1396.11.5

به نام خدا

جابر بابکی 96131020

پروژه اول

بزرگ فکر کن ، هوشمندانه تصمیم بگیر، اما کوچک شروع کن

مسیر پروژه:

عملیات LookUp

1. اگر subnet mask برابر 15 یا کمتر بوده در subTrie سطح اول که در حافظه on-chip قرار داده شده است جستجو می شود.
2. اگر subnet mask بیشتر از 15 و کمتر از 25 بوده باید جستجو و در PCTrie سطح دوم انجام شود .
3. اگر بزرگتر از 25 بوده باید جستجو در PCTrie سطح سوم انجام شود.

جدول کل توابع کاربردی در انجام پروژه

|  |  |  |
| --- | --- | --- |
| شماره | نام تابع | عملکرد |
| 1 | createTrie(String key, String next) | این تابع برای ساخت درخت Trie می باشد و با ورودی prefix و next hop در خت trie را می سازد |
| 2 | searchInTrie(String key, TrieNode trie) | با گرفتن prefix و درخت trie مورد نظر آن prefix را در آن درخت بررسی می کند |
| 3 | printDFS(String prefix, TrieNode n, boolean isLeft) | نمایش درخت Trie به صورت پیمایش عمقی |
| 4 | fullTrie(TrieNode trie, int lev) | این تابع یک درخت را می گیرد و آن را کامل می کند. |
| 5 | eliminate(TrieNode root) | این تابع یکی از دو شرط ساخت PCTrie را انجام می دهد و node set که دو فرزند داشته باشد را حذف می کند |
| 6 | createPCTrie(TrieNode root) | این تابع بعد از انجام دو شرط لازم برای Pctrie این PCTrie را به صورت bitmap و همچنین next hop را در آرایه دو بعدی قرار می دهد |
| 7 | hashing(List<String> subTrie, List<TrieNode> subTrieModelLevel, List<TreeBitmapModle> pctries) | عملیات هشینگ در این تابع انجام می شود و هر گره Root هر subTrie بر اساس تابع hash در جدول hash ذخیره می کند چنانچه برخوردی رخ دهد آن ها را blackSheep ذخیره می کند خروجی این تابع یک memberShip می یاشد |
| 8 | searchInTreeBitmap(TreeBitmapModle tree, String prefix) | عملیات جستجو در TreeBitmap به این صورت می باشد که ابتدا باید در رشته Bitmap مورد نظر جستجو کرده |
| 9 | lookUp(String inputIP) | عملیات lookup در 15 بیت اول |
| 10 | convert(String str) | تابع IP را دریافت می کند و به صورت باینری بر اساس subnet mask نشان می دهد. |
| 11 | readFromFile() | این تابع فایل ورودی را دریافت و در ون اون فایل بر اساس کد های regex که نوشتیم prefix ها و nexthop ها را بیرون می اورد. و خروجی که می دهد یک دیتا مدل PrefixNexthop است. |

جدول کلاس های کاربردی

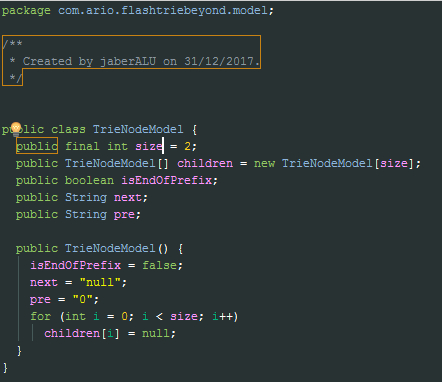
|  |  |  |
| --- | --- | --- |
| شماره | نام کلاس | عملکرد |
| 1 | Trie | در کلاس Trie عملیات توابع ساخت و جستجو و نمایش درخت Trie و همچنین ساخت PCTrie وجود دارد |
| 2 | MainLookUp | در این کلاس توابع مهمی همچون hashing و searchInTreeBitmap وجو دارد و به طور کلی عملیات در این کلاس انجام می شود |
| 3 | ReadTextFile | کلاسی کاربری که در آن تابع readFromFile وجود دارد که فایل ها را می خواند و prefix ها را list میکند و نمایش می دهد |
| 4 | IpToPrefix | در این تابع یک کلاس وجود دارد آن هم convert هست که Ip ها را به prefix تبدیل می کند |

جدول کلاس مدل

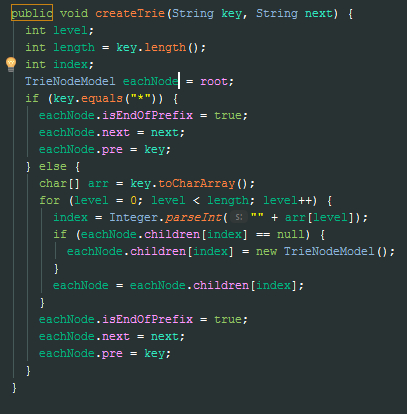
|  |  |  |
| --- | --- | --- |
| شماره | نام مدل | عملکرد |
| 1 | TrieNodeModel | این مدل در واقع هر گره در درخت Trie را تشکیل می دهد که شامل فیلد های اشاره گر گره سمت چپ و راست و همچنیم prefix و nexthop هست. |
| 2 | PCTrie2Modle | این مدل دو فیلد اصلی دارد یکی Treebitmap که در واقع رشته بیت هست و دیگری جدول دو بعدی هست که nexthop را نگه می دارد |
| 3 | HashModel | بر طبق member query sheep که در مقاله گفته شده سطر های جدول هش دارای سه فیلد اصلی verifyBit و rootNodeNHI و collision که این ها به عنوان فیلد این دیتا مدل در نظر گرفته شده اند |
| 4 | BlackSheepModel | طبق چیزی که در مقاله گفته شده سطر های درون جدول balsckSheep همان فیلد های hashtable هستند با این تفاوت که آدرس PCTrie مستقیم درون آن می باشد |
| 5 | MemberShipModel | این دیتا مدل در واقع حاولی HashModel و BlackSheepModel هست و این فیلد را برای تمام سطوح نگه می دارد |
| 6 | PrefixNextHob | این دیتا مدل موقع خوندن فایل ها اطلاعات درون این ها قرار می گیرد و در listView که یک کنترل اندرویدی هست نشان داده می شود. |

توضیحات و عملکرد هر تابع

1. ساخت درخت Trie

برای ساخت درخت Trie ابتدا باید دیتا مدل ساخت درخت Trie بررسی شود.

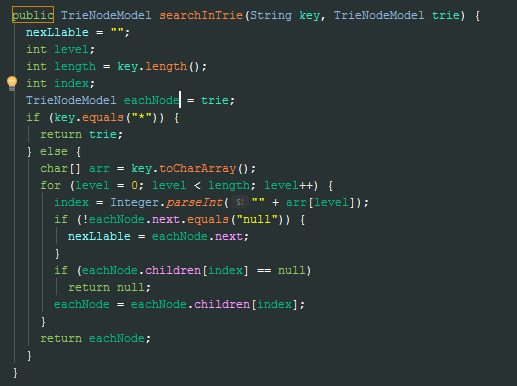
همونطور که در تصویر موجود هست دیتا مدل هر گره از درخت Trie شامل آرایه ای از جنس همان TrieNodeModel هست و در واقع با این کار که انجام دادم با دسترسی به یک گره می توان گرهای پایینی آن را هم پیدا کرد سایز این ارایه 2 هست یکی برای فرزند چپ و دیگری برای فرزند راست در واقع گره ها در این درخت را به صورت تو در تو در تشکیل دادم به جز فیلد children سه فیلد دیگه هم وجود دارد فیلد isEndOfPrefix که نشان دهنده این هست که اون گره یک prefix هست یا نه و nexthop و prefix فیلد های دیگر هستند به محض ایجاد شی از این کلاس constractor اون صدا زده می شود و مقدار های پیش فرض را وارد می کند.

حالا تابع ساخت Trie را بررسی می کنیم.

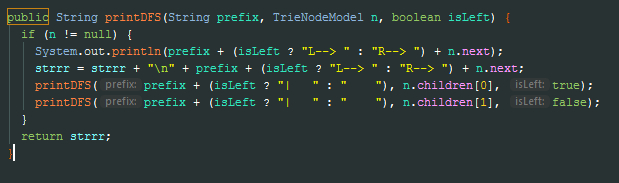
برای ساخت درخت Trie تابع createTrie صدا زده می شود، که ورودی آن یک key به عنوان prefix و nexthop دریافت می شود، ابتدای تابع یک نود ساخته می شود که با root خود یک نود global در به صورت فیلد در ابتدای کلاس Trie تعریف شده پر می شود، سپس if اول بررسی می کند که اگر ورودی استار هست فیلد های دیگر نود را پر می کند و تابع به پایان می رسد، اگر غیر استار باشد وارد else می شود و بر اساس طول prefix حلقه می چرخد و هر بار بر اساس یک یا صفر بودن فرزند چپ یا راست ایجاد می شود و به همین ترتیب نود جدید ایجاد می شود و در پایان در آخرین نودی که ایجاد شده است next hop و prefix قرار می گیرد و isEndOfPrefix به نشان نودی که حاوی prefix هست پر می شود.

1. جستجو در درخت Trie

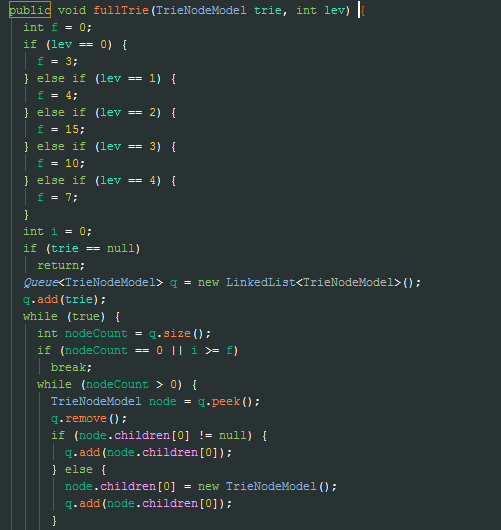
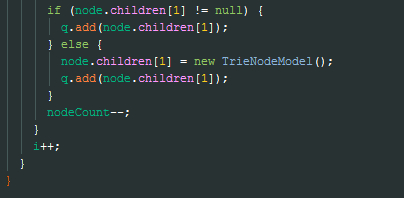
تابع searchInTrie در واقع prefix و درخت Trie داده می شود و سپس آن گره در صورت وجود برمیگرداند اگر ان گره وجود نداشته باشد نزدیکترین گره ای که prefix داشته را بر میگرداند .



در ابتدا نودی ساخته می شود و با Trie که به عنوان ورودی داده می شود پر می شود با این کار در واقع من هر subTire را بخوام میتونم سرچ بزنم توش و همچنین نزدیکترین پدر را پیدا کنم و این تابع خیلی کاربردی هست برای ما و پر کاربرد، یه فیلد استاتیک در کلاس Trie قرار دادم که در مراحل پیمایش اگر محتوای نودی غیر null بود درونش میذاره و در هر جای دیگه میتونم ازش استفاده کنم کهنشان دهنده نزدیکتزین جد یا پدر هست پس با این حرکت در واقع دو تابع را در یک تابع نوشتم.

1. پیمایش در درخت Trie

تابع پیمایش به صورت عمقی و بازگشتی نوشته شده است ورودی تابع می تواند یک Trie کلی یا هر subTrie باشد و در هر بار فراخوانی فرزند چپ یا راست که خود یک trie هستند وارد تابع می شود اون prefix که نوشتم یعنی پشت چیزی که نمایش داده میشه یه چیزی بنویسم و نشون داده بشه همچنین من در لاگ سیستم این درخت را نمایش می دهم و در نهایت در یک فیلد استاتیک نمایش را از جنس String هست قرار می دهم و سپس هر جایی که دوست داشتم می خوانم.

1. کامل کردن یک subTrie

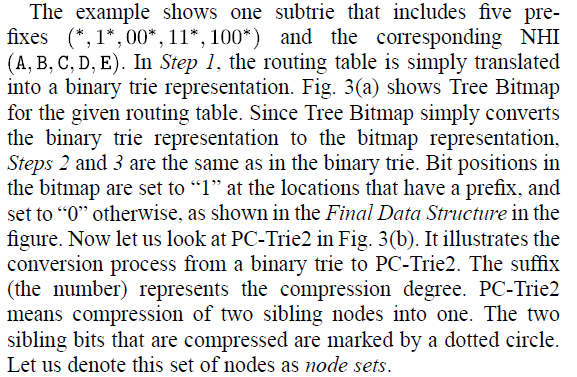
در این تابع ورودی یک subTrie و lev نشان دهنده این مورد هست که تا چه سطحی کامل شود من چون علاوه بر استفاده از فایل که داده شده مثال درون مقاله که تا سطح 3 کامل می کند و همچنین یه مثالی خودم زدم که دو سطحش تا 3 و سطح اخرش تا سطح 4 کامل می شو.د از طرفی در مورد IP های واقعی سطح اول از 0 تا 15 سطح دوم از 16 تا 24 و سطح سوم از بیت 25 تا 32 هست پس سطوح متفاوت هست و در ابتدای تابع تعیین شود. سپس گره اول را درون یه صف قرار می دهیم وفرزندان را بررسی می کنیم و چنانچه فرزندان آن null باشد برای آن فرزندی ایجاد و آن را درون صف قرار می دهیم این روند تا سطح مورد نظر انجام می شود.

1. ساخت PCTrie

به طور کلی من برای ساخت PCTrie سه مرحله من در نظر گرفتم:

مرحله اول : پیمایش درخت و گره هایی خود nextHop هستند ولی برادر آن ها دارای netxHop نیست پس باید نزدیکترین جد را پیدا کرده و nextHop ان را درونش قرار بدهیم که این خود یک تابع به نام createPCTrie شده است

مرحله دوم : این درختی که به صورت بالا درش آوردیم باید NodeSet هایی که هر دو فزند آن ها nodeSet بوده را nextHop ان ها را پاک بکنیم

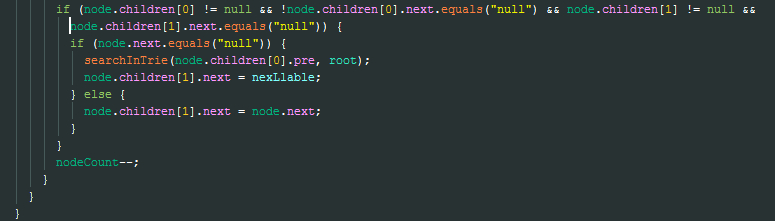
مرحله سوم : باید این درخت را TreeBitmap بکنیم و nextHop ان را درون ارایه دوبعدی قرارا بدهیم

توجه شود که من از PCTrie 2 استفاده کردم چون مقاله در نهایت بر روی IP ها این PCTrei2 را استفاده کرده است ولی تبدیل به PCTrie 4 یا 8 فقط در مرحله دوم تغییری ایحاد می کند که مقاله این کار را انجام نداده است.

به طور کلی PCTrie همان TreeBitmap معمولی هست با این تفاوت که قبلش توسط مرحله اول و دوم فشرده سازی بر روی درخت انجام می دهد و باعث کاهش حافظه مصرفی و افزایش پیچیدگی زمانی می شود. و ایده اصلی مقاله در همین رابطه بوده است.

5-1 مرحله اول: برادر پیدا کردن هر گره بی برادر

در این مرحله با استفاده از تابع مورد نظر تمام گره هایی که خود nestHop دارند ولی برادر آن ها خالی هست را نزدیکترین جد را پیدا کرده و قرار می دهیم.

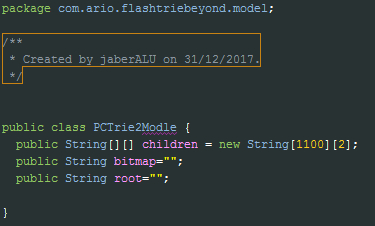


همانطور که دیده می شود پیمایش مانند قبلی ها در صف قرار می هد و چک کردن به این صورت است که اگر گره ای غیر null باشد و فرزند چپ مقدار nextHop اش برابر با “null” باشد و فرزند راست غیر null باشد و مقدار nextHop اش برابر غیر “null” باشد یعنی nextHop داشته باشد پس باید نزدیکترین جد برای ان پیدا شود و من در اینجا تابع searchInTrie که گفتم خیلی کاربرد دارد را با prefix برادر صدا می زنم و نزدیکترین پدر پیدا می شود. و قرار می دهم.

5-2 مرحله دوم : حذف nodeSet ها فرزند دار

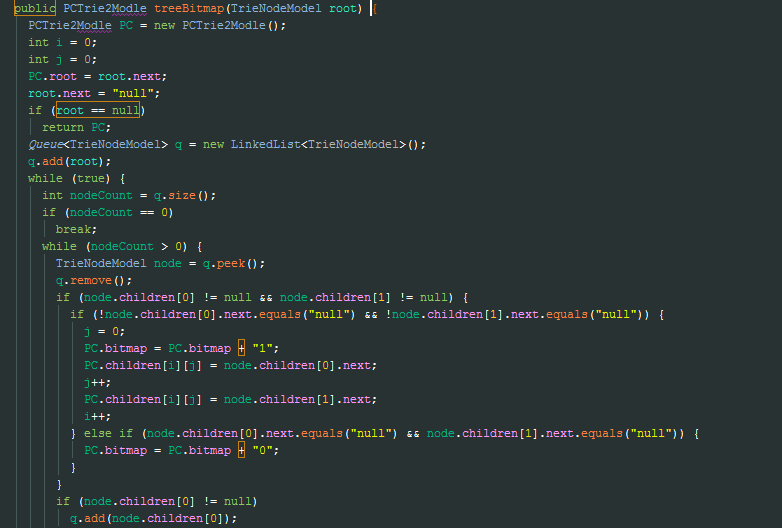
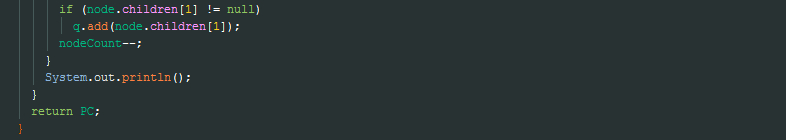
در این تابع با بررسی اینکه یک nodeSet فرزند چپ و راست داشته باشه در این صورت nextHop خودشون را برابر null قرار می دهم.

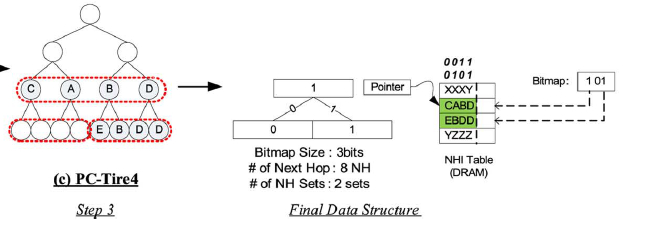
5-3 مرحله سوم: ساخت TreeBitmap

در این مرحله باید رشته بیتی به همراه جدول nextHop ها ایجاد شود، تفاوتی که در این مرحله وجود دارد این هست که ما دیتا مدل را عوض کردیم و دیگر باید با دیتا مدل جدید کار کنیم که PCTrie2Modle که ابتدا این را بررسی میکنیم:

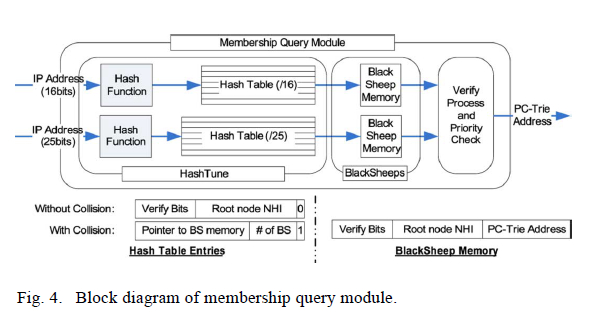
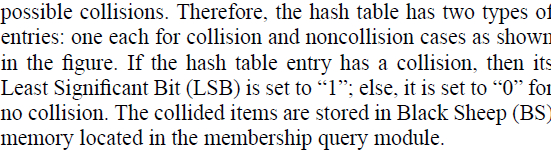
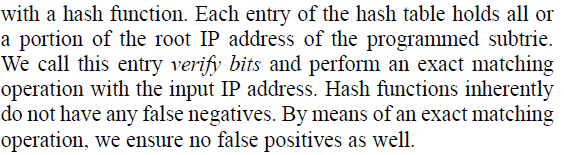
در این دیتا مدل ما یک ارایه دوبعدی داریم که برحسب که 1100 سطر به دلیل نصف بودن تعداد TreeBitmap ها سطح واقعی IP و دو ستون که نشان دهنده هر nodeSet می باشد و همچنین یک bitmap که رشته صفرو یک ها می باشد .

نکته ای که وجو دارد این هست که اگر subTrie فقط از یک ریشه تشکیل شده باشد و هیچ گره دیگری در ان subTrie وجود نداشته باشد در طی مراحل سوم اون گره حذف می شود به همین دلیل ما این جا یک فیلد root قرار دادیم که ریشه هر subTrie را قرار می دهد تا پس از PCTrie شدن گم نشود.

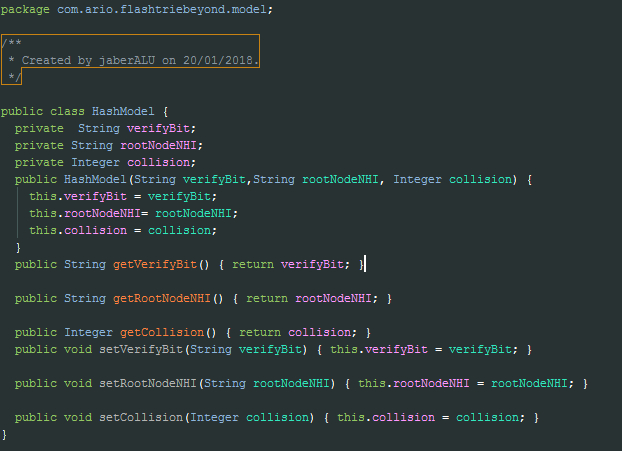
حالا به بررسی عملیات PCTrie در مرحله سوم می پردازیم:

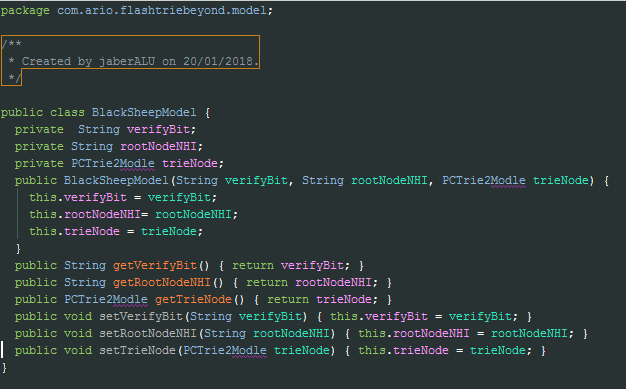
در این تابع اگر گره ای فرزند چپ و راست داشته باشد که هر دو nextHop داشته باشند به ترتیب آن ها را در فیلد ارایه دوبعدی childeren ذخیره می کنیم با استفاده از شمارنده های i,j و همچنین در فیلد bitmap 1 می گذاریم، اگر مخالف این حالت وجود داشته باشد فقط در bitmap صفر قرار می دهیم

1. تابع هشینگ

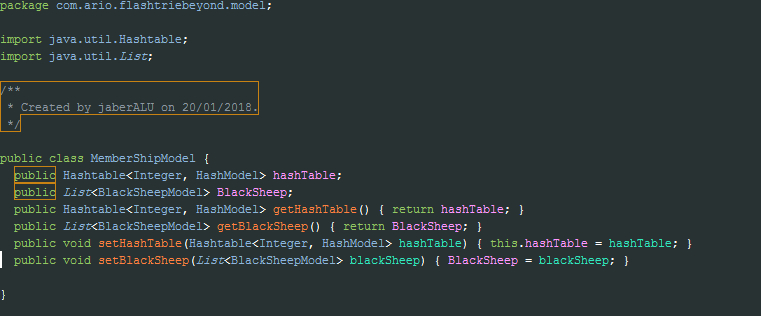
تا به اینجای کار ما PCTrie را انجام دادیم، و حالا باید عملیات هشینگ و سپس lookup را انجام دهیم بر طبق چیزی که مقاله گفته شده ما باید آدرس ریشه هر subTrie کلش یا بخش از آن را به عنوان کلید به تابع هش بدهیم و ادرس ریشه را بر اساس خروجی تابع هس درون جدول هش بر اساس ساختار سطر که گفته است ذخیره کنیم چنانچه تابع هش خروجی داده است که قبلا با ان کلید سطری در جدول هش وجو داشته باشد باید آن سطر جدید را در blackSheep مموری با ساختار سطر جدید دخیره بکنیم.

بر طبق تصویر موجود ما دو نوع سطر در جدول داریم که یکی verify Bits و Rootnode NHI و 0 و دیگری pointer BS و 1 موقعه ای که برخوردی رخ دهد ما باید درون سطر جدول هش سطر با بیت 1 قرار دهیم وآن را به BlackSheep ببریم و که انجا هم ساختار خودش را دارد و verifyBit و Root node NHI و آدرس PCTrie به طور مستقیم قرار دارد. ما برای هر سطح این جدول ها را داریم

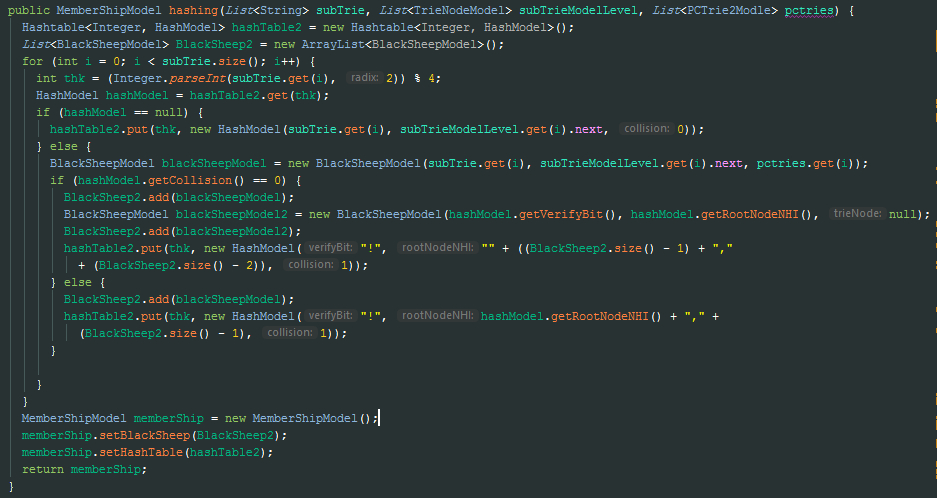
برای اینکه عملیات هشینگ به طور واضح تر انجام شود ابتدا دیتا مدل مورد استفاد را بررسی کنیم

دیتا مدلی که برای هر سطر جدول هش استفاده می شود، و دقیقا همان فیلد هایی هست که در مقاله گفته شده است به همراهsetter و getter مربوط به آن ها.

این هم دیتا مدل مربوط به blackSheep تنها فرقی که دارد این که فیلدی مستقیم به PCTrie اشاره می کند .

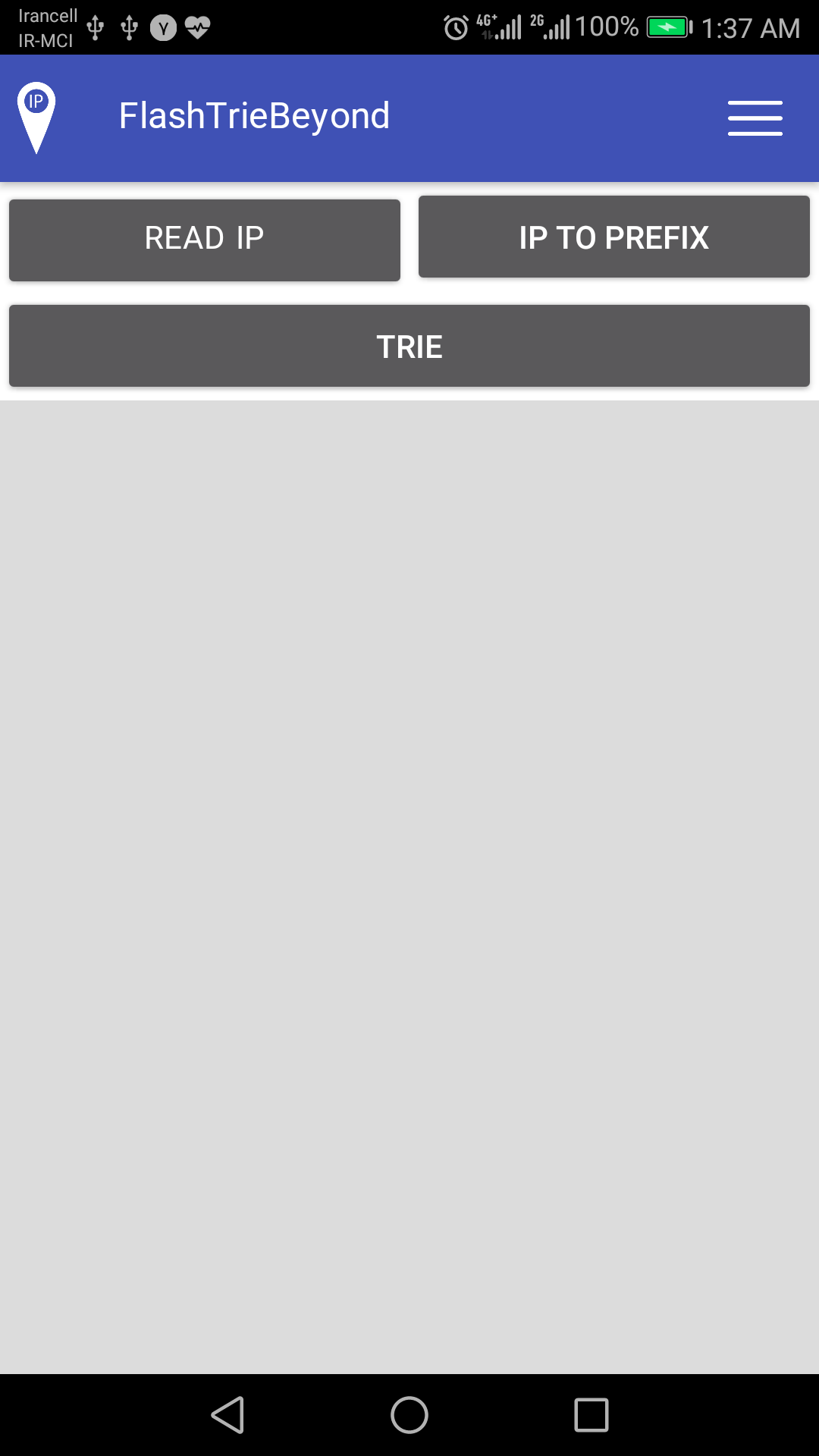


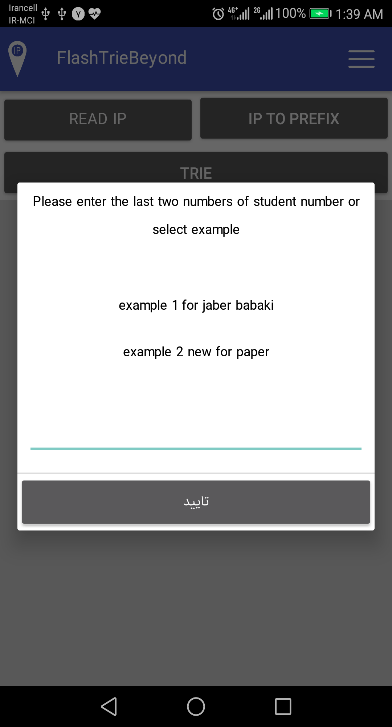
این دیتا مدل memberShip در واقع همه hashTable و BlackSheep ها را درون خود نگه می دارد،

حالا که با دیتا مدل آشنا شدیم می پردازیم به عملکرد تابع .

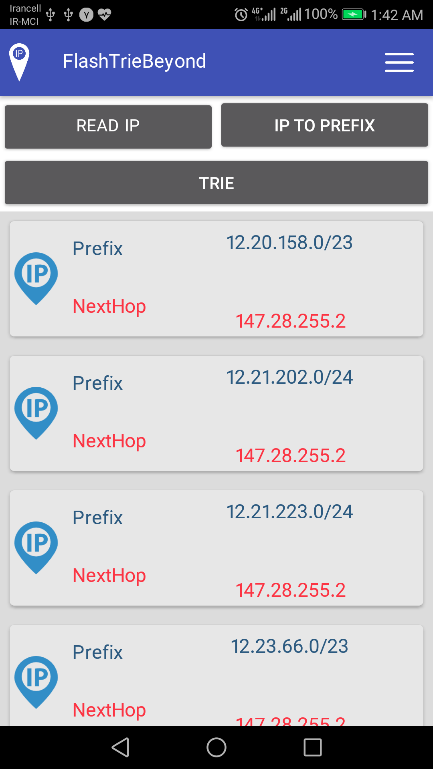
در اینجا ما می اییم طبق چیزی که گفتیم عمل می کنیم یعنی ریشه هر suntrie را با تابع هش ساده ی K mod M کلیدی را بدست می آوریم، وجای اون کلید در hashTable بررسی می کنیم اگر قبلا داده ای بوده پس این داده جدید به همراه داده قبلی باید وارد blackSheep شوند و در hashtable اشاره گری به اون وجود داشته باشد که من در اینجا اندیس list را ذخیره می کنم به عنوان اشاره گر به blackSheep و در نهایت این HshTable و List از جنس BlackSheep به عنوان MemebrShip برمی گردانم.

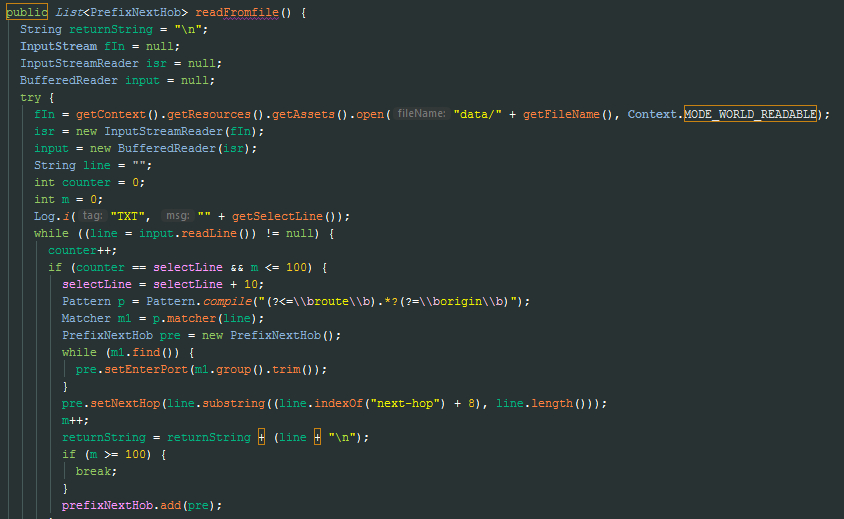
انجام عملیات lookup



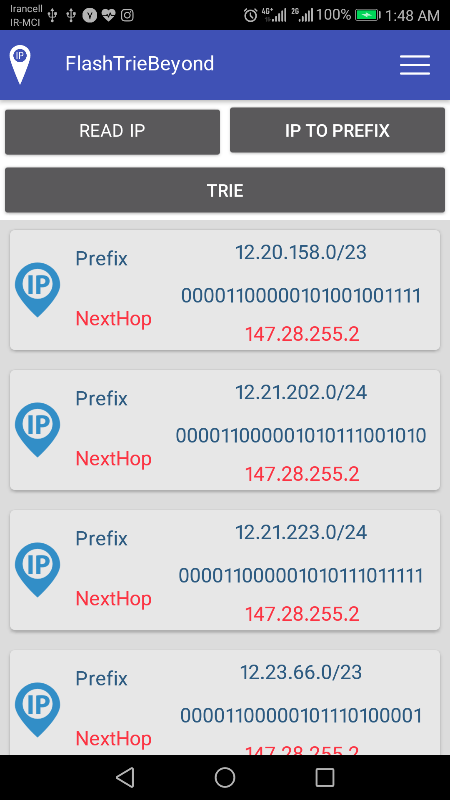
 در ابتدا که برنامه اجرا می شود عملیاتی انجام نمیشود و صفحه به صورت بالا هست، سپس با انتخاب گزینه Read Ip صفحه برای انتخاب IP باز می شود.

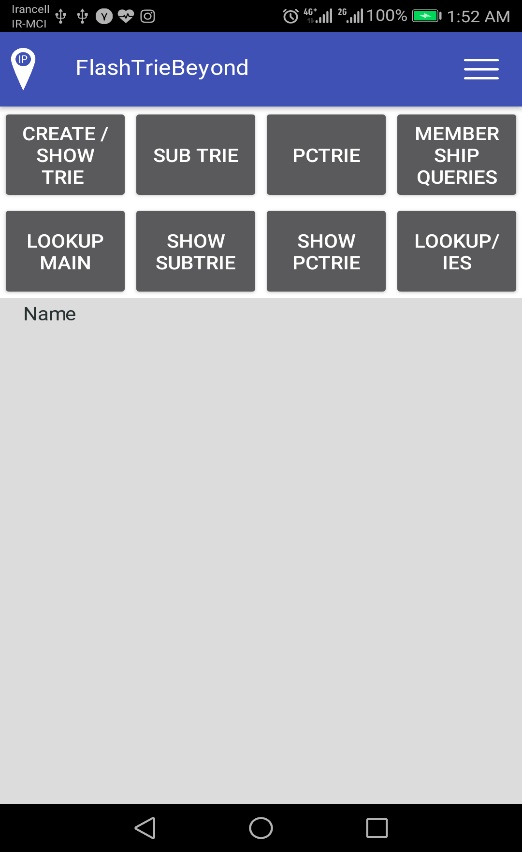
در اینجا می توان مثال مقاله را انتخاب کرد یا مثال که خودم زدم و یا اینکه شماره دانشجویی بدهیم و بر اساس فرمول گفته شده 100 تا خط انتخا و prefix و nextHop ها لیست شود.

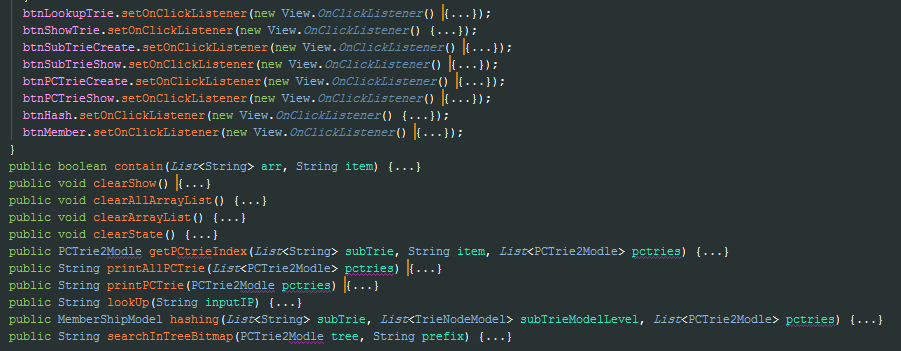


با وارد کردن عدد 20 به عنوان دو شماره آخر دانشجویی لیست زیر انتخاب می شود یعنی تابع readfromFile فراخوانی می شود که عملکزدش به شرح زیر است

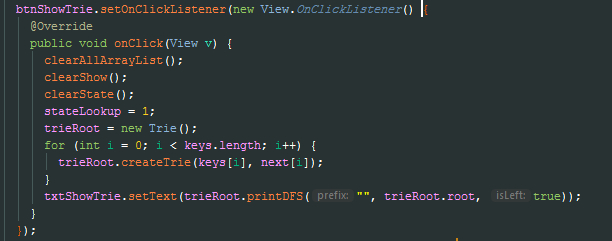
همانطور که نشان داده می شود تابع خواندن فایل که خط به خط خوانده و برا اساس الگویی که دادم 100 خط خوانده و درون لیست قرار می دهد و سپس نمایش می دهد.

سپس باید گزینه IpToPrefix را بزنیم تا IP ها به Prefix ها تبدیل شوند و در شکل زیر نشان داده می شود

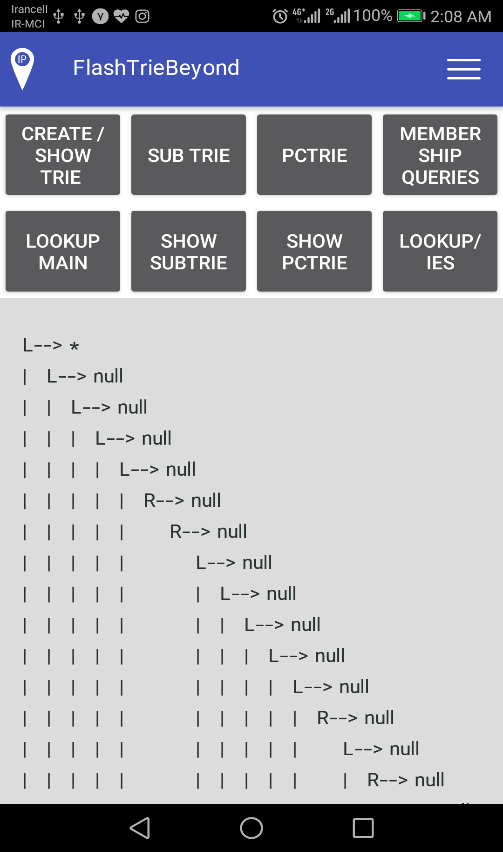
حال می توان با زدن گزینه Trie وارد صفحه Trie شد که به شکل زیر می باشد .

کد هایی که در این صفحه وجود دارد به این صورت هست

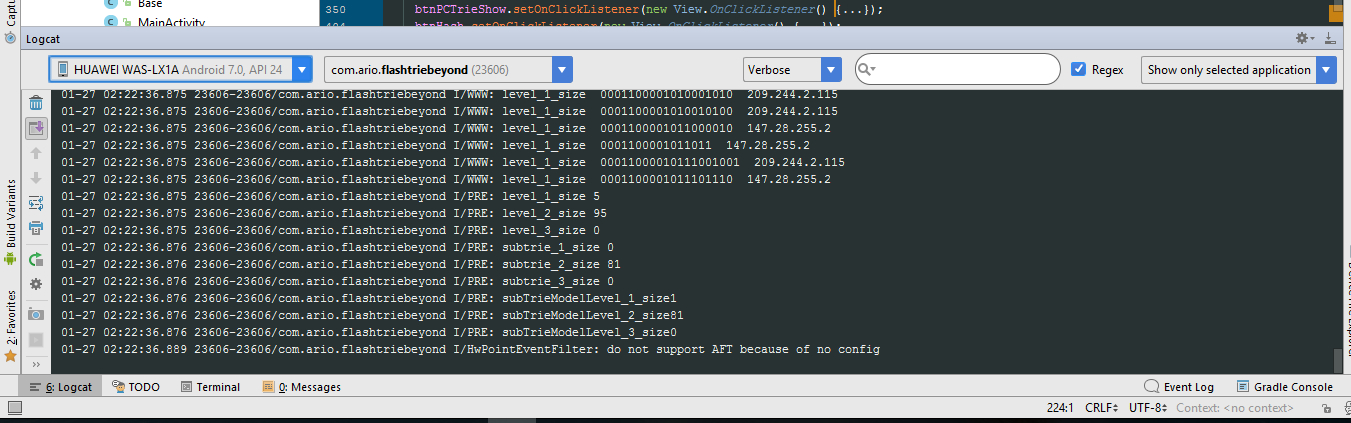
که مشاهده می شود هر دکمه برای خورش اکشنی دارد و به همراه یکسری توابع کاردی .

موقعه ای که کاربر بروی دکمه CreateTrie انتخاب می کند اکشن آن که به صورت زیر است اجر می شود:

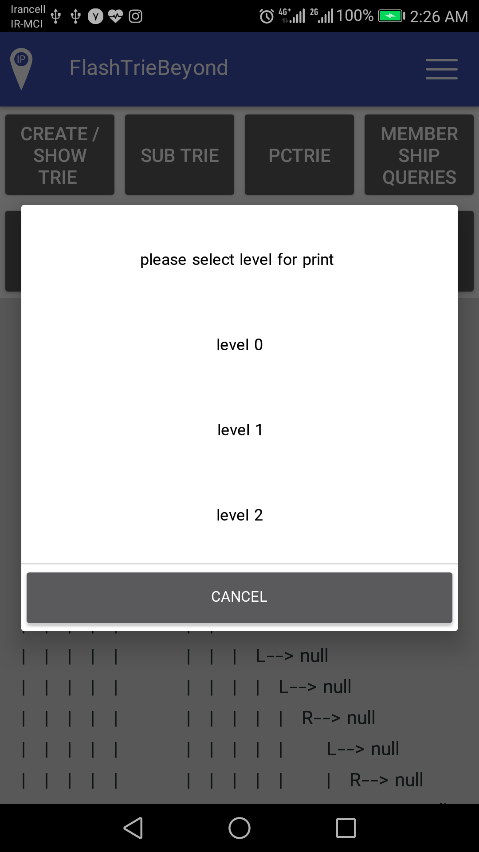
همانطور که دیده می شود بر اساس طول آرایه prefix ها حلقه میچرخد و تابع createTrie که بالاتر فراخوانی کردیم ساخته میشود و همه این ها درون شی trieRoot هست و در نهایت با استفاده از printDFS نمایش داده می شود که به صورت زی است .

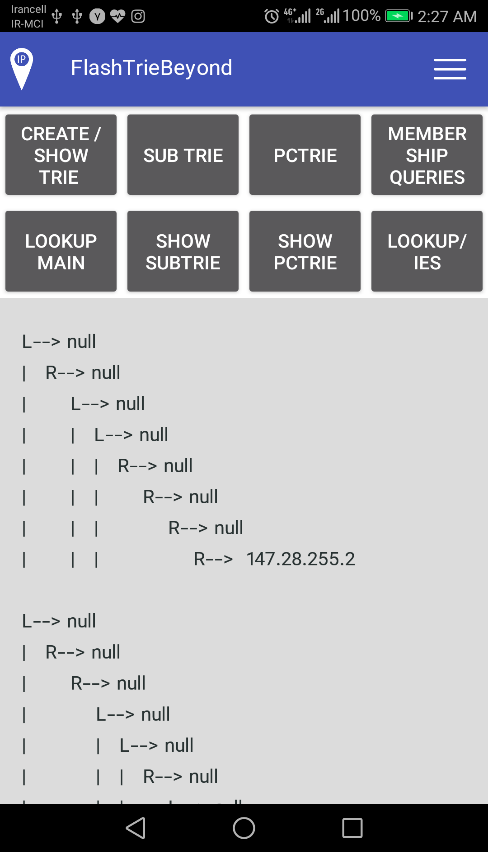


که میتوان اسکرول کرد و کل درخت را ببینیم ، در مرحله بعد subTrie باید subtrie بسازیم ایده ای که برای ساخت subTrie استفاده کردم به این صورت می باشد که درخت را هر بار تا یک سطحی بسازیم و از انجایی می دانیم هر سطح چند بیت است میتوانیم جستجویی تا اول هر سطح بزنیم و گره موردنظر که حاوی زیردرخت های خود است را بگیریم و درون لیست سطح دوم قرار دهیم . به صورت زیر :

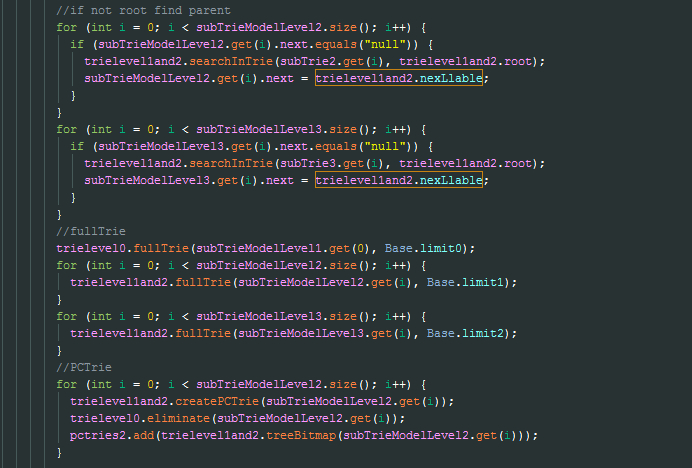
همونطور که مشخص است درخت را تا سطح مورد نطر می سازیم و جستجویی تا ابتدای اون سطح که میخواییم برش دهیم بر اساس بیت ها جدا میکنیم و درون لیست subTriModelLevel قرار می دهیم بر اساسی چیزه که در مقاله گفته باید در لیستی جدا نگه داری شوند.

این خروجی لاگ محیط اندروید استدیو هست که نشان می دهد در هر سطح ما چند تا subTrie داریم.

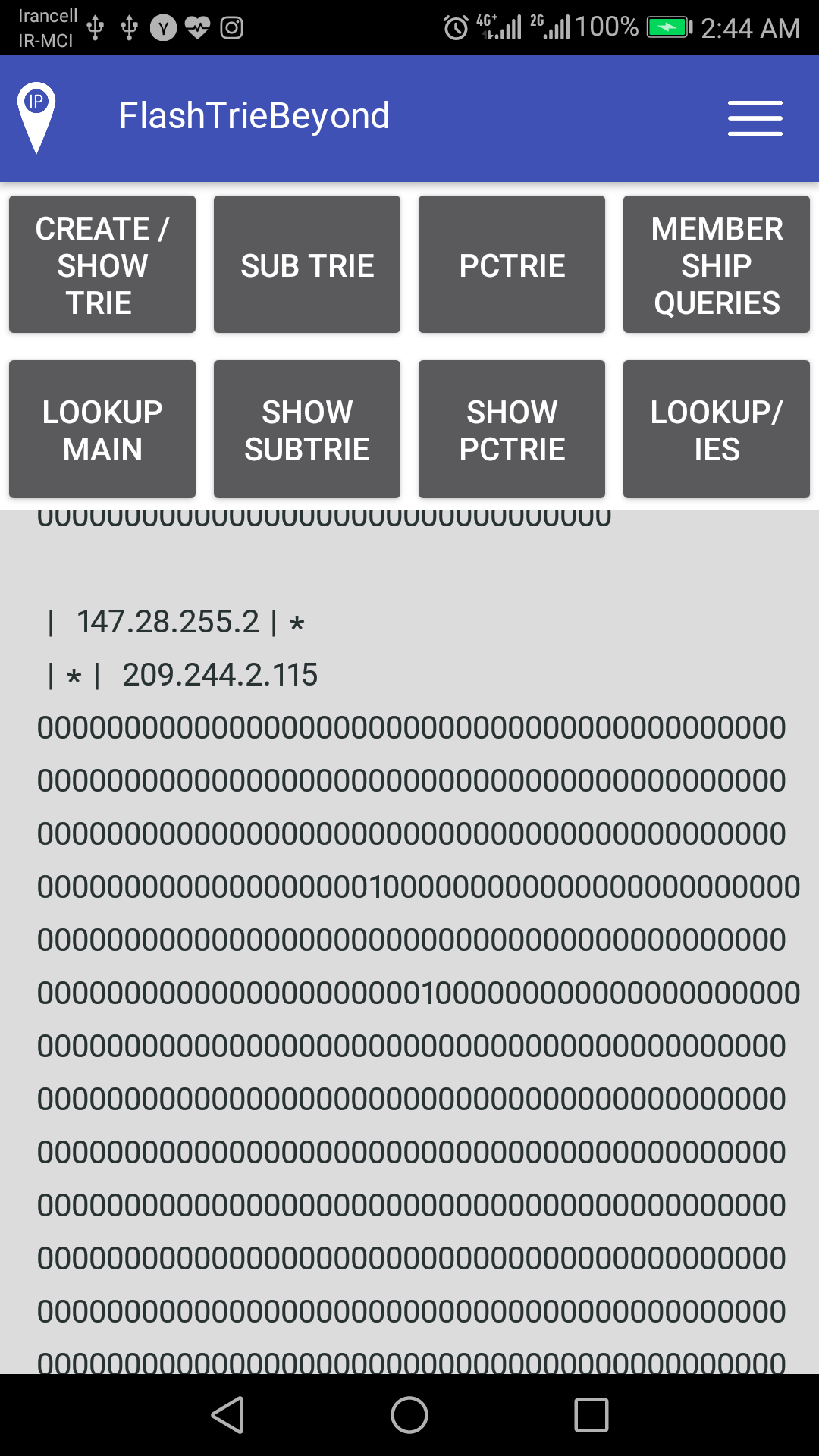
حال اکر subTrieShow را انتخاب کنیم ابتدا باید سطحی که میخواهیم subTrie هاش نمایش داده شود را انتخاب کنیم.

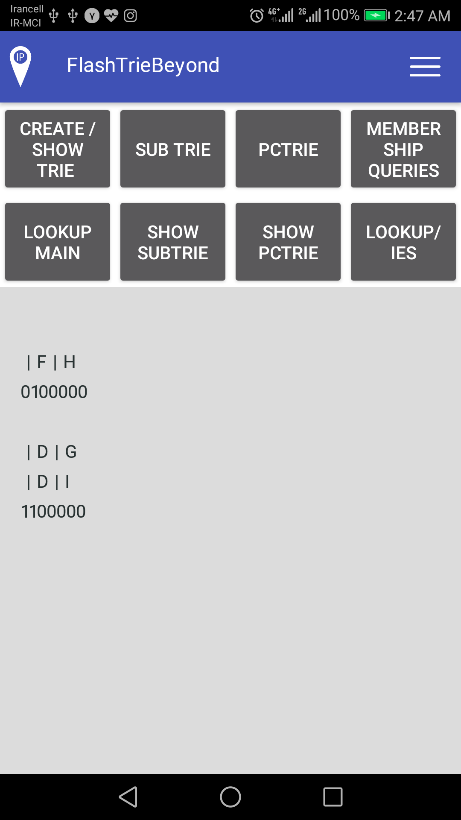
با فرض انتخاب سطح اول هر subTrie را به تابع PrintDfs میدهم و خروجی به صورت زیر است.

حال باید عملیات Pctrie بروی هر subTrie انجام شود با انتخاب گزینه create PCTrie اکشن زیر انجام می شود.



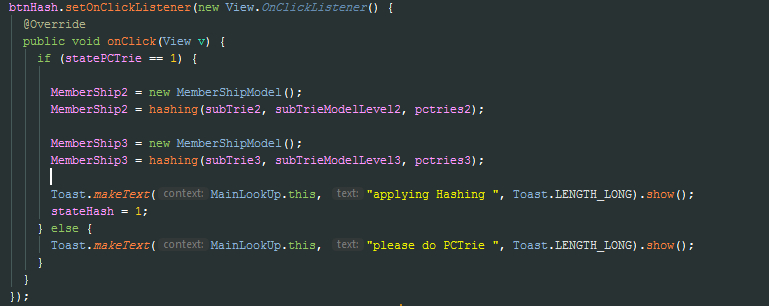
در اکشن PCTrie سه تا کار انجام می شود، ابتدا برای بر طبق گفته مقاله باید برای subTrie هایی که پدر ندارند نزدیک ترین پدر را پیدا کنیم که همونطور قبلا گفته شد تابع searchInTrie این کار را انجام می دهد برای هر subtrie و در هر سطحی باید این تابع فراخوانی شود تا هم subTrie ها پدر دار شوند

سپس عملیات کامل کردن هر subTrie انجام شود و در نهایت اون سه مرحله Pctrie برای هر subTrie فراخوانی می شود و درون لیست pctrie ها قرار می گیرد. حالا اگر نمایش PCTrie را بزنیم و یکی از سطوح را انتخاب کنیم شکل زیر ار مشاهده می کنیم :

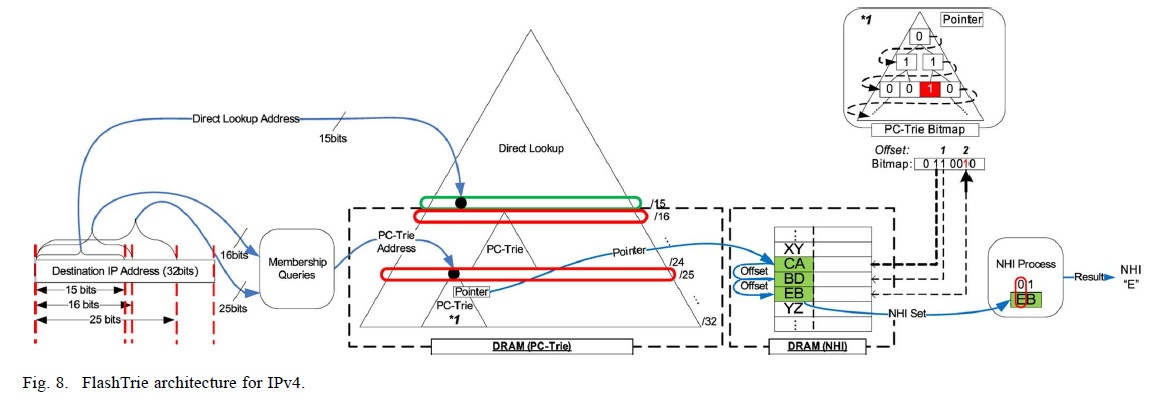
همانطور که دیده می شود به جایی استفاده از 2048 بیت با 1024 بییت توانستیم درخت را نمایش دهیم این فشرده سازی با سه بیت جداسازی و مثالی که مقاله زده است بهتر دیده می شود شکل زیر را ببینید:

شکل بالا PCTrie مثال درون مقاله هست که با در حالی که با 15 بیت TreeBitmap معمولی را نمایش میدادند در این مقاله و با استفاده از این ایده توانستیم با هفت بیت درخت را نمایش دهیم .

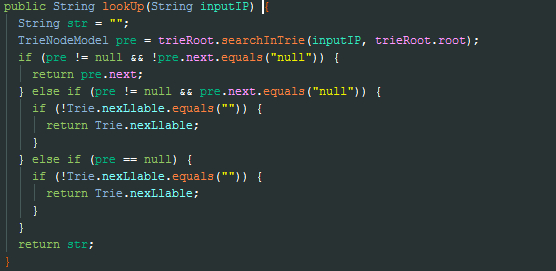
حالا باید عملیات memberQuery و lookup را انجام دهیم،

موقع ای که کاربر دکمه memberSheep را می زند اکشن زیر اتفاق می افتد.

برای هر سطح تابع hashing صدا زده می شود و خروجی memberQuery برمی گرداند که حاوی جدول هش و BlackSheep هست عملیات hashing بالاتر مفصل توضیح داده شده است.

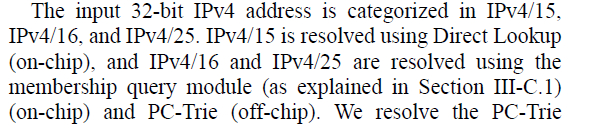
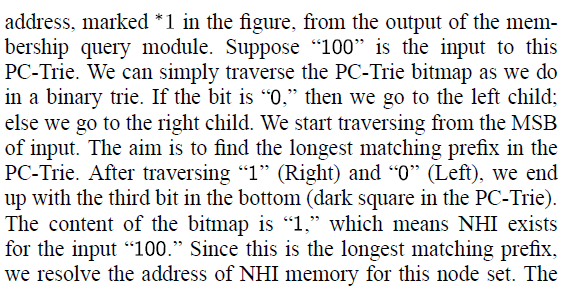
و حالا عملیات lookup موقعه ای که عملیات lookup زده می شود بر اساس تصویر زیر سه حالت ممکن است رخ دهد

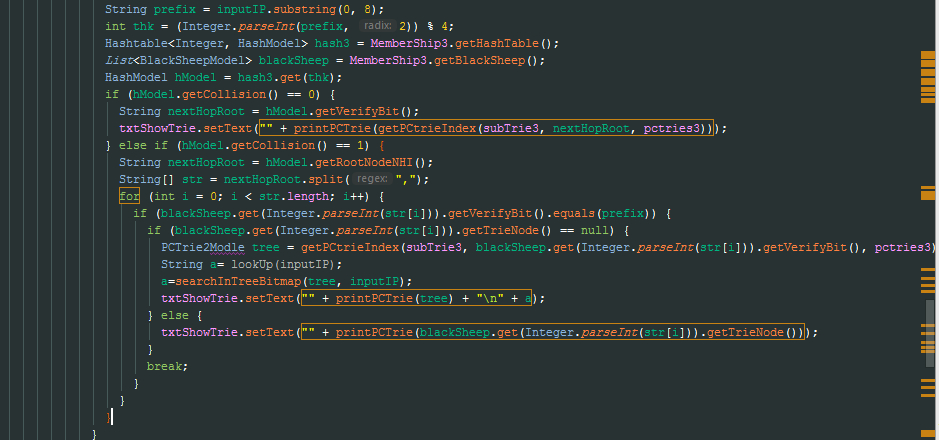
حالت اول اگر subnet 15 بیت یا کمتر باشد در on-Chip جستجو انجام می شود و خروجی داده می شود که کدش را من به صورت زیر نوشتم



که یک سرچ معمولی هست و من IP ورودی را به Prefix تبدیل و بر اساس subnet جدا کرده و به تابع بالا می دهم و تابع searchInTrie برای سطح اول فراخوانی و حالت های مختلف نتیجه بررسی و خروجی داده می شود

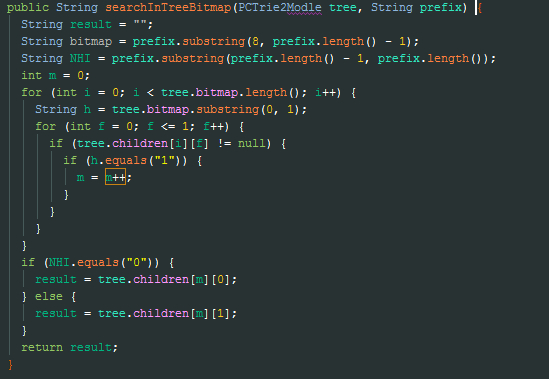
اما برای حالت های 15 تا 24 بیت و 25 تا 32 بیت بیاد روند زیر انجام شود:

 ابتدا باید بر اساس subnet برش داده شود و سپس به تابع هش داده شود خروجی تابع هش را در hashtable جستجو کنیم وسطر مورد نظر را پیدا کنیم اگر بیت اخر سطر یا همون collision برابر صفر بود حالا باید بگردیم بر اساس verify مورد نظر Pctrie متناظر را پیدا کنیم و سپس درون PCTrie مورد نظر ابتدا Bitmap را بررسی کنیم تا ببینیم در محل مورد نظر 1 وجو دارد یا صفر اگر یک بود حالا باید آرایه دوبعدی من نظر ب با ااندیس تعداد یک هایی که در Bitmap رد کردیم جستجو کنیم و بر اساس صفر یا یک بودن بیت کم ارزش محتوا خانه 0 یا 1 آرایه را به عنوان خروجی بدهیم اگر collision برابر یک بود باید برویم در BlackSheep بگردیم و در اونجا خوشبختانه آدرس PCTrie را نگه داشتیم و مستقیم باید همان عملیات جستجو درون Pctrie را انجام بدهیم .

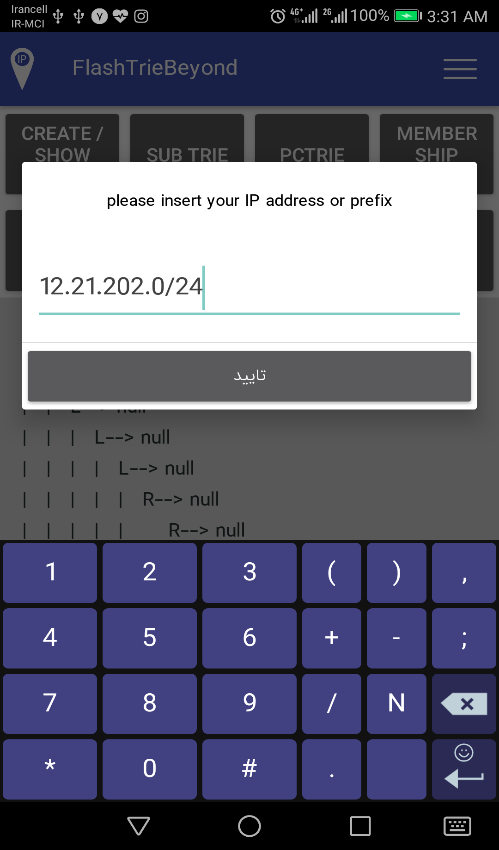


این عملیات برای بررسی کردن برخورد و در نهایت بدست آوردن آدرس PCTrie می باشد.

در ابتدا prefix به تابع هش داده می شود و بر اساس خروجی getCollision بررسی میشود اگر صفر بود فقط کافیه با getVerifyBit بتوانیم index اون PCTrie را بدست بیاوریم که این کار توسط تابع getPCTrieIndex انجام می شود

اگر برخورد داشت حالا باید بریم BlackSheep جستجو کنیم ولی از اونجایی که اندیس جاهایی که در BlackSheep باید بگردیم را درون getRootNodeNHI قرار دادیم فقط همان خونه ها در BlackSheep را جستجو می کنیم و با یه گام کمتر به PCTrie می رسیم حالا باید در PCTrie عملیات جستجو را انجام دهیم .

درون PCTrie مورد نظر ابتدا Bitmap را بررسی کنیم تا ببینیم در محل مورد نظر 1 وجو دارد یا صفر اگر یک بود حالا باید آرایه دوبعدی من نظر ب با ااندیس تعداد یک هایی که در Bitmap رد کردیم جستجو کنیم و بر اساس صفر یا یک بودن بیت کم ارزش محتوا خانه 0 یا 1 آرایه را به عنوان خروجی و در نهایت nextHop مورد نظر یافت می شود

خروجی این مرحله به صورت زیر است :