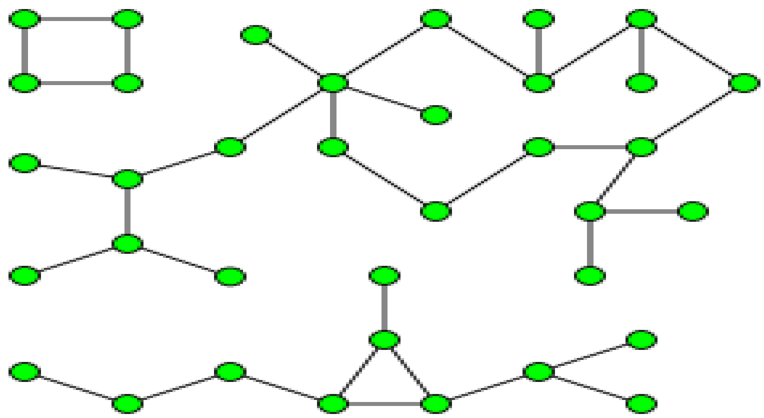
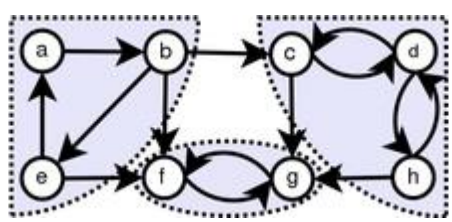
8.4. اجزای همبند

8.4.1. تعاریف

جز همبندی: جز همبندی زیرگرافی از یک گراف بدون جهت است که در آن هر دو گره توسط حداقل یک مسیر به هم وصل باشند و امکان اضافه کردن هیچ راس یا یالی به این جز وجود نداشته باشد به صورتی که این زیرگراف همچنان جز همبندی باقی بماند. برای مثال گراف زیر دارای 3 مولفه همبندی است:



مولفه قویا همبند: زیرگرافی از یک گراف جهت دار است که در آن به ازای هر دو راس a, b از a به b و از b به a مسیر وجود داشته باشد و امکان اضافه کردن هیچ راس یا یالی به این جز وجود نداشته باشد به صورتی که این زیرگراف همچنان یک مولفه قویا همبند باقی بماند. در گراف زیر دور مولفه های قویا همبند خط کشیده شده است:



گراف معکوس:گراف معکوس یا همان G^T که در آن راس u به راس v یال دارد اگر و فقط اگر در گراف G راس v به راس u یال داشته باشد. یعنی مجموعه یال‌های آن به صورت زیر است:

$$E^T = \{(u,v) \mid (v,u) \in E\}$$

8.4.2. تشخیص مولفه های قویا همبند در گراف جهت دار

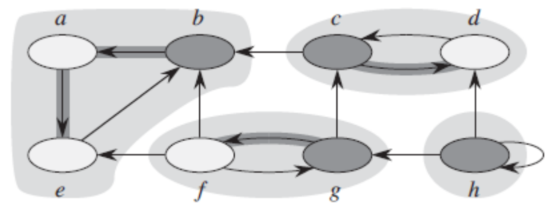
یکی از کاربرد های کلاسیک الگوریتم DFS تجزیه یک گراف جهت دار به مولفه های قویا همبندش می باشد. در این قسمت نشان میدهیم که چگونه با دو بار استفاده از الگوریتم DFS می توان این کار را انجام داد. بسیاری از الگوریتم هایی که با گراف های جهت دار کار می کنند، ابتدا گراف مورد نظر را به مولفه های قویا همبندش تجزیه می کنند، سپس در هر جز عملیات مورد نظرشان را انجام می دهند و در نهایت با توجه به ارتباط بین مولفه ها پاسخ ها را با هم ترکیب کرده و پاسخ نهایی را به دست می آورند.

الگوریتم زیر در پیچیدگی زمانی θ(V+E) مولفه های قویا همبند گراف $G = (V, E)$ با استفاده از دو DFS یکی بر روی G و دیگری بر روی G^T به دست می آورد:

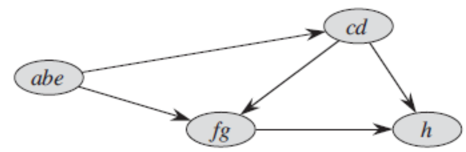
1. DFS(G) محاسبه شود و finish time هرکدام از راس ها به دست آید.(f(u) ها)
2. G^T محاسبه شود.
3. DFS (G^T) محاسبه شود. فقط با این فرض که در حلقه اصلی DFS بر حسب میزان f[u]، گره ها مرتب شود.
4. گره های هر درخت در جنگل به دست آمده از خط 3 را به عنوان یک مولفه قویا همبند جداگانه خروجی بده.

ایده ای که پشت این الگوریتم است برخواسته از یک خاصیت کلیدی component graph G^{SCC}=(V^{SCC}, E^{SCC}) ، که به این صورت تعریف می شود: فرض کنید که G شامل مولفه های قویا همبند C_1, C_2,..., C_k می شود. ست گره V^{SCC} به صورت $\{v_1, v_2,..., v_k\}$ و شامل گره v_i برای هرکدام از مولفه های C_i از گراف G می باشد. یال $v_i v_j$ در صورتی عضو مجموعه E^{SCC} می باشد که G برای یک x, y شامل یک یال جهتدار باشد به صورتی که x گره ای از C_i و y گره ای از C_j باشد. برای مثال در شکل زیر گراف شکل a گراف G و گراف شکل b گراف G^{SCC} آن می باشد.

(a)



(b)



خاصیت کلیدی مورد نظر ما این است که گراف G^{SCC} (component graph) یک DAG است که با استفاده از لم زیر ثابت می شود:

لم 1: فرض کنیم C, C’ دو جز قویا همبند هستند و uv یالی از C و u’v’ یالی از C’ باشد. اگر مسیر از u به u’ وجود داشته باشد، از v’ به v مسیری نداریم.

اثبات لم 1: اگر G شامل مسیر v’→v باشد، شامل مسیر های u→u’→v’ و v’→v→u نیز می شود. بنابراین هم از u به v’ و هم از v’ به u مسیر هست که این با فرض اولیه ما که C و C’ دو مولفه قویا همبند هستند در تناقض است، بنابراین از v’ به v مسیر موجود نمی باشد.

خواهیم دید که در DFS دوم با در نظر گرفتن گره ها به صورت نزولی finish time (که در DFS اول به دست آمده بود)، در واقع داریم گره های component graph را با ترتیب توپولوژیکال می بینیم.

برای جز همبند C تعریف می کنیم:

d(C) = min d(u) u∈C

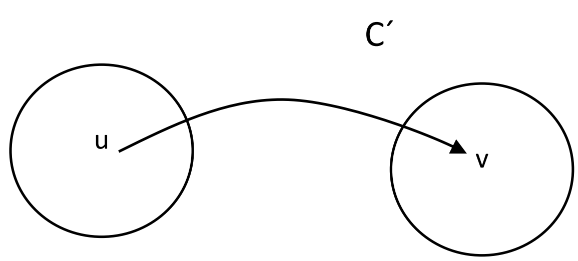
f(C) = max f(u) u∈C

لم زیر و زیر قضیه آن به ما خاصیت کلیدیی می دهد که مولفه های قویا همبند و fininshing time آن ها را در DFS اول با هم مرتبط می کند:

لم 2: اگر C, C’ دو جز قویا همبند باشند، به طوری که u راسی از C و v راسی از C’ باشد راس uv در گراف موجود باشد، f(C)>f(C’)

اثبات :دو حالت باید در نظر گرفته شود، $d(C) < d(C')$ و $d(C) > d(C')$، یعنی اینکه اول راسی از C، discover شود یا راسی از C´. اگر $d(C) < d(C')$ و اول راس x از C، discover شود، در زمان x.d رنگ همه راس ها در C و C´ سفید است و اگر u هر راس دیگر در C باشد، مسیری از x به u وجود دارد که همه راس های مسیر سفید رنگ هستند. همچنین چون $(u, v) \in E$، اگر w راسی در C´ باشد، مسیری از x به w وجود دارد که همه راسهای آن مسیر سفید رنگ هستند. طبق قضیه white-path، تمام راس های در C و C´ از نوادگان راس x هستند و چون زمان پایان x از همه نوادگانش بیشتر است $f(C) = x.f$ و $f(C) > f(C')$.

حال اگر داشته باشیم $d(C) > d(C')$ اول راس y از C´، discover شده است و در زمان y.d، همه راس ها در C´ سفید هستند و مسیری از y به هر راس در C´ وجود دارد که همه راس های آن سفید است. طبق قضیه white-path، همه راس های C´ از نوادگان y هستند پس $f(C') = y.f$. در زمان u.f همه راس های C سفید رنگ هستند. چون یال (u, v) از C به C´ وجود دارد طبق لم بالا یالی از C´ به C وجود ندارد و هیچ مسیری از y به راسی در C وجود ندارد پس در زمان y.f، همه راس های C همچنان سفید هستند و برای هر راس w در C داریم w.f>y.f. پس $f(C) > f(C')$.■



دو جز قویا همبند C و C´. اگر یال (u, v) از جز C به C´ موجود باشد $f(C) > f(C')$

زیر قضیه لم 2: اگر C و C' دو جز قویا همبند باشد و u راسی از C و u' راسی از C' باشد و uu' راسی از G^T باشد، $f(C)<f(C')$

اثبات: چون $(u, v) \in E^T$ پس $(v, u) \in E$ و چون اجزای قویا همبند G و G^T برابرند طبق لم بالا $f(C)< f(C')$..

حال به اثبات درستی الگوریتم می پردازیم: با استفاده از استقرا روی تعداد درخت های به دست آمده از DFS در خط 3 الگوریتم، اثبات می کنیم که گره های هر کدام از این درخت ها یک مولفه همبندی را تشکیل می دهند. پایه استقرا برای k = 0 که واضح است. فرض استقرا را این می گیریم که هر کدام از k درخت اول به دست آمده در خط 3 الگوریتم مولفه قویا همبند باشد. حال k+1 امین درخت را در نظر می گیریم. ریشه این درخت را u می نامیم و مولفه همبند حاوی u را C می نامیم. به دلیل نحوه انتخاب ریشه ها در DFS خط 3، f(u) = f(C)>f(C') به ازای هر کدام از مولفه های قویا همبند C به غیر از C' که هنوز ملاقات نشده اند. واضح است که در زمان ملاقات u همه گره های دیگر C سفیدند. بنابراین با استفاده از قضیه white path تمام گره های دیگر C در df tree مربوط به u نوادگان u هستند. همچنین با استفاده از زیر قضیه لم2، هر یالی که از G^T خارج می شود باید به مولفه های همبندی که قبلا ملاقات شده اند وارد شود. بنابراین هیچ گره ای در مولفه ی قویا همبندی به غیر از C نوه ی u در df tree خط 3 نخواهد بود. بنابراین گره های df tree مربوط به G^T که نواده u می باشند، دقیقا یک مولفه قویا همبند می سازند.

شبه کد مربوط به این الگوریتم به صورت زیر است:

```
1  Algorithm  find_strongly_connected_components(G,v)
 2  Input : G=(V,E),)
 3  Output : strongly connected component (SCC)
 4  begin
 5      while(stack does not contain all vertices)
 6         Choose an arbitrary vertex v not in S
 7         Depth_First_Search_Node1(G,V)
 8         Each time that depth-first search finishes
expanding a vertex u, push u onto S
 9      compute Gᵀ
10      While(stack is not empty)
11           pop vertex v form stack
12           Depth_First_Search_Node(Gᵀ,v)
13           Output set of visited vertices as a
seprate SCC
14           mark this set of vertices form stack and
graph
12  end
```

که در آن Depth_First_Search_Node همانند DFS است با این تفاوت که حلقه ی while آن به بیرو از آن انتقال یافته و همچنین برای Depth_First_Search_Node1 تفاوتی که در خط 8 نوشته شده را نیز دارا می باشد.