جست‌وجوی عمق‌اول

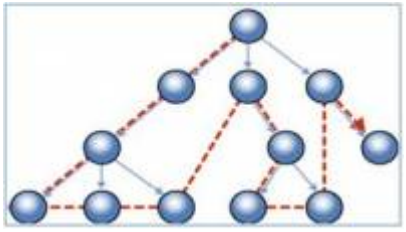
تعریف

جست‌وجوی عمق‌اول که به DFS ($DepthFirstSearch$) معروف است در واقع الگوریتمی برای پیمایش گراف است. شاید با کمی شک بتوان گفت که پرکاربردترین الگوریتم در گراف همین الگوریتم است چراکه هم کد آن کم است، هم هزینه زمانی و حافظه‌ای آن کم است، هم برای اکثر سوال‌های گراف نیاز به پیمایش است.

الگوریتم

الگوریتم به این شکل است که ابتدا یک رأس مانند s را انتخاب می‌کنیم و آن را ریشه می‌نامیم. ریشه را علامت‌گذاری می‌کنیم. سپس یک رأس دل خواه علامت نخورده‌ی مجاور با v را انتخاب می‌کنیم و آن را u می‌نامیم. u را یکی از بچه‌های v می‌کنیم، سپس u را علامت می‌زنیم. حال همین الگوریتم را روی u از ابتدا اجرا می‌کنیم (یعنی یکی از همسایه‌های مجاور و علامت نخورده u را انتخاب می‌کنیم و $\ldots$).

الگوریتم گفته شده زمانی به بن‌بست می‌خورد که به ازای راسی مانند u، تمام همسایه‌هایش علامت خورده باشند. در این صورت به راس پدر (رأسی که از آن وارد u شدیم) برمی‌گردیم و دوباره همین الگوریتم را از ادامه اجرا می‌کنیم (یعنی وارد پدر u می‌شویم و اگر همسایه‌ی علامت نخورده‌ای داشت وارد آن می‌شویم و اگر نداشت به رأس پدرش باز می‌گردیم).
برنامه زمانی متوقف میشود که به رأس v برگشته باشیم و تمام همسایه‌هایش علامت خورده باشند که در این صورت می‌گوییم الگوریتم پایان یافته است.
$\ldots$دقت کنید که اگر گراف شما همبند نباشد، این جست‌وجو تنها رأس‌های مؤلفه ریشه را پیمایش می‌کند پس اگر برای پیمایش روی تمام رأس‌ها این الگوریتم را به ازای هر رأس علامت‌نخورده‌ای تکرار می‌کنیم.



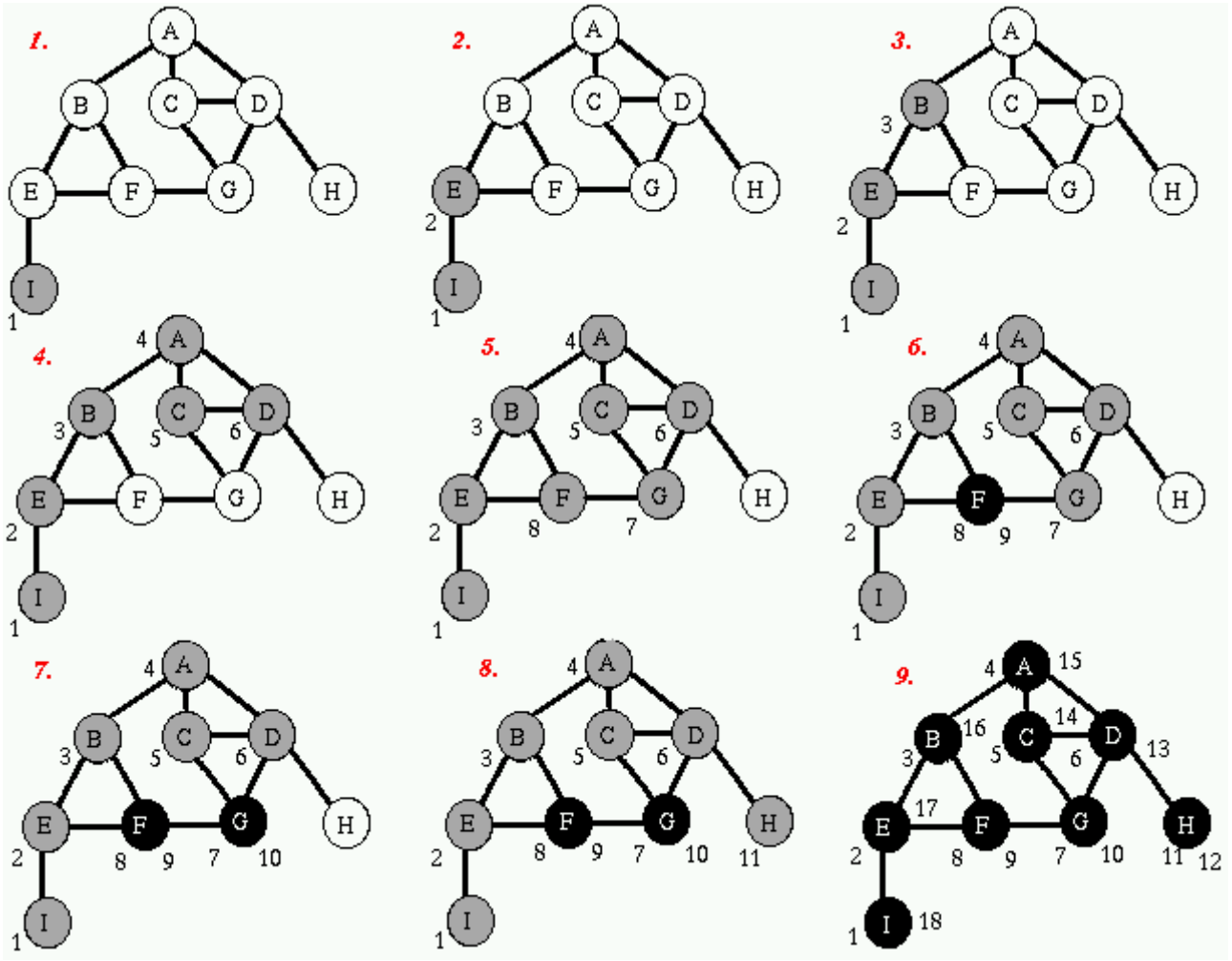
جست‌وجوی اول‌عمق به تنهایی کاربرد خاصی ندارد و در نتیجه محاسباتی که در کنار آن انجام میشود باعث اهمیت آن میشود. به طور کلی این محاسبات را میتوان به دو دسته پس‌ترتیب و پیش‌ترتیب تقسیم کرد. محاسبات پیش‌ترتیب برای هر رأسی هنگام اولین ورود به آن و محاسبات پس‌ترتیب هنگام آخرین خروج از آن انجام میشود.

ویژگی ها

جست‌وجوی اول‌عمق یال‌هایی که تشکیل دور می‌دهند را نمی‌رود؛ در نتیجه اگر یال‌های رفته شده را کنار هم بگذاریم، تشکیل یک درخت ریشه‌دار می‌دهند که به آن درخت جست‌وجوی اول‌عمق می‌گویند. همچنین زمان ورود و خروج رأس‌ها نیز ویژگی‌های منحصر به فردی دارد. مجموع این ویژگی‌هاباعث شده که این الگوریتم تبدیل به الگوریتمی مهم و کاربردی شود.

رنگ‌های الگوریتم

در طول پیمایش گراف، رأس‌ها را به طور خاصی رنگ می‌کنیم. در **ابتدای کار همه ی رأس‌ها را سفید می‌گیریم**. حال اولین زمانی که وارد هر رأس شدیم آن را خاکستری می‌کنیم و وارد رأس‌های همسایه‌اش می‌شویم. بدین ترتیب **رأس خاکستری یعنی رأسی که هنوز کار آن تمام نشده** و منتظر است تا کار بچه‌هایش تمام شود اما **رأس سفید یعنی رأسی که هنوز ملاقات نشده** است. حال هنگامی که تمام همسایه‌های یک رأس دیده شده بودند و در حال بازگشت به رأس پدر بودیم، آن راس را سیاه می‌کنیم. در نتیجه هر **رأسی که کارش تمام شود، سیاه می‌شود** پس **در آخر کار همه راس ها سیاه هستند**.



زمان ورود و خروج

زمان ورود یا شروع ($startingtime$) و **خروج** یا پایان ($finishingtime$) را به ترتیب این‌گونه تعریف می‌کنند: اولین زمان دیده شدن رأس و آخرین زمان دیده شدن رأس. یعنی زمانی که برای اولین بار وارد یک راس می‌شویم و آن را علامت گذاری می‌کنیم را زمان ورود و آخرین زمانی که از رأس خارج می‌شویم و تمام همسایه‌هایش دیده شده است و در حال بازگشت به راس پدر هستیم را زمان خروج می‌گیریم. پس اگر بخواهیم با رنگ‌ها این دو زمان را معادل کنیم، **زمان خاکستری شدن برابر زمان شروع** و **زمان سیاه شدن برابر زمان خروج** است.

در زیر سه لم در مورد این زمان‌ها آورده‌ایم که اثبات دو لم اول راحت است و لم سوم نیز با استفاده از این دولم ثابت می‌شود.

۱. لم: زمان شروع رأس v کمتر از u است اگر و تنها اگر v **جد** u باشد یا در درخت ریشه‌دار قبل از u آمده باشد.
۲. لم: زمان خاتمه رأس v کمتر از u است اگر و تنها اگر u **جد** v باشد یا در درخت ریشه‌دار قبل از u آمده باشد.
۳. لم: اگر زمان شروع رأس v کمتر از u و زمان پایان رأس u کمتر از v باشد، آنگاه v جد u است.

در نتیجه خاصیت خوبی که این درخت و این الگوریتم به ما می‌دهد این است که می‌توانیم فرض کنیم هنگامی که از یک رأس خارج می‌شویم، کار تمام زیر درخت آن تمام شده است. در نتیجه با توجه به جواب بچه‌های این رأس، جواب این راس را محاسبه می‌کنیم. برای همین معمولا در صورت نیاز به **برنامه‌ریزی پویا روی گراف از جست‌وجوی عمق‌اول استفاده می‌کنند**.

درخت ایجاد شده

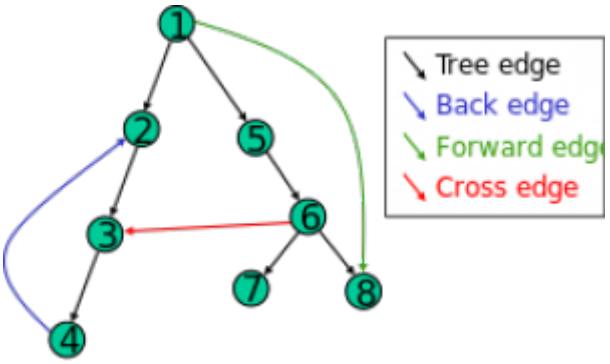
درخت جست‌وجوی عمق‌اول در واقع یک درخت ریشه‌دار است که ریشه آن همان رأسی است که جست‌وجو از آن آغاز شده است. این درخت شامل تمام یال‌هایی است که الگوریتم روی آن‌ها حرکت کرده و و پدر هر راسی، راسی است که از آن وارد این راس شده‌ایم. پس واضح است که برخی از یال‌ها در درخت نمی‌آیند و همچنین این درخت ریشه‌دار دور ندارد چون هر رأسی تنها یک پدر دارد! به طور کلی یال‌های گراف اصلی را میتوان به ۴ دسته تقسیم کرد:

۱. یال‌های درخت ساخته شده (**یال درختی** یا حرکت حرکت)
۲. از یک راس به جدش (**یال عقب‌رو** یا حرکت ــــ)
۳. از یک راس به زیر درختش (**یال جلورو** یا حرکت حرکت)
۴. هیچ کدام از سه مورد بالا (**یال میانی** یا حرکت حرکت)

در گراف‌های بدون‌جهت دسته دو و سه یکی هستند زیرا یال‌ها جهتی ندارند و یک یال جلورو حتما یک یال عقب‌رو است و برعکس. به همین ترتیب یالی از **نوع**

چهارم نیز ندارند؛ چراکه اگر پیمایش به یال میانی برمی‌خورد، وارد آن می‌شد و آن یال باید درختی می‌شد.

همچنین **در گراف های جهت‌دار یال میانی از سمت چپ درخت به سمت راست ندارید**. (یعنی یال میانی از ۴ به ۱ داریم اگر و تنها اگر زمان خاتمه ۱ زودتر از ۴ باشد)



شبه کد

۱. ابتدا همه رأس‌ها را سفید و بی‌علامت کن.
۲. رأس دل‌خواه v را به عنوان ریشه انتخاب می‌کن.
۳. رأس v را علامت بزن و خاکستری کن.
۴. کارهای پیش ترتیب روی v را انجام بده.
۵. به ازای تمام یال های w، که از راس v به u هستند کارهای را انجام بده:
۵.۱ اگر راس u علامت نخورده بود به خط 3 برو و u را به جای v فرض کن و برنامه را اجرا کن.
۵.۲ کارهای پس ترتیب روی یال ۱ را انجام بده.
۶ کارهای پس ترتیب روی v را انجام بده و این رأس را سیاه کن.
۷. کار این رأس را تمام کن و به رأس پدر بازگرد.

پیچیدگی الگوریتم

از آنجایی که به رأس حداکثر یک‌بار وارد می‌شویم در نتیجه الگوریتم هر یالی را حداکثر دوبار می‌بیند (یک‌بار به ازای هر سر یال) پس در کل زمان اجرای آن از $O(n+e)$ است. که n تعداد رأس‌ها و e تعداد یال‌ها است.

پیاده‌سازی

اگر گراف را به صورت لیست مجاورت داده باشند، کد آن به صورت زیر می‌شود.

```
#include <vector>
#include <iostream>

const int MAXN = 100 * 1000 + 10;

using namespace std;

bool mark[MAXN];
int color[MAXN];  // رنگ رأس
int start[MAXN];  // زمان شروع
int finish[MAXN];  // زمان پایان
vector <int> adj[MAXN];  // لیست مجاورت
int n;   // تعداد رأس‌ها
int m;   // تعداد یال‌ها
int now_time;  // زمان فعلی

void dfs(int v) {
    mark[v] = 1;
    color[v] = 1;         // این رأس را خاکستری کن
    start[v] = now_time++;
    // کارهای پیش‌ترتیب را انجام بده
    for(int i = 0; i < adj[v].size(); i++) {
        int u = adj[v][i];
        if(mark[u] != 1)
            dfs(u);
```

```
        // کارهای پس‌ترتیب این یال را انجام بده
    }
    color[v] = 2;        // این رأس را سیاه کن
    finish[v] = now_time;  // می‌توانید هنگام خروج هم زمان را اضافه کنید.
    // کارهای پس‌ترتیب این رأس را انجام بده
}


void input()
{
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int v, u;
        cin >> v >> u;
        adj[--v].push_back(--u);
        adj[u].push_back(v);
    }
}

int main()
{
    input();
    for (int i = 0; i < n; i++)
        if(mark[i] == 0)
            dfs(i);
}
```

مراجع

- کتاب طراحی الگوریتم با رویکردی خلاقانه (ترجمه‌ی فارسی) - بخش جست‌وجوی نخست‌ژرفا
- جست‌وجوی عمق‌اول - ویکی‌پدیا [ترجمه‌ی فارسی]