Báo cáo tiến độ

Ngày 09/11-15/11

GVHD: Huỳnh Xuân Phụng

Sinh Viên thực hiện:

17110213 Trần Cao Quyền

17110181 Nguyễn Trọng Luật

Kế Hoạch Báo Cáo:

1. Nội dung thực hiện trong tuần
   - Tiếp tục Xây dựng Giao Diện
   - Tạo cơ bản database
   - Nghiên cứu giải quyết vấn đề giao hàng
2. Nội dung thực hiện trong tuần tới
   - Tiếp tục hoàn thiện giao diện app

Nội dung báo cáo:

1/ Tóm tắt thuật toán:

a. Sử dụng server python để chia các đơn hàng được tìm theo ngày và khung giờ nhất định thành các cụm nhỏ với điều kiện khoảng cách(ví dụ: 8h - 11h)

b. Sau khi có được dữ liệu cụm, dùng API Bing map để lấy được ma trận trọng số của từng địa chỉ trong cụm, format lại ma trận để phù hợp bài toán

c. Từ dữ liệu được format, dùng thuật toán NNA để sắp xếp các đơn hàng theo khoảng cách ngắn nhất và phù hợp thời gian

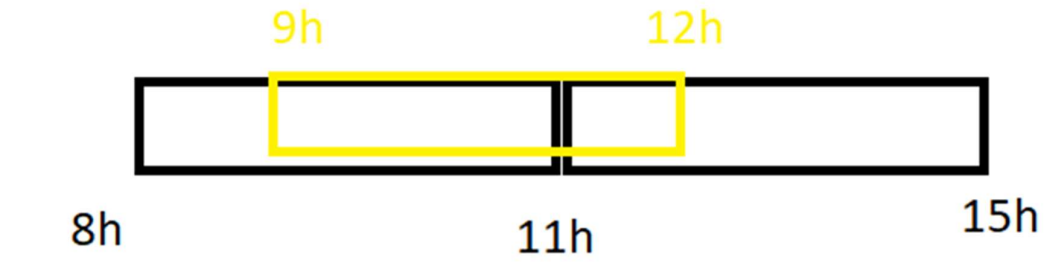
+ VD: từ điểm bắt đầu sẽ duyệt dữ liệu trọng số để tìm điểm gần nhất và có điều kiện thời gian được thỏa và lặp lại cho đến khi không còn điểm nào thỏa

2/ Phân chia cụm (Python):

a. Set khung thời gian giao trong ngày

```
def gettime(timestart,pltime):
    global timestartdefault
    timestartdefault=datetime.strptime(str(date+" " +timestart),'%m/%d/%Y %H:%M')
    global plustime
    plustime=int(pltime)
```

b. Tạo json theo khung thời gian để get api từ server nodejs để lấy đơn hàng về xử lý. Server python sẽ lấy những đơn hàng có thời gian trong khung giờ để xử lý.




```
def setjsontime(timerun, timeafter):
    return {
            "id":1,
            "timerange":{
                    "timeStart":str(timerun.strftime("%m/%d/%Y %H:%M")),
                    "timeEnd":str(timeafter.strftime("%m/%d/%Y %H:%M"))
                    }
            }
```

Output:

```
{'id': 1, 'timerange': {'timeStart': '11/03/2020 08:00', 'timeEnd': '11/03/2020 11:00'}}
```

c. Chuyển json vừa lấy từ nodejs về xử lý dữ liệu, lấy kinh độ và vĩ độ từ json cho vào mảng 2 chiều với 2 cột là kinh độ và vĩ độ và n địa chỉ

```
def getjsontoarray():
    X = np.array([[106,10]])
    url = "http://servertlcn.herokuapp.com/diachi/search"
    r = requests.post(url, json=setjsontime(timerun, timeafter))
    data=r.json()
    for i in range(0,len(data['data'])):
        X=np.concatenate([X,[[float(data['data'][i]['KinhDo']),float(data['data'][i]['ViDo'])]]])
    X=np.delete(X,0,axis = 0)
    return X
```

Output:

```
[[ 10.857785   106.739955  ]
 [ 10.86515577 106.74642416]
 [ 10.86028166 106.76050512]
 [ 10.86311602 106.75928203]
 [ 10.8645806  106.7626938 ]
 [ 10.85269069 106.76131921]
 [ 10.83901869 106.77444594]
 [ 10.84441907 106.78075986]
 [ 10.84629258 106.78403077]]
```

d. Chạy thuật toán để chia cụm, với n_clusters là số cụm được chia, affinity là mối quan hệ, ở đây là euclidean là khoảng cách các điểm và linkage là mối liên kết ở đây là single là khoảng cách giữa 2 điểm gần nhất của 2 cụm

```
def Cluster(A):
    if(len(A)>1):
        hc=AgglomerativeClustering(n_clusters=2,affinity='euclidean', linkage='single')
        return hc.fit_predict(A)
```

Output:

```
[[ 10.857785   106.739955  ]
 [ 10.86515577 106.74642416]
 [ 10.86028166 106.76050512]
 [ 10.86311602 106.75928203]
 [ 10.8645806  106.7626938 ]
 [ 10.85269069 106.76131921]]
[[ 10.83901869 106.77444594]
 [ 10.84441907 106.78075986]
 [ 10.84629258 106.78403077]]
```

e. Xử lý từng cụm tạo json

```
def outputdatatojson():
    a='{"data": [[],[]]}'
    data=json.loads(a)
    outputdata=data['data']
    cluster=0
    while(cluster<2):
        for j in range(0,len(A[y_hc==cluster])):
                for i in range(0,len(TempA)):
                    if(A[y_hc==cluster][j][0]==TempA[i][0] and A[y_hc==cluster][j][1]==TempA[i][1]):
                        outputdata[cluster].append(datatemp[i])
        cluster+=1
    return outputdata
```

```
def runcluster():
    global A
    A=getjsontoarray()
    global TempA
    TempA=getjsontoarray()
    global y_hc
    y_hc=Cluster(A)
    url = "http://servertlcn.herokuapp.com/diachi/search"
    r = requests.post(url, json=setjsontime(timerun, timeafter))
    data=r.json()
    global datatemp
    datatemp=data['data']
    j=outputdatatojson()
    return j
```

```
@app.route('/giaohang', methods=['GET'])
def Clusterr():
    try:
        j=runcluster()
        return  jsonify({'timeStart':timestartdefault.strftime("%H:%M"),'data':j,'plus':plustime})
    except KeyError:
        return f'Invalid input ({pd.series.index.min()} - {pd.series.index.max()})'
```

Output có dạng:

```
{
        'data':[
                [{diachi1},{diachi2}]
                [{diachi3},{diachi4}]
        ],
        'timeStart':'8:00',
        'plus':3
}
```

Xem chi tiết tại:

https://giaohangapi.herokuapp.com/giaohang

3/ Lấy trọng số, format lại cho phù hợp với bài toán (getDistance)

a. Sử dụng REST API Bing map để lấy được dữ liệu trọng số khoảng cách:(https://docs.microsoft.com/en-

us/bingmaps/rest-services/routes/calculate-a-distance-matrix)

b. Request Data Format:
   + Với origins là điểm bắt đầu, destinations là điểm đích
   + API có thể request tối đa 50x50 địa chỉ

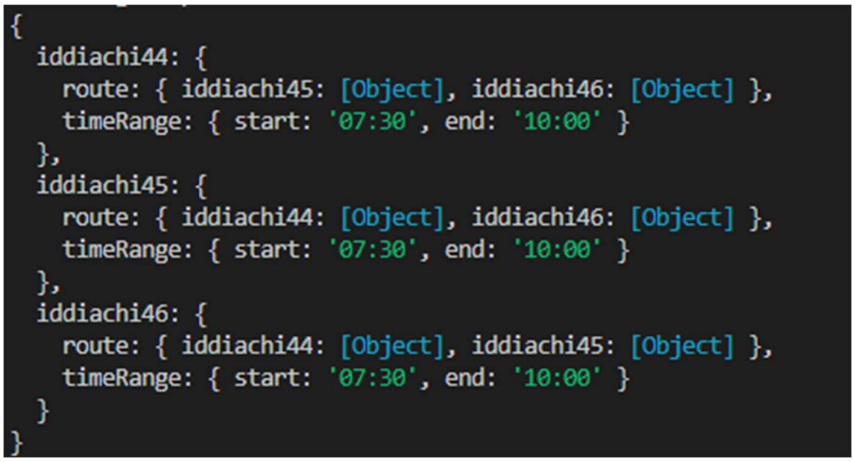
```
//Request Data Format
const listCoordRequest = {
    "origins": [],
    "destinations": [],
    "travelMode": "driving"
}
```

c. Sau khi có ma trận trọng số, format lại dữ liệu để phù hợp với bài toán:

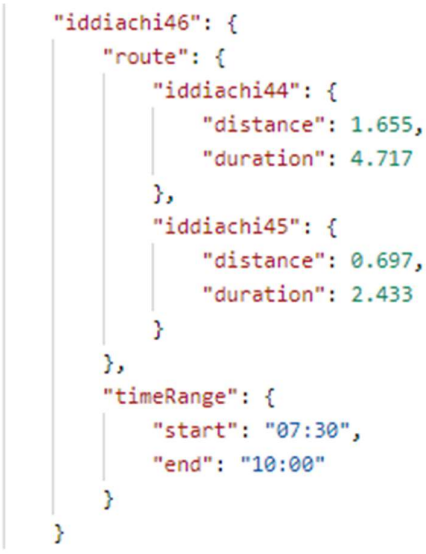
```
//Format dữ liệu trọng số để phù hợp với bài toán
for (let i of listCoord) {
    let timeStart = i.ThoiGianBatDau.split("T")[1].replace("Z", "").substr(0, 5);
    let timeEnd = i.ThoiGianKetThuc.split("T")[1].replace("Z", "").substr(0, 5);
    dataperRow = {};
    for (let e of dataMatrix) {
        if (checkrow < e.originIndex) {
            dataMatrix.splice(0, listCoord.length)
            break;
        }
        else {
            if (e.travelDistance != 0) {
                let temp = { distance: e.travelDistance, duration: e.travelDuration };
                dataperRow['iddiachi' + listCoord[e.destinationIndex].id] = temp;
            }
        }
    }
    checkrow++;
    matrixData['iddiachi' + i.id] = {
        route: dataperRow,
        timeRange: {
            start: timeStart,
            end: timeEnd
        }
    };
}
```

d. Sau khi format, dữ liệu sẽ có dạng như sau:

+ ví dụ: cụm địa chỉ với id (44, 45, 46)



+ Với mỗi địa chỉ sẽ có dạng như sau: ví dụ địa chỉ 46
    - Khoảng cách từ địa chỉ 46 tới địa chỉ 44 là: 1.655 Km
    - Thời gian để từ địa chỉ 46 tới địa chỉ 44 là: 4.717 phút



4/ Thuật toán NNA (getroute)

a. Xác định điểm khởi đầu:

```
//Xác định điểm khởi đầu và khởi tạo
var s = Object.keys(graph)[0];
var solutions = {};
solutions[s] = [];
solutions[s].dist = 0;
solutions[s].dur = timeStart;
```

b. Khởi tạo biến lưu trữ điểm hiện tại và thời gian đã đi từ đầu đến điểm hiện tại:

```
//Lưu trữ điểm hiện tại và thời gian đã đi
var currentPoint = s;
var currentDist = 0;
var currentDur = timeStart;
let lastDur = timeStart;
```

c. Bắt đầu duyệt mảng:

```
//Tạo vòng lặp duyệt mảng trọng số
while (true) {

    //Lấy ra dữ liệu trọng số của điểm hiện tại từ ma trận trọng số
    let adj = graph[currentPoint].route;
    //Khởi tạo biến kiểm tra khoảng cách ( tìm min )
    let distCheck = Infinity
```

+ Với mỗi địa chỉ, vòng lặp tiếp theo sẽ duyệt từng khoảng cách so với các địa chỉ còn lại (ví dụ địa chỉ 44, vòng lặp sẽ đi từng thành phần trong route là 45 rồi đến 46)

```
{
  iddiachi44: {
    route: { iddiachi45: [Object], iddiachi46: [Object] },
    timeRange: { start: '07:30', end: '10:00' }
  },
  iddiachi45: {
    route: { iddiachi44: [Object], iddiachi46: [Object] },
    timeRange: { start: '07:30', end: '10:00' }
  },
  iddiachi46: {
    route: { iddiachi44: [Object], iddiachi45: [Object] },
    timeRange: { start: '07:30', end: '10:00' }
  }
}
```

+ Tạo các biến kiểm tra khoảng cách, thời gian phù hợp để làm điểm tiếp theo

```
//Duyệt từng khoảng cách so với cái điểm còn lại tìm khoảng cách ngắn nhất
for (var subE in adj) {
    //Lấy giá trị khoảng cách
    var ndist = adj[subE].distance
    //Tạo biến thời gian ước lượng để tới được điểm tiếp theo
    //( dùng để kiểm tra có phù hợp khung giờ đưa ra hay không)
    var ndur = addTimes(lastDur, adj[subE].duration + 10)

    //Kiểm tra nếu đã tồn tại điểm đó trong mảng thì duyệt điểm tiếp theo
    //Kiểm tra đánh dấu đã đi qua
    if (solutions[subE]) {
        continue;
    }

    //Kiểm tra nếu thời điểm đến điểm tiếp theo nằm trong khung giờ đã đưa ra
    //Nếu không nằm trong khung giờ đã đưa ra thì duyệt điểm tiếp theo
    if (!checkTime(ndur, graph[subE].timeRange)) {
        continue;
    }
```

+ Tiếp tục kiểm tra điều kiện khoảng cách nếu đã thỏa điều kiện về thời gian (currentDur phải nằm trong khoảng timeRange của điểm đang xét), tìm điểm gần nhất so với điểm hiện tại:

```
        //Kiểm tra khoảng cách ngắn nhất trong tất cả các điểm
        //Chọn làm điểm tiếp theo
        if (ndist < distCheck) {
            distCheck = ndist;
            currentPoint = subE;
            currentDur = ndur
        }
    }
    //Nếu không tìm thấy điểm tiếp theo thì break vòng lặp, trả về mảng được sắp xếp
    if (distCheck == Infinity) {
        break;
    }

    //Lưu trữ điểm hiện tại, cập nhật lại các biến số và tiếp tục vòng lặp nếu chưa kết thúc
    solutions[currentPoint] = [];
    solutions[currentPoint].dist = currentDist;
    solutions[currentPoint].dur = currentDur;
    lastDur = currentDur;
}
```

+ Đối với những đơn hàng không thỏa điều kiện và chưa được sắp xếp thì sẽ được liệt kê vào một chuỗi đơn hàng riêng

```
//Những điểm không thỏa điều kiện về thời gian và chưa được sắp xếp
//Sẽ được lưu vào 1 chuỗi và được đánh dấu là outtime và được tạo thành 1 chuỗi đơn hàng riêng
let dataOutTime = []
for (var e in graph) {
    if (!solutions[e]) {
        let temp = {
            diachi: e,
            outTime: true
        }
        dataOutTime.push(temp);
    }
    else {
        continue
    }
}
```

+ Cuối cùng là format lại data để trả về

```
//Format lại data để response
let data = [];
for (var n in solutions) {
    let temp = {
        diachi: n,
        //dist: solutions[n].dist,
        estimatedTime: solutions[n].dur,
    }
    data.push(temp);
}
let result = {
    dataInTime: data,
    dataOutTime: dataOutTime
}
return result;
```

5/ Hàm kiểm tra thời gian

```
//Kiểm tra thời gian có nằm trong khung giờ hay không
function checkTime(time, timeRange) {
    var y = new Date('01/01/2001 ' + time).getTime();

    var a = new Date('01/01/2001 ' + timeRange.start).getTime();
    var b = new Date('01/01/2001 ' + timeRange.end).getTime();
    if (y < Math.max(a, b) && y >= Math.min(a, b)) {
        return true
    }
    return false
}
```

6/ Hàm cộng thời gian

```
//Cộng thời gian, trả về chuỗi
function addTimes(start, duration) {
    if (duration >= 60) {
        var hours = Math.floor(duration / 60);
        var minutes = duration % 60;
        duration = hours + ":" + minutes
    }
    else {
        duration = "00:" + duration
    }
    let totalH = 0;
    let totalM = 0;
    start = start.split(":");
    duration = duration.split(":");

    totalH += parseInt(start[0], 10) + parseInt(duration[0], 10);
    totalM += parseInt(start[1], 10) + parseInt(duration[1], 10);

    if (totalM >= 60) {
        totalH += Math.floor(totalM / 60);
        totalM = totalM % 60;
    }
    return totalH + ":" + totalM;
}
```

7/ Hàm chính (getDonhang)

a. Fetch data từ server python để có dữ liệu các cụm đơn hàng

```
//fetchData từ server python để lấy được các cụm đơn hàng được chia theo tỉ lệ khoảng cách
//Hỗ trợ để tạo thành các đơn hàng chia cho shipper
let settings = {
    method: "GET",
    // body: JSON.stringify(dataPost),
    // headers: { 'Content-Type': 'application/json' },
};
let dataFetch = await fetch('https://giaohangapi.herokuapp.com/giaohang', settings)
let dataReceive = await dataFetch.json();

mapData = dataReceive.data //Dữ liệu trả về từ api
timeStart = dataReceive.timeStart //Thời điểm bắt đầu của khung giờ xét duyệt đơn hàng
```

b. Với mỗi cụm sẽ chạy các hàm ở trên để có được dữ liệu sắp xếp các đơn hàng

```
// Với từng cụm đơn hàng, bắt đầu các bước sau:
const promises = mapData.map(async (e) => {
    // Lấy dữ liệu trọng số khoảng cách giữa các địa chỉ trong cụm đơn hàng
    let matrixData = await getDistance(e)
    //let dataRender = dijkstra(matrixData, timeStart)

    //Sử dụng thuật toán NNA để sắp xếp các đơn hàng theo khoảng cách và thời gian
    let test = getroute(matrixData, timeStart)
    resultData.push(test)
})
const results = await Promise.all(promises)
```

c. Sau cùng là response data:

```
//Trả dữ liệu về
return new Promise((resolve, reject) => {
    res.json({
        result: 'ok',
        data: resultData,
    })
})
```

+ Dữ liệu trả về:

- cụm 1:


```
{
    "dataInTime": [
        {
            "diachi": "iddiachi44",
            "estimatedTime": "08:00"
        },
        {
            "diachi": "iddiachi45",
            "estimatedTime": "8:12"
        },
        {
            "diachi": "iddiachi46",
            "estimatedTime": "8:24"
        }
    ],
    "dataOutTime": []
},
```


- cụm 2: với 2 đơn đơn bị outtime

```
    "dataInTime": [
        {
            "diachi": "iddiachi1",
            "estimatedTime": "08:00"
        },
        {
            "diachi": "iddiachi6",
            "estimatedTime": "8:18"
        },
        {
            "diachi": "iddiachi5",
            "estimatedTime": "8:29"
        },
        {
            "diachi": "iddiachi36",
            "estimatedTime": "8:42"
        }
    ],

    "dataOutTime": [
        {
            "diachi": "iddiachi2",
            "outTime": true
        },
        {
            "diachi": "iddiachi7",
            "outTime": true
        }
    ]
}
```