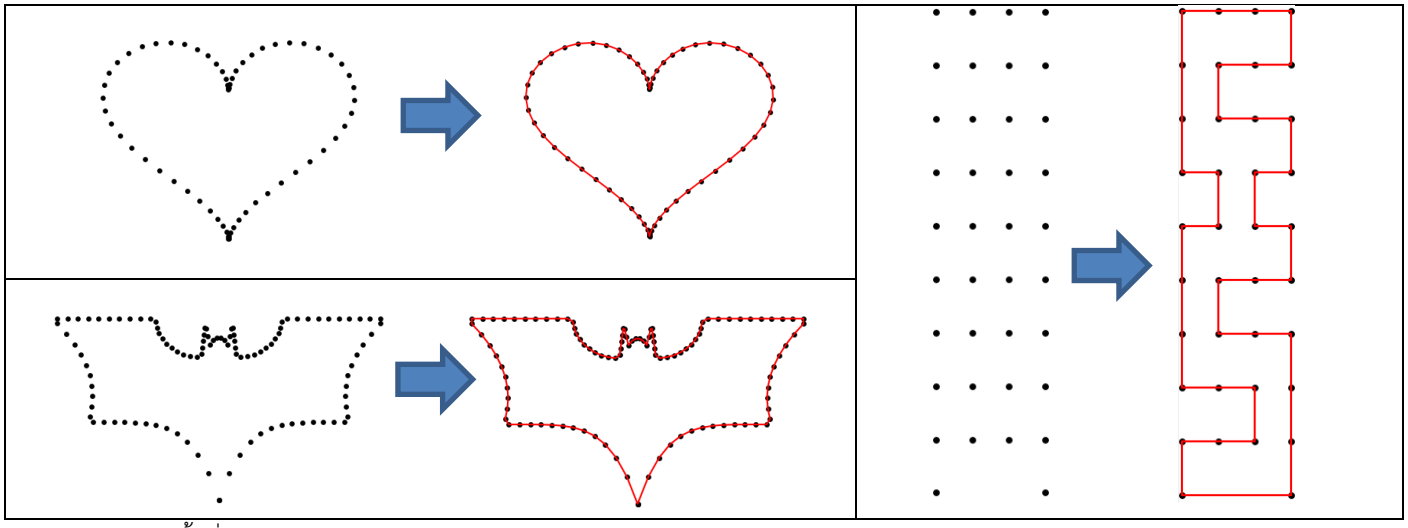
Assignment 5: Solving TSP using SA

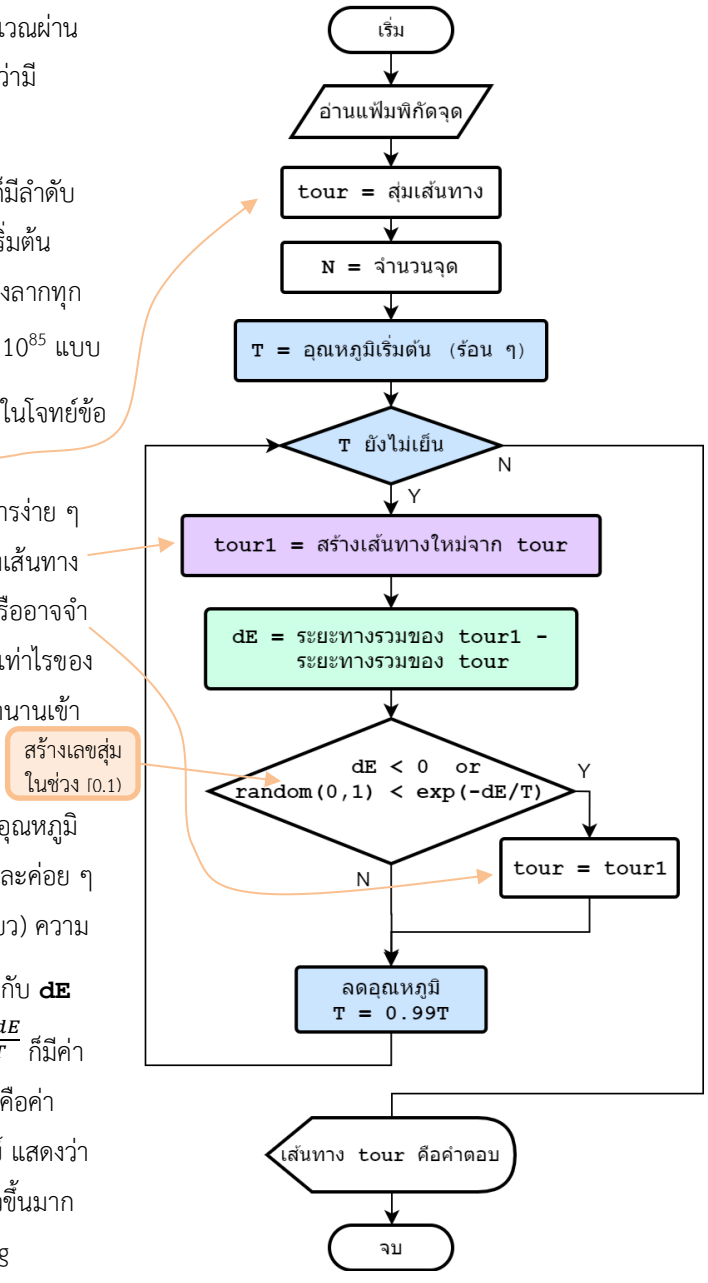
กำหนดให้มีจุดจำนวนหนึ่งบนระนาบสองมิติ ปัญหาคือ ต้องการหาวิธีลากเส้นผ่านทุกจุดผ่านจุดละครั้ง (ให้วนกลับมาที่จุดตั้งต้นด้วย) และที่ท้าทายคือ ให้ระยะทางรวมของเส้นที่ลากสั้นสุด ๆ ดังแสดงตัวอย่างในรูปข้างล่างนี้



หมายเหตุ: ปัญหานี้มีชื่อทางการว่า Travelling Salesman Problem (https://en.wikipedia.org/wiki/Travelling_salesman_problem) จุดคือเมือง และเส้นที่ลากคือ การเดินทางของพนักงานขายที่ต้องการตระเวณผ่านทุกเมืองเมืองละครั้งกลับที่เมืองเริ่มต้นด้วยระยะทางรวมที่น้อยสุด โดยถือว่ามีเส้นทางให้เดินทางตรง ๆ ทุก ๆ คู่เมืองใด ๆ

คำตอบที่ต้องการคือ ลำดับของจุดที่ต้องลากเส้น เช่น สมมติมี 4 จุด ก็มีลำดับการลากเส้นอยู่ (4-1) ! = 3! = 6 แบบ (ที่ลบ 1 เพราะเราต้องวกกลับจุดเริ่มต้น ลำดับ 1 → 2 → 3 → 4 → 1 จึงเหมือนกับ 2→ 3 → 4 → 1→ 2) การลองลากทุกแบบคงไม่ใช่วิธีที่ดีแน่ เช่น รูปหัวใจข้างบนนี้มี 63 จุด มี 62 ! ≈ 3.147 x 10^{85} แบบ

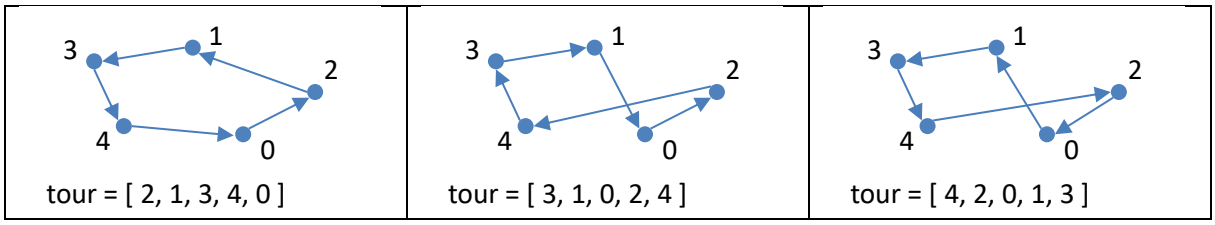
เนื่องจากปัญหานี้คลาสสิคมาก จึงมีผู้ออกแบบวิธีแก้ปัญหานี้มากมาย ในโจทย์ข้อนี้ เราจะใช้วิธีหนึ่งที่มีชื่อว่า Simulated Annealing https://en.wikipedia.org/wiki/Simulated_annealing มีหลักการง่าย ๆ คือ (ดูผังงานทางขวาประกอบ) เริ่มด้วยเส้นทางสุ่มหนึ่งทาง จากนั้นก็สร้างเส้นทางใหม่จากเส้นทางเดิม ถ้าเส้นใหม่สั้นกว่าก็จำไว้ ถ้ายาวขึ้น ก็อาจขว้างทิ้ง หรืออาจจำไว้ก็เป็นได้ ภายใต้ค่าความน่าจะเป็นอะไรบางอย่างที่ขึ้น กับว่า เป็นรอบที่เท่าไรของการทำงาน รอบแรก ๆ ก็จะยอมรับเส้นทางที่ยาวกว่าได้ง่าย แต่ยิ่งนานเข้านานเข้าโอกาสจะรับเส้นทางที่ยาวขึ้นก็น้อยลง ๆ



ให้สังเกตตัวแปร **T** ในผังงาน เป็นตัวแปรเก็บค่าอุณหภูมิ (เขาใช้คำว่าอุณหภูมิเพราะวิธีนี้จำลองกระบวนการอบเหนียวโลหะ ที่เริ่มหล่อด้วยอุณหภูมิสูงและค่อย ๆ ลดอุณหภูมิลง หากทำได้อย่างเหมาะสม จะทำให้ในท้ายสุดได้โลหะที่เหนียว) ความน่าจะเป็นในการยอมรับเส้นทางที่ยาวขึ้น จะขึ้นกับค่า $e^{\frac{-dE}{T}}$ ซึ่งขึ้นกับ **T** กับ **dE** ถ้า **T** มาก ค่า $e^{\frac{-dE}{T}}$ มีค่าใกล้ 1 คือ ยอมรับได้ง่าย ถ้า **T** น้อยลง ค่า $e^{\frac{-dE}{T}}$ ก็มีค่าลดลงเข้าหา 0 แสดงว่า จะยอมรับเส้นทางที่ยาวขึ้นยากขึ้น (สำหรับ **dE** คือค่าผลต่างของระยะทางของเส้นทางใหม่กับเส้นทางเดิม ถ้า **dE** น้อยกว่าศูนย์ แสดงว่าเส้นทางใหม่สั้นกว่า) ถ้าเส้นทางใหม่ยาวขึ้นไม่มาก ก็ยอมรับได้ง่ายกว่ายาวขึ้นมาก ลองดูภาพเคลื่อนไหวการใช้ simulated annealing กับปัญหา travelling salesman problem ได้ใน https://en.wikipedia.org/wiki/Simulated_annealing

สิ่งที่ต้องเข้าใจก่อนลงมือเขียนโปรแกรม มีหลายประเด็น (ค่อย ๆ อ่าน)

- input กับ output คืออะไร ?
  - input คือชื่อแฟ้มที่ภายในเก็บพิกัดของจุดต่าง ๆ ในระนาบ (ตรงนี้มี code อ่านแฟ้มได้เป็นลิสต์ที่เก็บพิกัดให้แล้ว)
  - output เป็นเส้นทางผ่านทุกจุด เช่น 2→4→1→0→3 ที่มีระยะทางรวมสั้น ๆ ยิ่งสั้น ยิ่งได้คะแนนมาก (สั้นสุดย่อมดีสุด แต่ถ้าต้องใช้กับแผนที่ที่มีจุดมาก ๆ อาจหาสั้นสุดได้ลำบาก)
- เส้นทางหรือลำดับของจุดที่พูดถึงมาตลอด เก็บอย่างไรในโปรแกรม แล้วจะคำนวณระยะทางอย่างไร ?
  - ใช้ลิสต์ 1 ตัว ให้ชื่อว่า **`tour`** เก็บลำดับของหมายเลขจุดของเส้นทาง เช่น โจทย์มีจุดทั้งหมด 5 จุด ลิสต์ `tour` ก็เก็บหมายเลข 0 ถึง 4 โดยลำดับของหมายเลขในลิสต์แทนเส้นทาง ดังตัวอย่างข้างล่างนี้

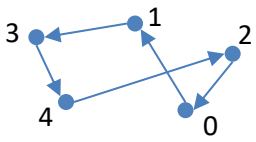


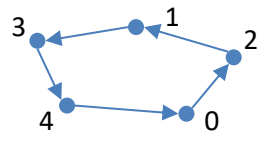
  - ใช้คำสั่ง **`dist(p1, p2)`** (เขียนเป็นชุดคำสั่งให้ใช้ได้แล้ว) ในการหาระยะสั้นสุด วัดตรง ๆ จากจุดหมายเลข `p1` ไปยังหมายเลข `p2`
  - ดังนั้น ถ้า `tour` มี 5 ช่อง ระยะทางรวมของเส้นทางที่กำหนดโดยลำดับของจุดใน `tour` ย่อมเท่ากับ

```
dist(tour[0], tour[1]) + dist(tour[1], tour[2]) + dist(tour[2], tour[3]) +
dist(tour[3], tour[4]) + dist(tour[4], tour[0])
```

  ในกรณีทั่วไป ต้องเขียนเป็นวงวนรวมระยะทางตามลำดับของจุดใน tour (อย่าลืมว่า ต้องมีระยะทางจากจุดท้ายสุดกลับมาจุดเริ่มต้นด้วย)

- แล้วต้องเขียนอะไรบ้าง ?
  - โปรแกรม (ในหน้าถัดไป) ทำงานยังไม่ค่อยน่าพอใจ ชุดคำสั่งทางคอลัมน์ขวาใต้แถบสีแสด เขียนทำงานตามผังงานที่แสดงก่อนนี้ (ต้องเขียนชุดคำสั่งเพิ่มเล็กน้อย) ส่วนชุดคำสั่งก่อนหน้าแถบสีแสด เกี่ยวข้องกับการแสดงผล การอ่านแฟ้ม และการคำนวณระยะทาง
  - จุดสำคัญที่ต้องปรับปรุง (บริเวณสีม่วง เขียว และ ฟ้า)
    - บริเวณพื้นสีฟ้า: เป็นคำสั่งควบคุมการทำงานแบบวงวน และการเลือกทำ ที่ต้องเขียนเพิ่มตามผังงาน
    - บริเวณพื้นสีม่วง: สร้างเส้นทางใหม่ จากเส้นทางเดิมใน `tour` (เขียนให้แล้ว แต่ไม่ค่อยดี)





      วิธีที่เขียนในโปรแกรมอาศัยการสุ่มเลข index ของ `tour` มาสองตำแหน่ง (ขอเน้นว่าสุ่มเลข index ไม่ได้สุ่มหมายเลขของจุด) โดยใช้คำสั่ง `random.randint(a,b)` ซึ่งคืนจำนวนเต็มสุ่มตั้งแต่ค่า `a` ถึง `b` เพื่อใช้สร้างลิสต์ใหม่ที่ได้ผลจากการสลับหมายเลขจุดที่เก็บในตำแหน่งที่สุ่ม เช่น `tour` เก็บ [4, 2, 0, 1, 3] ถ้าสุ่มได้ 1 กับ 2 ก็ได้เส้นทางใหม่จากการสลับข้อมูลที่ index 1 กับ 2 ของ `tour` คือสลับหมายเลขจุด `tour[1]` กับ `tour[2]` สร้างเป็นเส้นทางใหม่คือลิสต์ [4, 0, 2, 1, 3] (ในโปรแกรม เก็บใส่ตัวแปร `tour1`)

    - แล้วจะสร้างเส้นทางใหม่ แบบที่ต่างจากที่เขียนในตัวอย่างได้อย่างไร ?
      อันนี้ต้องจินตนาการเองว่า จะสร้างอย่างไรดี ที่ได้เส้นทางใหม่ อะไรที่มั่ว ๆ ก็ได้ ที่จะทำให้ระบบได้ทดลองเส้นทางโน่นเส้นทางนี่แบบใหม่ เผื่อจะเจออันที่ดีกว่า หรือถึงแม้แย่กว่า แต่ก็อาจนำไปสู่อันที่ดีกว่าในอนาคต (ซึ่งเราก็ไม่รู้ ก็ต้องสุ่มทำไปเรื่อย ๆ)
    - และก็ไม่จำเป็นต้องมีวิธีเดียวในการสร้างเส้นทางใหม่ ออกแบบไว้หลายวิธี แล้วก็สุ่มวิธีเอาว่าจะใช้แบบไหน ซึ่งอาจทำได้ดังนี้

```
r = random.random()    # random value in range [0, 1)
if r < 0.4:
    # use method 1 to create a new tour with probability of 0.4
    ...
elif r < 0.9:
    # use method 2 to create a new tour with probability of 0.5
    ...
else:
    # use method 3 to create a new tour with probability of 0.1
    ...
```

- บริเวณพื้นสีเขียว: หลังจากสร้างเส้นทางใหม่ได้ ก็ต้องคำนวณระยะทางรวมของเส้นทางใหม่ (`tour1`) ลบด้วย ระยะทางรวมของเส้นทางเดิม (`tour`) เก็บในตัวแปร **dE** (การคำนวณผลต่างตรงนี้ อาจทำแบบตรงไปตรงมา คือใช้วงวนบวกสะสมระยะทางตามลำดับของเส้นทางก็ได้ แต่ถ้าสังเกตดี ๆ จะพบว่า ไม่จำเป็นต้องทำแบบนั้น เพราะมีแค่ไม่กี่เส้นทางย่อยภายในที่เปลี่ยนแปลง ก็คำนวณเฉพาะตรงนั้น จะลดเวลาการคำนวณลงมากพอสมควร ดูโปรแกรมที่เขียนให้เป็นตัวอย่าง) ถ้า **dE** มีค่าลบ แสดงว่าเส้นทางใหม่สั้นกว่า ก็ยอมรับเส้นทางใหม่ แต่ถ้ายาวขึ้น (**dE** > 0) จะนำ **dE** กับค่า **T** (อุณหภูมิ) เข้าสูตรคำนวณความน่าจะเป็นในการยอมรับเส้นทางใหม่ (ดังที่อธิบายก่อนนี้)
- นอกจากนี้ อาจทดลองปรับปรุง ค่าเริ่มต้นของ **T** วิธีปรับลดค่าของ **T** และเงื่อนไขของ **T** ที่ทำให้ออกจากวงวน โดยการอบเหนียวมีหลักการคร่าว ๆ ว่า <u>เริ่มที่อุณหภูมิร้อน ๆ ลดอุณหภูมิทีละน้อย ๆ (อย่าฮวบฮาบ) และเลิกวนทำงาน เมื่อเย็นพอ</u>

- รู้ได้อย่างไรว่า ได้โปรแกรมที่ทำงานได้ดี ยอมรับได้
  - มีตัวอย่างแฟ้มแผนที่หลายแฟ้ม (อยู่ใน<u>ข้อมูลที่ใช้ทดสอบ</u>) หลายแฟ้ม มองดูด้วยตา ก็รู้ว่าคำตอบที่ดีสุดคืออะไร

    เช่น `I.txt, H.txt, E.txt, heart.txt, batman.txt, cat.txt, ...`

```
# Prog-05: Solving TSP by simple-SA
# ID: Name
# ...
# I hereby declare that I coded the SA part by myself.

import math
import random

from matplotlib import pyplot as plt
from matplotlib import animation, rc
#-----------------------------------------------
def animate(X, Y, tour_log):
  fig, ax = plt.subplots()

  line, = ax.plot(X, Y, color='blue')
  line.set_marker('o')
  line.set_markersize(4)
  min_x, max_x = min(X), max(X)
  min_y, max_y = min(Y), max(Y)
  w = max_x - min_x
  h = max_y - min_y
  width = 6 if w > h else 6*w/h
  height = 6 if h > w else 6*h/w
  fig.set_size_inches(max(4,width), max(4,height))
  dy = abs(0.1*(max_y - min_y))
  txt = ax.text(min_x, min_y-1.7*dy, '', fontsize=10)
  ax.set_ylim(min_y-2*dy, max_y+dy)
  ani = animation.FuncAnimation(fig, next_path,
                  frames=len(tour_log),
                  fargs=[X, Y, tour_log, line, txt],
                  interval=10,
                  repeat=False,
                  blit=True)
  plt.show()

def next_path(k, X, Y, tour_log, line, txt):
  T, tour = tour_log[k]
  N = len(tour)
  d = sum(D[tour[i]][tour[i-1]] for i in range(N))
  txt.set_text("T = " + str(round(T,3)) + \
               "\ntour length = " + str(round(d,2)))
  x = [X[p] for p in tour] + [X[tour[0]]]
  y = [Y[p] for p in tour] + [Y[tour[0]]]
  line.set_data(x, y)
  if k == len(tour_log)-1:
    txt.set_color('red')
    line.set_color('red')
  return (line, txt)

def read_map(fname):
  f = open(fname)
  X, Y = [], []
  for line in f:
    if line.strip()=='EOF': break
    d = line.split()
    X.append(float(d[1]))
    Y.append(float(d[2]))
  f.close()
  return X, Y

def dist(p1, p2):
  return D[p1][p2]
```

download code นี้ได้

ไม่เปลี่ยนแปลงคำสั่งสีแดงเด็ดขาด ก่อนส่งให้ลบคำสั่งพื้นเหลืองให้หมด (สีแดงพื้นเหลืองก็ลบ)

```
def calc_all_pair_distances(X, Y):
  N = len(X)
  D = [[0.0]*N for i in range(N)]
  for i in range(N):
    for j in range(i, N):
      D[i][j] = D[j][i] = \
                math.hypot(X[i]-X[j], Y[i]-Y[j])
  return D

#------------  main program ----------------------
map_name = input("Map file name: ")
X, Y = read_map(map_name)
D = calc_all_pair_distances(X, Y)
N = len(X)                  # number of points
tour_log = []
tour = list(range(N))       # [0,1,2,3,4,...,N-1]

#--- let's the SA begins --------------
loop = 0

T = N*10                    # starting temperature
????? ? ? ????? :           # while not finished

    # create a new tour from the current tour

    i = random.randint(0, N-2)
    j = random.randint(i+1, N-1)

    tour1 = list(tour)  # copy list
    tour1[i], tour1[j] = tour1[j], tour1[i]  # swap

    # compute dE = length_of_tour1 - length_of_tour

    dE = dist(tour1[i-1], tour1[i      ]) + \
         dist(tour1[i  ], tour1[i+1    ]) + \
         dist(tour1[j-1], tour1[j      ]) + \
         dist(tour1[j  ], tour1[(j+1)%N]) - \
         dist(tour[ i-1], tour[ i      ]) - \
         dist(tour[ i  ], tour[ i+1    ]) - \
         dist(tour[ j-1], tour[ j      ]) - \
         dist(tour[ j  ], tour[(j+1)%N ])

    # accept new tour if shorter or within prob.
    if ????????? :
        tour = tour1

    ? ?? ????   # lower the temperature

    # keep tour changes in tour_log
    loop += 1
    if loop % N == 0: tour_log += [[T, tour]]

tour_log += [[T, tour]] # the final tour


print(tour)


animate(X, Y, tour_log ) # let's animate
```

สิ่งที่ต้องทำ

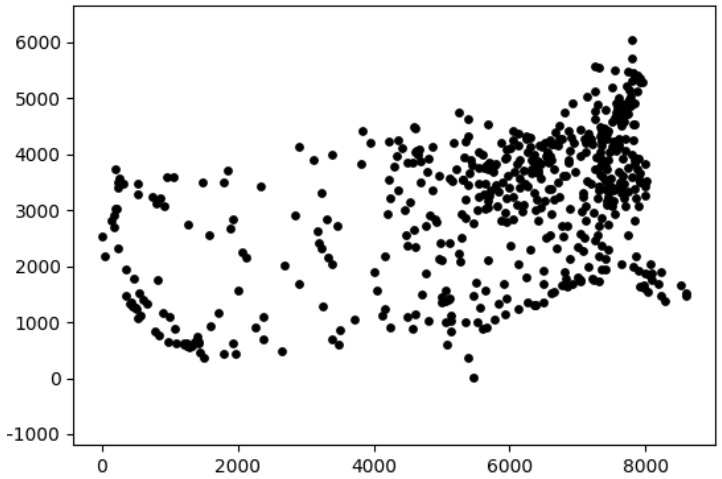
1. download โปรแกรมต้นแบบ และ ข้อมูลที่ใช้ทดสอบ (unzip ข้อมูลให้อยู่ใน folder เดียวกับตัวโปรแกรม)
2. เขียนคำสั่งควบคุมการทำงานแบบวงวน และการเลือกทำ (บริเวณสีฟ้า) ตามผังงานให้เรียบร้อย
   สำหรับการสร้างเลขสุ่มในช่วง [0, 1) ที่เขียนในผังงาน ทำได้ด้วยคำสั่ง `random.random()`
3. ลองสั่งทำงานโปรแกรมดู (กับแฟ้มแผนที่ เช่น heart.txt จะเห็นได้ว่า ได้เส้นทางที่ไม่ดี
4. หากไม่ต้องการดูภาพการเปลี่ยนแปลง แต่อยากดูผลสุดท้ายเลย ให้เปลี่ยนบรรทัดสุดท้ายของโปรแกรมเป็น

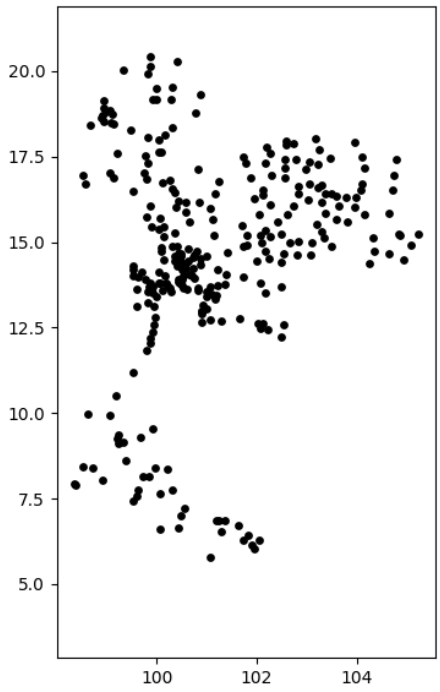
   ```
   animate( X, Y, tour_log[-1:] )
   ```

   ของเดิมคือ ให้นำการเปลี่ยนแปลงเส้นทางที่เก็บใน `tour_log` ไปแสดงทั้งหมด ถ้าเปลี่ยนเป็น `tour_log[-1:]` คือให้แสดงแค่เส้นทางสุดท้าย หรือถ้าเปลี่ยนเป็น `tour_log[-500:]` ก็จะแสดง 500 เส้นทางสุดท้าย เป็นต้น
5. ออกแบบวิธีการสร้างเส้นทางใหม่ให้ `tour1` จากเส้นทางใน `tour` และคำนวณ `dE` ด้วย (บริเวณสีม่วง และเขียว)
6. แล้วก็ลองสั่งทำงานดูกับข้อมูลทดสอบว่า ได้ผลดีขึ้นไหม
7. อาจลองปรับให้ การลดอุณหภูมิลดให้ช้าลงหน่อย เช่น จาก `T *= 0.99` เปลี่ยนเป็น `T *= 0.999` หรืออย่างอื่น ก็อย่าลืมว่า ถ้าเปลี่ยนช้า ก็ทำงานช้าตามไปด้วย
8. อาจลองสร้างหลายวิธี แล้วใช้การสุ่มเพื่อเลือกวิธีที่ได้เขียนไว้ตามความน่าจะเป็นที่ตั้งให้ (ดังโครงชุดคำสั่งที่เขียนให้ก่อนนี้)
9. ทำขั้นตอนที่ 5, 6, 7, 8 เรื่อย ๆ โดยทดลองกับแฟ้มข้อมูลที่ให้มา
10. อย่าลืมเขียน comment แสดงเลขประจำตัว ชื่อ และข้อความยืนยันการทำการบ้านด้วยตนเอง และไม่แก้ไขส่วนใด ๆ ที่เป็นสีแดงในโปรแกรมที่ให้มา
11. **ลบชุดคำสั่งที่มีพื้นสีเหลือง (แสดงในหน้าที่แล้ว) ออกให้หมด**
12. ตั้งชื่อแฟ้ม .py ให้ถูกต้อง แล้วส่งทาง CourseVille (ตั้งชื่อแฟ้มให้ตรงตามที่ระบุใน CourseVille ด้วย)

การให้คะแนน

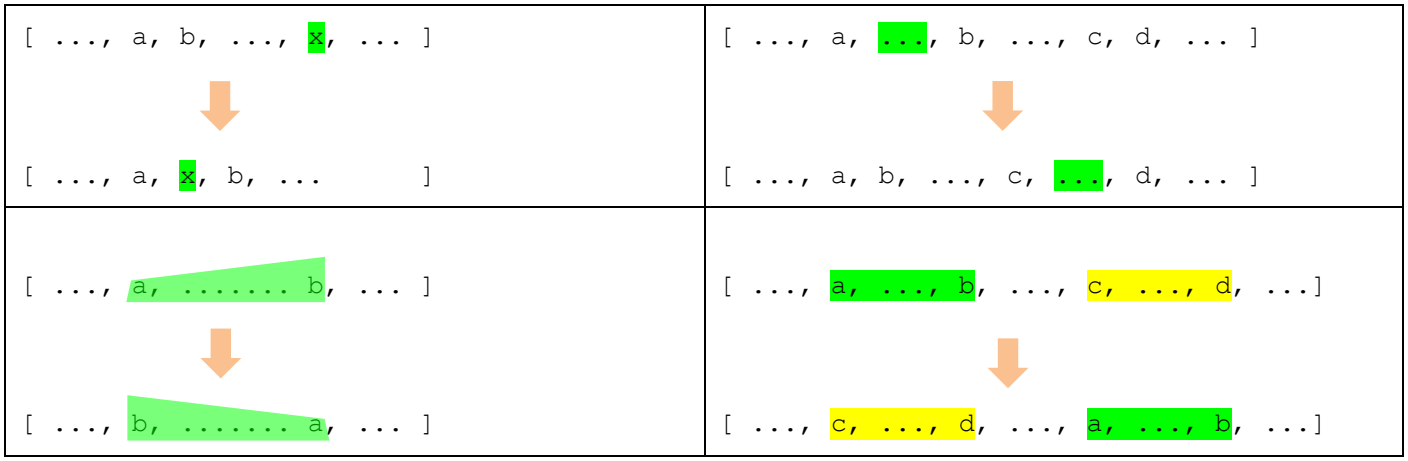
โปรแกรมที่ได้รับมาทั้งหมด จะถูกสั่งทำงานกับข้อมูลทดสอบชุดหนึ่ง คะแนนจะขึ้นกับผลที่ได้ และเวลาที่ใช้ เมื่อเทียบกับโปรแกรมทั้งหมด

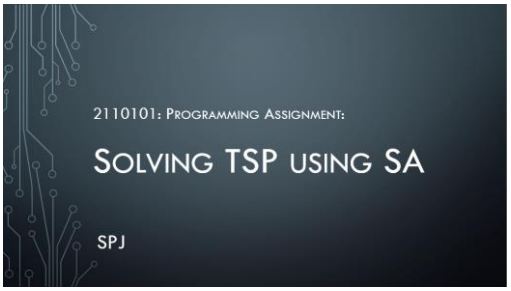




ข้อแนะนำ

ลองสร้างเส้นทางใหม่ในรูปแบบต่าง ๆ ข้างล่างนี้ดู





https://youtu.be/v5DlA49oHW8