最終課題レポート

1922074 木村太紀

1.研究発表講聴レポート

・現状の課題

1. 複数の移動手段を利用する際の不便さ
    - 各移動手段間の繋がりが煩雑になる
    - 初めて利用する人達には利用し辛い
2. 公共交通機関や道路の渋滞が起こる
    - 自家用車利用者による道路の渋滞が発生
    - バス等公共交通機関の円滑な運行が難しくなる
3. ラストワンマイルの移動手段不足
    - 地方では自家用車が無いと日常の移動も難しい
    - 都心部でも微妙な距離での移動手段は限定される
4. 移動弱者の増加
    - 地方の公共交通機関撤退により、交通空白地域拡大
    - 自治体の公共交通機関維持に関する財政圧迫

・現在存在するMaaSシステムにおける課題

- ユーザが必要とする機能を全て反映した場合、様々な選択オプションを操作する必要がある為、IT弱者等に移動の格差が生まれてしまう。

・移動の本質とは

1. **Knowledge(知識を豊かにしたい)**
2. **Physical(自分自身を成長させたい)**
3. **Mental(緊張を緩和したい)**
4. **Curiosity(楽しい事をしたい)**
5. **Relationship(人間関係を深めたい)**

- **Wish(何かを達成・変化させたい)**

・MaaSシステムにおける課題をどう解決するか

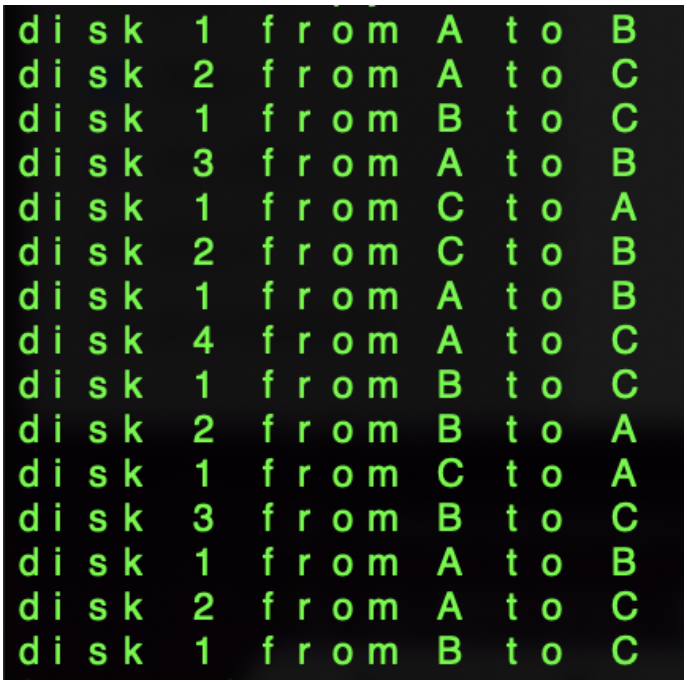
- **現状の課題**
    - 前述の様に、システムの操作が煩雑になってしまい、万人向けのシステムとは言えなくなっている
- **解決について**
    - 煩雑さの例として、主にどの公共交通機関を使用するかについての操作が挙げられる。
      よってそれの解決として、AIとビッグデータの活用が考えられる。現代では多くの人が公共交通機関の支払いをスマートフォンで行っており、それを用いて年代別にどの公共交通機関をよく利用するかにつ

いてのデータを収集する事が可能である。
このコンテンツを使用する際に自らの年代を登録する事で、前述のデータから、利用者がどの公共交通機関を使いたいかについてを予想する事が出来る。ぞれによって通常利用時は煩雑な公共交通機関を選択する操作を省く事が可能となる。
また、自ら交通機関を選択したい際の選択肢も用意する事で、柔軟な使用が出来る。

2.実装物まとめ

2-1.ハノイの塔(第4回課題)

```
def towers(top, frm, inter, to):
    if top == 1:
        print("disk 1 from " + frm + " to " + to)
    else:
        towers(top-1, frm, to, inter)
        print("disk " + str(top) + " from " + frm + " to " + to)
        towers(top-1, inter, frm, to)
if __name__ == "__main__":
    num = 4
    towers(num, "A", "B", "C")
```



- 第4回課題で制作したハノイの塔攻略スクリプト
- ターン毎の移動順を表示される様に工夫

2-2.クイックソート(第4回課題)

```
import random

N = 10
```

```
def input_data(data):
    for i in range(N):
        data.append(random.randint(0,N))
    return data
def quicksort(left,right):
    if left >= right:
        return
    else:
        pivot = data[right]
        partition = division(left,right,pivot)
        quicksort(left,partition -1)
        quicksort(partition + 1,right)
def division(left, right, pivot):
    p = 0
    leftnum = left
    rightnum = right - 1
    while left <= right:
        while data[rightnum] > pivot:
            rightnum = rightnum -1
        while data[leftnum] < pivot:
            leftnum = leftnum + 1
        if leftnum < rightnum:
            data[rightnum],data[leftnum] = data[leftnum],data[rightnum]
        elif leftnum >= rightnum:
            break
    temp = data[leftnum]
    data[leftnum] = pivot
    data[right] = temp
    p = leftnum
    return p

if __name__ == "__main__":
    data = []
    data = input_data(data)
    print(data)
    quicksort(0,N - 1)
    print(data)
```

```
[7, 10, 9, 10, 3, 4, 0, 5, 9, 6]
[0, 3, 4, 5, 6, 7, 9, 9, 10, 10]
```

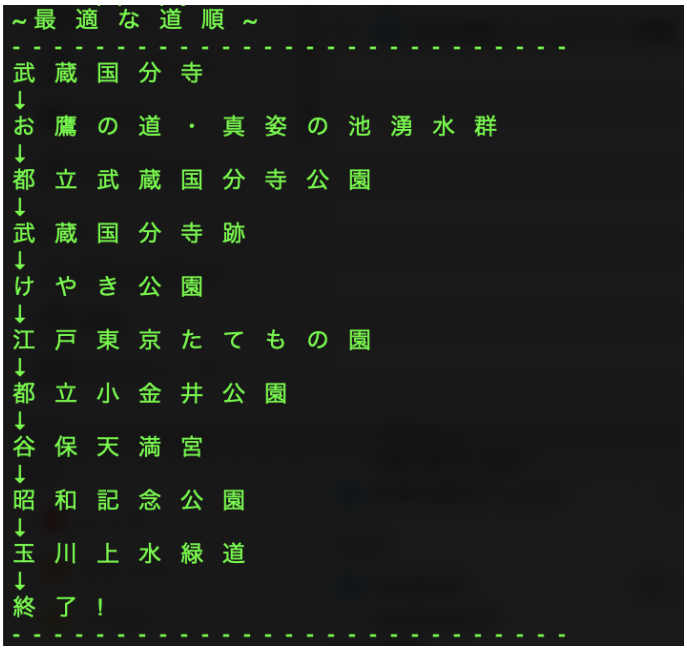
- 第4回課題で制作したクイックソートスクリプト

2-3.自主考案巡回セールスマン問題(第5回課題)

自主考案_セールスマン問題/app.pyにおける一部分

```
def set_point(point_A):
    if len(name_ex) > 0:
        for i in name_ex:
```

```
            if name[i] == point_A:
                point_A_id = i
        del name_ex[point_A_id]
        for i in name_ex:
            point_B = name[i]
            make_dist(point_A,point_B)
            #移動時間リスト(time_box)出力
def salesman(point_abc):
    loops = len(name_box)
    print("~最適な道順~")
    print("----------------------------")
    while loops > 0:
        point_A = point_abc
        min_count = 0
        set_point(point_A)
        print(point_A)
        print("↓")
        if loops > 1:
            for get_min in time_box:
                if min(time_box) == get_min:
                    name_box.remove(point_A)
                    point_abc = name_box[min_count]
                    loops = loops -1
                else:
                    min_count += 1
        else:
            print("終了!")
            print("----------------------------")
            break
        time_box.clear()
#地点を指定する事で算出
point_abc = "武蔵国分寺"
#set_point(point_A)
salesman(point_abc)
```



- 自作した巡回セールスマン問題
- スタート地点を指定するのみで巡回ルートが出力される様に自動化
- 地球表面が曲面な事を踏まえた距離計算を行った
- 出力表示に関しても視覚的に工夫を行なった
- DBの変化にも対応

2-4.ダイクストラによる巡回セールスマン問題(第6回課題)

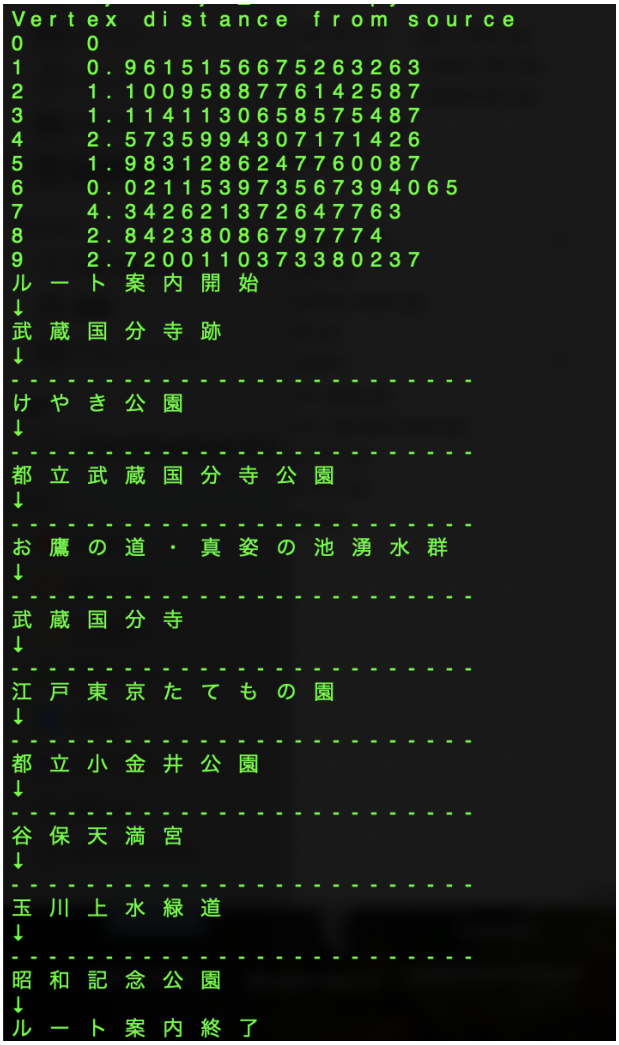
dijk_sales.pyにおけるデータ処理部分

```
#任意に入力でスタート地点を変更
#--------------------------------
#--------------------------------
start_point = "武蔵国分寺跡"
#--------------------------------
#--------------------------------
point_count = 0
for i in get_name:
    if i == start_point:
        point_number = point_count
    else:
        point_count += 1

g.dijkstra(point_number);
for i in range(0,len(elgo)):
    city_dic[get_name[i]] = elgo[i]

show_root = sorted(city_dic.items(), key=lambda x:x[1])
```

dijk_spot.pyにおける多次元距離リストの自動生成

```
        def set_point(point_A):
            dist_list = []
            for i in name:
                if name[i] == point_A:
                    point_A_id = i
            #del name_ex[point_A_id]
            for i in name_ex:
                point_B = name[i]
                dist = make_dist(point_A,point_B)
                if len(dist_list) < len(name):
                    dist_list.append(dist)
                    #print(dist_list)
            if len(dist_list) == len(name):
                multi_dimention.append(dist_list)


        for i in name:
        #point_A = "武蔵国分寺"
            point_A = name[i]
```

```
                set_point(point_A)
            return multi_dimention , return_name
```



- ダイクストラを用いた巡回セールスマン問題
- PostgreSQLから取得したデータを処理、加工をする段階において、dijk_sales.pyとの分割を行う事で、エラーが出た際や書き換えが必要な際の手間を減少させた
- Dijkstra.pyにおいて、スタート地点と各目的地を入力するのみで最短ルートが出力される様に各部の自動化を行った
- DBへの新たな書き込みや削除に対応

2-5.OneMaxにおける巡回セールスマン問題(第7回課題)

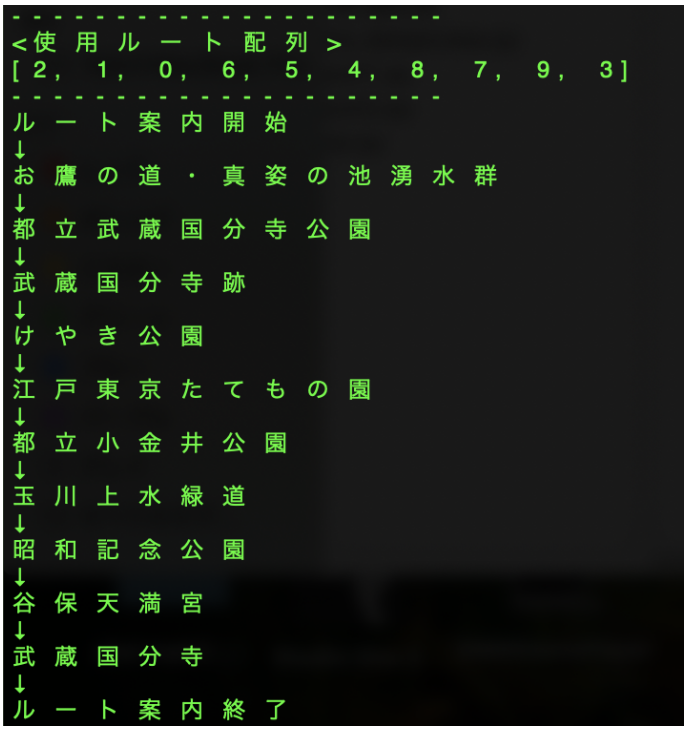
onemax_spot.pyの座標リスト作成

```
from onemax_dataaccess import DataAccess
import itertools
class spots:
    def send_spot(self):
        hoge = DataAccess()
        name = {}
        x2 = []
        y2 = []
```

```
        count = 0
        #ロケーション名を辞書に格納
        for id_n in range(1,11):
            x = list(itertools.chain.from_iterable(hoge.get_spots3(id_n)))
            name[count] = x[0]
            count += 1
        x = list(itertools.chain.from_iterable(hoge.get_spots()))
        y = list(itertools.chain.from_iterable(hoge.get_spots2()))

        #緯度経度を整形
        for i in range(0,10):
            x2.append(x[i] / 100)
            y2.append(y[i] / 1000)
        #緯度経度をタプルでまとめて格納
        pepoi = []
        for i in range(0,len(x2)):
            m = (x2[i],y2[i])
            pepoi.append(m)
        return pepoi,name
```

onemax_sales.py出力部分

```
spots_name = get_spots[1]
print("----------------------")
print("使用ルート配列")
print(route_list[0])
print("----------------------")
print("ルート案内開始")
print("↓")
#最も適応度が高いルートを選択
for i in route_list[0]:
    print(spots_name[i])
    print("↓")
print("ルート案内終了")
```



- OneMaxを用いた巡回セールスマン問題
- ダイクストラ使用のものと同様に、ostgreSQLから取得したデータを処理、加工をする段階においてonemax_sales.pyとの分割を行う事で、エラーが出た際や書き換えが必要な際の手間を減少させた
- DBへの新たな書き込みや削除に対応
- ファイルを実行するのみで出力される様に自動化されている

2-6.MaaSシステムの提案(最終課題)

・MaaS_proto.pyにおける多次元距離リストの処理部分

```
#--------------------------------
#距離とスポット名を持ってくる
get_form_return = hogehoge.dist_maker()
get_dist = get_form_return[0]
get_name = get_form_return[1]
#--------------------------------
#出力用辞書にて使うspotのidリスト
id_box = []
#--------------------------------
#スタート地点をspot_distに格納
point_count = 0
for i in get_name:
    if i == start_point:
        point_number = point_count
        spot_dist.append(get_dist[point_count])
        id_box.append(point_number)
    else:
        point_count += 1

#--------------------------------
#行きたい場所をspot_distに格納
for i in wanna_go:
```

```
        id_count = 0
        for num in get_name:
            if num == i:
                spot_dist.append(get_dist[id_count])
                id_box.append(id_count)
            else:
                id_count += 1
```

```
<開 始 地 点 >: 江 戸 東 京 た て も の 園

<目 的 地 群 >
・ け や き 公 園
・ 都 立 武 蔵 国 分 寺 公 園
・ お 鷹 の 道 ・ 真 姿 の 池 湧 水 群
・ 昭 和 記 念 公 園

<使 用 す る ダ イ ク ス ト ラ 算 出 リ ス ト >
~~~~~~~~~~~~~~~~~~~~~~~~~~
Vertex distance from source
0    0
1    2.365564485331647 8
2    2.384004159221588 4
3    2.434626101368746
4    0.703151054154104 8
~~~~~~~~~~~~~~~~~~~~~~~~~~

<最 短 ル ー ト を 表 示 >
ル ー ト 案 内 開 始
↓
江 戸 東 京 た て も の 園
↓
----------------------------
昭 和 記 念 公 園
↓
----------------------------
け や き 公 園
↓
----------------------------
都 立 武 蔵 国 分 寺 公 園
↓
----------------------------
お 鷹 の 道 ・ 真 姿 の 池 湧 水 群
↓
ル ー ト 案 内 終 了
```

- MaaSシステムプロトタイプ
- 会話型処理を用いた擬似アプリシステム
- DBから取得したデータを処理する流れはダイクストラによる巡回セールス問題と同様
- 開始地点と目的地群を選択する事で、動的に最短ルートを算出する
- 前述の入力のみで出力される様に自動化している
- DBへの新たな書き込みや削除に対応
- 出力結果に関しても一目で分かる様に工夫を行った