# RIVER FORMATION DYNAMICS FOR SOLVING TSP

# MEMBER

1

6410110337

นายพงศ์พิพัฒน์ ขุนชิต

2

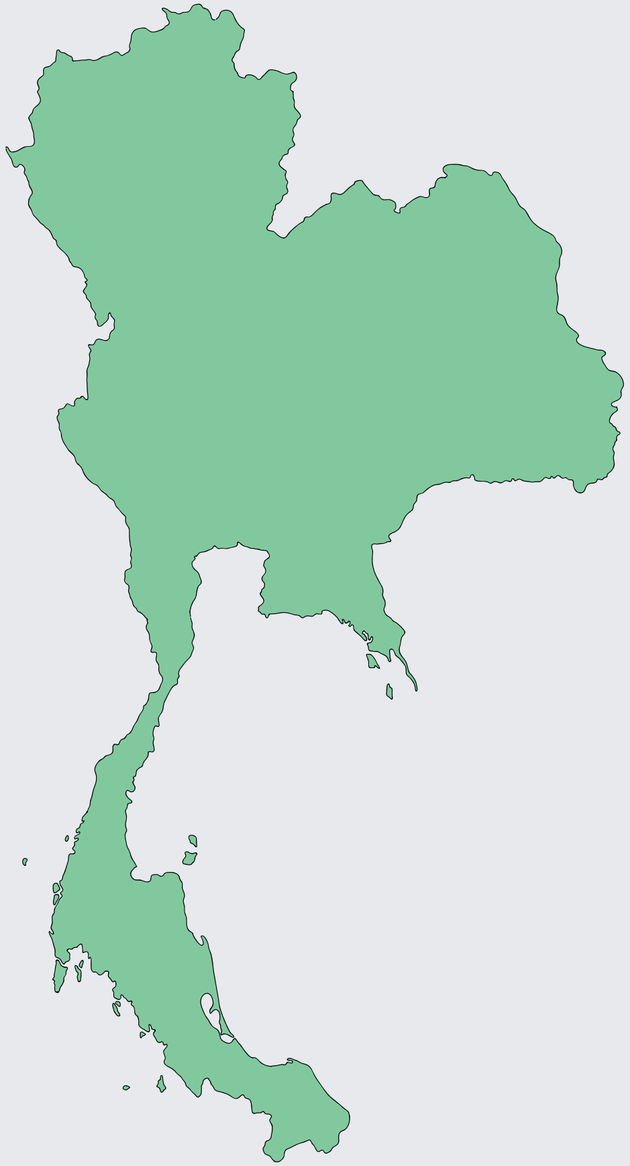
6410110701

นายภานุพงษ์ ลิม

3

6410110738

นายอนุวัฒน์ ขาวแก้ว



# PROBLEM

ท่องเที่ยวทุกจังหวัดในประเทศไทย
คำถาม หากเริ่มเดินทางจากจังหวัดสงขลา
จะเรียงจังหวัดในการไปเที่ยวอย่างไรให้ได้
ระยะทางที่สั้นที่สุดในการท่องเที่ยว

# USE RFD TO SOLVE

- หาเส้นทางใหม่โดยใช้ฟังก์ชัน **find_route** และคำนวณระยะทางของเส้นทางใหม่
- ถ้าระยะทางที่ได้มีค่าน้อยกว่าระยะทางของเส้นทางที่ดีที่สุด จะทำการกำหนดให้เส้นทางใหม่เป็นเส้นทางที่ดีที่สุด
- ทำการกัดเซาะเส้นทางของเส้นทางใหม่โดยใช้ฟังก์ชัน **flow_water**

```python
def rfd_tsp(
    num_iterations=1000
):

    best_route = []
    best_distance = 1000000
    all_distance = []
    # วนลูปหลักของอัลกอริธึม
    for iteration in range(num_iterations):
        new_route = find_route()  # หาเส้นทางใหม่ตามระดับน้ำและระดับความสูง
        new_distance = total_route_distance(new_route)

        # ถ้าเส้นทางใหม่ดีกว่า ให้เลือกเส้นทางนั้น และให้ "น้ำ" ไหลในเส้นทางนั้น
        if new_distance < best_distance:
            best_route = new_route
            best_distance = new_distance
        all_distance.append(best_distance)

        # ปล่อยให้น้ำไหลในเส้นทางที่พบ พร้อมกัดเซาะเส้นทาง
        flow_water(new_route)

        # แสดงผลทุก ๆ 100 รอบ
        if iteration % 100 == 0:
            print(f"Iteration {iteration}: Best Distance = {best_distance},\nBest Route = {best_route}\n")

    return best_route, best_distance, all_distance
```

# USE RFD TO SOLVE

```
def flow_water(route):
    global water_levels, height_levels
    for i in range(len(route) - 1):
        water_levels[route[i], route[i + 1]] += water_drop
        height_levels[route[i], route[i + 1]] -= erosion_rate  # กัดเซาะเส้นทาง
    water_levels[route[-1], route[0]] += water_drop
    height_levels[route[-1], route[0]] -= erosion_rate  # กัดเซาะเส้นทาง
```

- ฟังก์ชันการไหลของน้ำ
  - เพิ่มน้ำในเส้นทางระหว่างจุดนั้นไปยังจุดถัดไป
  - ลดความสูงของเส้นทางระหว่างจุดนั้นไปยังจุดถัดไป เนื่องจากการกัดเซาะ

# USE RFD TO SOLVE

[0 -> 1]

Before

- water_levels

| | |
|---|---|
| 0 | 0 |
| 0 | 0 |

- height_levels

| | |
|---|---|
| 1 | 1 |
| 1 | 1 |

After

- water_levels

| | |
|---|---|
| 0 | 0 + water |
| 0 | 0 |

- height_levels

| | |
|---|---|
| 1 | 1 - erosion |
| 1 | 1 |

# USE RFD TO SOLVE

- คำนวณความน่าจะเป็นสำหรับแต่ละเมืองถัดไปที่อาจถูกเลือก
- ถ้าความน่าจะเป็นรวมกันเป็น 0 จะสุ่มเลือกเมืองถัดไปจาก **remaining_cities**
- ถ้าความน่าจะเป็นไม่เท่ากับ 0 ก็จะสุ่มหยิบเมืองที่เหลือจาก **remaining_cities** แต่จะเพิ่ม **parameter weights** ไปเพื่อให้น้ำหนักในการสุ่มหยิบได้เพิ่มขึ้น

```python
def find_route():
    global water_levels, height_levels
    route = [56]  # เริ่มต้นจากเมืองสุ่มเมืองหนึ่ง
    while len(route) < num_cities:
        current_city = route[-1]
        remaining_cities = list(set(range(num_cities)) - set(route))
        # คำนวณความน่าจะเป็น
        probabilities = [
            (
                (
                    water_levels[current_city, next_city]
                    / height_levels[current_city, next_city]
                )
                if height_levels[current_city, next_city] > 0
                else 0
            )
            for next_city in remaining_cities
        ]
        # ตรวจสอบว่าความน่าจะเป็นรวมเป็นศูนย์หรือไม่
        total_probability = sum(probabilities)
        if total_probability == 0:
            next_city = random.choice(remaining_cities)  # ใช้การเลือกแบบสุ่ม
        else:
            # ทำให้ความน่าจะเป็นอยู่ในรูปแบบที่สามารถใช้ได้
            probabilities = [p / total_probability for p in probabilities]
            next_city = random.choices(
                remaining_cities, weights=probabilities, k=1
            )[0]
        route.append(next_city)
    return route
```

# USE RFD TO SOLVE

[ 0 -> ? ] : Next City Calcuation

- water_levels

| | | |
|---|---|---|
| 0 | 20 | 10 |
| 40 | 0 | 10 |
| 30 | 20 | 0 |

- height_levels

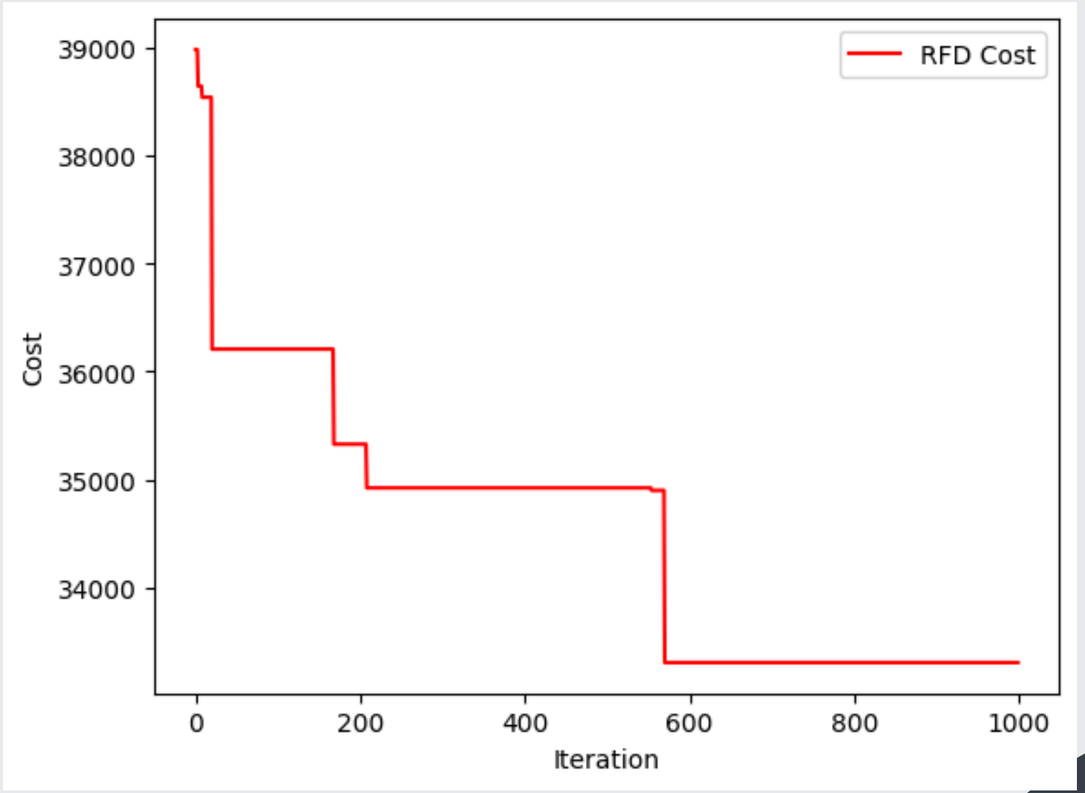
| | | |
|---|---|---|
| 1 | 0.9 | 0.7 |
| 0 | 1 | 0.5 |
| 0.3 | 0.5 | 1 |

propability = [ 22.22 ( 0 -> 1 ) , 14.286 ( 0 -> 2 ) ]

# OUTPUT

ตัวอย่างระหว่างการวนลูปหาเส้นทางที่ใช้ระยะเดินทางน้อยที่สุด

```
Iteration 0: Best Distance = 38982.84520931545,
Best Route = [56, 34, 50, 44, 57, 66, 33, 1, 53, 55, 6, 54, 60

Iteration 100: Best Distance = 36208.76328672418,
Best Route = [56, 4, 35, 58, 22, 32, 5, 51, 72, 20, 10, 15, 68

Iteration 200: Best Distance = 35328.65137898321,
Best Route = [56, 57, 27, 7, 26, 13, 51, 50, 62, 38, 73, 30, 7

Iteration 300: Best Distance = 34923.751392887265,
Best Route = [56, 59, 43, 48, 49, 17, 9, 73, 62, 35, 3, 50, 72

Iteration 400: Best Distance = 34923.751392887265,
Best Route = [56, 59, 43, 48, 49, 17, 9, 73, 62, 35, 3, 50, 72

Iteration 500: Best Distance = 34923.751392887265,
Best Route = [56, 59, 43, 48, 49, 17, 9, 73, 62, 35, 3, 50, 72

Iteration 600: Best Distance = 33306.46459911794,
Best Route = [56, 57, 11, 74, 64, 67, 54, 32, 68, 29, 26, 42,

Iteration 700: Best Distance = 33306.46459911794,
Best Route = [56, 57, 11, 74, 64, 67, 54, 32, 68, 29, 26, 42,

Iteration 800: Best Distance = 33306.46459911794,
Best Route = [56, 57, 11, 74, 64, 67, 54, 32, 68, 29, 26, 42,

Iteration 900: Best Distance = 33306.46459911794,
Best Route = [56, 57, 11, 74, 64, 67, 54, 32, 68, 29, 26, 42,
```



# THANK YOU

Q&A