بسم الله الرحمن الرحیم

گزارش پروژه دوم درس ساختمان داده – دکتر دهقان

پرهام رحیمی - 9531031

چکیده

این گزارشی از بررسی نحوه های مختلف پیاده سازی الگوریتمی مربوط به گراف هاست که در سال 2004 برای جدا سازی اجتماع های موجود در داده ورودی معرفی شد. در این گزارش سعی شده تاثیر حجم داده ی ورودی، نوع مرتب سازی و ساختمان داده ی استفاده شده برای ذخیره سازی داده ی ورودی در حافظه رم، بر روی سرعت و کارایی الگوریتم بررسی شود که در ادامه به شرح آن می پردازم.

مقدمه

تشخیص ساختارهای اجتماعاتی یک موضوع مهم در بسیاری از زمینه ها می باشد. این موضوع با مفاهیمی مانند شبکههای اجتماعی (روابط بین اعضا)، تحقیقات بیولوژیکی یا مسایل تکنولوژیکی (بهینه سازی زیرساخت های حجیم) در ارتباط می باشد.

در سال های اخیر شواهد با سرعت زیادی رشد نموده اند که بسیاری از سیستمهای موجود در زمینه های مختلف را می توان با شبکه ها مدل نمود؛ به عنوان مثال به وسیله ی چند راس و یال و با خواص ساختاری کلی. سیستم های تکنولوژیکی مانند اینترنت یا بیولوژیکی مانند شبکه های متابیولوکی و سیستم های اجتماعی نمونه ای از این سیستم ها می باشند.

از لحاظ کیفی، اجتماع زیرمجموعهای از رئوس یک گراف است که ارتباطات داخلی بین رئوس حجیم تر از ارتباط با بقیه شبکه می باشد. یافتن اجتماعات در گراف را می توان به طور عمومی به نگاشت شبکه به یک درخت تعبیر نمود. در این درخت برگ ها راس ها می باشند که بوسیله ی شاخه ها، راس ها یا گروه های از رئوس بهم متصل شده اند؛ این یک ساختار تو در تو از اجتماعات می باشد.

شرح الگوریتم

- الگوریتم "تعریف و تشخیص اجتماعات در شبکه ها":

الگوریتم نیومن-گیروان بسیار هزینه بر می باشد. برای برطرف نمودن این مشکل، ما یک الگوریتم تقسیم کننده که برپایه ی کمیت های محلی می باشد معرفی می کنیم که از الگوریتم نیومن-گیروان سریع تر می باشد. بخش اصلی الگوریتم های تقیسم کننده کمیتی است که با آن بتوان یال های بین اجتماعات

را تشخیص داد. در این الگوریتم، با حذف یال های بین اجتماع ها، گراف به زیر گراف های متراکم تر تقسیم می شود.

- ساختمان داده های استفاده شده و پیاده سازی:

برای خواندن از CSV text file با اسپلیت کردن هر خط دو داده ی راس مبدا و راس مقصد در هر یال در دو لیست از راس ها (راس های مبدا و راس های مقصد) ذخیره می شوند.

لیست مجاورت گراف داده شده که به وسیله ی Arraylist ای از Arraylist ها پیاده سازی شده است؛ که در آن هر گره ی Arraylist اولیه نماینده ی یک راس در گراف است و گره های Arraylist ای که به آن وصل است نماینده ی راس هایی هستند که به آن راس وصل هستند (مجاور آن راس هستند). و بر اساس داده های به دست آمده از فایل ساخته می شود.

با توجه به حجیم بودن داده های ورودی ماتریس مجاورت گراف به طور مستقیم قابل پیاده سازی نیست و به ناچار از sparse matrix استفاده می کینم، این ساختمان داده به وسیله ی Arraylist ای از آرایه های یک بعدی دو متغیره پیاده سازی شده است که خانه های آرایه دو سر یال ها را ذخیره میکنند.

برای پیاده سازی الگوریتم نیازمند به در اختیار داشتن یال ها هستیم برای همین در ساختمان داده ی Arraylist ای از آرایه های یک بعدی به ظرفیت سه، آن ها را ذخیره می کنیم؛ به این صورت که به ازای هر یال یک گره Arraylist ساخته می شود که در آن گره یک آرایه 3 متغیره قرار دارد که این متغیر ها به ترتیب: راس اول یال، راس دوم یال، و امتیاز یال هستند که پیش محاسبه امتیاز امتیاز هر یال به طور پیش فرض صفر در نظر گرفته می شود.

برای پیاده سازی الگوریتم نیاز به انواع سورت ها داریم که هر یک جداگانه پیاده سازی شده اند (کوییک و مرج به صورت بازگشتی و دو سورت بابل و اینزرشن به صورت iterative).

با این حساب و با توجه به ساختمان داده های استفاده شده به دست آوردن امتیاز هر یال ممکن می شود به این صورت که برای هر یال که در Arraylist با نام edge ذخیره شده اند امکان دسترسی به لیست مجاورتی دو راس دو سر آن یال هست و با این کار می توان راس هایی که مجاور هر دو راس هستند را شناسایی$O(V)$ کرد و به عبارت دیگر تعداد دور های 3 تایی$O(E+V1*V2)$ و درجه هر راس$O(1)$ را به دست آورد. و سپس امتیاز به دست آمده $O(E + V*V)$را در خانه آخر edge ذخیره کرد.

به این ترتیب گام های اول و دوم الگوریتم انجام می شود.

برای گام سوم یال را هم از edge $O(1)$و هم راس های دو سر آن را در همسایگی یکدیگر در لیست مجاورت حذف می کنیم$O(V)$.

برای بررسی گام چهارم از الگوریتم DFS استفاده شده است $O(VE)$ به این صورت که اگر در یک دور پیمایش کامل این الگوریتم همه ی اعضا visited شوند یعنی گراف همچنان همبند است و در غیر این

صورت گراف ناهمبند شده و الگوریتم خاتمه میابد. برای پیاده سازی DFS پشته (Stack) (push & pop : O(1)) نیز پیاده سازی شده است.

مشخصات سخت افزاری

	Mine	Friend
CPU Model	Intel Core i7 6600u	Intel Core i7 4700HQ
CPU Physical Core	2	4
CPU Virtual Core	4	8
CPU L1 Cash	64 KB	128 KB
CPU L2 Cash	512 KB	1024 KB
CPU L3 Cash	4 MB	6 MB
RAM Model	DDR4	DDR3
RAM Capacity	12 GB	8 GB
RAM Bus		
HDD/SSD Write Speed	SSD	HDD
HDD/SSD Read Speed	SSD	HDD
OS	Windows 10	Windows 10

داده ها

Name	Vertex Number	Edge Number	Average Vertex Degree
myr1	5	14	2.8
PK	13	42	3.23
mtest0	100	1578	1.58
mtest1	1000	38347	3.83

مقایسه ها و نتایج

تفاوت استفاده از ماتریس مجاورت و لیست مجاورت گراف:

اگر برای ماتریس همسایگی از ماتریس عادی استفاده می شد حافظه ی بیشتری از لیست مجاورت مصرف می کرد چرا که می بایست برای خانه های صفر ماتریس نیز فضای ذخیره سازی مصرف میشد ولی در ازای آن سرعت به مراتب بالاتری را ارائه میداد چرا که به هر خانه حافظه آن به صورت مستقیم می توانستیم دسترسی داشته باشیم. در صورتی که برای دسترسی به گره های لیست مجاورت باید یک دور آن را پیمایش

کنیم (O(n)). اما چون حافظه مصرفی ماتریس عادی انقد زیاد بود که از گنجایش اختصاص داده شده ی رم کامپیوتر من تجاوز میکرد مجبور به استفاده از ماتریس sparse شدم که در سرعت هم کند تر از لیست مجاورت عمل می کند. چرا که در ماتریس sparse هم برای دسترسی به یال ها باید با O(E) کل یال ها را پیمایش کرد و هم برای پیدا کردن یال های متصل به هر کدام از گره های دو سر آن یال کل یال ها را پیمایش کرد.