8- فصل هشتم

9- گراف

مسائل گراف در علوم کامپیوتر دارای اهمیت به‌سزایی هستند. و الگوریتم‌های مربوط به آن‌ها نقشی بنیادین را ایفا می‌کنند. یک گراف($G=(V,E)$) از دو مجموعه‌ی یال‌ها[1](E) و رأس‌ها[2](V) تشکیل می‌شود که هر یال دو رأس را یه یکدیگر متصل می‌کند.

در این بخش، روش‌هایی برای نمایش دادن یک گراف و پیمایش آن معرفی می‌شوند.

پیمایش یک گراف به معنی دنبال کردن نظام‌مند یال‌هاست به‌طوری‌که تمام رأس‌ها ملاقات شوند. بسیاری از الگوریتم‌ها، با پیمایش گرافی که به عنوان ورودی می گیرند ، شروع می شوند تا اطلاعاتی را درباره‌ی ساختار آن گراف ، به دست آورند.از این‌رو روش‌های پیمایش یک گراف، نقشی کلیدی در الگوریتم های مربوط به گراف دارند.

دو روش برای نمایش دادن قابل رایانش[3] یک گراف معرفی می شوند که عبارتند از: ماتریس مجاورت و لیست مجاورت. سپس دو روش برای پیمایش گراف، پیمایش سطحی[4] و پیمایش عمقی[5] گراف معرفی می شوند.

---

[1] Edges
[2] Vertices (nodes)
[3] Computable Represntation
[4] Breadth-First Search (BFS)
[5] Depth-First Search (DFS)

9-1- نمایش گراف

دو روش رایج برای نمایش یک گراف وجود دارد؛ ماتریس مجاورت و لیست مجاورت.

روش اول روش ماتریس مجاورت است اگر تعداد رأس‌ها در یک گراف n باشد، ماتریس مجاورت این گراف یک ماتریس n*n خواهد بود به طوری که در ماتریس A عنصر a_{ij}، برابر با یک است اگر و تنها اگر یال (v_i, v_j) در گراف موجود باشد، در غیر این‌صورت عدد صفر قرار می‌گیرد. ماتریس مجاورت بدون توجه به تعداد یال موجود در گراف به فضایی به اندازه‌ی n^2 نیاز دارد، اما توجه کنید که اگر ماتریس متقارن باشد (گراف غیرجهت‌دار) ذخیره نیمی از ماتریس کافی است. در ماتریس‌های پراکنده که تعداد یال‌ها در ماتریس است بیشتر عناصر ماتریس عدد ۰ خواهند بود. در این حالت اتلاف زیادی در حافظه خواهیم داشت که پیچیدگی آن در بدترین حالت $O(V^2)$ است.

به راحتی می توان گراف های وزن‌دار را نیز با ماتریس مجاورت نمایش داد، به این صورت که در ماتریس زمانی‌که بین دو رأس i و j یال وجود دارد وزن یال در ماتریس نوشته می‌شود و در غیر این‌صورت دو حالت داریم:

1-اگر وزن از جنس هزینه باشد (مانند طول یک جاده)، عدد بی نهایت نوشته میشود.

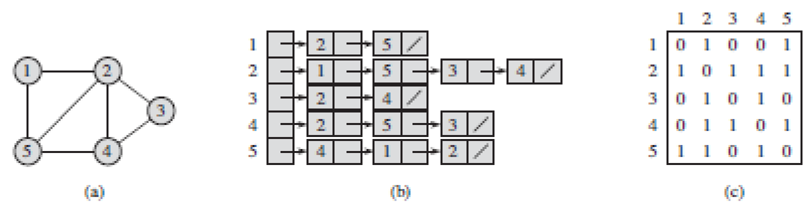
2-اگر وزن از جنس ظرفیت باشد (مانند دبی یک رود)، عدد صفر نوشته میشود.

روش لیست مجاورت رایج‌تر است زیرا می توان به وسیله ی آن گراف های پراکنده (sparse) را به شکلی بهینه (از لحاظ حافظه) ذخیره کرد. در بیشتر الگوریتم‌هایی که خواهیم دید ، گراف ورودی باید به صورت لیست مجاورت نمایش داده شود.

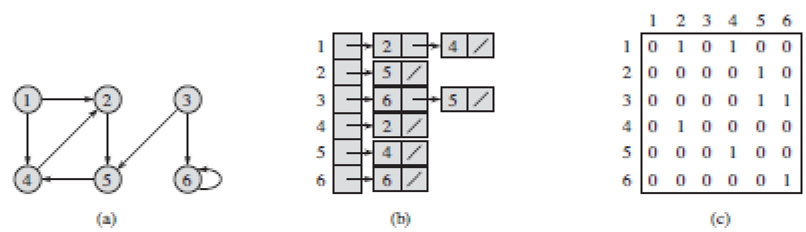
نمایش لیست مجاورت یک گراف G(V, E)، شامل یک آرایه ی |V|تایی از لینک لیست‌هاست. به ازای هر راس u در گراف، یک لینک لیست در این آرایه وجود دارد که به صورت Adj[u] (همسایه‌ی u) قابل دسترسی است و شامل تمام راس هایی چون v است به طوری که (u,v) عضو E است.

اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

اگر گراف جهت‌دار باشد ، حاصل جمع طول های لیست ها برابر با $|E|$ و اگر غیر جهتدار باشد برابر با $2|E|$ خواهد بود. پس هم در مورد گراف های جهتدار و هم غیر جهتدار ، نمایش گراف به صورت لیست مجاورت، به $O(V + E)$ حافظه نیاز دارد. به سادگی گراف‌های وزن‌دار را هم با روش لیست مجاورت می‌توان نمایش داد. به این صورت که به ازای هر (u,v)، می توان $w(u,v)$ را همراه با v در $Adj[u]$ ذخیره کرد.نمایش با لیست مجاورت ، نقطه ضعف نیز دارد. مثلا برای تشخیص اینکه آیا (u,v) در E هست یا نه باید در بدترین حالت تمام عناصر $Adj[u]$ را بررسی کنیم($O(V)$). این در حالی است که در مورد نمایش با ماتریس مجاورت ، این کار در $O(1)$ امکان پذیر است.



شکل -1 نمایش گراف b) با لیست مجاورت c) با ماتریس مجاورت



شکل 2- نمایش بهینه‌ی گراف نسبتا کم‌یالبا لیست مجاورت (b) نسبت به ماتریس مجاورت (c)

با اینکه نمایش با ماتریس مجاورت ، بیشتر از نمایش با لیست مجاورت حافظه می خواهد ، گاهی اوقات به دلیل سادگی، ماتریس مجاورت ترجیح داده می‌شود. در ضمن باید به این نکته توجه کرد که اگر گراف وزن‌دار نباشد ، هر یال را می توان با تنها یک بیت ذخیره کرد.

9-1-1- پیمایش سطحی گراف

پیمایش سطحی گراف، یکی از راحت‌ترین الگوریتم‌ها برای پیمایش یک گراف است. بسیاری از الگوریتم‌های مهم دیگر، شبیه این الگوریتم عمل می‌کنند. به عنوان مثال الگوریتم Prim برای مسأله ی درخت پوشای مینمم و الگوریتم دایکسترا برای مسأله‌ی Single source shortest paths .

در این الگوریتم ، به ازای هر گراف و یک راس دل خواه مانند s، ابتدا تمام راس های قابل دسترس از s، ملاقات می شوند. در ضمن، اگر گراف ورودی بدون وزن باشد، این الگوریتم کوتاه‌ترین فاصله‌ها را از رأس مبدأ تا دیگر رإس ها ، محاسبه می کند. پیمایش سطحی گراف، یک درخت نیز تولید می کند. درخت پیمایش سطحی[6]که شامل تمام راس های قابل دسترس از مبدا است و به ازای هر راس v ، مسیر از s تا v در این درخت ، برابر با یک کوتاه ترین مسیر از s تا v در گراف است. این الگوریتم هم در مورد گراف های جهت دار و هم غیر جهت دار، به کار می رود.

برای در نظر گرفتن وضعیت گراف در مراحل مختلف اجرای الگوریتم، هر راس با یکی از رنگ های سفید ، خاکستری یا سیاه ، رنگ زده می شود. تمام راس ها ابتدا سفید می شوند و ممکن است در آینده خاکستری شده و سپس سیاه شوند.

الگوریتم به این صورت است که ابتدا فقط مبدا را در درخت خود (درخت خروجی BFS) با رنگ خاکستری داریم و بقیه ی رئوس هم سفید هستند. سپس تمام رئوس "سفید" مجاور آن را به درخت و به عنوان بچه ی آن اضافه کرده و آنها را خاکستری میکنیم(discover). حال روی تمام بچه های آن به نوبت این کار را تکرار میکنیم به این صورت که ابتدا این تمام رئوس مجاور بچه هایش را که سفید هستند اضافه کرده و بعد از آن سراغ بچه های آنان (به عبارتی نوه های مبدا) میرویم. هر بار که تمام بچه های یک راس را به درخت اضافه کردیم، رنگ آن راس را سیاه میکنم و کار آن راس تمام میشود(finish). این کار را تا جایی که میتوانیم ادامه میدهیم.

شبه کد زیر ، الگوریتم BFS را نشان می دهد. همان طور که قبلا اشاره شد ، فرض بر این است که گراف ورودی ، به صورت لیست مجاورت ذخیره شده است. در ضمن ، اطلاعاتی چون u.color ، u.d و u.p نیز برای هر راس در نظر گرفته شده است که به ترتیب بیانگر رنگ ، فاصله ی u از s و پدر u هستند. در این الگوریتم از یک ساختمان داده‌ی صف ساده (FIFO Queue) استفاده شده است.

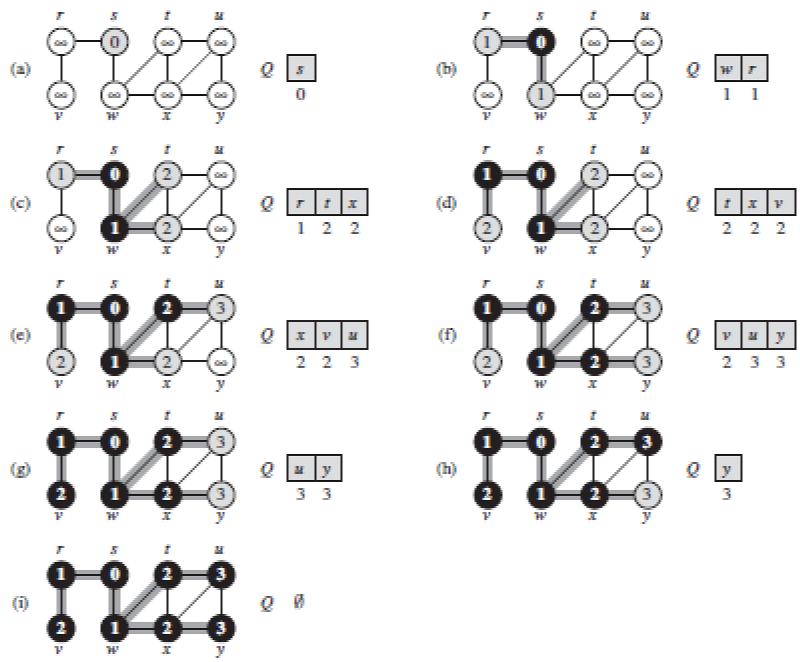
[6]BFS Tree

```
BFS(G, s)
1  for each vertex u ∈ G.V − {s}
2      u.color = WHITE
3      u.d = ∞
4      u.π = NIL
5  s.color = GRAY
6  s.d = 0
7  s.π = NIL
8  Q = ∅
9  ENQUEUE(Q, s)
10 while Q ≠ ∅
11     u = DEQUEUE(Q)
12     for each v ∈ G.Adj[u]
13         if v.color == WHITE
14             v.color = GRAY
15             v.d = u.d + 1
16             v.π = u
17             ENQUEUE(Q, v)
18     u.color = BLACK
```

خطوط 10 تا 18 تا زمانی که هنوز راس خاکستری وجود دارد ، تکرار می شوند. حلقه ی while دارای این invariant است که : در خط 10 Q شامل تمتم راس های خاکستری است.

اثبات بسیار راحت است. قبل از اولین تکرار حلقه ، اولین راس خاکستری ، s است و تنها راس در Q نیز s است. خط 11 راس خاکستری u را از سر صف حذف می کند. در حلقه ی for ، هر راس سفیدی که در همسایگی u قرار دارد ، با اجرای خط های 14 تا 17 کشف می شود و در آخر صف قرار می گیرد. زمانی که تمام راس های واقع در لیست مجاورت u بررسی شد ، u در خط 18 سیاه می شود. Loop invariant گفته شده همواره برقرار است زیرا هر بار که راسی خاکستری می شود ، وارد صف نیز می شود و هر بار که راسی از صف خارج می شود ، سیاه نیز می شود.

نتیجه ی اجرای BFS ، به ترتیب واقع شدن همسایه های یک راس در لیست مجاورت آن راس (خط 12) ، وابسته است. بنا بر این برای یک گراف ممکن است بیش از یک BFS-tree ، داشته باشیم. اما در تمام این درخت ها ، فاصله ها (d) با یکدیگر برابرند.



9-1-2- آنالیز زمان اجرای الگوریتم

قبل از اثبات ویژگی های مختلف BFS ، به بررسی زمان اجرای آن روی گرافی چون G=(V,E) ، می پردازیم. برای این کار از aggregate analysis استفاده شده است.

بعد از initialization ، هیچ راسی دوباره سفید نمی شود. پس با توجه به تست موجود در خط 13، هر راس حد اکثر یک بار وارد صف می شود. بنابر این هر راس حد اکثر یک بار از صف خارج می شود. پس زمان اختصاص داده شده به عملیات صف ، O(V) خواهد بود. به دلیل اینکه در این الگوریتم ، لیست مجاورت هر راس تنها زمانی بررسی می شود که آن راس از صف خارج شده باشد ، لیست مجاورت هر راس حد اکثر یک بار بررسی می شود و از آنجایی که مجموع طول های لیست های مجاورت راس ها O(E) است ، زمان اختصاص یافته به بررسی لیست های مجاورت نیز O(E) است. زمان لازم برای initialization هم O(V) است. پس زمان اجرای الگوریتم O(V+E) است. لازم به ذکر است که اگر در پیاده سازی این الگوریتم از ماتریس برای ذخیره سازی گراف استفاده کنیم، زمان اجرای الگوریتم $O(V^2)$ است.

9-1-3- کوتاه ترین مسیر ها

همان طور که گفته شد ، به ازای هر راس v ، BFS کوتاه ترین فاصله از s تا v را حساب کرده و در v.d ذخیره می کند. طول کوتاه ترین مسیر از s تا v با d(s,v) ، نشان داده می شود. اگر هیچ مسیری از s به v وجود نداشته باشد ، d(s,v) برابر با بی نهایت در نظر گرفته می شود. اکنون یکی از ویژگی های مهم را در باره کوتاه ترین مسیر ها بیان می کنیم.

لم 1

فرض کنید G=(V,E) ، یک گراف جهت دار یا غیر جهت دار باشد و S نیز یک راس دل خواه باشد. آنگاه به ازای هر یال (U,V) خواهیم داشت D(S,V) <= D(S, U) + 1 .

اثبات

اگر u از s قابل دسترسی باشد v نیز هست. در این صورت کوتاه ترین مسیر از s به v نمی تواند بلند تر از کوتاه ترین مسیر از s به u به علاوه ی یال (u,v) باشد. اگر u از s غیر قابل دسترس باشد نیز نامساوی برقرار است.

برای اثبات این که پس از اجرای BFS روی یک گراف ، به ازای هر راس v ، v.d برابر با d(s,v) خوهد بود ، ابتدا نشان می دهیم که در طول اجرا همواره v.d>= d(s,v) است.

لم 2

فرض کنید G=(V,E) ، یک گراف جهت دار یا غیر جهت دار باشد و از راس دل خواهی مانند S روی گراف ، BFS اجرا شده است. آنگاه پس از اتمام اجرای الگوریتم به ازای هر V خواهیم داشت V.D >= D(S,V) .

اثبات

میتوان از استقرا روی تعداد عملیات Enqueue (وارد کردن به صف) استفاده کرد. پایه ی استقرا مربوط به زمانی است که s را در خط 9 وارد Q کرده ایم. فرض استقرا در این مورد درست است زیرا s.d = 0 = d(s,s) و به ازای هر v ، v.d بی نهایت بوده پس از d(s,v) بیشتر است. اکنون راس سفیدی چون u را در نظر بگیرید

که در زمان بررسی لیست مجاورت u ، کشف شده. با توجه به فرض استقرا $u.d>= d(s,u)$ و داریم $v.d = u.d + 1 >= d(s,u) + 1 >= d(s,v)$

در نامساوی بالا از لم 1 استفاده شده است.

سپس v وارد Q می شود و v.d دیگر تغییر نمی کند. بنا بر این همواره $v.d>= d(s,v)$ خواهد بود.

برای اثبات اینکه $v.d = d(s,v)$ ، باید عملکرد Q را دقیق تر بررسی کنیم.

لم 3

فرض کنید در طول اجرای BFS بر روی $G=(V,E)$ ، صف Q شامل راس های $<V_1,V_2, \dots , V_R>$ باشد. آنگاه به ازای

$I = 1, 2, \dots , R-1$ خواهیم داشت:

$$V_R.D <= V_1.D + 1 \quad AND \quad V_I.D <= V_{I+1}$$

اثبات

با استقرا روی تعداد عملیات صف (Enqueue and Dequeue) اثبات می شود. در ابتدا که Q تنها شامل s است ، لم برقرار است. برای استدلال استقرایی باید ثابت کنیم بعد از ورود و خروج یک راس به صف ، همچنان لم برقرار است. اگر سر صف یعنی v_1از صف خارج شود ، v_2سر صف جدید می شود. (اگر صف خالی شود ، لم به وضوح برقرار است.) از روی فرض استقرا داریم

$v_1.d <= v_2.d$ و ضمنا $v_r.d<= v_1.d + 1$ پس $v_r.d<= v_2.d + 1$. پس تمام ویژگی های ذکر شده پس از خروج سر صف ، همچنان برقرار است. اکنون ثابت می کنیم این ویژگی ها پس از ورود یک راس جدید به آخر صف نیز برقرار خواهد بود.زمانی که راسی چون v را وارد صف می کنیم (خط 17) ، تبدیل به v_{r+1} می شود. در این زمان سر صف که u بوده است ، قبلا از صف خارج شده است و اکنون در حال بررسی لیست مجاورت u هستیم. (در واقع v را از روی لیست مجاورت u پیدا کرده ایم.) راسی که پس از حذف u سر صف جدید شده است را v_1 می نامیم. از روی فرض استقرا $v_1.d >= u.d$. بنا بر این

اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

$v_{r+1}.d = v.d = u.d + 1 <= v_1.d + 1$. از روی فرض استقرا همچنین داریم $v_r.d <= u.d + 1$ و بقیه ی نامساوی ها نیز بدون تغییر خواهند بود. بنابراین زمانی که راسی جدید وارد صف میشود نیز لم برقرار است.

نتیجه 4

فرض کنید که V_I و V_J ، در طول اجرای الگوریتم وارد صف شده اند. در ضمن V_I قبل از V_J وارد شده است. آنگاه در زمانی که V_J وارد شده ، $V_I.D <= V_J.D$ است.

اثبات

به راحتی از لم 3 و این ویژگی که هر راس حداکثر یک بار مقدار متناهی d می گیرد ، ثابت می شود.

قضیه 5 (درستی BFS)

فرض کنید $G=(V,E)$ یک گراف جهت دار یا غیر جهت دار باشد که BFS روی یک راس دل خواه از آن مانند S اجرا شده است. آنگاه در طول اجرای BFS ، تمام راس های قابل دسترس از S ، کشف می شوند و پس از اتمام اجرا ، به ازای هر V ، $V.D = D(S,V)$ است. ضمنا به ازای هر V که از S قابل دسترسی است ، یکی از کوتاه ترین مسیر ها از S به V مسیر $V \rightarrow V.P \dashrightarrow S$ است.

اثبات

فرض کنید راس هایی وجود دارند که در آن ها مقدار d برابر با طول کوتاه ترین مسیر از s نباشد. فرض کنید v یکی از این راس ها باشد که در آن $d(s,v)$ از بقیه ی راس ها کمتر است. به وضوح v نمی تواند s باشد. از لم 2 نتیجه می گیریم که $v.d > d(s,v)$ است. در ضمن v باید قابل دسترسی از s باشد. زیرا در غیر این صورت $v.d < d(s,v)$ می بود.(زیرا $d(s,v)$ برابر با بی نهایت است.) فرض کنید که u راسی باشد که در یکی از کوتاه ترین مسیر ها از s به v ، پدر v است. به طوری که $d(s,v) = d(s,u) + 1$ به دلیل روشی که با آن v را انتخاب کرده ایم ، $u.d = d(s,u)$ است. بنابراین

$$v.d > d(s,v) = d(s,u) + 1 = u.d + 1 \qquad (1)$$

اکنون زمانی را در نظر بگیرید که u از Q خارج میشده است. در لحظه v سفید ، خاکستری یا سیاه است. نشان خواهیم داد که در هر صورت به تناقض خواهیم رسید. اگر v سفید باشد آنگاه با استفاده از نتیجه ی 4 ، $v.d <= u.d$ است که با (1) در تناقض است. اگر v خاکستری باشد ، آنگاه v زمانی خاکستری شده که راسی چون wاز صف خارج می شده است. و w زودتر از u از صف خارج شده و $v.d = w.d + 1$ است. با نتیجه ی 4 $w.d <= u.d$ است. بنابراین ، $v.d = w.d + 1 = u.d + 1$ که با (1) در تناقض است. پس نتیجه می گیریم که به ازای هر v ، $v.d = d(s,v)$ است. پس تمام راس هایی چون v که از s قابل دسترس اند ، در پایان الگوریتم کشف می شوند. زیرا در غیر این صورت v.d بی نهایت بوده و از d(s,v) بیشتر می بود.

9-1-4- BFS-trees

برای گراف G=(V,E) و راس مبدا s، زیر گراف G_p، اینگونه تعریف می شود.

$$G_p = (V_p, E_p) \text{ where}$$

$$V_p = \{ v \$ V : v.p != NULL\} + \{s\}$$

$$E_p = \{(v.p, v) : v \$ V_p - \{s\}\}$$

G_pیک BFS-tree است اگر V_p شامل تمام راس های قابل دسترس از s باشد و به ازای هر v ، G_p شامل یک مسیر یکتا از s به v باشد به طوری که این مسیر یک کوتاه ترین مسیر در G باشد. BFS-tree یک درخت است زیرا همبند است و $|E_p| = |V_p| - 1$

لم 6 نشان می دهد که G_p که در طول اجرای الگوریتم به دست آمده ، یک BFS-tree است.

لم 6

زمانی که BFS روی یک گراف جهت دار یا غیر جهت دار چون G=(V,E) اجرا می شود ، G_p یک BFS-TREE است.

اثبات

خط 16 v.p را u قرار می دهد اگر و تنها اگر (u,v) یکی از یال های G باشد و d(s,v) غیر بینهایت باشد. یعنی v از s قابل دسترسی باشد. بنا بر این V_p شامل تمام راس هاییست که از s قابل دسترسی اند. چون G_p

اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

یک درخت است ، به ازای هر v در V_p ، یک مسیر ساده ی یکتا از s به v وجود دارد. با به کار گیریقضیه 5 به طور متوالی ، نتیجه می شود که هرکدام از این مسیر ها در G_p یک کوتاهترین مسیر در G هستند.

رویه ی زیر یکی از کوتاه ترین مسیر ها از s به vرا چاپ می کند.

```
PRINT-PATH(G, s, v)
if v == s
    print s
elseif v.π == NIL
    print "no path from" s "to" v "exists"
else PRINT-PATH(G, s, v.π)
    print v
```

این الگوریتم در زمانی خطی نسبت به |V| اجرا می شود.