به سوی علوم ریاضی

دوره ۶ شماره ۱، بهار ۱۴۰۵، صص. ۶۱–۸۳

DOI.10.22067/tmsj.2026.97896.1078

مقاله علمی–ترویجی

https://tmsj.um.ac.ir

الگوریتم‌ها: از دوران باستان تا عصر رایانه

جعفر الماسی‌زاده

گروه ریاضی کاربردی و علوم کامپیوتر، دانشکده ریاضی و آمار، دانشگاه اصفهان

j.almasizadeh@sci.ui.ac.ir

چکیده. در این مقاله به بررسی هفت الگوریتم قدیمی و برجسته می‌پردازیم که زمان کشف آنها به قرن‌ها قبل از اختراع رایانه برمی‌گردد. هدف اصلی از گزینش و معرفی این الگوریتم‌ها این است که خواننده متوجه این واقعیت شود که الگوریتم‌ها، گرچه به طور ضمنی، از دوران باستان مورد توجه ریاضیدانان بوده‌اند و این گونه نیست که بشر با اختراع رایانه پی به مفهوم الگوریتم برده باشد. با این نوشته خواننده می‌تواند از چشم‌اندازی تاریخی به الگوریتم‌ها بنگرد و قسمتی از مسیری را که الگوریتم‌دانان از ۴۰۰۰ سال قبل تاکنون پیموده‌اند ببیند. با این فرض که بیشتر خوانندگان این مقاله، افراد مشتاقی هستند که به تازگی پا در مسیر یادگیری الگوریتم‌ها گذاشته‌اند، همراه با ذکر نکات تاریخی، به جنبه‌های آموزشی و کاربردی هفت الگوریتم نیز می‌پردازیم. از جنبه آموزشی، نحوه طراحی هفت الگوریتم مبتنی بر رابطه‌های ریاضی بررسی می‌شود و از جنبه کاربردی، نحوه پیاده‌سازی هفت الگوریتم با زبان برنامه‌نویسی پایتون بررسی می‌شود. در پایان، چند نکته و پیشنهاد برای کسانی که می‌خواهند به بررسی دقیق‌تر الگوریتم‌ها از چشم‌انداز تاریخی بپردازند، بیان می‌شود.

۱. مقدمه

با اختراع رایانه در دهه ۱۹۴۰ میلادی انقلابی بزرگ در علم و فن‌آوری به وقوع پیوست. از آن زمان تاکنون، بشر به مدد رایانه‌ها به کشفیات شگرفی نائل شده است. ناگفته پیداست که رایانه ابزاری است در دست برنامه‌نویس و نوشتن برنامه‌های رایانه‌ای نیز مستلزم آگاهی برنامه‌نویس از الگوریتم‌های لازم است. آنقدر رایانه در زمینه‌های علمی مختلف (چه مهندسی، چه زیست‌شناسی، چه

2020 Mathematics Subject Classification. 01A99, 68W99, 97D50, 97P99

واژگان کلیدی. ریاضی، الگوریتم، مسأله، رایانه، تاریخ .

تاریخ: دریافت ۱۴۰۴/۱۱/۲۸ بازنگری ۱۴۰۵/۰۲/۱۰ پذیرش ۱۴۰۵/۰۳/۳۱ انتشار برخط ۱۴۰۵/۰۳/۳۱

نحوه ارجاع به این مقاله: ج. الماسی‌زاده، الگوریتم‌ها: از دوران باستان تاعصر رایانه، به سوی علوم ریاضی، ۶ (۱۴۰۵)، شماره ۱، ۶۱–۸۳.

اقتصاد یا هر زمینه دیگری) کاربرد پیدا کرده است که هیچ شخصی که به علم‌اندوزی در زمینه‌ای علمی مشغول باشد بی‌نیاز از دانستن الگوریتم‌ها نیست و در واقع، اصطلاح «الگوریتم» امروزه دست‌کم برای افرادی که به نحوی به موضوعات علمی و فن‌آورانه مشغول هستند، واژه‌ای به ظاهر آشنا است.

این مقاله قرار است افراد مشتاق را از جنبه تاریخی بیشتر با الگوریتم‌ها آشنا کند؛ با این توجیه که کسی که می‌خواهد عالمانه در زمینه‌ای علمی به تحقیق بپردازد لازم است با تاریخ شکل‌گیری و توسعه آن زمینه علمی آشنا باشد و زمینه علمی الگوریتم‌ها نیز از این قاعده مستثنی نیست. البته از آنجا که بررسی الگوریتم‌ها از چشم‌انداز تاریخی موضوعی آنقدر گسترده است که باید چند جلد کتاب را به آن اختصاص داد، ما در این مقاله تنها به بررسی چند نقطه عطف در تاریخ الگوریتم‌ها قبل از اختراع رایانه می‌پردازیم. نگاه از این چشم‌انداز تاریخی خواننده‌ای را که می‌خواهد پا به دنیای الگوریتم‌ها بگذارد، به میزانی متوجه خواهد کرد که بشر در طول تاریخ در چه بستری الگوریتم‌ها را کشف کرده است و در چه بستری الگوریتم‌ها را به کار برده است. از آنجا که در کنار بحث‌های تاریخی، نمی‌توان جنبه‌های آموزشی و کاربردی الگوریتم‌ها را نادیده گرفت، پس از بررسی تاریخی هر الگوریتم، به نحوه طراحی و پیاده‌سازی آن الگوریتم هم می‌پردازیم.

قبل از ورود به بحث درباره الگوریتم‌ها، ذکر نکاتی درباره واژه «الگوریتم» مفید به نظر می‌رسد. ریشه واژه algorithm نام ریاضیدان بزرگ ایرانی ابوجعفر محمد بن موسی خوارزمی (۷۸۰–۸۵۰) است. یکی از افرادی که خوارزمی را به دنیای غرب معرفی کرد، لئوناردو فیبوناچی[1] بود. لئوناردو فیبوناچی (۱۱۷۰–۱۲۵۰) که از او به عنوان بزرگترین ریاضیدان اروپایی در قرون وسطی یاد می‌شود، دستگاه ارقام دهدهی (ارقام صفر تا نه) را به اروپاییان معرفی کرد. این دستگاه اعداد را برای اولین بار ریاضیدانان هندی بین قرن اول و قرن چهارم میلادی و بنا بر نقلی در قرن ششم میلادی ابداع و استفاده کردند. بعدها ریاضیدانان مسلمان مانند خوارزمی و ابویوسف یعقوب بن اسحاق کِندی (۸۰۱–۸۷۳) ، با الگوبرداری از روی نمادهای ارقام هندی و تغییر شکل آنها، نمادهای ارقام دهدهی را که ما امروزه استفاده می‌کنیم یعنی ارقام ۰،۱،۲،۳،۴،۵،۶،۷،۸،۹ را طراحی کردند و برای حساب استفاده کردند. فیبوناچی آن دستگاه اعداد را که به نام «اعداد عربی» در نزد مسلمانان رایج بود، در کتاب خود به نام «کتاب حساب»[2] به اروپاییان معرفی کرد و بر سودمندی و سادگی آن برای انجام محاسبات نسبت به دستگاه اعداد رومی (که در آن زمان در اروپا برای حساب استفاده می‌شد و فوق‌العاده پیچیده و عملاً برای انجام عملیات حسابی روی اعداد کوچک مفید بود) تأکید کرد. پیشنهاد فیبوناچی مورد استقبال قرار گرفت و این دستگاه اعداد از قرن دوازدهم میلادی به بعد مورد استفاده ریاضیدانان اروپایی قرار گرفت و آنان از اواخر قرن پانزدهم میلادی شکل آنها را به نمادهای معادل 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 تغییر دادند.

فیبوناچی از بستر معرفی دستگاه اعداد دهدهی قسمتی از دستاوردهای علمی خوارزمی را نیز به دنیای غرب معرفی کرد. خوارزمی در کتاب‌های خود روش‌هایی را برای انجام عملیات حسابی جمع و ضرب و تقسیم و حتی محاسبه جذر اعداد و محاسبه ارقام عدد پی آورده بود؛ آن روش‌ها تمام ویژگی‌های الگوریتم را به معنایی که امروزه از کلمه «الگوریتم» در ذهن داریم داشتند و در واقع، بعدها به افتخار همان ابداعات و تلاش‌های علمی خوارزمی بود که کلمه الگوریتم از نام او گرفته شد. البته کلمه algorithm در ابتدا algorism بوده است و سپس در گذر زمان هم خود کلمه تغییر کرده است و هم معنای آن بسط پیدا کرده است. تا چند قرن، منظور از کلمه algorithm، منحصراً روش‌هایی بود که برای عملیات مقدماتی حساب (جمع و تفریق و ضرب و تقسیم) با اعداد عربی استفاده می‌شدند.

---

[1] Leonardo Fibonacci

[2] The book of calculation

امروزه گرچه واژه «الگوریتم» از کلیدی‌ترین واژه‌هایی است که در متون علمی دیده می‌شود، ولی به ندرت کتاب معتبری را می‌توان یافت که تعریف دقیقی از این واژه کرده باشد. شاید یکی از بهترین تعریف‌ها از واژه الگوریتم، تعریفی باشد که دونالد کنوث[۳]، رایانه‌دان[۴] برجسته، در کتاب خود ذکر کرده است:

> مفهوم الگوریتم پایه و اساس هر برنامه رایانه‌ای است؛ بنابراین ما قبل از هر چیز باید این مفهوم را به دقت تحلیل کنیم. از نظر لغوی معنای امروزی کلمه algorithm، کاملاً با معنای کلماتی چون procedure، technique، method، process، recipe، routine و rigmarole مشابه است جز اینکه کلمه «الگوریتم» متضمن چیزی است که آن را اندکی متفاوت می‌کند. هر الگوریتمی، علاوه بر اینکه صرفاً مجموعه‌ای متناهی از قواعد باشد که بتوان از روی آنها دنباله عملیات لازم برای حل یک نوع مسأله خاص را تعیین کرد، پنج ویژگی مهم نیز دارد:
>
> (۱) **تناهی:** اجرای هر الگوریتمی همیشه باید بعد از یک تعداد متناهی گام خاتمه پیدا کند.
>
> (۲) **قطعیت:** هر گام هر الگوریتمی باید به دقت تعریف شود؛ یعنی اقداماتی که در هر مورد قرار است انجام شوند، باید به طور دقیق، صریح و غیرمبهم توصیف شوند.
>
> (۳) **ورودی:** هر الگوریتمی صفر یا چند ورودی دارد؛ ورودی‌ها کمیت‌هایی هستند که یا در آغاز قبل از آنکه اجرای الگوریتم شروع شود، به آن داده می‌شوند یا به طور پویا هنگامی که الگوریتم در حال اجراست.
>
> (۴) **خروجی:** هر الگوریتمی یک یا چند خروجی دارد؛ خروجی‌ها کمیت‌هایی هستند که رابطه مشخصی با ورودی‌ها دارند.
>
> (۵) **کارایی:** در حالت کلی انتظار می‌رود که یک الگوریتم کارا هم باشد؛ به این معنا که هر عمل آن باید در اصل به اندازه‌ای مقدماتی باشد که هر کسی بتواند آن را با استفاده از قلم و کاغذ به طور دقیق و در مدت زمانی متناهی انجام دهد [۱۰].

از میان این ویژگی‌ها آن ویژگی که وجه تمایز واژه «الگوریتم» با بقیه واژه‌هاست ویژگی «قطعیت» است. در واقع، «الگوریتم» به راه‌حل‌هایی برای مسائل گفته می‌شود که مبهم نباشند و بتوان آنها را دقیقاً به یک شکل تفسیر کرد. به دلیل همین مشخصه الگوریتم است که می‌توان آن را به برنامه‌ای رایانه‌ای تبدیل کرد. پس اگر نتوان راه‌حلی برای یک مسأله را به برنامه‌ای رایانه‌ای تبدیل کرد یا آن راه‌حل، الگوریتم نیست یا آنکه گام‌هایی از آن واضح بیان نشده است. این نکته را هم باید در ذهن داشت که گرچه بیشتر الگوریتم‌ها با هدف پیاده کردن آنها و حل رایانه‌ای مسائل نوشته می‌شوند، خود مفهوم الگوریتم وابسته به وجود رایانه نیست.

واژه «الگوریتم» به معنای امروزی آن تا قبل از اختراع رایانه واژه‌ای رایج در نزد ریاضیدانان نبوده است و ریاضیدانان به الگوریتم‌ها «روش» می‌گفتند. بعد از اختراع رایانه در دهه ۱۹۴۰ نیز این کلمه تا سال‌ها مطرح نبوده است و منظور از کلمه الگوریتم تا سال ۱۹۵۰ عمدتاً الگوریتم اقلیدس بوده است. در دهه ۱۹۵۰ نیز گرچه افراد بسیاری به استفاده از الگوریتم یا به طراحی الگوریتم مشغول بودند، عملاً زمینه طراحی الگوریتم‌ها به عنوان یک موضوع علمی ویژه مطرح نبود و به الگوریتم‌ها برچسب‌هایی مانند «روش» یا «روال» یا «فرایند» یا «فن» و کلماتی از این دست می‌زدند. حتی امروزه هم در بسیاری از متون معتبر الگوریتم‌ها از واژه‌هایی مانند «روش» یا «روال» به جای واژه الگوریتم استفاده می‌شود. اما درست این است که اگر ما خاص بودن واژه «الگوریتم» را پذیرفته‌ایم نباید از واژه‌هایی به جای واژه «الگوریتم» استفاده کنیم که دقیقاً همان معنا را نمی‌رسانند.

در دهه ۱۹۶۰ میلادی رشته علم رایانه در دانشگاه‌های آمریکا ایجاد شد و در همان دهه زمینه علمی طراحی و تحلیل الگوریتم‌ها نیز که محوری‌ترین حوزه در رشته علم رایانه است، بسط یافت و بسیار بیشتر از گذشته مورد توجه قرار گرفت. (مقاله [۸] را ببینید.)

---

[3] Donald Ervin Knuth

[4] Computer scientist

و اکنون آنقدر این زمینه علمی بسط و اشاعه پیدا کرده است که هیچ زمینه علمی دیگری را نمی‌توان یافت که کاربرد الگوریتم‌ها در آن مطرح نباشد.

۲. هفت الگوریتم قدیمی

در این بخش به بررسی هفت الگوریتم قدیمی و زیبا می‌پردازیم. این هفت الگوریتم عبارتند از الگوریتم ضرب روسی، الگوریتم اقلیدس، الگوریتم غربال اراتستنس، الگوریتم گاوس، الگوریتم هرون، الگوریتم نیوتن و الگوریتم هرنر. اهمیت این الگوریتم‌ها صرفاً از جنبه تاریخی نیست، بلکه نحوه طراحی و پیاده‌سازی هر یک از الگوریتم‌ها را می‌توان مثالی مفید دانست برای کسانی که می‌خواهند مهارت‌های حل مسأله خود را تقویت کنند یا پا به عرصه برنامه‌نویسی بگذارند. ما هر یک از الگوریتم‌ها را از سه جنبه بررسی می‌کنیم: جنبه تاریخی و جنبه طراحی و جنبه کاربردی. از جنبه تاریخی به این می‌پردازیم که طبق مستندات، الگوریتم را اولین بار چه کسانی استفاده می‌کرده‌اند؛ از جنبه طراحی، به مبنا و شیوه طراحی الگوریتم می‌پردازیم؛ و از جنبه کاربردی به اهمیت مسأله‌ای که با الگوریتم حل می‌شود و به نحوه پیاده‌سازی درست الگوریتم با زبان برنامه‌نویسی پایتون می‌پردازیم (فرض بر این است که خواننده با زبان پایتون آشنا است).

لازم به ذکر است که هر یک از این الگوریتم‌ها در متون قدیمی به شیوه‌ای متفاوت و احتمالاً ناآشنا برای عموم افراد بیان شده‌اند. در اینجا برای معرفی و توضیح الگوریتم‌ها از کلمات و نمادهایی استفاده می‌شود که برای خوانندگان واضح و روشن باشد. خواننده‌ای که کنجکاو است بداند گذشتگان از چه نمادهایی برای تشریح الگوریتم‌ها استفاده می‌کردند؛ به کتاب [۳] مراجعه کند.

۱.۲. **الگوریتم ضرب روسی.**

علم حساب را باید قدیمی‌ترین شاخه از ریاضیات و پایه‌ای برای بقیه شاخه‌ها دانست. در حقیقت نمی‌توان تصور کرد که بشر قبل از آنکه بتواند با اعداد کار کند به مسائل ریاضی دیگری بپردازد. مستندات متعددی در دست است که در تمدن‌های باستانی مانند سومریان و بابِلیان و مصر باستان و چین باستان، یافتن راه‌هایی برای عملیات حسابی (جمع و تفریق و ضرب و تقسیم) مورد توجه ریاضیدانان بوده است. یکی از الگوریتم‌های حسابی باستانی برای عملیات ضرب اعداد، الگوریتمی است که امروزه به نام «الگوریتم ضرب کشاورز روسی» [۵] یا «الگوریتم ضرب روسی» شناخته می‌شود.

این الگوریتم را طبق مستندات شاید بتوان قدیمی‌ترین الگوریتمی دانست که انسان‌ها آن را مکتوب کرده‌اند و به دست ما رسیده است. گونه‌ای از این الگوریتم روی الواح پاپیروسی در مصر باستان حکاکی شده است که قدمت آنها به ۱۶۵۰ سال قبل از میلاد مسیح برمی‌گردد و کاتب آن الواح مدعی بوده است که ۳۵۰ سال قبل از آن نیز این روش را مصری‌ها می‌دانسته‌اند. به بیان دیگر، قدمت این الگوریتم که افرادی آن را «الگوریتم ضرب مصری» می‌نامند، به تقریباً ۴۰۰۰ سال قبل برمی‌گردد.

این الگوریتم قرن‌ها معمول‌ترین روش ضرب در اروپا بوده است و حتی تا اواخر قرن بیستم در مدارس کشورهای اروپای شرقی به دانش‌آموزان آموزش داده می‌شد. علت شهرت الگوریتم به نام کشاورز روسی این بوده است که طبق مشاهدات جهانگردان غربی در قرن نوزدهم، کشاورزان روسی از آن زیاد استفاده می‌کردند.

۱.۱.۲. *طراحی الگوریتم.*

مبنای الگوریتم ضرب روسی (مصری) را می‌توان با دو رابطه بازگشتی مشخص کرد. فرض کنید m و n دو عدد صحیح مثبت باشند. اگر n عددی زوج باشد می‌توان حاصل‌ضرب n در m را طبق این رابطه بازگشتی محاسبه کرد:

$$n \times m = \frac{n}{2} \times 2m,$$

و اگر n عددی فرد باشد می‌توان حاصل‌ضرب n در m را طبق این رابطه بازگشتی محاسبه کرد:

$$n \times m = \frac{n-1}{2} \times 2m + m.$$

[5] Russian peasant

با الگوریتم طبق این دو رابطه بازگشتی، عدد اول را نصف یا تقریباً نصف می‌کنیم و عدد دوم را دو برابر می‌کنیم و آنقدر این کار را تکرار می‌کنیم تا آنکه سرانجام عدد اول ۱ شود و شرط خاتمه که $1 \times m = m$ است برقرار شود. توجه کنید که ما با این الگوریتم برای محاصبه حاصل‌ضرب دو عدد صحیح چند رقمی، هیچ دو رقمی از آن دو عدد را در هم ضرب نمی‌کنیم! ما باید یا زوج یا فرد بودن عددی را مشخص کنیم؛ یا دو عدد را با هم جمع کنیم؛ یا یک عدد را دو برابر کنیم؛ یا یک عدد را نصف و اگر فرد باشد، به پایین گرد کنیم.

محاسبه دستی حاصل‌ضرب اعداد صحیح با این الگوریتم، از محاسبه دستی با الگوریتم آشنایی که امروزه در مدارس به دانش‌آموزان آموزش داده می‌شود و طبق آن هر رقم عدد دوم را در هر رقم اول ضرب می‌کنیم و سپس حاصل‌ضرب‌های میانی را با هم جمع می‌کنیم، ساده‌تر است؛ گرچه وقتی اعداد به شکل دودویی نمایش داده شوند، دو الگوریتم معادل خواهند بود. مثالی از نحوه اجرای دستی الگوریتم ضرب کشاورز روسی در کتاب [۱۴] ذکر شده است.

۲.۱.۲. *پیاده‌سازی الگوریتم.* مسأله محاسبه حاصل‌ضرب اعداد صحیح مسأله‌ای است که از زمان باستان تاکنون مطرح بوده است. اگر از کاربردهای فراوان عملیات ضرب اعداد صحیح در زندگی روزمره بگذریم، باز این مسأله مسأله‌ای کاربردی به حساب می‌آید. مثلاً در حوزه رمزنگاری، ساخت چند سامانه رمز سریع، مستلزم استفاده از الگوریتمی سریع برای محاسبه حاصل‌ضرب اعداد صحیح چند ده رقمی یا چند صد رقمی است.

از چشم‌انداز کاربردی الگوریتم ضرب روسی، سریع‌ترین الگوریتم ممکن برای محاسبه حاصل‌ضرب اعداد صحیح نیست، اما باید به آن به چشم الگوریتمی کاربردی نگریست. برنامه پیاده‌ساز الگوریتم را طبق دو رابطه‌ای که ذکر شد می‌توان به راحتی نوشت. در این برنامه از عملیات ضرب یک عدد در دو، $m = m \times 2$، استفاده شده است، اما می‌توان همین عملیات ضرب را به عملیات جمع $m = m + m$ تبدیل کرد.

Implements Russian peasant algorithm

```
def RussianPeasant(n, m):
    p = 0
    while n != 1:
        if n % 2 == 1:
            p = p + m
        n = n // 2
        m = m * 2
    return p + m
```

۲.۲. **الگوریتم اقلیدس.** مسأله محاسبه بزرگترین مقسوم‌علیه مشترک (ب.م.م) اعداد صحیح، یکی از مسائل پایه‌ای در نظریه اعداد است. این مسأله از مسائل مورد توجه ریاضیدانان یونان باستان بود. روشی که آنان برای حل این مسأله استفاده می‌کردند، به نام «الگوریتم اقلیدس» معروف شده است.

اقلیدس[۶] (۲۶۵–۳۲۵ قبل از میلاد) از برجسته‌ترین ریاضیدانان یونان باستان بود. اطلاعات مهمی از زندگی اقلیدس در دست نیست جز آنکه او در مدرسه اسکندریه مصر ریاضیات و به ویژه هندسه تدریس می‌کرد. کتاب «اصول»[۷] او که در آن اصول هندسه را

[6] Euclid

[7] Elements

پایه‌ریزی کرده است، یکی از اثرگذارترین کتاب‌هایی است که در طول تاریخ بشر نوشته شده است. اقلیدس در یکی از مجلدات کتاب اصول خود، برای محاسبه بزرگترین مقسوم‌علیه مشترک دو عدد صحیح راهی را توضیح داده است که امروزه آن را به عنوان الگوریتم اقلیدس می‌شناسیم. این الگوریتم در نوشته‌هایی که قدیمی‌تر از کتاب اصول اقلیدس هستند به چشم نمی‌خورد، ولی در نوشته‌های بعضی از دیگر ریاضیدانان یونان باستان، از جمله آریستارخوس ساموسی[۸] (۲۳۰–۳۱۰ قبل از میلاد) و ارشمیدس[۹] (۲۱۲–۲۸۷ قبل از میلاد)، نکته اصلی در طراحی الگوریتم (که تقلیل اعداد در هر مرحله به اعداد کوچکتر است) به چشم می‌خورد.

۱.۲.۲. *طراحی الگوریتم.* فرض کنید m و n دو عدد صحیح نامنفی باشند و هر دو با هم صفر نباشند. بزرگترین مقسوم‌علیه مشترک m و n را با $\gcd(m,n)$ نشان می‌دهیم. با الگوریتم اقلیدس برای محاسبه ب.م.م دو عدد m و n، با این فرض که $m \geq n$ باشد از این رابطه بازگشتی استفاده می‌کنیم:

$$\gcd(m,n) = \gcd(n, m \bmod n).$$

با الگوریتم اقلیدس مبتنی بر این رابطه بازگشتی ساده، مسأله محاسبه ب.م.م دو عدد را به مسأله محاسبه ب.م.م دو عدد کوچکتر تقلیل می‌دهیم و آنقدر این فرایند تقلیل اندازه مسأله را ادامه می‌دهیم تا آنکه سرانجام یکی از دو عدد صفر شود. از آنجا که $\gcd(m,0) = m$ است، می‌توان نتیجه گرفت ب.م.م دو عدد اصلی، عدد غیرصفر در آخرین گام الگوریتم است. اثبات درستی الگوریتم اقلیدس در کتاب‌های متعددی ذکر شده است. به عنوان مثال، کتاب [۱۸] را ببینید.

۲.۲.۲. *پیاده‌سازی الگوریتم.* مسأله محاسبه ب.م.م دو یا چند عدد صحیح در طراحی چند سامانه رمز (مثلاً سامانه رمز آر.اِس.اِی[۱۰]) پیش می‌آید. از آنجا که سرعت عملیات رمزنگاری یکی از شاخصه‌های مطلوب سامانه‌های رمز است، محاسبه سریع ب.م.م اعداد بزرگ در عمل مهم است.

گرچه الگوریتم‌های متعددی برای مسأله محاسبه ب.م.م اعداد شناسایی شده‌اند، اما ساده‌ترین و سریع‌ترین الگوریتم در میان آنها الگوریتم اقلیدس است. برنامه پیاده‌ساز الگوریتم اقلیدس را می‌توان به این صورت نوشت:

Computes gcd(m, n) by Euclid's algorithm

```
def Euclid(m, n):
    while n != 0:
        r = m % n
        m = n
        n = r
    return m
```

به دو نکته درباره این برنامه پایتونی باید توجه کرد. اول آنکه در اینجا نامنفی بودن دو عدد صحیح m و n فرض شده است. اما طبق تساوی $\gcd(m,n) = \gcd(|m|,|n|)$، می‌توان ابتدا در صورتی که یک یا هر دو عدد منفی باشد، آنها را مثبت کرد و سپس با الگوریتم اقلیدس ب.م.م دو عدد نامنفی را محاسبه کرد. دوم آنکه در اینجا فرض شده است که $m \geq n$ است. اگر $m < n$ باشد، بعد از تکرار اول حلقه جای m و n عوض می‌شود و روند حل مسأله طبق رابطه بازگشتی، ادامه می‌یابد.

[8] Aristarchus of Samos

[9] Archimedes

[10] RSA

۳.۲. **الگوریتم غربال اراتستنس.** اعداد اول از زمان باستان، هم از منظر زیباشناختی و هم از منظر فلسفی مورد توجه ریاضیدانان بوده‌اند. یکی از مسائل جذاب در آن دوران، مسأله شناسایی اعداد اول بود و امروزه نیز این مسأله البته بیشتر از چشم‌انداز کاربردی مسأله‌ای مهم به حساب می‌آید. یکی از روش‌هایی که ریاضیدانان باستان برای شناسایی اعداد اول استفاده می‌کردند، روشی است که ما امروزه آن را به نام «الگوریتم غربال اراتستنس»[11] می‌شناسیم.

اراتستنس (۱۹۴–۲۷۶ قبل از میلاد) در یونان باستان به دنیا آمد. او مدتی را در فرهنگستان افلاطون در آتن گذراند و بعدها به دعوت پادشاه وقت به اسکندریه در مصر باستان رفت و کتابدار اعظم کتابخانه بزرگ اسکندریه شد. اراتستنس نه تنها در ریاضیات بلکه در علوم دیگری از جمله تاریخ و جغرافیا و نجوم و فلسفه هم متبحر بود. او علاوه بر دستاوردهایش در ریاضیات، برای کارهایش در زمینه تاریخ‌نگاری دوران باستان و اندازه‌گیری شعاع کره زمین نیز معروف است.

اراتستنس یک نسل بعد از اقلیدس می‌زیسته است. هم الگوریتم غربال اراتستنس و هم الگوریتم اقلیدس، گویای این است که بررسی ویژگی‌های اعداد طبیعی به طور عام و اعداد اول به طور خاص، از زمان باستان برای ریاضیدانان جذاب بوده است.

الگوریتم غربال اراتستنس را ریاضیدانی به نام نیکوماخوس[12] (۶۰–۱۲۰) که چند قرن بعد از اراتستنس می‌زیسته است، در یکی از کتاب‌هایش با عنوان «مقدمه‌ای بر حساب» که در آن به بررسی روشمند ویژگی‌های اعداد پرداخته است، توضیح داده است.

۱.۳.۲. *طراحی الگوریتم.* الگوریتم غربال اراتستنس تمام اعداد اولی را می‌یابد که از عدد طبیعی مثبت مشخصی، بزرگتر نباشند. الگوریتم برای یافتن مجموعه تمام اعداد اولی که کوچکتر یا مساوی با عدد طبیعی n باشند، با مجموعه اعداد طبیعی ۲ تا n شروع می‌کند و مرحله به مرحله تمام اعدادی را که در محدوده ۲ تا n قرار دارند و اول نیستند می‌یابد و آنها را «غربال می‌کند»، یعنی به عنوان عددی نامطلوب از مجموعه کنار می‌گذارد. سرانجام پس از غربال کردن همه اعداد مرکب، همه اعداد اول و فقط هم آنها در مجموعه اعداد باقی خواهند ماند.

به بیان دقیق‌تر، طبق الگوریتم در مرحله اول، تمام مضارب عدد اول ۲(جز خود ۲) را از مجموعه غربال می‌کنیم؛ سپس در مرحله دوم، تمام مضارب عدد اول ۳ (جز خود ۳) را از مجموعه غربال می‌کنیم و به همین شیوه مضارب اعداد اول ۵ و ۷ و بقیه اعداد اولی را که در محدوده ۲ تا n قرار دارند می‌یابیم، و حذف می‌کنیم. این فرایند تا زمانی ادامه می‌یابد که دیگر نتوان مضربی را از مجموعه حذف کرد. آنچه نهایتاً در مجموعه باقی می‌ماند همه اعداد اولی است که در محدوده ۲ تا n قرار دارند.

توجه کنید که در هر مرحله، برای حذف مضارب عدد اول p کافی است از $p \times p$ شروع کنیم، چون هر یک از مضارب کوچک‌تر p در یکی از مراحل قبلی الگوریتم حذف شده‌اند. بنابراین، بزرگترین عدد اولی که مضارب آن باید حذف شود از $\lfloor \sqrt{n} \rfloor$ بیشتر نخواهد بود. مثالی از نحوه اجرای الگوریتم غربال اراتستنس در کتاب [۱۸] ذکر شده است.

۲.۳.۲. *پیاده‌سازی الگوریتم.* می‌توان برنامه پیاده‌ساز الگوریتم اراتستنس را به این صورت نوشت:

Implements the sieve of Eratosthenes

```
def Sieve(n):
    from math import floor, sqrt
    A = [None] * (n+1)
    for p in range(2, n+1):
        A[p] = p
```

[11] Sieve of Eratosthenes

[12] Nicomachus

```
    for p in range(2, floor(sqrt(n)) + 1):
        if A[p] != 0:
            j = p * p
            while j <= n:
                A[j] = 0
                j = j + p
    L = []
    for p in range(2, n+1):
        if A[p] != 0:
            L.append(A[p])
    return L
```

در قسمت اول این برنامه، آرایه‌ای (لیستی) پوچ به نام A و به طول $n+1$ ایجاد می‌کنیم و با نادیده گرفتن دو خانه اول آن، تمام اعداد طبیعی در محدوده ۲ تا n را در خانه‌های ۲ تا n آن می‌گذاریم. سپس در قسمت دوم برنامه، با شروع از عدد اول ۲ و ختم به $\lfloor\sqrt{n}\rfloor$، مرحله به مرحله تمام اعدادی را که مضربی از یک عدد اول باشند از آرایه حذف می‌کنیم. دستور $A[j] = 0$ به این معناست که عدد j مضرب عددی اول است و باید آن را غربال کرد. در قسمت سوم برنامه، آرایه A را بررسی می‌کنیم و تمام عناصر غیرصفر آن را که چیزی نیستند جز اعداد اول در محدوده ۲ تا n، در لیست خروجی L می‌گذاریم.

امروزه مسأله شناسایی و استفاده از اعداد اول بزرگ در چند حوزه کاربردی، از جمله در طراحی الگوریتم‌های جستجو و طراحی الگوریتم‌های رمزنگاری پیش می‌آید. (مثلاً کتاب [۶] را ببینید.) به همین دلیل شناسایی سریع اعداد اول بزرگ، مسأله‌ای کاربردی است. البته در بیشتر موارد آنچه هدف است تعیین اول بودن یا نبودن عددی خاص است نه تعیین همه اعداد اول کوچک‌تر از عددی خاص. اما به هر حال مزیت الگوریتم اراتستنس این است که با آن مجموعه همه اعداد اول در یک بازه را به دست خواهیم آورد. بدیهی است که اجرای الگوریتم اراتستنس، هر چقدر n بزرگتر شود کندتر خواهد شد، اما در عمل می‌توان از آن به عنوان الگوریتمی کاربردی استفاده کرد.

۴.۲. **الگوریتم گاوس.** مسأله حل دستگاه‌های معادلات خطی، یکی از قدیمی‌ترین مسائل عددی است و حتی در زمان باستان، البته در ابعاد بسیار کوچک، مطرح بوده است.

معروف‌ترین الگوریتم برای این مسأله، الگوریتمی است که به یاد کارل فردریش گاوس[۱۳] (۱۷۷۷–۱۸۵۵) ، ریاضیدان آلمانی، «الگوریتم حذفی گاوس»[۱۴] نامگذاری شده است. اما قدمت این الگوریتمی که نام گاوس روی آن گذاشته شده است، به قرن‌ها قبل از زمان زندگی او برمی‌گردد و حتی گفته می‌شود که تا نیمه قرن بیستم به این الگوریتم این نام داده نشده بود. قدیمی‌ترین متنی که اصل این الگوریتم در آن دیده شده، نوشته‌ای چینی است به نام «نه فصل درباره هنر

[13] Carl Friedrich Gauss

[14] Gaussian elimination

ریاضی»[۱۵] که در ابتدای فصل هشتم آن، به چگونگی حل یک مسأله کشاورزی (که می‌توان به بیان امروزی آن را به شکل یک دستگاه خطی سه معادله در سه مجهولی بیان کرد) پرداخته شده است. نویسندگان و عصر دقیق نگارش این کتاب نامعلوم است اما در متنی که در سال ۲۶۳ میلادی نوشته شده است، ادعا شده است که آن کتاب چندین قرن قدمت دارد. این روش را بعدها از چین به ژاپن می‌برند و از آنجا به اروپا. در اروپا، افرادی مانند ایزاک نیوتن در قرن هیجدهم و کارل گاوس در قرن نوزدهم این روش را برای حل دستگاه‌های معادلات خطی استفاده کردند و به بقیه نیز شناساندند و آن را روش رایج برای حل دستگاه‌های معادلات خطی در اروپا کردند. نیوتن این روش را در اوائل قرن هیجدهم در یکی از کتاب‌هایش ذکر کرده بود و مدعی بود که خودش این روش حذفی را کشف کرده است. البته به هیچ وجه بعید نیست که افرادی مانند نیوتن و گاوس، نه تنها کاربر این الگوریتم بوده‌اند، بلکه خود بدون هیچ پیش‌زمینه‌ای چنین الگوریتم ساده‌ای را کشف کرده باشند.

کارل فردریش گاوس را به حق یکی از بزرگترین ریاضیدانان تاریخ می‌دانند. صرف‌نظر از این الگوریتمی که به اشتباه به نام او معروف شده است، الگوریتم‌های مهم دیگری کشف شده‌اند که می‌توان یا کاشف آن الگوریتم‌ها را گاوس دانست یا آنکه نظریه‌های گاوس را مبنای آن الگوریتم‌ها دانست. الگوریتم «تبدیل فوریه سریع» یکی از آن الگوریتم‌های کاربردی است که در سال ۱۹۶۵ در مقاله [۵] معرفی شده است، اما طبق آنچه در مقاله [۹] ذکر شده است، اصل این الگوریتم را گاوس در سال ۱۸۰۵ منتشر کرده است.

۱.۴.۲. *طراحی الگوریتم.* الگوریتم گاوس در واقع متشکل از دو الگوریتم است و در دو مرحله به حل دستگاه معادلات خطی $Ax = b$ می‌پردازد: مرحله حذفی پیشرو و مرحله جایگذاری پسرو.

در مرحله حذفی پیشرو، ماتریس ضرایب A را با حذف عناصر زیر قطر اصلی آن به ماتریس بالامثلثی A' و بردار b را هم به بردار b' تبدیل می‌کنیم؛ یعنی دستگاه معادلات خطی $Ax = b$ را به دستگاه معادلات خطی $A'x = b'$ تبدیل می‌کنیم. با الگوریتم برای رسیدن به یک ماتریس بالامثلثی، هر بار یکی از عناصر روی قطر اصلی ماتریس را به عنوان «محور» و سطر آن را به عنوان «سطر محوری» می‌گیریم و عناصر زیرین آن عنصر در همان ستون را با اضافه کردن ضریبی از «سطر محوری» به سطرهای زیرین آن صفر می‌کنیم. به عبارت دیگر، از حذف «ضرایب زیر قطر» مجهول x_1 شروع می‌کنیم و به همین ترتیب، ضرایب مجهولات در زیر قطر را صفر می‌کنیم تا آنکه تنها ضریب زیر قطر مجهول x_{n-1} را حذف کنیم. به علت حرکت پیشرونده الگوریتم از حذف ضرایب x_1 تا حذف ضریب x_{n-1}،به آن «الگوریتم حذفی پیشرو» گفته می‌شود.

در مرحله جایگذاری پسرو، با الگوریتم بردار مجهول x در دستگاه جدید $A'x = b'$ را با جایگذاری متوالی مقادیر معلوم محاسبه می‌کنیم؛ یعنی برای پیدا کردن بردار مجهولات، ابتدا با تنها یک تقسیم، مجهول x_n را از آخرین معادله پیدا می‌کنیم و سپس، با جایگذاری مقدار آن در معادله قبل از آن، مجهول x_{n-1} را معلوم می‌کنیم. به همین ترتیب با جایگذاری مقادیر مجهول تاکنون پیدا شده در معادلات قبل‌تر مجهولات بعدی را تا x_1 پیدا می‌کنیم. به علت حرکت پسرونده الگوریتم در جایگذاری مقادیر مجهول معلوم شده در معادلات ماقبل، به آن «الگوریتم جایگذاری پسرو» گفته می‌شود.

[15] The Nine Chapters on the Mathematical Art

الگوریتم گاوس را می‌توان روی هر دستگاه معادلات خطی اجرا کرد. اگر دستگاه جواب یکتا داشته باشد، الگوریتم جواب یکتا را پیدا می‌کند؛ و اگر دستگاه جواب یکتا نداشته باشد اجرای الگوریتم ادامه نخواهد یافت. اگر در نقطه‌ای در فرایند تبدیل دستگاه اصلی به دستگاه بالامثلثی، محور صفر شد اجرای الگوریتم باید موقتاً یا دائماً متوقف شود؛ در چنین حالتی اگر با تعویض سطرها، بتوانیم محور را در سطر فعلی غیرصفر کنیم که اجرای الگوریتم ادامه پیدا می‌کند و گرنه دستگاه جواب یکتا نخواهد داشت. در حالتی که دستگاه جواب یکتا نداشته باشد، باید با بررسی دستگاه موجود مشخص کرد که آیا دستگاه جواب ندارد یا آنکه بی‌نهایت جواب دارد.

۲.۴.۲. *پیاده‌سازی الگوریتم.* برای پیاده‌سازی الگوریتم گاوس، دو تابع پایتونی برای پیاده‌سازی دو الگوریتم حذفی پیشرو و جایگذاری پسرو می‌نویسیم. با تابع حذفی پیشرو، بردار b را به ماتریس A می‌افزاییم و سپس مرحله به مرحله عناصر زیر عناصر قطری ماتریس را صفر می‌کنیم. از آنجا که اگر محور صفر باشد، نمی‌توان فرایند بالامثلثی کردن ماتریس را ادامه داد، در هر مرحله، بزرگترین عنصر در زیر عنصر محوری را می‌یابیم و سطر آن عنصر را با سطر محوری عوض می‌کنیم تا بزرگترین عنصر ممکن به عنوان عنصر محوری استفاده شود. سپس با کم کردن ضرایبی مناسب از سطر محوری از سطرهای زیرین آن، تمام عناصر زیر عنصر محوری را صفر می‌کنیم.

در صورتی که نتوان در مرحله‌ای، محوری غیرصفر را یافت، نتیجه می‌گیریم که دستگاه یا جواب ندارد یا آنکه بی‌نهایت جواب دارد. اگر آخرین عنصر در سطر محوری، عددی غیرصفر باشد، دستگاه جواب ندارد. در چنین حالتی، حداقل دو معادله در میان معادله‌ها قرار دارند که یا همه ضرایب در طرف چپ یکی از آن دو معادله با ضرایب متناظر با آنها در معادله دیگر برابر هستند اما طرف راست دو معادله برابر نیستند؛ یا آنکه همه نسبت‌های ضرایب در یکی از دو معادله به ضرایب متناظر با آنها در معادله دیگر، برابر هستند، اما نسبت طرف راست معادله‌ها با آن نسبت‌ها برابر نیست. اگر آخرین عنصر در سطر محوری، صفر باشد، دستگاه بی‌نهایت جواب دارد. در چنین حالتی، حداقل دو معادله در میان معادله‌ها قرار دارند که یا همه ضرایب در طرف چپ یکی از آن دو معادله با ضرایب متناظر با آنها در معادله دیگر، و طرف راست دو معادله نیز برابر هستند؛ یا آنکه همه نسبت‌های ضرایب در یکی از دو معادله به ضرایب متناظر با آنها در معادله دیگر و نسبت طرف راست معادله‌ها برابر هستند.

Applies Gaussian elimination to matrix A of a system's coefficients, augmented with vector b of the system's right-hand side values

```
def ForwardElimination(A, b):
    n = len(A)
    for i in range(0, n):
        A[i].append(b[i])
    for i in range(0, n):
        pivotrow = i
        for j in range(i+1, n):
            if abs(A[j][i]) > abs(A[pivotrow][i]):
                pivotrow = j
```

```
        if A[pivotrow][i] == 0:
            if A[pivotrow][n] != 0:
                return 'no solution'
            else:
                return 'infinitely many solutions'
        for k in range(0, n+1):
            A[i][k], A[pivotrow][k] = A[pivotrow][k], A[i][k]
        for j in range(i+1, n):
            temp = A[j][i] / A[i][i]
            for k in range(i, n+1):
                A[j][k] = A[j][k] - A[i][k] * temp
    return A
```

با تابع جایگذاری‌های پسرو، بردار مجهول x در دستگاه جدید $A'x = b'$ را با جایگذاری متوالی مقادیر معلوم محاسبه می‌کنیم. بدیهی است که این تابع را تنها وقتی باید فراخوانی کنیم که با فراخوانی تابع حذفی پیشرو، یک ماتریس بالامثلثی به دست آمده باشد و مشخص شده باشد که جواب دستگاه یکتا است.

Implements the backward substitution stage of Gaussian elimination

```
def BackwardSubstitution(A, b):
      A = ForwardElimination (B,b)
    n = len (A)
    x = [None] * n
    for i in range (n-1, -1,-1) :
        sum = 0.0
        for j in range (i+1,n) :
            sum = sum + A[i] [j] *x[j]
            x[i] = (A[i] [n] - sum) /A[i] [i]
        print (x)
```

یکی از موضوعاتی که باید در پیاده‌سازی الگوریتم گاوس به آن توجه کرد، موضوع انباشته شدن خطاهای گردسازی اعداد حقیقی است. از آنجا که تعداد عملیات حسابی الگوریتم، به ویژه وقتی که دستگاه بزرگ باشد، زیاد خواهد بود، احتمال تحریف جواب نهایی دستگاه از جواب دقیق وجود خواهد داشت. برای اطمینان از درستی خروجی الگوریتم، می‌توان در پایان جوابی را که با الگوریتم به دست می‌آید در معادلات جایگذاری کرد.

مسأله حل دستگاه‌های معادلات خطی مسأله‌ای بسیار کاربردی است. برای مثال، تحلیل یک مدار الکتریکی منجر به یک دستگاه معادلات خطی می‌شود و جواب آن دستگاه، مقادیر جریان‌های گذرنده از شاخه‌های مدار است. یا مثلاً در حوزه تحلیل داده‌ها، ممکن است در فرایند تحلیل، مسأله حل دستگاه معادلات خطی پیش آید. الگوریتم‌های متعددی برای حل دستگاه‌های معادلات خطی کشف شده‌اند. اما الگوریتم گاوس، الگوریتم اصلی برای حل دستگاه معادلات خطی است، به ویژه وقتی که ماتریس ضرایب دستگاه خلوت یا بزرگ نباشد. برای بررسی دقیق‌تر الگوریتم گاوس و آشنایی با چند الگوریتم دیگر برای حل دستگاه‌های معادلات خطی که آنها نیز از جنبه تاریخی و کاربردی جالب هستند، به کتاب [۴] مراجعه کنید.

۵.۲. **الگوریتم هرون.** مسأله محاسبه جذر اعداد از مسائلی است که از زمان باستان تاکنون مطرح بوده است. مثال‌هایی از مقدار تقریبی جذر اعداد روی الواح بابِلیان (که قدمت آن‌ها تقریباً به ۲۰۰۰ سال قبل از میلاد مسیح می‌رسد) نقش بسته است. یکی از آن مثال‌ها محاسبه قطر یک «در» مستطیلی شکل به طول ۴۰ و عرض ۱۰ است که منجر به محاسبه $\sqrt{40^2+10^2}=10\sqrt{17}$ می‌شود. بابِلیان به مقدار تقریبی $41+\frac{15}{60}$ رسیدند. اطلاعاتی ظاهراً در دست نیست که بتوان از روی آن به این پی برد که بابِلیان چگونه به چنین مقادیری رسیدند.

طبق مستندات موجود، هرون اسکندرانی[۱۶] (در قرن اول میلادی) اولین کسی بوده است که روشی را برای محاسبه جذر تقریبی اعداد در نوشته‌های خود ذکر کرده است. البته از آنجا که بابِلیان در نوشته‌های خود مثال‌هایی از مقدار جذر اعداد را ذکر کرده‌اند، «الگوریتم هرون» به نام «الگوریتم بابلی» نیز معروف شده است.

هرون از ریاضیدانان و مهندسان و مخترعان یونان باستان بود و در اسکندریه مصر به دنیا آمد. علاوه بر الگوریتم محاسبه جذر، فرمولی نیز برای محاسبه مساحت مثلث برحسب طول سه ضلع آن به نام هرون معروف شده است. طبق فرمول هرون، اگر a و b و c طول سه ضلع یک مثلث باشند و $p=\frac{a+b+c}{2}$ باشد، آنگاه مساحت مثلث $A=\sqrt{p(p-a)(p-b)(p-c)}$ خواهد بود. می‌توان مبتنی بر این فرمول الگوریتمی برای محاسبه مساحت مثلث نوشت؛ جالب اینجاست که استفاده از این فرمول نیازمند دانستن الگوریتمی برای محاسبه جذر است و هرون الگوریتمی نیز برای محاسبه جذر داشت.

علاوه بر الگوریتم هرون، الگوریتم‌های دیگری در گذشته نیز برای محاسبه جذر استفاده می‌شده است. برای مثال، در کتاب [۳] توضیحی مختصر راجع به الگوریتمی که چینیان و هندیان در قرن سوم استفاده می‌کردند و توضیحی مفصل راجع به الگوریتمی که مسلمانان و اروپاییان در قرون وسطی استفاده می‌کردند، داده شده است.

۱.۵.۲. *طراحی الگوریتم.* با الگوریتم هرون، برای محاسبه جذر یک عدد، از مقدار تقریبی اولیه‌ای شروع می‌کنیم و دنباله‌ای از مقادیر تقریبی را تولید می‌کنیم به گونه‌ای که هر جمله در دنباله مقادیر تقریبی، نسبت به جملات قبل از آن، به مقدار دقیق جذر عدد نزدیک‌تر باشد.

فرض کنید a عددی مثبت باشد. برای محاسبه $\sqrt{a}$، از مقدار تقریبی r شروع می‌کنیم. دو حالت پیش می‌آید: اگر $r>\sqrt{a}$ باشد پس $\frac{a}{r}<\sqrt{a}$ خواهد بود؛ و اگر $r<\sqrt{a}$ باشد، پس $\frac{a}{r}>\sqrt{a}$ خواهد بود. در هر دو حالت $\sqrt{a}$ عددی بین دو عدد r و $\frac{a}{r}$ خواهد بود. از آنجا که می‌توان با اطمینان گفت که میانگین حسابی دو عدد r و $\frac{a}{r}$ به عدد $\sqrt{a}$

[16] Heron of Alexandria

نزدیک‌تر است، مقدار تقریبی بعدی را میانگین دو عدد در نظر می‌گیریم؛ یعنی مقدار تقریبی بعدی $\frac{1}{2}(r + \frac{a}{r})$ خواهد بود. طبق همین نکته است که دنباله‌ای از مقادیر تقریبی را تولید می‌کنیم.

۲.۵.۲. *پیاده‌سازی الگوریتم.* امروزه نیز الگوریتم هرون به عنوان یکی از ساده‌ترین و بهترین الگوریتم‌های محاسبه تقریبی جذر اعداد استفاده می‌شود. برای پیاده‌سازی الگوریتم هرون، از مقدار تقریبی r_0 شروع می‌کنیم و بر مبنای این رابطه بازگشتی، دنباله‌ای از مقادیر تقریبی را تولید می‌کنیم:

$$r_{n+1} = \frac{1}{2}\left(r_n + \frac{a}{r_n}\right), \quad n = 0, 1, 2, \ldots.$$

برای محاسبه مقدار تقریبی $\sqrt{a}$ از r_0 شروع می‌کنیم و طبق رابطه بازگشتی، مقادیر r_1 و r_2 و غیره را محاسبه می‌کنیم. از آنجا که دنباله مقادیر $\{r_n\}$ بی‌پایان است، عملاً لازم است حدی برای تعداد تکرارهای الگوریتم مشخص کنیم. هرچقدر تعداد تکرارهای الگوریتم بیشتر باشد، اجرای آن بیشتر طول خواهد کشید اما آخرین مقدار تقریبی $\sqrt{a}$ به مقدار دقیق آن هم نزدیک‌تر خواهد بود. با وجود گذاشتن حدی روی تعداد تکرارهای الگوریتم، برنامه را به گونه‌ای می‌نویسیم که اگر قبل از رسیدن به حداکثر تعداد تکرارها، آخرین مقدار تقریبی که رایانه محاسبه کرده است به مقدار دقیق خیلی نزدیک باشد، اجرای آن متوقف شود. این معیار برای توقف اجرای برنامه به این معناست که اگر ε آستانه خطا (اختلاف مقدار تقریبی و مقدار دقیق) باشد، آنگاه $|r_n^2 - a| < \varepsilon$ باشد.

Computes square roots by Heron's algorithm

```
def Heron(a, r, eps, N):
    n = 1
    print('iteration:', 0, ' r:', r, sep='')
    while n <= N:
        r = 1/2 * (r + a/r)
        print('iteration:', n, ' r:', r, sep='')
        if abs(r**2 - a) < eps:
            break
        n = n + 1
```

۲.۶. **الگوریتم نیوتن.** مسائل حل معادلات خطی و مربعی، در تمدن‌های باستانی از جمله تمدن‌های بین‌النهرین و مصر باستان و چین باستان و هند باستان مطرح بود. قرن‌ها بعد ریاضیدانان مسلمان از جمله خوارزمی (۷۸۰–۸۵۰) و خیام (۱۰۴۸–۱۱۳۱) و ابن‌بنای مراکشی (۱۲۵۶–۱۳۲۱)، نیز راه‌حل‌هایی را برای معادلات درجه اول و دوم و سوم منتشر کردند. بعد از آن ریاضیدانان اروپایی، از جمله لئوناردو فیبوناچی (۱۱۷۰–۱۲۵۰) و جرولامو کاردانو[۱۷] و

[17] Gerolamo Cardano

(۱۵۰۱–۱۵۷۶)، به حل چنین معادلاتی پرداختند. یکی از نقاط اوج در تاریخ حل معادلات غیرخطی، انتشار الگوریتمی است که ما امروزه آن را به عنوان «الگوریتم نیوتن» می‌شناسیم.

ایزاک نیوتن[۱۸] (۱۶۴۲–۱۷۲۷) ریاضیدان و فیزیکدان و دین‌شناس انگلیسی بود. نیوتن در ریاضیات و در فیزیک نظریه‌پرداز و نوآور بود. یکی از کارهای برجسته او در ریاضیات، پیشنهاد روشی برای حل معادلات غیرخطی بود. نیوتن الگوریتمش را با مثال‌های عددی، در سال ۱۶۶۹ در یکی از کتاب‌هایش توصیف کرد.

الگوریتم نیوتن به نام «الگوریتم نیوتن–رافسون» نیز معروف است. جوزف رافسون[۱۹] (۱۶۶۸–۱۷۱۵) از ریاضیدانان انگلیسی معاصر نیوتن بود. او همین الگوریتم را در سال ۱۶۹۰ در یکی از رساله‌هایش منتشر کرد. در آن زمان نیوتن الگوریتم خود را منتشر نکرده بود اما اساس کار او را ریاضیدانی به نام جان والیس[۲۰] (۱۶۱۶–۱۷۰۳) در سال ۱۶۸۵ منتشر کرده بود.

۱.۶.۲. *طراحی الگوریتم.* تقریباً همه الگوریتم‌هایی که برای حل معادلات غیرخطی استفاده می‌شوند، جواب تقریبی معادله را در چند مرحله می‌یابند؛ یعنی از یک حدس اولیه شروع می‌کنند و در هر یک از مراحل بعدی جواب دقیق‌تری را جایگزین جواب فعلی می‌کنند. الگوریتم نیوتن هم از یک مقدار تقریبی اولیه شروع می‌کند و گام به گام دنباله‌ای از مقادیر تقریبی تولید می‌کند؛ با این هدف که هر جمله در دنباله نسبت به جملات قبل از آن، به ریشه معادله نزدیک‌تر باشد.

نکته اصلی در الگوریتم نیوتن این است که تابع غیرخطی و مشتق‌پذیر $f(x)$ را با تابعی خطی تقریب بزنیم. با این فرض که x_0 مقدار تقریبی اولیه ریشه باشد، از آنجا که تابع $f(x)$ مشتق‌پذیر است و شیب خط مماس بر نمودار تابع $f(x)$ در نقطه $(x_0, f(x_0))$، برابر با مشتق تابع در آن نقطه است، معادله خط مماس بر منحنی تابع $f(x)$ در نقطه $(x_0, f(x_0))$ را می‌توان به این صورت نوشت:

$$l(x) = f'(x_0)(x - x_0) + f(x_0).$$

چون در نقطه x_0، مقدار تابع خطی $l(x)$ و تابع غیرخطی $f(x)$ یکسان است و در نقاط مجاور x_0، مقدار $l(x)$ به مقدار $f(x)$ نزدیک است، می‌توان فرض کرد که نقطه تقاطع خط مماس با محور x نیز نزدیک به نقطه تقاطع تابع $f(x)$ با محور x باشد. نقطه تقاطع خط مماس با محور x، جایی است که $l(x) = 0$ شود:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

در صورتی که حدس اولیه x_0 نسبتاً نزدیک به ریشه انتخاب شده باشد، انتظار این است که x_1 نسبت به نقطه x_0، به ریشه واقعی معادله نزدیک‌تر باشد. با تکرار همین روند می‌توان گام به گام دنباله‌ای به دست آورد از جملاتی که مقادیر تقریبی ریشه معادله هستند و هر جمله در دنباله نسبت به جملات قبل از آن، به ریشه نزدیک‌تر باشد. ارتباط هر جمله دنباله و جمله بعدی آن را می‌توان با این رابطه بازگشتی مشخص کرد:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, \ldots.$$

[18] Isaac Newton

[19] Joseph Raphson

[20] John Wallis

نیوتن در نوشته‌های خود، روش یافتن ریشه‌های یک تابع چندجمله‌ای را توضیح داده است. اما او صریحاً نه از خط مماس استفاده کرد و نه از مشتق؛ بلکه روش خود را بر این مبنا که می‌توان هر تابع چندجمله‌ای را با یک خط تقریب زد، توضیح داده است. اما رافسون روش خود را به نحوی توضیح داده است که استنباط رابطه بازگشتی از آن راحت‌تر است. مزیت رابطه بازگشتی این است که می‌توان مستقیماً مبتنی بر آن راه‌حلی الگوریتمی را برای مسأله توصیف کرد.

جالب است به ارتباط الگوریتم هرون با الگوریتم نیوتن توجه کنیم: الگوریتم هرون حالت خاصی از الگوریتم نیوتن است. از آنجا که محاسبه $\sqrt{a}$ معادل با یافتن ریشه مثبت معادله مربعی $f(x) = x^2 - a = 0$ است، با جایگذاری تابع $f(x)$ و تابع مشتق آن در فرمول بازگشتی الگوریتم نیوتن، فرمول بازگشتی الگوریتم هرون به دست می‌آید:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x_n^2 - a}{2x_n} = \frac{x_n^2 + a}{2x_n} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right).$$

۲.۶.۲. *پیاده‌سازی الگوریتم.* الگوریتم نیوتن هم الگوریتمی ساده است و هم کارا؛ و به همین دلیل امروزه معروف‌ترین و پرکاربردترین الگوریتم برای حل معادلات غیرخطی تک‌مجهولی است. می‌توان برنامه پیاده‌ساز الگوریتم نیوتن را به این صورت نوشت:

Finds a root of an equation by Newton's algorithm

```
def Newton(x, eps, delta, N):
    n = 1
    print('iteration:', 0, ' x:', x, ' f(x):', f(x), sep='')
    while n <= N:
        if abs(df(x)) < delta:
            print("small derivative")
            break
        e = f(x) / df(x)
        x = x - e
        print('iteration:', n, ' x:', x, ' f(x):', f(x), sep='')
        if abs(e) < eps and abs(f(x)) < eps:
            break
        n = n + 1
```

از آنجا که در هر تکرار این الگوریتم، محاسبه مقادیر جدیدی از تابع و مشتق آن لازم است نوشتن دو «تابع پایتونی» برای محاسبه تابع و مشتق آن نیز لازم است.

با وجود سادگی و کارایی الگوریتم، باید در پیاده‌سازی آن به نکاتی توجه کرد. موضوع این است که ممکن است دنباله جواب‌های تقریبی که با این الگوریتم تولید می‌شود، به ریشه معادله همگرا نباشد. برای افزایش سرعت همگرایی الگوریتم، لازم است به دنبال جواب اولیه‌ای باشیم که به ریشه نزدیک باشد. شاید بتوان با شناخت از مسأله واقعی و همچنین رسم

نمودار تابع، قبل از اجرای الگوریتم، مقدار تقریبی اولیه خوبی برای ریشه حدس زد. در بیشتر موارد، حدس اولیه خوب باعث می‌شود که الگوریتم به سرعت به ریشه همگرا شود. اما در مواردی حتی حدس اولیه خوب نیز لزوماً همگرایی سریع یا حتی همگرایی را نتیجه نخواهد داد. برای مثال، در حالاتی که $f'(x_n) = 0$ یا نزدیک به صفر باشد، اجرای الگوریتم باید متوقف شود؛ و در حالاتی که $f''(x_n) = 0$ باشد، دنباله جواب‌های تقریبی الگوریتم ممکن است واگرا شود یا آنکه به کندی به ریشه همگرا شود.

همچنین از آنجا که در حالات مطلوب انتظار داریم که با الگوریتم در چند تکرار به جواب برسیم، می‌توان با تعیین حد بالایی برای تکرارهای الگوریتم، حالات نامطلوب (همگرایی کند یا واگرایی یا نوسان جواب‌ها به چپ و راست) را سریعاً شناسایی کرد. و البته باید در برنامه پیاده‌ساز الگوریتم امکان اینکه $f'(x)$ در جایی در طول محاسبات صفر یا نزدیک به صفر شود را در نظر داشت. اگر $f'(x_n) = 0$ یا نزدیک به صفر باشد، می‌توان آخرین مقدار تقریبی را که الگوریتم محاسبه کرده است، به عنوان ریشه تقریبی معادله استفاده کرد.

برای اطمینان از آنکه مقدار آخرین جواب تقریبی که الگوریتم محاسبه کرده است، به مقدار دقیق ریشه خیلی نزدیک است، لازم است معیاری برای اندازه‌گیری خطا در نظر بگیریم. در اینجا اندازه خطا را اختلاف دو جمله متوالی در دنباله تقریب‌ها در نظر می‌گیریم؛ یعنی به عنوان معیار اندازه خطا از نامساوی $|x_n - x_{n-1}| < \varepsilon$ استفاده می‌کنیم. البته از آنجا که ممکن است با وجود کوچک بودن مقدار خطا، باز آخرین مقدار x به ریشه نزدیک نباشد، باید با توجه به آخرین مقدار $f(x)$، نزدیک بودن یا نبودن x به ریشه را بررسی کرد.

پس اجرای الگوریتم تا وقتی ادامه می‌یابد که یا مقدار مشتق تابع از آستانه‌ای کوچک‌تر شود یا مقدار خطا از آستانه‌ای کمتر شود یا آنکه تعداد تکرارها از آستانه‌ای بیشتر شود. از آنجا که برنامه به گونه‌ای نوشته شده است که رایانه مقدار جواب‌های تقریبی را به محض محاسبه چاپ می‌کند، در صورت توقف اجرای الگوریتم قبل از به پایان رسیدن تکرارها، آخرین جوابی که چاپ شده است به عنوان مقدار تقریبی ریشه استفاده خواهد شد.

الگوریتم نیوتن را بیشتر در کتاب‌هایی می‌توان یافت که به الگوریتم‌های عددی اختصاص داده شده‌اند. یکی از کتاب‌های معتبر و خواندنی برای آشنایی با بسیاری از الگوریتم‌های عددی (از جمله الگوریتم نیوتن) و نحوه پیاده‌سازی آنها کتاب [۴] است.

۷.۲. **الگوریتم هرنر.** توابع چندجمله‌ای دسته‌ای مهم از توابع هستند که در زمینه‌های مختلف علمی و مهندسی مکرراً پیش می‌آیند. دقیقاً معلوم نیست که مسائل درباره توابع چندجمله‌ای (مانند جمع و تفریق و ضرب و تقسیم توابع چندجمله‌ای) از چه زمانی مطرح شدند. اما حداقل این را می‌توان با اطمینان گفت که نیوتن و ریاضیدانان بعد از او، توجه‌ای ویژه به توابع چندجمله‌ای داشتند.

پایه‌ای‌ترین مسأله‌ای که می‌توان درباره توابع چندجمله‌ای مطرح کرد، نحوه محاسبه مقادیر چنین توابعی است. گرچه با الگوریتم‌های ساده‌ای می‌توان مقادیر توابع چندجمله‌ای را محاسبه کرد، زیباترین و بهترین الگوریتم برای این مسأله الگوریتمی قدیمی است که آن را ویلیام هرنر[۲۱] (۱۷۸۶–۱۸۳۷) ریاضیدان انگلیسی در اوائل قرن نوزدهم منتشر کرد و به همین دلیل آن الگوریتم به عنوان «قاعده هرنر»[۲۲] معروف شده است. با وجود اینکه الگوریتم، منتسب به هرنر است،

[21] William George Horner

[22] Horner's rule

اما طبق مستندات، ایزاک نیوتن از همین الگوریتم ۱۵۰ سال قبل از هرنر استفاده می‌کرده است و حتی روشی معادل با این الگوریتم در قرن سیزدهم در چین استفاده می‌شده است.

۱.۷.۲. *طراحی الگوریتم.* شکل کلی و متعارف توابع چندجمله‌ای درجه n را در نظر بگیرید:

$$P_n(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0.$$

می‌توان با محاسبه مستقیم تک‌تک جملات تابع و سپس محاسبه مجموع آن جملات، مقدار تابع را در کل با $\frac{n(n+1)}{2}$ ضرب و n جمع محاسبه کرد. طبق الگوریتم هرنر، برای محاسبه مقدار تابع از «نمایش تودرتوی» چندجمله‌ای استفاده می‌کنیم. نکته این است که می‌توانیم شکل کلی تابع $P_n(x)$ را به این صورت نامتعارف بازنویسی کنیم:

$$P_n(x) = (\ldots(a_n x + a_{n-1})x + \ldots)x + a_0.$$

با الگوریتم برای ارزیابی ضرب‌های تودرتو، از ارزیابی عبارت درون داخلی‌ترین کمانک (که یک چندجمله‌ای درجه اول است) شروع می‌کنیم و گام به گام به سمت بیرون حرکت می‌کنیم، تا آنکه نهایتاً مقدار چندجمله‌ای درجه n را تعیین کنیم.

در حالی که با الگوریتمی سرراست می‌توان هر تابع چندجمله‌ای درجه n را با $\frac{n(n+1)}{2}$ ضرب و n جمع محاسبه کرد، با الگوریتم هرنر می‌توان هر تابع چندجمله‌ای درجه n را با n ضرب و n جمع ارزیابی کرد. (به ازای هر ضریب صفری، یکی از تعداد جمع‌ها کم خواهد شد.) در واقع، الگوریتم هرنر، الگوریتمی بهینه برای محاسبه مقادیر توابع چندجمله‌ای است و طراحی الگوریتمی کاراتر از آن ممکن نیست.

الگوریتم هرنر در بعضی از کتاب‌های الگوریتم مورد بحث قرار گرفته است. برای مثال، کتاب [۱۴] را ببینید.

۲.۷.۲. *پیاده‌سازی الگوریتم.* در عمل ساده‌ترین و سریع‌ترین الگوریتم برای ارزیابی توابع چندجمله‌ای الگوریتم هرنر است. می‌توان برنامه پیاده‌ساز الگوریتم هرنر را به این صورت نوشت:

Evaluates a polynomial at a given point by Horner's rule

```
def Horner(P, x):
    n = len(P) - 1
    p = P[n]
    for i in range(n-1, -1, -1):
        p = x * p + P[i]
    return p
```

همان‌طور که واضح است، برنامه پیاده‌ساز الگوریتم بسیار کوتاه است. توجه کنید که با این برنامه به طور ضمنی، طبق نمایش تودرتوی تابع چندجمله‌ای به ارزیابی آن می‌پردازیم. یعنی برای پیاده‌سازی الگوریتم لازم نیست تابع چندجمله‌ای را صریحاً به شکل تودرتو نمایش دهیم؛ تنها مشخص کردن ضرایب تابع چندجمله‌ای برای ارزیابی آن کافی است. به علاوه لازم نیست نتایج میانی را ذخیره کنیم، چون به محض محاسبه کمیت‌های میانی، از آنها سریعاً در گام بعدی استفاده خواهد شد.

۳. چند نکته و پیشنهاد

ما در این مقاله هفت الگوریتم قدیمی را بررسی کردیم. با بررسی همین الگوریتم‌ها می‌توان به نتایجی رسید. الگوریتم‌هایی که در اینجا بررسی شدند این پیام را همراه خود دارند که مفهوم الگوریتم، گرچه به طور ضمنی، از چند هزار سال قبل مورد توجه ریاضیدانان بوده است. حتی عامه مردم نیز با مسائل الگوریتمی مواجه می‌شدند. به عنوان مثال، معمای الگوریتمی معروف «کشاورز – گرگ – بز – کلم» و دو معمای الگوریتمی دیگر که با عنوان «معماهای عبور از رودخانه» شناخته می‌شوند و از جمله مسائلی هستند که عامه مردم را به خود جذب می‌کنند، در متنی بسیار قدیمی ذکر شده‌اند که بعضی از افراد آن را نوشته کشیش و شاعری انگلیسی به نام آلکوئین یورکی[۲۳] (۷۳۵–۸۰۴) می‌دانند. (سه معمای عبور از رودخانه در کتاب [۱۵] که به معرفی ۱۷۲ معمای الگوریتمی اختصاص داده شده است، آورده شده‌اند.)

شاید تا قبل از اختراع رایانه، کسانی که الگوریتمی را کشف می‌کردند یا از آن استفاده می‌کردند، متوجه تفاوت ظریف اما اساسی آنها با بقیه روش‌های حل مسأله نبودند. اختراع رایانه بود که نه تنها این تفاوت ظریف را برجسته و نمایان کرد، بلکه بستر را برای کشف و استفاده از هزاران الگوریتم دیگر هموار کرد. تا قبل از اختراع رایانه، افرادی که الگوریتمی را کشف می‌کردند، چندان به کاربرد گسترده آن فکر نمی‌کردند چون ابزاری برای این کار در اختیار نداشتند. اما بعد از اختراع رایانه، الگوریتم‌ها عمدتاً از این جهت مد نظر بوده‌اند که در عمل و برای حل مسائل واقعی سودمند باشند.

در بررسی تاریخی الگوریتم‌ها، به این واقعیت توجه کردیم که بعضی از الگوریتم‌ها را ممکن است در طول دهه‌ها یا قرن‌ها افراد مختلفی کشف (مجدد) کرده و استفاده کرده باشند. نامگذاری یک الگوریتم به اسم فرد خاصی که از آن استفاده کرده است، لزوماً به این معنا نیست که کاشف الگوریتم آن فرد و مبدأ زمانی کشف الگوریتم، زمان زندگی آن فرد خاص بوده است. مثال‌هایی را دیدیم که الگوریتمی را شخصی کشف کرده است و بعدها به نام کسی دیگر معروف شده است.

نگاه جامع و دقیق به الگوریتم‌ها از چشم‌انداز تاریخی، مستلزم دسته‌بندی الگوریتم‌ها است. یک معیار، دسته‌بندی الگوریتم‌ها بر اساس مسأله‌ای است که حل می‌کنند؛ یعنی الگوریتم‌هایی که در طول تاریخ برای حل یک مسأله خاص (مثلاً مسأله یافتن ریشه‌های معادلات غیرخطی) یا حل دسته‌ای از مسائل که مبنای نظری آنها مشترک است (مانند مسائل نظریه اعداد) پیشنهاد شده‌اند، در یک دسته گذاشته شوند. معیار منطقی دیگر برای دسته‌بندی الگوریتم‌ها، این است که آیا آنها قبل از اختراع رایانه کشف شده‌اند یا بعد از اختراع رایانه. ما در این مقاله، اولاً خود را محدود به بررسی الگوریتم‌هایی کردیم که قبل از اختراع رایانه کشف شده‌اند و ثانیاً تنها به بررسی الگوریتم‌هایی پرداختیم که برای حل مسائل عددی پیشنهاد شده‌اند. در حقیقت تا قرن هفدهم میلادی، عمده دستاوردهای ریاضیدانان را می‌توان یا در زمینه حساب (نظریه اعداد) دانست یا در زمینه هندسه. از قرن هیجدهم که آن را زمان طلوع ریاضیات جدید می‌دانند، مسائلی (مانند حل معادلات غیرخطی و درون‌یابی و انتگرال‌گیری عددی) مطرح شدند که امروزه در شاخه «آنالیز عددی» مورد بررسی قرار می‌گیرند. آنقدر این موضوعات تا قرن بیستم بسط پیدا کرد و به موضوعات غالب در ریاضیات تبدیل شد که در دهه ۱۹۴۰، کاربران رایانه، آن را عمدتاً ابزاری برای حل سریع مسائل عددی می‌دانستند. چندین سال بعد از اختراع رایانه بود که به تدریج کاربرد رایانه برای حل مسائل الگوریتمی در دیگر حوزه‌های ریاضی، مانند مسائل گراف و مسائل ترکیبیاتی و مسائل هندسی روشن شد.

---

[23] Alcuin of York

بررسی تاریخی حوزه‌های دیگری از الگوریتم‌ها، از جمله حوزه‌های الگوریتم‌های هندسی یا الگوریتم‌های گراف یا الگوریتم‌های ترکیبیاتی، موضوع تحقیقاتی و آموزشی مفید و جالبی است؛ به ویژه آنکه در این زمینه‌ها تاکنون تحقیقی جامع منتشر نشده است. در این زمینه‌ها نمونه‌هایی از مسائل و الگوریتم‌های برجسته را می‌توان ذکر کرد:

- **الگوریتم بند کفشی**[۲۴]: محاسبه مساحت شکل‌های هندسی از مسائلی است که از زمان باستان مورد توجه هندسه‌دانان بوده است. ریاضیدانی به نام آلبرشت مایستر[۲۵] (۱۷۲۵–۱۷۸۸) مبتنی بر کارهای ریاضیدانان دیگر، فرمولی را برای محاسبه مساحت چندضلعی‌های ساده، در سال ۱۷۶۹ پیشنهاد کرد. نحوه رسیدن به این فرمول که به نام فرمول «بند کفشی» معروف است، در کتاب [۱۷] توضیح داده شده است.
- **الگوریتم پیمایش عمقی:** با این الگوریتم، می‌توان گرافی را پیمایش و با اطلاعات حاصل از پیمایش، مسائل متعددی را درباره گراف حل کرد. طبق مستندات، این الگوریتم در قرن نوزدهم برای یافتن مسیر خروجی از مازها استفاده می‌شده است [۱۶]. البته این الگوریتم را می‌توان تعمیمی از «قاعده دست راست» دانست که بشر از زمان باستان برای خروج از ماز آن را می‌دانسته است. طبق این قاعده، به محض ورود به ماز، دست راست خود را روی دیوار می‌گذاریم و آن را از روی دیوار برنمی‌داریم تا زمانی که از ماز خارج شویم.
- **الگوریتم فلوری و الگوریتم هایرهولزر:** لئونارد اویلر[۲۶] (۱۷۰۷–۱۷۸۳)، ریاضیدان سوئیسی در سال ۱۷۳۶ ثابت کرد که مسأله «هفت پل کونیگسبرگ» جواب ندارد. (ترجمه‌ای از مقاله اصلی اویلر و نکاتی تاریخی درباره مسأله در مقاله [۷] ذکر شده است.) جواب این مسأله به بیان امروزی، دوری در یک گراف است که از هر یال آن دقیقاً یک بار بگذرد. به همین دلیل بعدها چنین دورهایی را «دورهای اویلری» نامیدند. دو الگوریتم متفاوت برای مسأله یافتن دورهای اویلری، از ریاضیدانی آلمانی به نام کارل هایرهولزر[۲۷] در سال ۱۸۷۳ پس از مرگ او و از ریاضیدانی فرانسوی به نام میشل فلوری[۲۸] در سال ۱۸۸۳، منتشر شده است. (این دو الگوریتم در کتاب [۲] و مقاله [۷] بحث شده‌اند.)
- **مسأله مربع جادویی:** یکی از مسائل ترکیبیاتی که نباید آن را از چشم‌انداز تاریخی نادیده گرفت، مسأله «مربع جادویی» است. مربع جادویی مرتبه n را به بیان ریاضی، می‌توان به شکل ماتریسی $n \times n$ تصور کرد که هر یک از اعداد ۱ تا n^2 به گونه‌ای در خانه‌های آن گذاشته شده باشند که مجموع اعداد در هر سطر و مجموع اعداد در هر ستون و مجموع اعداد در هر دو قطر اصلی ماتریس، برابر با هم و $n(n^2+1)/2$ باشد. در تمدن‌های باستانی ایران و هند و چین، از چند هزار سال قبل چنین مربعاتی استفاده می‌شده است. دانشمندان مسلمان چنین مربعاتی را «مربع وفقی» می‌نامیدند و توجه‌ای ویژه به آنها داشتند. توجه آنان به این مسأله صرفاً از جنبه ریاضی نبوده است؛ از گذشته تاکنون افرادی معتقدند که چنین مربعاتی اگر با اعداد یا نمادهایی خاص پر شوند، برای رفع مشکلاتی (مثل درمان بیماری‌هایی خاص) مفید هستند. بعدها دانشمندان اروپایی با این

---

[24] shoelace

[25] Albrecht Meister

[26] Leonhard Euler

[27] Carl Hierholzer

[28] Michel Fleury

نوع ساختارهای عددی آشنا می‌شوند و اسم «مربع جادویی» را روی آن می‌گذارند. در کتاب [۳] مربعات جادویی از چشم‌انداز تاریخی بررسی شده است.

- **مسأله n - وزیر:** یکی دیگر از مسائل ترکیبیاتی که از چشم‌انداز تاریخی جالب است، مسأله n - وزیر است. هدف از حل این مسأله، گذاشتن n وزیر روی یک صفحه شطرنجی $n \times n$ است، به گونه‌ای که هیچ دو وزیری در یک سطر یا در یک ستون یا در یک قطر نباشند. این مسأله را ظاهراً برای اولین بار یک بازیکن شطرنج آلمانی به نام مکس بزِل (۱۸۲۴–۱۸۷۱) [۲۹] در سال ۱۸۴۸ مطرح کرد و کارل گاوس در سال ۱۸۵۰ راه‌حلی را برای مسأله ۸ وزیر منتشر کرد. نکاتی تاریخی درباره این مسأله و توضیحاتی درباره راه‌حل‌های آن و کاربردهای آن در مقاله [۱] ذکر شده است.

ما در این تحقیق، خود را محدود به الگوریتم‌هایی کردیم که قبل از اختراع رایانه کشف شده‌اند. یک موضوع تحقیقاتی جالب دیگر، بررسی الگوریتم‌های برجسته‌ای است که بعد از اختراع رایانه از دهه ۱۹۴۰ به بعد منتشر شده‌اند. تعداد چنین الگوریتم‌هایی بسیار زیاد است اما بعضی از آنها از این لحاظ که حل رایانه‌ای سریع مسائلی را ممکن کردند یا زمینه‌ساز کشف الگوریتم‌های دیگری شدند، الگوریتم‌های برجسته‌ای به حساب می‌آیند و سزاوار است که در وهله اول نقش این الگوریتم‌ها در پیشرفت‌های علمی و صنعتی مورد تحقیق قرار گیرد. چند مثال از نام (و تاریخ انتشار) الگوریتم‌های شاخصی که بعد از اختراع رایانه منتشر شده‌اند، عبارتند از: الگوریتم مرتب‌سازی ادغامی (۱۹۴۵)، الگوریتم سیمپلکس (۱۹۴۷)، الگوریتم دایکسترا (۱۹۵۹)، الگوریتم مرتب‌سازی سریع (۱۹۶۲)، الگوریتم کاراتسوبا (۱۹۶۲)، الگوریتم استراشن (۱۹۶۹) و الگوریتم کارمارکار (۱۹۸۴).

تحقیقات تاریخی در زمینه‌هایی مانند آنهایی که به آنها اشاره شد، وقتی جامع خواهد بود که ما بدانیم دستاوردهای الگوریتمی گذشتگان صرفاً راه‌حل‌هایی نیست که به شکل الگوریتم در متون آورده شده‌اند. در واقع، گذشتگان به چند طریق زمینه علمی الگوریتم‌ها را گسترش داده‌اند و راه را برای دستاوردهای الگوریتمی آیندگان هموار کرده‌اند. در مواردی، در یکی از متون قدیمی، راه‌حلی برای مسأله‌ای ذکر شده است که می‌توان آن راه‌حل را امروزه الگوریتم نامید. در مواردی، فرمولی یا قضیه‌ای در یک متن قدیمی آورده شده است که نمی‌توان آن را الگوریتم نامید، اما می‌توان مبتنی بر آن فرمول یا قضیه، الگوریتمی طراحی کرد و اینکه در مواردی، مسأله‌ای در یکی از متون قدیمی مطرح شده است اما یا الگوریتمی برای آن مشخص نشده است یا آنکه فقط نمونه یا نمونه‌های خاصی از مسأله حل شده است و بعدها افراد دیگری الگوریتم‌هایی را برای حل آن مسأله پیشنهاد کرده‌اند. برای مثال، گرچه اویلر الگوریتمی برای حل مسأله «هفت پل کونیگسبرگ» ارائه نکرد، اما جواب او به همین مسأله، منجر به شکل‌گیری نظریه گراف شد. از این منظر باید کار اویلر را از نقاط اوج در تاریخ الگوریتم‌ها دانست.

به علاوه، نباید ربط الگوریتم‌هایی را که در چند دهه گذشته منتشر شده‌اند با دستاوردهای ریاضیدانان در زمان‌های قبل از انتشار آن الگوریتم‌ها نادیده گرفت. برای مثال، الگوریتم‌های هندسی که بعد از اختراع رایانه طراحی شده‌اند، مبتنی بر همان اصولی هستند که اقلیدس آنها را در کتاب اصول خود پایه‌ریزی کرد. یا آنکه ما امروزه از الگوریتم‌هایی استفاده می‌کنیم که مبتنی بر قضیه باقیمانده چینی (که طبق مستندات، اولین بار در نوشته‌های سون تزو[۳۰] (در قرن سوم

[29] Max Bezzel

[30] Sun Tzu

میلادی) ، ریاضیدان چین باستان، آورده شده است) و قضیه فرما که طبق مستندات، اولین بار در نوشته‌های پیر دو فرما[۳۱] (۱۶۰۱–۱۶۶۵) ، ریاضیدان فرانسوی، آورده شده است طراحی شده‌اند و با آنها می‌توان مسائلی کاربردی را بسیار سریع‌تر حل کرد.

در زمینه تاریخ الگوریتم‌ها تاکنون فقط یک کتاب اختصاصی به زبان فرانسوی با عنوان «تاریخ الگوریتم‌ها: از سنگ‌ریزه تا ریزتراشه» منتشر شده است. البته این کتاب به زبان انگلیسی نیز ترجمه شده است [۳]. این کتاب گرچه ارزشمند و خواندنی است، اما بعضی از حوزه‌های الگوریتمی مانند الگوریتم‌های گراف را نادیده گرفته است و عمدتاً به معرفی الگوریتم‌های عددی پرداخته است. ظاهراً کتاب خاص دیگری که مفصلاً به بحث‌های تاریخی درباره الگوریتم‌ها پرداخته باشد، منتشر نشده است و باید چنین نکات پراکنده‌ای را در کتاب‌ها و مقالات مختلف یافت. به عنوان مثال، چهار جلد کتاب معروف «هنر برنامه‌نویسی رایانه‌ای» [۱۰، ۱۱، ۱۲، ۱۳] حاوی اطلاعات ارزشمندی درباره کشفیات الگوریتمی از زمان باستان تا عصر بعد از اختراع رایانه است. در میان کتاب‌های درسی الگوریتم، کتاب «راهنمای طراحی الگوریتم» [۲۰] بیشتر از بقیه کتاب‌ها به ذکر نکات تاریخی پرداخته است. یکی دیگر از کتاب‌های جذاب، کتاب [۱۹] است که در سال ۱۹۹۵ درباره زندگی و کشفیات ۱۵ رایانه‌دان برجسته منتشر شده است، و حاوی نکات تاریخی سودمندی درباره تعدادی از الگوریتم‌هایی است که در پنج دهه اول عصر بعد از اختراع رایانه کشف شده‌اند. به عنوان مثالی دیگر، می‌توان به کسانی که می‌خواهند با معماهای الگوریتمی، چه از جنبه آموزشی و چه از جنبه تاریخی، آشنا شوند، خواندن کتاب بی‌نظیر «معماهای الگوریتمی» [۱۵] را پیشنهاد کرد.

مراجع

1. J. Bell and B. Stevens, *A survey of known results and research areas for n-queens*. Discrete Mathematics, 309 (2009) no. 1, 1–31.
2. N.L. Biggs, E.K. Lloyd, and R.J. Wilson, *Graph theory 1736-1936*. Clarendon Press, 1976.
3. Jean-Luc Chabert, ed. *A History of Algorithms: From the Pebble to the Microchip*. Translated by Chris Weeks. Springer, 1998.
4. S. Chapra and R. Canale, *Numerical Methods for Engineers*, 7th ed. McGrawHill, 2016.
5. J.W. Cooley, and J.W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Mathematics of Computation, 19 (1965) no. 90, 297–301.
6. T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. MIT Press, 2022.
7. J. Ebert, *Computing Eulerian trails*, Information Processing Letters, 28 (1988) 93–97.
8. G.E. Forsythe, *What to do till the computer scientist comes*, American Mathematical Monthly, 75 (1968) no. 5, 454–462.
9. M.T. Heideman, D.H. Johnson, and C.S. Burrus, *Gauss and the history of the fast Fourier transform*, IEEE ASSP Magazine, 1 (1984) no. 4, 14–21.

[31] Pierre de Fermat

10. D.E. Knuth, *The art of Computer Programming, Volume 1: Fundamental Algorithms*, 3rd ed. Addison-Wesley, 1997.
11. D.E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 3rd ed. Addison-Wesley, 1998.
12. D.E. Knuth, *The art of Computer Programming, Volume 3: Sorting and Searching*, 2nd ed. Addison-Wesley, 1998.
13. D.E. Knuth, *The art of Computer Programming, Volume 4A, Combinatorial Algorithms, Part 1*, Addison-Wesley, 2011.
14. A. Levitin, *Introduction to the Design & Analysis of Algorithms*, 3rd ed. Pearson, 2012.
15. A. Levitin and M. Levitin, *Algorithmic Puzzles*, Oxford University Press, 2011.
16. E. Lucas, *Recreations Mathematiques*, Gauthier-Villares, Paris, 1891.
17. J. O'Rourke, *Computational Geometry in C*, 2nd ed. Cambridge University Press, 1998.
18. K. Rosen, *Discrete Mathematics and its Applications*, 8th ed. McGraw-Hill, 2019.
19. D. Shasha and C. Lazere, *Out of Their Minds: The Lives and Discoveries of 15 Great Computer Scientists*, Copernicus, 1998.
20. S.S. Skiena, *Algorithm Design Manual*, 3rd ed. Springer, 2020.

Towards Mathematical Sciences

Vol. 6, No. 1, June 2026, pp. 61–83

DOI.10.22067/tmsj.2026.97896.1078

Expository Article

ALGORITHMS: FROM ANCIENT TIMES TO COMPUTER ERA

JAAFAR ALMASIZADEH [1*]

[1]Department of Mathematics and Statistics, University of Isfahan, Isfahan, Iran.
j.almasizadeh@sci.ui.ac.ir

Abstract. In this paper, we examine seven old and prominent algorithms that were discovered centuries before the invention of computers. The main purpose of selecting and introducing these algorithms is to show that algorithms have implicitly attracted the attention of mathematicians since ancient times, and that the concept of algorithm did not emerge only with the invention of computers. This manuscript enables readers to look at algorithms from a historical perspective and see part of the path algorithmists have traveled over the past 4000 years. Assuming that most readers of this paper are enthusiasts who have just started learning algorithms, in addition to mentioning historical notes, we also discuss the educational and practical aspects of these seven algorithms. From an educational perspective, the design process of the seven algorithms based on mathematical arguments is explained. From a practical perspective, the implementation of the seven algorithms in the Python programming language is explained. Finally, several comments and suggestions are provided for those who want to deal with algorithms more closely from a historical perspective.

2020 Mathematics Subject Classification. 01A99, 68W99, 97D50, 97P99

Keywords. Mathematics, Algorithm; Problem; Computer; History;

Date: Received 17-02-2026 Revised 30-04-2026 Accepted 21-06-2026 Available Online 21-06-2026