جزوه جلسه بیستم داده ساختارها و الگوریتم

۹ آذر ۱۴۰۰

فهرست مطالب

۱ محدودیت های DFS

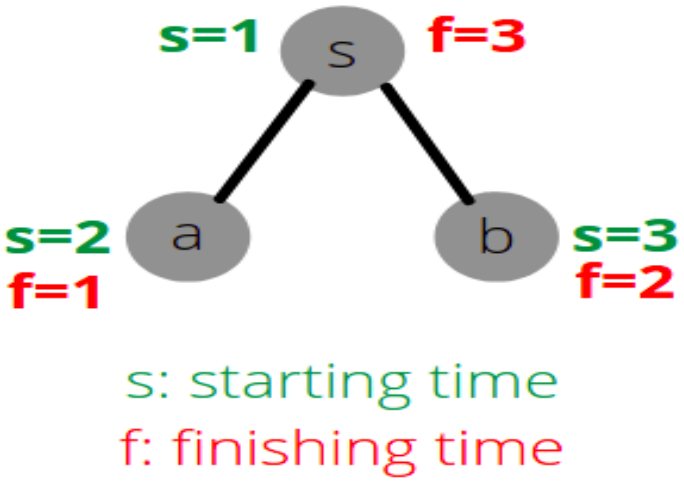
این الگوریتم برای گراف هایی با تعداد راس و یال فراوان می تواند منجر به بروز خطا بشود. حافظه ای که کامپیوتر از سیستم عامل می گیرد، بخش کمی از آن را به stack اختصاص می دهد و به همین دلیل،پیمایش یک مسیر طولانی می تواند منجر به بروز خطای پر شدن حافظه استک بشود.

۲ کاربردهای الگوریتم DFS

پیش از شروع، دو مفهوم در مورد الگوریتم DFS:

- **starting time**: زمان اجرای تابع visit را برای هر راس نشان می دهد. (بعد از شروع اجرای تابع و خط def visit(s)
- **finishing time**: زمانی را نشان میدهد که اجرای تابع visit برای راس تمام شده است. (در انتهای تابع و خارج از حلقه for)

توجه شود که زمان های فوق منطقی هستند و صرفا توالی زمانی را نشان میدهند.
مثال یک گراف سه راسی در شکل زیر آمده است.



شکل ۱: زمان های شروع و پایان برای یک گراف سه راسی

حال به ذکر کاربردها می پردازیم:

۱. پیدا کردن دور در گراف جهت دار و بدون جهت

۲. بررسی دو بخشی بودن گراف برای رنگ آمیزی (رنگ آمیزی گراف به نحوی است که رنگ دو سر هر یال متفاوت باشد)

۳. مرتب سازی توپولوژیک گراف جهت دار بدون دور (DAG)

۴. مولفه های همبندی گراف (با اجرای هر حلقه for در گراف، رئوسی که دیده شده باشند در یک مولفه قرار می گیرند.)

۵. پیدا کردن مولفه های قویا همبند در گراف جهت دار (بین هر دو راس در یک مولفه قویا همبند مسیر رفت و برگشت وجود دارد)

۶. پیدا کردن مولفه های دوهمبند راسی و یالی در گراف بدون جهت (در گراف دوهمبند راسی بین هر دو راس، ۲ مسیر وجود دارد که مسیرها راس مشترک ندارند و در گراف دوهمبند راسی بین هر دو مسیر بین ۲ راس، راس مشترک می تواند وجود داشته باشد ولی مسیرها یال مشترک ندارند)

۷. پیدا کردن راس یا یال برشی (با حذف راس یا یال برشی، مولفه های همبندی گراف افزایش می یابد)

با اضافه کردن starting time و finishing time، عملیات بالا در زمان $\Theta(n+e)$ انجام می گیرد.
حال به بررسی مورد اول، سوم و پنجم می پردازیم

۱.۲ پیدا کردن دور در گراف

ادعا میکنیم در هر گراف جهت دار یا بدون جهت دور وجود دارد اگر و تنها اگر یال بازگشتی (backward) داشته باشیم.
برای گراف جهت دار و بدون جهت اگر یال بازگشتی داشته باشیم، طبق تعریف یال بازگشتی دور نیز تشکیل می شود. حال اگر فرض کنیم دور داشته باشیم برای اثبات وجود یال بازگشتی مسیر $<v_1, v_2, ..., v_k>$ را در نظر می گیریم.

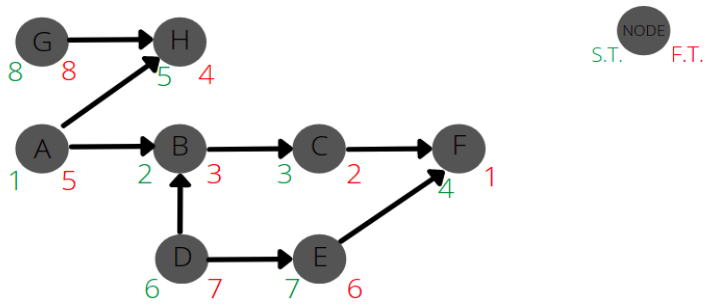
- لم: اگر یال (v_i, v_{i+1}) در گراف وحود داشته باشد، قبل از به پایان رسیدن $\text{visit}(v_i)$ تابع $\text{visit}(v_{i+1})$ با پایان می رسد.

اگر $\text{visit}(v_i)$ بعد از انمام $\text{visit}(v_{i+1})$ شروع شده باشد که لم اثبات می شود.
اگر $\text{visit}(v_{i+1})$ بعد از $\text{visit}(v_i)$ شروع شود، یا بعد مستقیما بعد از ویزیت v_i شروع

شده یا غیرمستقیم و بعد از فراخوانی visit برای چند راس دیگر. در هر دو حالت نیز مسئله اثبات شده است.
حال فرض کنید v_0 اولین راسی باشد که آنرا ویزیت می کنیم. به صورت استقرایی می توان ثابت کرد قبل از اتمام $\text{visit}(v_i)$، تابع $\text{visit}(v_k)$ برای رئوسی که $k > i$ است شروع شده وقبل از به پایان رسیدن $\text{visit}(v_i)$ به پایان رسیده است. پس پیش از اتمام $\text{visit}(v_0)$، تابع ویزیت برای همه v_iها شروع شده است. پس یال (v_k, v_0) یک یال بازگشتی است.

۲.۲ مرتب سازی توپولوژیک یک گراف جهت دار بدون دور

فرض کنید تعدادی کار داریم که بعضی از آنها پیش نیاز بعضی دیگر هستند. با مرتب سازی توپولوژیک میتوان ترتیبی از کارها ارائه داد که در آن پیش نیازی ها رعایت شده باشد.
برای این کار می توان رئوس را به ترتیب عکس finishing timeها مرتب کرد و خروجی داد. شکل زیر یک مثال از گراف ۸ راسی است.



طبق شکل بالا، یک ترتیب برای انجام کارها

G, D, E, A, H, B, C, F

می باشد.
توجه داشته باشید این مسئله با BFS قابل حل نیست. زیرا ممکن است بعضی رئوس پیمایش نشوند.
برای اثبات درستی این الگوریتم نیز از لم قبلی میتوان اثبات کرد اگر یال (u,v) در گراف وجود داشته باشد، قبل از اینکه $\text{visit}(u)$ به پایان برسد $\text{visit}(v)$ شروع می شود.
همچنین به دلیل عدم وجود دور اثبات فوق انجام می شود.

۳.۲ پیدا کردن مولفه های قویا همبند

با یک بار اجرای الگوریتم DFS نیز می توان مولفه های قویا همبند را پیدا کردن اما الگوریتم ذکر شده نیازمند دو بار اجرای الگوریتمDFS است.

۱. یکبار DFS را روی گراف اجرا می کنیم و finishing time رئوس را به دست می آوریم.

۲. جهت یال ها را عوض کرده و مجددا با اجرای DFS، زمان اتمام رئوس را محاسبه میکنیم.

۳. finishing timeها را به صورت نزولی مرتب می کنیم و راس ها را در هر درخت حاصل از جست و جوی عمق اول به عنوان یک مولفه قویا همبند خروجی می دهیم.

برای اطلاعات بیشتر به صفحه ویکی پدیا الگوریتم مراجعه کنید.
`https://en.wikipedia.org/wiki/Kosaraju's_algorithm`