توضیح الگوریتم NIBLPA :

این الگوریتم مشابه LPA عادی کار می کند . تفاوت این دو در این است :

- LPA با تابع رندوم راس ها را در لیست X قرار می دهد و به ترتیب لیست X ، برای هر راس برچسبی که بیشترین تکرار را در همسایه اش دارد به عنوان برچسب جدیدش انتخاب می شود . در صورت داشتن چند راس با برچسب ماکسیمم ، بار دیگر به صورت رندوم یکی از آن ها انتخاب می شود .

- NIBLPA بر حسب NI راس ها را در یک Vector مرتب سازی می کند . NI خود بر حسب KS بدست می آید . (تعریف KS : به هر راس گراف یک مقدار k اختصاص داده می شود ؛ به این صورت که هر راس در مجموعه ی یک زیرگراف به نام k-shell با راس های متصل به هم قرار دارد که درجه هر راس آن حداقل k می باشد ؛ k بزرگترین مقدار ممکن است به طوری که راس مورد نظر نمی تواند در یک k+1 shell قرار داشته باشد .) برای محاسبه ی آن تا زمانی که دیگر یالی نتواند پاک شود ، یال های رئوس با درجه ی کوچکتر مساوی k را پاک می کنیم و KS آن را برابر با k قرار می دهیم . سپس به سراغ k بعدی می رویم تا در انتها تمام یال ها پاک شده باشند . سپس با پیمایش راس ها مشابه LPA سعی می کنیم برچسب با بیشترین تکرار در همسایگی هر راس را پیدا کنیم و برچسبش را به آن برچسب تغییر دهیم تا اجتماعات در گراف تشخیص داده شوند . برای راس هایی که چند برچسب همسایگی با برچسب ماکسیمم دارند و هیچ کدام برچسب مشابه راس انتخابی ندارند ، (در غیر این صورت برچسب راس تغییر نمی کند چه در LPA چه در NIBLPA) بر حسب NI پارامتر LI آن ها محاسبه می شود و برچسبی که ماکسیمم LI را داشته باشد برچسب آن راس خواهد بود . اگر چند برچسب بدین شکل بودند به صورت رندوم از میان آن ها انتخاب می کنیم و برچسب گذاری می کنیم . در انتهای هر دو الگوریتم راس ها با برچسب یکسان داخل یک community قرار می گیرند .

توضیح مختصر توابع موجود در بخش NIBLPA پروژه :

❖ Begin : به هر راس یک برچسب مجزا اختصاص می دهد .

❖ findKS : مقدار k را با توجه به توضیحات بالا برای هر راس محاسبه کرده و درون آرایه ی KS قرار می دهد .

❖ findNI : مقدار NI را با توجه به فرمول $$\mathrm{NI}(i) = Ks(i) + \alpha * \sum_{j \in N(i)} \frac{Ks(j)}{d(j)},$$ برای هر راس محاسبه می کند .

به طوری که منظور از N(i) مجموعه ی تمام رئوس مجاور راس i می باشد و منظور از d(j) درجه ی راس j است .

توجه : ∝ یک مقدار ثابت در طول الگوریتم و عددی بین 0 و 1 است که مقدار آن می تواند در ابتدای اجرای الگوریتم توسط کاربر وارد شود . ما در اجرای 100 باره ی هر تست کیس ، مقادیر 0.00 ، 0.01 ، 0.02 ... 0.99 را به آن اختصاص داده ایم تا در هنگام میانگین گیری ، حد متوسطی را برای جواب نهایی بر اساس مقادیر مختلف ∝ در برگرفته باشیم .

❖ Merge , sort : پیاده سازی الگوریتم merge – sort برای استفاده در تابع sortX

❖ sortX : راس ها را به ترتیب نزولی NI شان مرتب کرده و در آرایه ی X قرار می دهد .

❖ caclulateMaxLI : برای هر راس i که در ورودی تابع داده شود ، مقدار LI اش را با توجه به فرمول $$\mathrm{LI}(l) = \sum_{j \in N^l(i)} \frac{\mathrm{NI}(j)}{d(j)}.$$ محاسبه می کند و در آرایه ی LI قرار می دهد . به طوری که منظور از $N^L(i)$ مجموعه ی تمام رئوس با برچسب ماکسیمم در بین همسایگان راس i است .

❖ updateLabels : با توجه به توضیحات مطرح شده در بالا ، اگر در بین برچسب رئوس مجاور یک راس ، چندین برچسب با بیشترین تکرار موجود باشد ، برچسب با بیشترین LI انتخاب می شود :

$$c_i = \arg\max_{l \in l\max} \mathrm{LI}(l),$$

- ❖ divideCommunity : در انتهای الگوریتم ، هنگامی که عمل آپدیت کردن برچسب ها دیگر ادامه نمی یابد ، رئوس با برچسب های یکسان ، در یک اجتماع قرار داده می شوند که به صورت hashSet پیاده سازی شده است . همچنین خروجی result در این تابع مقدار دهی می شود ؛ به این صورت که درهر vector از آن ، تمام رئوس با برچسب های مشابه قرار داده شده اند .

- ❖ detectCommunity : با صدا زدن این تابع ، تمام الگوریتم NIBLPA به ترتیب و گام به گام اجرا می شود .