

به نام خدا

ساختمان داده

آرش شفیعی



الگوریتم‌های گراف

الگوریتم‌های گراف

- گراف یک ساختار گستته است که با استفاده از آن می‌توان تعدادی مفهوم که با یکدیگر در ارتباط هستند را مدلسازی کرد.
- برای مدلسازی مفاهیم از رئوس گراف و برای مدلسازی ارتباط بین مفاهیم از یال‌های گراف استفاده می‌کنیم.
- گراف‌ها در علوم کامپیوتر و دیگر شاخه‌های علوم بسیار پر استفاده‌اند. برای مثال برای مدلسازی یک شبکه کامپیوتری که از تعدادی کامپیوتر و تعدادی مسیر ارتباطی تشکیل شده است، می‌توانیم از یک گراف استفاده کنیم و از الگوریتم‌های گراف برای یافتن مسیر بهینه برای انتقال یک بسته در شبکه بهره بگیریم. همچنین در شبکه‌های اجتماعی می‌توان افراد و سازمان‌ها را به عنوان رئوس یک گراف در نظر گرفت و ارتباط بین افراد و سازمان‌ها را با استفاده از یال‌های گراف مدلسازی کرد. از الگوریتم‌های گراف جهت تحلیل این شبکه اجتماعی برای یافتن اطلاعات در گراف می‌توان استفاده کرد.

الگوریتم‌های گراف

- در زبان‌شناسی می‌توان از گراف‌ها جهت نمایش ارتباط بین کلمات در یک زبان استفاده کرد و از گراف به دست آمده در پردازش زبان طبیعی، و ترجمه‌های ماشینی استفاده کرد.
- در علوم فیزیک و شیمی می‌توان از گراف‌ها جهت مدلسازی مولکول‌ها استفاده کرد و ساختار مولکول‌ها و روابط آنها و نیروهای بین اتم‌ها و مولکول‌ها را شبیه سازی کرد.
- در علوم اجتماعی می‌توان از گراف‌ها جهت مدلسازی ارتباط انسان‌ها و نحوه منتشر شدن اطلاعات و افکار بین انسان‌ها و جوامع استفاده کرد.
- در علوم زیست‌شناسی می‌توان از گراف‌ها جهت بررسی روابط بین گونه‌های جانوری و گیاهی و همچنین بررسی ساختار ژن‌ها استفاده کرد.

الگوریتم‌های گراف

- در این قسمت با روش‌های نمایش گراف و جستجوی گراف‌ها آشنا می‌شویم. با استفاده از روش‌های جستجوی گراف‌ها می‌توانیم ساختار یک گراف و ویژگی‌های آن را بشناسیم.

الگوریتم‌های گراف

- یک گراف را می‌توان با استفاده از دو مجموعه رأس‌ها^۱ (V) و یال‌ها^۲ (E) نمایش داد. بدین ترتیب دوتایی $G = (V, E)$ گرافی را نمایش می‌دهد که در آن V مجموعه‌ای است از رئوس و E مجموعه‌ای است از یال‌ها. یک یال دو رأس را به یکدیگر متصل می‌کند.

^۱ vertex set

^۲ edge set

الگوریتم‌های گراف

- علاوه بر روش استاندارد نمایش یک گراف توسط مجموعه‌ها، می‌توانیم یک گراف را توسط مجموعه‌ای از لیست‌های مجاورت^۱ یا یک ماتریس مجاورت^۲ نشان دهیم.
- توسط لیست مجاورت می‌توان گراف‌های خلوت^۳ را که در آنها $|E|$ بسیار کوچک‌تر از $|V|^2$ است نمایش داد. وقتی گراف متراکم^۴ است، می‌توان از نمایش ماتریس مجاورت استفاده کرد.

^۱ adjacency lists

^۲ adjacency matrix

^۳ sparse graph

^۴ dense

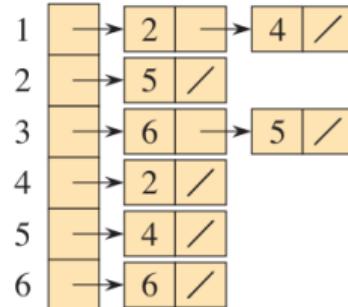
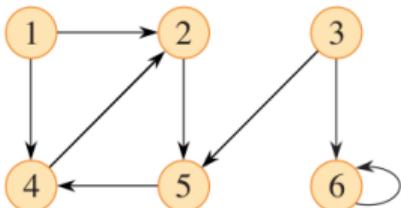
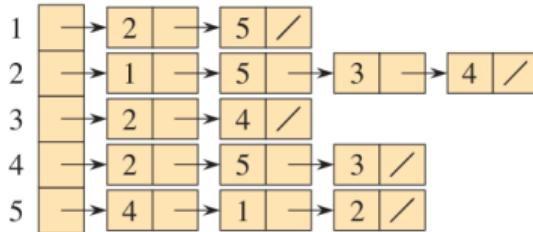
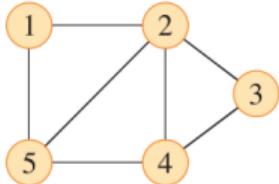
الگوریتم‌های گراف

- در نمایش لیست مجاورت^۱ برای گراف $G = (V, E)$ از آرایه Adj شامل $|V|$ عنصر استفاده می‌کنیم. به ازای هر $u \in V$ ، لیست مجاورت $\text{Adj}[u]$ شامل رئوس v است به طوری که $(u, v) \in E$. پس $[u, v] \in \text{Adj}[u]$ شامل همه رئوسی است که در گراف G مجاور u هستند. در پیاده‌سازی این روش، عناصر این آرایه می‌توانند اشاره‌گر به رئوس مجاور باشند.

^۱ adjacency-list representation

الگوریتم‌های گراف

- در شکل زیر دو گراف توسط لیست مجاورت نشان داده شده‌اند.



الگوریتم‌های گراف

- اگر G یک گراف جهت دار باشد، مجموع اندازه همه لیست‌های مجاورت برابر است با $|E|$ ، زیرا هر یک از یال‌های (u, v) توسط درایه $\text{Adj}[u]$ نشان داده می‌شود.
- اگر G یک گراف بدون جهت باشد، آنگاه مجموع اندازه همه لیست‌های مجاورت برابر است با $2|E|$ ، زیرا هر یک از یال‌های (u, v) هم در $\text{Adj}[u]$ و هم در $\text{Adj}[v]$ نمایش داده می‌شود.
- فضای حافظه‌ای که لیست مجاورت برای نگهداری گراف نیاز دارد برابر است با $\Theta(V + E)$.

الگوریتم‌های گراف

- توسط لیست مجاورت می‌توانیم گراف‌های وزن دار^۱ را نیز نمایش دهیم. در یک گراف وزن دار، هریک از یال‌ها دارای یک وزن است که توسط تابع وزن $w: E \rightarrow \mathbb{R}^2$ تولید می‌شود.
- برای مثال فرض کنید $G = (V, E)$ یک گراف وزن دار با تابع وزن w باشد. آنگاه می‌توانیم وزن $w(u, v)$ از یال $E \in (u, v)$ را در کنار رأس v در لیست مجاورت u ذخیره کنیم و نمایش دهیم.
- یکی از معایب لیست مجاورت این است که برای پیدا کردن یال (u, v) سریع‌ترین روش ممکن جستجوی v در لیست مجاورت $\text{Adj}[u]$ است.
- ماتریس مجاورت برای پیدا کردن یک یال می‌تواند از لیست مجاورت سریع‌تر عمل کند.

^۱ weighted graphs

^۲ weight function

الگوریتم‌های گراف

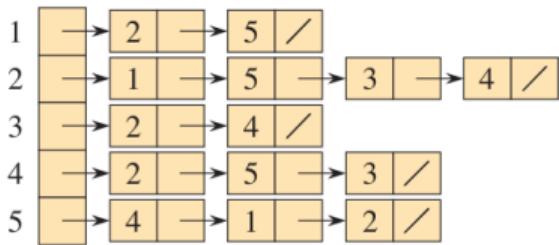
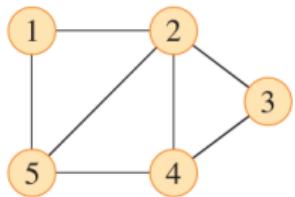
- در نمایش ماتریس مجاورت^۱ برای گراف $G = (V, E)$ فرض می‌کنیم هر رأس یک شماره از ۱ تا $|V|$ داشته باشد. سپس گراف G را با استفاده از ماتریس $A = (a_{ij})$ با اندازه $|V| \times |V|$ نشان می‌دهیم به طوری که

$$a_{ij} = \begin{cases} 1 & \text{اگر } (i, j) \in E \\ 0 & \text{در غیر اینصورت} \end{cases}$$

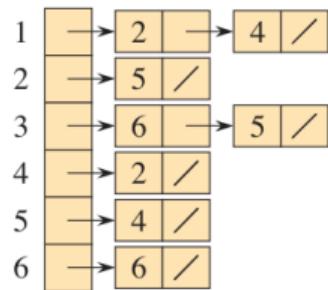
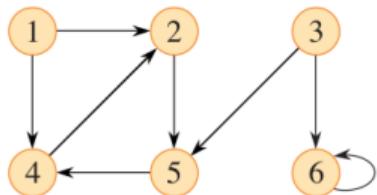
^۱ adjacency matrix representation

الگوریتم‌های گراف

- در شکل زیر دو ماتریس مجاورت نشان داده شده‌اند.



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	0	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

الگوریتم‌های گراف

- ماتریس مجاورت به حافظه $(V^2)\Theta$ برای نگهداری گراف نیاز دارد. برای یافتن یال (u, v) در گراف می‌توانیم درایه $A[u, v]$ را بررسی کنیم و بنابراین یافتن یک یال در گراف در زمان $(1)\Theta$ انجام می‌شود.
- برای یک گراف بدون جهت، گراف نسبت به قطر اصلی‌اش متقارن است، زیرا $A[u, v] = A[v, u]$ برابراست با A بنابراین ماتریس مجاورت برابر با ترانهاده^۱ آن است و داریم $A = A^T$.
- اما در یک گراف جهت‌دار می‌توانیم یالی از u به v داشته باشیم بدون اینکه یال از v به u وجود داشته باشد.

^۱ transpose

الگوریتم‌های گراف

- با استفاده از ماتریس مجاورت می‌توانیم یک گراف وزن دار را نیز نمایش دهیم.
- برای مثال، اگر $G = (V, E)$ یک گراف وزن دار با تابع وزن w باشد، می‌توان وزن $w(u, v)$ برای یال $\in E$ را به عنوان درایه ماتریس مجاورت در سطر u و ستون v ذخیره کرد.
- استفاده از ماتریس مجاورت در عمل راحت‌تر است اما به ازای گراف‌های بزرگ خلوت ممکن است فضای حافظه مورد نیاز آنها بسیار زیاد شود. برای ذخیره‌سازی گراف‌های خلوت جهت‌دار، لیست مجاورت نسبت به ماتریس مجاورت فضای بسیار کمتری اشغال می‌کند.

جستجوی سطح اول

- جستجوی سطح اول^۱ یکی از ساده‌ترین الگوریتم‌های جستجوی گراف است که در بسیاری از الگوریتم‌های گراف استفاده می‌شود. برای مثال الگوریتم دایکسترا برای یافتن کوتاه‌ترین مسیر بین دو رأس از جستجوی سطح اول استفاده می‌کند.
- به ازای گراف دلخواه $(V, E) = G$ و یک رأس مبدأ^۲ به نام s ، الگوریتم جستجوی سطح اول همه یال‌های گراف G را با شروع از رأس s بررسی می‌کند.
- با شروع از رأس s ، الگوریتم سطح اول ابتدا رئوسی را بررسی می‌کند که به s نزدیک‌ترند، بدین معنی که برای رسیدن به آن رئوس از s باید از تعداد یال‌های کمتری عبور کرد.

^۱ breadth-first search

^۲ source vertic

جستجوی سطح اول

- یال‌هایی که در جستجوی سطح اول به ترتیب بررسی می‌شوند، یک درخت سطح اول می‌سازد که ریشه آن s است و به ازای هر یک از رئوس v ، یک مسیر ساده از s به v کوتاهترین مسیر از s به v در گراف را نشان می‌دهد. در اینجا کوتاهترین مسیر درواقع مسیری است که دارای کمترین تعداد یال باشد.
- جستجوی سطح اول، بدین دلیل سطح اول نامیده می‌شود که به ازای هر رأس v ابتدا رئوس مجاور آن بررسی می‌شوند، قبل از اینکه رئوس مجاور مجاور آن بررسی شوند. بنابراین اگر نزدیکترین رئوس به یک رأس را در سطح در نظر بگیریم و دورترین رئوس را در عمق، جستجوی سطح اول، قبل از بررسی رئوس در عمق، همه رئوس در سطح را بررسی می‌کند. بنابراین برخلاف جستجوی عمق اول که به ازای هر رأس v یک رأس مجاور v ممکن است قبل از یک رأس مجاور v پیمایش شود، در جستجوی سطح اول همه رئوس مجاور v قبل از پیمایش رئوس مجاور مجاور v پیمایش می‌شوند.

جستجوی سطح اول

- در جستجوی سطح اول با شروع از رأس s ابتدا رئوس مجاور^۱ یا همسایه‌هایی^۲ بررسی می‌شوند که فاصله آنها از s برابر ۱ است، سپس همسایه‌ها با فاصله ۲ بررسی می‌شوند، پس از آن همسایه‌ها با فاصله ۳ و به همین ترتیب الی آخر، تا وقتی که همه رئوس بررسی شده باشند.
- در جستجوی سطح اول از یک صفت استفاده می‌شود که در آن ابتدا همسایه‌ها با فاصله ۱، سپس همسایه‌ها با فاصله ۲ و به همین ترتیب الی آخر در صفت قرار می‌گیرند. بنابراین با خارج کردن همسایه‌ها از صفت به ترتیب فاصله، گراف به صورت سطح اول بررسی می‌شود.

^۱ adjacent

^۲ neighbour

جستجوی سطح اول

- در الگوریتم جستجوی سطح اول می‌توانیم برای هر رأس ۳ رنگ در نظر بگیریم : سفید، خاکستری و سیاه. همهٔ رئوس در ابتدا به رنگ سفید هستند و رئوسی که هیچ مسیری از S به آنها وجود ندارد تا انتها به رنگ سفید باقی می‌مانند. وقتی یک رأس برای اولین بار با شروع از S پیمایش می‌شود، آن رأس به رنگ خاکستری تبدیل می‌شود. رنگ خاکستری بدین معنی است که آن رأس در مرز جستجو قرار گرفته است. مرز جستجو در واقع مرز میان رئوس پیمایش نشده و رئوس پیمایش شده است. صفحی که در جستجوی سطح اول استفاده می‌شود، شامل همهٔ رئوس خاکستری است.
- رئوس خاکستری به ترتیب از صفات خارج می‌شوند و به رنگ سیاه تبدیل می‌شوند و رئوس سفید همسایه آنها که تاکنون پیمایش نشده‌اند به رنگ خاکستری تبدیل می‌شوند و وارد صفات می‌شوند.

جستجوی سطح اول

- یک الگوریتم جستجوی سطح اول، یک درخت سطح اول می‌سازد که ریشه آن رأس s است. هرگاه در فرایند جستجو، یک رأس سفید v که در لیست همسایه‌های رأس خاکستری u قرار دارد پیدا می‌شود، رأس v و یال (u, v) به درخت اضافه می‌شوند. می‌گوییم رأس u ، سلف^۱ یا پدر رأس v و رأس v خلف^۲ یا فرزند رأس u در درخت سطح اول است. از آنجایی که هر رأس قابل دسترس از طریق s تنها یک بار بررسی می‌شود، هر رأس تنها یک پدر دارد.
- تنها رأس ریشه، یعنی رأس s دارای پدر نیست.
- اگر رأس u بر روی یک مسیر ساده درخت از ریشه s به رأس v قرار بگیرد، آنگاه رأس u جد^۳ رأس v است و رأس v نواده^۴ رأس u است.

¹ predecessor

² successor

³ ancestor

⁴ descendant

جستجوی سطح اول

- در الگوریتم جستجوی سطح اول که بررسی خواهیم کرد، $v.\text{color}$ رنگ رأس v است که می‌تواند سفید، خاکستری یا سیاه باشد، $v.d$ فاصله رأس v از رأس s است و $v.\text{pred}$ پدر رأس v است.
- رأس $v.\text{pred}$ ¹ پدر v رأس v است، و رأس v فرزند ² رأس $v.\text{pred}$ است.

¹ predecessor

² successor

جستجوی سطح اول

الگوریتم زیر جستجوی سطح اول را نشان می‌دهد.

Algorithm Breadth-First Search

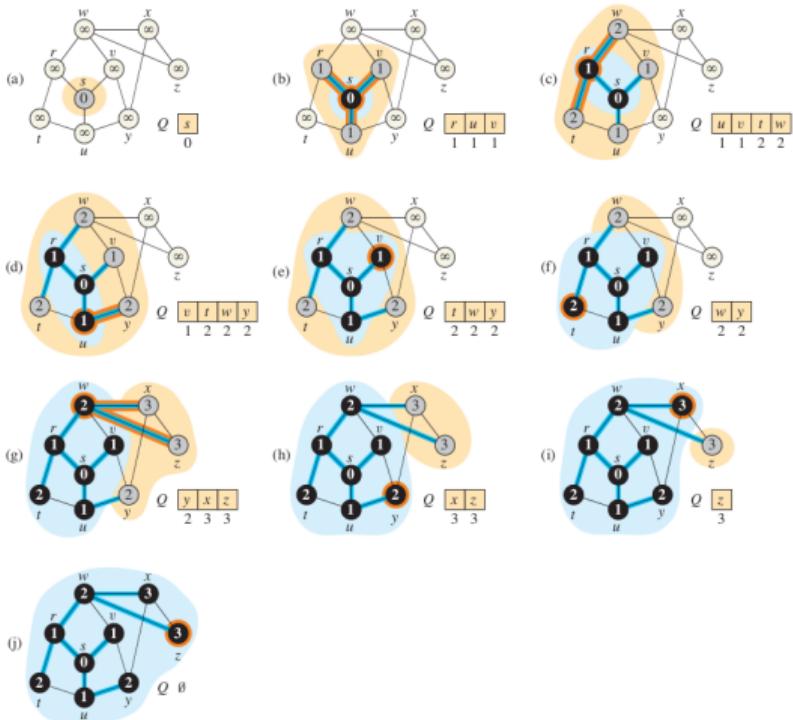
```
function BFS(G,s)
1: for each vertex u ∈ G.V - {s} do
2:   u.color = White
3:   u.d = ∞
4:   u.pred = Nil
5: s.color = Gray
6: s.d = 0
7: s.pred = Nil
8: Q = ∅
9: Enqueue(Q,s)
```

Algorithm Breadth-First Search

```
10: while !empty(Q) do
11:   u = Dequeue(Q)
12:   for each vertex v in G.Adj[u] do      ▷ search the neighbors of u
13:     if v.color == White then          ▷ is v being discovered now?
14:       v.color = Gray
15:       v.d = u.d + 1
16:       v.pred = u
17:       Enqueue(Q,v)                  ▷ v is now on the frontier
18:   u.color = Black      ▷ u is now behind the frontier
```

جستجوی سطح اول

- در شکل زیر یک گراف به صورت سطح اول پیمایش شده است.



جستجوی سطح اول

- تحلیل الگوریتم جستجوی سطح اول : برای تحلیل الگوریتم جستجوی سطح اول می‌توانیم از تحلیل تجمعی استفاده کنیم. در این الگوریتم هیچ‌گاه یک رأس از رنگ خاکستری یا سیاه به رنگ سفید در نمی‌آید. هر رأس حداکثر یک بار وارد صف می‌شود و بنابراین هر رأس حداکثر یک بار از صف خارج می‌شود. عملیات اضافه کردن و برداشتن از صف در زمان $O(1)$ انجام می‌شود. خارج کردن رئوس از صف در زمان $O(V)$ انجام می‌گیرد. همچنین بررسی لیست مجاورت هر رأس حداکثر یک بار انجام می‌شود و مجموع طول همه لیست‌های مجاورت برابر است با $\Theta(E)$ ، بنابراین زمان لازم برای اجرای الگوریتم برابر است با $O(V + E)$. نتیجه می‌گیریم جستجوی سطح اول در زمان خطی نسبت به اندازه لیست مجاورت اجرا می‌شود.
- می‌توان ثابت کرد الگوریتم جستجوی سطح اول کوتاهترین مسیر از s به هریک از رئوس را محاسبه می‌کند.

جستجوی عممق اول

- جستجوی عممق اول به جای اینکه جستجو را در سطح شروع کند و همه رئوس مجاور را در ابتدا پیمایش کند، به ازای هر رأس پیمایش شده، مجاور رأس را پیمایش می‌کند و به عبارت دیگر جستجو در عمق انجام می‌دهد.
- برای روشن تر شدن جستجوی سطحی و عمقی مثال زیر را در نظر بگیرید. می‌خواهیم مطلبی را درون چندین کتاب جستجو کنیم. در یک جستجوی سطحی ابتدا به سراغ صفحه اول همه کتاب‌ها می‌رویم تا این که در نهایت همه کتاب‌ها را بررسی کنیم. در یک جستجوی عمقی ابتدا کتاب اول را تا انتها مطالعه می‌کنیم و در صورتی که مطلب مورد نظر را پیدا نکردیم به سراغ کتاب دوم می‌رویم تا این که در نهایت همه کتاب‌ها را جستجو کنیم. در این مثال هیچ یک از جستجوها مزیتی بر دیگری ندارد چرا که مطلب مورد نظر ممکن است در صفحه آخر کتاب اول باشد که در این صورت جستجوی عمقی زودتر به جواب می‌رسد و یا ممکن است مطلب مورد نظر در صفحه اول کتاب آخر باشد که در این صورت جستجوی سطح اول زودتر به جواب می‌رسد.

جستجوی عمق اول

- جستجوی عمق اول یال‌های بررسی نشده رئوس تازه پیدا شده را زودتر از یال‌های بررسی نشده رئوس قبل‌پیدا شده بررسی می‌کند. وقتی فرایند جستجو به نقطه‌ای رسید که یال‌های یک رأس همگی بررسی شده بودند، الگوریتم پسگرد می‌کند تا به رئوسی برسد که یال‌های آنها هنوز بررسی نشده‌اند.
- در صورتی که یک رأس با شروع از رأس آغازین قابل دسترس نباشد، گراف همبند نیست و برای جستجوی کامل گراف، الگوریتم یکی از رئوس را به عنوان مبدأ جدید انتخاب کرده و جستجو را از رأس جدید آغاز می‌کند.

جستجوی عممق اول

- همانند جستجوی سطح اول، در جستجوی عممق اول توسط رنگ رأس‌ها وضعیت آنها مشخص می‌شود. هر رأس در ابتدا سفید است، هنگامی که برای اولین بار پیدا می‌شود به رنگ خاکستری تبدیل می‌شود و در پایان هنگامی که بررسی شد (بدین معنی که همه رئوس در لیست مجاورت آن پیدا شدند) به رنگ سیاه در می‌آید.

جستجوی عمق اول

- در جستجوی عمق اول هر رأس دارای دو برچسب زمان اول ۷.۵ زمانی نشان می‌دهد که رأس برای بار اول پیدا شده است و به رنگ خاکستری درآمده است و برچسب دوم ۷.f مشخص می‌کند که رأس ۷ به طور کامل بررسی شده است بدین معنی که همه رئوس لیست مجاورت آن پیدا شده‌اند و رأس ۷ به رنگ مشکی درآمده است.

^۱ time stamp

- الگوریتم زیر جستجوی عمق اول را نشان می دهد.

Algorithm Depth-First Search

```
function DFS(G)
1: for each vertex  $u \in G.V$  do
2:    $u.color = \text{White}$ 
3:    $u.pred = \text{Nil}$ 
4: time = 0
5: for each vertex  $u \in G.V$  do
6:   if  $u.color == \text{White}$  then
7:     DFS-Visit(G,u)
```

Algorithm DFS-Visit

```
function DFS-VISIT(G,u)
1: time = time + 1 ▷ white vertex u has just been discovered
2: u.s = time
3: u.color = Gray
4: for each vertex v in G.Adj[u] do ▷ explore each edge(u,v)
5:   if v.color == White then
6:     v.pred = u
7:     DFS-Visit(G,v)
8: time = time + 1
9: u.f = time
10: u.color = Black ▷ blacken u; it is finished
```

جستجوی عمق اول

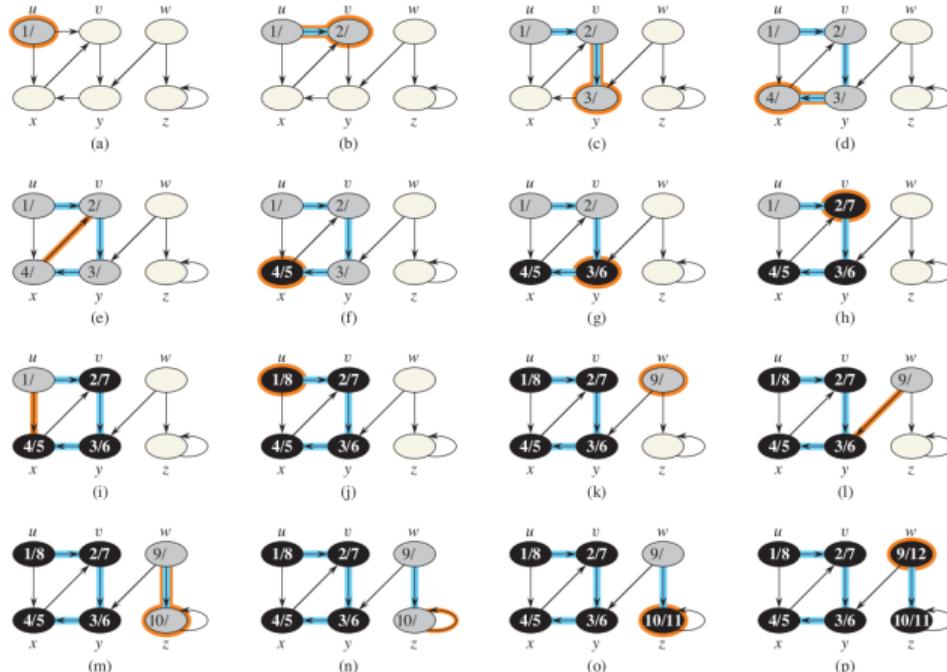
- وقتی الگوریتم جستجوی عمق اول به اتمام می‌رسد هر رأس دارای دو برچسب زمان است که یکی زمان پیدا شدن^۱ و دیگری زمان به پایان رسیدن^۲ را نشان می‌دهد.

^۱ discovery time

^۲ finish time

جستجوی عمق اول

- در مثال زیر، گراف توسط الگوریتم عمق اول بررسی شده است.



جستجوی عممق اول

- در اینجا نیز برای تحلیل الگوریتم از تحلیل تجمعی استفاده می‌کنیم.
- الگوریتم DFS-Visit برای هر رأس $v \in V$ تنها یک بار فراخوانی می‌شود، چرا که این الگوریتم برای رئوس سفید فراخوانی می‌شود و آنها را به رنگ خاکستری تبدیل می‌کند. الگوریتم DFS-Visit به ازای هر رأس v در یک حلقه تکرار $|Adj[v]|$ بار تکرار می‌شود. بنابراین برای همه رئوس، این حلقه $\sum_{v \in V} |Adj[v]| = \Theta(E)$.
- پس زمان اجرای الگوریتم جستجوی عممق اول $\Theta(V + E)$ است.

جستجوی عمقوال

- می‌توان اثبات کرد که در جستجوی عمقوال زمان پیدا شدن و زمان به اتمام رسیدن رئوس گراف یک ساختار پرانتر گذاری کامل دارند. اگر به ازای یافته شدن رأس \mathbf{l} یک پرانتر به صورت " \mathbf{l} " باز کنیم و به ازای به اتمام رسیدن بررسی رأس \mathbf{l} پرانتر را به صورت " \mathbf{l} " ببندیم، یک عبارت با پرانتر گذاری کامل به دست می‌آید بدین معنی که پرانترها تودرتو هستند.

جستجوی عمق اول

- از جستجوی عمق اول در مرتب‌سازی توپولوژیکی^۱ یا مرتب‌سازی موضعی یک گراف بدون دور^۲ استفاده می‌شود.
- یک مرتب‌سازی توپولوژیکی در یک گراف بدون دور $G = (V, E)$ رئوس گراف را به گونه‌ای مرتب می‌کند که اگر G شامل یال (u, v) باشد، آنگاه u قبل از v در آرایه مرتب شده قرار می‌گیرد.
- مرتب‌سازی توپولوژیکی تنها برای گراف‌های جهت‌دار بدون دور^۳ تعریف می‌شود.
- مرتب‌سازی توپولوژیکی به گونه‌ای است که اگر رئوس مرتب شده ببروی یک خط افقی قرار بگیرند، جهت همه یال‌های از چپ به راست است.

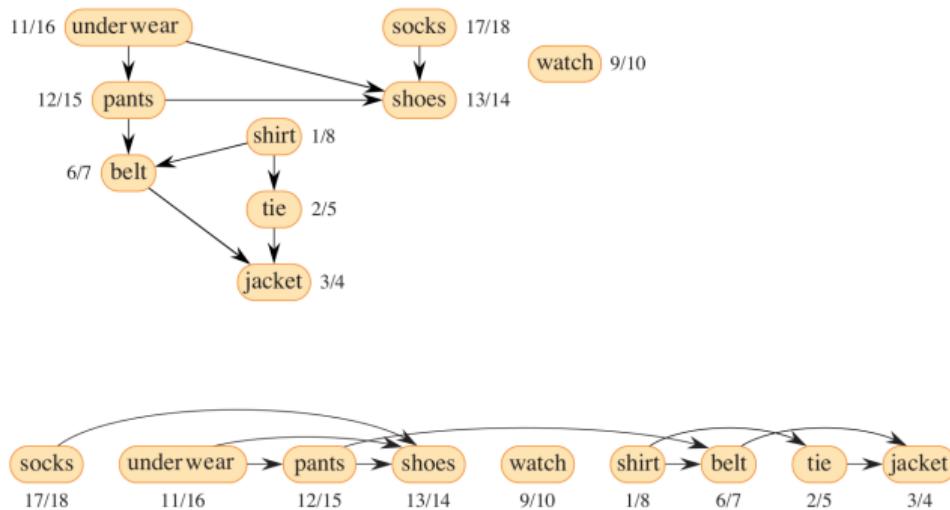
^۱ topological sort

^۲ acyclic graph

^۳ directed acyclic graph

جستجوی عمق اول

- از یک گراف جهت دار بدون دور می توان برای نمایش دادن رویدادها^۱ استفاده کرد.
- برای مثال در شکل زیر برای یک گراف شامل تعدادی رویداد مرتب سازی توپولوژیکی انجام شده است.



¹ event