**RedBus**

To fetch list of stations of RefBus one has to browse through the DOM elements of [www.redbus.in](http://www.redbus.in/).

**Install**

1)If in linux download and install oracle java

2)Install neo4j in your system.(<http://www.neo4j.org/download>)

sudo apt-get install neo4j

3)Install your favourite IDE

4)Downlaod all the following jar files and add it to your IDE Library

Apache Commons and httpclient jars

==========================================

commons-codec-1.6

commons-io-2.4

commons-logging-1.1.1

fluent-hc-4.2.5

httpclient-4.2.5

httpclient-cache-4.2.5

httpcore-4.2.4

httpmime-4.2.5

Link:<http://hc.apache.org/downloads.cgi> download httpclient

<http://commons.apache.org/proper/commons-io/download_io.cgi>

<http://commons.apache.org/proper/commons-codec/download_codec.cgi>

Jackson for JSoN jars

==========================================

jackson-annotations-2.2.0

jackson-core-2.2.0

jackson-databind-2.2.0

Link:choose latest ones from all the three folder http://repo1.maven.org/maven2/com/fasterxml/jackson/core/

Neo4j Jars (present in neo4j download directory -> lib)

==========================================

use all the jars present.

Link:<http://docs.neo4j.org/chunked/stable/tutorials-java-embedded-setup.html>

**How to run the program**

**buses.Populater.java** is the main thread of execution.To run the program the following information is to be provided:

Populater(String complete, String uncomplete, String db\_path,String downloadLocation,**int** threadSleep)

Initially there will be two .txt files. One contain the list of stations that are already downloaded and populated in the database(“completed\_station.txt” in the code).The location of the file is passed through **complete** string.

The **uncomplete** parameter will have the location of file which contains new stations name and code that need to be downlaoded and populated in the database.

Both the files should be strictly in a particular format

1)Mumbai Station Name

2)210 Staion id

3) BlankLine

4)Goa StationName

5)20 Station id

**db\_path** is the path of folder where neo4j database will be created.If the folder exist it will use the existing one or will create a new one.

**DownloadLocation** Location of the folder where the downloaded data will be stored

**threadSleep** number of milliseconds for a thread to sleep before donwloading a new request.

**NB**:Before staring populating the neo4j database please make sure that the neo4j-service is shutdown.

In linux: sudo service neo4j-service stop

**public** **void** populate(String date) {}

This function(in populater.java) should be called to start population the database.The **date** should be in format like “27-Apr-2014”,”03-March-2014”.The system will start querying for buses availability on this date.

So to get maximum number of results the date should be around 15 days ahead of the present system date.

The date for which the bus availability is made is logged in **log/date-log.txt**

**HOW TO GET ALL DATA IN .CSV FORMAT?**

To get data in .csv format run the **Converter.java** file.

**Working**

To conver the database to any format one just has to write a callback function called **parseRow.**This function will be called by the program with a paramaeter Map<String,Object> where keys will be the following: n1.id,n1.name,n2.id,n2.name,r.rtid,r.dept\_time,r.arr\_time,r.duration,r.fare0,r.rating,r.isAc,r.isNAc,r.isSlpr

**n1** denotes the starting station(origin node)

**n2** denotes the destination station(dest node)

**n1.id** city id

**n1.name** city name

**r** dentoes the bus between them

**rtid** is the busId

**r.fare0** is the minimum fare of all the fare available for the bus

**isAc** is AC?

**IsNAC** is Non AC?

**IsSlpr** is Sleeper?

**Rating** rating of the bus given by redbus out of 5.0

**WORKING OF THE PROGRAM**

1)**function:populate**After popoulate(in populater.java) is called, it will first download the data for all the Stations and store it in the *downlaodLocation* folder.

a)Since we have two list of stations.One for the station which are downloaded and populated and one which are yet to be downloaded.Let call the downloaded stations list as **completed** and new station list as **uncompleted**.

Now each station has to be checked with every other station for bus availability.So this will take O(N2) time.And each station will have to be queried over other stations.So the code implements two for loop (one instide other) to make the N2 query.So this is how the downloading goes:

outer forloop inner forloop

i)uncompleted uncompleted (each uncompleted station is queried with other uncompleted station for bus availabiltiy)

ii)uncompleted completed

iii)completed uncompleted

2)**function:downloadData**

Before downloading the file the program first checks if the file is already downloaded or not.The folder to be checked are “downloadLocation” and “downlaodLocation\_complete”.

The downloaded file is stored as “originId-originName-destId-destName.txt”.

3)**function:populateData**

Once all the data is downlaoded the system starts populating the neo4j database.Again the same for loop is run(same as while downloading it).

It searches for the file in the downloadLocation folder.If the file is present it parses it,format/edit the file if require,(Like in some of the json files received from redbus are not syntatically correct) and store it in database.Once the database is populated the file is moved to downloadLocation\_complete to avaoid populating the database again.

4)**class:Parser.java**

The data received by redbus is in json Format.To parse it the program uses light weight jackson library.(<http://wiki.fasterxml.com/JacksonInFiveMinutes>)

All the important data is extraced and stored as an object of **RedBuses class.**

5)**class:Buses.java**

This class contains all the request specific data one need to send to redbus to fetch information.It also have method which extract the Stations obect from station file.

**6)NeoDB**

Suppose there is a bus going from Goa(city id 210) to Mumbai city id(462).The bus has an rtid of '4669442',rating:4.1,fare:1200 etc.Goa has only one voarding point i.e Mandovi Bridge(id:2323) and Mumbai has many boarding points such as dadar,andheri,panvel.Suppose the bus arrived on mandovi bridge at 6:30 PM and reach Mumbai-Dadar at 09:00 AM,Andheri at 09:15 AM and Panvel at 10:00 AM.

Storing Time

Instead of storing arrival or departure time as 05:00 AM it is converted to total minutes since 12 AM. For instance

01:01 AM will be 61

01:01 PM will be 781

02:30 AM will be 150

This is how the database stores the information.We have

1)City Node Describe City

2)HAS\_BUS Relationship Describe bus properties and connect two cities

3)Point Node Describe Boarding and Departure points

3)To Relationship connect two points and contain the bus arrival time information

Node1:

|  |  |
| --- | --- |
| id | 210 |
| name | Goa |

Node2:

|  |  |
| --- | --- |
| id | 462 |
| name | Mumbai |

r(HAS\_BUS edge):

|  |  |
| --- | --- |
| rtid | 4669442 |
| duration | 690 |
| dept\_time | 1110 |
| first\_dp\_time | 540 |
| arr\_time | 360 |
| isAc | true |
| first\_bp\_id | 429940 |
| rating | 4.1 |
| first\_bp\_time | 1110 |
| fare0 | 1200 |
| isNAc | false |
| isSlpr | false |
| first\_dp\_id | 429634 |
| fare1 | 1250 |

**NEODB Representation :**(Node1:City) -[r:HAS\_BUS] -> (Node2:City)

//Node1 connected to Node 2 by bus r having properties described above.

first\_bp\_id is the boarding point id of the Point where the bus stops first at origin Station.In above case its id of Mandavi brodge Point.

first\_bp\_time is the time when the bus reach the first boarding point.

Same for departure(dp)

Point1:

**id:429940**

**Name:Mandovi Bridge**

Point2:

**id:429634**

**name:Dadar**

Point3:

**id:429639**

**name:Andheri**

Point4:

**id:429636**

**name:Panvel**

a(:TO)

**rtid:4669442**

**start\_time:*540***

**end\_time:555**

start\_time is when the bus starts from that point and end\_time is when the bus reaches the next point

b(:TO)

**rtid:4669442**

**start\_time:555**

**end\_time:600**

**NEODB Representation :** (Point2:Point) – [a:TO] -> (Point3:Point) – [b:TO] -> (Point4:Point)

We use cypher queries to construct the database.For all buses from one city to another a single query is constructed with all the informationa and is then executed to populated the database

NOTE:The program will ensure that the same data is not being downloaded again.But the same data can be populated again in the database.So make sure to avoid duplication of data.Try populating the data only when all the data is downloaded.

Some problems I encountered and asked solution is stackoverflow:

<http://stackoverflow.com/questions/21508297/cypher-query-to-get-duplicatesame-id-relationship-between-nodes>

to start neo4j-service to view it in browser you should make this command:

sudo chown all -R neo4j-Database-Location (all here is neo4j group one costruct while installing it.It can have any name)

The program should have the read and write access for where the neo4j database folder is located

**BusIndia**

Install

same as for BusIndia excpt

Download Jsoup Jar file

Link:<http://jsoup.org/download>

**How to run the program**

Busindia.java is the main execution file.

Creating object Busindia(boolean stationExist,String db\_path,String pathToConfig)

**stationExist**: If we have already downlaoded the station list then no need to download it again and it should be set as true.If we want to download station list again set it to false.The downloaded file will be stored as stationlist.json

**db\_path** : path to neo4j database.

**PathToConfig**:Location of folder which contains neo4j.properties file.This will help to configure neo4j before running it if required,otherwise set it as null.

Calling start method will start the process of donwloading and populating the files.

start(String stationFile,boolean downloadData,boolean populateData,int threads,String checkInDate,String checkOutDate,String journeyDate,String maxValidreserve,String folderLocataion)

stationFile : location of json file which contains all the station which are needed to be uploaded

downloadData: whether the data is already downloaded or need to be downloaded.setting it to true will downlaod the data

populateData: whether the program needs to populate the data now ? If true,it will start populating database

threads:total number of threads pinging busindia at a time to download bus\_availability information.(3 is recommended)

checkInDate : put the present date. Format dd/mm/yyyy

checkOutDate: checkInDate + 1. if check in is 23/02/2013 checkout is 24/02/2013

journeyDate: date for which bus is queried between two station.(Recommended 15 days from the current date)

maxValidreserve: 1 month more than the checkInDate. Probably the last day for which ticket can be booked.

FolderLocataion:Location where the data need to be downloaded.

**WORKING OF THE PROGRAM**

The working of both the busindia and redbus code is almost the same.Some of the differences are:

Busindia has 10000 buses.Downlaoding this data will take a lot of time.So instead of downloading all the data , StationExtractor.java extract all the important stations from busindia stationlist.

**NeoDb data structure**

For BusIndia, the code is not downlaoding the boarding and departure points.As downloading them require first parsing the main file and then based on the id downloading the boarding points.This will increase the program time too.

Therefore there is no To relationship and Point node.The database contains :

node:City node denoting city in s graph

relationship:HAS\_BUS edge connecting two cities in a graph.

City node eg:

id:410

name:mumbai

HAS\_BUS relationship eg:

|  |  |
| --- | --- |
| radOServiceID | 5671,2,GURJARNAGRI,4,42,13:55,21:00,EXPRESS,GSRTC,Y,2100PNAABDGJR47,26/04/2014,10:55:00,2,3529,172,S0 |
| duration | 835 |
| dept\_time | 1260 |
| rtid | 2100PNAABDGJR47 |
| fare | 496 |
| arr\_time | 655 |
| via | SINNAR,NSK,DHARAMPUR,VLD,SRT, |
| busClass | GURJARNAGRI |
| corporation | GSRTC |
| available\_seat | 42 |

RadOServiceID is used to query for boarding points and seat availabiltu of particular dare and that's why is stored if in future we decide to download the boarding and departure points.

**Parsing-**Parser.java

Unlike the redbus where the program is downloading the json file here we are downloading HTML file.So to extract data from html we are using a html scrapper call Jsoup.

The required data are in class=”Products” in the HTML file.So parsing accordingly fetch us the data.