به نام خدا

طراحی الگوریتم‌ها

آرش شفیعی

کتاب‌های مرجع

- مقدمه‌ای بر الگوریتم‌ها از کرمن، لایسرسون، ریوست، و استاین [1]
- اصول الگوریتم‌ها از ریچارد نئاپولیتان [2]
- هنر برنامه نویسی از دونالد کنوث [3]

[1] Introduction to Algorithms، by Cormen، Leiserson، Rivest، and Stein
[2] Foundations of Algorithms، by Richard Neapolitan
[3] The Art of Computer Programming، by Donald Knuth

مقدمه

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.
- تا سال ۱۹۵۰ کلمهٔ الگوریتم بیشتر برای الگوریتم اقلیدس [2] برای پیدا کردن بزرگ‌ترین مقسوم‌علیه مشترک [3] دو عدد به‌کار می‌رفت که در کتاب اصول اقلیدس [4] توصیف شده است.

[1] algebra
[2] Euclid's algorithm
[3] greatest common divisor
[4] Euclid's Element

مقدمه

- 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.

مقدمه

- 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

- 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) شرحی از چندین پارامتر یا متغیر آزاد، و (۲) شرحی از ویژگی‌هایی که جواب مسئله دارد.
- یک پارامتر یا متغیر آزاد 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 یک 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

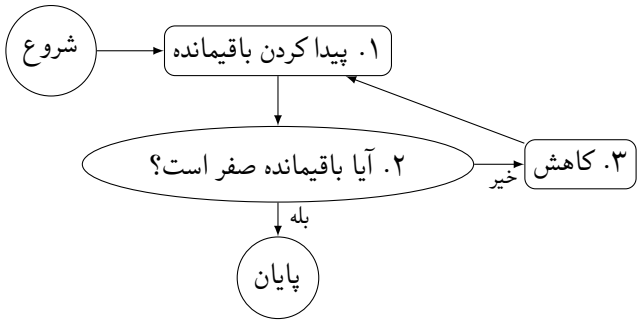
- یک الگوریتم برای یک مسئله محاسباتی درست است اگر به ازای هر نمونه از مسئله که با تعدادی ورودی معین شده است، (۱) توقف کند، بدین که در زمان محدود به اتمام برسد و (۲) خروجی تعیین شده توسط شرح مسئله را تولید کند. یک الگوریتم درست در واقع یک مسئلهٔ محاسباتی را حل می‌کند.
- یک الگوریتم نادرست ممکن است به 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.

مقدمه

- 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 با 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

- 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+۱ .

- 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

- 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 در یک سامانهٔ حمل و نقل هرچه به ازای یک سفر مسیر کوتاه‌تری طی شود، هزینه پایین می‌آید.

[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 مسئله انپی کامل 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.

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 زیادی وجود دارند که خروجی خوبی نزدیک به خروجی مورد انتظار در زمان معقول تولید می‌کنند.

- 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 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 [1] نامیده می‌شوند.

[1] online algorithm

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
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 [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

مثال : مسئله

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

- مجموعه ای از شهرها، $C = \{c_1, c_2, ..., c_m\}$
- 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 مشخص می‌کند، $len : R \to \mathbb{N}$

و همچنین شرحی از جواب مسئله یعنی:

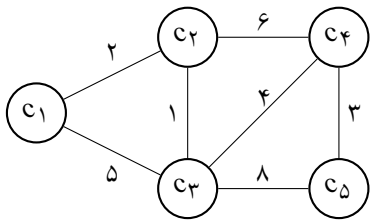
- مسیر $P = \langle c_{f(1)}, c_{f(2)}, ..., c_{f(n)} \rangle$ با $f(1) = x$ ، $f(n) = y$ ، $1 < n \leq m$ وجود داشته باشد، به طوری که مقدار پارامتر L کمینه باشد.

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

مثال : نمونهٔ مسئله

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

$$
\begin{aligned}
C &= \{c_1, c_2, c_3, c_4, c_5\} \\
R &= \{(c_1, c_2), (c_1, c_3), (c_2, c_3), (c_2, c_4), (c_3, c_4), (c_3, c_5), (c_4, c_5)\} \\
len &= \{((c_1, c_2), 2), ((c_1, c_3), 5), ((c_2, c_3), 1), ((c_2, c_4), 6), \\
&\quad ((c_3, c_4), 4), ((c_3, c_5), 8), ((c_4, c_5), 3)\}
\end{aligned}
$$



مثال : یک الگوریتم ساده برای مسئله کوتاهترین مسیر

۱ همهٔ مسیرها از c_x به c_y را پیدا می‌کنیم.

۲ طول همهٔ آن مسیرها را محاسبه می‌کنیم.

۳ کوتاهترین مسیر را به دست می‌آوریم.

تابع پیچیدگی زمانی

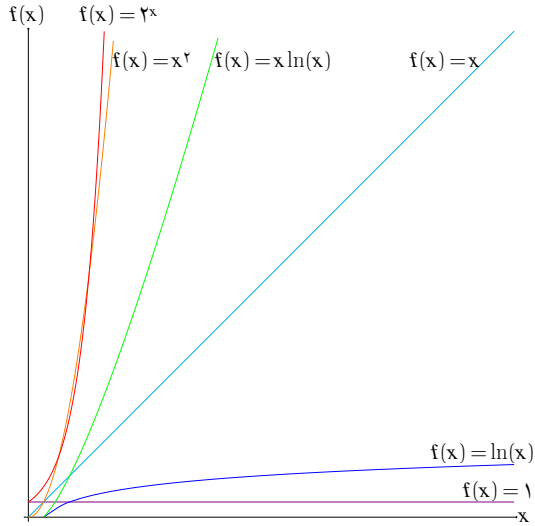
- تابع پیچیدگی زمانی برای یک الگویتم، زمان (و یا تعداد گام‌هایی) را مشخص می‌کند که الگوریتم برای پیدا کردن جواب مسئله نیاز دارد، به طوری که این زمان تابع اندازهٔ ورودی مسئله است.

- بنابراین اگر ورودی یک مسئله n باشد و زمان لازم برای محاسبهٔ جواب مسئله توسط یک الگوریتم $f(n)$ باشد، می‌گوییم زمان محاسبه [1] یا زمان اجرا [2] یا پیچیدگی زمانی [3] الگوریتم از مرتبهٔ $f(n)$ است.

- می‌توانیم نرخ رشد توابع [4] مختلف را با یکدیگر مقایسه کنیم.

[1] computation time
[2] running time
[3] time complexity
[4] 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.

اندازهٔ n	۲۰	۴۰	۶۰
تابع پیچیدگی f(n)			
n	۰/۰۰۰۰۰۲ ثانیه	۰/۰۰۰۰۰۴ ثانیه	۰/۰۰۰۰۰۶ ثانیه
n^2	۰/۰۰۰۴ ثانیه	۰/۰۰۱۶ ثانیه	۰/۰۰۳۶ ثانیه
n^3	۰/۰۰۸ ثانیه	۰/۰۶۴ ثانیه	۰/۲۱۶ 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	$۱/۳ \times ۱۰^{۱۳}$ قرن

مثال : پیچیدگی زمانی الگوریتم سادهٔ کوتاهترین مسیر

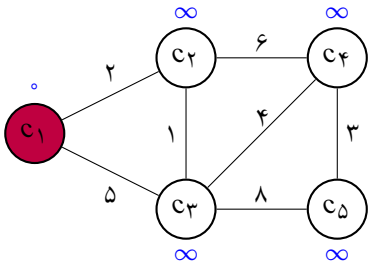
- در بدترین حالت همهٔ شهرها با یک مسیر به هم متصل شده‌اند.
- در این صورت تنها تعداد همهٔ مسیرها از شهر c_x به شهر c_y با طول n برابر است با $(n-1)!$.
- بنابراین تنها برای شمردن همهٔ مسیرهای با طول n و مقایسهٔ طول آنها به $(n-1)!$ گام زمانی نیاز داریم.
- رشد تابع $n!$ از رشد تابع 2^n نیز سریع تر است.
- اگر بررسی هر مسیر فقط یک میکروثانیه زمان ببرد، برای شمارش و بررسی همهٔ مسیرها در مجموعه‌ای با تنها ۶۰ شهر به $۲/۶ \times ۱۰^{۶۶}$ قرن زمان نیاز داریم. ($۶۰! = ۸/۳ \times ۱۰^{۸۱}$)

الگوریتم کوتاهترین مسیر دایکسترا

- الگوریتم کوتاهترین مسیر دایکسترا توسط ادسخر دایکسترا [1] در سال ۱۹۵۶ ابداع شد.
- این الگوریتم از حافظه‌ای جانبی استفاده می‌کند و کوتاهترین مسیر را برای تمام زیر مسئله‌های مسئله اصلی پیدا می‌کند، یعنی:
- با شروع از شهر c_x این الگوریتم کوتاهترین مسیرها را از شهر c_x به تمام همسایه‌های این شهر محاسبه می‌کند و به همین ترتیب کوتاهترین مسیر از همسایه‌های c_x به همهٔ همسایه‌های آنها، الی آخر.
- بدین ترتیب الگوریتم دایکسترا همهٔ کوتاهترین مسیرها را از شهر c_x به همهٔ شهرهای دیگر از جمله شهر c_y محاسبه می‌کند.

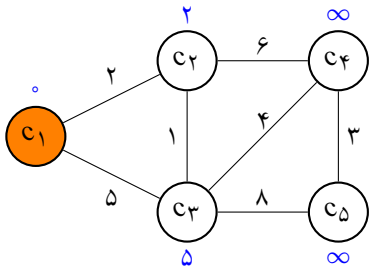
[1] Edsger Dijkstra

مثال: الگوریتم کوتاهترین مسیر دایکسترا



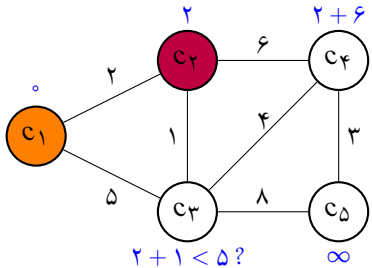
$$P = \langle c_1, c_2, c_3, c_4, c_5 \rangle$$

مثال: الگوریتم کوتاهترین مسیر دایکسترا



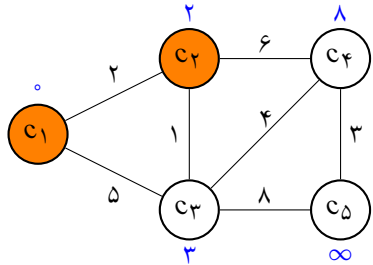
$$P = \langle c_1, c_2, c_3, c_4, c_5 \rangle$$

مثال: الگوریتم کوتاهترین مسیر دایکسترا



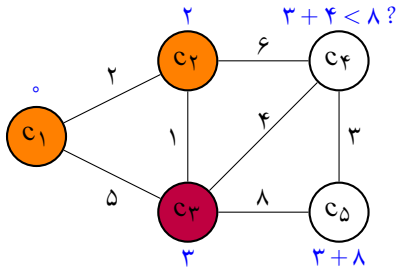
$$P = \langle c_1, c_2, c_3, c_4, c_5 \rangle$$

مثال: الگوریتم کوتاهترین مسیر دایکسترا



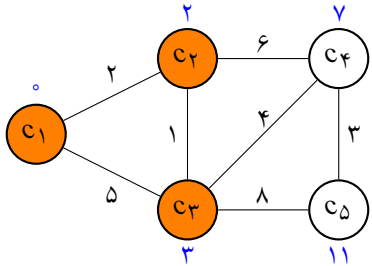
$$P = \langle c_1, c_2, c_3, c_4, c_5 \rangle$$

مثال: الگوریتم کوتاهترین مسیر دایکسترا



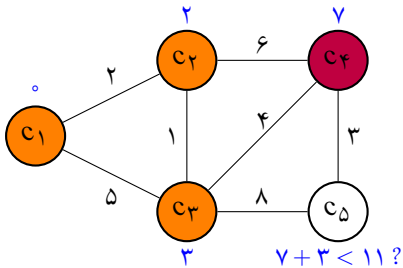
$$P = \langle c_1, c_2, c_3, c_4, c_5 \rangle$$

مثال: الگوریتم کوتاهترین مسیر دایکسترا



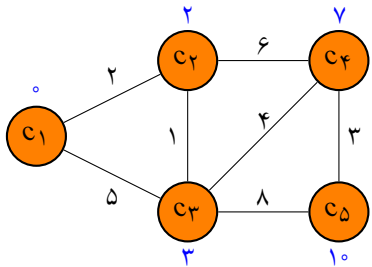
$$P = \langle c_1, c_2, c_3, c_4, c_5 \rangle$$

مثال: الگوریتم کوتاهترین مسیر دایکسترا



$$P = \langle c_1, c_2, c_3, c_4, c_5 \rangle$$

مثال: الگوریتم کوتاهترین مسیر دایکسترا



$$P = \langle c_1, c_2, c_3, c_4, c_5 \rangle$$

پیچیدگی زمانی الگوریتم کوتاهترین مسیر دایکسترا

- هر شهر در یک گام باید بررسی شود و در بدترین حالت (حالتی که همهٔ شهرها به هم متصل باشند) به ازای هر شهر، همهٔ شهرهای دیگر بررسی می‌شوند.
- بنابراین در صورتی که n شهر وجود داشته باشد، پیچیدگی الگوریتم n^2 است.
- با استفاده از الگوریتم دایکسترا برای مجموعه‌ای از ۶۰ شهر به تنها ۰/۰۰۳۶ ثانیه زمان نیاز داریم.