

به نام خدا

طراحی الگوریتم‌ها

آرش شفیعی



- مقدمه‌ای بر الگوریتم‌ها از کرمن، لایرسون، ریوست، و استاین<sup>1</sup>
- طراحی الگوریتم از کلاینبِرگ و تاردوس<sup>2</sup>
- اصول الگوریتم‌ها از ریچارد نئاپولیتان<sup>3</sup>
- هنر برنامه نویسی از دونالد کنوث<sup>4</sup>

---

<sup>1</sup> Introduction to Algorithms, by Cormen, Leiserson, Rivest, and Stein

<sup>2</sup> Algorithm Design, by Jon Kleinberg and Eva Tardos

<sup>3</sup> Foundations of Algorithms, by Richard Neapolitan

<sup>4</sup> The Art of Computer Programming, by Donald Knuth

## 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 الجبروالمقابل را نیز به تألیف رسانده است که کلمه جبر<sup>1</sup> 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. احتمالاً 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.

---

<sup>1</sup> Algebra

- کلمه الگوریتم در بسیاری از کتاب‌ها برای الگوریتم اقلیدس<sup>1</sup> که برای پیدا کردن بزرگ‌ترین مقسوم‌علیه مشترک<sup>2</sup> دو عدد به‌کار می‌رود و در کتاب اصول اقلیدس<sup>3</sup> توصیف شده است، استفاده 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. در این صورت خواهیم داشت  $0 \leq r < n$ .
  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 مرحله ۱ می‌رویم.

---

<sup>1</sup> Euclid's algorithm

<sup>2</sup> Greatest common divisor

<sup>3</sup> Euclid's Element

- الگوریتم در واقع یک روند<sup>1</sup> یا دستورالعمل<sup>2</sup> برای حل یک مسئله 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.

---

<sup>1</sup> procedure

<sup>2</sup> recipe

- می‌توان گفت یک الگوریتم ابزاری است برای حل یک مسئله محاسباتی معین.
- یک مسئله با تعدادی گزاره رابطه 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 همه نمونه‌های یک مسئله، خروجی‌ها با دریافت ورودی‌ها تولید شوند. بنابراین روند یک الگوریتم در رابطه بین ورودی‌ها و خروجی‌ها صدق می‌کند.
- به عنوان مثال، فرض کنید می‌خواهیم دنباله‌ای از اعداد را با ترتیب صعودی مرتب کنیم. این مسئله که مسئله مرتب سازی<sup>1</sup> نام دارد، یک مسئله بنیادین در علوم کامپیوتر به حساب می‌آید که منشأ به وجود آمدن بسیاری از روش‌های طراحی الگوریتم نیز می‌باشد.

---

<sup>1</sup> sorting problem

- 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  $\langle a_1, a_2, \dots, a_n \rangle$  و خروجی مسئله عبارت است از دنباله‌ای به صورت  $\langle a'_1, a'_2, \dots, a'_n \rangle$  که از جابجا کردن عناصر دنباله ورودی به دست آمده است به طوری که  $a'_1 \leq a'_2 \leq \dots \leq a'_n$ .
- 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<sup>1</sup> تشکیل شده است از یک ورودی معین و شرح ویژگی خروجی مسئله. بنابراین دنباله ورودی  $\langle 58, 42, 36, 42 \rangle$  به علاوه شرح مسئله مرتب سازی یک نمونه از مسئله مرتب سازی نامیده می‌شود.

---

<sup>1</sup> instance of a problem



- 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 رابطه 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 نمونه مسئله با تعیین مقادیر پارامترهای مسئله به دست می‌آید.
- یک الگوریتم، روندی گام به گام است برای پیدا کردن جواب یک مسئله است.

- سرعت اجرای مسئله مرتب سازی به اندازه ورودی یعنی تعداد عناصر دنباله نامرتب و روند الگوریتم بستگی دارد.
- الگوریتم‌های زیادی برای حل یک مسئله وجود دارند که هر کدام می‌توانند مزایا و معایبی داشته باشند. به طور مثال یک الگوریتم از میزان حافظه بیشتری استفاده می‌کند، اما زمان کمتری برای محاسبه نیاز دارد و الگوریتم دیگر با میزان حافظه 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.

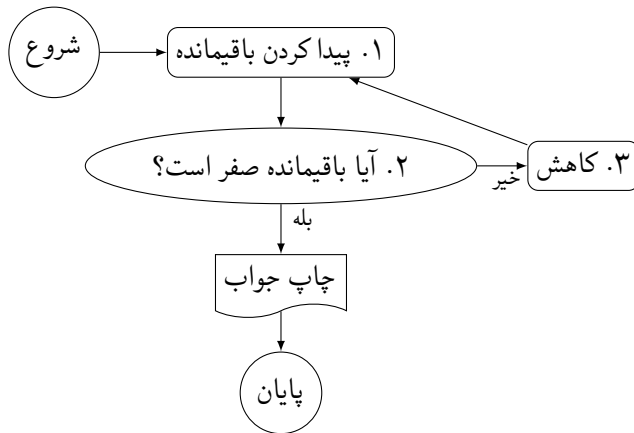
- یک الگوریتم برای یک مسئله محاسباتی درست است اگر به ازای هر نمونه از مسئله که با تعدادی ورودی معین شده است، (۱) توقف کند، بدین معنی که در زمان محدود به اتمام برسد و (۲) خروجی تعیین شده توسط شرح مسئله را تولید کند. می‌گوییم یک الگوریتم درست یک مسئله محاسباتی را حل می‌کند.
- یک الگوریتم نادرست ممکن است 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 برنامه 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<sup>1</sup> 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 متوازی‌الاضلاع یا اشکالی شبیه به مستطیل استفاده می‌شود. هر گام به گام بعدی توسط یک علامت فلش متصل می‌شود. شروع و پایان را معمولاً با دایره نشان می‌دهند.

---

<sup>1</sup> flowchart

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



- الگوریتم‌ها در زمینه‌های زیاد و متنوعی کاربرد دارند.
- به عنوان مثال در پروژه ژنوم‌های انسانی هدف پیدا کردن الگوهای ژن‌ها در دی‌ان‌ای<sup>1</sup> 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 مسئله محاسباتی راه حل‌های زیادی دارد که بنابر اهمیت معیارهای مورد نیاز استفاده کننده 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.

---

<sup>1</sup> DNA

- دسته‌ای از مسئله‌های محاسباتی وجود دارند که گرچه برای محاسبه 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 بی‌استفاده باشد. بعدها این مفهوم معقول را به طور رسمی و دقیق تعریف خواهیم کرد.
- این دسته از مسئله‌ها ان‌پی کامل<sup>1</sup> 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 در زمان معقول<sup>2</sup> پیدا کرد. بنابراین هیچ‌کس نمی‌داند آیا برای مسائل ان‌پی کامل الگوریتم کارآمد<sup>3</sup> وجود دارد یا خیر.

---

<sup>1</sup> NP-Complete problems

<sup>2</sup> reasonable

<sup>3</sup> efficient

- یک ویژگی دیگر مسائل ان پی کامل این است که اگر برای یکی از آنها الگوریتم کارامد پیدا شود، برای همه 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 ان پی کامل است.
- گرچه برای مسئله‌های ان پی کامل الگوریتم دقیق کارامد پیدا نشده است، ولی الگوریتم‌های تقریبی<sup>1</sup> زیادی وجود دارند که خروجی قابل قبولی نزدیک به خروجی مورد انتظار در زمان معقول تولید می‌کنند.

---

<sup>1</sup> Approximation algorithms



- 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 از الگوریتم‌ها به نام الگوریتم‌های موازی<sup>1</sup> 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<sup>2</sup> نامیده می‌شوند.

---

<sup>1</sup> Parallel algorithms

<sup>2</sup> Online algorithms

- 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  $A = [a_1, a_2, \dots, a_n]$  را داشته باشیم، می‌خواهیم عناصر آرایه یعنی  $a_i$  ها را جابجا کنیم و آرایه خروجی  $[a'_1, a'_2, \dots, a'_n]$  را به دست آوریم، به طوری که به ازای هر  $1 \leq i < n$  داشته باشیم  $a'_i \leq a'_{(i+1)}$ .

- یکی از الگوریتم‌های ارائه شده برای این مسئله الگوریتم مرتب سازی درجی<sup>1</sup> است.
- به طور خلاصه این الگوریتم به صورت زیر عمل می‌کند. فرض کنید یک آرایه با  $n$  عنصر از درایه ۱ تا درایه  $k$  مرتب شده باشد. حال برای مرتب سازی آرایه از درایه ۱ تا درایه  $k+1$  باید عنصر  $k+1$  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 درایه ۱ تا  $k+1$  مرتب کرده‌ایم. این کار را تا جایی ادامه می‌دهیم که کل آرایه مرتب شود.

---

<sup>1</sup> insertion sort

- به طور خلاصه این الگوریتم را می توانیم به صورت زیر بنویسیم.

---

## Algorithm Insertion Sort

---

```
function INSERTION-SORT(A, n)
    ▷ A is an array of n elements
1: for i = 2 to n do
2:     key = A[i]
3:     j = i - 1
4:     while j > 0 and A[j] > key do
5:         A[j+1] = A[j]
6:         j = j-1
7:     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<sup>1</sup> 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.

---

<sup>1</sup> loop invariant

## تابع پیچیدگی زمانی

- تابع پیچیدگی زمانی برای یک الگوریتم، زمان (و یا تعداد گام‌هایی) را مشخص می‌کند که الگوریتم برای پیدا کردن جواب مسئله نیاز دارد، به طوری که این زمان تابع اندازه ورودی مسئله است.
- بنابراین اگر ورودی یک مسئله  $n$  باشد و تعداد گام‌های لازم برای محاسبه جواب مسئله توسط یک الگوریتم  $f(n)$  باشد، می‌گوییم زمان محاسبه <sup>1</sup> یا زمان اجرا <sup>2</sup> یا پیچیدگی زمانی <sup>3</sup> الگوریتم  $f(n)$  است.
- هدف طراحی الگوریتم‌ها، پیدا کردن الگوریتم‌هایی برای مسائل است که زمان اجرای آنها تا حد امکان کوچک باشد. هر تابع با یک نرخ معین رشد می‌کند. می‌توانیم نرخ رشد توابع <sup>4</sup> مختلف را با یکدیگر مقایسه کنیم.

---

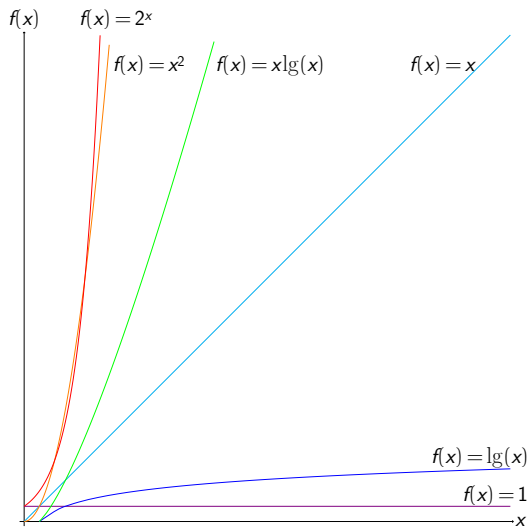
<sup>1</sup> computation time

<sup>2</sup> running time

<sup>3</sup> time complexity

<sup>4</sup> growth rate of functions

## مقایسه رشد توابع



## مقایسه رشد توابع پیچیدگی

- اگر هر گام در یک الگوریتم فقط یک میکروثانیه زمان ببرد، می‌توانیم زمان تقریبی محاسبه به ازای توابع رشد متفاوت را به صورت زیر با یکدیگر مقایسه کنیم.

اندازه $n$	۲۰	۴۰	۶۰
تابع پیچیدگی $f(n)$			
$n$	۰/۰۰۰۰۰۲ ثانیه	۰/۰۰۰۰۰۴ ثانیه	۰/۰۰۰۰۰۶ ثانیه
$n^2$	۰/۰۰۰۰۰۴ ثانیه	۰/۰۰۰۰۱۶ ثانیه	۰/۰۰۰۰۳۶ ثانیه
$n^3$	۰/۰۰۰۰۰۸ ثانیه	۰/۰۰۰۰۶۴ ثانیه	۰/۰۰۰۰۲۱۶ ثانیه
$n^5$	۳/۲ ثانیه	۱/۷ دقیقه	۱۳ دقیقه
$2^n$	۱ ثانیه	۱۲/۷ روز	۳۶۶ قرن
$3^n$	۵۸ دقیقه	۳۸۵۵ قرن	$۱۰^{۱۳} \times ۱/۳$ قرن



## مسئله کوتاهترین مسیر

مسئله کوتاهترین مسیر بین دو شهر  $c_x$  و  $c_y$  تشکیل شده است از شرحی از پارامترهای مسئله یعنی:

- مجموعه ای از شهرها،  $C = \{c_1, c_2, \dots, c_m\}$

- مجموعه ای از جاده‌ها، به طوری که هر جاده دو شهر را به هم متصل می‌کند،  $R \subseteq C \times C$

- تابعی که به ازای هر دو شهر به هم متصل شده توسط یک جاده، طول جاده را مشخص می‌کند،

$$\text{len} : R \rightarrow \mathbb{N}$$

و همچنین شرحی از جواب مسئله یعنی:

- مسیر  $P = \langle c_{f(1)}, c_{f(2)}, \dots, c_{f(n)} \rangle$  با  $1 \leq f(i) \leq m$  ،  $f(1) = x$  ،  $f(n) = y$  ،  $1 < n \leq m$  وجود داشته باشد، به طوری که مقدار پارامتر  $L$  کمینه باشد.

$$L = \sum_{i=1}^{n-1} \text{len}(c_{f(i)}, c_{f(i+1)})$$

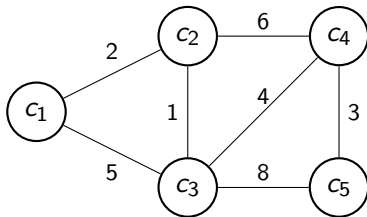
## نمونه مسئله کوتاهترین مسیر

کوتاهترین مسیر بین دو شهر  $c_1$  و  $c_5$  را پیدا کنید به طوری که

$$C = \{c_1, c_2, c_3, c_4, c_5\}$$

$$R = \{(c_1, c_2), (c_1, c_3), (c_2, c_3), (c_2, c_4), (c_3, c_4), (c_3, c_5), (c_4, c_5)\}$$

$$\text{len} = \{((c_1, c_2), 2), ((c_1, c_3), 5), ((c_2, c_3), 1), ((c_2, c_4), 6), \\ ((c_3, c_4), 4), ((c_3, c_5), 8), ((c_4, c_5), 3)\}$$



# یک الگوریتم ساده برای مسئله کوتاهترین مسیر

۱. همه مسیرها از  $c_x$  به  $c_y$  را پیدا می‌کنیم.
۲. طول همه آن مسیرها را محاسبه می‌کنیم.
۳. کوتاهترین مسیر را به دست می‌آوریم.

## پیچیدگی زمانی الگوریتم ساده‌ کوتاهترین مسیر

- در بدترین حالت همه شهرها با یک مسیر به هم متصل شده‌اند.
- در این صورت تعداد همه مسیرها از شهر  $c_x$  به شهر  $c_y$  شامل  $m$  شهر برابر است با  $(m - 2)!$ .
- بنابراین تنها برای شمردن همه مسیرها شامل  $m$  شهر و مقایسه طول آنها به  $(m - 2)!$  گام زمانی نیاز داریم.
- رشد تابع  $n!$  از رشد تابع  $2^n$  نیز سریع تر است.
- اگر بررسی هر مسیر فقط یک میکروثانیه زمان ببرد، برای شمارش و بررسی همه مسیرها در مجموعه‌ای با تنها  $60$  شهر به  $10^{66} \times 2/6$  قرن زمان نیاز داریم. ( $60! = 8/3 \times 10^{81}$ )

# الگوریتم کوتاهترین مسیر دایکسترا

- الگوریتم کوتاهترین مسیر دایکسترا توسط ادسخر دایکسترا<sup>1</sup> در سال ۱۹۵۶ ابداع شد.
- این الگوریتم کوتاهترین مسیر از  $c_x$  به  $c_y$  را محاسبه می‌کند.
- پیچیدگی زمانی این الگوریتم از مرتبه  $n^2$  است.
- با استفاده از الگوریتم دایکسترا برای مجموعه‌ای از ۶۰ شهر به تنها ۰/۰۰۳۶ ثانیه زمان نیاز داریم.

---

<sup>1</sup> Edsger Dijkstra