الگوریتم

الگوریتم های مختلفی که روی هر ساختار داده ای پیاده سازی می شوند، توسط دو معیار پیچیدگی حافظه ای و زمان مورد ارزیابی قرار می گیرند .

تعریف
نحوه بیان الگوریتم
ارزیابی کارائی الگوریتم ها
پیچیدگی حافظه
پیچیدگی زمانی

تعریف

به طور خلاصه مجموعه ای از دستوالعمل ها برای حل یک مسئله را الگوریتم می گویند. کلمه الگوریتم از نام ریاضیدان قرن نهم ابوجعفرمحمد ابن موسی الخوارزمی گرفته شده است .

تعریف دقیق تر الگوریم به صورت زیر است :

یک الگوریتم مجموعه ای متناهی از دستورات برای حل یک مسئله خاص توسط انسان یا ماشین است، که ترتیب انجام عملیات در آن مشخص شده و عملیات در زمان معینی خاتمه پیدا می کند. هر دستورالعمل در الگوریتم باید مختصر، دقیق، و صریح باشد .

یک الگوریتم پنج خاصیت زیر را باید دارا باشد:

۱. متناهی بودن. یک الگوریتم باید همیشه بعد از تعدادی گام به پایان برسد.
۲. صراحت. فعلی که در هر قدم الگوریتم انجام می گیرد باید مختصر، صریح و غیر مبهم باشد.
۳. ورودی. مقادیری هستند که ابتدا، قبل از شروع، به الگوریتم داده می شوند .
٤. خروجی. مقادیری هستند توسط الگوریم تولید می شود و رابطه مشخصی با ورودی ها دارند.
٥. کارائی. دستورات الگوریتم درحد کفایت باید ساده و دقیق باشند تا یک انسان مانند یک روبات بتواند آنها را با استفاده از قلم و کاغذ بدون احتیاج به فکر کردن در زمان معینی انجام بدهد.

الگوریتم ها قرن ها برای حل مسائلی که بشر با آنها روبرو بوده استفاده می شده اند. تقریبا کلیه برنامه های کامپیوتر، بجز برنامه های کاربردی هوش مصنوعی، دربرگیرنده الگوریتم هستند. مشهورترین الگوریتم در تاریخ، الگوریتم اقلیدسی، مربوط به زمان یونان باستان است که برای محاسبه بزرگترین مقسوم علیه مشترک دو عدد صحیح به کار می رفته است و هنوز در دنیای ریاضی کاربرد دارد .

خلق الگوریتم های زیبا، ساده و با کمترین مراحل، یکی از چالش های برنامه نویسی است.

نحوه بیان الگوریتم

الگوریتم های می توانند با نمادهای مختلفی بیان شوند:

- زبان طبیعی. استفاده از عبارات زبان طبیعی برای بیان الگوریتمی ممکن است باعث طولانی و مبهم شدن آن بشود و برای الگوریتم های پیچیده و فنی بندرت استفاده می شود.
- زبان های برنامه نویسی. در ابتدا بیان الگوریتم با زبان های برنامه نویسی موجود طرفدار داشت.
- Pseudo Code و فلوچارت. راه های ساختیافته ای برای نمایش الگوریتم، که درحین استقلال از زبان برنامه نویسی خاصی، از ابهام پرهیز می کنند.

امروزه الگوریتم ها معمولا با استفاده از Pseudo Code در یک زبان برنامه نویسی که معمولا پیاده سازی نشده بیان می شوند. در طی این درس از کدهای ساختگی که در ادامه شرح داده می شوند برای بیان الگوریتم ها استفاده می شود .

```
variable := value
```

اختصاص مقداری به یک متغیر را نشان می دهد.

```
if (condition) then
  statements1
else
  statements2
end if
```

برای بیان تصمیم گیری، اگر شرط برقرار باشد عبارت ۱ انجام می گیرد و اگر برقرار نباشد عبارت ۲

```
while (condition)
  statement
end while
```

برای نمایش حلقه تکرار، اگر شرط برقرار باشد دستورات تکرار می شوند.

```
repeat until (condition)
  statement
end loop
```

برای نمایش حلقه تکرار، تا وقتی شرط برقرار نشده باشد دستورات تکرار می شوند. درصورت برقرار بودن شرط از حلقه خارج می شود .

```
for (counter:=value1 to value2 )
  statement
end for
```

برای نمایش تکرار عبارتی به تعداد معینی، شمارنده از مقدار۱ شروع شده در هربار تکرار یک واحد به آن اضافه می شود تا به مقدار۲ برسد. برای شمارش معکوس به جای to از down to قرار می گیرد .

انتهای الگوریتم توسط کلمه end مشخص می شود.

مثال. الگوریتم زیر به متغیر x مقادیر ۱ تا ۱۰ بجز عدد ۵ را اختصاص می دهد .

```
x:=0
while (x < 10)
  if x = 4 then
    x := x + 2
  else
    x := x + 1
  end if
end while
end
```

ارزیابی کارائی الگوریتم ها

الگوریتم های مختلفی برای حل یک مسئله ممکن است طراحی شده باشند. برای انتخاب بهترین الگوریتم باید معیاری جهت مقایسه کارائی الگوریتم ها داشته باشیم. ارزیابی در دو مرحله انجام می شود؛ آنالیز کارائی و اندازه گیری کارائی است .

آنالیز کارائی یک تخمین اولیه است با دو معیار پیچیدگی فضائی (space complexity) و پیچیدگی زمانی (time complexity) سنجیده می شود که رفتار الگوریتم را در زمان اجرا با مجموعه ای از ورودی های منتخب توصیف می کنند.

بعد از پیاده سازی الگوریتم با یک زبان برنامه نویسی، آمار حقیقی درباره زمان و حافظه مصرف شده توسط الگوریتم در حین اجرا جمع آوری می شود .

پیچیدگی حافظه

پیچیدگی حاظه ای میزان فضائی از حافظه است که برنامه برای اجرای کامل به آن نیاز دارد. فضای مورد نیاز در هربرنامه مجموع قسمت های زیر است :

- بخش ثابت فضا که معمولا شامل فضای دستورالعمل، فضای متغیرهای با اندازه ثابت و فضای لازم برای ذخیره ورودی و خروجی های برنامه است.
- بخش متغیر فضا شامل فضای پشته و فضای موردنیاز برای مقادیر متغیرهائی که اندازه آنها بستگی به مسئله و مشخصات ورودی دارد.

در تحلیل فضای لازم روی تخمین بخش متغیر تاکید نداریم زیرا برای هرمسئله ابتدا باید مشخصات موردی را تعیین کنیم که کار دشواری است .

پیچیدگی زمانی

زمان اجرا مقدار زمانی از کامپیوتر است که برنامه برای اجرای کامل مصرف می کند. برای محاسبه پیچیدگی زمان الگوریتم ابتدا تعداد قدم های الگوریتم به صورت تابعی از اندازه مسئله مشخص می شود، برای انجام این کار تعداد تکرارعملیات اصلی الگوریتم محاسبه می شود و به صورت تابع f(n) (که n تعداد ورودی هاست) بیان می شود. سپس تابع g(n)، که مرتبه بزرگی تابع f(n) را وقتی اندازه ورودی به اندازه کافی بزرگ است نشان می دهد، بدست می آید. در نهایت پیچیدگی الگوریتم برای نشان دادن رفتار الگوریتم با ورودی های مختلف با استفاده از نمادها **O** ، **Θ** و **Ω** بیان می شود.

تعریف Big-**O** (حدبالا)
تابع *f(n)* را نظر بگیرید که برای کلیه $n \geq 0$ است، می گوئیم $f(n) = O(g(n))$ اگر ثابت های مثبت n_0 و c_1 وجود داشته باشند به طوریکه از یک n_0 به بعد همیشه $f(n) \leq cg(n)$ برقرار باشد.
این نماد حدبالائی برای تابع *f(n)* می دهد و وقتی بکار می رود که رفتار الگوریتم بدترین حالت و بیشترین زمان اجرا را برای مقادیر معین ورودی دارد

نماد Big-**Ω** (حدپائین)
تابع *f(n)* را نظر بگیرید که برای کلیه $n \geq 0$ است ، می گوئیم $f(n) = \Omega(g(n))$ اگر ثابت های مثبت n_0 و c_1 وجود داشته باشند به طوریکه از یک n_0 به بعد همیشه $f(n) \geq cg(n)$ برقرار باشد.
این نماد حد پائینی برای تابع *f(n)* می دهد و وقتی بکار می رود که رفتار الگوریتم بهترین حالت و کمترین زمان اجرا را برای مقادیر معین ورودی دارد

نماد Big-**Θ** (حدمتوسط)
تابع *f(n)* را نظر بگیرید که برای کلیه $n \geq 0$ است، می گوئیم $f(n) = \Theta(g(n))$ اگر ثابت های مثبت n_0، c_1 و c_2 وجود داشته باشند به طوریکه از یک n_0 به بعد همیشه $c_1 g(n) \leq f(n) \leq c_2 g(n)$ برقرار باشد.
این نماد حدمتوسطی برای تابع f(n) می دهد و زمان اجرای الگوریتم را به صورت میانگینی از تعداد عملیات انجام شده با کلیه نمونه ورودی های مسئله نشان می دهد.

قضیه. اگر $f(n)=a_m n^m+a_{m-1}n^{m-1}+\ldots+a_1 n+a_0$ در اینصورت $f(n)=O(n^m)$ است.

نکته. اگر زمان الگوریتم وابسته به ورودی نباشد با نماد $O(1)$ نشان داده می شود.
نکته. باید به اندازه کافی الگوریتم را درک کرده باشیم تا بهترین و بدترین رفتار را تولید و محاسبه کنیم. چون برآورد رفتار آماری ورودی ها امری دشوار است، در اکثر موارد به بدترین حالت قناعت می کنیم.

نکته. اگر الگوریتم شامل بخش های مختلفی باشد که هر قسمت پیچیدگی متفاوتی دارد، مرتبه بزرگی هر قسمت را پیدا کرده و بزرگترین مرتبه را بعنوان پیچیدگی کل الگوریتم درنظرمی گیریم.این نماد حدمتوسطی برای تابع $f(n)$ می دهد و زمان اجرای الگوریتم را به صورت میانگینی از تعداد عملیات انجام شده با کلیه نمونه ورودی های مسئله نشان می دهد .

مثال. الگوریتم مرتب سازی حبابی را درنظر بگیرید .

```
for (i:=1 to n-1)
  for (j:=1 to n-1)
    if a_j>a_{j+1} then exchange(a_j,a_{j+1})
```

با درنظر گرفتن عمل مقایسه بعنوان عملگر اصلی، دستور If در الگوریتم فوق $(n-1)^2$ بار تکرار می شود. بنابراین $f(n)=(n-1)^2=n^2-2n+1$ و طبق قضیه $g(n)=O(n^2)$ است. بنابراین پیچیدگی الگوریتم فوق برابر با $O(n^2)$ می باشد.

نکته. اگر زمان الگوریتم وابسته به ورودی نباشد با نماد $O(1)$ نشان داده می شود.

نکته. باید به اندازه کافی الگوریتم را درک کرده باشیم تا بهترین و بدترین رفتار را تولید و محاسبه کنیم. چون برآورد رفتار آماری ورودی ها امری دشوار است، در اکثر موارد به بدترین حالت قناعت می کنیم.

نکته. اگر الگوریتم شامل بخش های مختلفی باشد که هر قسمت پیچیدگی متفاوتی دارد، مرتبه بزرگی هر قسمت را پیدا کرده و بزرگترین مرتبه را بعنوان پیچیدگی کل الگوریتم درنظرمی گیریم.

غالبا پیچیدگی $g(n)$ یکی از توابع زیر است: n (پیچیدگی خطی)، $\log n$ (لگاریتمی)، n^a (چندجمله ای) و a^n که $a\geq 2$ (نمائی).

در زیر مربته اجرائی چند تابع به ترتیب صعودی نوشته شده است.

$$O(1)< O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$$