



## ساختمان داده‌ها (۲۲۸۲۲)

مدرس: حسین بومری

[زمستان ۹۹]

نگارنده: آئیریا محمدی

جزوه جلسه ۲۶: جستجوی عمق اول/ مولفه‌های هم‌بندی/ مرتب‌سازی توپولوژیک

### ۱ مرور

در دو جلسه اخیر با گراف و دو طریق نگهداری آن (ماتریس و لیست مجاورت) آشنا شدیم و دو راه برای پیمایش آن یاد گرفتیم (جستجوی اول سطح و اول عمق). در این جلسه به پیاده‌سازی این دو الگوریتم می‌پردازیم و شباهت‌هایشان را خواهیم دید.

فرض کنیم که ما می‌خواهیم یک گراف را پیمایش کنیم. در جستجوی اول سطح با شروع از یک گره و ملاقات آن کلیه گره‌های مجاور آن نیز ملاقات می‌شوند. این روند برای نقاط همسایه نیز تکرار می‌شود. در جستجوی اول عمق از یک نقطه شروع می‌کنیم و پس از ملاقات آن به یک سمت حرکت می‌کنیم. یعنی از آن به راس بعدی رفته و از راس بعدی تا وقتی که در همسایه‌اش کسی دیده نشده نباشد این روند را ادامه می‌دهیم. آنگاه عقب‌گرد کرده تا به راس اول برسیم و الگوریتم تمام شود. جلسه پیش یک پیاده‌سازی هم دیدیم و تحلیل کردیم مرتبه زمانی آن برای گراف با ماتریس از مرتبه  $O(V^2)$  می‌شود و با لیست مجاورت از مرتبه  $O(V + E)$ .

### ۲ پیاده‌سازی DFS

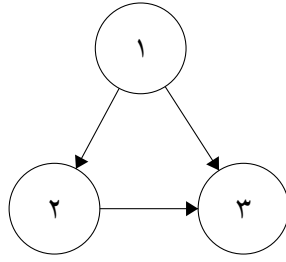
پیشنهاد اول برای این الگوریتم به شکل زیر و به کمک پشته است.

```
ds
ds.push(start)
while(ds.notEmpty) {
  cur = ds.pop()
  visited[cur]=true
  for each non visited v in N(cur)
    ds.push(v)
}
```

شکل ۱: پیاده‌سازی اول

اما مشکلی که در این جا وجود دارد این است ممکن است راسی دوبار دیده شود.<sup>۱</sup> پس ابتدا بررسی می‌کنیم که راس قبلاً دیده نشده باشد.

<sup>۱</sup> توجه کنید آن‌چه که به آن توجه شده است این است که راسی که بررسی می‌کنیم دیده نشده باشد اما ممکن است یک راس پیش از دیده شدن توسط چندین همسایه‌اش به پشته اضافه شود.



شکل ۲: مشکل دوبار دیده شدن. راس ۳ دو بار توسط راس‌های ۱ و ۲ به پشته اضافه می‌شود.

حال سوال پیش می‌آید که زمان اجرای این الگوریتم در صورت ذخیره گراف به شکل‌های ماتریس یا لیست مجاورت چگونه خواهد بود. ماتریس و لیست مجاورت در زمان اجرای foreach که راس‌های همسایه را نگاه می‌کند تفاوت ایجاد خواهند کرد.

حلقه while به تعداد دفعاتی که در پشته چیزی وارد می‌کنیم اجرا می‌شود و هر راس تنها یک بار از if رد می‌شود (شکل ۳) چون تنها بار اولی که هر راس را می‌بینیم visited اش false است. پس برای هر راس همسایه‌هایش یک بار بیشتر درج نمی‌شود و جمعاً از  $O(E)$  درج انجام می‌شود پس حلقه بیرونی نیز به همین اندازه اجرا خواهد شد. اگر گراف به شکل لیست مجاورت نگهداری شده باشد همسایه‌های رئوس جمعاً در  $O(E)$  دیده می‌شوند. اما از ماتریس مجاورت استفاده کنیم برای هر راس باید مجاورت داشتش با همه رئوس دیگر را بررسی کنیم که  $O(V^2)$  خواهد بود.

پس DFS با ماتریس مجاورت مرتبه زمانی اش  $O(V^2)$  خواهد بود یا  $O(V^2 + E)$  برای وقتی که یال‌های چندگانه داشته باشیم. اگر الگوریتم را با لیست اتصال بزنیم مرتبه for به جای  $O(V^2)$  به  $O(E)$  تبدیل خواهد شد و مرتبه کل از  $O(V + E)$  خواهد بود.

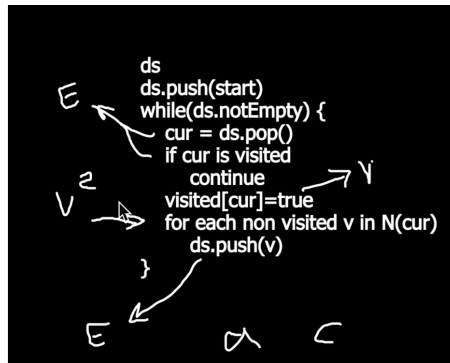
**شباهت با BFS** برای پیاده‌سازی BFS هم می‌توانیم در کد بالا پشته را به صف تبدیل کنیم!

## ۳ back edge, forward edge, cross edge

**تعریف** اگر در درخت جستجوی به دست آمده یک گره به نوه‌اش یالی در گراف اصلی داشته باشد به آن forward edge می‌گوییم. اگر یک گره به پدر غیرمستقیم خود (اجداد پدرش) یال داشته باشد back edge داریم و اگر بین دو گره که هیچ کدام جد دیگری نباشد یال

```
ds
ds.push(start)
while(ds.notEmpty) {
  cur = ds.pop()
  if cur is visited
    continue
  visited[cur]=true
  for each non visited v in N(cur)
    ds.push(v)
}
```

شکل ۳: پیاده‌سازی DFS با پشته

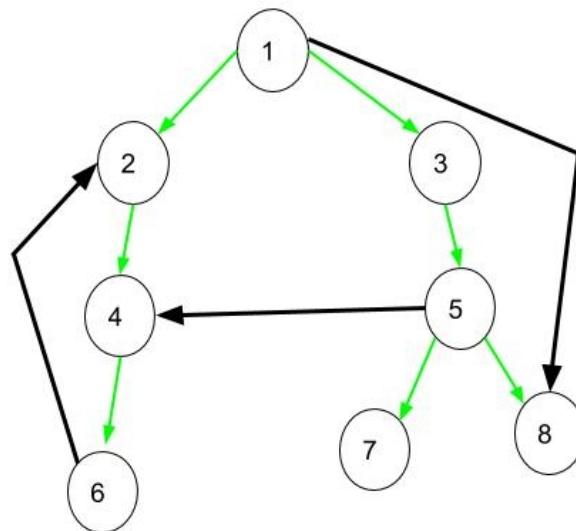


شکل ۴: تحلیل زمانی DFS با ماتریس مجاورت

داشته باشیم cross edge داریم.

**سوال** آیا DFS می‌تواند forward edge داشته باشد؟ بله. back edge چگونه؟ بله. cross edge چه؟ اگر گراف دوطرفه باشد cross به وجود نمی‌آید. ولی اگر جهت‌دار باشد ممکن است/ ولی یال‌هایی که هستند از راست به چپ می‌باشند/ یعنی از کسانی که دیرتر دیده شدند به سمت کسانی که زودتر دیده شدند.

**سوال** توی BFS آیا forward edge داریم؟ نه. چون تمام همسایه‌های یک راس گراف در درخت جستجو BFS مثبت منفی یک اختلاف سطح دارند. ولی back edge اگر گراف جهت‌دار باشد می‌توانیم داشته باشیم (و در غیر این صورت نه). cross edge نیز داریم اما یا هم سطح خود گره/ یا به سطح قبلی و یا بعدی.



شکل ۵: از راست به چپ یال‌های پیشرو (forward edge) / عبوری (cross edge) و پستی (بازگشتی) (back edge)

## ۴ کاربرد های DFS و BFS

BFS نزدیک ترین فاصله راس ها تا راس ما رو میده (شماره سطر ها همان فاصله می شوند). و از DFS برای پیدا کردن دور در گراف می توان استفاده کرد. یک کاربرد دیگر این دو الگوریتم را در جلوتر می بینیم.

### ۱.۴ connectivity

حال درباره اتصال<sup>۲</sup> صحبت می کنیم.

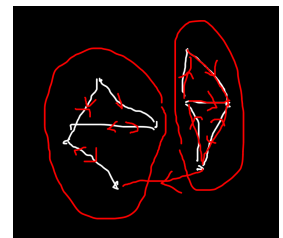
**تعریف اتصال** در گراف بدون جهت دو راس که به هم مسیر دارند را متصل<sup>۳</sup> می گوئیم.

**تعریف مولفه هم بندی** مجموعه راس هایی که هر دو راس آن به هم مسیر دارند و از هیچ راس خارج از این مجموعه به آن ها مسیری نباشد یک مولفه هم بندی<sup>۴</sup> تشکیل می دهند.

با پیدا کردن مولفه های هم بندی گراف را می توان به یک سری زیر گراف شکست که به هم هیچ یالی ندارند. پیدا کردن مولفه های هم بندی با هر دو الگوریتم BFS و DFS امکان پذیر است. کافی است یک حلقه بالای الگوریتم های مطرح شده ( شکل ۳ ) اضافه کنیم برای کامپوننت های مختلف. پیمایش با شروع از هر راس که تمام شود در پایانش یک مولفه هم بندی (شامل آن راس) را می دهد.

در گراف جهت دار بین دو مولفه هم بندی می تواند یال (تنها از سمت یک مولفه) هم باشد. (شکل ۱.۴) البته در آن جا مولفه قویا هم بندی<sup>۵</sup> گفته می شود.

فرض کنید مولفه های هم بندی مختلف داریم. حال اگر به جای رأس ها به مولفه های هم بندی نگاه کنیم می توان گراف را منقبض کرد. هر یالی که از راس های انقباض به بیرون است بیرون می ماند و اگه یالی به داخل است داخل آن می ماند.



اگر تمام راس های مولفه های قویا هم بند را منقبض کنیم، گراف حاصل دیگر دور ندارد.<sup>۶</sup>

**تعریف DAG** به گراف هایی که دور ندارند (Directed acyclic graph (DAG می گویند.

شکل ۶: دو مولفه قویا هم بندی و یال بین آن دو

و خاصیت DAG این است که می توان آن را سطح بندی کرد. البته لزوماً توی هر سطح یک راس نمی افتد اما جهت یال ها حتماً از یک سطح به سطح پایین تر خواهد بود. انگار برای راس ها ترتیب قائل شدیم. هیچ دو سطح هم نمی توانند به هم یال داشته باشند (وگرنه از اول در یک سطح وارد نمی شدند).

<sup>۲</sup>connectivity

<sup>۳</sup>connected

<sup>۴</sup>connected component

<sup>۵</sup>strongly connected component

<sup>۶</sup>چرا که اگر دور داشت دو مولفه به هم مسیر رفت و برگشت پیدا می کردین و هر دو راس در یک مولفه به هم مسیر دارند پس دو راس دو مولفه مختلف به هم مسیر می داشتند و باید در آن صورت در یک مولفه می بودند که این امکان پذیر نیست.

## ۲.۴ ترتیب جزئی

برای مجموعه رأس‌های  $A$  زیر مجموعه  $A^2$   $C \subset A^2$  را در نظر بگیرید که  $\forall (a, b) \in C \rightarrow a < b$ . این مجموعه  $A$  دارای ترتیب جزئی است.

با سطح‌بندی راس‌ها می‌توان برایشان ترتیب جزئی تعریف کرد. به این گونه که کسانی که دیرتر قرار گرفتند از قبلی‌ها بزرگ‌تر نیستند (یا هستند). جهت یال‌ها نیز از بالا به پایین خواهد بود.

پس هر ترتیب جزئی را می‌توان با dag نشان داد. برای ترتیب کلی چه باید کرد؟<sup>۷</sup> (این که هر دو تایی قابل مقایسه باشد).

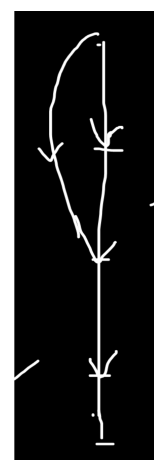
برای ترتیب جزئی نیاز است که عملگرمان valid باشد و این یعنی که گرافش نباید دور داشته باشد و اگر یالی دارد یعنی اولی از دومی کوچک‌تر است. ترتیب‌های جزئی با یک عملگر کوچک‌تر تعریف می‌شوند (مساوی نداریم).

گراف جهت‌دار بدون دور لزوماً تعدی ندارد اما گراف ترتیب جزئی تعدی دارد. اگر برای هر مسیر یک یال داشته باشیم یا داشتن مسیر به معنای کوچک‌تری مبدا از مقصد باشد یک تناظر یک به یک بین dag دارای تعدی و ترتیب جزئی وجود خواهد داشت. در آن صورت ترتیب کلی هم می‌شود تعریف کرد که گراف آن شبیه به یک خط خواهد بود.

حال این سوال وجود دارد که چگونه از dag ترتیب جزئی نتیجه بگیریم. جلسه بعد درباره topological sort و الگوریتم آن صحبت می‌کنیم.

### کلیدواژه‌ها

- شباهت DFS و BFS
- پیاده سازی DFS با پشته
- پیاده سازی BFS با صف
- مرتبه زمانی DFS با ماتریس و لیست مجاورت
- مولفه‌های (قویا) هم‌بندی
- الگوریتم اولیه برای پیدا کردن مولفه‌های هم‌بندی
- ترتیب جزئی و کلی



شکل ۷: ترتیب کلی

<sup>۷</sup> به مجموعه مرتبی که همه اعضا آن قابل قیاس باشد، ترتیب کامل (total order یا linear order) می‌گوییم.