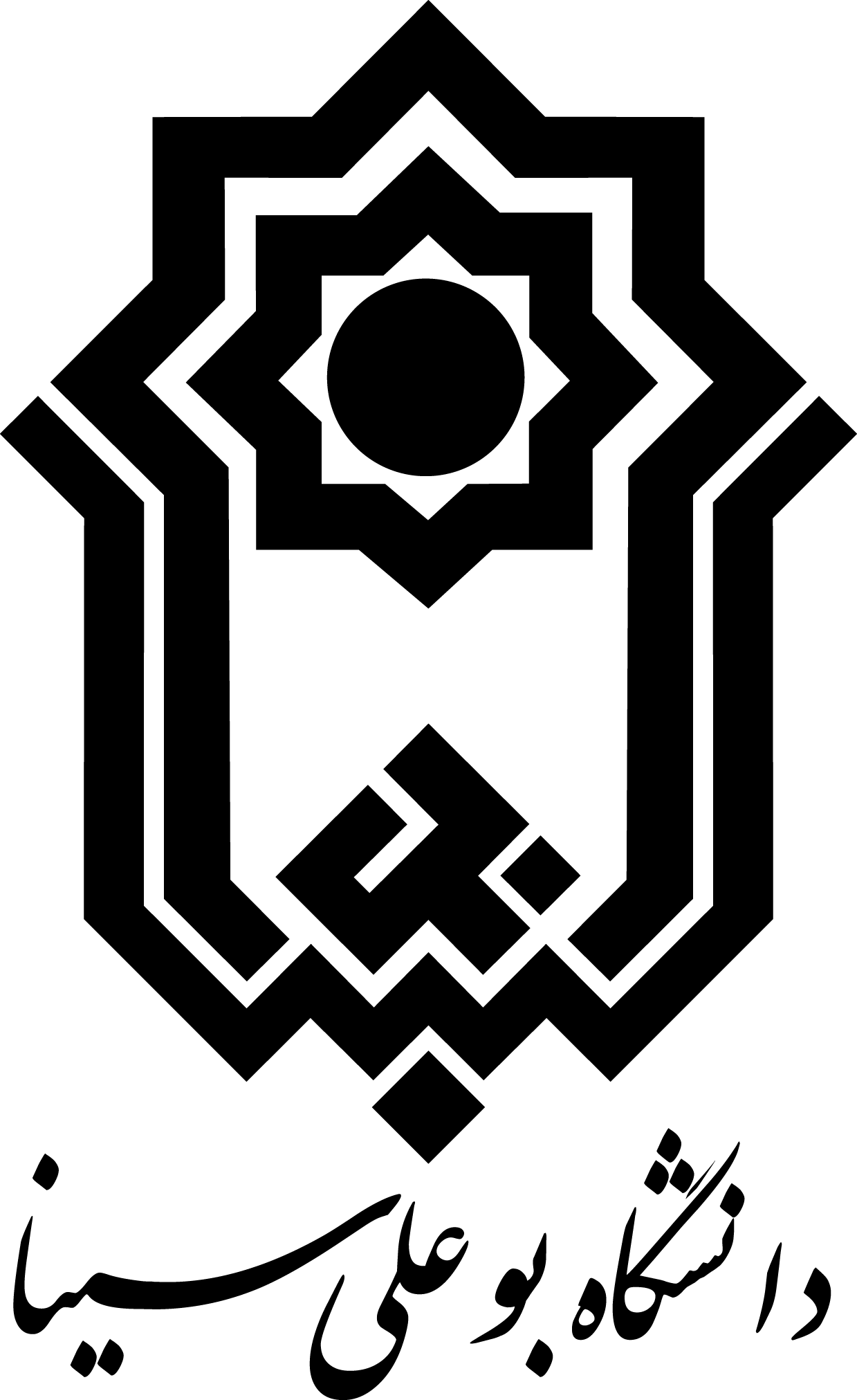
به نام خدا



نام و نام خانوادگی:سید فرهاد حسینی

شماره دانشجویی : 9612358016

نام درس : طراحی الگوریتم

استاد مربوطه : دکتر منصوری زاده

# پروژه انتخابی :

## حل مساله max flow با الگوریتم dfs و dijkstra

چون پیاده سازی با الگوریتم dijksrta در یکی از تمرینات انجام شده پیاده سازی این مساله با الگوریتمی مشابه الگوریتم پریم هم قابل انجام است .

# فایل های پروژه :

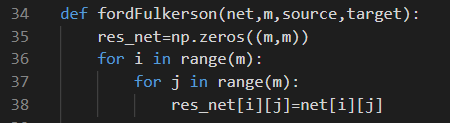
این پروژه دارای سه بخش میباشد که باید مساله شار بیشینه را با کمک الگوریتم های دایکسترا ، پریم و پیمایش عمقی پیاده سازی کنیم . پیاده سازی این الگوریتم ها در فایل های dijkstra.py و prim.py و dfs.py میباشد. علاوه براینها چون برنامه ورودی را از فایل میخواند چند فایل متنی هم در کنار کد ها قرار داده ایم .

# معرفی الگوریتم فورد فالکرسون :

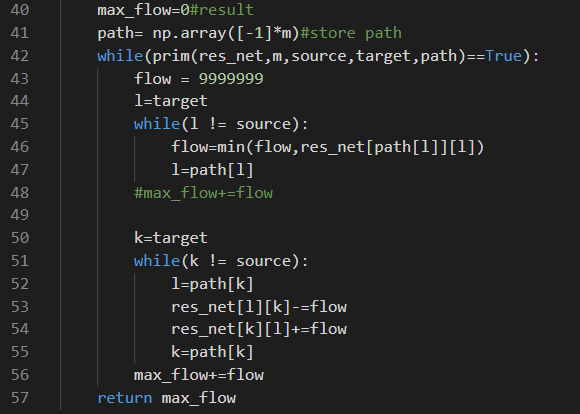
این الگوریتم برای پیدا کردن شار بیشینه از مبدا به مقصد استفاده میشود . این الگوریتم هر بار یک مسیر از مبدا به مقصد پیدا میکند و میزان گلوگاه آن مسیر (کمترین وزن موجود در مسیر ) را از کل مسیر کم میکند . این عمل آنقدر تکرار میشود تا دیگر مسیری از مبدا به مقصد وجود نداشته باشد (عملا cut رخ داده باشد ).

در مرحله ی پیداکردن مسیر از مبدا به مقصد از الگوریتم های مختلفی میشود استفاده کرد که در این پروژه سه تا از آنها را بررسی میکنیم .

# بررسی کد الگوریتم فورد فالکرسون :



همانطور که در خط 34 مشاهده میشود این تابع کل شبکه را بعنوان ورودی میگیرد . به همراه تعداد گره ها ، گره مبدا و همچنین گره مقصد . در خطوط 35 تا 38 یک ماتریس تشکیل میشود و ماتریس شبکه در آن ریخته میشود . این شبکه جدید همان شبکه ای است که هر بار مقدار ظرفیت باقیمانده از یالها را نشان میدهد و تغییرات روی این شبکه انجام میشود .



در خط 40 یک متغیر تعریف شده که پاسخ نهایی در آن ریخته میشود . مقدار اولیه آن 0 است چون فعلا جریانی از مبدا به مقصد نداریم .

یک آرایه به نام path تعریف کرده ایم که وقتی شروع به پیدا کردن مسیر میکنیم مسیر در آن ذخیره میشود واین کمک میکند که راه را گم نکنیم . خط شماره 42 نقطه ی اصلی این پروژه است که میگوید اگر مسیری از مبدا به مقصد وجود دارد اعمال زیر را انجام بده .

یک متغیر به نام flow داریم که وظیفه ی ذخیره کردن گلوگاه مسیر را دارد . l هم یک متغیر کمکی است که با آن از مقصد به مبدا برمیگردیم . در خطوط 45 تا 47 گلوگاه مسیر پیدا میشود .

در خطوط 50 تا 55 به کمک دو متغیر کمکی lوk مسیر مبدا به مقصد طی میشود و میزان گلوگاه از هر یال کم میشود . (از یال ها کم میشود و به یال های معکوس اضافه میشود )

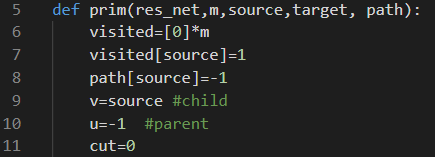
در خط 56 مقدار جریان این مسیر به مقدار جریان هایی که تا الان حساب کردیم اضافه میشود .

زمانی که دیگر مسیری از مبدا به مقصد وجود نداشت یعنی جریان بیشینه پیدا شده و آنرا برمیگردانیم.

# بررسی کد الگوریتم پریم :

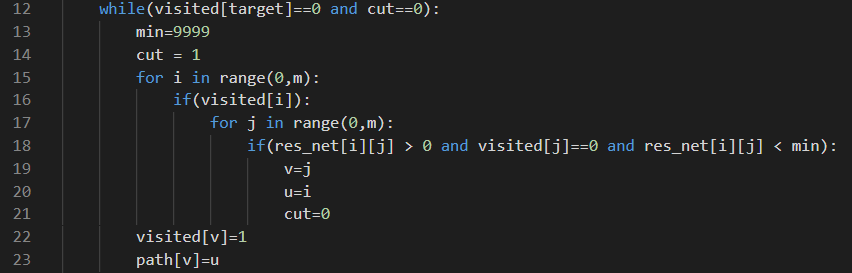
الگوریتم پریم برای بدست آوردن درخت پوشای کمینه استفاده میشود . در مساله ی شار بیشینه زمانی که میخواهیم مسیری از مبدا به مقصد پیدا کنیم میتوانیم از این الگوریتم کمک بگیریم . مرتبه زمانی این الگوریتم O(n^2) میباشد زیرا از هر راس تمام یالهای خروجی را بررسی میکند . n راس داریم و در بدترین حالت اگر گراف کامل باشد n یال خروجی هم داریم .

# بررسی کد الگوریتم پریم :



ورودی این الگوریتم همانند الگوریتم های دیگر مسیر یابی ،شبکه، تعداد گره ها ، مبدا ، مقصد و یک آرایه که برای ذخیره کردن مسیر از آن استفاده میشود .

در ابتدا فرض میکنیم که هیچ راسی دیده نشده . راس مبدا را وارد مجموعه دیده شده ها میکنیم و چون از گره ای به مبدا نیامدیم آنرا 1- میدهیم . دو متغیر کمکی v و u را میگیریم که بتوانیم گراف را طی کنیم .یک متغیر هم به نام cut تعریف میکنیم که نشاندهنده این است که اگر cut رخ داده دیگر ادامه ندهیم .



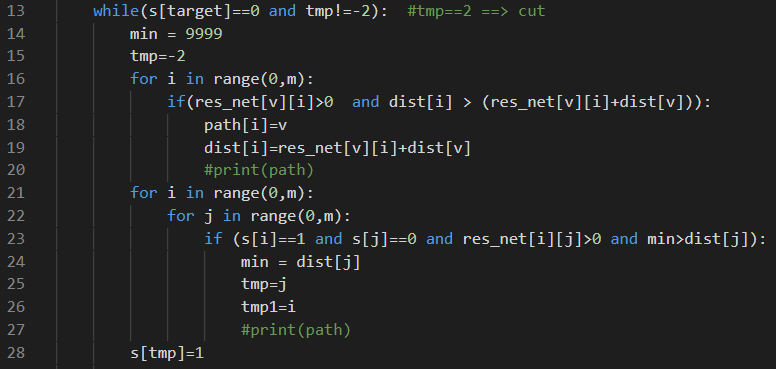
تا زمانی که به هدف نرسیدیم و هنوز cut ای رخ نداده اعمال زیر را انجام بده .

چون در الگوریتم پریم باید هر بار کمترین یال را پیدا کنیم متغید min را تعریف میکنیم . در این قسمت cut را فعال میکنیم تا اگر وارد قسمت بعدی کد نشدیم بفهمیم که تکه شدن رخ داده .در خطوط 15 تا 21 هم کمترین یالی که راس آنرا تابحال ندیدیم پیدا میشود و آن راس در خط 22 وارد مجموعه دیده شده ها میشود و مسیر منتهی به آن در خط 23 ثبت میشود.

# معرفی الگوریتم دایکسترا:

الگوریتم دایکسترا برای بدست آوردن کوناه ترین مسیر از یک راس به بقیه ی راسها میباشد و چون یکی از رئوس گره مقصد میباشد پس این الگوریتم کمک کننده است . مرتبه زمانی این الگوریتم O(n^2) میباشد .

# بررسی کد الگوریتم دایکسترا :



در خط شماره 13 میگوید که اگر هنوز به هدف نرسیده ایم مراحل زیر را انجام بده . ( قسمت دوم شرط همان وجود یا عدم وجود cut میباشد ).

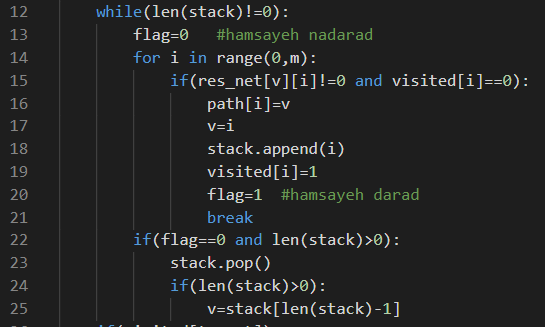
در خط شماره 17 شرطی وجود دارد که میگوید اگر از گره فعلی به گره ای راه است (همه ی گره ها را چک میکند) و فاصله اش از این مسیر از فاصله ای که قبلا تا مبدا برایش پیدا کرده بودیم کمتر است باید این مسیر و فاصله را جایگزین قبلی کند .

در خطوط 21 تا 27 هم گره های مجاور به گره های مجموعه ی s را چک میکند و آن گره ای که کمترین مسافت از مبدا به آن است را وارد مجموعه ی s میکند .

# معرفی الگوریتم پیمایش عمقی:

یکی از الگوریتم های پیمایش گراف پیمایش اول عمق میباشد . در پیاده سازی این الگوریتم از پشته استفاده میشود . روش کار به این صورت است است که هر گره را که دیدیم وارد پشته میکنیم و به یک گره دیده نشده در همسایه اش میرویم و همین کار را ادامه میدهیم و زمانی گره ای را از پشته در می آوریم که هیچ همسایه ی دیده نشده ای نداشته باشد . چون ای الگوریتم به شکل متصل به هم گره ها را پیمایش میکند برای این مساله مناسب است . مرتبه زمانی این الگوریتم O(n+e) میباشد چون هر گره را پیمایش میکنیم و سپس اولین مسیر خروجی را وارد میشویم .

# بررسی کد الگوریتم پیمایش عمقی :



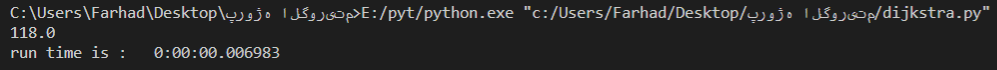
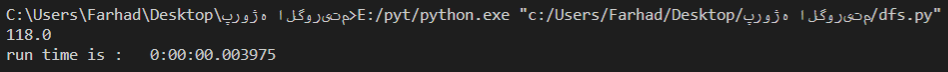
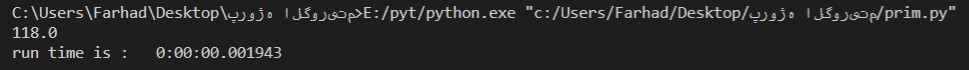
در خط 12 ام میگوییم تازمانی که استک خالی نشده مراحل زیر را ادامه بده .( اگر استک خالی شود یعنی تمام همسایه های مبدا بررسی شدند و برای خارج کردن هر گره از استاک باید همه ی همسایگانش بررسی شده باشند . )

در خطوط 14 تا 21 از گره فعلی اولین همسایه ای را که دیده نشده را پیدا میکنیم و وارد آن میشویم وآن را وارد مجموعه دیده شده ها میکنیم و در استک قرار میدهیم . سپس در خط 22 بررسی میکنیم که اگر یک گره همسایه ی دیده نشده ای ندارد آنرا از استک خارج میکنیم .

این کارها تا زمانی ادامه میابد که کل استک خالی شود .

# مقایسه ی زمان اجرای الگوریتم ها بصورت عملی :

الگوریتم های دایکسترا ، پیمایش سطحی ، پیمایش عمقی و پریم بر روی ورودی های یکسان اجرا شدند ونتایج زیر بدست آمد .



همانطور که مشاهده میشود مقایسه ی مدت زمان اجرای الگوریتم فورد فالکرسون به کمک چهار الگوریتم بصورت زیر است .

پیمایش سطحی < پریم < پیمایش عمقی < دایکسترا

# منابع و مراجع :

در رابطه با فهمیدن منطق الگوریتم ها از سایت [www.codingame.com](http://www.codingame.com) و همچنین سایت [www.geeksforgeeks.org](http://www.geeksforgeeks.org) استفاده شد .