به نام خدا

تمرین چهارم — سیستم‌های توزیع‌شده

مدرس: صابر صالح — ۲ـ۱۳۹۷

نکات مهم

لطفا ابتدا به نکات زیر توجه کنید:

- برای پیاده‌سازی این تمرین از زبان Python استفاده نمایید.
- پاسخ تمرین را فقط در Quera آپلود کنید.
- به شیوه‌ی ورودی و خروجی دقت کنید.
- مهلت ارسال تمرین تا پایان روز ۱۳۹۸/۳/۲۸ می‌باشد.

موفق باشید.

۱ سوال ۱

در این تمرین، کد spark ای بنویسید که به وسیله‌ی آن بتوان در یک شبکه اجتماعی به کابران پیشنهادهایی برای ایجاد دوستی [1] بدهد. برای اینکار الگوریتم باید دو نفری را که تعداد زیادی دوستان مشترک دارند به هم پیشنهاد دهد.

۱.۱ داده‌ها

- داده ها در فایل q1_data.txt قرار دارند.
- هر خط از این فایل به شکل <user> <TAB> <friends> است. در ابتدای هر خط نام کاربر (<user>) قرار دارد. سپس با یک فاصله، اسامی دوستان (<friends>) این شخص پشت سر هم قرار دارد که با یک ویرگول جدا شده است. دقت کنید که در این داده ها دوستی ها دو طرفه است یعنی یال های گراف کاربران جهت دار نیست.

۲.۱ توضیحات بیشتر

- برای پیاده سازی این الگوریتم برای هر کاربر ۱۰ شخص را به عنوان پیشنهاد دوستی مشخص کنید که در حال حاضر با هم دوست نیستند و بیشترین تعداد دوستان مشترک را با هم دارند.
- اگر شخصی کمتر از ۱۰ دوست درجه ۲ داشت (یعنی مجموعه دوستان دوستانش به ۱۰ نفر نمی‌رسید) همه‌ی آن ها را نمایش دهید (به صورت نزولی از تعداد دوست مشترک بیشتر به کمتر). اگر شخصی هیچ دوستی نداشت لیست خالی قرار دهید.
- به صورت کوتاه توضیح دهید که چگونه الگوریتم خود را پیاده کردید.

[1] Friendship recommendation

◄ برای تست کد خود، جواب خروجی برای کاربر شماره ۱۱ به شکل زیر است.

```
ID:11_ 27552,27667,27620,33192,27600,32072,27573,7785,27589,27617
```

دقت کنید که خروجی برای هر کاربر به شکل نمونه آورده شده باشد. یعنی به شکل

```
ID:<user>_ <recommendation>}
```

◄ دقت کنید که کدها دارای خطوط خالی و یا جدا شدن بدون <TAB> هستند، در هنگام پردازش داده‌ها شروطی را قرار دهید که این داده ها پردازش نشوند.

۲ PageRank

PageRank (PR) الگوریتمی می‌باشد که توسط گوگل سرچ استفاده می‌شود تا در search engine خود به webpage ها rank بدهد. این الگوریتم یک روش برای اندازه گیری اهمیت website ها می‌باشد. اساس کار PageRank بوسیله شمردن تعداد و نیز کیفیت لینک‌ها به یک page می‌باشد که بر این اساس تشخیص می‌دهد که یک website به چه اندازه مهم است. در این روش فرض می‌شود که website های مهم احتمالاً لینک‌های بیشتری از بقیه سایت‌ها دریافت می‌کنند.

۱.۲ الگوریتم

شما الگوریتم خود را بدینگونه شروع می‌کنید که در ابتدا برای rank هر page یک مقدار اولیه در نظر می‌گیرید. این مقدار اولیه برابر $\frac{1}{n}$ می‌باشد که در این رابطه n بیان کننده تعداد node ها در گراف می‌باشد. سپس برای هر وب سایت x شما باید PageRank آن را به صورت iterative محاسبه نمایید که از فرمول زیر بدست می‌آید:

$$PR(x) = \frac{1-d}{N} + d \sum_{y \to x} \frac{PR(y)}{out(y)} \quad (۱)$$

در فرمول بالا d یک عدد بین 0 و 1 می‌باشد (که بصورت معمول مقدار آن را 0.85 در نظر می‌گیرند). همچنین $\mathrm{out(y)}$ بیان کننده تعداد یال‌هایی می‌باشد که از راس y خارج شده‌اند.

حال شما باید این‌کار را برای همه node ها در گراف انجام بدهید و این‌کار را باید به تعداد $iteration$ ها که یکی از ورودی‌های الگوریتم است تکرار کنید.

۲.۲ Pseudo-code

Algorithm 1 PageRank Algorithm

procedure PageRank(G, *iteration*) ▷ G: inlink file, *iteration*: # of iteration
  $d \leftarrow 0.85$ ▷ damping factor: 0.85
  $oh \leftarrow G$ ▷ get outlink count hash from G
  $ih \leftarrow G$ ▷ get inlink hash from G
  $N \leftarrow G$ ▷ get # of pages from G
  for all p in the graph **do**
    $opg[p] \leftarrow \frac{1}{N}$ ▷ initialize PageRank
  while $iteration > 0$ **do**
    $dp \leftarrow 0$
    for all p that has no out-links **do**
      $dp \leftarrow dp + d \times \frac{opg[p]}{N}$ ▷ get PageRank from pages without outlink
    for all p in the graph **do**
      $npg[p] \leftarrow dp + \frac{1-d}{N}$ ▷ get PageRank from random jump
      for all ip in $ih[p]$ **do**
        $npg[p] \leftarrow npg[p] + \frac{d \times opg[ip]}{oh[ip]}$ ▷ get PageRank from inlink
    $opg \leftarrow npg$ ▷ update PageRank
    $iteration \leftarrow iteration - 1$

۳ سوال ۳ (امتیازی)

۱.۳ مقدمه

در این سوال قصد داریم الگوریتم MST (Minimum Spanning Tree) را پیاده کنیم. هدف این است که برای یک گراف بدون جهت، درختی را پیدا کنیم که تمام گره های گراف را شامل شود و مجموع وزن یال های این درخت کمترین مقدار بین تمام درخت های ممکن با این دو ویژگی باشد.

۲.۳ MST

در اینجا فرض اولیه این است که هر گره وزن یال های مجاور خود را می‌داند. همچنین هر گره دارای یک <uid> است و هیچ دو وزن یالی مانند هم نیست. در نتیجه درخت خروجی برای این الگوریتم درختی منحصر به فرد است. پایان الگوریتم زمانیست که هر گره تمام یال های متصل به خود را در دو گروه "یال های جزو درخت" و "یال های خارج درخت" قرار داده باشد. در ادامه توصیه می‌شود که بخش 15.5 از کتاب مرجع را به دقت مطالعه کنید.

۳.۳ گروه بندی یال‌ها

در هر گره، ابتدا تمام یال های مجاورش را به صورت basic تعریف می‌کنید. زمانی که مشخص شود این یال جزو MST نیست، این یال را به گروه rejected که همان گروه "یال جزو درخت نیست" اضافه می‌کنیم. همچنین این یال را از گروه basic حذف می‌کنیم. زمانی که مشخص شد یالی جزو درخت MST است، آن را به گروه branch اضافه می‌کنیم و از گروه basic حذف می‌کنیم. در پایان اجرای الگوریتم، هر گره باید تمام یال های موجود در گروه basic را به یکی از گروه های rejected یا branch تخصیص داده باشد.

۴.۳ پیام ها

پیام های زیر را تعریف می‌کنیم. علاوه بر پیام‌های زیر می‌توانید در صورت نیاز یک یا دو پیام دیگر با کاربردی که خود تعیین می‌کنید تعریف کنید.

۱.۴.۳ initiate

این پیام توسط رهبر هر کامپوننت فرستاده می‌شود. این پیام به گره‌ها می‌گویند که عمل پیدا کردن MWOE [۲] خود را شروع کنند. همچنین مقادیر core و level نیز توسط این پیام به تمام اعضای کامپوننت فرستاده می‌شود.

```
{"uid": <node_id>, "type": "initiate", "value": <(core, level)>}
```

۲.۴.۳ report

بعد از اینکه هر گره MWOE خود و تمام descendant های خود در کامپوننت را فهمید، مقدار کمینه این مقادیر به همراه راهی که این پیام طی می‌کند را به گره parent خود (گره‌ای که پیام initiate را داده است) می‌فرستد. به طور مثال اگر گره ۵ بعد از دریافت پیام report از تمام descendant هایش متوجه شد که گره ۶ کمترین مقدار MWOE را داراست، در صورتی که پیام initiate از گره ۴ به گره ۵ رسیده باشد، گره ۵ پیام report را با مقادیر path = "6 5" و mwoe=$MOWE_6$ به گره ۴ می‌فرستد.

```
{"uid": <node_id>, "type": "report", "value": <(path, mwoe)>}
```

ساختار متغیر path را به دلخواه تعیین کنید. اما دقت کنید که مسیری که از رهبر به گره دارای

[2] Minimum Weight Outgoing Edge

MWOE است باید مشخص باشد و پیام change_root نباید به صورت broadcast فرستاده شود.

test ۳.۴.۳

زمانی که گره i دنبال mwoe خود می‌گردد، به تمام یال های موجود در گروه basic خود پیام test را می‌فرستد. جواب این پیام مشخص می‌کند که هر گره j در طرف دیگر یال با گره i در یک کامپوننت است یا خیر.

```
{"uid": <node_id>, "type": "test", "value": <(core, mwoe)>}
```

accept ۴.۴.۳

در صورتی که گره i با لول بزرگتر یا مساوی گره j، پیام test را از گره j دریافت کند، در صورتی که در یک کامپوننت نبودند (مقدار level و core متفاوتی داشتند) این پیام را به گره j ارسال می‌کند.

```
{"uid": <node_id>, "type": "accept", "value": <(core, level)>}
```

reject ۵.۴.۳

زمانی که گره i پیام تست را از گره j که در یک کامپوننت با گره i است دریافت می‌کند این پیام را به گره j می‌فرستد.

```
{"uid": <node_id>, "type": "reject", "value": None}
```

change_root ۶.۴.۳

زمانی که گره رهبر مقدار MWOE کل کامپوننت را تعیین کرد، در صورتی که این مقدار یال مجاور خودش نباشد، این پیام را با استفاده از مقدار path دریافتی توسط پیام report به گره دارای MWOE

می‌فرستد. این پیام به گره دارای MWOE می‌گوید که به کامپوننت مجاور MWOE وصل شود.

```
{"uid": <node_id>, "type": "change_root", "value": path}
```

connect ۷.۴.۳

این پیام توسط گره متصل به یال MWOE به گره طرف دیگر این یال فرستاده می‌شود. در صورتی که پیام connect از هر دو سر این یال بگذرد عمل merge اتفاق می‌افتد. در غیر این صورت عمل absorb اتفاق می‌افتد.

```
{"uid": <node_id>, "type": "change_root", "value": <(core,level)>}
```

end ۸.۴.۳

زمانی که گره رهبر مقدار MWOE را بی‌نهایت تشخیص داد، این پیام را از طریق درخت MST به تمام گره های دیگر Broadcast می‌کند. هر گره بعد از Broadcast کردن پیام end توسط خودش، پایان برنامه خودش را اعلام می‌کند و می ایستد. بعد از دریافت پیام end توسط تمام گره ها، الگوریتم پایان می‌یابد.

```
{"uid": <node_id>, "type": "end", "value": None}
```

۵.۳ حالت های هر گره

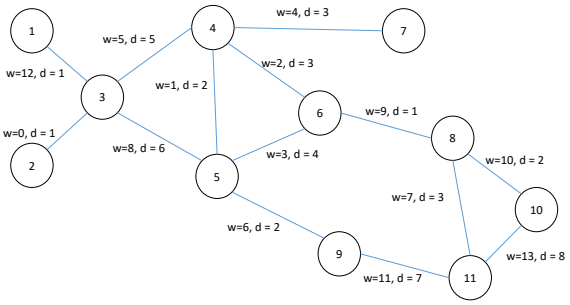
برای راحتی کار برای هر گره ۳ حالت را تعریف می‌کنیم: sleep ، search و found . نحوه‌ی تغییر بین این حالت ها را به دلخواه خود تعریف کنید. اما به طور کلی زمانی که یک گره در حال یافتن MWOE است در حالت search و زمانی که پیام report را به گره parent خود می‌فرستد، به حالت found می‌رود. همچنین در ابتدا همه‌ی گره ها در حالت sleep قرار دارند.

۶.۳ ورودی و خروجی

بخشی از فایل وردی که در input.txt قرار دارد، در اینجا آورده شده است. گرافی که در این فایل قرار دارد همان گرافی است که در اسلاید های آموزشی شما موجود است. هر خط از ورودی به شکل زیر است. دقت شود که گراف بدون جهت است و هر دو گره دو سر هر یال باید وزن و تاخیر یال را بدانند.

```
{<head_node> <tail_node> <weight> <delay>}
```

```
1 3 12 1
2 3 0 1
3 4 5 5
3 5 8 6
```



۷.۳ نکات نهایی

سعی کنید که قبل از پیاده سازی به طور کامل الگوریتم را بفهمید، حالت های مختلف را بررسی کنید. کد را به صورت کلاس بندی شده بزنید و سعی کنید در زمان نوشتن برای هر تابع توضیحات مناسبی بنویسید. همراه با کد گزارشی را نیز بنویسید و تمام حالت های خاص که ممکن است اتفاق بیفتد را در آن ذکر کنید. همچنین مشکلاتی که در حین پیاده سازی با آن مواجه شده‌اید را نیز در آن ذکر کنید. پیشنهاد می‌شود که برای خوانایی کد ماژول pylint را نصب و نحوه‌ی استفاده از آن را مشاهده کنید. این ماژول مشکلات کدی که زدید مانند نبود کامنت یا متغیرها با نامگذاری اشتباه را به شما می‌گوید. همچنین پیشنهاد می‌شود این لینک را نیز مشاهده کنید. علت تاکید بر خوانایی کد به این دلیل است که حجم کد ممکن است زیاد شود و برای رفع باگ به مشکل بر بخورید.