دانشگاه صنعتی اصفهان، دانشکده برق و کامپیوتر

طراحی الگوریتم ها – تکلیف سری اول گراف – پاسخ نامه

---

1. با توجه به مدل حرکت شخص در صفحه ی شطرنجی که در اختیار قرار گرفته، میتوان گفت که شخص از هر خانه نهایتاً میتواند 8 خانه ی دیگر را بازدید کند. بنابراین با یک درخت که دارای branching factor نهایتاً 8 است مواجه هستیم. ریشه ی این درخت خانه ی 0,0 است و اگر از خانه ی a به خانه ی b بتواند با یک حرکت برود، خانه ی a یک پدر برای خانه ی b خواهد بود. با توجه به اینکه یک سری از خانه‌ها ممکن است توسط آب پر شده باشد یا ... در بیشتر اوقات مقدار branch ما کمتر از 8 است. (برای مثال وقتی که m یا n برابر 0 باشد یا m = n باشد تعداد branching ما کمتر میشود)
اگر شخص از خانه ی a بتواند خانه ی b را بازدید کند، پس از خانه ی b نیز میتواند خانه ی a را بازدید کند.
پس اگر شخص در خانه ی a باشد و خانه ی b یک خانه ی مجاز باشد که بتواند با یک حرکت به آن برود، کافی است مقدار نظیر خانه ی b را یکی اضافه کنیم.
و نهایتاً مقادیر نظیر خانه‌ها را میشماریم. تعدادی از آن‌ها زوج خواهد بود و تعدادی از آن‌ها فرد خواهد بود. نهایتاً تعداد زوج ها و تعداد فرد ها را میشماریم.
دقت کنید یک سری از خانه‌ها visit نمیشود و مقدار visited آن‌ها برابر false خواهد بود. این خانه‌ها را نه جزو زوج و نه جزو فرد به حساب نباید بیاوریم.

کد مربوط به این سؤال همراه این پاسخ نامه پیوست شده است و بهتر است به آن هم نگاهی بیندازید.

2. با کمی دقت به صورت سؤال میتوان دریافت که گرافی که این سؤال تشکیل میدهد شامل دور هایی جهت دار است و یک یکسری راس و درخت دیگر که به این دور ها متصل است. بنابراین با یک پیمایش ساده میتوان طول عمیق‌ترین برگ را به دست آورد. بدین صورت که پیمایش را از یک راس i شروع کرده و به صورت dfs ادامه میدهیم و با دیدن هر راس جدید، مقدار m که نشان دهنده ی طول کنونی است را افزایش میدهیم تا نهایتاً به یک راس visited شده برسیم(دور) در این حالت میدانیم با شروع از راس i میتوانیم به حداکثر عمق m برسیم. این کار را برای تمام راس هایی که تا الان ندیده‌ایم تکرار میکنیم و نهایتاً بزرگترین m را چاپ میکنیم.
تنها باید دقت کنید که یک مسیر را چند بار پیمایش نکنید تا دچار Time Limit نشید

کد مربوط به این سؤال همراه این پاسخ نامه پیوست شده است و بهتر است به آن هم نگاهی بیندازید.

3. این سؤال با استفاده از Topological Sort میتوان حل کرد. درواقع ما باید تعداد راس هایی را بشماریم که در گراف جهت دار ما به شکل یک root عمل میکنند و در Topological Sort ما دارای اولویت بالاتری هستند.

کد مربوط به این سؤال همراه این پاسخ نامه پیوست شده است و بهتر است به آن هم نگاهی بیندازید.

4. این مسئله، مسأله ی گراف دو بخشی است.
گرافی دوبخشی است که بتوان راس های آن را به دو گروه a و b تقسیم کرد به شکلی که هیچ یالی از عناصر a به عناصر دیگر داخل a وجود نداشته باشد (یال های عناصر داخل گروه a تنها به عناصر داخل گروه b متصل اند) و همین قانون برای عناصر گروه b نیز صادق باشد.

برای بررسی دوبخشی بودن یک گراف کافی است یک dfs (یا bfs) بزنیم و اگر راس فعلی در گروه a بود، راس هایی که این این راس والد آن است را در گروه b قرار دهیم. حال اگر به یک راس رسیدیم که قبلاً visited بود (یک دور) چک میکنیم که راس فعلی که والد است و راس فرزند که قبلاً visited شده است در دو گروه مجزا باشند.
اگر این شرط رخ نداد گراف دوبخشی نیست.

کد مربوط به این سؤال همراه این پاسخ نامه پیوست شده است و بهتر است به آن هم نگاهی بیندازید.

نهایتاً چند نکته در پیاده‌سازی ها لازم بود بگم ::

1. برای گراف های بزرگ گاهی اوقات استفاده از ماتریس مجاورت باعث memory limit می‌شود و بهتر است از لیست مجاورت استفاده شود.

2. برای تابع های بازگشتی خود گراف را هیچ موقع به تابع به عنوان آرگومان ندهید و بهتر است از گراف به شکل reference و یا به صورت global استفاده کنید وگرنه برای گراف های بزرگ دچار memory limit خواهید شد.

3. در نوشتن توابع bfs و dfs دقت کنید که دچار حلقه نشوید.