورت زیر Push کنید:

Print D STACK:

اکنون پشته خالی است، بنابراین جستجوی عمقی G که از J شروع شده است در همینجا به پایان می‌رسد. بدین ترتیب گره‌های چاپ شده

J, K, G, C, F, E, D

دقیقاً گره‌هایی هستند که از J قابل دسترسی می‌باشند.

۸-۸ مجموعه‌های جزوأ مرتب، مرتب سازی موضعی

فرض کنید S یک گراف باشد به طوری که هر گره v از S نمایش یک کار (Task) و هر یال (u, v) به معنی پایان کار u باشد که پیشناز شروع کار v است. فرض کنید S چنین گرافی شامل یک دور یا حلقه نظری

$$P = (u, v, w, u)$$

است معنی آن این است که تا وقتی u به پایان نرسیده است v نمی‌تواند شروع شود و تا وقتی v به پایان نرسیده است w نمی‌تواند شروع شود و تا وقتی w به پایان نرسیده است u نمی‌تواند شروع شود. بنابراین هر یک از کارهای داخل دور نمی‌تواند به پایان برسد. برطبق آن، S یک چنین گرافی که کارها را نشان می‌دهد و یک رابطه پیشنازی است نمی‌تواند دور یا حلقه داشته باشد.

فرض کنید S یک گراف بدون دور یا بدون حلقه باشد. رابطه \rightarrow را روی S در نظر بگیرید که به صورت زیر تعریف می‌شود

اگر یک مسیر از u به v وجود داشته باشد.

این رابطه دارای سه خاصیت زیر است :

(۱) برای هر عنصر u در S ، داریم $u < u$ (ضد انعکاسی).

(۲) اگر $v < u$ آنگاه $u < v$ (ضد تقارنی).

(۳) اگر $v < w$ و $w < u$ آنگاه $v < u$ (ترانзیتی).

یک چنین رابطه‌ای $<$ روی S یک ترتیب جزئی S نامیده می‌شود همچنین به S با چنین ترتیبی مجموعه جزئی مرتب یا Partially Ordered SET یا Poset می‌گویند. بنابراین گراف S بدون دور یا

بدون حلقه را می‌توان به صورت یک مجموعه جزئی مرتب موردنظر قرار داد.

از طرف دیگر، فرض کنید S یک مجموعه جزئی مرتب با ترتیب جزئی بیشتر که با $<$ نمایش داده می‌شود. آنگاه S را می‌توان به صورت گرافی درنظر گرفت که گره‌های آن عناصرهای S هستند و یال‌های آن به صورت زیر تعریف می‌شوند :

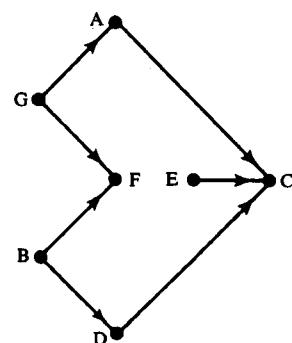
(u, v) یک یال در S است اگر $v < u$

علاوه بر این می‌توان نشان داد که یک مجموعه جزئی مرتب S با درنظر گرفتن آن به صورت یک گراف، هیچ دوری یا حلقه‌ای ندارد.

مثال ۸-۹

فرض کنید S گراف شکل ۸-۱۵ باشد.

Adjacency Lists	
A:	C
B:	D, F
C:	
D:	C
E:	C
F:	
G:	A, F



(ب)

(الف)

شکل ۸-۱۵

ملاحظه می‌کنید که S هیچ دوری یا حلقه‌ای ندارد. بنابراین S را می‌توان به صورت یک مجموعه

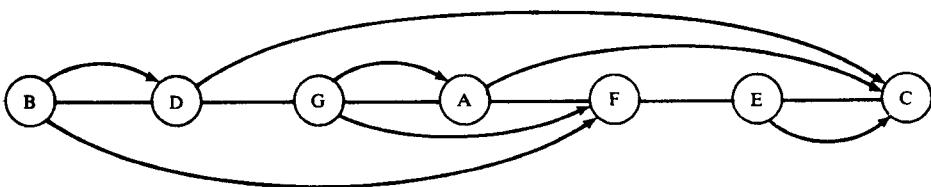
جزئاً مرتب در نظر گرفت. توجه دارید که $G < C$ ، چون از G به C یک مسیر وجود دارد. به همین ترتیب، $B < C$ و $B < F$ از طرف دیگر A چون هیچ مسیری از B به A وجود ندارد. علاوه بر این B

مرتب‌سازی موضعی Topological Sorting

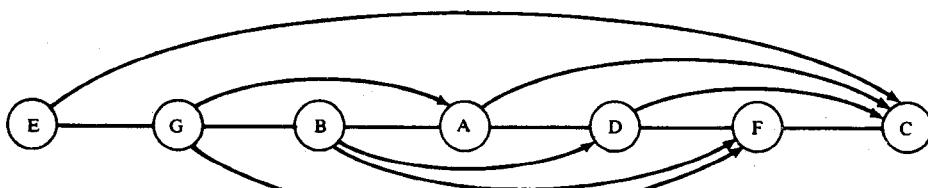
فرض کنید S یک گراف جهت‌دار بدون دور یا بدون حلقه (یا یک مجموعه جزوی مرتب) باشد. یک مرتب‌سازی موضعی T گراف S یک ترتیب خطی از گره‌های S است که در آن ترتیب جزئی اولیه حفظ می‌شود. به بیان دیگر اگر $u < v$ در S باشد (یعنی اگر یک مسیر از u به v در S وجود داشته باشد)، آنگاه u قبل از v در ترتیب خطی T ظاهر می‌شود. شکل ۱۶-۸ دو مرتب‌سازی موضعی مختلف گراف S شکل ۱۵-۸ را نشان می‌دهد. يالها در شکل ۱۶-۸ به این خاطر گنجانده شده است تا بیان کند که با جهت ترتیب خطی مطابقت می‌کند.

نتیجه اصلی و نظری مطالب این بخش به صورت زیر است:

قضیه ۱۶-۸: فرض کنید S یک گراف جهت‌دار متناهی و بدون دور (بدون حلقه) یا یک مجموعه جزوی مرتب متناهی باشد. آنگاه برای مجموعه S یک مرتب‌کردن موضعی T وجود دارد:



(الف)



(ب)

شکل ۱۶-۸ دو مرتب‌کردن موضعی

ترجمه دارید که این قضیه تنها وجود مرتب‌سازی موضعی را بیان می‌کند. اکنون الگوریتمی ارائه

می دهیم که یک چنین مرتب سازی موضعی را پیدا می کند.

ایده اصلی در پس این الگوریتم پیدا کردن یک مرتب سازی موضعی T گراف بدون حلقه S آن است که هر گره N با درجه ورودی صفر یعنی بدون هیچ گره قبلی می تواند به عنوان عنصر اول در مرتب کردن T انتخاب شود. بنابراین تا وقتی که گراف S خالی نیست الگوریتم ما دو مرحله زیر را تکرار می کند:

(۱) گره N با درجه ورودی صفر را پیدا می کند.

(۲) N و یال های آن را از گراف S حذف می کند.

ترتیبی که در آن گره ها از گراف S حذف می شوند از آرایه کمکی **QUEUE** استفاده می کند که به طور موقت تمام گره ها با درجه ورودی صفر رانگه می دارد. علاوه بر این الگوریتم از فیلد **INDEG** به گونه ای استفاده می کند که **INDEG(N)** حاوی درجه ورودی جاری گره N است. الگوریتم به شرح زیر است:

الگوریتم **C**: این الگوریتم مرتب سازی موضعی T گراف بدون دور یا بدون حلقه S را پیدا می کند.

۱- **INDEG(N)** درجه ورودی هر گره N از گراف S را پیدا کنید. این عمل را می توان با پیمایش هر لیست مجاورتی مانند مسئله ۱۵-۸ انجام داد.

۲- تمام گره ها با درجه ورودی صفر را در داخل صف قرار دهید.

۳- مراحل ۴ و ۵ را تا وقتی صفت خالی نیست تکرار کنید.

۴- گره ابتدای صف را با قراردادن $1 + \text{FRONT} := \text{FRONT}$ حذف کنید.

۵- برای M هر همسایه گره N مراحل زیر را تکرار کنید :

(الف) قرار دهید $1 - \text{INDEG}(M) = \text{INDEG}(M)$:

[این دستور یال از N تا M را حذف می کند].

(ب) اگر $0 = \text{INDEG}(M)$ آنگاه M را به انتهای صف اضافه کنید.

[پایان حلقه]

[پایان حلقه مرحله ۳]

۶- پایان **Exit**.

مثال ۸-۱۰

گراف S شکل ۸-۱۵-۸ (الف) را در نظر بگیرید. الگوریتم **C** را برای پیدا کردن T مرتب شده موضعی، گراف S بکار می بندیم. مراحل الگوریتم به شرح زیر است:

۱- **INDEG(N)** درجه ورودی هر گره N گراف S را پیدا کنید. نتیجه می شود:

$$\text{INDEG(A)} = 1 \quad \text{INDEG(B)} = 0 \quad \text{INDEG(C)} = 3 \quad \text{INDEG(D)} = 1$$

$$\text{INDEG(E)} = 0 \quad \text{INDEG(F)} = 2 \quad \text{INDEG(G)} = 0$$

این کار را می توان مانند مسئله ۸-۱۵ انجام داد.

۲- در آغاز به صفحه هر گره با درجه ورودی صفر را به صورت زیر اضافه کنید :

$$\text{FRONT} = 1, \quad \text{REAR} = 3, \quad \text{QUEUE: } B, E, G$$

۳ الف - عنصر ابتدا B را از صفحه با قراردادن ۱ به صورت زیر حذف کنید :

$$\text{FRONT} = 2, \quad \text{REAR} = 3 \quad \text{QUEUE: } B, E, G$$

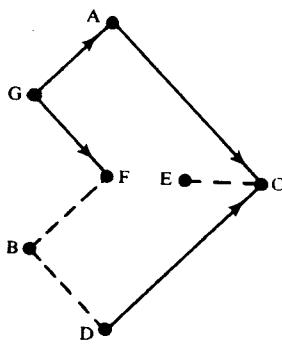
۳ ب - درجه ورودی هر همسایه B را به صورت زیر یک واحد کاهش دهید :

$$\text{INDEG}(F) = 2 - 1 = 1 \quad \text{و} \quad \text{INDEG}(D) = 1 - 1 = 0$$

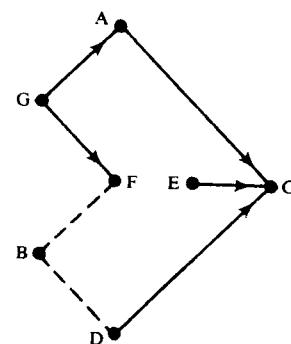
از لیست مجاورتی B در شکل ۸-۱۵ (ب) استفاده کردیم تا همسایه های گره B یعنی F و D را پیدا کنیم. همسایه D به انتهای صفحه می شود چون درجه ورودی آن اکنون صفر است.

$$\text{FRONT} = 2, \quad \text{REAR} = 4 \quad \text{QUEUE: } B, E, G, D$$

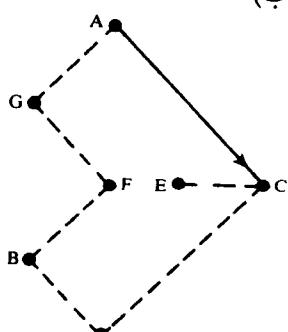
گراف S اکنون شبیه شکل ۸-۱۷ (الف) است که در آن گره B و یال های از B حذف شده اند که به صورت خط چین نشان داده شده است.



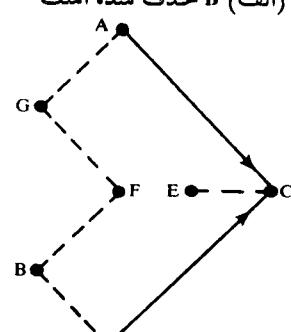
(ب) حذف شده است.



(الف) حذف شده است



(د) حذف شده است.



(ج) حذف شده است

شکل ۸-۱۷

۴ الف - عنصر ابتدا E را از صف با قراردادن $FRONT := FRONT + 1$ به صورت زیر حذف کنید:

$$FRONT = 3, \quad REAR = 4 \quad QUEUE: B, E, G, D$$

۴ ب - درجه ورودی هر همسایه E را به صورت زیر یک واحد کاهش دهید:

$$INDEG(C) = 3 - 1 = 2$$

چون درجه ورودی غیر صفر است، QUEUE تغییر نمی‌کند. اکنون گراف S شبیه شکل ۱۷-۸ (ب) است که در آن گره E و یال آن حذف شده است.

۵ الف - عنصر ابتدا G را از صف با قراردادن $FRONT := FRONT + 1$ به صورت زیر حذف کنید:

$$FRONT = 4, \quad REAR = 4 \quad QUEUE: B, E, G, D$$

۵ ب - درجه ورودی هر همسایه G را به صورت زیر یک واحد کاهش دهید:

$$INDEG(F) = 1 - 1 = 0 \quad INDEG(A) = 1 - 1 = 0$$

هم A و هم F به انتهای صف به صورت زیر اضافه می‌شوند. گراف S اکنون شبیه شکل ۱۷-۸ (ج) است که در آن G و دو یال آن حذف شده است.

$$FRONT = 4, \quad REAR = 6 \quad QUEUE: B, E, G, D, A, F$$

۶ الف - عنصر ابتدای صف را با قراردادن $FRONT := FRONT + 1$ به صورت زیر حذف کنید.

$$FRONT = 5, \quad REAR = 6 \quad QUEUE: B, E, G, D, A, F$$

۶ ب - درجه ورودی هر همسایه D را به صورت زیر یک واحد کاهش دهید:

$$INDEG(C) = 2 - 1 = 1$$

چون درجه ورودی غیر صفر است، QUEUE تغییر نمی‌کند. گراف S اکنون شبیه شکل ۱۷-۸ (د) است که در آن D و یال آن حذف شده است.

۷ الف - عنصر ابتدای صف را با قراردادن $FRONT := FRONT + 1$ به صورت زیر حذف کنید:

$$FRONT = 6, \quad REAR = 6 \quad QUEUE: B, E, G, D, A, F$$

۷ ب - درجه ورودی هر همسایه A را به صورت زیر یک واحد کاهش دهید.

$$INDEG(C) = 1 - 1 = 0$$

از آنجاکه درجه ورودی C صفر است آن را به انتهای صف اضافه کنید.

$$FRONT = 6, \quad REAR = 7 \quad QUEUE: B, E, G, D, A, F, C$$

۸ الف - عنصر ابتدای صف را با قراردادن $FRONT := FRONT + 1$ به صورت زیر حذف کنید.

$$FRONT = 7, \quad REAR = 7 \quad QUEUE: B, E, G, D, A, F, C$$

۸ - گره F همسایه‌ای ندارد، از این رو هیچ تغییری روی نمی‌دهد.

الف - C عنصر ابتدای صف را با قراردادن $1 + \text{FRONT} := \text{FRONT}$ به صورت زیر حذف کنید:

$$\text{FRONT} = 8, \quad \text{REAR} = 7 \quad \text{QUEUE: } B, E, G, D, A, F, C$$

۹ - گره C هیچ همسایه‌ای ندارد، بنابراین هیچ تغییر دیگری روی نمی‌دهد.

اکنون صف هیچ عنصر ابتداندارد، از این رو الگوریتم به پایان رسیده است. عنصرهای آرایه QUEUE که مرتب‌کردن موضعی مورد نظر T گراف S را به دست می‌دهد به صورت زیر است:

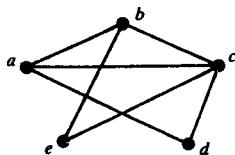
$$T: \quad B, E, G, D, A, F, C$$

الگوریتم می‌توانست در مرحله ۷ ب متوقف شود که در آن REAR برابر تعداد گره‌های گراف S است.

مسائله‌های حل شده

چند اصطلاح نظریه گراف

مسئله ۱ - ۸: گراف (بدون جهت) G شکل ۱۸ - ۸ (الف) را در نظر بگیرید. (الف) گراف G را برحسب مجتمعه گره‌های آن V و مجموعه یال‌های آن E بنویسید. (ب) درجه هر گره را پیدا کنید.



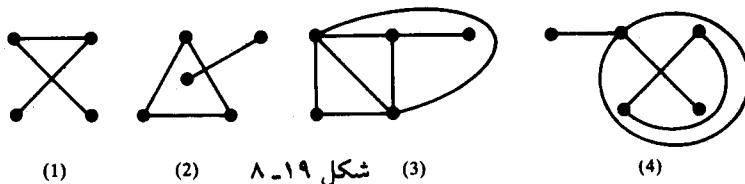
شکل ۱۸ - ۸

حل: (الف) ۵ گره a, b, c, d, e وجود دارد. از این رو $V = \{a, b, c, d, e\}$. هفت زوج $[x, y]$ از گره‌ها وجود دارد به طوری که گره x با گره y متصل است. از این رو

$$E = \{[a, b], [a, c], [a, d], [b, c], [b, e], [c, d], [c, e]\}$$

(ب) درجه یک گره برابر تعداد یال‌هایی است که به آن گره تعلق دارند. برای مثال $\deg(a) = 3$ ، چون a به سه یال تعلق دارد $[a, b], [a, d], [a, e]$. بهمین ترتیب $\deg(b) = 3, \deg(c) = 4, \deg(d) = 2$ و $\deg(e) = 2$.

مسئله ۲ - ۸: گراف‌های چندگانه شکل ۱۹ - ۸ را در نظر بگیرید. کدامیک از آنها (الف) همبند (ب) بدون حلقه یا بدون دو (ج) گراف هستند؟

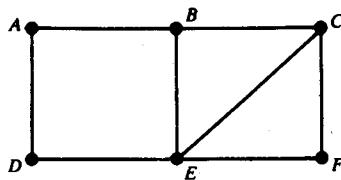


حل : (الف) تنها گرافهای چندگانه ۱ و ۳ همبند هستند.

(ب) تنها گراف چندگانه ۴ یک حلقه (یعنی یک یال با یک نقطه ابتدا و انتهای) دارد.

(ج) تنها گرافهای چندگانه ۱ و ۲ گراف هستند. گراف چندگانه ۳ یالهای چندگانه و گراف چندگانه ۴ یالهای چندگانه و یک حلقه دارد.

مسأله ۳-۸: گراف همبند G شکل ۸-۲۰ را در نظر بگیرید.



شکل ۸-۲۰

(الف) تمام مسیرهای ساده از گره A به گره F را پیدا کنید. (ب) فاصله بین A و F را به دست آورید. (ج) قطر گراف G را به دست آورید. (قطیر گراف G حداقل فاصله موجود بین هر دو گره آن است).

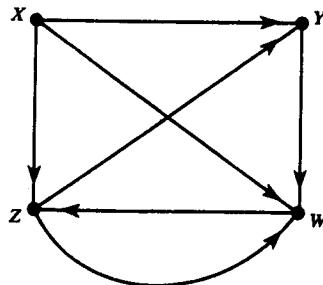
حل : (الف) یک مسیر ساده از A به F مسیری است که هیچ گره و از این رو هیچ یال تکراری ندارد. هفت مسیر ساده از این نوع وجود دارد :

$$\begin{array}{cccc}
 (A, B, C, F) & (A, B, E, F) & (A, D, E, F) & (A, D, E, C, F) \\
 (A, B, C, E, F) & (A, B, E, C, F) & (A, D, E, B, C, F) &
 \end{array}$$

(ب) فاصله A تا F برابر ۳ است، چون یک مسیر ساده (A, B, C, F) از A به F به طول ۳ وجود دارد. و هیچ مسیر کوتاهتری از A به F وجود ندارد.

(ج) فاصله بین A و F برابر ۳ است. همچنین فاصله بین هر دو گره بزرگتر از ۳ نیست. از این رو قطر گراف G برابر ۳ است.

مسأله ۴-۸: گراف (جهت دار) G شکل ۸-۲۱ را در نظر بگیرید.



شکل ۸-۲۱

(الف) تمام مسیرهای ساده از X تا Z را بدست آورید. (ب) تمام مسیرهای ساده از Y تا Z را بدست آورید. (ج) $\text{outdeg}(Y)$ و $\text{indeg}(Y)$ را بدست آورید. (د) آیا گراف منبع یا چشمی دارد؟

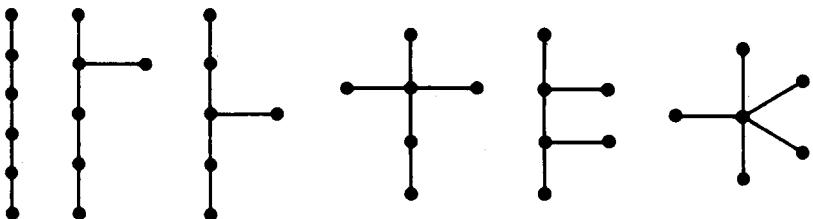
حل : (الف) سه مسیر ساده از X به Z وجود دارد که عبارتند از: (X, Y, W, Z) ، (X, Z) ، (Y, W, X) .

(ب) تنها یک مسیر ساده از Y به Z وجود دارد که عبارت است از: (Y, W, X) .
 (ج) چون دو یال به Y وارد می‌شوند (یعنی با Y به پایان می‌رسند). درنتیجه $\text{indeg}(Y) = 2$. از آنجا که تنها یک یال Y را ترک می‌کند (یعنی با Y شروع می‌شود) درنتیجه $\text{outdeg}(Y) = 1$.

(د) X یک منبع است چون هیچ یالی وارد X نمی‌شود (یعنی $\text{indeg}(X) = 0$) اما چند یال X را ترک می‌کنند (یعنی $\text{outdeg}(X) > 0$). گراف چاه ندارد. چون هر گره یک درجه خروجی غیر صفر دارد (یعنی هر گره نقطه اولیه یک یال است).

مسئله ۸-۵: تمام درختهایی را رسم کنید که دقیقاً 6 گره دارند. (گراف G مشابه گراف ' G' است اگر یک تناظر یک به یک بین مجموعه گره‌ها V گراف G و مجموعه گره‌ها ' V' گراف ' G' وجود داشته باشد. به طوری که (U, V) یک یال در G است اگر و فقط اگر زوج (' u ' ، ' v ') متناظر با آن از گره‌ها یک یال در ' G باشد.)

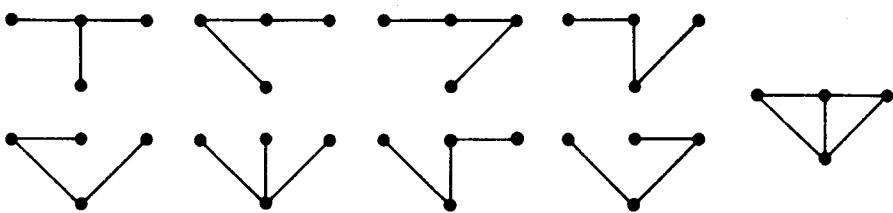
حل : شش درخت از این نوع وجود دارد که در شکل ۸-۲۲ نشان داده شده است.



شکل ۸-۲۲

قطر درخت اول ۵، قطر دو درخت بعدی ۴ و دو درخت بعد از آن ۳ و قطر درخت آخر ۲ است. هر درخت دیگر با ۶ گره، مشابه یکی از این درختها است.

مسئله ۶-۸: تمام درختهای فراگیر گراف G را که در شکل ۲۳-۸ (الف) نشان داده شده است پیدا کنید. (یک درخت T یک درخت فراگیر گراف همبند G نامیده می‌شود اگر T دارای همان گره‌های گراف G باشد و تمام یال‌های T در بین یال‌های G باشد).



(ب)

(الف)

شکل ۲۳-۸

حل: هشت درخت فراگیر وجود دارد که در شکل ۲۳-۸ (ب) نشان داده شده است. از آنجاکه G ۶ گره دارد، هر درخت فراگیر T باید $3 = 1 - 4$ یال داشته باشد، بدین ترتیب هر درخت فراگیر را می‌توان با حذف ۲ یال از ۵ یال گراف G بدست آورد. این کار را می‌توان با ۱۰ روش انجام داد بجز آنکه دو تا از آنها، منتهی به گراف ناهمبند می‌شود. از این‌رو هشت درخت فراگیر نشان داده شده همگی درخت فراگیر گراف G هستند.

نمایش ترتیبی گرافها Sequential Graphs

مسئله ۷-۸: گراف G شکل ۲۱-۸ را در نظر بگیرید. فرض کنید گره‌ها به صورت زیر در آرایه DATA در حافظه ذخیره شده‌اند:

DATA: X, Y, Z, W

(الف) ماتریس مجاورتی (همسایگی) گراف G را بدست آورید.

(ب) ماتریس مسیر گراف G را با استفاده از توانهای مختلف ماتریس مجاورتی A بدست آورید.

(ج) آیا گراف G همبند قوی است؟

حل: (الف) معمولاً گره‌ها برطبق روشی که در حافظه ظاهر می‌شوند مرتب می‌شوند، یعنی فرض می‌کنیم

: ماتریس مجاورتی گراف G عبارت است از :

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

از این رو $a_{ij} = 1$ اگر یک گره از v_i به v_j وجود داشته باشد در غیر اینصورت $0 = a_{ij}$.

(ب) چون G ۴ گره دارد، A^2, A^3, A^4, A^5, A^6 را محاسبه کنید:

$$A^2 = \begin{pmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \quad A^3 = \begin{pmatrix} 0 & 1 & 2 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

$$A^4 = \begin{pmatrix} 0 & 2 & 2 & 3 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 0 & 1 & 1 & 1 \end{pmatrix} \quad B_4 = \begin{pmatrix} 0 & 5 & 6 & 8 \\ 0 & 1 & 2 & 3 \\ 0 & 3 & 3 & 5 \\ 0 & 2 & 3 & 5 \end{pmatrix}$$

اکنون با قراردادن $1 = p_{ij}$ به جای درایه غیرصفر ماتریس B_4 ، ماتریس مسیر P به دست می‌آید. بنابراین

$$P = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

(ج) ماتریس مسیر نشان می‌دهد که از v_2 به v_1 هیچ مسیری وجود ندارد. در واقع هیچ مسیری از هر گره به v_1 وجود ندارد. بنابراین گراف G همبند قوی نیست.

مسئله ۸: گراف G شکل ۲۱-۸ و ماتریس مجاورتی آن A را که در مسئله ۷-۸ بدست آمد درنظر بگیرید. P ماتریس مسیر گراف G را با استفاده از الگوریتم وارشاو بجای توانهای A پیدا کنید.

حل: ماتریسهای P_0, P_1, P_2, P_3, P_4 را محاسبه کنید که در ابتدا $A = P_0$ و

$$P_k[i, j] = P_{k-1}[i, j] \vee (P_{k-1}[i, j] \wedge P_{k-1}[k, j])$$

یعنی $P_k[i, j] = 1$ اگر $P_{k-1}[i, j] = 1$ یا هم $P_{k-1}[i, k] = 1$ و هم $P_{k-1}[k, j] = 1$ است.

در آن صورت :

$$P_1 = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad P_2 = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$P_3 = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \quad P_4 = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

ملاحظه می‌کنید که $A = P_0 = P_1 = P_2 = P_3$ تغییرات در P_3 به دلایل زیر صورت می‌گیرد:

$$P_2(3, 2) = 1 \quad \text{چون } P_3(4, 2) = 1$$

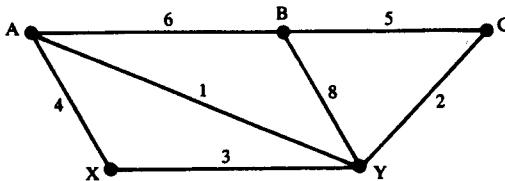
$$P_2(3, 4) = 1 \quad \text{چون } P_3(4, 4) = 1$$

تغییرات در P_4 به طور مشابه صورت می‌گیرد. ماتریس آخر، P_4 ، ماتریس مسیر P مورد نظر گراف G است.

مسأله ۸-۹: گراف وزن داده شده (بدون جهت) G در شکل ۸-۲۴ را در نظر بگیرید. فرض کنید گره‌ها در آرایه DATA در حافظه به صورت زیر ذخیره شده‌اند.

DATA: A, B, C, X, Y

ماتریس وزن (w_{ij}) گراف G را پیدا کنید.



شکل ۸-۲۴

حل: فرض کنید $Y = v_1 = A, v_2 = B, v_3 = C, v_4 = X, v_5 = Y$ در نتیجه W ماتریس وزن گراف G به صورت زیر است:

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

در اینجا w_{ij} نمایش وزن یال از v_i به v_j است چون G بدون جهت است، W یک ماتریس متقارن است یعنی $w_{ji} = w_{ij}$.

مسأله ۱۰-۸: فرض کنید G یک گراف (بدون جهت) باشد که بدون دور یا بدون حلقه است. فرض کنید (p_{ij}) ماتریس مسیر G باشد.

(الف) چه وقت می‌توان یک یال $[j, l]$ به گراف G اضافه کرد طوری که G همچنان بدون دور یا بدون حلقه باقی بماند؟

(ب) هرگاه یک یال $[j, l]$ به G اضافه شود تغییرات ماتریس مسیر P چگونه است؟

حل : (الف) هرگاه یال $[j_i, v_j]$ به گراف G اضافه شود یک دور یا حلقه تشکیل می‌شود اگر و فقط اگر قبل از این روز وجود داشته باشد. از این رو وقتی $0 = p_{ij}$ است می‌توان یال را به گراف G اضافه کرد.

(ب) نخست قرار دهید $1 = p_{ij}$ چون یال یک مسیر از v_i به v_j است. علاوه بر این قرار دهید $1 = p_{si} = p_{ji} = 1$ به بیان دیگر اگر هم یک مسیر P_1 از v_s به v_i و هم یک مسیر P_2 از v_j به v_i وجود داشته باشد، آنگاه $P_1, P_2, [v_s, v_i], [v_i, v_j]$ یک مسیر از s به j تشکیل می‌دهد.

مسئله ۱۱-۸: یک درخت فراگیر حداقل T از یک گراف وزن داده شده G ، یک درخت فراگیر G (مسئله ۶-۸) است هرگاه در بین تمام درختهای فراگیر G حداقل وزن را داشته باشد.

(الف) یک الگوریتم برای پیدا کردن درخت فراگیر حداقل T از یک گراف وزن داده شده G ارائه دهید.

(ب) درخت فراگیر حداقل T گراف شکل ۲۴-۸ را پیدا کنید.

حل : (الف)

- Algorithm P8.11:** This algorithm finds a minimum spanning tree T of a weighted graph G .
1. Order all the edges of G according to increasing weights.
 2. Initialize T to be a graph consisting of the same nodes as G and no edges.
 3. Repeat the following $M - 1$ times, where M is the number of nodes in G :
Add to T an edge E of G with minimum weight such that E does not form a cycle in T .
[End of loop.]
 4. Exit.

این الگوریتم درخت فراگیر حداقل T گراف وزن داده شده G را پیدا می‌کند.

۱- تمام یال‌های G را بر طبق صعودی بودن وزنها مرتب کنید.

۲- T را مقدار اولیه دهید تا یک گراف مشتمل از همان گره‌های G و هیچ یال باشد.

۳- کارهای زیر را $M - 1$ مرتبه تکرار کنید که در آن M تعداد گره‌های G است: به T یک یال E از G با حداقل وزن را به گونه‌ای اضافه کنید که تشکیل یک دور یا حلقه در T را نمی‌دهد.

[پایان حلقه]

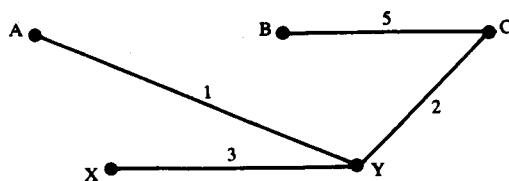
۴- پایان Exit

مرحله ۳ را می‌توان با استفاده از نتیجه‌های مسئله ۱۰-۸ پیاده‌سازی کرد. مسئله ۱۰-۸ (الف) به

ما می‌گوید که هر یال e را می‌توان طوری به T اضافه کرد که تشکیل هیچ دوری را ندهد یعنی طوری که همچنان بدون دور باقی بماند. و مسئله ۱۰-۸ (ب) به ما می‌گوید که چگونه P ماتریس مسیر T را حفظ کنیم هنگامی که یال e به T اضافه می‌شود.

(ب) با استفاده از الگوریتم P8.11 درخت فراگیر حداقل T شکل ۲۵-۸ به دست می‌آید. هرچند $[A, X, B, C]$ دارد اما نمی‌توان $[A, X]$ را به T اضافه کرد چون با $[A, Y]$ و $[X, Y]$ یک دور وزن کمتری از

تشکیل خواهد شد.



شکل ۸-۲۵

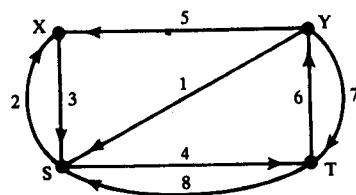
مسئله ۱۲ - ۸: فرض کنید گراف وزن داده شده G به وسیله آرایه گره‌ای DATA و ماتریس وزن W به صورت زیر در حافظه نگهداری می‌شود:

DATA: X, Y, S, T

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

تصویری از گراف G رسم کنید.

حل: تصویر این گراف در شکل ۸-۲۶ رسم شده است.



شکل ۸-۲۶

گره‌ها با درایه‌های DATA برچسب‌گذاری شده‌اند. علاوه‌بر این اگر $w_{ij} \neq 0$ آنگاه یک یال از v_i به v_j با وزن w_{ij} وجود دارد. فرض کنید $v_1 = S$, $v_2 = X$, $v_3 = Y$, $v_4 = T$ و $v_5 = A$, ترتیبی که در آن گره‌ها در آرایه DATA ظاهر شده‌اند.

نمایش پیوندی گرافها

مسئله ۱۳-۸: گراف G به صورت زیر در حافظه ذخیره شده است:

NODE	A	B		E		D	C	
NEXT	7	4	0	6	8	0	2	3
ADJ	1	2		5		7	9	
	1	2	3	4	5	6	7	8

START = 1, AVAILN = 5

DEST	2	6	4		6	7	4		4	6
LINK	10	3	6	0	0	0	0	4	0	0
	1	2	3	4	5	6	7	8	9	10

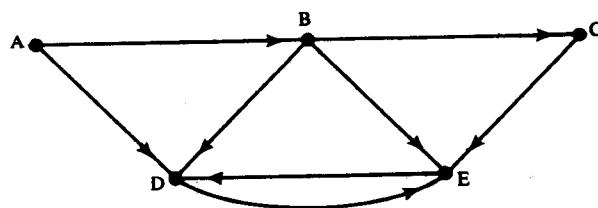
AVAILE = 8

گراف G را رسم کنید.

حل: نخست همسایه‌های هر NODE[K] را با پیمایش لیست مجاورتی آن پیدا می‌کنیم که اشاره گر دارد. نتیجه می‌شود:

E: 6(D)	C: 4(E)	A: 2(B)	و 6(D)
D: 4(E)	B: 6(D), 4(E)	C: 7(C)	

آنگاه نمودار گراف را مانند شکل ۸-۲۷ رسم کنید.



شکل ۸-۲۷

مسئله ۱۴-۸: اگر عملیات زیر انجام شود تغییرات ایجاد شده در نمایش پیوندی گراف G مسئله ۱۳-۸ را تعیین کنید: (الف) گره F به G اضافه شود. (ب) یال (B,E) از G حذف شود. (ج) یال (A,F) به گراف G اضافه شود. گراف حاصل، G را رسم کنید.

حل : (الف) لیست گره مرتب شده نیست، بنابراین F با استفاده از نخستین گره آزاد موجود به صورت زیر به ابتدای لیست اضافه می شود :

START = 5	NODE	A	B		E	D	D	C	
NEXT	7	4	0	6	1	0	2	3	
AVAILN	8	ADJ	1	2		5	0	7	9
	1	2	3	4	5	6	7	8	

ملاحظه می کنید که لیست یال تغییر نمی کند.

(ب) گره E را از لیست مجاورتی گره B به صورت زیر حذف کنید :

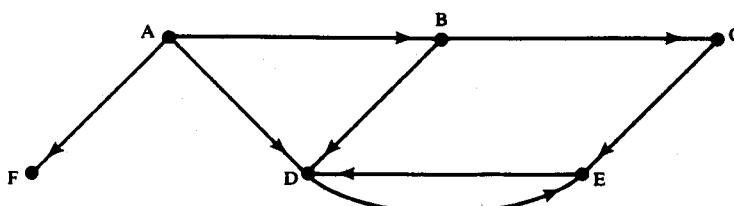
AVAILE = 3	DEST	2	6			6	7	4		4	6
LINK	10	6	1	0	0	0	0	4	0	0	
	1	2	3	4	5	6	7	8	9	10	

ملاحظه می کنید که لیست گره تغییر نمی کند.

(ج) با استفاده از نخستین یال آزاد موجود، مکان 5 گره F به ابتدای لیست مجاورتی گره A اضافه می شود. تغییرات به صورت زیر هستند :

ADJ[1] = 3	DEST	2	6			6	7	4		4	6
AVAILE = 8	LINK	10	6	1	0	0	0	0	4	0	0
	1	2	3	4	5	6	7	8	9	10	

تنها تغییر در لیست گره $3 = ADJ[1]$ است. ملاحظه می کنید که مناطق سایه زده شده تغییرات درون لیستها را بیان می کنند. گراف تازه شده G در شکل ۸-۲۸ نشان داده شده است.



شکل ۸-۲۸

مسئله ۱۵-۸: فرض کنید گراف G در حافظه به صورت

GRAPH(NODE, NEXT, ADJ, START, DEST, LINK)

نگهداری می‌شود. یک زیربرنامه **Procedure** بنویسید که درجه ورودی **INDEG** و درجه خروجی **OUTDEG** هر گره G را پیدا کند.

حل: نخست با استفاده از اشاره‌گر **PTR** به منظور مقدار اولیه صفر دادن به آرایه‌های **INDEG** و **OUTDEG** لیست گره را پیمایش می‌کنیم. آنگاه با استفاده از اشاره‌گر **PTRA** همچنین برای هر مقدار **PTRA** لیست گره را پیمایش می‌کنیم و با استفاده از اشاره‌گر **PTRB**، همسایه‌های **NODE[PTRA]**، **DEST[PTRB]** را پیمایش می‌کنیم. هر باری که یک یال ملاقات می‌شود **PTRA** مکان گره ابتدایی آن و **DEST[PTRB]** مکان گره انتهایی آن را به دست می‌دهد. برطبق آن هر یال آرایه‌های **INDEG** و **OUTDEG** را به صورت زیر تازه می‌کند:

$$\text{OUTDEG[PTRA]} := \text{OUTDEG[PTRA]} + 1$$

و

$$\text{INDEG[DEST[PTRB]]} := \text{INDEG[DEST[PTRB]]} + 1$$

بیان رسمی زیربرنامه **Procedure** به صورت زیر است:

Procedure P8.15: **DEGREE(NODE, NEXT, ADJ, START, DEST, LINK, INDEG, OUTDEG)**

This procedure finds the indegree **INDEG** and outdegree **OUTDEG** of each node in the graph G in memory.

1. [Initialize arrays **INDEG** and **OUTDEG**.]
 - (a) Set **PTR** := **START**.
 - (b) Repeat while **PTR** ≠ **NULL**: [Traverses node list.]
 - (i) Set **INDEG[PTR]** := 0 and **OUTDEG[PTR]** := 0.
 - (ii) Set **PTR** := **NEXT[PTR]**.
- [End of loop.]
2. Set **PTRA** := **START**.
3. Repeat Steps 4 to 6 while **PTRA** ≠ **NULL**: [Traverses node list.]
 4. Set **PTRB** := **ADJ[PTRA]**.
 5. Repeat while **PTRB** ≠ **NULL**: [Traverses list of neighbors.]
 - (a) Set **OUTDEG[PTRA]** := **OUTDEG[PTRA]** + 1 and **INDEG[DEST[PTRB]]** := **INDEG[DEST[PTRB]]** + 1.
 - (b) Set **PTRB** := **LINK[PTRB]**.
 - [End of inner loop using pointer **PTRB**.]
 6. Set **PTRA** := **NEXT[PTRA]**.
- [End of Step 3 outer loop using the pointer **PTRA**.]
7. Return.

مسئله ۱۶-۸: فرض کنید G یک گراف بدون جهت متناهی باشد. آنگاه G متشکل از تعداد متناهی مؤلفه همبند جدا از هم است. الگوریتمی را بیان کنید که تعداد مؤلفه‌های همبند گراف G را پیدا کند.

علاوه بر این الگوریتم باید یک تعداد مؤلفه COMP(N) به هر گره N در یک مؤلفه همبند جایگزین کند طوری که تعداد مؤلفه‌ها از 1 تا NCOMP تغییر کند.

حل: ایده اصلی الگوریتم آن است که برای پیدا کردن تمام گره‌های N که از گره آغازین A قابل دسترس است و در آنها تعداد مؤلفه یکسان را جایگزین می‌کند از جستجوی عرضی یا جستجوی عمیق استفاده کند، الگوریتم به صورت زیر است:

Algorithm P8.16: Finds the connected components of an undirected graph G.

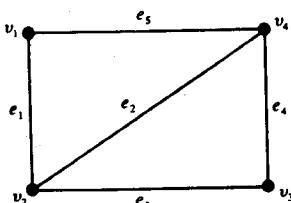
1. Initially set $\text{COMP}(N) := 0$ for every node N in G, and initially set $L := 0$.
2. Find a node A such that $\text{COMP}(A) = 0$. If no such node A exists, then:
 Set $NCOMP := L$, and Exit.
 Else:
 Set $L := L + 1$ and set $\text{COMP}(A) := L$.
3. Find all nodes N in G which are reachable from A (using a breadth-first search or a depth-first search) and set $\text{COMP}(N) = L$ for each such node N.
4. Return to Step 2.

مسئله‌های گوناگون

مسئله ۸-۱۷: فرض کنید G یک گراف بدون جهت با m گره v_1, v_2, \dots, v_m و n یال e_1, e_2, \dots, e_n باشد. آنگاه ماتریس برخورد گراف G ماتریس $m \times n$ ای مانند $(m_{ij}) = M$ است که در آن

$$m_{ij} = \begin{cases} 1 & \text{اگر گره } v_i \text{ به یال } e_j \text{ تعلق داشته باشد} \\ 0 & \text{در غیر اینصورت} \end{cases}$$

ماتریس برخورد گراف G شکل ۸-۲۹ را بدست آورید.



شکل ۸-۲۹

حل: چون G، 4 گره و 5 یال دارد، M یک ماتریس 5×4 است. اگر v_i به e_j بعنوان داشت قرار دهید $m_{ij} = 1$. این عمل ماتریس M زیر را نتیجه می‌دهد:

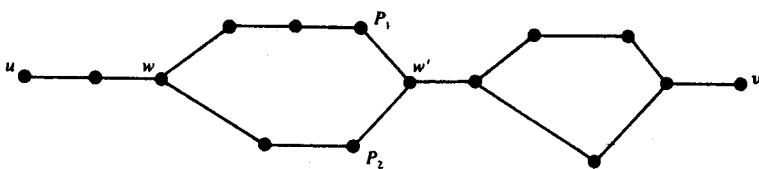
$$M = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

مسئله ۱۸ - ۸: فرض کنید u و v گره‌هایی مجزا در یک گراف بدون جهت G باشند.

ثابت کنید:

(الف) اگر یک مسیر از u به v وجود داشته باشد آنگاه یک مسیر ساده Q از u به v وجود دارد.

(ب) اگر دو مسیر مجزا P_1 و P_2 از u به v وجود داشته باشد، آنگاه G شامل یک دور یا حلقه است.



شکل ۸-۳۰

(الف) فرض کنید $P = (v_0, v_1, \dots, v_n)$ باشد که در آن $v_0 = u$ و $v_n = v$. اگر $v_1 = v$ آنگاه $P' = (v_0, \dots, v_i, v_{i+1}, \dots, v_n)$

یک مسیر از u به v است که کوتاهتر از P است. با تکرار این عمل، نهایتاً مسیر Q از u به v بدست می‌آید که گره‌هایش متمایز هستند. بدین ترتیب Q یک مسیر ساده از u به v است.

(ب) فرض کنید w یک گره در P_1 و P_2 باشد به گونه‌ای که گره‌های بعدی در P_1 و P_2 متمایز هستند. فرض کنید w' نخستین گره بعد از w باشد که روی هر دو P_1 و P_2 قرار داشته باشد. (شکل ۸-۳۰ را ببینید). آنگاه زیرمسیرهای P_1 و P_2 بین w و w' هیچ گره مشترکی ندارند بجز w و w' . از این رو این دو زیرمسیر تشکیل یک دور یا حلقه را می‌دهند.

مسئله ۱۹ - ۸: قضیه ۲-۸ را ثابت کنید: فرض کنید A ماتریس مجاورت گراف G باشد. آنگاه $\sum_{k=1}^{\infty} A^k(i, j)$ درایه z_{ij} ام در ماتریس A^K تعداد مسیرهایی از i به j را بدست می‌دهد که طول K دارند.

حل: با استقراء روی K قضیه راثابت می‌کنیم. نخست توجه دارید که یک مسیر به طول ۱ از i به j دقیقاً یک یال (i, j) است. بنابراین تعریف ماتریس مجاورت A به صورت $a_{ij} = a_{ij}(i, j)$ دارد. از این رو قضیه برای $K = 1$ درست است. فرض کنید $K > 1$. (فرض می‌شود G گره دارد). از

$$\text{آنجاکه } A^K = A^{K-1} A$$

$$a_{ij}(i, j) = \sum_{s=1}^m a_{i-1}(i, s) a_1(s, j)$$

بنابراین تعریف استقراء $a_{ij}(s, K-1)$ تعداد مسیرهایی به طول $1 - K$ از i به j را بدست می‌دهد و $a_{ij}(s, K)$ تعداد مسیرهایی به طول ۱ از s به j را بدست می‌دهد. بنابراین $a_{ij}(s, K) = a_{ij}(s, K-1) a_1(s, j)$.

مسیرهای به طول K از v_i به v_j را به دست می‌دهد که در آن $\leq K$ گره مانده به آخرین گره است. بدین ترتیب تمام مسیرهای به طول K از v_i به v_j را می‌توان با جمع $(i, s)a_1(s, j) \dots a_k(s, j)$ به ازای تمام S به دست آورد. یعنی $(j, s)a_k(i, s)$ تعداد مسیرهای به طول K از v_i به v_j است. بدین ترتیب اثبات قضیه به پایان می‌رسد.

مسئله ۲۰-۸: فرض کنید G یک گراف بدون جهت متناهی بدون دور یا بدون حلقه باشد. هر یک از احکام زیر را ثابت کنید:

- (الف) اگر G حداقل یک یال داشته باشد، آنگاه G یک گره v با درجه ۱ دارد.
- (ب) اگر G همبند باشد طوری که G یک درخت باشد و اگر G ، m گره داشته باشد آنگاه G ، $m-1$ یال دارد.

(ج) اگر G ، m گره و $m-1$ یال داشته باشد آنگاه G یک درخت است.

حل: (الف) فرض کنید $P = v_0, v_1, \dots, v_n$ یک مسیر ساده با حداقل طول باشد. فرض کنید $\deg(v_0) \neq 1$ و فرض کنید $[u, v_0]$ یک یال و $u \neq v_1$ باشد. اگر $u = v_1$ به ازای $1 > i$ باشد آنگاه $C = (v_i, v_0, \dots, u, v_1, v_n)$ یک دور یا حلقه است. اگر $i \neq 1$ آنگاه $P' = (u, v_0, \dots, v_i)$ یک مسیر ساده با طول بزرگتر از P است و هر حالت ممکنی به یک تناقض می‌شود. از این رو $\deg(v_0) = 1$.

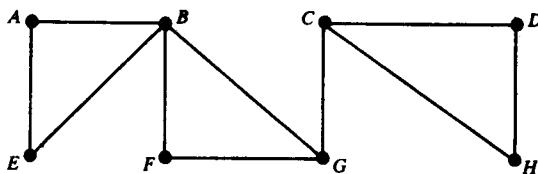
(ب) این حکم را با استقراء روی m ثابت می‌کنیم. فرض کنید $1 = m$ باشد. آنگاه G از یک گره منفرد تشکیل می‌شود و $0 = m-1$ یال دارد. بنابراین نتیجه برای $1 = m$ برقرار است، فرض کنید $1 < m$. اینگاه G یک گره v دارد به طوری که $\deg(v) = 1$. v و تنها یال آن $[v, w]$ را از گراف G حذف کنید تا گراف G' بدست آید. آنگاه G' همچنان همبند باقی می‌ماند و G یک درخت با $m-1$ گره است. بنایه استقراء G' ، $m-2$ یال دارد. از این رو G دارای $1 = m$ یال است. بدین ترتیب نتیجه برقرار است.

(ج) فرض کنید T_1, T_2, \dots, T_s نمایش دهنده مؤلفه‌های همبند گراف G باشند. آنگاه هر T_i یک درخت است. از این رو هر T_i یک گره بیشتر از یال‌ها دارد. از این رو G ، S گره بیشتر از یال‌ها دارد. اما G تنها یک گره بیشتر از یال‌ها دارد. از این رو $1 = S$ است و G یک درخت است.

مسئله‌های تکمیلی

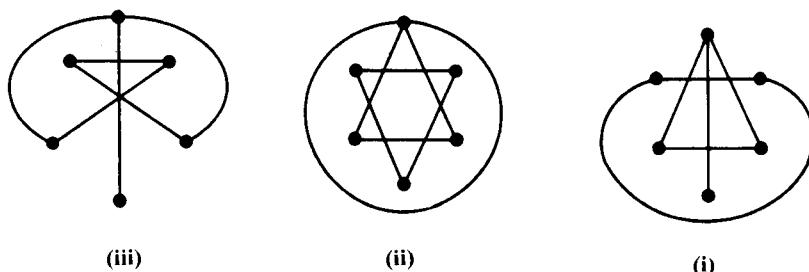
چند اصطلاح نظریه گراف

مسئله ۲۱-۸: گراف بدون جهت شکل ۲۱-۸ را درنظر بگیرید. مطلوب است تعیین (الف) تمام مسیرهای ساده از گره A به گره H (ب) قطر گراف G و (ج) درجه هر گره.



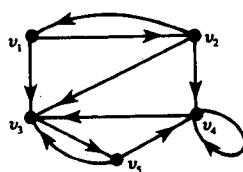
شکل ۸-۳۱

مسئله ۸-۲۲: کدامیک از گرافهای چندگانه شکل ۸-۲۲ (الف) همبند (ب) بدون حلقه (ج) گراف می‌باشد.



شکل ۸-۳۲

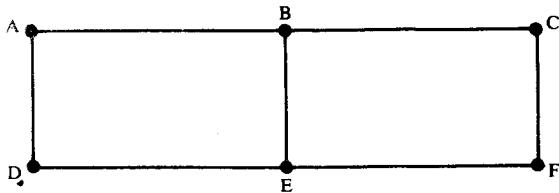
مسئله ۸-۲۳: گراف جهت‌دار G شکل ۸-۳۳ را در نظر بگیرید. (الف) درجه ورودی و درجه خروجی هر گره را پیدا کنید (ب) تعداد مسیرهای ساده از v_1 به v_4 را به دست آورید. (ج) آیا گراف منع یا چشممه دارد؟



شکل ۸-۳۳

مسئله ۸-۲۴: تمام درختهای (غیر مشابه) با S یا تعداد کمتر گره را رسم کنید. هشت درخت از این نوع وجود دارد.

مسئله ۸-۲۵: تعداد درختهای فراگیر گراف G شکل ۸-۳۴ را پیدا کنید.

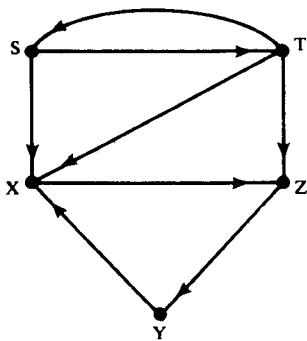


شکل ۸-۳۴

نمایش ترتیبی گرافها، گرافهای وزن داده شده
مسئله ۸-۲۶: گراف G شکل ۸-۳۵ را در نظر بگیرید. فرض کنید گره‌ها در آرایه **DATA** به صورت زیر در حافظه ذخیره می‌شوند.

DATA: X, Y, Z, S, T

(الف) A ماتریس مجاورت گراف G را به دست آورید. (ب) P ماتریس مسیر گراف G را بدست آورید.
(ج) آیا گراف G همبند قوی است؟

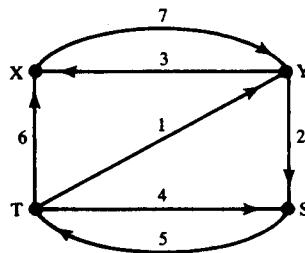


شکل ۸-۳۵

مسئله ۸-۲۷: گراف وزن داده شده G شکل ۸-۳۶ را در نظر بگیرید. فرض کنید گره‌ها در آرایه **DATA** به صورت زیر در حافظه ذخیره می‌شوند.

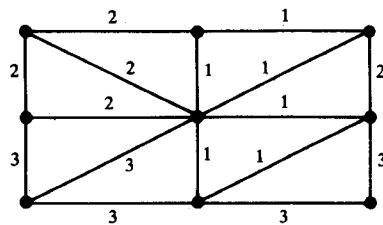
DATA: X, Y, S, T

(الف) ماتریس وزن W گراف G را بدست آورید. (ب) ماتریس Q کوتاهترین مسیر را با استفاده از الگوریتم ۸.۲ وارشاو بدست آورید.



شکل ۸-۳۶

مسأله ۲۸-۸: درخت فرآگیر حداقل گراف G شکل ۸-۳۷ را بدست آورید.



شکل ۸-۳۷

مسأله ۲۹-۸: ماتریس زیر ماتریس برخورد M گراف بدون جهت G است.

$$M = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

توجه دارید که G دارای ۵ گره و ۸ یال است. گراف G را رسم کنید و A ماتریس مجاورت آن را آورید.

مسأله ۳۰-۸: ماتریس زیر ماتریس مجاورت A گراف بدون جهت G است.

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

توجه دارد که G ، ۵ گره دارد، گراف G را رسم کنید و ماتریس برخورد M آن را تعیین کنید.

نمایش پیوندی گرافها

مسئله ۳۱-۸: فرض کنید گراف G به صورت زیر در حافظه ذخیره شده است:

NODE	A		C	E		D		B
NEXT	4	0	8	0	7	3	2	1
ADJ	6		1	10		2		9
	1	2	3	4	5	6	7	8

START = 6, AVAILN = 5

DEST	8	8		1	4	3	3		6	3
LINK	5	7	8	0	0	0	0	0	4	0
	1	2	3	4	5	6	7	8	9	10

AVAILE = 3

گراف G را رسم کنید.

مسئله ۳۲-۸: در مسئله ۳۱-۸ اگر یال (C,E) حذف شود و یال (D,E) اضافه شود تغییراتی، که در نمایش پیوندی گراف G ایجاد می‌شود را تعیین کنید.

مسئله ۳۳-۸: در مسئله ۳۱-۸ اگر گره F و یال (E,F) و (F,D) به G اضافه شوند تغییراتی که در نمایش پیوندی گراف G ایجاد می‌شود را تعیین کنید.

مسئله ۳۴-۸: در مسئله ۳۱-۸ اگر گره B از گراف G حذف شود تغییراتی را که در نمایش پیوندی گراف G ایجاد می‌شود را تعیین کنید.

مسئله های ۳۲-۳۸ در ارتباط با گراف G هستند که به وسیله نمایش پیوندی در حافظه ذخیره می‌شود:

GRAPH[NODE, NEXT, ADJ, START, AVAILN, DEST, LINK, AVAILE]

مسئله ۳۵-۸: یک زیر برنامه Procedure برای انجام هر یک از عملیات زیر بنویسید:

(الف) لیست گره‌های بعدی گره معلوم ND را چاپ کند.

(ب) لیست گره‌های قبلی گره معلوم ND را چاپ کند.

مسئله ۳۶-۸: زیر برنامه‌ای بنویسید که معین کند آیا گراف بدون جهت است یا خیر؟

مسئله ۳۷-۸: زیر برنامه‌ای بنویسید که تعداد گره‌های گراف G را پیدا کند و بدنبال آن A ماتریس مجاورت $M \times M$ گراف G را تعیین کند. گره‌ها با توجه به ترتیب شان در لیست گره G مرتب شده‌اند.

مسئله ۳۸-۸: زیر برنامه‌ای بنویسید که معین کند آیا در گراف G منبع یا چشمه‌ای وجود دارد یا خیر؟

مسئله‌های ۳۹-۴۰ تا ۸-۴۰ در ارتباط با گراف وزن داده شده G هستند که با استفاده از نمایش پیوندی به صورت زیر در حافظه ذخیره می‌شود:

GRAPH[NODE, NEXT, ADJ, START, AVAILN, WEIGHT, DEST, LINK, AVAILE]

مسئله ۳۹-۸: زیربرنامه‌ای بنویسید که کوتاهترین مسیر از گره معلوم NA را تا گره معلوم NB پیدا کند.

مسئله ۴۰-۸: زیربرنامه‌ای بنویسید که طولانی‌ترین مسیر ساده از گره معلوم NA تا گره معلوم NB را پیدا کند.

برای مسئله‌های زیر برنامه بنویسید

مسئله ۴۱-۸: فرض کنید گراف G به وسیله عدد صحیح M که نمایش گره‌های $1, 2, \dots, M$ است و یک لیست از N زوج مرتب از اعداد صحیح که نمایش یال‌های گراف G است به عنوان ورودی برنامه داده می‌شود. زیربرنامه‌ای برای هر یک از عملیات زیر بنویسید:

(الف) A ماتریس مجاورت $M \times M$ گراف G را پیدا کند.

(ب) با استفاده از ماتریس مجاورت A و الگوریتم وارشاو، P ماتریس مسیر گراف G را پیدا کنید. زیربرنامه بالا را با استفاده از داده‌های زیر آزمایش کنید.

$$M = 5; N = 8; (3, 4), (5, 3), (2, 4), (1, 5), (3, 2), (4, 2), (3, 1), (5, 1), \quad (i)$$

$$M = 6; N = 10; (1, 6), (2, 1), (2, 3), (3, 5), (4, 5), (4, 2), (2, 6), (5, 3), (4, 3), (6, 4) \quad (ii)$$

مسئله ۴۲-۸: فرض کنید گراف وزن داده شده G به وسیله عدد صحیح M که نمایش گره‌های $1, 2, \dots, M$ است و یک لیست از N سه تایی مرتب (a_i, b_i, w_i) از اعداد صحیح به عنوان ورودی برنامه داده می‌شود به طوری که زوج (a_i, b_i) یک یال G و w_i وزن آن است، زیربرنامه‌ای برای هر یک از عملیات زیر بنویسید:

(الف) W ماتریس وزن $M \times M$ گراف G را پیدا کنید.

(ب) با استفاده از ماتریس وزن W و الگوریتم وارشاو ۸.۲، Q ماتریس کوتاهترین مسیر بین گره‌ها را پیدا کنید. زیربرنامه بالا را با استفاده از داده‌های زیر آزمایش کنید.

$$M = 4; N = 7; (1, 2, 5), (2, 4, 2), (3, 2, 3), (1, 1, 7), (4, 1, 4), (4, 3, 1). \quad (Compare\ with\ Example\ 8.4.) \quad (i)$$

$$M = 5; N = 8; (3, 5, 3), (4, 1, 2), (5, 2, 2), (1, 5, 5), (1, 3, 1), (2, 4, 1), (3, 4, 4), (5, 4, 4). \quad (ii)$$

مسئله ۴۳-۸: فرض کنید گراف خالی G با استفاده از نمایش پیوندی زیر در حافظه ذخیره می‌شود:

GRAPH(NODE, NEXT, ADJ, START, AVAILN, DEST, LINK, AVAILE)

فرض کنید $NODE$ فضایی برای ۸ گره و $DEST$ فضایی برای ۱۲ یال در اختیار دارد. برنامه‌ای بنویسید که عملیات زیر را روی گراف G اجرا کند:

(الف) گره‌های A, B, C, D را از ورودی بگیرد.

(ب) یال‌های $(B, D), (D, A), (C, B), (A, C), (A, B)$ و (C, D) را از ورودی بگیرد.

(ج) گره‌های E, F را از ورودی بگیرد.

	CITY	LEFT	RIGHT	ADJ
1	Atlanta	0	2	12
2	Boston	0	0	1
3	Houston	0	0	14
4	New York	3	8	4
5		6		
6		0		
7	Washington	0	0	10
8	Philadelphia	0	7	6
9	Denver	10	4	8
10	Chicago	1	0	2

START = 9, AVAILN = 5

	NUMBER	PRICE	ORIG	DEST	LINK
1	201	80	2	10	3
2	202	80	10	2	0
3	301	50	2	4	0
4	302	50	4	2	5
5	303	40	4	8	7
6	304	40	8	4	9
7	305	120	4	9	0
8	306	120	9	4	13
9	401	40	8	7	0
10	402	40	7	8	11
11	403	80	7	1	0
12	404	80	1	7	16
13	501	80	9	3	15
14	502	80	3	9	0
15	503	140	9	1	0
16	504	140	1	9	0
17					18
18					19
19					20
20					0

(د) یال‌های (E, B) و (F, B), (F, E), (B, E) را از ورودی بگیرد.

(ه) یال‌های (D, A) و (B, D) را حذف کند.

(و) گره A را حذف کند.

مسئله‌های ۴۴-۸ تا ۴۸-۸ در ارتباط با داده‌های شکل ۳۸-۸ هستند که در آن شهرها به صورت درخت جستجوی دودویی ذخیره می‌شوند.

مسئله ۴۴-۸: زیربرنامه‌ای با ورودی CITYA و CITYB بنویسید که با فرض وجود پرواز، شماره پرواز و هزینه پرواز از شهر A به شهر B را پیدا کند. زیربرنامه را با استفاده از (الف) CITYA = Chicago, CITYB = New York, CITYA = Washington, CITYB = Denver؛ (ب) CITYA = Boston و (ج) CITYA = Philadelphia آزمایش کنید.

مسئله ۴۵-۸: زیربرنامه‌ای با ورودی CITYA و CITYB بنویسید که مسیر پرواز از شهر A به شهر B را با حداقل تعداد توقف پیدا کند همچنین هزینه آن را محاسبه کند. زیربرنامه را با استفاده از (الف) CITYA = Denver, CITYB = Washington؛ (ب) CITYA = Boston, CITYB = Houston و (ج) CITYA = New York, CITYB = Atlanta آزمایش کنید.

مسئله ۴۶-۸: زیربرنامه‌ای با ورودی CITYA و CITYB بنویسید که ارزان‌ترین مسیر پرواز از شهر A به شهر B را پیدا کند علاوه بر این هزینه آن را محاسبه کند. زیربرنامه را با استفاده از داده‌های مسئله ۴۵-۸ آزمایش کنید. نتیجه‌ها را با هم مقایسه کنید.

مسئله ۴۷-۸: زیربرنامه‌ای بنویسید که یک رکورد را از فایلی که شماره پرواز NUMB را به دست می‌دهد حذف کند. برنامه را با استفاده از (الف) NUMB = 503 و (ب) NUMB = 504 آزمایش کنید.

مسئله ۴۸-۸: زیربرنامه‌ای بنویسید که به عنوان ورودی رکورد زیر
(NUMBNEW, PRICENEW, ORIGNEW, DESTNEW)

راده‌یافت کند. زیربرنامه را با استفاده از داده‌های زیر آزمایش کنید.

(الف) NUMBNEW = 505, PRICENEW = 80, ORIGNEW = Chicago, DESTNEW = Denver
NUMBNEW = 506, PRICENEW = 80, ORIGNEW = Denver, DESTNEW = Chicago

(ب) NUMBNEW = 601, PRICENEW = 70, ORIGNEW = Atlanta, DESTNEW = Miami
NUMBNEW = 602, PRICENEW = 70, ORIGNEW = Miami, DESTNEW = Atlanta

توجه دارید که یک شهر جدید ممکن است به درخت جستجوی دودویی شهرها اضافه شود.

مسئله ۴۹-۸: الگوریتم مرتب سازی موضعی را به یک برنامه تبدیل کنید طوری که یک گراف G را

مرتب کند. فرض کنید G از طریق مجموعه گره‌ها V و مجموعه يالها E به عنوان ورودی داده شده است.

برنامه را با استفاده از گره‌های A, B, C, D, E, X, Y, Z و T و يالهای

(الف) (Z, B) و (S, C) , (Z, X) , (Y, X) , (C, B) , (B, T) , (X, D) , (S, Z) , (A, Z)

(ب) (X, T) و (C, S) , (B, A) , (X, S) , (C, T) , (S, Y) , (Y, B) , (A, X) , (D, Y) , (A, Z)

(ج) (X, A) و (T, Y) , (Z, Y) , (B, T) , (A, S) , (D, Z) , (Z, X) , (Y, Λ) , (B, Z) , (Λ, C)

آزمایش کنید.

مسأله ۵۰-۸: برنامه‌ای بنویسید که تعداد مؤلفه‌های همبند گراف نامرتب G را پیدا کند و همچنین شماره مؤلفه را در هر یک از گره‌های آن جایگزین کند. فرض می‌شود G به وسیله مجموعه گره‌های آن V و مجموعه يالهای (بدون جهت) آن E وارد کامپیوتر شده است. برنامه را با استفاده از گره‌های $S, Z, Y, X, D, C, B, \Lambda$ و T و يالهای

(الف) $[X, S]$ و $[D, B]$, $[Z, A]$, $[A, S]$, $[D, T]$, $[S, Z]$, $[Y, C]$, $[B, T]$, $[A, X]$

(ب) $[S, Z]$ و $[T, B]$, $[Y, B]$, $[X, S]$, $[D, T]$, $[S, C]$, $[A, X]$, $[D, B]$, $[Z, C]$

اجرا کنید.



فصل ۹

مرتب کردن و جستجوی اطلاعات

۱- مقدمه

مرتب کردن و جستجوی اطلاعات از عملیات اساسی و اصلی علم کامپیووتر است. مرتب کردن عبارت است از عمل تجدید آرایش داده ها با یک ترتیب مشخص، مثلاً برای داده های عددی، ترتیب صعودی یا نزولی اعداد یا برای داده های کاراکتری ترتیب الفبایی آنها. جستجو کردن عبارت است از عمل پیدا کردن مکان یک عنصر داده شده در بین مجموعه ای از عناصر.

برای مرتب کردن و جستجوی اطلاعات الگوریتم های متعددی وجود دارد. بعضی از آنها نظری HeapSort و جستجوی دودویی قبلاً در طول کتاب مورد بحث و بررسی قرار گرفته است. الگوریتم خاصی که برای این منظور انتخاب می شود بستگی به خصوصیات داده ها و خصوصیات عملیاتی دارد که ممکن است روی داده ها انجام شود. بنابراین می خواهیم پیچیدگی هر الگوریتم را تعیین کنیم یعنی مایل هستیم زمان اجرا، ($n \log n$) هر الگوریتم را به صورت تابعی از n تعداد داده های ورودی به دست آوریم. گاهی اوقات علاوه بر آن، حافظه موردنیاز الگوریتم ها را مورد توجه و بررسی قرار می دهیم.

مرتب کردن و جستجوی اطلاعات، اغلب روی یک فایل از رکوردها بکار می رود، از این رو لازم است چند اصطلاح استاندارد را یادآوری کنیم. هر رکورد در یک فایل F می تواند شامل چند فیلد باشد اما فیلد ویژه ای وجود دارد که مقدارهای آن بطور منحصر بفردی، رکوردهای داخل فایل را معین می کنند. چنین

فیلدی K یک کلید اولیه یا اصلی و مقادیر k_1, k_2, \dots, k_n چنین فیلدی کلیدها یا مقادیر کلیدی نامیده می‌شود. مرتب کردن فایل F معمولاً به مرتب کردن F نسبت به کلید اولیه خاصی گفته می‌شود و جستجوی اطلاعات در F به جستجوی رکورد با مقدار کلیدی معین گفته می‌شود.

در این فصل نخست الگوریتم‌های مرتب کردن بررسی می‌شود و آنگاه الگوریتم‌های جستجوی اطلاعات مورد بحث قرار می‌گیرد. لازم به تذکر است که در بعضی از کتابها، جستجوی اطلاعات قبل از مرتب کردن آورده می‌شود.

۹-۲ مرتب کردن

فرض کنید A یک لیست n عنصری از A_1, A_2, \dots, A_n در حافظه باشد. منظور از مرتب کردن A ، عمل تجدید آرایش محتوای A است به طوری که با ترتیب صعودی (عددی یا فرهنگ لغتی) باشند یعنی طوری که

$$A_1 \leq A_2 \leq A_3 \dots \leq A_n$$

چون A دارای n عنصر است با $n!$ طریق که محتویات A می‌توانند مقادیر مختلف بگیرد. این روشها دقیقاً متناظر با $n!$ جایگشت اعداد $1, 2, \dots, n$ است. بنابراین الگوریتم مرتب کردن باید این $n!$ حالت ممکن را در نظر بگیرد.

۹-۱ مثال

فرض کنید آرایه DATA شامل 8 عنصر به شرح زیر است :

DATA: 77, 33, 44, 11, 88, 22, 66, 55

پس از مرتب کردن، DATA باید در حافظه به صورت زیر درآید :

DATA: 11, 22, 33, 44, 55, 66, 77, 88

از آنجاکه DATA شامل 8 عنصر است، به تعداد $8! = 40320$ طریق مختلف اعداد 11, 22, ..., 88 را می‌توانند در DATA ظاهر شود.

پیچیدگی الگوریتم‌های مرتب کردن

پیچیدگی الگوریتم مرتب کردن، زمان اجرای الگوریتم را به صورت تابعی از n تعداد عنصرهایی که قرار است مرتب شود اندازه می‌گیرد. متنزه می‌شویم که هر الگوریتم مرتب کردن S از عملیات زیر تشکیل می‌شود که در آن A_1, A_2, \dots, A_n شامل عناصری است که قرار است مرتب شوند و B مکان کمکی است:

(الف) مقایسه‌ها، که آزمایش می‌کند آیا $A_1 < A_2$ یا آیا $B < A_1$ است یا خیر.

(ب) جایگزینی‌ها، که محتوای A_i و A_j یا B را جایجا می‌کند.

(ج) جایگزینی‌ها، که قرار می‌دهد $A_i = B$ و آنگاه قرار می‌دهد $B := A_i$ یا $A_i := B$.

معمولًاً، تابع پیچیدگی تنها تعداد مقایسه‌ها را اندازه می‌گیرد، چون تعداد عملیات دیگر تقریباً برابر ضریب ثابتی از تعداد مقایسه‌ها است.

دو حالت اصلی وجود دارد که پیچیدگی آن مورد توجه قرار می‌گیرد که عبارتند از بدترین حالت و حالت میانگین. در مطالعه حالت میانگین این فرض احتمالی را درنظر می‌گیریم که تمام n جایگشت دارای احتمال وقوع یکسان هستند. دانشجو می‌تواند برای بحث مشروح پیچیدگی به بخش ۵-۲ مراجعه کند.

قبل از این، مرتب کردن حبابی (بخش ۹-۴)، مرتب کردن QuickSort (بخش ۶-۵) و HeapSort (بخش ۱۰-۷) مورد مطالعه قرار گرفته است. تعداد تقریبی مقایسه‌ها و ترتیب پیچیدگی این الگوریتم‌ها به طور خلاصه در جدول زیر ارائه شده است:

Algorithm	Worst Case	Average Case
Bubble Sort	$\frac{n(n-1)}{2} = O(n^2)$	$\frac{n(n-1)}{2} = O(n^2)$
Quicksort	$\frac{n(n+3)}{2} = O(n^2)$	$1.4n \log n = O(n \log n)$
Heapsort	$3n \log n = O(n \log n)$	$3n \log n = O(n \log n)$

نخست توجه کنید که مرتب کردن حبابی روش بسیار کندی است. مزیت اصلی آن در سادگی الگوریتم آن است، ملاحظه کنید که پیچیدگی حالت میانگین ($n \log n$) مرتب کردن HeapSort برابر پیچیدگی QuickSort است، اما پیچیدگی بدترین حالت آن (n^2) به نظر سریع تر از QuickSort، $O(n \log n)$ می‌آید. با وجود این، بطور تجربی آشکار است که به جز در موارد نادر QuickSort بهتر و برتر از HeapSort است.

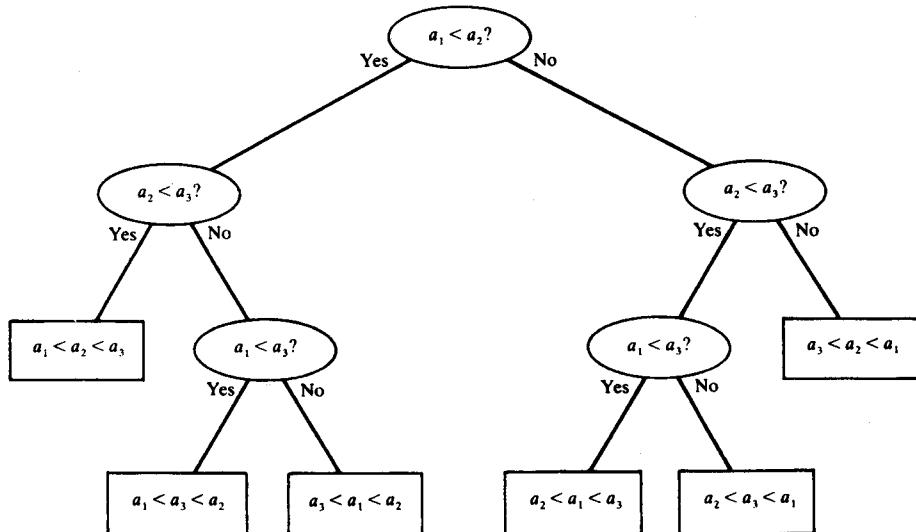
کرانهای پائین

دانشجو ممکن است سؤال کند آیا الگوریتمی وجود دارد که بتواند n عنصر را در زمانی کمتر از $O(n \log n)$ مرتب کند. جواب منفی است. دلیل آن در زیر ارائه شده است:

فرض کنید S الگوریتمی باشد که n عنصر a_1, a_2, \dots, a_n را مرتب می‌کند. فرض می‌کنیم یک درخت تصمیم‌گیری T متناظر با الگوریتم S وجود دارد به طوری که T یک درخت جستجوی دودویی

گسترش یافته است که در آن گره‌های خارجی متناظر با $n!$ طریقی است که n عنصر می‌توانند در حافظه ظاهر شوند و گره‌های داخلی در آن متناظر با مقایسه‌های مختلفی است که ممکن است هنگام اجرای الگوریتم S انجام شود. آنگاه تعداد مقایسه‌ها در بدترین حالت برای الگوریتم S برابر طول طولانی‌ترین مسیر در درخت تصمیم‌گیری T یا به بیان دیگر عمق D درخت T است. علاوه بر این میانگین تعداد مقایسه‌ها برای الگوریتم S برابر \bar{E} میانگین طول مسیر خارجی درخت T است.

شکل ۱-۹ درخت تصمیم‌گیری T را برای مرتب کردن $3 = n$ عنصر نشان می‌دهد.



شکل ۱-۹ درخت تصمیم‌گیری برای مرتب کردن $3 = n$ عنصر

مالحظه می‌کنید که T دارای $6 = 3! = 3 \times 2 \times 1$ گره خارجی است. مقادیر D و \bar{E} برای این درخت عبارتند از:

$$\bar{E} = \frac{1}{6} (2 + 3 + 3 + 3 + 3 + 2) = 2.667 \quad D = 3$$

درنتیجه الگوریتم S متناظر با آن (در بدترین حالت) به حداقل $3 = D$ مقایسه احتیاج دارد و در حالت میانگین به $2.667 = \bar{E}$ مقایسه جهت مرتب کردن $3 = n$ عنصر احتیاج دارد. بنابراین مطالعه پیچیدگی بدترین حالت و حالت میانگین یک الگوریتم مرتب کردن S ، به مطالعه مقادیر D و \bar{E} در درخت تصمیم‌گیری متناظر با آن منتهی می‌شود. با وجود این، نخست چند واقعیت در زمینه درختهای دودویی گسترش یافته (بخش ۱۱-۷) را یادآور می‌شویم. فرض کنید T یک درخت دودویی گسترش یافته با N

گره خارجی، عمق D و طول مسیر خارجی $E(T)$ باشد. هیچیک از چنین درختی نمی‌تواند بیش از 2^D گره خارجی داشته باشد و از این رو

$$D \geq \log_2 N \text{ یا به بیان دیگر } 2^D \geq N$$

علاوه بر این T در بین تمام چنین درختهایی با N گره هنگامی که T یک درخت کامل است. حداقل طول مسیر خارجی $E(L)$ خواهد داشت. در چنین حالتی :

$$E(L) = N \log N + O(N) \geq N \log N$$

$N \log N$ از این واقعیت به دست می‌آید که N مسیر با طول $\log N + 1$ یا $\log N$ وجود دارد همچنین $O(N)$ از این واقعیت ناشی می‌شود که حداقل N گره در عمیق‌ترین سطح وجود دارد: با تقسیم $E(L)$ بر N تعداد مسیرهای خارجی میانگین طول مسیر خارجی \bar{E} به دست می‌آید. بدین ترتیب برای هر درخت دودویی گسترش یافته T با N گره خارجی داریم :

$$\bar{E} = \frac{E(L)}{N} \geq \frac{N \log N}{N} \approx \log N$$

اگرnon فرض کنید T درخت تصمیم‌گیری متناظر با الگوریتم مرتب کردن S باشد که n عنصر را مرتب می‌کند. آنگاه T دارای $n!$ گره خارجی است. با جایگذاری $n!$ به جای N در فرمول بالا نتیجه می‌شود :

$$\bar{E} \geq \log n! \approx n \log n \quad \text{و} \quad D \geq \log n! \approx n \log n$$

شرط $n \log n! \approx n \log n$ از فرمول استرلینگ به دست می‌آید که :

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e} \right)^n \left(1 + \frac{1}{12n} + \dots \right)$$

بدین ترتیب $n \log n$ یک کران پائین برای هم بدترین حالت و هم حالت میانگین است. به بیان دیگر $O(n \log n)$ بهترین حالت ممکن برای هر الگوریتمی است که n عنصر را مرتب می‌کند.

مرتب کردن فایل‌ها، مرتب کردن اشاره گرهای

فرض کنید فایل F متشکل از رکوردهای $R_1, R_2, R_3, \dots, R_n$ در حافظه ذخیره شده است. منظور از مرتب کردن F ، مرتب کردن F نسبت به یک فیلد K با مقادیر متناظر $k_1, k_2, k_3, \dots, k_n$ است. به عبارت دیگر، رکوردهای طوری مرتب می‌شوند که

$$k_1 \leq k_2 \leq \dots \leq k_n$$

فیلد K کلید مرتب کردن نامیده می‌شود. یادآور می‌شویم که K کلید اولیه یا اصلی نامیده می‌شود اگر مقادیر آن به طور منحصر بفردی رکوردهای داخل F را معین کند. مرتب کردن فایل نسبت به یک کلید

دیگر رکوردها را به طریق دیگری مرتب خواهد کرد.

مثال ۹-۲

فرض کنید فایل پرسنلی یک شرکت برای هر کارمند از اطلاعات زیر تشکیل شده است :

Name	Social Security Number	Sex	Monthly Salary
------	------------------------	-----	----------------

با مرتب کردن این فایل نسبت به کلید **Name**، ترتیب دیگری از رکوردها نسبت به مرتب کردن این فایل بر حسب کلید **Social Security Number** به دست می آید.

شرکت ممکن است بخواهد این فایل را براساس فیلد حقوق **Salary** مرتب کند حتی اگر این فیلد کارمندان را به طور منحصر بفردی تعیین نکند. مرتب کردن فایل نسبت به کلید جنسیت **Sex** احتمالاً استفاده‌ای نخواهد داشت. این عمل تنها کارمندان را به دو زیر فایل تجزیه می‌کند که در یکی از این زیر فایلها کارمندان مرد و در زیر فایل دیگر کارمندان زن هستند.

هنگامی که رکوردها خیلی طولانی هستند مرتب کردن فایل F با عوض کردن ترتیب رکوردها می‌تواند خیلی پرهزینه باشد. علاوه بر این می‌توانند در حافظه فرعی یا ثانویه باشند که در این صورت برای انتقال رکوردها در مکانهای مختلف وقت زیادی صرف می‌شود. برطبق آن، بجای مرتب کردن خود رکوردها بهتر است یک آرایه کمکی **POINT** ایجاد کنیم که شامل اشاره‌گرهای رکوردهای داخل حافظه است و آنگاه آرایه **POINT** را نسبت به یک فیلد **KEY** مرتب کنیم. به بیان دیگر، **POINT** را مرتب می‌کنیم طوری که

$$\text{KEY}[\text{POINT}[1]] \leq \text{KEY}[\text{POINT}[2]] \leq \dots \leq \text{KEY}[\text{POINT}[N]]$$

توجه دارید که انتخاب فیلد دیگر **KEY** ترتیب دیگری از آرایه **POINT** را نتیجه می‌دهد.

مثال ۹-۳

شکل ۹-۲ (الف) فایل پرسنلی یک شرکت را در حافظه نشان می‌دهد. شکل ۹-۲ (ب) سه آرایه **POINT**، **PTRNAME** و **PTRSSN** را نشان می‌دهد. آرایه **POINT** شامل مکان رکوردها در حافظه است **PTRNAME** اشاره‌گر مرتب شده را براساس فیلد **NAME** نشان می‌دهد یعنی **NAME[PTRNAME[1]] < NAME[PTRNAME[2]] < ... < NAME[PTRNAME[9]]**

POINT	PTRNAME	PTRSSN
1	2	6
2	3	9
3	4	2
4	6	12
5	7	4
6	9	14
7	10	3
8	12	7
9	14	10
		14

NAME	SSN	SEX	SALARY	
1				
2	Davis	192-38-7282	Female	22 800
3	Kelly	165-64-3351	Male	19 000
4	Green	175-56-2251	Male	27 200
5				
6	Brown	178-52-1065	Female	14 700
7	Lewis	181-58-9939	Female	16 400
8				
9	Cohen	177-44-4557	Male	19 000
10	Rubin	135-46-6262	Female	15 500
11				
12	Evans	168-56-8113	Male	34 200
13				
14	Harris	208-56-1654	Female	22 800

(ب)

(الف)

شکل ۹-۲

و **PTRSSN** اشاره‌گر مرتب شده را براساس فیلد **SSN** نشان می‌دهد یعنی
 $SSN[PTRSSN[1]] < SSN[PTRSSN[2]] < \dots < SSN[PTRSSN[9]]$

نام (EMP) یک کارمند داده شده است بهسادگی می‌توان مکان **NAME** را در حافظه با استفاده از آرایه **RTRNAME** و الگوریتم جستجوی دودویی پیدا کرد. به همین ترتیب، اگر شماره تأمین اجتماعی **NUMB** یک کارمند داده شده باشد. بهسادگی می‌توان با استفاده از آرایه **PTRSSN** و الگوریتم جستجوی دودویی مکان رکورد کارمند را در حافظه پیدا کرد. علاوه براین ملاحظه می‌کنید که حتی، لازم نیست رکوردها، در مکان‌های متوالی حافظه ظاهر شوند. بعاین ترتیب اضافه و حذف رکوردها به آسانی انجام می‌شود.

۳-۹ مرتب کردن درجی یا جای دادنی

فرض کنید آرایه **A** با n عنصر $A[N], A[2], A[1], \dots, A[2], A[1]$ در حافظه است. الگوریتم مرتب کردن درجی عمل جستجو و خواندن آرایه **A** را از $A[N]$ تا $A[1]$ انجام می‌دهد و هر عنصر $A[K]$ را در مکان صحیح‌ش

در زیر آرایه ازقبل مرتب شده $A[0] - A[1], A[2], A[3], \dots, A[N-1]$ درج می‌کند (جای می‌دهد) یعنی مرحله ۱. $A[1]$ خودش بطور بدیهی، مرتب است.

مرحله ۲. $A[2]$ را یا قبل از یا بعد از $A[1]$ درج کنید طوری که $A[1], A[2], A[3]$ مرتب شده باشد.

مرحله ۳. $A[3]$ را در مکان صحیحش در $A[1], A[2], A[3]$ درج کنید یعنی قبل از $A[1]$ و $A[2]$ یا پس از $A[2]$ طوری که $A[1], A[2], A[3], A[4]$ مرتب شده باشند.

مرحله ۴. $A[4]$ را در مکان صحیحش در $A[1], A[2], A[3], A[4]$ درج کنید طوری که $A[1], A[2], A[3], A[4]$ مرتب شده باشد

.....
مرحله N . $A[N]$ را در مکان صحیحش در $A[1], A[2], \dots, A[N-1]$ درج کنید طوری که $A[N], A[2], A[1]$ مرتب شده باشند.

وقتی n عدد کوچکی است اغلب این الگوریتم مرتب کردن مورد استفاده قرار می‌گیرد. برای مثال این الگوریتم نزد بازیکنان! بازی bridge و فتنی که برای نخستین بار کارت‌ها را مرتب می‌کنند! بسیار معروف است. تنها مسئله‌ای که باقی می‌ماند تصمیم‌گیری در مورد چگونگی درج کردن $A[K]$ در مکان صحیحش در زیر آرایه مرتب شده $A[0], A[1], \dots, A[K-1], A[K], A[1], \dots, A[N-1]$ است. این کار با مقایسه $A[K]$ با $A[0], A[1], \dots, A[K-1]$ مقایسه $A[K]$ با $A[0], A[1], \dots, A[K-1]$ است. این کار با مقایسه $A[J]$ با $A[0], A[1], \dots, A[J-1]$ می‌تواند انجام می‌شود، تا اولین عنصری مانند $A[J]$ روبرو شویم که $A[J] \leq A[K]$ است. آنگاه هر یک از عناصرهای $A[0], A[1], \dots, A[J-1], A[J+1], \dots, A[N-1]$ یک مکان به جلو برد و آنگاه $A[K]$ در مکان $J+1$ ام آرایه درج می‌شود.

اگر همیشه یک عنصر $A[J]$ وجود داشته باشد طوری که $A[J] \leq A[K]$ الگوریتم ساده می‌شود. در غیر اینصورت باید دائمًا تحقیق کنیم که آیا باید $A[K]$ با $A[J]$ مقایسه شود یا خیر. این شرط را می‌توان با معرفی یک عنصر نگهبان $\infty = A[0]$ یا یک عدد بسیار کوچک انجام داد.

Pass	$A[0]$	$A[1]$	$A[2]$	$A[3]$	$A[4]$	$A[5]$	$A[6]$	$A[7]$	$A[8]$
$K = 1:$	$-\infty$	77	33	44	11	88	22	66	55
$K = 2:$	$-\infty$	77	33	44	11	88	22	66	55
$K = 3:$	$-\infty$	33	77	44	11	88	22	66	55
$K = 4:$	$-\infty$	33	44	77	11	88	22	66	55
$K = 5:$	$-\infty$	11	33	44	77	88	22	66	55
$K = 6:$	$-\infty$	11	33	44	77	88	22	66	55
$K = 7:$	$-\infty$	11	22	33	44	77	88	66	55
$K = 8:$	$-\infty$	11	22	33	44	66	77	88	55
Sorted:	$-\infty$	11	22	33	44	55	66	77	88

شكل ۳-۹ مرتب کردن درجی برای $n = 8$ عنصر

مثال ۹-۴

فرض کنید آرایه A با 8 عنصر به صورت زیر داده شده است.

77, 33, 44, 11, 88, 22, 66, 55

شکل ۹-۳ الگوریتم مرتب کردن درجی را نشان می‌دهد. عناصرهای داخل دایره بیانگر A[K] در هر مرحله از الگوریتم است و پیکان، مکان صحیح درج شدن A[K] را نشان می‌دهد.
بیان رسمی الگوریتم مرتب کردن درجی به شرح زیر است:

Algorithm 9.1: (Insertion Sort) INSERTION(A, N).

This algorithm sorts the array A with N elements.

1. Set $A[0] := -\infty$. [Initializes sentinel element.]
2. Repeat Steps 3 to 5 for $K = 2, 3, \dots, N$:
3. Set TEMP := $A[K]$ and PTR := $K - 1$.
4. Repeat while TEMP < $A[PTR]$:
 - (a) Set $A[PTR + 1] := A[PTR]$. [Moves element forward.]
 - (b) Set PTR := PTR - 1.
- [End of loop.]
5. Set $A[PTR + 1] := TEMP$. [Inserts element in proper place.]
- [End of Step 2 loop.]
6. Return.

ملاحظه می‌کنید که در الگوریتم یک حلقه داخلی وجود دارد که اساساً توسط متغیر PTR کنترل می‌شود.
همچنین یک حلقه خارجی وجود دارد که از K به عنوان اندیس استفاده می‌کند.

پیچیدگی مرتب کردن درجی

f(n) تعداد مقایسه‌های الگوریتم مرتب کردن درجی را می‌توان به سادگی محاسبه کرد. قبل از همه، خاطرنشان می‌کنیم که بدترین حالت وقتی اتفاق می‌افتد که آرایه A به ترتیب عکس باشد و حلقه خارجی بخواهد از حداقل تعداد $K - 1$ مقایسه استفاده کند. از این‌رو:

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

علاوه براین، می‌توان نشان داد که در حلقه داخلی به طور متوسط تقریباً $\frac{1}{2}(K-1)$ مقایسه وجود دارد.
بنابراین، برای حالت میانگین:

$$f(n) = \frac{1}{2} + \frac{2}{2} + \dots + \frac{(n - 1)}{2} = \frac{n(n + 1)}{4} = O(n^2)$$

بنابراین الگوریتم مرتب کردن، وقتی n خیلی بزرگ باشد، الگوریتم خیلی کندی است. نتیجه‌های بالا به طور خلاصه در جدول زیر ارائه شده است:

Algorithm	Worst Case	Average Case
Insertion Sort	$\frac{n(n - 1)}{2} = O(n^2)$	$\frac{n(n + 1)}{4} = O(n^2)$

توجه کنید: برای پیدا کردن مکانی که در آن $A[K], A[1], \dots, A[2]$ در زیر آرایه درج می‌شود به جای یک جستجوی خطی با انجام جستجوی دودویی می‌توان در زمان اجرا صرفه‌جویی کرد. این کار به طور متوسط مستلزم $\log n$ مقایسه، به جای $(K - 1) / 2$ مقایسه است. علاوه براین هنوز هم نیازمند جلوبردن $K / 2$ عنصر هستیم. بدین ترتیب مرتبه پیچیدگی هیچ تغییری نمی‌کند. علاوه براین معمولاً مرتب کردن درجی تنها وقتی بکار گرفته می‌شود که n کوچک باشد، در چنین مواردی، جستجوی خطی تقریباً همان کارایی جستجوی دودویی را دارد.

۹-۴ مرتب کردن انتخابی

فرض کنید آرایه A با n عنصر $A[N], A[1], \dots, A[2]$ در حافظه است. الگوریتم مرتب کردن انتخابی برای مرتب کردن آرایه A به صورت زیر عمل می‌کند. نخست کوچکترین عنصر داخل لیست را پیدا می‌کند و آن را در مکان اول لیست قرار می‌دهد. آنگاه کوچکترین عنصر دوم داخل لیست را پیدا می‌کند و آن را در مکان دوم لیست قرار می‌دهد و الى آخر. به بیان دقیقتر:

مرحله ۱. LOC مکان کوچکترین عنصر لیست N عنصری $A[N], A[1], \dots, A[2]$ را پیدا کنید، همچنین جای A و $A[LOC]$ را عوض کنید. آنگاه $A[1]$ مرتب شده است.

مرحله ۲. مکان LOC کوچکترین عنصر زیر لیست $1 - N$ عنصری $A[N], A[1], \dots, A[3], A[2]$ را پیدا کنید، همچنین جای A و $A[LOC]$ را عوض کنید. آنگاه $A[2], A[1]$ مرتب شده است. چون $A[2] \leq A[1]$.

مرحله ۳. مکان LOC کوچکترین عنصر زیر لیست $2 - N$ عنصری $A[N], A[1], \dots, A[4], A[3]$ را پیدا کنید. همچنین جای A و $A[LOC]$ را عوض کنید. آنگاه $A[3], A[2], A[1]$ مرتب شده است، چون

$$A[2] \leq A[3]$$

.....

.....

مرحله ۱ - N : مکان LOC کوچکترین عنصر از میان دو عنصر $A[N - 1], A[N]$ را پیدا کنید. همچنین جای A و $A[LOC]$ را عوض کنید. آنگاه $A[N], A[1], \dots, A[2], A[1]$ مرتب شده است، چون

$$A[N - 1] \leq A[N]$$

بدین ترتیب A پس از $1 - N$ مرحله مرتب می‌شود.

مثال ۹-۵

فرض کنید آرایه A با ۸ عنصر به صورت زیر داده شده است:

77, 33, 44, 11, 88, 22, 66, 55

با اعمال الگوریتم مرتب کردن انتخابی بر A داده‌های شکل ۹-۴ نتیجه می‌شود.

Pass	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
K = 1, LOC = 4	(77)	33	44	(11)	88	22	66	55
K = 2, LOC = 6	11	(33)	44	77	88	(22)	66	55
K = 3, LOC = 6	11	22	(44)	77	88	(33)	66	55
K = 4, LOC = 6	11	22	33	(77)	88	(44)	66	55
K = 5, LOC = 8	11	22	33	44	(88)	77	66	(55)
K = 6, LOC = 7	11	22	33	44	55	(77)	(66)	88
K = 7, LOC = 7	11	22	33	44	55	66	(77)	88
Sorted:	11	22	33	44	55	66	77	88

شکل ۴-۹ مرتب کردن انتخابی برای $n = 8$ عنصر

ملاحظه می‌کنید که LOC مکان کوچکترین عنصر را در بین $A[N], \dots, A[K+1], A[K]$ در مرحله K به دست می‌دهد. عناصر داخل دایره نشان‌دهنده عناصری هستند که باید جایشان با هم عوض شود. تنها مسئله‌ای که در مرحله K ام باقی می‌ماند تعیین LOC مکان کوچکترین عنصر در بین عناصرهای $A[N], \dots, A[K+1], A[K]$ است. این کار را می‌توان با استفاده از متغیر MIN انجام داد که مقدار جاری کوچکترین عنصر را هنگام خواندن زیرآرایه از $A[N]$ تا $A[K]$ نگه می‌دارد. به طور مشخص، نخست قرار دهید $\text{MIN} := A[K]$ و $\text{LOC} := A[K]$ همچنین به دنبال آن لیست را پیمایش کنید و MIN را با هر عنصر دیگر $A[J]$ به صورت زیر مقایسه کنید:

(الف) اگر $\text{MIN} \leq A[J]$ ، آنگاه تنها به عنصر بعدی بروید.

(ب) اگر $\text{MIN} > A[J]$ ، آنگاه LOC را با دستورهای جایگزینی $\text{LOC} := A[J]$ و $\text{MIN} := A[J]$ تازه کنید. پس از مقایسه MIN با آخرین عنصر $A[N]$ ، MIN حاوی کوچکترین عنصر در میان عناصرهای A[N], ..., A[K+1], A[K] است و LOC حاوی مکان کوچکترین عنصر است.

فرایند بالا را می‌توان به طور مستقل به صورت یک زیربرنامه Procedure بیان کرد:

Procedure 9.2: $\text{MIN}(A, K, N, \text{LOC})$

An array A is in memory. This procedure finds the location LOC of the smallest element among $A[K], A[K+1], \dots, A[N]$.

1. Set $\text{MIN} := A[K]$ and $\text{LOC} := K$. [Initializes pointers.]
2. Repeat for $J = K + 1, K + 2, \dots, N$:
 - If $\text{MIN} > A[J]$, then: Set $\text{MIN} := A[J]$ and $\text{LOC} := J$.
 - [End of loop.]
3. Return.

به سادگی می‌توان الگوریتم مرتب‌کردن انتخابی را به صورت زیر بیان کرد:

Algorithm 9.3: (Selection Sort) SELECTION(A, N)

This algorithm sorts the array A with N elements.

1. Repeat Steps 2 and 3 for K = 1, 2, . . . , N - 1:
2. Call MIN(A, K, N, LOC).
3. [Interchange A[K] and A[LOC].]
Set TEMP := A[K], A[K] := A[LOC] and A[LOC] := TEMP.
[End of Step 1 loop.]
4. Exit.

پیچیدگی الگوریتم مرتب‌کردن انتخابی

نخست توجه کنید که $f(n)$ تعداد مقایسه‌ها در الگوریتم مرتب‌کردن انتخابی مستقل از ترتیب اولیه عناصر است. ملاحظه می‌کنید که $\text{MIN}(A, K, N, LOC)$ به $n - K$ مقایسه احتیاج دارد. به بیان دیگر در مرحله $1, 2, \dots, n-1$ مقایسه انجام می‌شود تا کوچکترین عنصر پیدا شود در مرحله $2, 3, \dots, n-1$ مقایسه انجام می‌شود تا دومین عنصر کوچک پیدا شود و الى آخر. بنابراین:

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

نتیجه بالا به طور خلاصه در جدول زیر ارائه شده است:

Algorithm	Worst Case	Average Case
Selection Sort	$\frac{n(n - 1)}{2} = O(n^2)$	$\frac{n(n - 1)}{2} = O(n^2)$

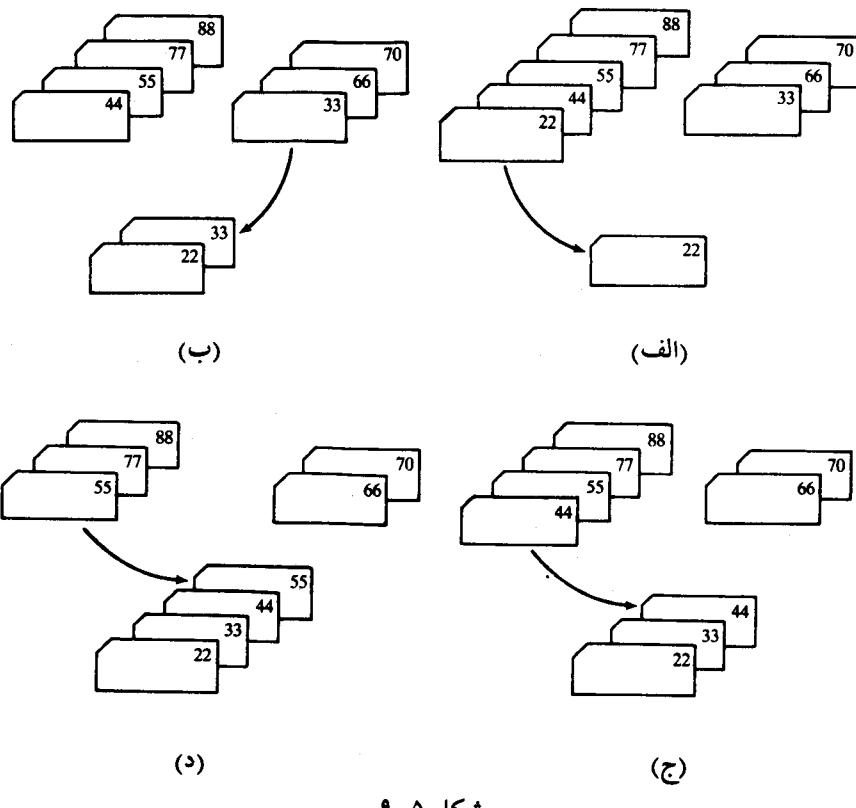
توجه کنید: تعداد جابجایی‌ها و جایگزینی‌ها به ترتیب اولیه عناصر در آرایه A بستگی دارد اما مجموع این عملیات بزرگتر از ضریبی از n^2 نیست.

۹-۵ ادغام‌کردن

فرض کنید A یک لیست مرتب شده با r عنصر و B یک لیست مرتب شده با s عنصر باشد. عمل ترکیب عناصر آرایه A و B در یک لیست مرتب شده C با $r+s = n$ عنصر، ادغام‌کردن نام دارد. یک راه ساده برای ادغام دو آرایه فوق آن است که عناصر B را پس از عناصر A قرار دهیم و آنگاه از یک الگوریتم مرتب‌کردن برای تمام لیست استفاده کنیم. این روش از این واقعیت که A و B به صورت انفرادی مرتب شده هستند استفاده نمی‌کند. الگوریتم ۹.۴ این بخش، الگوریتمی با کارآیی به مراتب بیشتر است. با وجود این، نخست ایده اصلی این الگوریتم را با ذکر دو مثال بیان می‌کنیم.

فرض کنید دو دسته کارت مرتب شده در اختیار دارید. دسته کارتها مانند شکل ۹-۵ درهم ادغام می‌شوند یعنی در هر مرحله دو کارت جلویی با هم مقایسه می‌شوند و کارت با شماره کوچکتر در دسته

ترکیب شده قرار می‌گیرد. هرگاه یکی از دسته‌ها خالی شود، تمام کارتهای باقیمانده در دسته دیگر در انتهای دسته کارت ترکیب شده قرار داده می‌شوند. به همین ترتیب فرض کنید دو صفت از دانشجویان داریم که به ترتیب بزرگی قدشان مرتب شده‌اند و فرض کنید بخواهیم آنها را در یک صفت مرتب شده، در هم ادغام کنیم. صفت جدید در هر مرحله با انتخاب کوتاه‌قدقترین دانشجو از دو دانشجویی که در جلوی صفت خودشان هستند تشکیل می‌شود. هرگاه یکی از صفات، دانشجویی نداشت، صفت دانشجویان باقیمانده در انتهای صفت ترکیب شده قرار می‌گیرند.



شکل ۹-۵

بحث بالا را اکنون به یک الگوریتم رسمی تبدیل می‌کنیم که آرایه r عنصری مرتب شده A و آرایه s عنصری مرتب شده B را در آرایه $r+s$ عنصر ادغام می‌کند. قبل از هرچیز باید توجه داشته باشید که، همواره باید مکان‌های کوچکترین عنصر A و کوچکترین عنصر B را که هنوز در C قرار نگرفته است، نگه داریم. فرض کنید NA و NB به ترتیب نمایش این مکان‌ها باشند. علاوه براین فرض کنید PTR

نمایش مکانی در C باشد که قرار است پر شود. بنابراین در آغاز قرار می‌دهیم $1 := NA$ و $1 := PTR$. در هر مرحله از الگوریتم دو عنصر

$B[NB]$ و $A[NA]$

را مقایسه کنید و عنصر کوچکتر را در $C[PTR]$ قرار دهید. آنگاه با دستور $PTR := PTR + 1$ را واحد افزایش می‌دهیم همچنین یا $NA := NA + 1$ یا $NB := NB + 1$ را با دستور یک واحد افزایش می‌دهیم و این بسته به آن است که آیا عنصر جدید در C از A گرفته شده است یا از B. علاوه براین، اگر $NA > r$ ، آنگاه عنصرهای باقیمانده B در C جایگزین می‌شوند یا اگر $NB > s$ ، آنگاه عنصرهای باقیمانده A در C جایگزین می‌شوند. بیان رسمی این الگوریتم به صورت زیر است:

Algorithm 9.4: MERGING(A, R, B, S, C)

Let A and B be sorted arrays with R and S elements, respectively. This algorithm merges A and B into an array C with $N = R + S$ elements.

1. [Initialize.] Set $NA := 1$, $NB := 1$ and $PTR := 1$.
2. [Compare.] Repeat while $NA \leq R$ and $NB \leq S$:
 - If $A[NA] < B[NB]$, then:
 - (a) [Assign element from A to C.] Set $C[PTR] := A[NA]$.
 - (b) [Update pointers.] Set $PTR := PTR + 1$ and $NA := NA + 1$.
 - Else:
 - (a) [Assign element from B to C.] Set $C[PTR] := B[NB]$.
 - (b) [Update pointers.] Set $PTR := PTR + 1$ and $NB := NB + 1$.
 - [End of If structure.]
- [End of loop.]
3. [Assign remaining elements to C.]
- If $NA > R$, then:
 - Repeat for $K = 0, 1, 2, \dots, S - NB$:
 - Set $C[PTR + K] := B[NB + K]$.
 - [End of loop.]
- Else:
 - Repeat for $K = 0, 1, 2, \dots, R - NA$:
 - Set $C[PTR + K] := A[NA + K]$.
 - [End of loop.]
- [End of If structure.]
4. Exit.

پیچیدگی الگوریتم ادغام کردن

ورویدی الگوریتم ادغام کردن را تعداد کل عنصرهای A و B یعنی $n = r+s$ عنصر تشکیل می‌دهد. هر مقایسه یک عنصر را در آرایه C جایگزین می‌کند که درنهایت n عنصر دارد. بنابراین $f(n)$ تعداد مقایسه‌های نامی تواند بیشتر از n باشد:

$$f(n) \leq n = O(n)$$

به بیان دیگر، الگوریتم ادغام کردن زمان اجرای خطی دارد.

ماتریسهای غیرمنفرد یا وارون پذیر

فرض کنید A ، B و C سه ماتریس باشند، که الزاماً منفرد نیستند. فرض کنید A با r عنصر و کران پائین LBA مرتب شده باشد و B با s عنصر و کران پائین LBB مرتب شده باشد و C کران پائین LBC داشته باشد. آنگاه $UBB = LBB + s - 1$ و $UBA = LBA + r - 1$ به ترتیب کران های بالای A و B باشند. عمل ادغام A و B اکنون با تغییر در الگوریتم بالا به صورت زیر انجام می شود:

Procedure 9.5: MERGE(A , R , LBA , S , LBB , C , LBC)

This procedure merges the sorted arrays A and B into the array C .

1. Set $NA := LBA$, $NB := LBB$, $PTR := LBC$, $UBA := LBA + R - 1$, $UBB := LBB + S - 1$.
2. Same as Algorithm 9.4 except R is replaced by UBA and S by UBB .
3. Same as Algorithm 9.4 except R is replaced by UBA and S by UBB .
4. Return.

ملاحظه می کنید که این زیر برنامه MERGE نامیده شده حال آن که الگوریتم 9.4 نامیده شده است. دلیل بیان این حالت خاص آن است که این زیر برنامه در بخش بعد در مبحث ادغام و مرتب کردن مورد استفاده قرار می گیرد.

جستجوی دودویی و الگوریتم درج کردن

فرض کنید r تعداد عناصر آرایه مرتب شده A خیلی کوچکتر از s تعداد عناصر آرایه مرتب شده B است. عمل ادغام آرایه A و B را می توان به صورت زیر انجام داد. برای هر عنصر $A[K]$ از آرایه A ، برای پیدا کردن مکان صحیح جهت درج کردن $[A[K]]$ در B ، از یک جستجوی دودویی روی B استفاده می کند. هر جستجوی به حداقل $\log s$ مقایسه احتیاج دارد. از این رو این جستجوی دودویی و الگوریتم درج کردن جهت ادغام دو آرایه A و B به حداقل $\log s \cdot \log r$ مقایسه احتیاج دارد. تأکید می کنیم که تنها وقتی که $s < r$ یعنی وقتی r خیلی کوچکتر از s باشد، این الگوریتم بسیار کاراتر از الگوریتم 9.4 الگوریتم ادغام کردن متداول است.

مثال ۹

فرض کنید A دارای 5 عنصر و B دارای 100 عنصر است. آنگاه ادغام A و B با الگوریتم 9.4 تقریباً از 100 مقایسه استفاده می کند. از طرف دیگر، تنها به تقریباً $7 = \log 100$ مقایسه احتیاج دارد تا با استفاده از

جستجوی دودویی مکان صحیح یک عنصر A جهت درج شدن در B تعیین شود. از این‌رو تنها به تقریباً $35 = 5.7$ مقایسه احتیاج است تا عمل ادغام A و B با استفاده از جستجوی دودویی و الگوریتم درج کردن صورت گیرد.

جستجوی دودویی و الگوریتم درج کردن این واقعیت را که A مرتب شده است درنظر نمی‌گیرد. بنابراین، این الگوریتم را می‌توان به دو طریق و به صورت زیر اصلاح کرد. در اینجا فرض می‌شود که A دارای 5 عنصر و B دارای 100 عنصر است.

(۱) کوچک‌کردن مجموعه هدف. فرض کنید پس از جستجوی اول متوجه می‌شویم که A[1] باید پس از B[16] درج شود. آنگاه تنها به استفاده از جستجوی دودویی روی [B[17], B[18], ..., B[100]] احتیاج است تا مکان صحیح جهت درج A[2] پیدا شود. و الی آخر.

(۲) پریدن. مکان موردنظر برای درج A[1] در B نزدیکی [B[20]] است (یعنی $B[s/r]$ نه در نزدیکی [B[50]]). از این‌رو نخست از یک جستجوی خطی روی B[20], B[40], B[60], B[80] و B[100] استفاده می‌کنیم تا B[K] به گونه‌ای تعیین شود که $A[1] \leq B[K]$ و آنگاه از یک جستجوی دودویی روی [B[K - 20], B[K - 19], ..., B[K]] استفاده می‌کنیم. این کار مشابه استفاده از حالت‌های پرش یا انگشتی در یک فرهنگ لغت است که مکان تمام کلماتی را که حرف اولشان یکی است نشان می‌دهد. جزئیات الگوریتم اصلاح شده به عنوان تمرین به دانشجو واگذار می‌شود.

۶-۹ ادغام و مرتب کردن

فرض کنید آرایه A با n عنصر $A[1], A[2], \dots, A[N]$ در حافظه است. الگوریتم ادغام و مرتب کردن که A را مرتب می‌کند نخست با یک مثال مشخص و ساده توضیح داده می‌شود:

مثال ۹-۷

فرض کنید آرایه A حاوی 14 عنصر زیر است:

66, 33, 40, 22, 55, 88, 60, 11, 80, 20, 50, 44, 77, 30

هر مرحله از الگوریتم ادغام و مرتب کردن، از ابتدای آرایه A شروع می‌کند و جفت زیرآرایه‌های مرتب شده را به صورت زیر درهم ادغام می‌کند:
مرحله ۱. هر جفت عناصر داده شده را درهم ادغام کنید تا لیست اعداد دوتایی مرتب شده زیر به دست آید:

33, 66

22, 40

55, 88

11, 60

20, 80

44, 50

30, 77

مرحله ۲. هر جفت اعداد دو تایی بالا را درهم ادغام کنید تا لیست اعداد چهارتایی مرتب شده زیر به دست آید :

22, 33, 40, 66

11, 55, 60, 88

20, 44, 50, 80

30, 77

مرحله ۳. هر جفت اعداد چهارتایی مرتب شده بالا را درهم ادغام کنید تا دو زیرآرایه مرتب شده زیر بدست آید :

11, 22, 33, 40, 55, 60, 66, 88

20, 30, 44, 50, 77, 80

مرحله ۴. دو زیرآرایه مرتب شده بالا را درهم ادغام کنید تا یک آرایه مرتب شده بدست آید.

11, 20, 22, 30, 33, 40, 44, 50, 55, 60, 66, 77, 80, 88

اکنون آرایه اولیه A مرتب شده، است.

الگوریتم ادغام و مرتب کردن بالا برای مرتب کردن آرایه A دارای خاصیت مهم زیر است. پس از مرحله K، آرایه A به دو زیرآرایه مرتب شده تجزیه می شود که در آن هر زیرآرایه بجز احتمالاً زیرآرایه آخری، حاوی دقیقاً $L = 2^K$ عنصر است. از این رو این الگوریتم به حد اکثر logn مرحله جهت مرتب کردن آرایه n عنصری A احتیاج دارد.

توصیف غیررسمی بالا از ادغام و مرتب کردن، اکنون به یک الگوریتم رسمی تبدیل می شود که دو قسم تقسیم می شود. قسمت اول یک زیربرنامه Procedure به نام MERGEPASS است که از زیربرنامه 9.5 Procedure برای اجرای یک مرحله از الگوریتم استفاده می کند و قسمت دوم به طور مکرر از MERGEPASS استفاده می کند تا وقتی که A مرتب شود.

زیربرنامه MERGEPASS روی آرایه n عنصری A بکار گرفته می شود که از یک دنباله از زیرآرایه های مرتب شده تشکیل می شود. علاوه بر این، هر زیرآرایه از L عنصر تشکیل شده است به استثنای زیرآرایه آخر، که ممکن است کمتر از L عنصر داشته باشد. با تقسیم n بر $L * 2$ ، Q خارج قسمت به دست می آید که تعداد زوج زیرآرایه مرتب شده L عنصری را به ما می گوید یعنی :

$$Q = \text{INT}(N / (2 * L))$$

از INT(X) برای نمایش مقدار صحیح X استفاده کردیم. قرار دهید $S = 2 * L * Q$ = تعداد کل عناصر در زوج زیرآرایه Q را بدست می آوریم. از این رو $N - S = R$ = تعداد عناصر باقیمانده را نمایش می دهد. این زیربرنامه نخست جفت Q تایی اولیه زیرآرایه های L عنصری را در هم ادغام می کند. آنگاه زیربرنامه حالتی را در نظر می گیرد که در آن تعداد فردی از زیرآرایه وجود دارد (وقتی $R \leq L$) یا در آن زیرآرایه آخری کمتر از L عنصر دارد.

بیان رسمی زیربرنامه MERGEPASS و الگوریتم ادغام و مرتب کردن به صورت زیر است :

Procedure 9.6: MERGEPASS(A, N, L, B)

The N-element array A is composed of sorted subarrays where each subarray has L elements except possibly the last subarray, which may have fewer than L elements. The procedure merges the pairs of subarrays of A and assigns them to the array B.

1. Set Q := INT(N/(2*L)), S := 2*L*Q and R := N - S.
2. [Use Procedure 9.5 to merge the Q pairs of subarrays.]
Repeat for J = 1, 2, ..., Q:
 - (a) Set LB := 1 + (2*J - 2)*L. [Finds lower bound of first array.]
 - (b) Call MERGE(A, L, LB, A, L, LB + L, B, LB).
[End of loop.]
3. [Only one subarray left?] If R ≤ L, then:
Repeat for J = 1, 2, ..., R:
Set B(S + J) := A(S + J).
[End of loop.]
Else:
Call MERGE(A, L, S + 1, A, R, L + S + 1, B, S + 1).
[End of If structure.]
4. Return.

Algorithm 9.7: MERGESORT(A, N)

This algorithm sorts the N-element array A using an auxiliary array B.

1. Set L := 1. [Initializes the number of elements in the subarrays.]
2. Repeat Steps 3 to 6 while L < N:
 3. Call MERGEPASS(A, N, L, B).
 4. Call MERGEPASS(B, N, 2 * L, A).
 5. Set L := 4 * L.
[End of Step 2 loop.]
6. Exit.

از آنجا که می‌خواهیم آرایه مرتب شده نهایتاً در حافظه اولیه A قرار گیرد، باید زیربرنامه MERGEPASS را با تعداد دفعات زوجی اجرا کنیم.

پیچیدگی الگوریتم ادغام و مرتب کردن

فرض کنید $f(n)$ تعداد مقایسه‌هایی باشد که برای مرتب کردن آرایه n عنصری A با استفاده از الگوریتم ادغام و مرتب کردن مرور دنیاز است. یادآور می‌شویم که این الگوریتم به حداقل $\log n$ مرحله نیاز دارد. علاوه بر این هر مرحله مجموعاً n عنصر را درهم ادغام می‌کند. همچنین با بررسی پیچیدگی ادغام کردن، هر مرحله مستلزم حداقل n مقایسه است. بنابراین برای هر دوی بدترین حالت و حالت میانگین داریم:

$$f(n) \leq n \log n$$

ملاحظه می‌کنید که این الگوریتم همان مرتبه **HeapSort** و همان مرتبه میانگین **QuickSort** را دارد. اشکال اصلی که بر ادغام و مرتب کردن گرفته می‌شود، آن است که نیازمند یک آرایه کمکی با n عنصر است. هر یک از الگوریتم‌های مرتب کردن دیگری که مورد مطالعه قرار گرفت، تنها به تعداد محدودی از خانه اضافی احتیاج دارد که مستقل از n است.

نتیجه‌های بالا به طور خلاصه در جدول زیر ارائه شده است :

Algorithm	Worst Case	Average Case	Extra Memory
Merge-Sort	$n \log n = O(n \log n)$	$n \log n = O(n \log n)$	$O(n)$

۹-۷ مرتب کردن مبنایی RADIX

مرتب کردن مبنایی روشی است که افراد بسیاری به طور شهودی از آن استفاده می‌کنند یا هنگامی که لیست بزرگی از اسامی را به صورت الفبایی مرتب می‌کنیم از آن استفاده می‌کنیم. در اینجا مبنا 26 است، علت آن 26 حرف الفبایی زبان انگلیسی است. بطور مشخص لیست اسامی نخست براساس حرف اول هر اسم مرتب می‌شود. به بیان دیگر اسامی به 26 دسته مرتب می‌شوند که در آن دسته اول، از اسامی ای تشکیل می‌شود که حرف اول آنها با A شروع می‌شود، دسته دوم از اسامی ای تشکیل می‌شود که حرف اول آنها با B شروع می‌شود و الی آخر. در طی مرحله دوم، هر دسته براساس حرف دوم اسم به صورت الفبایی مرتب می‌شود و الی آخر. اگر هیچ اسمی برای مثال، بیشتر از 12 حرف نداشته باشد، اسامی حداقل در 12 مرحله به صورت الفبایی مرتب می‌شوند.

مرتب کردن مبنایی روشی است که در کارت مرتب کن مورد استفاده قرار می‌گیرد، یک کارت مرتب کن از 13 صندوق دریافت تشکیل می‌شود که به صورت زیر برچسب‌گذاری شده است :

(ردکن) R , 9 , 8 , 7 , 6 , 5 , 4 , 3 , 2 , 1 , 0 , 11 , 12

هر صندوق به غیر از R متناظر با ردیفی روی یک کارت است که در آن یک سوراخ می‌تواند منگنه شود. اعداد دهدی که در آن مبنا 10 است به صورت ساده و بدیهی منگنه می‌شوند و از این رو تنها از 10 صندوق اول کارت مرتب کن استفاده می‌کند. کارت مرتب کن در اعداد مبنا از مرتب کردن رقم به ترتیب عکس استفاده می‌کند. به بیان دیگر فرض کنید به یک کارت مرتب کن یک مجموعه از کارت‌ها داده شده است که در آن هر کارت شامل یک عدد 3 رقمی است که در ستون‌های 1 تا 3 منگنه شده‌اند. کارتها نخست براساس ارقام یکان مرتب می‌شوند. در مرحله دوم، کارتها براساس رقم دهگان مرتب می‌شوند. در مرحله سوم و آخر، کارتها براساس رقم صدگان مرتب می‌شوند. برای روشن شدن مطلب یک مثال می‌زنیم.

مثال ۹-۸

فرض کنید ۹ کارت به صورت زیر منگنه شده‌اند:

348, 143, 361, 423, 538, 128, 321, 543, 366

با دادن آنها به یک کارت مرتب‌کن، اعداد طی سه مرحله مرتب می‌شوند که مراحل آن در شکل ۹-۶ نشان داده شده است.

Input	0	1	2	3	4	5	6	7	8	9
348									348	
143				143						
361		361								
423				423						
538									538	
128									128	
321		321								
543				543						
366							366			

شکل ۹-۶ (الف) مرحله اول

در مرحله اول، ارقام یگان داخل صندوق مرتب می‌شوند. صندوقها وارونه رسم شده‌اند، از این‌رو ۳۴۸ در پائین صندوق شماره ۸ است. کارتها صندوق به صندوق، از صندوق شماره ۹ تا صندوق شماره ۰ جمع می‌شوند. توجه دارید که ۳۶۱ اکنون پائین این گروه اعداد و ۱۲۸ در بالای گروه اعداد است. کارتها اکنون مجدداً به کارت مرتب‌کن داده می‌شوند.

Input	0	1	2	3	4	5	6	7	8	9
361										
321				321						
143										
423					423					
543										
366										
366										
348										
538										
128										
				128		538				

شکل ۹-۷ (ب) مرحله دوم

در مرحله دوم، ارقام دهگان داخل صندوق مرتب می‌شوند. مجدداً کارتها صندوق به صندوق جمع می‌شوند و دوباره به عنوان ورودی به کارت مرتبکن داده می‌شوند.

Input	0	1	2	3	4	5	6	7	8	9
321				321						
423			128		423					
128		128								
538						538				
143		143					543			
543										
348				348						
361				361						
366				366						

شکل ۷-۹ (ج) مرحله سوم

در مرحله سوم و آخر، ارقام صدگان داخل صندوق مرتب می‌شوند.
هنگامی که کارتها پس از مرحله سوم جمع آوری می‌شوند، اعداد به ترتیب زیر هستند:

128, 143, 321, 348, 361, 366, 423, 538, 543

به این ترتیب کارتها اکنون مرتب هستند.

C تعداد مقایسه‌های موردنیاز برای مرتبکردن n عدد 3 رقمی از این نوع، به صورت زیر محدود است:

$$C \leq 9 * 3 * 10$$

عدد 9 از تعداد نه کارت، 3 از سه رقمی بودن هر عدد و 10 از مبنای d = رقم، گرفته شده است.

پیچیدگی مرتب کردن مبنایی

فرض کنید A لیست n عنصری A_1, A_2, \dots, A_n داده شده است. فرض کنید d نمایش مبنای باشد مثلاً برای ارقام دهدی $10 = d$ ، برای حروفها $26 = d$ و برای بیتها $2 = d$ است، همچنین فرض کنید هر عنصر A_i با s رقم زیر نمایش داده می‌شود:

$$A_i = d_{i1} d_{i2} \dots d_{is}$$

الگوریتم مرتب کردن مبنایی نیازمند s مرحله، یعنی تعداد ارقام هر عنصر است. در مرحله K هر رقم d_{ik} با هر یک از d رقم مقایسه می‌شود. از این رو $C(n)$ تعداد مقایسه‌ها برای الگوریتم به صورت محدود است:

$$C(n) \leq d * s * n$$

اگرچه d مستقل از n است اما s به n بستگی دارد. در بدترین حالت، $s = n$ ، از این‌رو $O(n^2) = O(n \cdot C(n))$ ، از این‌رو $C(n) = O(\log n)$. به بیان دیگر، مرتب‌کردن مبنایی تنها و قبیل خوب اجرا می‌شود که s تعداد ارقام در این نمایش A ‌ها کوچک باشد.

عیب دیگر مرتب‌کردن مبنایی آن است که ممکن است به $n * d$ خانه حافظه احتیاج داشته باشد. این مطلب از این واقعیت ناشی می‌شود که تمام عناصر را می‌توان در طی یک مرحله داده شده به یک صندوق فرستاد. این عیب را می‌توان با استفاده از لیستهای پیوندی به عوض آرایه، به حداقل رساند که عنصرها را در مرحله داده شده ذخیره می‌کند. با وجود این همچنان به $n * 2$ خانه حافظه نیاز‌مندیم.

۹-۸ جستجوی اطلاعات و اصلاح داده‌ها

فرض کنید S یک مجموعه از داده‌ها باشد که به وسیله یک جدول، با استفاده از یک نوع ساختمان داده در حافظه ذخیره می‌شود. جستجوکردن عملی است که **LOC** مکان یکی از اطلاعات داده شده **ITEM** را در حافظه پیدا می‌کند یا پیغامی می‌دهد که آیا **ITEM** در S وجود دارد یا خیر. هرگاه **ITEM** در S وجود داشته باشد جستجو را موفق، در غیراینصورت ناموفق گویند. الگوریتم جستجویی که مورد استفاده قرار می‌گیرد اساساً به نوع ساختمان داده‌ای بستگی دارد که برای ذخیره S در حافظه بکار گرفته می‌شود.

اصلاح داده‌ها : عبارت است از عملیات اضافه کردن، حذف کردن و تازه کردن داده‌ها. در اینجا منظر از اصلاح داده‌ها اساساً اضافه کردن و حذف داده‌ها است. این عملیات ارتباط نزدیکی با جستجوکردن دارند زیرا معمولاً برای حذف باید به جستجوی مکان **ITEM** پرداخت و برای اضافه کردن **ITEM** در جدول باید مکان صحیح آن را برای این منظور جستجو کرد. عملیات اضافه کردن یا حذف نیز به مقدار معینی زمان برای اجرا احتیاج دارد که اساساً به نوع ساختمان داده مورداً استفاده بستگی دارد.

به بیان کلی‌تر، یک توازن بین ساختمان داده‌ای که دارای الگوریتم‌های جستجوی سریع است و ساختمان داده‌ای که دارای الگوریتم‌های اصلاح سریع است وجود دارد. این وضعیت در زیر با مثال توضیح داده شده است که در آن بطور خلاصه جستجوکردن و اصلاح داده‌های سه نوع ساختمان داده، که قبل‌اً در این کتاب مورد مطالعه قرار گرفت ارائه شده است.

(۱) آرایه مرتب شده : در اینجا می‌توان از جستجوی دودویی استفاده کرد که مکان **LOC** یکی از اطلاعات داده شده **ITEM** را با زمان $O(\log n)$ پیدا می‌کند. از طرف دیگر عملیات جستجوکردن و حذف خیلی کند هستند زیرا بطور متوسط باید $O(n / 2)$ عنصر برای اضافه شدن یا حذف معینی جابجا شوند. بدین ترتیب از یک آرایه مرتب شده بهتر است هنگامی استفاده شود که تعداد زیادی عمل جستجو

وجود داشته باشد اما خواسته شود تنها تعداد اندکی داده اصلاح شوند.

(۲) لیست پیوندی : در اینجا می توان تنها جستجوی خطی را انجام داد که مکان LOC یکی از اطلاعات داده شده ITEM را پیدا می کند. این جستجو می تواند خیلی خیلی کند باشد و به زمان $O(n)$ احتیاج دارد. از طرف دیگر، در عملیات اضافه و حذف کردن تنها لازم است تعداد کمی از اشاره گرها تغییر کنند، بدین ترتیب یک لیست پیوندی هنگامی مورد استفاده قرار می گیرد که تعداد زیادی از داده ها اصلاح می شوند مثلًا در پردازش رشته ها.

(۳) درخت جستجوی دودویی : این ساختمان داده مجموعاً مزایای آرایه مرتب شده و لیست پیوندی را دارا است. به عبارت دیگر، عمل جستجو تنها به جستجو یک مسیر معین P در درخت T تقلیل می یابد که به طور متوسط نیازمند تنها $O(\log n)$ مقایسه است. علاوه بر این، درخت T در حافظه بوسیله یک نمایش پیوندی ذخیره می شود، بنابراین پس از پیدا شدن مکان عنصر اضافه شونده و حذف شونده، تنها لازم است چند اشاره گر تغییر کنند. اشکال عمده درخت جستجوی دودویی آن است که درخت ممکن است خیلی نامتوازن Unbalanced باشد، درنتیجه طول مسیر P به عوض $O(n)$ ممکن است (n) باشد، این مطلب جستجو را تقریباً به یک جستجوی خطی تبدیل می کند.

توجه کنید: با استفاده از درخت جستجوی دودویی با ارتفاع متوازن سناریوی بدترین حالت بالا را می توان حذف کرد که پس از هر عمل اضافه و حذف، از تو متوازن می شود. الگوریتم های این گونه تجدید توازن نسبتاً پیچیده است و در محدوده درس ساختمان داده ها قرار ندارد.

جستجو در فایل ها، جستجوی اشاره گرها

فرض کنید فایل F متشکل از رکوردهای R_1, R_2, \dots, R_n در حافظه ذخیره شده است. معمولاً منظور از جستجو در فایل F پیدا کردن مکان LOC رکورد در حافظه با مقدار کلیدی معلوم نسبت به فیلد کلیدی اولیه یا اصلی K است. یک راه ساده برای انجام جستجو استفاده از آرایه مرتب شده ای از اشاره گرها کمکی است که در بخش ۹-۲ توضیح داده شده است. آنگاه با استفاده از جستجوی دودویی به سرعت می توان LOC مکان رکورد با کلید داده شده را پیدا کرد. درحالی که تعداد زیادی عمل حذف و اضافه در فایل انجام می شود، به عوض یک آرایه مرتب شده کمکی می توان از یک درخت جستجوی دودویی کمکی استفاده کرد. در هر حال، جستجو در فایل F به جستجوی مجموعه ای از عناصر S منتهی می شود که در بالا مورد بحث و بررسی قرار گرفت.

۹-۹ درهم سازی HASHING

زمان جستجوی تمام الگوریتم هایی که تاکنون مورد بررسی قرار گرفت بستگی به n یعنی تعداد

عنصر مجموعه داده‌ها، S دارد. این بخش یک روش جستجو موسم به درهم‌سازی یا آدرس دهی درهم‌ساز را توضیح می‌دهد که اساساً مستقل از عدد n است.

اصطلاحی که در ارائه درهم‌سازی بکار می‌رود در ارتباط با مدیریت فایل‌ها است و قبل از همه، فرض می‌کنیم فایل F از n رکورد تشکیل شده با مجموعه‌ای از کلیدها K وجود دارد که به طور منحصر به فرد رکوردهای F را مشخص می‌کنند. در وهله دوم، فرض می‌کنیم که F به وسیله یک جدول T با m خانه حافظه ذخیره می‌شود و فرض می‌کنیم که L مجموعه آدرس خانه‌های حافظه در T است. جهت سهولت در نمادگذاری فرض می‌کنیم که کلیدها در K و آدرسها در L اعداد صحیح (دهدهی) هستند. روش‌های مشابه‌ای وجود دارد که با اعداد صحیح دو دویی یا با کلیدهایی کار می‌کند که، رشته‌های کاراکتری هستند نظیر اسمامی، زیرا در روش استاندارد رشته‌ها را با اعداد صحیح نمایش می‌دهند.

به کمک مثال زیر موضوع درهم‌سازی را توضیح می‌دهیم.

مثال ۹-۹

شرکتی را در نظر بگیرید که 68 کارمند دارد و هر کارمند یک شماره کارمندی 4 رقمی دارد که به عنوان کلید اولیه یا اصلی در فایل کارمندان شرکت استفاده می‌شود. درواقع می‌توانیم از شماره کارمندی به عنوان آدرس رکورد در حافظه استفاده کنیم. برای جستجو به هیچ مقایسه‌ای احتیاج نیست. متأسفانه، این روش نیازمند فضایی به اندازه 10000 خانه حافظه است حال آن که در عمل، از فضایی کمتر از 30 خانه حافظه از این نوع استفاده می‌شود. واضح است که این توازن حافظه برای زمان مقرون به صرفه نیست.

ایده‌کلی استفاده از کلید در تعیین آدرس یک رکورد ایده بسیار جالبی است اما باید طوری اصلاح شود تا فضای زیادی از حافظه به هدر نرود. این اصلاح، شکل یک تابع H از K مجموعه کلیدها، به L مجموعه آدرس‌های حافظه را به خود می‌گیرد. چنین تابعی

$$H: K \rightarrow L$$

یک تابع درهم‌ساز یا یک تابع درهم‌سازی نامیده می‌شود. متأسفانه باید گفت که تابع H مقادیر متمايز و مختلف را نتیجه نمی‌دهد، یعنی ممکن است از دو مقدار کلیدی مختلف K_1 و K_2 یک آدرس درهم‌ساز به دست آید. به این وضعیت تصادم یا برخورد **COLLISION** می‌گویند همچنین باید از روش‌هایی برای حل مسئله برخورد استفاده کرد. بنابراین، مبحث درهم‌سازی به دو قسمت تقسیم می‌شود: (۱) تابع‌های درهم‌ساز و (۲) حل مسئله برخورد. ما این دو قسمت را مستقلاً بررسی می‌کنیم.

تابع‌های درهم‌ساز

دو معیار اصلی که در انتخاب تابع درهم‌ساز $L \rightarrow H:K$ مورد استفاده قرار می‌گیرد به شرح زیر است.

اول از همه، تابع H باید بسیار ساده و هنگام انجام محاسبات، سریع باشد. ثانیاً تابع H باید تاحد ممکن، آدرسها در هم سازی همه عناصر مجموعه \mathcal{A} را بطور یکنواخت توزیع کند طوری که تعداد برخوردها، حداقل باشد. طبیعی است که هیچ تضمینی وجود ندارد که شرط دوم بدون آن که از قبل راجع به کلیدها و آدرسها اطلاعی داشته باشیم بتواند بطور کامل محقق شود. باوجود این، روشهای کلی معینی وجود دارد که در حل مسئله برخورد به ما کمک می‌کند. یکی از این روشهای روش "تجزیه" یک کلید k به قطعه‌های مختلف و ترکیب قطعات به صورتی است که تشکیل آدرس در هم ساز (K) H را بدنه‌ند. اصطلاح "در هم سازی" از این روش "تجزیه" یک کلید به قطعات مختلف گرفته شده است.

واما بعداً چند تابع در هم ساز معروف را معرفی می‌کنیم. تأکید می‌کنیم که هر یک از این تابع‌های در هم ساز می‌توانند بسادگی و با سرعت زیاد در کامپیوتر ارزیابی شوند.

(الف) روش تقسیم. در K عدد m را که بزرگتر از n تعداد کلیدها است انتخاب کنید. m معمولاً طوری انتخاب می‌شود که یک عدد اول یا عددی بدون مقسوم علیه‌های کوچک باشد، زیرا این کار غالباً تعداد برخوردها را به حداقل می‌رساند. تابع در هم ساز H به صورت زیر تعریف می‌شود:

$$H(k) = k \pmod m + 1 \quad \text{یا} \quad H(k) = k \pmod m$$

در اینجا $(\text{mod } m)$ نمایش باقیمانده تقسیم k بر m است. فرمول دوم هنگامی مورد استفاده قرار می‌گیرد که بخواهیم آدرسها در هم سازی بجای تغییر از ۰ تا $m-1$ ، از ۱ تا m تغییر کنند.

(ب) روش میان مربع عدد. کلید k به توان دو می‌رسد. آنگاه تابع در هم ساز H به وسیله

$$H(k) = 1$$

تعریف می‌شود که در آن ۱ با حذف ارقام اول و آخر k^2 به دست می‌آید. تأکید می‌کنیم که برای تمام کلیدها باید از مکان‌های یکسان در k^2 استفاده شود.

(ج) روش تاکردن. کلید k به k_1, k_2, \dots, k_m قسمت تجزیه می‌شود که در آن هر قسمت بجز احتمالاً قسمت آخر، تعداد ارقام مساوی دارند که برای آدرس موردنظر مورد استفاده قرار می‌گیرند. آنگاه این قسمتها با هم جمع شده در حاصل جمع از آخرین رقم انتقالی چشم پوشی می‌شود. به بیان دیگر:

$$H(k) = k_1 + k_2 + \dots + k_r$$

که در آن از سمت چپ ترین رقم انتقالی در صورت وجود، چشم پوشی می‌شود. گاهی اوقات قبل از عمل جمع، برای انجام عملیات بیشتر، ارقام هر یک از قسمتها را k_4, k_2, \dots که دارای شماره زوج هستند مقلوب می‌شوند.

مثال ۹-۱۰

شرکت مثال ۹-۹ را در نظر بگیرید که به هر یک از ۶۸ کارمندش، یک عدد ۴ رقمی منحصر بفرد به

عنوان شماره کارمندی نسبت می‌دهد. فرض کنید L از ۱۰۰ آدرس دورقمی ۹۹,۰۱,۰۰,...,۰۲,۰۱ تشكيل شده است. تابع درهم‌ساز بالا را بر هر یک شماره‌های کارمندی زیر اعمال می‌کنیم:

$$3205, \quad 7148, \quad 2345$$

(الف) روش تقسیم. یک عدد اول m که نزدیک به ۹۹ است مانند $m = 97$ را انتخاب کنید. آنگاه

$$H(3205) = 4, \quad H(7148) = 67, \quad H(2345) = 17$$

یعنی باقیمانده تقسیم عدد ۳۲۰۵ بر ۹۷ برابر ۴، باقیمانده تقسیم ۷۱۴۸ بر ۹۷ برابر ۶۷ و باقیمانده تقسیم عدد ۲۳۴۵ بر ۹۷ برابر ۱۷ است. درحالی که آدرس‌های حافظه به جای ۰۰ با ۰۱ شروع می‌شود، تابع درهم‌ساز ۱ $H(k) = k \pmod{m} + 1$ را طوری انتخاب می‌کنیم تا

$$H(3205) = 4 + 1 = 5, \quad H(7148) = 67 + 1 = 68, \quad H(2345) = 17 + 1 = 18$$

به دست آید.

(ب) روش میان مریع عدد. محاسبات زیر انجام می‌شود:

$k:$	3205	7148	2345
$k^2:$	10 272 025	51 093 904	5 499 025
$H(k):$	72	93	99

ملاحظه می‌کنید که، برای آدرس‌های درهم سازی رقم‌های چهارم و پنجم با شمارش از طرف راست انتخاب شده است.

(ج) روش تاکردن. کلید k را به دو قسمت تجزیه کنید، با جمع این دو قسمت، آدرس‌های درهم‌سازی زیر حاصل می‌شود:

$$H(3205) = 32 + 05 = 37, \quad H(7148) = 71 + 48 = 19, \quad H(2345) = 23 + 45 = 68$$

در $H(7148)$ ملاحظه می‌کنید که از رقم اصلی ۱ چشم پوشی کرده‌ایم. به بیان دیگر، می‌توان قبل از عمل جمع، قسمت دوم تجزیه عدد را به صورت مقلوب نوشت، به این ترتیب، آدرس‌های درهم‌سازی زیر تولید می‌شود:

$$H(3205) = 32 + 50 = 82, \quad H(7148) = 71 + 84 = 55, \quad H(2345) = 23 + 54 = 77$$

حل مسئله برخورد

فرض کنید بخواهیم رکورد جدید R با کلید k را به فایل F اضافه کنیم، اما فرض کنید آدرس خانه حافظه (k) قبلاً اشغال شده است. به این وضعیت برخورد یا تصادم می‌گویند. این بخش دو روش کلی حل مسئله برخورد را توضیح می‌دهد. روش خاصی که دانشجو انتخاب می‌کند بستگی به چند عامل دارد. یک عامل مهم، نسبت n تعداد کلیدها در K (یعنی تعداد رکوردها در F) به m تعداد آدرس‌های درهم‌ساز در L است. این نسبت $\lambda = n/m$ ضریب باردهی نامیده می‌شود.

قبل از هر چیز، نشان می‌دهیم که اجتناب از برخورد تقریباً غیرممکن است. به طور مشخص فرض کنید یک کلاس دارای 24 دانشجو است و فرض کنید جدول، فضایی برای 365 رکورد در اختیار دارد قرار است یک تابع تصادفی درهم‌ساز روز تولد دانشجو را به صورت آدرس درهم‌ساز اختیار کند. هرچند ضریب باردهی $\lambda = \frac{24}{365} \approx 7\%$ بسیار کوچک است اما می‌توان نشان داد احتمال این که بیش از دو دانشجو دارای یک روز تولد باشند پنجاه - پنجاه است.

کارایی یک تابع درهم‌ساز برای حل مسئله برخورد، به وسیله میانگین تعداد جستجوها probes (مقایسه‌های کلیدی) اندازه‌گیری می‌شود که لازمه‌اش تعیین مکان رکورد با کلید k ی داده شده است. اساساً کارایی تابع درهم ساز به ضریب باردهی بستگی دارد. بهویژه این‌که، ما اغلب به دو کمیت زیر علاقمندیم:

$$S(\lambda) = \text{میانگین تعداد جستجوها برای جستجوی موفق}$$

$$U(\lambda) = \text{میانگین تعداد جستجوها برای جستجوی ناموفق}$$

این کمیتها برای روشهای حل مسئله برخورد مورد بررسی قرار خواهند گرفت.

آدرس دهی بازه جستجوی خطی و اصلاح آن

فرض کنید یک رکورد جدید R با کلید k را می‌خواهیم به جدول حافظه T اضافه کنیم اما خانه حافظه با آدرس درهم‌سازی $h = H(k)$ قبلاً پر شده است. روش طبیعی برای حل مسئله برخورد آن است که اولین فضای حافظه موجود بعد از $T[h]$ را در R ، جایگزین کنیم. فرض می‌کنیم که جدول T با m خانه $T[1], T[2], \dots, T[m]$ حافظه به صورت چرخشی یا حلقوی است، طوری که $T[1]$ بعد از $T[m]$ می‌آید. بنابراین با این روش برخورد، برای رکورد R در جدول T از طریق جستجوی خطی مکانهای $[h], [h+1], [h+2], \dots, [m]$ عمل جستجو را انجام می‌دهیم تا رکورد R پیدا شود یا به خانه حافظه خالی برخورد کنیم که بیانگر ناموفق بودن جستجو است.

حل مسئله برخورد به شکل بالا، جستجوی خطی نامیده می‌شود. ثابت می‌شود که میانگین تعداد جستجوها برای یک جستجوی موفق و برای یک جستجوی ناموفق کمیتهای مربوطه زیرند:

$$U(\lambda) = \frac{1}{2} \left(1 + \frac{1}{(1-\lambda)^2} \right) \quad \text{و} \quad S(\lambda) = \frac{1}{2} \left(1 + \frac{1}{1-\lambda} \right)$$

که در اینجا $n/m = \lambda$ ضریب باردهی است.

مثال ۹-۱۱

فرض کنید جدول T دارای 11 خانه حافظه $T[1], T[2], \dots, T[11]$ است و فرض کنید فایل F از 8 رکورد

با آدرس‌های درهم‌سازی زیر تشکیل شده است :

Record: A, B, C, D, E, X, Y, Z
 $H(k)$: 4, 8, 2, 11, 4, 11, 5, 1

فرض کنید 8 رکورد با ترتیب بالا وارد جدول T شده‌اند. آنگاه فایل F به صورت زیر در حافظه ظاهر خواهد شد:

Table T: X, C, Z, A, E, Y, __, B, __, __, D
 Address: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

اگرچه Y تنها رکورد با آدرس درهم‌ساز 5 است اما این رکورد در T[5] جایگزین نمی‌شود چون T[5] قبل توسط E اشغال شده است زیرا قبل از آن در T[4] بخورد وجود داشته است. به همین ترتیب، در T[1] ظاهر نمی‌شود.

S میانگین تعداد جستجوها، در جستجوی موفق به صورت زیر محاسبه می‌شود:

$$S = \frac{1+1+1+1+2+2+2+3}{8} = \frac{13}{8} \approx 1.6$$

U میانگین تعداد جستجوها، در جستجوی ناموفق به صورت زیر محاسبه می‌شود:

$$U = \frac{7+6+5+4+3+2+1+2+1+1+8}{11} = \frac{40}{11} \approx 3.6$$

در حاصل جمع اول تعداد جستجوهایی با هم جمع می‌شوند که هر یک از 8 رکورد را پیدا می‌کند و در حاصل جمع دوم تعداد جستجوهایی با هم جمع می‌شوند که هر خانهٔ خالی مربوط به 11 خانه را پیدا می‌کند.

یکی از عیبهای اصلی جستجوی خطی آن است که وقتی ضریب باردهی بزرگتر از 50 درصد است رکورد تمایل به جمع شدن در یک جا یا تمایل به خوشه شدن Cluster دارند، یعنی رکوردها، در نزدیکی هم ظاهر می‌شوند. این گونه خوشه شدن رکوردها ذاتاً زمان جستجوی میانگین برای یک رکورد را افزایش می‌دهد. دو روش به حداقل رساندن جمع شدن رکوردها در یک جا یا خوشه شدن رکوردها به شرح زیر است :

(۱) جستجوی درجه دوم. فرض کنید Rکورد با کلید k دارای آدرس درهم‌سازی $h = H(k)$ است. آنگاه به جای جستجوی مکانهایی با آدرس‌های $h, h+1, h+2, h+3, \dots, h+i^2, \dots$ را جستجوی خطی می‌کنیم. اگر m تعداد خانه‌های حافظه در جدول T، یک عدد اول باشد، آنگاه دنباله آدرس‌های بالا به نصف خانه حافظه در T دسترسی پیدا خواهد کرد.

(۲) درهم‌سازی مضاعف. درینجا برای حل مسئله بخورد، از تابع درهم‌ساز دوم $H'(k)$ به شرح زیر استفاده می‌شود. فرض کنید Rکورد با کلید k دارای آدرس‌های درهم‌سازی $h = H(k)$ و $h' = H'(k)$ است.

ما مکانهایی با آدرس‌های

$$h, h + h', h + 2h', h + 3h', \dots$$

را جستجوی خطی می‌کنیم. اگر m یک عدد اول باشد، آنگاه دنباله آدرس‌های بالا به تمام خانه‌های حافظه در T دسترسی پیدا خواهد کرد.

توجه کنید: پیاده‌سازی عمل حذف، یکی از عیبهای اصلی هر یک از انواع روش‌های آدرس‌دهی باز است. به طور مشخص، فرض کنید رکورد R از مکان $T[r]$ حذف شده است. پس از آن فرض کنید هنگام جستجوی رکورد دیگر R' را ملاقات کنیم. این کار الزاماً به معنی ناموفق بودن جستجو نیست. بنابراین، هنگام حذف رکورد R ، باید خانه $T[r]$ را برچسب‌گذاری کنیم تا مشخص شود که این خانه حاوی یک رکورد بوده است. بنابراین، هنگامی که فایل F دائمًا دستخوش تغییر است، بشرط آدرس‌دهی باز می‌تواند مورد استفاده قرار گیرد.

زنگیری کردن

زنگیری کردن شامل ذخیره دو جدول در حافظه است. مانند گذشته، قبل از همه فرض می‌کنیم T در حافظه وجود دارد که شامل رکوردهای F است با این تفاوت که T اکنون فیلد اضافی **LINK** دارد که برای این منظور بکار گرفته می‌شود تا تمام رکوردهای داخل T با یک آدرس‌دهی درهم‌ساز h را بتوان به هم پیوند داد تا تشکیل یک لیست پیوندی بدهنند. در مرحله دوم، یک جدول آدرس‌دهی درهم‌ساز **LIST** وجود دارد که شامل اشاره‌گرهای به لیستهای پیوندی داخل T است.

فرض کنید رکورد جدید R با کلید k به فایل F اضافه شده است. R را در اولین مکان حافظه موجود در جدول T قرار می‌دهیم و آنگاه R را با اشاره‌گر $[LIST[H(k)]]$ به لیست پیوندی اضافه می‌کنیم. اگر لیستهای پیوندی رکوردها متغیر شده باشند، آنگاه R به سادگی به ابتدای لیست پیوندی اضافه می‌شود. جستجو برای یک رکورد یا حذف یک رکورد چیزی جزء جستجو یک گره یا حذف یک گره از یک لیست پیوندی نیست که به تفصیل در فصل ۵ مورد بحث و بررسی قرار گرفت. ثابت می‌شود که میانگین تعداد جستجوها، با استفاده از زنگیری کردن، برای یک جستجوی موفق یا برای یک جستجوی ناموفق برابر مقادیر تقریبی زیر است:

$$\frac{1}{1/\lambda) \approx 1 + \frac{1}{2}\lambda} \quad \text{و} \quad S(\lambda) \approx e^{-\lambda} + \lambda$$

دراینجا ضریب باردهی $n/m = \lambda$ می‌تواند بزرگتر از 1 باشد، چون m تعداد آدرس‌های درهم‌ساز L (نه تعداد خانه‌ها در T) ممکن است کوچکتر از n تعداد رکوردها در F باشد.

مثال ۹-۱۲

بار دیگر اطلاعات مثال ۱۱-۹ را در نظر بگیرید که در آن ۸ رکورد با آدرس‌های درهم‌سازی زیر داده شده است:

Record: A, B, C, D, E, X, Y, Z
 $H(k)$: 4, 8, 2, 11, 4, 11, 5, 1

با استفاده از زنجیری کردن، رکوردها مطابق شکل ۷-۹ در حافظه ظاهر می‌شوند.

Table T

The diagram illustrates the mapping of records from the LIST table to the INFO and LINK tables. The LIST table has 12 rows (indices 1 to 11). The INFO table has 15 rows (indices 1 to 14). The LINK table has 15 rows (indices 1 to 14). Arrows show the mapping: LIST[1] points to INFO[1] and LINK[1]; LIST[2] points to INFO[2] and LINK[2]; LIST[3] points to INFO[3] and LINK[3]; LIST[4] points to INFO[4] and LINK[4]; LIST[5] points to INFO[5] and LINK[5]; LIST[6] points to INFO[6] and LINK[6]; LIST[7] points to INFO[7] and LINK[7]; LIST[8] points to INFO[8] and LINK[8]; LIST[9] points to INFO[9] and LINK[9]; LIST[10] points to INFO[10] and LINK[10]; LIST[11] points to INFO[11] and LINK[11].

	LIST	INFO	LINK
1	8	A	0
2	3	B	0
3	0	C	0
4	5	D	0
5	7	E	1
6	0	X	4
7	0	Y	0
8	2	Z	0
9	0		10
10	0		11
11	6		12
			13
			14
			0

AVAIL = 9

شکل ۷-۹

ملاحظه می‌کنید که مکان رکورد R در جدول T به آدرس درهم‌سازی اش وابسته نیست. به سادگی رکورد در اولین گره لیست AVAIL جدول T قرار می‌گیرد. در حقیقت، جدول T نیازمند داشتن تعداد عنصرهای یکسان مانند جدول آدرس دهنده درهم‌ساز نیست.

عیب عمده زنجیری کردن آن است که لازم است $3m$ خانه حافظه برای اطلاعات درنظر گرفته شود. بطور مشخص، m خانه حافظه برای فیلد اطلاعات **INFO**، m خانه برای فیلد پیوندی **LINK** و m خانه برای آرایه اشاره‌گر **LIST** وجود دارد. فرض کنید هر رکورد تنها به یک کلمه **Word** برای فیلد اطلاعات

احتیاج داشته باشد. آنگاه پسندیده‌تر آن است که به منظور حل مسأله برخورد برای جدولی با $3m$ خانه حافظه که ضریب بازدهی $\leq 1/3$ دارد به جای استفاده از روش زنجیری کردن از روش آدرس دهی باز استفاده شود.

مسأله‌های تکمیلی

مسأله ۹-۱ : یک زیربرنامه **RANDOM(DATA,N,K)** بنویسید تا در آرایه **DATA**، **N** عدد صحیح تصادفی بین ۱ تا **K** را جایگزین کند.

مسأله ۹-۲ : مرتب کردن درجی را به یک زیربرنامه **INSERTSORT(A,N)** تبدیل کنید تا آرایه **A** با **N** عنصر را مرتب کند. برنامه را با استفاده از داده‌های زیر اجرا 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 ۹-۳ : مرتب کردن درجی را به یک زیربرنامه **INSERTCOUNT(A,N,NUMB)** تبدیل کنید تا آرایه **A** با **N** عنصر را مرتب کند و **NUMB** تعداد مقایسه‌ها را محاسبه کنید.

مسأله ۹-۴ : یک برنامه **TESTINSERT(N, AVE)** بنویسید که ۵۰۰ بار زیربرنامه **TESTINSERT(N, AVE)** را تکرار کند و **AVE** را میانگین ۵۰۰ مقدار **NUMB** را به دست آورد. از نظر تئوری، $AVE \approx N^{2/4}$ به عنوان هر ورودی از $RANDOM(A, N, 5 * N)$ از مسأله ۹-۱ استفاده کنید. برنامه را با استفاده از $100 = N$ اجرا کنید. درنتیجه از نظر تئوری $AVE \approx N^{2/4} = 2500$ است.

مسأله ۹-۵ : مرتب کردن **QuickSort** را به صورت زیربرنامه **QUICKCOUNT(A, N, NUMB)** بنویسید که آرایه **A** با **N** عنصر را مرتب کند و علاوه براین **NUMB** تعداد مقایسه را بشمارد. (به بخش ۵-۶ رجوع کنید).

مسأله ۹-۶ : یک برنامه **TESTQUICKSORT(N, AVE)** بنویسید که ۵۰۰ بار **TESTQUICKSORT(N, AVE)** را تکرار کند و **AVE** را میانگین ۵۰۰ مقدار **NUMB** را بدست آورد. از نظر تئوری، $AVE \approx N \log_2 N$ به عنوان هر ورودی از $RANDOM(A, N, 5 * N)$ از مسأله ۹-۱ استفاده کنید. برنامه را با استفاده از $100 = N$ آزمایش کنید. درنتیجه از نظر تئوری $AVE \approx 700$ است.

مسأله ۹-۷ : زیربرنامه **Procedure 9.2** را به یک زیربرنامه **MIN(A, LB, UB, LOC)** تبدیل کنید که **LOC** مکان کوچکترین عنصر در بین عناصر $[A[UB], \dots, A[LB + 1], A[LB]]$ را پیدا کند.

مسئله ۹ : مرتب کردن درجی را به صورت یک زیر برنامه **SELECTSORT(A, N)** بنویسید که یک آرایه **N** عنصری را مرتب کند. برنامه را با استفاده از داده‌های زیر آزمایش کنید.

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

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

جستجو کردن، درهم سازی

مسئله ۹ : فرض کنید لیست پیوندی مرتب نشده‌ای در حافظه داده شده است. یک زیر برنامه **Procedure**

SEARCH(INFO, LINK, START, ITEM, LOC)

که (الف) **LOC** مکان عنصر **ITEM** را در لیست پیدا کند یا با هر جستجوی ناموفق قرار دهد = **LOC** **NULL** (ب) در صورت موفق بودن جستجو، جای عنصر **ITEM** را با عنصر ابتدای لیست عرض کند. به چنین لیستی یک لیست خود سازماندهی شده می‌گویند. این لیست دارای این خاصیت است عناصری که اکثر اوقات مورد دسترسی قرار می‌گیرند به طرف ابتدای لیست منتقل می‌شوند.

مسئله ۱۰ : شماره کارمندی ۴ رقمی زیر (مثال ۱۰-۹ را ببینید). درنظر بگیرید :

9614, 5882, 6713, 4409, 1825

با استفاده از (الف) روش تقسیم با $m = 97$ (ب) روش میان مریع عدد (ج) روش تاکردن بدون مقلوب کردن عدد و (د) روش تاکردن با مقلوب کردن عدد، آدرس دهی درهم ساز ۲ رقمی هر شماره کارمندی را بدست آورید.

مسئله ۱۱ : داده‌های مثال ۱۱-۹ را درنظر بگیرید. فرض کنید ۸ رکورد **A,B,C,D,E,X,Y,Z** به ترتیب عکس، به داخل جدول **T** وارد می‌شوند. (الف) نشان دهید که چگونه فایل **F** در حافظه ذخیره می‌شود.

(ب) میانگین تعداد جستجوها برای یک جستجوی موفق و **U** میانگین تعداد جستجوها برای یک جستجوی ناموفق را پیدا کنید. این نتیجه را با نتیجه مثال ۹-۱۱ مقایسه کنید.

مسئله ۱۲ : داده‌های مثال ۱۲-۹ و شکل ۷-۹ را درنظر بگیرید. فرض کنید رکوردهای اضافی زیر به فایل اضافه شده‌اند :

(P, 2), (Q, 7), (R, 4), (S, 9)

در اینجا عنصر سمت چپ زوج مرتب بالا رکورد، و عنصر سمت راست آدرس درهم سازی است. (الف) جدول تازه شده **T** و **LIST** را به دست آورید. (ب) میانگین تعداد جستجوها برای یک جستجوی

موفق و $\lceil \rceil$ تعداد میانگین جستجوها برای یک جستجوی ناموفق را محاسبه کنید.

مسئله ۱۳-۹: یک زیربرنامه **MID(KEY,HASH)** بنویسید که با استفاده از روش میان مربع عدد، آدرس درهم‌سازی 2 رقمی **HASH** یک کلید با شماره کارمندی 4 رقمی را پیدا کنید.

مسئله ۱۴-۹: یک زیربرنامه **FOLD(KEY,HASH)** بنویسید که با استفاده از روش تاکردن با مقلوب کردن عدد، آدرس درهم‌سازی 2 رقمی **HASH** یک کلید با شماره کارمندی 4 رقمی را پیدا کند.

واژه‌نامه ریاضی و کامپیوٹر

A

Absolute value	قدر مطلق
Ackermann function	تابع آکرمان
Adjacency matrix	ماتریس مجاورت - ماتریس همسایگی
Adjacent nodes	گره‌های مجاور
Algebraic expression	عبارت جبری
Algorithm	الگوریتم
Algorithmic notation	نمایش الگوریتمی
Ancestor	جد
Arithmetic expression	عبارت محاسباتی
Array	آرایه
circular	حلقوی
jagged	پله‌ای - دندانه‌ای
multidimensional	چند بعدی
parallel	موازی
Assignment statement	دستور جایگزینی
Attribute	خاصیت - مشخصه
Average case	حالات میانگین

B

Base address	آدرس پایه - آدرس مبنا
Base criteria	معیار پایه‌ای

Base value

Big O notation

Binary search

complexity of

Binary search tree

مقدار پایه

نماد O ای بزرگ

جستجوی دودویی

پیچیدگی

درخت جستجوی دودویی

searching in

Binary tree

complete

depth of

extended

height of

traversing

جستجو در

درخت دودویی

کامل

عمق

گسترش یافته

ارتفاع

پیمایش

Bit matrix

ماتریس بیتی - ماتریس بولین

Boolean matrix

ماتریس بولین

شاخه

Breadth - first search

جستجوی عرضی - جستجوی ردیفی

Bubble sort

مرتب کردن حبابی

complexity of

پیچیدگی

C

Ceiling function

تابع سقف

Chaining

زنگیری کردن

Character set

مجموعه کاراکترها

Character type	نوع کاراکتری
Child	بچه
Circular array	آرایه حلقوی
Circular list	لیست حلقوی
Coding	کدگذاری کردن
Collision resolution	حل مسئله برخورد
Column	ستون
Column - major order	روش ستونی
Complete binary tree	درخت دودویی کامل
Complete graph	گراف کامل
Complexity of algorithm	پیچیدگی الگوریتم
Concatenation	اتصال - پیوند
Conditional flow	جريان شرطی
Connected graph	گراف همبند - گراف متصل
strongly	قوی
Control, flow of	جريان کنترلی
Copying	کپی کردن
Cycle	دور - حلقه

D

Data	داده‌ها - اطلاعات
Data base management	مدیریت بانک اطلاعاتی
Data modification	اصلاح داده‌ها - تغییر داده‌ها
Data structure	ساختمان داده‌ها
Decision tree	درخت تصمیم‌گیری
Degree of a node	درجه یک گره
Deleting	حذف
Dense list	لیست متراکم - لیست فشرده
Depth	عمق

Depth - first search	جستجوی عمقی
Deque	صف دوسره
Descendent	نسل
Diagonal	قطر
Digraph	دایگراف
Directed graph	گراف جهت‌دار
Divide - and - conquer	تقسیم کردن و پیروز شدن
Division method	روش تقسیم
Double hashing	درهای سازی مضاعف

E

Edge	یال
Elementary item	عنصر ابتدایی
Empty	خالی
Empty string	رشته پوج
Entity	موجودیت
Exit	خروج از الگوریتم - خروج
Exponent	نما
Extended binary tree	درخت دودویی گسترش یافته
External	خارجی
External node	گره خارجی
External path length	طول مسیر خارجی

F

Factorial function	تابع فاکتوریل
Father	پدر
Fibonacci sequence	دنباله فیبوناچی
Field	میدان - فیلد
File	فایل - پرونده
File management	مدیریت فایل
File searching	جستجو در فایل
File sorting	مرتب کردن فایل
Finite graph	گراف متناهی

Fixed - length records

رکوردهای با طول ثابت

Fixed - length storage

حافظه با طول ثابت

Floor function

تابع کف

Folding method

روش تاکردن

Free - storage list

لیست حافظه آزاد

Function subalgorithm

زیرالگوریتم تابعی

G**Garbage collection**

جمع آوری حافظه بلااستفاده

General tree

درخت عمومی

representation of نمایش

Generation

نسل

Global variables

متغیرهای سراسری

Graph

گراف

complete کامل

labeled برچسب دار - شمار دار

linked representation of نمایش پیوندی

sequential representation of نمایش ترتیبی

simple ساده

weighted وزن داده شده - وزن دار

H**Hanoi, Towers of**

برج های هانوی

Hash addressing

آدرس دهی درهم ساز

Hash function

تابع درهم ساز

Hashing

درهم سازی

Header linked list

لیست پیوندی دارای سر لیست

Header list, two - way

لیست پیوندی دو طرفه دارای سر لیست

Header node

سرگره - سر لیست

ارتفاع یک درخت

روش هورنر

Huffman's algorithm

الگوریتم هافمن

I**Identifier**

شناسه

شماره شناسایی

ماتریس برخورد

درجه ورودی

شاخص

شاخص گذاری کردن

نمایش میانوندی

نقطه اولیه - نقطه ابتدایی

Inorder threading

نخ کشی میان ترتیبی (Inorder)

مرتب کردن جای دادنی

Integer value مقدار صحیح

Internal node گره داخلی

Internal path length طول مسیر داخلی

Isolated node گره منفرد

Item عنصر

Iteration logic منطق تکرار

J**Jagged array**

آرایه پله ای

- آرایه ای دندانه ای

K**Key**

کلید

Key value

مقدار کلیدی

L**Labeled graph**

گراف برچسب دار

- گراف شماره دار

Leaf

برگ

Left child	بچه چپ	reachability	قابل دسترس
Left subtree	زیردرخت چپ	sparse	خلوت - ثنک
Left successor	ریشه بعدی چپ - گره	triangular	مثاشی
	بعدی چپ	tridiagonal	سه قطعی
Length	طول	weight	وزن
of path	مسیر		
of string	رشته		
Level	سطح		
level number	عدد سطح		
Linear array	آرایه خطی		
deleting from	حذف از		
Linear probing	جستجوی خطی		
Linear search	جستجوی خطی		
Link field	فیلد پیوند - فیلد اتصال		
Linked list	لیست پیوندی		
circular	چرخشی - حلقوی		
copying	کپی کردن		
header	دارای سر لیست		
traversing	پیمایش		
two - way	دو طرفه		
Linked storage	حافظه پیوندی		
free - storage	حافظه آزاد		
Load factor	ضریب باردهی		
Local variables	متغیرهای محلی		
Logic	منطق		
Logical	منطقی		
Loop	حلقه - حلقة تكرار		
Lower bound	کران پائین		
M			
Matrix	ماتریس		
adjacency	مجاورتی - همسایگی		
bit	بیت		
Boolean	بولین		
incidence	برخورد		
path	مسیر		
N			
Neighbor	همسایه		
Nextpointer field	فیلد اشاره گر بعدی		
Node	گره		
external	خارجی		
header	سر لیست		
internal	داخلی		
isolated	منفرد - تنها		
terminal	انتهایی		
Nonlocal variables	متغیرهای غیر محلی		
NULL	پرج		
Null pointer	اشاره گر پرج		

Null string	رشته تهی - رشته بوج	Priority queue	صف اولویت
Null tree	درخت خالی	Probe	جستجو
Number, priority	عدد اولویت	quadratic	درجه دوم
		Procedure	زیربرنامه procedure
O			
<i>O</i> notation	نماد <i>O</i>		
One - dimensional array			
	آرایه یک بعدی		
One - way list	لیست یکطرفه		
Open addressing	آدرس دهنی باز		
	- آدرس دهنی آزاد		
Operations	عملیات		
Outdegree	درجه خروجی		
Overflow	سرربیزی		
stack	پشتہ		
P			
Page	صفحه		
Parallel arrays	آرایه‌های موازی		
Parent node	گره پدر		
Partial ordering	مرتب کردن جزئی		
Pass	گذر - مرحله		
Path length	طول مسیر		
Path matrix	ماتریس مسیر		
Pattern matching	تطبیق الگو		
Permutation	جایگشت		
Pointer	اشاره گر		
Polish notation	نمادگذاری لهستانی		
Polynomial	چند جمله‌ای		
Poset	مجموعه جزئی مرتب		
Postfix notation	نمادگذاری پسوندی		
Postorder traversal	postorder پیمایش		
Primary key	کلید اولیه - کلید اصلی		
Prime number	عدد اول		
Priority number	عدد اولویت		
Q			
Quadratic probing		جستجوی درجه دوم	
Queue		صف	
	priority	اولویت	
R			
Radix sort	مرتب کردن مبنایی		
Reachable in a graph		دسترسی در یک گراف	
Reachability matrix		ماتریس قابل دسترسی	
		نوع اعشاری	
Real type		انتهای صفت	
Rear		رکورد	
Record		با طول ثابت	
	fixed - length	با طول متغیر	
	variable - length	ساختار رکوردی	
Record structure		بازگشتی	
Recursion		بازگشتی	
Recursive procedure		زیربرنامه بازگشتی	
Recursively defined		تعريف شده به صورت بازگشتی	
		آرایه منظم	
Regular array		Heap سازی مجلد	
ReHeap		تابع باقیمانده	
Remainder function		جریان تکراری	
Repetitive flow		جانشینی	
Replacement		بچه راست	
Right son		زیردرخت راست	
Right subtree		گره بعدی راست	
Right successor			

Root	ریشه	پیچیدگی کران پائین
Rooted tree	درخت ریشه‌دار	درخت فراگیر
Row	سطر	مبنا یابی
Row - major order	روش سطري	موضعی
S		
Scalar	اسکالار	ماتریس خلوت
Search	جستجو کردن	ماتریس مربعی
binary	دودویی	پشته
linear	خطی	وضعيت - حالت
sequential	ترتیبی	رشته
Searching	جستجو	گراف همبندی توانی
binary search tree	درخت جستجوی دودویی	اندیس - زیرنویس
files	فایلها	زیردرخت
graph	گراف	گره بعدی
linked list	لیست پیوندی	نماد جمع
pointer	اشارة‌گر	ماتریس متقارن
two - way list	لیست دوطرفه	
selection logic	منطق انتخاب	
Selection sort	مرتب کردن انتخابی	جدول
Sequence logic	منطق توالی دستورات	گره انتهایی
Sequential sort	مرتب کردن ترتیبی	نقطه انتهایی
Shortest - path algorithm	الگوریتم کوتاهترین مسیر	نخ‌کشی
Sibling	همردیف - برادر	درخت نخ‌کشی شده
Side effect	اثر جانبی	Time - space tradeoff
Similar graph	گراف مشابه - گراف مشابه	توازن بین زمان و حافظه
Similar tree	درخت مشابه	بالا
Simple path	مسیر ساده	بالای پشتہ
Son	بچه - پسر	Topological sorting
Sorting	مرتب کردن	مرتب کردن موضعی
bubble	حبابی	برجهای هانوی
insertion	درجی - جای دادنی	بستگی تراکنده ری
lower bound complexity		پیمایش
		درخت دودویی
		گراف
		آرایه خطی

linked list	لیست پیوندی	Variable - length records
Two - way list	لیست دو طرفه	رکوردهای با طول متغیر
Tree	درخت	Variable - length storage
binary	دودویی	حافظه با طول متغیر
decision	تصمیم‌گیری	Root
depth of	عمق	Vertices
general	عمومی	Visiting
height of	ارتفاع	ملاقات - بررسی کردن
threaded	نخ‌کشی شده	
Triangular matrix	ماتریس مثلثی	
Tridiagonal matrix	ماتریس سه قطری	W
Two - dimensional array	آرایه دو بعدی	Warshall's algorithm
Two - way header list		الگوریتم وارشال
	لیست دو طرفه دارای سر لیست	Weight
Two - way list	لیست دو طرفه	ماتریس وزن
Type	نوع	Weighted graph
		- گراف وزن دار
		- گراف وزن داده شده
U		Weighted path length
Underflow	زیر ریزی	طول مسیر وزن داده شده
Unilaterally connected	همبند یک طرفه	Word processing
Upper bound	کران بالا	بردازش رشتہ
		Worst case
V		X
Variable	متغیر	محور X ها
global	سراسری	
local	محلی	
subscripted	اندیس دار	
Variable length	طول متغیر	Y
Variable	متغیر	نتیجه دادن
Z		Z
Zone		ناحیه - منطقه

لیست کتب منتشر شده توسط دانشگاه هومزگان

- ۱۶- مهندسی شکاف های توزیع انرژی الکتریک
 ۱۷- مهندسی مدل سازی اتمالات پلیمر (FACTS)ac (جلد اول)
 ۱۸- مهندسی سازه های خاک
Character Study 2.35
 ۱۹- شوه آنور
 ۲۰- بجهین مهندسی هماش علوم و فنون دریائی و جوی ایران
 ۲۱- نقش هر اهل دیدروبلک در طراس پایا
 ۲۲- انتقاد مهندسی
 ۲۳- زمین شناسی تاریخی (جلد اول)
 ۲۴- مدیریت مؤثر پروره
 ۲۵- وزارت هیئت بریم در سیستمهای فرا آوری مواد
 ۲۶- تئیکت هنرمندی ایجاد مکانیک آزادمانیک
 ۲۷- خنجر منور قدر بهبود
 ۲۸- ساختها و سکانیها
 ۲۹- روش تحقیق و مریج شناسی
 ۳۰- خنجر منور قدر نشست
 ۳۱- مجموعه از خلاصه دروس مهندسی عمران ۴
 ۳۲- اقسام آزمونهای کارشناسی ارزش (دوره موجانی)
 ۳۳- مجموعه مقالات سوین هماش تاریخ و راهنمایی
 ۳۴- مجموعه مقالات همهین هماشی زبان و ادبیات فارسی
 ۳۵- ارتقای روش راهنمایی
 ۳۶- روش همراهی از ایندی و پیغام طولانی سعدی و حافظ
 ۳۷- ابتدا و سطیحی
 ۳۸- طراحی راکتورهای پیمانی
 ۳۹- تجزیه و تحلیل هندسه شیمی
 ۴۰- انتقال چرم و ملکات راحد
 ۴۱- مهندسی سیستمهای توزیع
 ۴۲- طراحی، محاسبات و متون، راه، کتابال، زمین
 ۴۳- انجام خاک و تهیه نمونه های تپه گران
 ۴۴- طراحی سازه های سفت با استفاده از نرم افزار ETAB 2000
 ۴۵- مهندسی انتقال اتمالات پلیمر AC (جلد دوم)
 ۴۶- آنالیز مهندسی
 ۴۷- انتقال چرم و ملکات راحد
 ۴۸- مهندسی سیستمهای توزیع
 ۴۹- اینجا و آینه
 ۵۰- اینجا و آینه
ETABS2000
 ۵۱- مهندسی سازه های انتقال اتمالات پلیمر
 ۵۲- مهندسی های انتقال اتمالات پلیمر AC (جلد دوم)
 ۵۳- اینجا و آینه
 ۵۴- اینجا و آینه
 ۵۵- اینجا و آینه
 ۵۶- اینجا و آینه
 ۵۷- اینجا و آینه
 ۵۸- اینجا و آینه
 ۵۹- اینجا و آینه
 ۶۰- اینجا و آینه
 ۶۱- اینجا و آینه
 ۶۲- اینجا و آینه
 ۶۳- اینجا و آینه
 ۶۴- اینجا و آینه
 ۶۵- اینجا و آینه
 ۶۶- اینجا و آینه
 ۶۷- اینجا و آینه
 ۶۸- اینجا و آینه
 ۶۹- اینجا و آینه
 ۷۰- اینجا و آینه
 ۷۱- اینجا و آینه
 ۷۲- اینجا و آینه
 ۷۳- اینجا و آینه
 ۷۴- اینجا و آینه
 ۷۵- اینجا و آینه
 ۷۶- اینجا و آینه
 ۷۷- اینجا و آینه
 ۷۸- اینجا و آینه
 ۷۹- اینجا و آینه
 ۸۰- اینجا و آینه
 ۸۱- اینجا و آینه
 ۸۲- اینجا و آینه
 ۸۳- اینجا و آینه
 ۸۴- اینجا و آینه
 ۸۵- اینجا و آینه
 ۸۶- اینجا و آینه
 ۸۷- اینجا و آینه
 ۸۸- اینجا و آینه
 ۸۹- اینجا و آینه
 ۹۰- اینجا و آینه
 ۹۱- اینجا و آینه
 ۹۲- اینجا و آینه
 ۹۳- اینجا و آینه
3D Studio MAX
 ۹۴- مهندسی سیستمهای هماشی زبان و ادبیات فارسی
 ۹۵- مجموعه مقالات پیچیدن هماشی زبان و ادبیات فارسی
 ۹۶- راهنمای تکاری کاربردی
 ۹۷- مجموعه مقالات سیهار تابع و ذخیره مدلی دریائی
 ۹۸- مجموعه مقالات همهین هماشی زبان و ادبیات فارسی
 ۹۹- مجموعه مقالات چهارمین هماشی زبان و ادبیات فارسی
 ۱۰۰- اینجا و آینه
 ۱۰۱- شیوه توزیع
 ۱۰۲- مجموعه مقالات پیچیدن هماشی زبان و ادبیات فارسی
 ۱۰۳- مجموعه مقالات سیهار تابع و ذخیره مدلی دریائی
 ۱۰۴- مجموعه مقالات ششمین هماشی زبان و ادبیات فارسی
 ۱۰۵- اینجا و آینه
 ۱۰۶- راهنمای طراحی و پیشکشی برای شناخته دهنده (جلد سوم)
 ۱۰۷- بینایی آینه
 ۱۰۸- اینجا و آینه
 ۱۰۹- AutoCAD 2000 (دوره دوم)
 ۱۱۰- AutoCAD 2000 (دوره جلدی)
3D Studio MAX 3.1
 ۱۱۱- خودآموز کامل (دوره دو)
3D Studio MAX 3D (جلد اول)
 ۱۱۲- تئوری مدل سازی (دوره اول)
 ۱۱۳- تئوری مهندسی مدار مازمان
 ۱۱۴- فرآنهنگ موضعی غزل های سعدی
 ۱۱۵- زمین شناسی مهندسی
- ۱- هنری نیومن
 ۲- مقاومت شناورها
 ۳- تماذل و پایداری شناورها
Chemical Reaction Engineering
 ۴- مهندسی ایات مصنوعی و استرات سلوار
 ۵- مهندسی مانشهای الکترونیک
Unit-Operations of Chemical Engineering
 ۶- عملیات واحد (چاپ چشم)
 ۷- Heat Transfer - ۱۰
 ۸- مهندسی مدل سازی (دوره سه جلدی) (چاپ دو)
 ۹- مهندسی مدل سازی (چاپ چشم)
 ۱۰- اصول ساختمندان دادها (چاپ هفتم)
 ۱۱- دارس و تهمام زبانی
 ۱۲- راهنمایی مکانیکی لایهند (جلد اول)
MS DOS
 ۱۳- اتوکد ۱۳ در محیط ویندوز (دوره دو جلدی)
 ۱۴- اتوکد ۲۰ در محیط ویندوز (دوره پنجم)
 ۱۵- اتوکد ۲۱ در محیط ویندوز (دوره پنجم)
 ۱۶- اتوکد ۲۲ در محیط ویندوز (دوره پنجم)
 ۱۷- اتوکد ۲۳ در محیط ویندوز (دوره پنجم)
 ۱۸- اتوکد ۲۴ در محیط ویندوز (دوره پنجم)
 ۱۹- اتوکد ۲۵ در محیط ویندوز (دوره پنجم)
 ۲۰- اتوکد ۲۶ در محیط ویندوز (دوره پنجم)
3D Studio
 ۲۱- ترسیم دو بعدی و سه بعدی (دوره پنجم)
HCMOS
 ۲۲- مادن های الکترونیک (چاپ دوم)
 ۲۳- توامس کارلابل
 ۲۴- شگردی های استریووگرافی (چاپ سوم)
2000 Solved problems in Electronics
 ۲۵- مادن های استریووگرافی سه بعدی (دوره پنجم)
2000 Solved problems in Electromagnetics
 ۲۶- مادن های استریووگرافی (چاپ سوم)
 ۲۷- اتوکد در محیط ویندوز (چاپ سوم)
 ۲۸- TTL ۱- ۳۸ (چاپ سوم)
 ۲۹- TTL ۲- ۳۹ (چاپ دوم)
 ۳۰- CMOS ۱- ۴۰ (چاپ سوم)
 ۳۱- CMOS ۲- ۴۱ (چاپ دوم)
 ۳۲- مکانیک سیالات
 ۳۳- اینجا و آینه
 ۳۴- مادن های استریووگرافی (چاپ سوم)
 ۳۵- مادن های استریووگرافی (چاپ سوم)
 ۳۶- شنیدن در رویه های پرورست
 ۳۷- اتوکد در محیط ویندوز (چاپ سوم)
 ۳۸- TTL ۱- ۳۸ (چاپ سوم)
 ۳۹- TTL ۲- ۳۹ (چاپ دوم)
 ۴۰- CMOS ۱- ۴۰ (چاپ سوم)
 ۴۱- CMOS ۲- ۴۱ (چاپ دوم)
 ۴۲- مادن های استریووگرافی (چاپ سوم)
 ۴۳- اینجا و آینه
 ۴۴- اینجا و آینه
 ۴۵- اینجا و آینه
 ۴۶- اینجا و آینه
 ۴۷- اینجا و آینه
 ۴۸- آسان یادگیری
 ۴۹- راهنمای کاربردی MAX 3D Studio (دوره در)
 ۵۰- راهنمای کاربردی (چاپ دوم)
 ۵۱- کلید مهندسی هوان (سازه- خاک و پیره) (چاپ سوم)
 ۵۲- شبکه اینترنت در ایران و جهان (چاپ دوم)
 ۵۳- میلیات انتقال چرم
 ۵۴- توپیشی اکتشافی
 ۵۵- شنیدن معنی - دفتر اول
 ۵۶- ناکس پرس
 ۵۷- سایبان ۱
 ۵۸- سایبان ۲
 ۵۹- کاربرد آمار در مدیریت و انتقاد (چهار گزینه ای)

DATA STRUCTURES

حاوی مطالعه در زمینه

- نمادهای ریاضی و تابعهای کامپیوتری
- پیچیدگی الگوریتمها
- پردازش رشته‌ها
- آرایه‌ها، رکوردها و اشاره‌گرها
- لیستهای پیوندی
- لیستهای پیوندی دارای سرلیست
- لیستهای دو پیوندی
- پشتۀا، صفحه‌ها، زیربرنامه‌های بازگشتنی
- درختها
- پیمايش درختهای دودویی
- درختهای جستجوی دودویی
- درخت HEAPSORT، HEAP
- الگوریتم هافمن
- گرافها و کاربردهای آن
- نمایش گرافها در حافظه کامپیوتر
- الگوریتم و ارشال، کوتاهترین مسیر
- مرتب کردن اطلاعات و جستجوی آنها
- انواع روش‌های مرتب سازی
- روش‌های HASHING
- ادغام و مرتب کردن اطلاعات



دانشگاه هرمسدان