گزارش پروژه‌ی چهارم درس طراحی الگوریتم‌ها (گرگ و گوسفند)

امیرعلی صادقی فرشی (۹۹۱۲۸۳۴)

توضیح الگوریتم

مسئله‌ی Edge-disjoint Paths و Max Flow و Min Cut با هم ارتباط دارند. در این مسئله اگر ظرفیت Flow را برای یال‌های موجود در گراف، یک در نظر بگیریم و سپس Max Flow را برای آن بیابیم، در صورتی که $Max\ Flow \geq 2$ باشد، می‌توانیم دو مسیر بدون یال مشترک برای گرگ و گوسفند مسئله داشته باشیم. در غیر این‌صورت چنین دو مسیری وجود ندارد.

بنابراین با استفاده از الگوریتم فورد فولکرسون، ابتدا Max Flow را می‌یابیم. با توجه به این که پیدا شدن صرفاً دو مسیر برای ما کافی است، اگر Flow به مقدار دو برسد می‌توانیم الگوریتم را متوقف کنیم. سپس با استفاده از ماتریس Residual که توسط الگوریتم فورد فولکرسون تولید می‌شود، می‌توانیم ماتریس Flow رو بسازیم و سپس می‌توانیم مسیرهایی که از خانه‌ی اول شروع می‌شود و به خانه‌ی آخر منتهی می‌شود را بیابیم.

ساختار گراف بدین‌صورت ورودی گرفته می‌شود:

۱. در سطر اول عدد n که تعداد گره‌هاست از کاربر ورودی گرفته می‌شود.

۲. در سطرهای بعدی زوج‌هایی به فرمت "u v" ورودی گرفته می‌شود که عدد اول گره متناظر با ابتدای یال و عدد دوم متناظر با انتهای یال است. گره‌ها از ۱ تا n شماره‌گذاری می‌شوند. همچنین گره شماره‌ی ۱ باید متناظر با گره شروع و گره شماره‌ی n باید متناظر با گره پایان باشد.

۳. پس از وارد کردن تمام یال‌ها، عبارت end باید وارد شود تا برنامه دیگر ورودی نگیرد.

توابع مورد استفاده در این کد در زیر توضیح داده شده‌اند:

wolf_sheep: این تابع ماتریس همسایگی مربوط به گراف را دریافت می‌کند و پس از انجام الگوریتم، در صورت عدم وجود مسیر False برمی‌گرداند و در صورت وجود، آن دو رو چاپ می‌کند.

find_path: این تابع به روش DFS در گراف Residual پیمایش می‌کند تا مسیری از شروع به پایان پیدا کند. در صورت پیدا شدن، مسیر را برمی‌گرداند و در غیر این‌صورت False برمی‌گرداند.

augment: این تابع دو ورودی دارد که عبارتند از: گراف Residual و یک مسیر. با در نظر گرفتن گراف Residual، این تابع مسیری که به‌عنوان ورودی به آن داده شده را در گراف Residual اضافه می‌کند. این تابع در واقع بخشی از الگوریتم فورد فولکرسون است. برای هر یال که در مسیر وجود دارد، یک واحد از یال Forward کم می‌کند و یک واحد به یال Backward اضافه می‌کند که چون مقادیر یال‌ها باینری هستند، به معنی False و True کردن آن‌هاست. خروجی این تابع ماتریس همسایگی گراف Residual تغییرداده‌شده است.

safe_paths: این تابع در تابع wolf_sheep زمانی فراخوانی می‌شود که مطمئن شده‌ایم که دو مسیر بدون یال مشترک وجود دارد. ورودی‌های آن گراف Residual و ماتریس همسایگی گراف است. ماتریس Flow از روی این دو بدین شکل دوباره ساخته می‌شود:

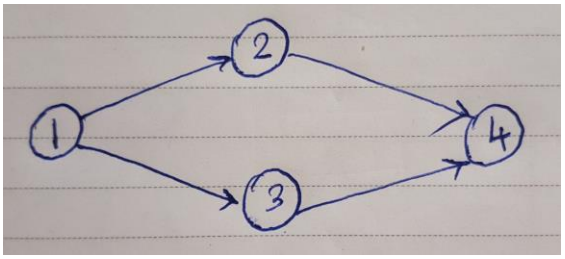
$$Flow = Residual^{\mathrm{T}}\ \&\ Adjacency$$

که در آن عبارت اول به معنی ترانهاده‌ی ماتریس Residual است و & به‌معنی عملگر AND است که درایه‌به‌درایه بر روی دو ماتریس انجام می‌شود. این ماتریس دو مسیر از شروع به پایان را چاپ می‌کند.

شبیه‌سازی

برنامه با گراف‌های زیر تست شد:

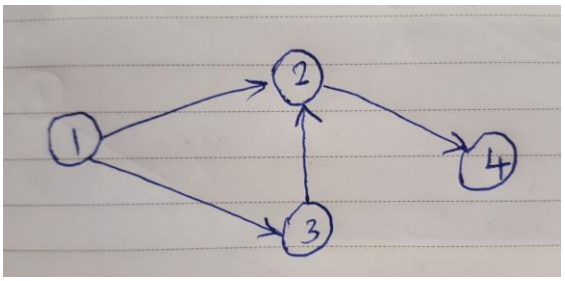
۱.




```
4
1 2
1 3
2 4
3 4
end
[1, 2, 4]
[1, 3, 4]

Process finished with exit code 0
```

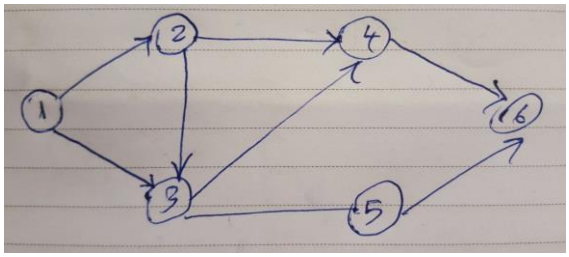
۲.




```
4
1 2
1 3
3 2
2 4
end
Sorry sheep :(

Process finished with exit code 0
```

۳. (صفحه‌ی بعد)




```
6
1 2
1 3
2 3
2 4
3 4
3 5
4 6
5 6
end
[1, 2, 4, 6]
[1, 3, 5, 6]

Process finished with exit code 0
```