به نام خدا

علی غفاری ثابت - ۹۸۲۳۹۰۳

توضیح کد:

۱- ابتدا کلاس node را تعریف می کنیم که دارای ویژگی های value، visit و neighbors است. visit یک boolean است که نشان می دهد این راس قبلا مشاهده شده است یا خیر. همچنین neighbors لیستی از node های همسایه با راس فعلی را نگه می دارد. توابع change_visit، add_neghbors و get_neghbors به ترتیب وضعیت مشاهده را تغییر می دهند، یک همسایه به راس اضافه می کند و همسایه های یک راس را دریافت می کند. همچنین توابع visited و get_value به ترتیب وضعیت مشاهده شدن و مقدار راس را بر می گردانند.

۲- کلاس graph نیز شامل مقدار تعداد راس های گراف و لیستی از راس های گراف است. در این کلاس دو تابع add_edge و get_node به ترتیب یک یال به گراف اضافه می کند و تعداد راس های گراف را بر می گردانند.

۳- در بدنه اصلی کد تعداد تست ها و در هر تست شماره تقاطع مقصد و یال های را از کاربر دریافت می کنیم. سپس برای هر تست تابع pathes را صدا می زنیم.

۴- در تابع paths گراف نقشه، راس فعلی، مسیر فعلی و شماره راس مقصد را دریافت می کنیم سپس اگر مقدار گره فعلی همان مقدار گره مقصد بود مسیر ذخیره شده در path را چاپ می کنیم. سپس اگر مقدار تابع promising برای راس فعلی True بود وضعیت مشاهده شدن راس را تغییر داده و آن را به لیست path اضافه می کنیم و برای تمام همسایه های آن راس مجددا تابع paths را صدا می زنیم. بعد از صدا زدن paths برای تمام همسایه های راس فعلی مقدار راس فعلی را از لیست path خارج کرده و وضعیت مشاهده شدن را تغییر می دهیم تا به حالت اول خود برگردند.

۵- تابع promising نیز گراف نقشه و راس فعلی را دریافت کرده و بررسی می کند که قبلا مشاهده شده است یا خیر تا از تشکیل حلقه جلوگیری شود. (اگر راس فعلی مشاهده نشده باشد مقدار True برگردانده می شود به این معنی که این راس با رئوس قبلی مشاهده شده تشکیل حلقه نمی دهد.)

پیچیدگی زمانی:

تعداد گره های امید بخش برای یک گراف کامل با ۱۰ راس و برای رفتن از راس ۱ به راس ۱۰ به صورت تقریبی و با روش مونته کارلو برابر 1019083 است. کدی که به کمک آن این عدد را تخمین زدیم در کنار فایل اصلی قرار دارد.

تصویر محیط اجرا:

```
ali@Ali:~/Desktop/University/3992/algorithm_design/projects/project3/individual-project-3-aghs8055/firefighters$ python3 firefighters.py
Enter count of cases: 2
Enter value of destination: 6
Enter edges:
1 2
1 3
3 4
3 5
4 6
5 6
2 3
2 4
0 0
Case #1:
1-2-3-4-6
1-2-3-5-6
1-2-4-3-5-6
1-2-4-6
1-3-4-6
1-3-5-6
1-3-2-4-6
Enter value of destination: 4
Enter edges:
2 3
3 4
5 1
1 6
7 8
8 9
2 5
5 7
3 1
1 8
4 6
6 9
0 0
Case #2:
1-5-2-3-4
1-5-7-8-9-6-4
1-6-4
1-6-9-8-7-5-2-3-4
1-3-2-5-7-8-9-6-4
1-3-4
1-8-7-5-2-3-4
1-8-9-6-4
ali@Ali:~/Desktop/University/3992/algorithm_design/projects/project3/individual-project-3-aghs8055/firefighters$
```