1. 前提条件

以下のツールがWindows10にインストールされている必要があります。

- Docker for windows
- Redis
- Go言語

2. 山口市の地図データのダウンロード

ダウンロードの必要はありません。(to-path)/yamaguchi/yama_dbの中にあるyamaguchi.zipを解凍してyamaguchi.osmを使用して下さい。

3. ディレクトリの作成

- gitによる環境の取得

```
$ git cline https://github.com/C-Anemone/yamaguchi
```

で、環境を作って下さい。

- (to-path)/yamaguchi/yama_dbの中にあるyamaguchi.zipを解凍してyamaguchi.osmを使用して下さい。

4. postgreSQL用のdockerコンテナの生成

- **[Step1]** dockerを起動した後、(to-path)/yamaguchiの中で、"docker-compose build, ""docker-compose up -d"を実行して下さい。

```
C:\Users\ebata\yamaguchi>docker-compose build
```

```
C:\Users\ebata\yamaguchi>docker-compose up -d
Creating network "yamaguchi_default" with the default driver
Creating volume "yamaguchi_db-data" with default driver
Building db
[+] Building 5.1s (16/16) FINISHED
 => [internal] load build definition from Dockerfile
0.3s
 => => transferring dockerfile: 1.02kB
0.0s
 => [internal] load .dockerignore
0.2s
 => => transferring context: 2B
0.0s
 => [internal] load metadata for docker.io/pgrouting/pgrouting:12-3.0-master
4.6s
 => [auth] pgrouting/pgrouting:pull token for registry-1.docker.io
0.0s
 => [ 1/11] FROM docker.io/pgrouting/pgrouting:12-3.0-
master@sha256:748f50655751b87ec3ab9951f0c8cbed3b3a1040f4258  0.0s
 => CACHED [ 2/11] RUN apt update
0.0s
 => CACHED [ 3/11] RUN apt install -y libpqxx-6.2
0.0s
 => CACHED [ 4/11] RUN apt install -y build-essential cmake wget        libboost-
program-options-dev libexpat1     0.0s
 => CACHED [ 5/11] RUN cd /usr/local/src
0.0s
 => CACHED [ 6/11] RUN wget
https://github.com/pgRouting/osm2pgrouting/archive/v2.3.6.tar.gz
0.0s
 => CACHED [ 7/11] RUN tar xvf v2.3.6.tar.gz
0.0s
 => CACHED [ 8/11] RUN cd osm2pgrouting-2.3.6  && mkdir build  && cd build  &&
cmake ..  && make  && make install  0.0s
 => CACHED [ 9/11] RUN apt purge -y --autoremove         build-essential
cmake         wget         libex  0.0s
 => CACHED [10/11] RUN apt autoremove -y
0.0s
 => CACHED [11/11] RUN rm -rf /var/lib/apt/lists/*
0.0s
 => exporting to image
0.2s
 => => exporting layers
0.0s
 => => writing image
sha256:cce75fdf5dae133107d15f6b29752f9c62a446a46977a5f7ddd6739ecfa74909
0.0s
 => => naming to docker.io/library/yamaguchi_db
0.0s
WARNING: Image for service db was built because it did not already exist. To
rebuild this image you must use `docker-compose build` or `docker-compose up --
build`.
Creating yamaguchi_db_1        ... done
Creating yamaguchi_osm2pgsql_1 ... done
```

- **[Step2]** dockerコンテナを再起動します(不要かもしれないけど年の為)

```
C:\Users\ebata\yamaguchi>docker-compose restart
Restarting yamaguchi_db_1        ... done
Restarting yamaguchi_osm2pgsql_1 ... done
```

- **[Step3]** dockerコンテナの中にログインします

```
C:\Users\ebata\yamaguchi>docker container exec -it yamaguchi_db_1 bash

(もしかしたら"yamaguchi-db-1"となっているかもしれない)
root@d696dd7664af:/#
```

- **[Step4]** dockerコンテナの中から、psqlでpostgreSQLのコンソールに入る

```
root@d696dd7664af:/# psql -U postgres
psql (12.5 (Debian 12.5-1.pgdg100+1))
Type "help" for help.
```

- **[Step5]** yama_dbができていることを確認して下さい

```
postgres=# \l
                                 List of databases
   Name    |  Owner   | Encoding |  Collate   |   Ctype    |   Access privileges
-----------+----------+----------+------------+------------+-----------------------
-
 postgres  | postgres | UTF8     | en_US.utf8 | en_US.utf8 |
 template0 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres
+
           |          |          |            |            | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres
+
           |          |          |            |            | postgres=CTc/postgres
 yama_db   | postgres | UTF8     | en_US.utf8 | en_US.utf8 |
(4 rows)
```

- **[Step6]** yama_dbに接続して下さい

```
postgres=# \connect yama_db;
You are now connected to database "yama_db" as user "postgres".
```

- **[Step7]** 拡張機能(postgis)を作成して下さい

```
yama_db=# create extension postgis;
CREATE EXTENSION
```

- **[Step8]** 拡張機能(pgrouting)を作成して下さい。

```
yama_db=# create extension pgrouting;
CREATE EXTENSION
```

- **[Step9]** yama_dbの内容を確認して下さい。

```
yama_db=# \dt

              List of relations
 Schema |      Name       | Type  |  Owner
--------+-----------------+-------+----------
 public | spatial_ref_sys | table | postgres
(1 row)
```

- **[Step13]** postgresからログアウトして下さい。

```
yama_db-# \q
root@d696dd7664af:/#
```

5. データベース(PostgreSQL + postGIS)の起動方法と、利用方法と、終了方法

ここまで"(to-path)\yamaguchi"と記載してきましたが、多分、私の環境をそのまま記載した方が分かりやすいと思いますので、以下は、私の環境が使っているディレクトリ"c:\Users\ebata\yamaguchi"とそのまま記載します(Windows10の環境なので)が、あなたは、あなたの環境のディレクトリで(例えば、/home/yamaguchiでも、/usr/local/src/yamaguchiでもなんでも)使って下さい。

- **Step1** Dockerを起動する(起動方法は、各OS毎に異なります)
- **Step2** $cd c:\Users\ebata\yamaguchi
- **Step3** $ docker-compose restart

```
$ docker-compose restart
Restarting yamaguchi_db_1        ... done
Restarting yamaguchi_osm2pgsql_1 ... done
```

で、終了時は、シャットダウンしても特に問題はないと思いますが、キレイに終了する為には、アプリケーション(psqlやQGIS)を終了させた後、以下を実施して下さい。

- **Step4** $ docker-compose stop

docker-compose **down**とすると、dockerコンテナを作り直す必要がありますので、注意して下さい(私は、結構な頻度でやってしいます)。

6. yamaguchi.omsのデータベースへの読み込み

dockerコンテナの中にログインしていることを確認して下さい。

- **[Step1]** コンテナの中から書き込みを実施して下さい

```
root@d696dd7664af:/# osm2pgrouting -f /yama_db/yamaguchi.osm -c
/usr/local/share/osm2pgrouting/mapconfig_for_cars.xml -d yama_db -U postgre
```

書き込みが成功すると、以下のようなメッセージが出てきます。

```
Execution starts at: Mon Jun 20 12:15:20 2022

************************************************
           COMMAND LINE CONFIGURATION             *
************************************************
Filename = /yama_db/yamaguchi.osm
Configuration file = /usr/local/share/osm2pgrouting/mapconfig_for_cars.xml
host = localhost
port = 5432
dbname = yama_db
username = postgres
schema=
prefix =
suffix =
Don't drop tables
Don't create indexes
Don't add OSM nodes
************************************************
Testing database connection: yama_db
database connection successful: yama_db
Connecting to the database
connection success

Creating tables...
TABLE: ways_vertices_pgr created ... OK.
TABLE: ways created ... OK.
TABLE: pointsofinterest created ... OK.
TABLE: configuration created ... OK.
Opening configuration file: /usr/local/share/osm2pgrouting/mapconfig_for_cars.xml
    Parsing configuration
```

```
Exporting configuration ...
  - Done
Counting lines ...
wc: /yama_db/yamaguchi.osm: No such file or directory
File not found/yama_db/yamaguchi.osm
root@d696dd7664af:/# osm2pgrouting -f /yama_db/yamaguchi.osm -c
/usr/local/share/osm2pgrouting/mapconfig_for_cars.xml -d
 yama_db -U postgres
Execution starts at: Mon Jun 20 12:15:46 2022

************************************************
           COMMAND LINE CONFIGURATION             *
************************************************
Filename = /yama_db/yamaguchi.osm
Configuration file = /usr/local/share/osm2pgrouting/mapconfig_for_cars.xml
host = localhost
port = 5432
dbname = yama_db
username = postgres
schema=
prefix =
suffix =
Don't drop tables
Don't create indexes
Don't add OSM nodes
************************************************
Testing database connection: yama_db
database connection successful: yama_db
Connecting to the database
connection success

Creating tables...
TABLE: ways_vertices_pgr already exists.
TABLE: ways_vertices_pgr already exists.
TABLE: ways_vertices_pgr already exists.
TABLE: ways_vertices_pgr already exists.
Opening configuration file: /usr/local/share/osm2pgrouting/mapconfig_for_cars.xml
    Parsing configuration

Exporting configuration ...
  - Done
Counting lines ...
  - Done
Opening data file: /yama_db/yamaguchi.osm       total lines: 1300578
    Parsing data


End Of file


    Finish Parsing data

Adding auxiliary tables to database...
```

```
Export Ways ...
    Processing 71811 ways:
[*************|                                    ] (27%) Total processed: 20000
Vertices inserted: 10110      Split ways inserted 11838
[*************************|                        ] (55%) Total processed: 40000
Vertices inserted: 1219       Split ways inserted 1674
[****************************************|         ] (83%) Total processed: 60000
Vertices inserted: 501 Split ways inserted 760
[*********************************************|] (100%) Total processed:
71811      Vertices inserted: 1006       Split ways inserted 1421

Creating indexes ...

Processing Points of Interest ...
#########################
size of streets: 71811
Execution started at: Mon Jun 20 12:15:46 2022
Execution ended at:   Mon Jun 20 12:16:07 2022
Elapsed time: 21.35 Seconds.
User CPU time: -> 11.8897 seconds
#########################
root@d696dd7664af:/#
```

- **[Step2]** dockerコンテナの中から、postgresのコンソールに入る

```
root@d696dd7664af:/# psql -U postgres
psql (12.5 (Debian 12.5-1.pgdg100+1))
Type "help" for help.
```

- **[Step3]** yama_dbに接続して、中身を確認

```
postgres=# \c yama_db
You are now connected to database "yama_db" as user "postgres".
```

```
yama_db=# \dt
               List of relations
 Schema |       Name        | Type  |  Owner
--------+-------------------+-------+----------
 public | configuration     | table | postgres
 public | pointsofinterest  | table | postgres
 public | spatial_ref_sys   | table | postgres
 public | ways              | table | postgres
 public | ways_vertices_pgr | table | postgres
(5 rows)
```

- **[Step4]** yama_dbにyamaguchi.osmが格納されているのを確認して下さい。

```
yama_db=# select * from ways;
  gid  |   osm_id   | tag_id |         length         |      length_m      |
name         | source | target | so
urce_osm | target_osm |          cost          |      reverse_cost       |
cost_s        |    reverse_cost_s    |
 rule | one_way | oneway  |     x1      |     y1     |     x2      |     y2     |
maxspeed_forward | maxspeed_backward |
 priority |
                            the_geom

(中略)
     1 |  119875274 |    112 | 0.00018190464535143468 | 16.763517132871453 |
|    607 |    370 | 13
45406652 | 1345403555 | 0.00018190464535143468 |  0.00018190464535143468 |
1.2069732335667445 |   1.2069732335667445 |
      |       0 | UNKNOWN |  131.513154 | 34.2121682 | 131.5133359 | 34.2121695 |
50 |               50 |
      2.5 |
0102000020E61000000200000F8F9EFC16B7060402720DC53281B41409BCB683F6D706040B6DAC35E
281B4140
     2 |  119878462 |    106 |  0.0006487341597351198 |  69.73913991276467 | 山口
旭線             |   1290 |     74 | 13
45450448 |  827247622 |  0.0006487341597351198 |   0.0006487341597351198 |
2.7895655965105868 |   2.7895655965105868 |
      |       0 | UNKNOWN | 131.4773843 | 34.1861939 | 131.4770967 | 34.1867754 |
90 |               90 |
     1.15 |
0102000020E61000000200000F78370BB466F60407B9BA333D5174140F2704C60446F604042599D41
E8174140
     3 |  119875986 |    112 | 0.00014882019351711238 |  16.30415372755039 |
|   9899 |     51 | 93
24969629 |  749767421 | 0.00014882019351711238 |  0.00014882019351711238 |
1.173899068383628 |    1.173899068383628 |
      |       0 | UNKNOWN | 131.5135876 | 34.2047716 | 131.5135457 | 34.2046288 |
50 |               50 |
      2.5 |
0102000020E610000002000000323443 4F6F7060402693AEF4351A41400C5064F76E706040EDE8C946
311A4140
     4 |  432394449 |    101 |  0.0013109005330034158 | 120.99683553542219 | 中国
自動車道         |   7194 |      1 | 43
14592540 |  255182378 |  0.0013109005330034158 |  -0.0013109005330034158 |
3.3506815994424604 |  -3.3506815994424604 |
      |       1 | YES     | 131.3851168 | 34.1267699 |  131.386426 | 34.1267217 |
130 |               130 |
        1 |
0102000020E61000000300000007CA477E0526C6040004FFFFE39104140DD0DFD6E556C60409BD145F9
381041405F0A0F9A5D6C6040C5
(後略)
```

7. (付録)agend_dbを作る場合

dbにログインした後、

```
create database agent_db;
\c agent_db

create table user_list(
ID     int,
Age    int,
TYPE   varchar(10),
Departure_name   varchar(20),
Departure_number   int,
Departure_lat double precision,
Departure_lng double precision,
Arrival_name   varchar(20),
Arrival_number   int,
Arrival_lat double precision,
Arrival_lng double precision
);

 id | age | type | departure_name | departure_number | departure_lat |
departure_lng | arrival_name | arrival_number | arrival_lat | arrival_lng
----+-----+------+----------------+------------------+---------------+------------
---+--------------+---------------+-------------+-------------
(0 rows)
```

これで、以下のようなテーブルができる

```
agent_db=# \dt
           List of relations
 Schema |   Name    | Type  |  Owner
--------+-----------+-------+----------
 public | user_list | table | postgres
(1 row)
```

以下のプログラムを実行することで、テーブルにデータが入る

```
package main

import (
    "database/sql"
    "encoding/csv"
    "fmt"
    "log"
    "os"
```

```
    "strconv"

    _ "github.com/lib/pq"
)

type user_list struct {
    Id               int
    Age              int
    Type             string
    Departure_name   string
    Departure_number int
    Departure_lat    float64
    Departure_lng    float64
    Arrival_name     string
    Arrival_number   int
    Arrival_lat      float64
    Arrival_lng      float64
}

type LocInfo struct {
    Num  int
    Name string
    Node int
    Lat  float64 // Lat→Lonの順番にしているのは、GoogleMAPの表示に合わせているため
    Lon  float64 // 普通は、Lat がX軸、LonがY軸
}

var UL [200]user_list // とりあえず200人分

var last_id int // csvファイルの行数

func main() {
    // postgresqlの初期化
    dbMap, err := sql.Open("postgres",
        "user=postgres password=password host=192.168.0.23 port=15432
dbname=yama_db sslmode=disable")
    log.Println("------------------ map db open ------------------")
    if err != nil {
        log.Fatal("OpenError: ", err)
    }
    defer dbMap.Close()

    // csvファイルの読み込み
    file, err := os.Open("c:\\users\\ebata\\kai_20220522holyday18_user_list.csv")
    if err != nil {
        log.Fatal(err)
    }
    defer file.Close()

    r := csv.NewReader(file)
    rows, err := r.ReadAll() // csvを一度に全て読み込む
    if err != nil {
        log.Fatal(err)
    }
```

```
    // 行ごとに
    for i, row := range rows {

        id := i - 1 // iは1から始まるので、ここで1引いておく

        if i == 0 {
            continue // CSVのヘッダー行を無視
        }

        UL[id].Age, _ = strconv.Atoi(row[1]) // 整数変換
        UL[id].Type = row[2]

        departure_lat, err := strconv.ParseFloat(row[5], 64)
        if err != nil {
            log.Fatal(err)
        }
        departure_lng, err := strconv.ParseFloat(row[6], 64)
        if err != nil {
            log.Fatal(err)
        }

        UL[id].Departure_number, UL[id].Departure_lng, UL[id].Departure_lat =
fixPosition(dbMap, departure_lng, departure_lat)

        arrival_lat, err := strconv.ParseFloat(row[9], 64)
        if err != nil {
            log.Fatal(err)
        }

        arrival_lng, err := strconv.ParseFloat(row[10], 64)
        if err != nil {
            log.Fatal(err)
        }

        UL[id].Arrival_number, UL[id].Arrival_lng, UL[id].Arrival_lat =
fixPosition(dbMap, arrival_lng, arrival_lat)

        last_id = id // idの最大値を更新
    }

    dbAgent, err := sql.Open("postgres",
        "user=postgres password=password host=192.168.0.23 port=15432
dbname=agent_db sslmode=disable")
    log.Println("------------------ map db open ------------------")
    if err != nil {
        log.Fatal("OpenError: ", err)
    }
    defer dbAgent.Close()

    dbAgent.Query("delete from user_list")

    for id := 0; id < last_id+1; id++ {
```

```
        UL[id].Id = id
        fmt.Println(UL[id])

        ins, err := dbAgent.Prepare("insert into user_list(id, age, type,
departure_name, departure_number, departure_lat, departure_lng, arrival_name,
arrival_number, arrival_lat, arrival_lng)
VALUES($1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$11)")
        if err != nil {
            log.Println("Prepare", err)
            os.Exit(-1)
        }

        _, err = ins.Exec(
            UL[id].Id,
            UL[id].Age,
            UL[id].Type,
            UL[id].Departure_name,
            UL[id].Departure_number,
            UL[id].Departure_lat,
            UL[id].Departure_lng,
            UL[id].Arrival_name,
            UL[id].Arrival_number,
            UL[id].Arrival_lat,
            UL[id].Arrival_lng)

        if err != nil {
            log.Println("ins exec", err)
            os.Exit(-1)
        }
    }
}

// 指定した座標に近いDB上の座標を取得
func fixPosition(db *sql.DB, _x1, _y1 float64) (int, float64, float64) {

    // Scan用の仮変数
    var source int
    var longitude float64
    var latitude float64
    var dist float64

    upperLimitMeter := 1500.0 // 近傍ノードの上限を1500 mに設定
    str := fmt.Sprintf(
        // 修正前: ways (道) の中から最近傍を取得
        // "SELECT source, x1 AS longitude, y1 AS latitude,
ST_Distance('SRID=4326;POINT(%v %v)'::GEOGRAPHY, the_geom) AS dist FROM ways WHERE
ST_DWithin(the_geom, ST_GeographyFromText('SRID=4326;POINT(%v %v)'), %.1f) ORDER
BY dist LIMIT 1",
        // 修正後: ways_vertices_pgr (点座標) の中から最近傍を取得
        "SELECT id AS source, lon AS longitude, lat AS latitude,
ST_Distance('SRID=4326;POINT(%v %v)'::GEOGRAPHY, the_geom) AS dist FROM
ways_vertices_pgr WHERE ST_DWithin(the_geom,
ST_GeographyFromText('SRID=4326;POINT(%v %v)'), %.1f) ORDER BY dist LIMIT 1",
        _x1, _y1, _x1, _y1, upperLimitMeter,
```

```
    )
    //fmt.Println(str)

    rows, err := db.Query(str)
    if err != nil {
        log.Fatal(err)
    }
    defer rows.Close()

    foundGoodMapNode := false

    for rows.Next() {
        foundGoodMapNode = true
        if err := rows.Scan(&source, &longitude, &latitude, &dist); err != nil {
            fmt.Println(err)
        }
        //fmt.Println(source, longitude, latitude, dist)
    }

    if !foundGoodMapNode {
        log.Println("Warning: in func fixPosition: Good Map Node not found for
query point (",
            _x1, ",", _y1, ")")
    }

    return source, longitude, latitude
}
```

因みに、"kai_20220522holyday18_user_list.csv"の中身はこんな感じ

```
id,age,type,departure_name,departure_number,departure_lat,departure_lng,arrival_na
me,arrival_number,arrival_lat,arrival_lng
0,43,resident,,,34.173408,131.470684,,,34.155862,131.501246
1,24,resident,,,34.179449,131.482543,,,34.164116,131.471791
2,42,resident,,,34.168739,131.470768,,,34.160989,131.491124
3,21,resident,,,34.169494,131.469934,,,34.173498,131.471351
4,58,resident,,,34.185295,131.47414,,,34.191481,131.49456
5,48,resident,,,34.150778,131.480747,,,34.16536,131.471872
6,56,resident,,,34.16536,131.471872,,,34.174066,131.479312
7,73,resident,,,34.155731,131.500845,,,34.16776,131.472831
8,47,resident,,,34.167237,131.471785,,,34.155775,131.476531
9,21,resident,,,34.154931,131.50468,,,34.156678,131.49581
10,37,resident,,,34.16727,131.472899,,,34.171253,131.471177
11,40,resident,,,34.147241,131.474921,,,34.150675,131.486268
12,67,resident,,,34.173683,131.476347,,,34.173643,131.471027
13,28,resident,,,34.183079,131.484303,,,34.174245,131.474592
14,46,resident,,,34.146154,131.472711,,,34.159611,131.491548
15,25,resident,,,34.162497,131.489283,,,34.147212,131.475984
```

8. (付録)bike_logを作る場合

```
//////バイクログのDB構築///////

テーブルを消すとき
drop table bike_log;

// user_logテーブルの内容の全削除
    dbAgent.Query("delete from user_list;")


create table bike_log(
    stationId int,
    num_outgoing int,
    num_incoming int
);
```

9. (付録)user_logを作る場合

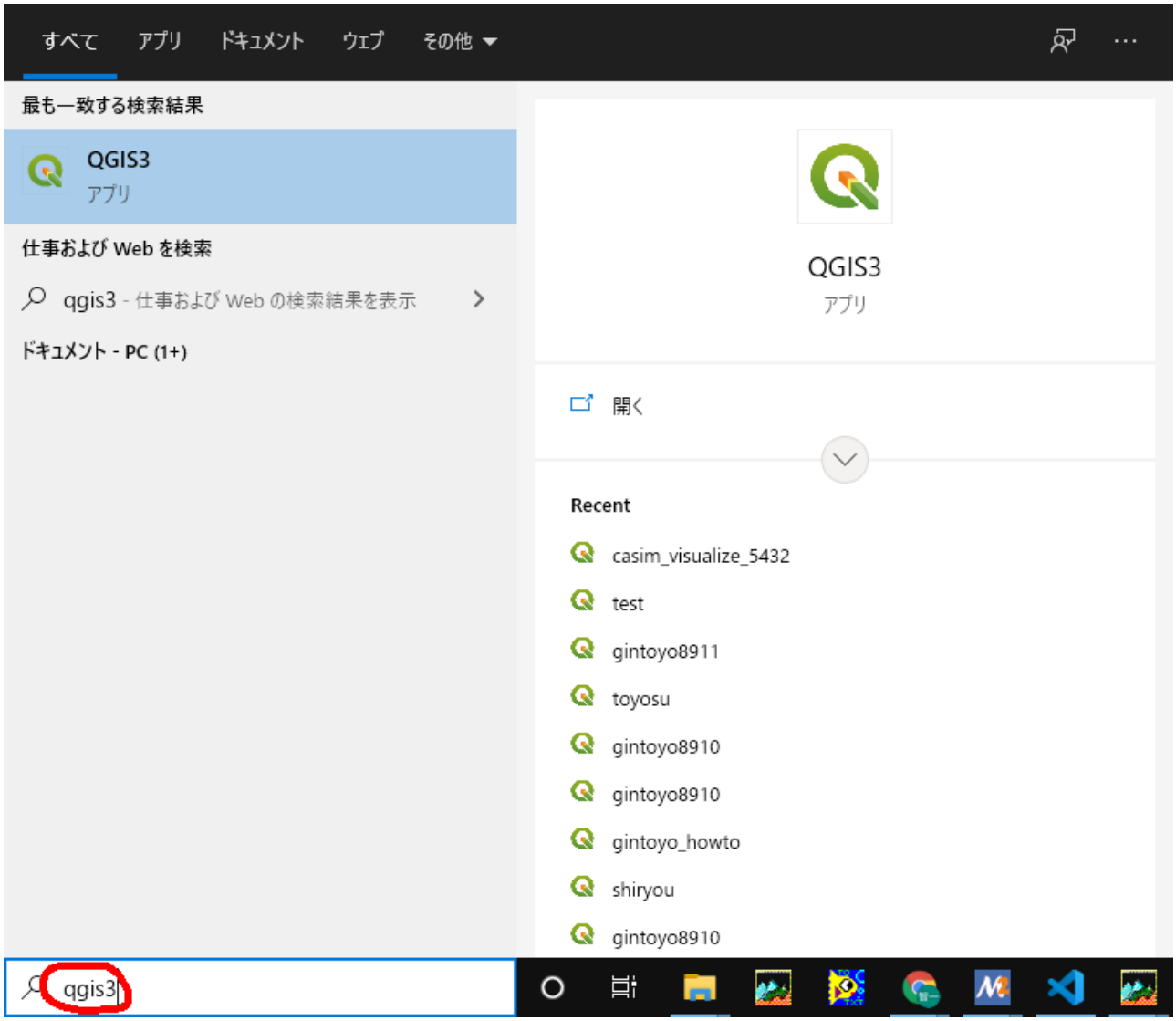
```
テーブルを消すとき
drop table user_log;


create table user_log(
    ID     int,
    Age    int,
    Walk_dis1 double precision,
    Bike_dis double precision,
    Walk_dis2 double precision,
    Ride_bike_time double precision,
    Drop_bike_time double precision,
    Arrival_time double precision,
    complaint double precision
);
```

10. 孤立ノード対応

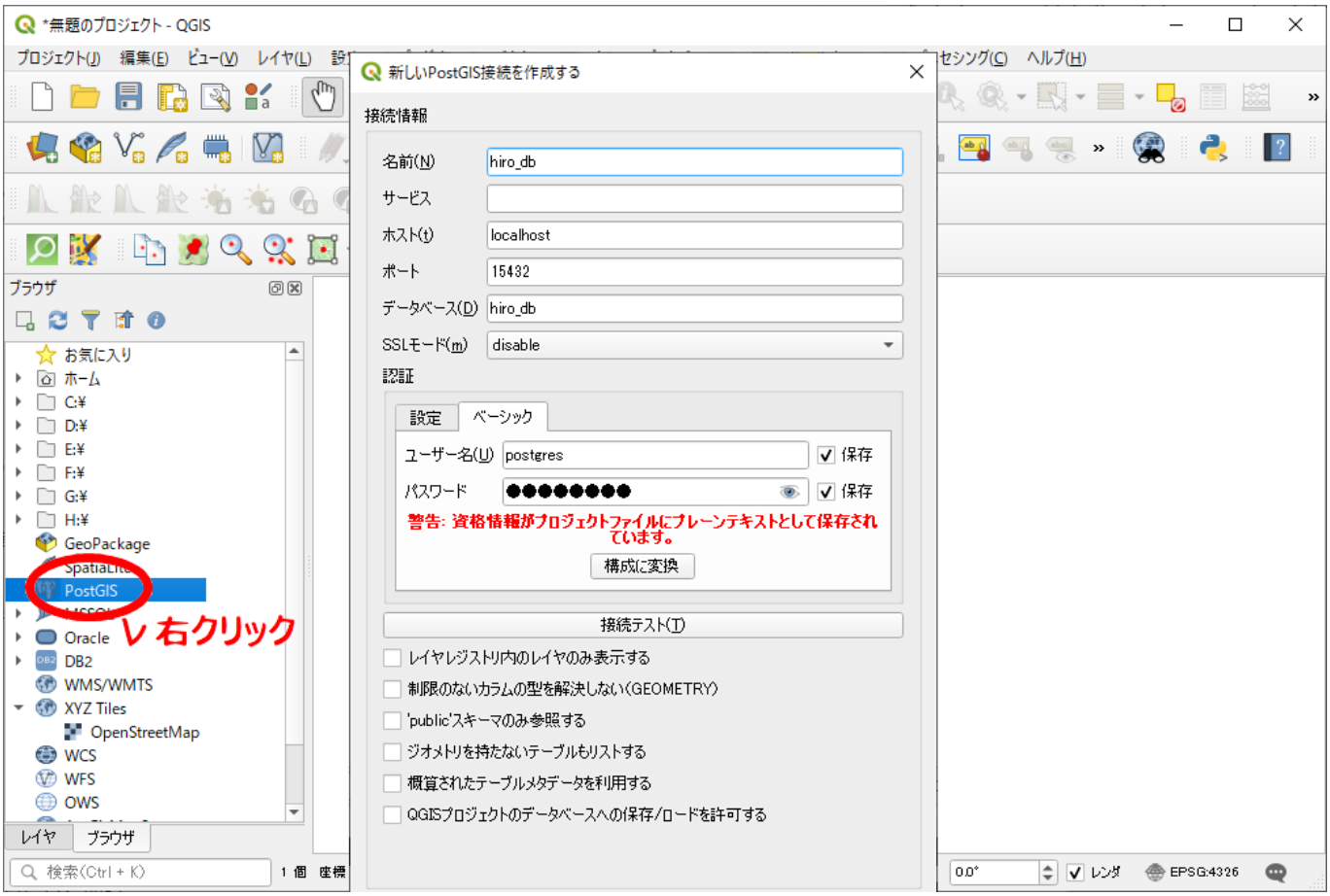
- OpenStreetMapのOSMファイルで発生する「孤立ノード問題」に対応するために、孤立ノードを削除する処理を実施。
- 処理内容については、Tools/prune_isonodes.goに記載済み。

11. QGIS3に表示する

- **[Step1]** QGIS3を立ち上げます。



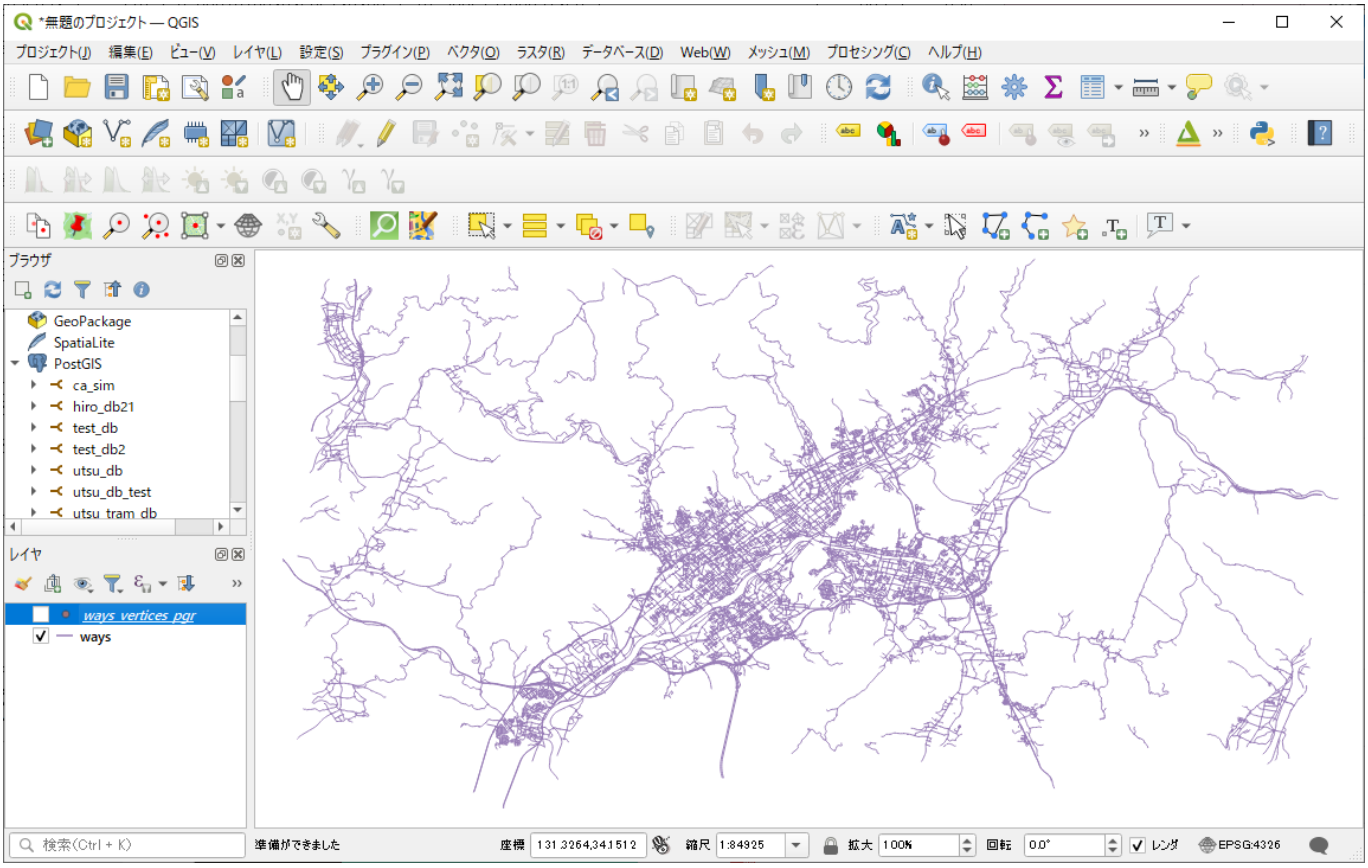
- **[Step4]** "PostGIS"のメニューからDBとリンクする



接続情報は以下の通りです。

```
名前(N)           yama_db
ホスト(t)         localhost
ポート            15432
データベース(D)    yama_db
SSLモード(m)       disable

認証（ベーシックのタブを選ぶ)
ユーザ名(U)        postgres     (保存をチェック)
パスワード         password     (保存をチェック)
「構成に変換」ボタンをクリック
```

レイヤに、waysを選んで、このような表示がでてくれば成功です



以上