به نام خدا

گزارش تمرین چهارم کامپیوتری

محمد مهدی صباغی (95109123)

توضیح کد:

در این اینجا ابتدا توابعی را که در کد تعریف کرده ایم توضیح میدهیم. سه تابع تعریف شده است که اولی parseNeighbors است که هنگامی که یک خط از فایل خوانده شد آن را به سه قسمت تقسیم کرده و به فرمت خواسته شده بر میگرداند. دومین تابع gettingNeighbors است که علت تعریف برای نود هایی بود که خروجی از آن ها وجود ندارد و اگر صرفا تابع اول استفاده میشد هیچ کلیدی برای این نود ها وجود نداشت و در پیاده کردن الگوریتم به مشکل میخوریم. ولی اینجا در ادامه می بینیم اضافه کردیم این کلید با مقدار صفر کمک بزرگی خواهد کرد. تابع سوم probs_compute است که قسمت اصلی کد است. هنگامی که preprocessing تمام شد از این تابع استفاده میکنیم تا نود هایی value برای آن ها تنها یک صفر است یعنی که خروجی از آن ها وجود ندارد و n-1 تا احتمال برابر با استفاده از generator و دستور yield که نیازی به حافظه جداگانه ندارد و به ترتیب خروجی ها را می دهد استفاده کردیم، همچنین در صورتی که value تنها صفر نباشد یعنی اینکه آن نود خروجی داشته است. بنابراین در ابتدا جمع وزن ها را از یک نود حساب کرده و احتمال ها را بیرون میدهد.

در ادامه برای اینکه بتوانیم مقادیر را از سیستم بگیریم main را تعریف کردیم و با استفاده از 4 ،sys مقدار خواسته شده را از سیستم گرفتیم. همانطور که در صورت گفته شده spark و sparkContext را تعریف کردیم. سپس فایل را خوانده و با استفاده از دستور sc.parallelize آن را به 4 پارت تقسیم کردیم. سپس همانطور که در پاراگراف قبل توضیح داده شد، با دو تابع parseNeighbors و gettingNeighbors دو نوع key_value ساختیم. نوع اول کلید شماره نود و مقدار یک tuple است که نود خروجی و وزن آن درون آن است. در نوع دوم کلید شماره نود خروجی و مقدار آن صفر است. هنگامی که این دو را به هم با استفاده از union چسباندیم و groupbyKey کردیم در خروجی آن یک کلید به ازای هر نود وجود دارد که اگر خروجی داشته باشد به جز صفر مقادیر دیگری هم درون آن میباشد. همچنین با دستور count از روی این rdd تعداد نود ها را به دست آوردیم. سپس از روی این بردار احتمالات را ساختیم و اینها را به هم وصل کرده با استفاده تابع probs_compute احتمالاتی که برای هر نود ایجاد میشود را محاسبه کردیم و با استفاده از دستور reducebyKey مجموع احتمال برای هر نود به دست آمد. همچنین با استفاده از یک map احتمال پرش را اضافه کردیم. دقت کنید که:

$$1/n\, ee^T p = 1/n\, e$$

بنابراین کافی است که هر کدام از احتمالات قبلی را در $1-\alpha$ ضرب کرده و با α/n جمع کنیم تا احتمال درست به دست بیاید. در ادامه هم بردار احتمال جدید را با قبلی مقایسه کرده تا اگر نرم 1 فاصله آن ها کمتر از بتا بود متوقف شویم.

همچنین همه بردار های احتمال در فولدر breakpoints و بردار نهایی در result ذخیره شد.

دو ایده که میتوانستیم پیاده کنیم ولی به علت پیچیدگی از آن صرف نظر کردیم، اول این بود که ابتدا بلوک هایی را در گراف تشخیص دهیم که بیشترین همبستگی را دارند تشخیص دهیم و هر کدام از آن ها را یک پارت جداگانه کنیم. به این ترتیب تعداد shuffle ها بسیار کمتر خواهد شد. ایده دوم هم این بود که به جای اینکه کل prob_contribs(سهم های نود ها بردار احتمال قبل از جمع کردن) را همه را باهم reduce کنیم، این کار را قسمت قسمت میکردیم و به این صورت باز هم تعداد shuffle ها لگاریتمی کم میشد.

ما تنها rdd لینک را ذخیره کردیم چرا که در هر دور محاسبه نیاز بود. اما بقیه آن ها (probs, contribs) در هر دور تغییر میکردند و نیازی به ذخیره آن ها نبود.

همچنین همانطور که در قسمت قبل گفتم اگر که میشد گراف را به بلوک هایی بسیار کوچکتر شکاند سرعت الگوریتم هنگامی که پارت ها را همان بلوک میگذاشتیم بسیار بیشتر بود چون شافل ها بسیار کمتر میشدند.