به نام خدا

ساختمان داده

آرش شفیعی

کتاب‌های مرجع

- مقدمه‌ای بر الگوریتم‌ها از کرمن، لایسرسون، ریوست، و استاین [1]
- طراحی الگوریتم از کلاینبرگ و تاردوس [2]
- هنر برنامه نویسی از دونالد کنوث [3]

[1] Introduction to Algorithms, by Cormen, Leiserson, Rivest, and Stein
[2] Algorithm Design, by Jon Kleinberg and Eva Tardos
[3] The Art of Computer Programming, by Donald Knuth

مقدمه

مقدمه

- یک الگوریتم [1] یک روند محاسباتی [2] است که مقادیری را به عنوان ورودی [3] دریافت کرده و مقادیری را به عنوان خروجی [4] تولید می‌کند.
- بنابراین یک الگوریتم دنباله‌ای است از گام‌های محاسباتی که ورودی را به خروجی تبدیل می‌کند.

[1] algorithm
[2] computational procedure
[3] input
[4] output

مقدمه

- یک ساختمان داده [1] یا ساختار داده یا داده ساختار روشی است برای سازمان دادن و ذخیره کردن داده‌ها به طوری که دسترسی و تغییر داده‌ها تسهیل شود.
- داده‌های ورودی یک الگوریتم توسط یک ساختمان داده معین به الگوریتم داده می‌شوند و همینطور داده‌های خروجی توسط یک ساختمان داده تعیین شده از الگوریتم دریافت می‌شوند.
- استفاده از ساختمان داده‌های مناسب در طراحی یک الگوریتم اهمیت بالایی دارد.

[1] data structure

مقدمه

- کلمهٔ الگوریتم از نام دانشمند ایرانی محمدبن موسی الخوارزمی گرفته شده است. خوارزم منطقه‌ای است در آسیای مرکزی که در حال حاضر در ازبکستان و ترکمنستان قرار دارد و در کنار دریاچهٔ آرال (دریاچهٔ خوارزم) قرار گرفته است.

- خوارزمی کتاب الجبرو المقابله را نیز به تألیف رسانده است که کلمه جبر [1] در زبان انگلیسی نیز از همین کتاب گرفته شده است. جبر در عربی معنای شکسته بندی است و مقابله به معنی در مقابل یکدیگر قرار دادن است. احتمالا به دلیل این که معادلات جبری متغیرهای متفاوت را جمع آوری می‌کنند و در مقابل یکدیگر قرار می‌دهند، به علم حل معادلات، جبر و مقابله گفته می‌شده است. خوارزمی روش‌هایی برای حل معادلات جبری در کتاب خود ابداع کرده است که به روش‌های الخوارزمی و بعدها در غرب به روش‌های الگوریتمی معروف شده است.

[1] Algebra

مقدمه

- تا سال ۱۹۵۰ کلمهٔ الگوریتم بیشتر برای الگوریتم اقلیدس [1] برای پیدا کردن بزرگ‌ترین مقسوم‌علیه مشترک [2] دو عدد به‌کار می‌رفت که در کتاب اصول اقلیدس [3] توصیف شده است.

- الگوریتم پیدا کردن بزرگ‌ترین مقسوم‌علیه مشترک را می‌توانیم به صورت زیر وصف کنیم.
  ۱. (پیدا کردن باقیمانده.) عدد m را بر n تقسیم می‌کنیم. فرض کنید باقیمانده r باشد خواهیم داشت $0 \leqslant r < n$
  ۲. (آیا باقیمانده صفر است؟) اگر $r = 0$ ، الگوریتم پایان می‌یابد و n جواب مسئله است.
  ۳. (کاهش.) قرار می‌دهیم $m \leftarrow n$ و $n \leftarrow r$ و به مرحلهٔ ۱ می‌رویم.

---
[1] Euclid's algorithm
[2] Greatest common divisor
[3] Euclid's Element

- الگوریتم در واقع یک روند [1] یا دستورالعمل [2] برای حل یک مسئلهٔ محاسباتی است.
- به طور غیر رسمی می‌توانیم بگوییم یک الگوریتم در واقع یک روند محاسباتی گام‌به‌گام است که مجموعه‌ای از مقادیر را که ورودی الگوریتم نامیده می‌شوند دریافت می‌کند و مجموعه‌ای از مقادیر را که خروجی الگوریتم نامیده می‌شوند در زمان محدود تولید می‌کند. بنابراین یک الگوریتم دنباله‌ای است از گام‌های محاسباتی که ورودی‌ها را به خروجی تبدیل می‌کند.

[1] procedure
[2] recipe

مقدمه

- می‌توان گفت یک الگوریتم ابزاری است برای حل یک مسئله محاسباتی معین.
- یک مسئله با تعدادی گزاره رابطهٔ بین ورودی‌ها و خروجی‌ها را در حالت کلی مشخص می‌کند. یک نمونه از مسئله، در واقع با جایگذاری اعداد و مقادیر برای مسئله کلی به دست می‌آید. یک الگوریتم روشی گام‌به‌گام را شرح می‌دهد که با استفاده از آن در حالت کلی برای همهٔ نمونه‌های یک مسئله، خروجی‌ها با دریافت ورودی‌ها تولید شوند. بنابراین روند یک الگوریتم در رابطهٔ بین ورودی‌ها و خروجی‌ها صدق می‌کند.
- به عنوان مثال، فرض کنید می‌خواهید دنباله‌ای از اعداد را با ترتیب صعودی مرتب کنید. این مسئله که مسئله مرتب سازی [1] نام دارد، یک مسئله بنیادین در علوم کامپیوتر به حساب می‌آید که منشأ به وجود آمدن بسیاری از روش‌های طراحی الگوریتم نیز می‌باشد.

[1] sorting problem

مقدمه

- مسئله مرتب سازی را به طور رسمی به صورت زیر تعریف می‌کنیم.
- ورودی مسئله مرتب سازی عبارت است از دنباله‌ای از n عدد به صورت $\langle a_1, a_2, ..., a_n \rangle$ و خروجی مسئله عبارت است از دنباله‌ای به صورت $\langle a'_1, a'_2, ..., a'_n \rangle$ که از جابجا کردن عناصر دنبالهٔ ورودی به‌دست آمده است به طوری که $a'_1 \leq a'_2 \leq ... \leq a'_n$.
- بنابراین به ازای دنباله ورودی $\langle 58, 42, 36, 42 \rangle$ دنباله خروجی $\langle 36, 42, 42, 58 \rangle$ جواب مسئله است.
- یک نمونه از یک مسئله [1] تشکیل شده است از یک ورودی معین و شرح ویژگی خروجی مسئله. بنابراین دنبالهٔ ورودی $\langle 58, 42, 36, 42 \rangle$ به علاوه شرح مسئله مرتب سازی یک نمونه از مسئلهٔ مرتب سازی نامیده می‌شود.

[1] instance of a problem

مقدمه

- بنابراین به طور خلاصه، یک مسئله تشکیل شده است از (۱) توصیفی از چند پارامتر (متغیر آزاد)، که ورودی‌های مسئله نامیده می‌شوند و (۲) گزاره‌هایی برای بیان رابطهٔ ورودی‌ها و مقادیر خروجی (جواب) مسئله، یا به عبارت دیگر ویژگی‌هایی که جواب مسئله دارد.

- یک پارامتر کمیتی است که مقدار آن مشخص نشده و توسط حروف و یا کلمات، نامی بر آن نهاده شده است.

- یک نمونهٔ مسئله با تعیین مقادیر پارامترهای مسئله به دست می‌آید.

- یک الگوریتم، روندی گام به گام است برای پیدا کردن جواب یک مسئله است.

مقدمه

- سرعت اجرای مسئله مرتب سازی به اندازه ورودی یعنی تعداد عناصر دنباله نامرتب و روند الگوریتم بستگی دارد.
- الگوریتم‌های زیادی برای حل مسئله مرتب سازی وجود دارند که هر کدام می‌توانند مزایا و معایبی داشته باشند. به طور مثال یک الگوریتم از میزان حافظهٔ بیشتری استفاده می‌کند، اما زمان کمتری برای محاسبه نیاز دارد و الگوریتم دیگر با میزان حافظهٔ کمتر در زمان بیشتری محاسبه می‌شود که به فراخور نیاز می‌توان از یکی از الگوریتم‌ها استفاده کرد.
- عوامل دیگری مانند معماری کامپیوتر، نوع پردازنده و میزان حافظه نیز در زمان اجرای یک الگوریتم مؤثرترند اما این عوامل فیزیکی هستند و صرف نظر از عوامل فیزیکی می‌توان الگوریتم‌ها را از لحاظ میزان حافظه مورد نیاز و زمان اجرا با یکدیگر مقایسه کرد.

مقدمه

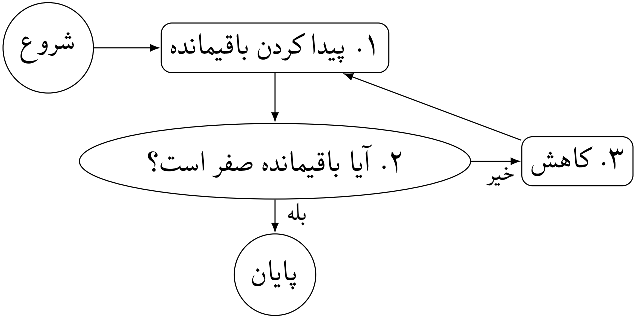
- یک الگوریتم برای یک مسئله محاسباتی درست است اگر به ازای هر نمونه از مسئله که با تعدادی ورودی معین شده است، (۱) توقف کند، بدین که در زمان محدود به اتمام برسد و (۲) خروجی تعیین شده توسط شرح مسئله را تولید کند. می‌گوییم یک الگوریتم درست یک مسئلهٔ محاسباتی را حل می‌کند.
- یک الگوریتم نادرست ممکن است به ازای برخی از ورودی‌ها توقف نکند یا ممکن است به ازای برخی از ورودی‌ها خروجی نادرست تولید کند.
- یک الگوریتم را می‌توان با استفاده از یک زبان طبیعی مانند فارسی یا انگلیسی توصیف کرد و یا برای توصیف آن از یک برنامهٔ کامپیوتری یا یک زبان ساده شده مانند فلوچارت یا شبه‌کد استفاده کرد. تنها نیازمندی یک الگوریتم توصیف دقیق گام‌های الگوریتم است و زبان مورد استفاده برای توصیف اهمیتی ندارد.

- یک الگوریتم را به صورت یک فلوچارت [1] می‌توانیم رسم کنیم.
- یک فلوچارت یا روندنما نموداری است که روند انجام کاری را نشان می‌دهد.
- یک فلوچارت،الگوریتم را به صورت تصویری به نمایش می‌گذارد. در یک فلوچارت معمولاً برای گام‌های محاسباتی از مستطیل و برای گام‌های شرطی از بیضی یا لوزی استفاده می‌شود. همچنین در گام‌هایی که ورودی از کاربر گرفته می‌شود یا خروجی برای نمایش به کاربر چاپ می‌شود از متوازی‌الضلاع استفاده می‌شود. هر گام به گام بعدی توسط یک علامت فلش متصل می‌شود. شروع و پایان را معمولا با دایره نشان می‌دهند.

[1] flowchart

مقدمه

- برای مثال الگوریتم اقلیدس را می‌توان به صورت زیر رسم کرد.



مقدمه

- در الگوریتم‌ها معمولا از علامت $\leftarrow$ یا $:=$ برای عملیات انتساب استفاده می‌شود. برای مثال $m \leftarrow n$ یعنی m را با مقدار فعلی n مقدار دهی می‌کنیم.

- معمولاً از علامت $==$ یا $=$ برای تساوی استفاده می‌شود. برای مثال می‌توانیم بپرسیم آیا مقدار m برابراست با مقدار n و برای مثال می‌نویسیم اگر $m == n$ به مرحله بعد می‌رویم.

- به عنوان مثال دیگر، برای افزایش مقدار یک متغیر به اندازهٔ یک واحد می‌نویسیم $n \leftarrow n + 1$ یعنی مقدار n برابراست با مقدار فعلی n به علاوهٔ یک. معمولاً این عبارت را به این صورت می‌خوانیم : مقدار n برابر می‌شود با $n + 1$.

- در نشانه گذاری ریاضی معمولاً دنباله‌ها را با استفاده از اندیس‌ها نمایش می‌دهیم برای مثال دنباله $v_1, v_2, \cdots, v_n$ یک دنباله از n متغیر است. در الگوریتم‌ها معمولاً از عملگر زیرنویس [1] که با دو براکت باز و بسته [ ] نمایش داده می‌شود استفاده می‌کنیم. بنابراین i امین عنصر دنبالهٔ $v_1, \cdots, v_n$ را به صورت $v[i]$ نمایش می‌دهیم.

[1] subscript

مقدمه

- الگوریتم‌ها در زمینه‌های زیاد و متنوعی کاربرد دارند.

- به عنوان مثال در پروژه ژنوم‌های انسانی هدف پیدا کردن الگوهای ژن‌ها در دی‌ان‌ای [1] انسان است که برای این کار از الگوریتم‌های کامپیوتری استفاده می‌شود. به عنوان چند مثال دیگر می‌توان از الگوریتم کوتاهترین مسیر برای مسیریابی بسته‌های اینترنتی در شبکه‌های کامپیوتری، الگوریتم‌های رمز نگاری برای تبادل امن اطلاعات، الگوریتم‌های تخصیص منابع و زمانبندی در کاربردهایی مانند زمانبندی پروازها و تخصیص خلبان و خدمه به هواپیماها با کمترین هزینه ممکن و الگوریتم‌های فشرده سازی داده‌ها نام برد.

- معمولاً یک مسئلهٔ محاسباتی راه حل‌های زیادی دارد که بنابر معیارهای مورد اهمیت برای استفاده کنندهٔ الگوریتم، الگوریتمی انتخاب می‌شود که در یک یا چند معیار مورد نظر بهترین باشد. برای مثال یک الگوریتم ممکن است در زمان کمتری اجرا شود ولی حافظه بیشتری اشغال کند و الگوریتم دیگر به حافظه کمتری نیاز داشته باشد اما در زمان بیشتری اجرا شود.

[1] DNA

مقدمه

- دسته‌ای از مسئله‌های محاسباتی وجود دارند که گرچه برای محاسبهٔ آنها الگوریتم وجود دارد ولی هیچ یک از الگوریتم‌های موجود نمی‌توانند مسئله را در زمان معقول حل کنند. منظور از زمان معقول زمانی است که آنقدر زیاد نباشد که حل آن مسئله در آن مقدار زمان بی‌معنی شود و دریافت جواب پس از آن زمان بی استفاده باشد. بعدها این مفهوم معقول را به طور رسمی و دقیق تعریف خواهیم کرد.
- این دسته از مسئله‌ها ان‌پی کامل [1] نامیده می‌شوند. گرچه برای این دسته از مسائل هیچ الگوریتمی در زمان معقول پیدا نشده است، اما هیچ‌کس نیز اثبات نکرده است که برای آنها نمی‌توان الگوریتمی پیدا کرد. بنابراین هیچ‌کس نمی‌داند آیا برای مسائل ان‌پی کامل الگوریتم کارامد وجود دارد یا خیر.

[1] NP-Complete problems

مقدمه

- یک ویژگی دیگر مسائل ان‌پی کامل این است که اگر برای یکی از آنها الگوریتم کارامد پیدا شود، برای همهٔ آنها الگوریتم کارامد پیدا خواهد شد چرا که این مسائل قابل تبدیل به یکدیگرند.
- فرض کنید یک مسئله جدید به ما داده شده است. ابتدا تلاش می‌کنیم برای آن مسئله یک الگوریتم کارامد پیدا کنیم. چنانچه نتوانستیم برای آن الگوریتمی کارامد پیدا کنیم، می‌توانیم سعی کنیم تا اثبات کنیم که مسئله ان‌پی کامل است.
- گرچه برای مسئله‌های ان‌پی کامل الگوریتم دقیق کارامد پیدا نشده است، ولی الگوریتم‌های تقریبی [1] زیادی وجود دارند که خروجی قابل قبولی نزدیک به خروجی مورد انتظار در زمان معقول تولید می‌کنند.

[1] Approximation algorithms

مقدمه

- در سالیان قبل با پیشرفت تکنولوژی سرعت پردازنده‌ها افزایش می‌یافت. در سال‌های اخیر سرعت پردازنده‌ها به حد فیزیکی خود نزدیک شده است، بدین معنا که از لحاظ فیزیکی امکان افزایش سرعت وجود ندارد. بنابراین در تکنولوژی‌های جدید در یک پردازنده از چند واحد پردازشی یا هسته استفاده می‌شود.
- برای استفادهٔ بهینه از این پردازنده‌های چند هسته‌ای دسته‌ای از الگوریتم‌ها به نام الگوریتم‌های موازی [1] به‌وجود آمده‌اند.
- در بسیاری از الگوریتم‌ها فرض بر این است که ورودی قبل از شروع الگوریتم در دسترس است اما در برخی مواقع، ورودی به مرور زمان وارد می‌شود. برای مثال در یک سیستم عامل واحدهای کاری در هر لحظه ممکن است به وجود بیایند و الگوریتم بر اساس وضعیت موجود باید تصمیم بگیرد چگونه واحدهای کاری را زمانبندی کند. الگوریتم‌هایی که ورودی را به مرور زمان دریافت می‌کنند الگوریتم‌های برخط [2] نامیده می‌شوند.

[1] Parallel algorithms
[2] Online algorithms

مرتب سازی درجی

- یکی از مسائل مهم در علوم و مهندسی کامپیوتر، مسئله مرتب سازی است. یک آرایه از چندین عنصر را در نظر بگیرید. می‌خواهیم عناصر این آرایه را از کوچک به بزرگ مرتب کنیم. به عبارت دیگر اگر آرایه $A = [a_1, a_2, ..., a_n]$ را داشته باشیم، می‌خواهیم عناصر آرایه یعنی a_i ها را به گونه‌ای جابجا کنیم که به ازای هر $1 \leq i < n$ داشته باشیم $a_i \leq a_{(i+1)}$.

مرتب سازی درجی

- یکی از الگوریتم‌های ارائه شده برای این مسئله الگوریتم مرتب سازی درجی [1] است.
- به طور خلاصه این الگوریتم به صورت زیر عمل می‌کند. فرض کنید یک آرایه با n عنصر از درایهٔ ۱ تا درایهٔ k مرتب شده باشد. حال برای مرتب سازی آرایه از درایهٔ ۱ تا درایهٔ k+۱ باید عنصر k+۱ را در بین عناصر ۱ و k طوری قرار دهیم که از عنصر قبلی خود بزرگ‌تر و از عنصر بعدی خود کوچک‌تر باشد. بدین ترتیب آرایه را از درایهٔ ۱ تا k+۱ مرتب کرده‌ایم. این کار را تا جایی ادامه می‌دهیم که کل آرایه مرتب شود.

[1] insertion sort

مرتب سازی درجی

- به طور خلاصه این الگوریتم را می‌توانیم به صورت زیر بنویسیم.

Algorithm Insertion Sort

```
function INSERTION-SORT(A, n)
▷ A is an array of n elements
for i = 2 to n do
    key = A[i]
    j = i - 1
    while  j > 0 and A[j] > key  do
        A[j+1] = A[j]
        j = j-1
    A[j+1] = key
```

مرتب سازی درجی

- این الگوریتم دارای گام‌هایی است که در یک حلقه تکرار می‌شوند تا در نهایت کل آرایه مرتب شود. در هر مرحله اتمام حلقه، قسمتی از آرایه مرتب شده و قسمتی از آرایه نامرتب است و باید در آینده مرتب شود.
- یک ویژگی که قبل و بعد از هر تکرار حلقه درست باشد ثابت حلقه [1] گفته می‌شود.
- برای مثال ثابت حلقه در الگوریتم مرتب سازی درجی این است که زیر آرایه $A[1 : i - 1]$ در هر تکرار حلقه قبل از شروع حلقه مرتب است.
- ثابت‌های حلقه برای اثبات درستی یک الگوریتم به کار می‌روند. کافی است نشان دهیم که این ثابت حلقه قبل از اولین تکرار حلقه درست است و همچنین اگر قبل از یک تکرار حلقه درست باشد، قبل از تکرار بعدی نیز درست است. در این اثبات در واقع از استقرای ریاضی استفاده می‌کنیم. همچنین برای اثبات درستی الگوریتم باید نشان دهیم که حلقه پایان می‌پذیرد.

[1] loop invarint

تحلیل الگوریتم‌ها

- آنالیز الگوریتم یا تحلیل الگوریتم [1] به معنای پیش بینی منابع مورد نیاز برای اجرای یک الگوریتم است. منابع مورد نیاز شامل زمان محاسبات، میزان حافظه، پهنای باند ارتباطی و مصرف انرژی می‌شود.

- معمولا برای یک مسئله الگوریتم‌های متعددی وجود دارند که هر یک می‌تواند از لحاظ تعدادی از معیارهای ارزیابی بهینه باشد.

- برای تحلیل الگوریتم از یک مدل محاسباتی استفاده می‌کنیم. در اینجا از مدل محاسباتی ماشین دسترسی تصادفی [2] استفاده می‌کنیم. در این مدل محاسباتی فرض می‌کنیم زمان مورد نیاز برای اجرای دستورات و دسترسی به حافظه، ثابت و به میزانی معین است.

- دستورات معمول در این مدل محاسباتی شامل دستورات محاسباتی ریاضی (مانند جمع و تفریق و ضرب و تقسیم و باقیمانده و کف و سقف)، دستورات جابجایی داده (مانند ذخیره، بارگیری و کپی) و دستورات کنترلی (مانند شرطی و انشعابی و فراخوانی تابع) می‌شوند.

---

[1] algorithm analysis

[2] random-access machine (RAM)

تحلیل الگوریتم‌ها

- عملیات محاسبه توان جزء دستورات اصلی مدل محاسباتی رم به حساب نمی‌آید، اما بسیاری از ماشین‌ها با عملیات انتقال بیت‌ها در زمان ثابت می‌توانند اعداد توانی را محاسبه کنند.
- همچنین در این مدل، سلسله مراتب حافظه مانند حافظه نهان [1] که در کامپیوترهای واقعی پیاده سازی شده است، وجود ندارد.
- مدل محاسباتی ماشین دسترسی تصادفی یک مدل ساده همانند ماشین تورینگ است که در آن دسترسی تصادفی به حافظه وجود دارد و عملیات ساده تعریف شده‌اند.

[1] cache memory

تحلیل الگوریتم‌ها

- تحلیل الگوریتم‌ها به منظور محاسبهٔ زمان اجرا و میزان حافظه مورد نیاز الگوریتم‌ها به کار می‌رود.
- زمان اجرا و میزان حافظهٔ مورد نیاز یک الگوریتم به ازای ورودی‌های مختلف متفاوت است و این مقادیر بر اساس اندازهٔ ورودی الگوریتم محاسبه می‌شوند.
- زمان اجرا و میزان حافظه مورد نیاز، معیارهایی برای سنجش کارایی الگوریتم‌ها هستند.
- در این قسمت در مورد روش‌های مختلف تحلیل الگوریتم صحبت خواهیم کرد.
- عوامل زیادی در زمان اجرای یک الگوریتم تأثیر می‌گذارند که از آن جمله می‌توان به سرعت پردازنده، کامپایلر استفاده شده برای پیاده سازی الگوریتم، اندازهٔ ورودی الگوریتم و همچنین ساختار الگوریتم اشاره کرد.

تحلیل الگوریتم‌ها

- برخی از این عوامل در کنترل برنامه نویس نیستند. برای مثال سرعت پردازنده عاملی است تأثیرگذار در سرعت اجرا که با پیشرفت صنعت سخت افزار بهبود می‌یابد و در کنترل برنامه نویس نیست. اما ساختار الگوریتم عاملی است که توسط طراح الگوریتم کنترل می‌شود و نقش مهمی در سرعت اجرا دارد.
- صرف نظر از عوامل فیزیکی، می‌توان سرعت اجرای برنامه را تابعی از اندازهٔ ورودی الگوریتم تعریف کرد که تعداد گام‌های لازم برای محاسبهٔ خروجی را بر اساس اندازه ورودی الگوریتم بیان می‌کند.
- تعداد گام‌های یک الگوریتم برای محاسبه یک مسئله به ساختار آن الگوریتم بستگی دارد و تابعی از اندازهٔ ورودی مسئله است.
- البته غیر از اندازهٔ ورودی، ساختار ورودی هم بر سرعت اجرای برنامه تأثیرگذار است. بنابراین سرعت اجرای برنامه را معمولاً در بهترین حالت (یعنی حالتی که ساختار ورودی به گونه‌ای است که الگوریتم کمترین زمان را برای اجرا بر روی یک ورودی با اندازه معین نیاز دارد) و بدترین حالت محاسبه می‌کنیم. همچنین می‌توان زمان اجرای برنامه را در حالت میانگین به دست آورد.

تحلیل الگوریتم‌ها

- یک روش برای تحلیل زمان مورد نیاز برای اجرای الگوریتم مرتب‌سازی، اجرای آن الگوریتم بر روی یک کامپیوتر و اندازه‌گیری زمان اجرا آن است.
- اما این اندازه‌گیری به ماشین مورد استفاده و کامپایلر و زبان برنامه نویسی مورد استفاده و اجرای برنامه‌های دیگر برروی آن ماشین بستگی دارد. نوع پیاده سازی و اندازهٔ ورودی نیز دو عامل دیگر در سرعت اجرای برنامهٔ مرتب سازی است.
- روش دیگر برای محاسبه زمان اجرای الگوریتم مرتب‌سازی، تحلیل خود الگوریتم است. در این روش محاسبه می‌کنیم هر دستور در برنامه چندبار اجرا می‌شوند. سپس فرمولی به دست آوریم که نشان دهندهٔ زمان اجرای برنامه است. این فرمول به اندازهٔ ورودی الگوریتم بستگی پیدا می‌کند ولی عوامل محیطی مانند سرعت پردازنده در آن نادیده گرفته می‌شود. از این روش می‌توان برای مقایسهٔ الگوریتم‌ها استفاده کرد.

تحلیل الگوریتم‌ها

- اندازهٔ ورودی [1] در بسیاری از مسائل مانند مسئله مرتب‌سازی تعداد عناصر تشکیل دهندهٔ ورودی است. در مسئله مرتب‌سازی اندازه ورودی در واقع تعداد عناصر آرایهٔ ورودی برای مرتب‌سازی است.
- در برخی از مسائل اندازهٔ ورودی در واقع تعداد بیت عدد صحیح ورودی است. برای مثال اندازهٔ ورودی مسئله تجزیهٔ یک عدد به عوامل اول، خود عدد ورودی است.
- در برخی مسائل تعداد ورودی‌ها بیش از یک پارامتر است، بنابراین اندازهٔ ورودی به بیش از یک پارامتر بستگی پیدا می‌کند. برای مثال در الگوریتم پیدا کردن کوتاه‌ترین مسیر در یک گراف، اندازهٔ ورودی تعداد رئوس و تعداد یال‌ها است.

[1] input size

تحلیل الگوریتم‌ها

- زمان اجرای [1] یک الگوریتم وابسته به تعداد دستورات اجرا شده و تعداد دسترسی‌ها به حافظه است. در هنگام محاسبات برای تحلیل الگوریتم فرض می‌کنیم برای اجرای یک دستور در برنامه به یک زمان ثابت نیاز داریم. یک دستور در اجراهای متفاوت ممکن است زمان اجرای متفاوتی داشته باشد ولی فرض می‌کنیم خط k ام برنامه، در زمان c_k اجرا شود.

- کل زمان اجرای یک برنامه، مجموع زمان اجرای همهٔ دستورات آن است. دستوری که m بار در کل برنامه تکرار می‌شود و در زمان c_k اجرا می‌شود، در کل به mc_k واحد زمان برای اجرا نیاز دارد.

- معمولاً زمان اجرای یک الگوریتم با ورودی n را با $T(n)$ نشان می‌دهیم.

[1] execution time

تحلیل الگوریتم مرتب‌سازی درجی

- الگوریتم مرتب‌سازی درجی را یک بار دیگر در نظر می‌گیریم.

Algorithm Insertion Sort

```
function INSERTION-SORT(A, n)
▷ A is an array of n elements
for i = 2 to n do
   key = A[i]
   j = i - 1
   while  j > 0 and A[j] > key  do
      A[j+1] = A[j]
      j = j-1
   A[j+1] = key
```

تحلیل الگوریتم مرتب‌سازی درجی

- برای محاسبه زمان اجرای الگوریتم مرتب‌سازی درجی، ابتدا تعداد تکرار هر یک از خط‌های برنامه را می‌شماریم.
- در این برنامه خط ۱ تعداد n بار و خطوط ۲ و ۳ و ۷ هر یک $n-1$ بار تکرار می‌شوند.
- تعداد تکرار خطوط ۴ و ۵ و ۶ به تعداد تکرار حلقه بستگی دارد.
- زمان اجرای یک الگوریتم علاوه بر اندازه ورودی به ساختار ورودی نیز بستگی دارد. در الگوریتم مرتب‌سازی مسلماً مرتب‌سازی یک آرایهٔ مرتب شده از مرتب‌سازی یک آرایهٔ مرتب نشده سریع‌تر انجام می‌شود.

تحلیل الگوریتم مرتب‌سازی درجی

- زمان اجرای یک الگوریتم را معمولا در بهترین حالت و بدترین حالت محاسبه می‌کنیم. در بهترین حالت آرایهٔ ورودی الگوریتم مرتب شده است. بنابراین در بهترین حالت در هر بار اجرای خط ۴ ، برنامه از حلقه `while` خارج می‌شود و بنابراین خط ۴ تعداد $n-1$ بار اجرا می‌شود و خطوط ۵ و ۶ اجرا نمی‌شوند.

- زمان کل اجرای برنامه را می‌توانیم به صورت زیر بنویسیم.

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1) \\
&= (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7)
\end{aligned}
$$

- زمان اجرای این الگوریتم در بهترین حالت را می‌توانیم به صورت $an + b$ بنویسیم به ازای اعداد ثابت a و b و اندازه ورودی n . بنابراین زمان اجرا در این حالت یک تابع خطی[1] از n است.

[1] linear function

تحلیل الگوریتم مرتب‌سازی درجی

- حال زمان اجرای الگوریتم مرتب سازی درجی را در بدترین حالت محاسبه می‌کنیم. در بدترین حالت آرایهٔ ورودی به صورت معکوس مرتب شده است و بنابراین هر یک از عناصر آرایه نیاز به بیشترین تعداد جابجایی دارد.
- در حلقهٔ `while` هر یک از عناصر $A[i]$ باید با همهٔ عناصر $A[1 : i-1]$ مقایسه شود بنابراین حلقه باید تعداد i بار به ازای $2, 3, ..., n$ تکرار شود.
- پس به طور کل خط ۴ باید به تعداد زیر تکرار شود.

$$\sum_{i=2}^{n} i = \Big(\sum_{i=1}^{n} i\Big) - 1 = \frac{n(n+1)}{2} - 1$$

تحلیل الگوریتم مرتب‌سازی درجی

- هر یک از خطوط ۵ و ۶ الگوریتم به ازای $i = 2, 3, ..., n$ تعداد $i - 1$ بار تکرار می‌شود.
- بنابراین برای خطوط ۵ و ۶ تعداد تکرار برابر است با :

$$\sum_{i=2}^{n}(i-1) = \sum_{i=1}^{n-1} i = \Big(\sum_{i=1}^{n} i\Big) - n = \frac{n(n+1)}{2} - n = \frac{n(n-1)}{2}$$

تحلیل الگوریتم مرتب سازی درجی

- زمان اجرای برنامه در بدترین حالت را می‌توانیم به صورت زیر محاسبه کنیم.

$$
\begin{aligned}
T(n) =\ & c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n(n+1)}{2} - 1\right) \\
& + c_5\left(\frac{n(n-1)}{2}\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7(n-1) \\
=\ & \left(\frac{c_4 + c_5 + c_6}{2}\right)n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4 - c_5 - c_6}{2} + c_7\right)n \\
& - (c_2 + c_3 + c_4 + c_7)
\end{aligned}
$$

تحلیل الگوریتم مرتب‌سازی درجی

- بنابراین زمان اجرای الگوریتم مرتب سازی درجی در بدترین حالت را می‌توانیم به صورت $an^2 + bn + c$ بنویسیم به طوری که a و b و c اعداد ثابت و n ورودی برنامه است. پس زمان اجرای الگوریتم در بدترین حالت یک تابع مربعی [1] یا تابع درجه دوم از n است.

[1] quadratic function

تحلیل الگوریتم مرتب‌سازی درجی

- در حالت کلی از آنجایی که تعداد تکرارها در حلقه `while` مشخص نیست، زمان اجرای الگوریتم را می‌توانیم به صورت زیر بنویسیم که در آن t_i تعداد متغیر تکرارهای حلقه `while` است.

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{i=2}^{n} t_i$$
$$+ c_5 \sum_{i=2}^{n} (t_i - 1) + c_6 \sum_{i=2}^{n} (t_i - 1) + c_7(n-1)$$

تحلیل الگوریتم مرتب‌سازی درجی

- معمولاً در تحلیل الگوریتم‌ها، بدترین حالت [1] زمان اجرا را محاسبه می‌کنیم.

- دلیل این امر آن است که زمان اجرا در بدترین حالت در واقع یک کران بالا [2] برای زمان اجرا است و الگوریتم نمی‌تواند به زمانی بیشتر از آن نیاز داشته باشد. پس می‌توانیم تضمین کنیم که الگوریتم در زمانی که در بدترین حالت محاسبه کرده‌ایم اجرا می‌شود. همچنین در بسیاری از مواقع برای بسیاری از الگوریتم‌ها بدترین حالت بسیار اتفاق می‌افتد.

- دلیل دیگر برای تحلیل الگوریتم در بدترین حالت این است که زمان اجرا در بدترین حالت و در حالت میانگین [3] تقریبا معادل یکدیگرند. برای مثال در الگوریتم مرتب‌سازی درجی، در حالت میانگین در حلقهٔ `while` هر یک از $A[i]$ ها باید با نیمی از عناصر $A[1 : i-1]$ مقایسه شوند. بنابراین $t_i = i/2$. اگر کل زمان اجرا در حالت میانگین را محاسبه کنیم، زمان اجرا یک تابع درجه دوم از اندازهٔ ورودی به دست می‌آید. بنابراین زمان اجرا در بدترین حالت و حالت میانگین تقریبا برابرند.

---

[1] worst case
[2] upper bound
[3] average case

تحلیل مجانبی

- در تحلیل الگوریتم‌ها معمولاً در مورد مرتبهٔ رشد [1] یا نرخ رشد توابع [2] صحبت می‌کنیم و جزئیات را در محاسبات نادیده می‌گیریم. در واقع محاسبهٔ زمان اجرا را به صورت حدی در نظر می‌گیریم وقتی که اندازهٔ ورودی بسیار بزرگ باشد. وقتی n به بینهایت میل کند هر تابع درجه دوم با ضریب ثابتی از n^2 برابر است. در این حالت می‌گوییم زمان اجرا برنامه از مرتبه n^2 است.

- برای نشان دادن مرتبه بزرگی از حرف یونانی Θ (تتا) استفاده می‌کنیم. می‌گوییم زمان اجرای مرتب‌سازی درجی در بهترین حالت برابر است با $\Theta(n)$ و زمان اجرای آن در بدترین حالت برابر است با $\Theta(n^2)$، بدین معنی که برای n های بسیار بزرگ زمان اجرای الگوریتم در بدترین حالت تقریبا برابر است با n^2.

- زمان اجرای یک الگوریتم از یک الگوریتم دیگر بهتر است اگر زمان اجرای آن در بدترین حالت مرتبه رشد کمتری [3] داشته باشد.

[1] order of growth
[2] rate of growth
[3] lower order of growth

تحلیل مجانبی

- مرتبه رشد [1] زمان اجرای یک الگوریتم، معیار مناسبی برای سنجش کارایی [2] یک الگوریتم است که به ما کمک می‌کند یک الگوریتم را با الگوریتم‌های جایگزین آن مقایسه کنیم.
- گرچه محاسبه دقیق زمان اجرا در بسیاری مواقع ممکن است، اما این دقت در بسیاری مواقع ارزش افزوده‌ای ندارد چرا که به ازای ورودی‌های بزرگ مرتبه رشد زمان اجرا تعیین کننده مقدار تقریبی زمان اجرا است.
- تحلیل مجانبی [3] در آنالیز ریاضی روشی است برای توصیف رفتار حدی توابع. در تحلیل الگوریتم‌ها نیز می‌خواهیم تابع زمان اجرا را با استفاده از تحلیل مجانبی بررسی کنیم تا زمان اجرا را وقتی ورودی الگوریتم بدون محدودیت بزرگ می‌شود بسنجیم.

[1] order of growth
[2] efficiency
[3] asymptotic analysis

تحلیل مجانبی

- نماد O [1] در تحلیل مجانبی توابع، کران بالای مجانبی [2] یک تابع را مشخص می‌کند.
- تابع $f(x)$ برابر است با $O(g(x))$ اگر تابع $f(x)$ از تابع $g(x)$ سریع‌تر رشد نکند. به عبارت دیگر تابع $f(x)$ به ازای n های بسیار بزرگ از ضریب ثابتی از $g(x)$ کوچکتر است.
- برای مثال می‌گوییم تابع $2n^3 + 3n^2 + n + 4$ دارای کران بالای n^3 است و می‌نویسیم این تابع $O(n^3)$ است.
- همچنین می‌توانیم بگوییم این تابع $O(n^4)$ و $O(n^5)$ و به طور کلی $O(n^c)$ به ازای $c \geqslant 3$ است، چرا که سرعت رشد آن از این تابع بیشتر نیست.

---
[1] O-notation
[2] asymptotic upper bound

تحلیل مجانبی

- به ازای تابع دلخواه $g(n)$ ، مجموعهٔ $O(g(n))$ شامل همهٔ توابعی است که کران بالای آنها $g(n)$ است و به صورت زیر تعریف می‌شود.

$$O(g(n)) = \{f(n) : \exists c, n_0 > 0, \forall n \geqslant n_0, 0 \leqslant f(n) \leqslant cg(n)\}$$

- به عبارت دیگر تابع $f(n)$ به مجموعه توابع $O(g(n))$ تعلق دارد اگر عدد مثبت c وجود داشته باشد به طوری‌که به ازای اعداد n بزرگ‌تر از n_0 داشته باشیم $f(n) \leqslant cg(n)$.

- طبق این تعریف توابع $f(n)$ باید توابع غیر منفی باشند.

تحلیل مجانبی

- از آنجایی که نماد O در واقع یک مجموعه را تعریف می‌کند می‌توانیم بنویسیم $f(n) \in O(g(n))$ ، اما گاهی برای سادگی می‌نویسیم $f(n) = O(g(n))$ و می‌خوانیم $f(n)$ از $O(g(n))$ است، یا $g(n)$ کران بالای تابع $f(n)$ است.

- برای مثال $4n^2 + 100n + 500 = O(n^2)$. باید نشان دهیم c و n_0 وجود دارند که در شرایط تعریف شده صدق می‌کنند. به عبارت دیگر $4n^2 + 100n + 500 \leqslant cn^2$ به ازای $n_0 = 1$ برای اینکه این نامعادله درست باشد داریم $c = 604$.

تحلیل مجانبی

- نماد Ω [1] یا نماد اومگا کران پایین مجانبی [2] یک تابع را در تحلیل مجانبی مشخص می‌کند.
- تابع $f(x)$ برابر است با $\Omega(g(x))$ اگر تابع $f(x)$ از تابع $g(x)$ سریع‌تر رشد کند. به عبارت دیگر تابع $f(x)$ به ازای n های بسیار بزرگ از ضریب ثابتی از $g(x)$ بزرگتر است.
- برای مثال می‌گوییم تابع $2n^3 + 3n^2 + n + 4$ دارای کران پایین n^3 است و می‌نویسیم این تابع $\Omega(n^3)$ است.
- همچنین می‌توانیم بگوییم این تابع $\Omega(n^2)$ و $\Omega(n)$ و به طورکلی $\Omega(n^c)$ به ازای $c \leqslant 3$ است.

[1] Ω-notation
[2] asymptotic lower bound

تحلیل مجانبی

- به ازای یک تابع دلخواه $g(n)$ ، مجموعهٔ $\Omega(g(n))$ شامل همهٔ توابعی است که کران پایین آنها $g(n)$ است و به صورت زیر تعریف می‌شود.

$$\Omega(g(n)) = \{f(n) : \exists c, n_0 > 0, \forall n \geqslant n_0, 0 \leqslant cg(n) \leqslant f(n)\}$$

- برای مثال $4n^2 + 100n + 500 = \Omega(n^2)$. به عبارت دیگر $4n^2 + 100n + 500 \geqslant cn^2$ به ازای همهٔ n_0 های مثبت این نامعادله درست است اگر $c = 4$.

تحلیل مجانبی

- نماد Θ [1] یا نماد تتا، کران اکید مجانبی [2] یک تابع در تحلیل مجانبی را مشخص می‌کند.
- تابع $f(x)$ برابر است با $\Theta(g(x))$ اگر تابع $f(x)$ از تابع $g(x)$ نه سریع‌تر رشد کند و نه کندتر. به عبارت دیگر تابع $f(x)$ به ازای n های بسیار بزرگ از ضریب ثابتی از $g(x)$ بزرگتر است و از ضریب ثابتی از $g(x)$ کوچکتر است.
- اگر نشان دهیم یک تابع دارای کران بالای $f(n)$ و دارای کران پایین $f(n)$ است و یا عبارت دیگر $O(f(n))$ و $\Omega(f(n))$ است، آنگاه آن تابع دقیقا از مرتبه $f(n)$ است و یا به عبارت دیگر $\Theta(f(n))$ است.
- برای مثال می‌گوییم تابع $2n^3 + 3n^2 + n + 4$ از مرتبه n^3 است و می‌نویسیم این تابع $\Theta(n^3)$ است.

[1] Θ-notation
[2] asymtotically tight bound

تحلیل مجانبی

- به ازای تابع دلخواه $g(n)$ ، مجموعه $\Theta(g(n))$ شامل همهٔ توابعی است که کران اکید آنها $g(n)$ است، یعنی همه توابعی که $g(n)$ هم کران بالای آنها است و هم کران پایین آنها.
- به عبارت دیگر

$$\Theta(g(n)) = \{f(n) : \exists c_1, c_2, n_0 > 0, \forall n \geqslant n_0, 0 \leqslant c_1 g(n) \leqslant f(n) \leqslant c_2 g(n)\}$$

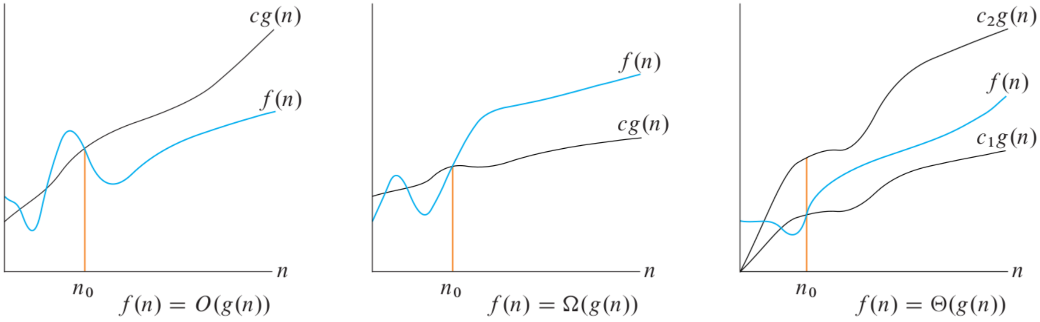
- می‌توانیم ثابت کنیم که به ازای دو تابع $f(n)$ و $g(n)$ داریم $f(n) \in \Theta(g(n))$ اگر و تنها اگر $f(n) \in O(g(n))$ و $f(n) \in \Omega(g(n))$.

تحلیل مجانبی

- نمادهای O، Ω، و Θ بر روی توابع گسسته عمل می‌کند، یعنی توابعی که دامنهٔ آنها بر روی اعداد حسابی $\mathbb{N}$ و برد آنها بر روی اعداد حقیقی $\mathbb{R}$ تعریف شده است. از این نمادها برای تحلیل مجانبی زمان اجرای الگوریتم‌ها یعنی $T(n)$ استفاده می‌کنیم.

تحلیل مجانبی

- در شکل زیر مفاهیم نمادهای مجانبی نشان داده شده‌اند.



تحلیل مجانبی الگوریتم مرتب‌سازی درجی

- حال الگوریتم مرتب‌سازی درجی را یک بار دیگر در نظر می‌گیریم.

Algorithm Insertion Sort

```
function INSERTION-SORT(A, n)
▷ A is an array of n elements
for i = 2 to n do
   key = A[i]
   j = i - 1
   while  j > 0 and A[j] > key  do
      A[j+1] = A[j]
      j = j-1
   A[j+1] = key
```

تحلیل مجانبی الگوریتم مرتب‌سازی درجی

- می‌خواهیم اثبات کنیم زمان اجرای این الگوریتم در بدترین حالت $\Theta(n^2)$ است. باید اثبات کنیم زمان اجرای الگوریتم در بدترین حالت $O(n^2)$ و $\Omega(n^2)$ است.
- این الگوریتم در یک حلقه `for` به تعداد $n - 1$ بار تکرار می‌شود. به ازای هر بار تکرار در این حلقه یک حلقه درونی `while` وجود دارد که در بدترین حالت $i - 1$ بار تکرار می‌شود و i حداکثر n است بنابراین تعداد کل تکرارها حداکثر $(n - 1)(n - 1)$ است، که این مقدار از n^2 کوچکتر است. بنابراین زمان اجرای این الگوریتم $O(n^2)$ است.

تحلیل مجانبی الگوریتم مرتب‌سازی درجی

- حال می‌خواهیم نشان دهیم زمان اجرای این الگوریتم در بدترین حالت $\Omega(n^2)$ است. برای این کار باید نشان دهیم حداقل یک ورودی وجود دارد که زمان اجرای آن حداقل از مرتبه n^2 است.

- فرض کنید یکی از ورودی‌های الگوریتم، آرایه‌ای است که طول آن مضرب 3 است و در این ورودی بزرگ‌ترین عناصر آرایه در یک سوم ابتدای آرایه قرار دارند. برای این‌که این آرایه مرتب شود همهٔ این عناصر باید به یک‌سوم انتهای آرایه انتقال پیدا کنند. برای این انتقال حداقل هر عنصر باید $n/3$ بار به سمت راست حرکت کند تا از ثلث میانی آرایه عبور کند. این انتقال باید برای حداقل یک‌سوم عناصر اتفاق بیافتد، پس زمان اجرا در این حالت حداقل $(n/3)(n/3)$ است یا به عبارت دیگر $\Omega(n^2)$ است.

- از آنجایی که مرتبه رشد مرتب‌سازی درجی در بدترین حالت حداکثر و حداقل از مرتبه n^2 است یعنی مرتبه رشد آن $O(n^2)$ و $\Omega(n^2)$ است، بنابراین می‌توانیم نتیجه بگیریم مرتبه رشد آن در بدترین حالت از مرتبه n^2 است یا به عبارت دیگر $\Theta(n^2)$ است.

داده ساختارها

- مجموعه‌ها بنیادهای اصلی علوم کامپیوتر هستند.
- در علوم ریاضی، مجموعه‌ها ثابت و بدون تغییر هستند، در صورتی که در علوم کامپیوتر مجموعه‌ها با استفاده از الگوریتم‌ها تغییر می‌کنند. مجموعه‌ها می‌توانند بزرگ یا کوچک شوند و به مرور زمان تغییر کنند. برای مثال مجموعه‌ای از داده‌های دانشجویان را در نظر بگیرید که به مرور زمان با ورود دانشجویان به دانشگاه داده‌هایی به آن اضافه می‌شود و با خروج دانشجویان از دانشگاه داده‌هایی از آن حذف می‌شود.
- بنابراین مجموعه‌ها در علوم کامپیوتر پویا [1] هستند.

[1] dynamic

داده ساختارها

- در درس ساختمان داده روش‌هایی را برای نمایش مجموعه‌های پویا و اعمال تغییر این مجموعه‌ها معرفی می‌کنیم.
- الگوریتم‌ها نیاز دارند عملیات متفاوتی برروی مجموعه‌ها انجام دهند. برای مثال، برخی از الگوریتم‌ها ممکن است نیاز داشته باشند اعضایی به مجموعه اضافه کنند یا اعضایی را از یک مجموعه حذف کنند یا اینکه بخواهند عضویت یک عنصر را در یک مجموعه بررسی کنند.
- معمولا در یک پیاده‌سازی شیءگرا، هریک از اعضای یک مجموعه یک شیء هستند که هرکدام با یک کلید مشخص شناسایی می‌شوند. هر شیء می‌تواند ویژگی‌های دیگری نیز داشته باشد که اطلاعا دیگری از یک عضو مجموعه را نگهداری می‌کنند. همچنین ویژگی‌هایی از شیء برای اعمال تغییرات برروی مجموعه مورد نیازند که در ساختار داده‌های مختلف متفاوت‌اند و به آنها یک‌به‌یک خواهیم پرداخت.

- عملیاتی که نیاز داریم برروی مجموعه‌ها انجام دهیم را به دو دسته تقسیم می‌کنیم : عملیات پرس و جو [1] که اطلاعاتی را در مورد مجموعه باز می‌گرداند و عملیات اعمال تغییرات [2] که تغییراتی برروی اعضای مجموعه اعمال می‌کند.

[1] query
[2] modifying operation

داده ساختارها

- در اینجا عملیات مهم مورد نیاز بر روی مجموعه‌ها را معرفی می‌کنیم.
- جستجو Search(S،k) : به ازای مجموعه S و مقدار کلید k ، اشاره‌گر x را به عنصری از S به طوری که x.key=k باز می‌گرداند و در صورتی که کلید هیچ‌یک از عناصر S برابر با k نباشد مقدار NIL را باز می‌گرداند.
- درج Insert(S،x) : عنصری که با اشاره‌گر x به آن اشاره شده است را به مجموعه S اضافه می‌کند.
- حذف Delete(S،x) : عنصری که با اشاره‌گر x به آن اشاره شده است را از مجموعه S حذف می‌کند. توجه کنید این تابع اشاره‌گری به یک عنصر دریافت می‌کند و نه یک مقدار کلید.

- بیشینه Maximum(S) : اشاره‌گری به عنصری از S با بزرگ‌ترین مقدار کلید باز می‌گرداند.
- کمینه Minimum(S) : اشاره‌گری به عنصری از S با کوچکترین مقدار کلید باز می‌گرداند.
- عنصر بعدی Successor(S،x) : به ازای عنصر x در مجموعه مرتب S ، اشاره‌گری به کوچکترین عنصری که بزرگترین عنصر S باشد مقدار NIL بازگردانده می‌شود.
- عنصر قبلی Predecessor(S،x) : به ازای عنصر x در مجموعه مرتب S ، اشاره‌گری به بزرگ‌ترین عنصری که مقدار آن از x کوچک‌تر است باز می‌گرداند. در صورتی که x کوچکترین عنصر S باشد مقدار NIL بازگردانده می‌شود.
- زمان اجرای هر یک از این عملیات را بر اساس اندازه مجموعه (n) می‌سنجیم.

داده ساختارها

- انواع پیاده‌سازی‌های مختلف مرتبه زمان اجرای عملیات را تغییر می‌دهد. برای مثال ممکن است در یک پیاده‌سازی درج و حذف سریع‌تر و جستجو کندتر باشد و در یک پیاده‌سازی درج و حذف کندتر و جستجو سریع‌تر باشد. بسته به نوع استفاده می‌توانیم از داده ساختار مناسب استفاده کنیم.