تمرین برنامه نویسی سوم

NETWORK ROUTING

سید محمد امین اطیابی (۸۱۰۱۹۸۵۵۹)

مرتضی نوری (۸۱۰۱۹۸۴۸۱)

نحوه اجرا

با وارد کردن دستور زیر در پوشه اصلی پروژه ، فایل اجرایی در محل فعلی ایجاد می‌شود .

```
make
```

با وارد کردن دستور زیر برنامه شروع به کار می‌کند و می‌توان دستورات برنامه را وارد نمود .

```
./main.out
```

برنامه به صورت کاملا پویا پیاده سازی شده است و محدودیتی برای تعداد لینک‌ها وجود ندارد .

ساختار

کدهای برنامه‌ها در پوشه Source قرار گرفته است . این پوشه دارای سه کلاس است:

- Network : تمام کارهای شبکه ما در این کلاس صورت می‌گیرد . هماهنگی میان لینک‌ها و اجرای الگوریتم‌های مسیریابی توسط این کلاس انجام می‌شود
- Node : اطلاعات مورد نیاز گره‌های شبکه در این کلاس ذخیره می‌شوند . اطلاعاتی اعم از مسیر به به نودهای دیگر
- CommandParser : این کلاس وظیفه پردازش دستورات ورودی را دارد . اطلاعات متنی وارد شده در خط فرمان ، به اطلاعات قابل فهم برای دیگر بخش‌های برنامه تبدیل شده و به آن‌ها تحویل می‌شود

شرح کد و عملکرد آن

کلاس Node

این کلاس صرفا وظیفه ذخیره داده‌ها را بر عهده دارد .

- void init(int count) : این متد در زمان شروع اجرا الگوریتم‌های مسیریابی صدا زده می‌شود و به تعداد نودهای شبکه ، فضای ذخیره سازی مقدار دهی اولیه می‌کند .
- void clear() : این متد داده‌ها ذخیره شده را پاک می‌کند تا برای اجرا بعدی آماده باشند .

کلاس CommanParser

این کلاس دستورات ورودی را پردازش می‌کند .

متد parse با دریافت کل دستور ورودی تشخیص می‌دهد چه دستوری وارد شده است . پس از تشخیص دستور ، آرگومان‌های دستور را به متد مربوط به هر کدام از دستورات می‌فرستد تا آن‌ها را تجزیه کنند .

کلاس Network

متدهای addLink ، modifyLink و removeLink توپولوژی شبکه‌ها را تغییر می‌دهند و لینکی به شبکه ما اضافه یا کم می‌کنند .

متد show توپولوژی شبکه ما را نمایش می‌دهد .

متدهای lsrp و dvrp به کمک یکسری متدها کمکی دیگر عملیات مسیریابی را بر عهده دارند .

پیاده سازی الگوریتم‌های Dijkstra و Bellman-Ford به صورت پویا (Dynamic) انجام شده است .

فایل main

در تابع اصلی این فایل پس ورودی از کاربر دریافت شده و به کلاس CommandParser تحویل داده می‌شود .

تحلیل نتایج

نتیجه اجرا دو الگوریتم مسیریابی گفته شده بر روی توپولوژی داده شده کنار فایل‌های ارسال موجود است که حال به بررسی آن‌ها می‌پردازیم .

زمان اندازه‌گیری شده بدون در نظر گرفتن زمان عملیات I/O بوده و بعد از اتمام الگوریتم به ازای هر نود است.

با توجه به اینکه می‌دانیم الگوریتم Dijkstra عملکرد بهتری از الگوریتم Bellman-Ford دارد ، باید توقع زمان اجرا کمتری برای این الگوریتم داشته باشیم و این در نتایج قابل مشاهده است . البته پروتوکول مسیریابی Link-State می‌توانست عملکرد بهتری نیز داشته باشد که در این پیاده سازی از آن صرف نظر شده است . (کدهای روش دیگر موجود است و در صورت نیاز می‌توان با روش دیگر آن‌ها را اجرا نمود)

الگوریتم Bellman-Ford امکان تشخیص یال با وزن منفی را نیز دارد و در حالت کلی جامع‌تر است . برای همین کمی مرتبه زمانی بیشتری از الگوریتم Dijkstra دارد .