به نام خدا

تمرین چهارم — سیستم‌های توزیع‌شده

مدرس: صابر صالح — ۲ـ۱۳۹۸

نکات مهم

لطفا ابتدا به نکات زیر توجه کنید:

- برای پیاده‌سازی این تمرین از Spark و زبان Python3 استفاده نمایید.
- پاسخ تمرین را فقط در Quera آپلود کنید.
- به شیوه‌ی اجرای برنامه دقت کنید.
- مهلت ارسال تمرین تا پایان روز ۱۳۹۹/۳/۲۳ می‌باشد.

موفق باشید.

۱ مسئله رتبه‌بندی

یک گراف جهت‌دار که نمایش دهنده شبکه‌ای از وب‌سایت‌ها است به ما داده شده‌است. هدف ما یافتن یک متریک خوب برای اندازه‌گیری اهمیت هر راس در گراف است که با استفاده از آن بتوانیم رتبه‌بندی‌ای از وب‌سایت‌ها به دست آوریم.

در مقالات آکادمیک، یکی از متریک‌هایی که برای محاسبه اهمیت یک مقاله به کار می‌برند، تعداد ارجاعاتی است که دریافت کرده‌است. ما نیز می‌توانیم از یک روش مشابه در رتبه‌بندی وب‌سایت‌ها استفاده کنیم و اهمیت یک وب‌سایت را با تعداد وب‌سایت‌هایی که به آن پیوند دارند، اندازه‌گیری کنیم. البته در حالت ایده‌آل ترجیح می‌دهیم اهمیتی که یک وب‌سایت از پیوند شدن توسط وب‌سایت دیگر دریافت می‌کند، با اهمیت وب‌سایت پیوند دهنده و کیفیت پیوند (بزرگی فونت یا محل قرارگیری) نیز ارتباط داشته‌باشد. برای سادگی فرض می‌کنیم کیفیت پیوند را می‌توانیم با یک عدد از 0.1 تا 10 با دقت یک دهم نمایش دهیم و این عدد را وزن یال متناظر با پیوند در نظر می‌گیریم.

در ابتدا برای این که وب‌سایت‌ها نتوانند با پیوند دادن به خود، اهمیت خود را بالا ببرند تمام طوقه‌ها را حذف می‌کنیم.

حال می‌توانیم برای دقیق‌تر کردن شهود مسئله، از Random Walk استفاده کنیم. کاربری را در نظر می‌گیریم که در حال مرور وب‌سایت‌ها است و با توجه به کیفیت پیوندها بر روی آن‌ها کلیک می‌کند. برای مثال اگر کاربر در یک وب‌سایت قرار دارد که سه پیوند به وب‌سایت‌های دیگر با کیفیت‌های 1.4 و 1.4 و 0.7 دارد، احتمال کلیک کردن او بر روی پیوندها به ترتیب برابر 0.4 و 0.4 و 0.2 است. هدف ما بررسی رفتار این کاربر در زمانی است که بی‌نهایت گام برمی‌دارد. برای بیان بهتر شرایط از ماتریس

انتقال کاربر استفاده می‌کنیم که در آن $w(e_{i,j})$ نشان‌دهنده وزن یال از راس i به راس j است:

$$H_{i,j} = \begin{cases} \frac{w(e_{i,j})}{\sum_{k=1}^{k=n} w(e_{i,k})} & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

حال $H_{i,j}^k$ نشان‌دهنده احتمال رفتن از راس i به راس j در دقیقا k گام در یک Random Walk بر روی گراف است.

با داشتن این ماتریس انتقال و استفاده از قضایای موجود در زنجیره مارکوف می‌توانیم به توزیعی برسیم که نشان‌دهنده زمان نسبی است که به‌طور میانگین کاربر در هر وب‌سایت، در یک گشت نامتناهی می‌گذراند. اما یک مشکل این است که کاربر در یک وب‌سایت که به هیچ وب‌سایت دیگری پیوند ندارد، گیر می‌کند و هیچ وقت به یک وب‌سایت که از دیگر وب‌سایت‌ها پیوند ندارد وارد نمی‌شود، که این دو مشکل در استفاده از توزیع به دست آمده برای رتبه‌بندی وب‌سایت‌ها خلل ایجاد می‌کنند.

برای حل این مشکلات ابتدا از راس‌های بدون یال خروجی، به تمامی دیگر راس‌ها یال با کیفیت برابر ایجاد می‌کنیم. حال زمانی که کاربر در وب‌سایتی باشد که به هیچ وب‌سایت دیگری پیوند ندارد، در گام بعدی به احتمال برابر می‌تواند به هر یک از وب‌سایت‌های دیگر برود. این شرایط را نیز می‌توانیم با ماتریس $S = H + A$ نمایش می دهیم که در آن A را به صورت زیر تعریف می کنیم:

$$A_{i,j} = \begin{cases} \frac{1}{n-1} & \text{if } i \neq j \text{ and } i \text{ has no outgoing edge} \\ 0 & \text{otherwise} \end{cases}$$

سپس به کاربر توانایی پرش احتمالی نیز اضافه می‌کنیم. برای شهود یافتن می‌توان اینگونه در نظر

گرفت که در ابتدای هر گام کاربر با احتمال مشخصی انتخاب می‌کند که از پیوندهای موجود در وب‌سایت فعلی استفاده کند یا به یک وب‌سایت دیگر بپرد. حالت دوم را در دنیای واقعی می‌توان وارد کردن مستقیم آدرس وب‌سایت توسط کاربر، در نظر گرفت. این شرایط را نیز با ماتریس $G = (1-\alpha)S + \alpha\frac{1}{n}ee^T$ نمایش می‌دهیم که ee^T ماتریس تمام 1 و $0 < \alpha < 1$ احتمال پرش کاربر است.

در پیاده‌سازی بهتر است به جای استفاده از ماتریس با تمام خانه‌های برابر برای نمایش مقصد پرش کاربر($\frac{1}{n}ee^T$)، از ماتریسی که بر اساس ترجیحات احتمالی کاربران برای وارد کردن آدرس مستقیم وب‌سایت ایجاد شده‌است، استفاده کرد. زیرا برای مثال احتمال وارد کردن آدرس وب‌سایت دانشگاه شریف با وارد کردن آدرس صفحه اصلی گوگل برای کاربر برابر نیست. ما در این تمرین برای حفظ سادگی، از این مورد صرف نظر می‌کنیم.

حال به حل کردن مسئله اصلی می‌پردازیم. ما به دنبال بردار سطری π از احتمالات می‌گردیم که:

$$\pi G = \pi$$

در این تمرین قصد حل این معادله به صورت iterative و با استفاده از روش ساده‌ای به نام power iteration داریم.

در ابتدا یک بردار توزیع اولیه

$$\pi^{(0)} = [\frac{1}{n}, \ldots, \frac{1}{n}]$$

ایجاد می‌کنیم و سپس مسیر همگرا زیر را ادامه می‌دهیم:

$$\pi^{(t+1)} = \pi^{(t)}G$$

در این تمرین به اثبات همگرایی نمی‌پردازیم، اما این همگرایی به دلیل افزودن توانایی پرش و تغییر سطرهای تمام صفر در ماتریس، وجود دارد.

$$\lim_{t\to\infty} \pi^{(t)} = \pi$$

این الگوریتم را معمولا به تعداد مشخصی بار تکرار می‌کنند اما در بعضی حالت‌ها، بردار π سریع‌تر از پایان تعداد تکرار مشخص شده الگوریتم، به همگرایی می‌رسد و الگوریتم را سریع‌تر متوقف می‌کنیم. در این تمرین زمانی می‌گوییم بردار همگرا شده‌است که نرم‌ـ۱ تفاضل بردار با بردار گام قبلی، کمتر از آستانه مشخص شده که با β نمایش می‌دهیم، باشد. یعنی:

$$||\pi^{(t)} - \pi^{(t-1)}||_1 < \beta$$

در این حالت بردار $\pi^{(t)}$ را به عنوان خروجی انتخاب می‌کنیم.

حتی با وجود رسیدن به راه‌حل مسئله، همچنان مشکل بسیار بزرگ بودن ماتریس G و سطر $\pi^{(k)}$ به اندازه‌ای که هیچ کدام از آن‌ها نمی‌توانند در memory یک سیستم قرار گیرند، وجود دارد. اما با توجه به اینکه روند ایجاد ماتریس G از روی ماتریس H را داریم و تعداد کمی از خانه‌های هر سطر H غیر صفر هستند، پس تنها با نگه‌داری شماره ستون خانه‌های غیر صفر در هر سطر و مقدار آن خانه‌ها، می‌توان سطری از H را در memory قرار داد. همچنین برای نگه‌داری ماتریس G فضای ذخیره‌سازی بسیار بیشتری نسبت به نگه‌داری ماتریس H به شکلی که گفته‌شد، مورد نیاز است که این فضا در دسترس سیستم‌های ما نیست.

حال شما باید با فرض گفته‌شده محاسبات را با کمک map و reduce به شکلی تغییر دهید که به صورت توزیع‌شده امکان‌پذیر شوند.

دقت کنید که $G = (1-\alpha)(H+A) + \alpha\frac{1}{n}ee^T$ است و نیازی به تشکیل و نگه‌داری کل ماتریس A نیز ندارید و نباید این‌کار را انجام دهید. صرفا می‌توان خانه مورد نظر از ماتریس A در هر لحظه را با توجه به اینکه راس متناظر با آن خانه، یال خروجی دارد یا خیر، به‌دست آورد. برای اطلاعاتی که در هر iteration نیاز هستند و محاسبه آن‌ها زمان‌بر است (مانند بررسی اینکه یک راس یال خروجی دارد یا خیر) حتما پیش‌پردازش و نگه‌داری کنید. دقت کنید که برخی از این اطلاعات نیز می‌توانند بزرگ باشند و در memory یک سیستم قرار نگیرند و لازم باشد آن‌ها را به صورت توزیع‌شده، محاسبه و استفاده کنید.

راه‌حل ارائه‌شده به الگوریتم PageRank شباهت دارد و نحوه پیاده‌سازی توزیع‌شده این الگوریتم، می‌تواند به شما در حل این تمرین کمک کند. می‌توانید از این لینک و این لینک استفاده‌کنید.

شما در این تمرین باید راه‌حل ارائه شده را با توجه به محدودیت‌های گفته شده، در pySpark پیاده‌سازی کنید و برنامه شما در نهایت برای مقدار خواسته شده k ، بردار $\pi^{(k)}$ را و در صورتی که زودتر همگرایی اتفاق افتاد، بردار مورد نظر را خروجی دهد.

نحوه اجرا

برنامه شما به صورت زیر اجرا می‌شود:

```
pagerank.py <file> <teleport-probability> <max-iterations> <convergence-threshold>
```

که در آن <file> آدرس فایل به صورت کامل، <teleport-probability> احتمال پرش یا همان α ، <max-iterations> حداکثر تعداد تکرار الگوریتم یا نشان دهنده k است که در صورت نرسیدن به همگرایی تا قبل از آن باید $\pi^{(k)}$ را خروجی دهید و <convergence-threshold> آستانه همگرایی یا همان β است.

نمونه اجرا:

```
pagerank.py /home/sample-test.txt 0.15 10 0.01
```

فایل ورودی با فرمت متن ذخیره شده‌است و در هر خط از آن مانند خط زیر، سه عدد که با فاصله از یکدیگر جدا شده‌اند قرار دارند که نشان‌دهنده یک یال از گراف هستند.

```
<source_id:int> <destination_id:int> <weight:float>
```

برای مثال یک فایل ورودی می‌تواند مانند زیر باشد.

```
1 2 5.2
1 4 3
1 3 2.5
1 1 2.0
2 3 1
2 4 4.3
3 1 2
3 2 2.3
```

فایل ورودی ترتیب خاصی ندارد.

تضمین می‌شود راسی وجود ندارد که هم یال خروجی و هم یال ورودی نداشته باشد.

برای ذخیره خروجی، فرض کنید حجم rdd بردار نهایی از مموری سروری که قرار است خروجی در آن ذخیره شود کمتر است و آن را با ساختار

```
(<vector_id:int>, <value:float>)
```

به ۱ پارتیشن تبدیل کنید و با دستور

```
myRDD.saveAsTextFile('./result')
```

ذخیره کنید. شما باید بتوانید در فایل part-00000 ایجاد شده در پوشه result ، خروجی خود را به صورت متنی مشاهده کنید.

نکات

- در این لینک توضیحات بسیار مفیدی همراه با نمونه کد برای کار با اسپارک وجود دارد که می‌تواند منبع خوبی برای شما باشد.
- برای دسترسی به اسپارک، از تابع getOrCreate موجود در Spark Session Builder استفاده کنید و تنظیمات پیشفرض آن از جمله تعداد کارگرها و حداکثر میزان ممووری را تغییر ندهید. در پایان برنامه نیز اسپارک را stop کنید.
- برای نگه‌داری داده‌ها از داده‌ساختار rdd استفاده کنید. استفاده از dataFrame, graphX یا توابع sql مجاز نیست. همچنین تنها مجاز به استفاده از spark و کتابخانه‌های پیشفرض python3 مانند sys هستید.
- داده‌های اولیه را حداقل به ۴ پارتیشن تقسیم کنید. این پارتیشن بندی را به صورتی انجام دهید که با کاهش میزان داده‌ای که قرار است در مراحل بعدی در فرایند shuffle جابه‌جا شوند، باعث کاهش زمان اجرای برنامه در مراحل بعدی شود. با توجه به اینکه تعداد مناسب پارتیشن‌ها برای بازده بهتر به ساختار سرور وابسته‌است، پیشنهاد می‌کنیم سعی کنید هر مجموعه داده را در طول برنامه نیز در ۴ پارتیشن نگه دارید.
- برنامه باید دارای چکپوینت‌هایی برای بازیابی باشد، اگر سرور در بین راه دچار مشکل شد نباید محاسبات قبلی از بین برود. نیازی به پیاده‌سازی قابلیت بازیابی نیست و تنها چکپوینت کردن

مجموعه داده در زمان‌های مناسب کافی است. این چکپوینت‌ها را در پوشه checkpoints در محل اجرای برنامه خود ذخیره کنید.

- برنامه شما باید از نظر زمان اجرا، عملکرد خوبی داشته باشد و یکی از معیارهای اصلی نمره‌دهی، زمان اجرای برنامه است. یکی از راه‌های کاهش زمان اجرا استفاده از caching است. یکی از توابعی که می‌تواند در این زمینه به شما کمک کند، تابع persist است که می‌توانید توضیحات بیشتر در رابطه با آن و انواع سطح استفاده از آن را در این لینک مطالعه کنید. شما باید از این تابع یا توابع با کاربرد مشابه در برنامه خود استفاده کنید.

گزارش

در کنار فایل نهایی پایتون، باید گزارشی با فرمت PDF نیز با محتوای موارد زیر تحویل دهید.

- نحوه دقیق عملکرد برنامه خود را به صورت خط به خط شرح دهید.
- اگر از ایده خاصی برای کاهش میزان محاسبات استفاده کردید، شرح دهید.
- اگر ایده دیگری برای کاهش میزان محاسبات داشتید و نتوانستید پیاده‌سازی کنید، شرح دهید.
- در هر بار استفاده از تابع persist دلیل استفاده از سطح ذخیره‌سازی مورد استفاده خود یا اگر از تابع جایگزینی استفاده کردید، دلیل خود را شرح دهید.
- دلیل مناسب بودن نحوه پارتیشن بندی خود و کم کردن حجم داده مورد نیاز به shuffle در گام‌های بعدی را توضیح دهید. آیا با داشتن موضوع هر وب‌سایت، در صورتی که تعداد موضوعات کم و تعداد وب‌سایت‌ها در هر موضوع زیاد باشد، می‌توان با پارتیشن بندی با استفاده از موضوعات، میزان داده‌ای که قرار است در فرآیند shuffle جابه‌جا شود را احتمالا کاهش داد؟ توضیح دهید.