Software Programming

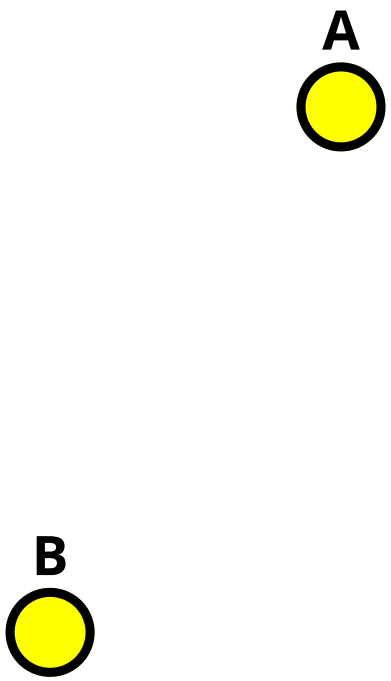
2019-2 SW프로그래밍 (YCS1002)

SWP11

Traveling Salesman Problem

Traveling Salesman Problem

- "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?"

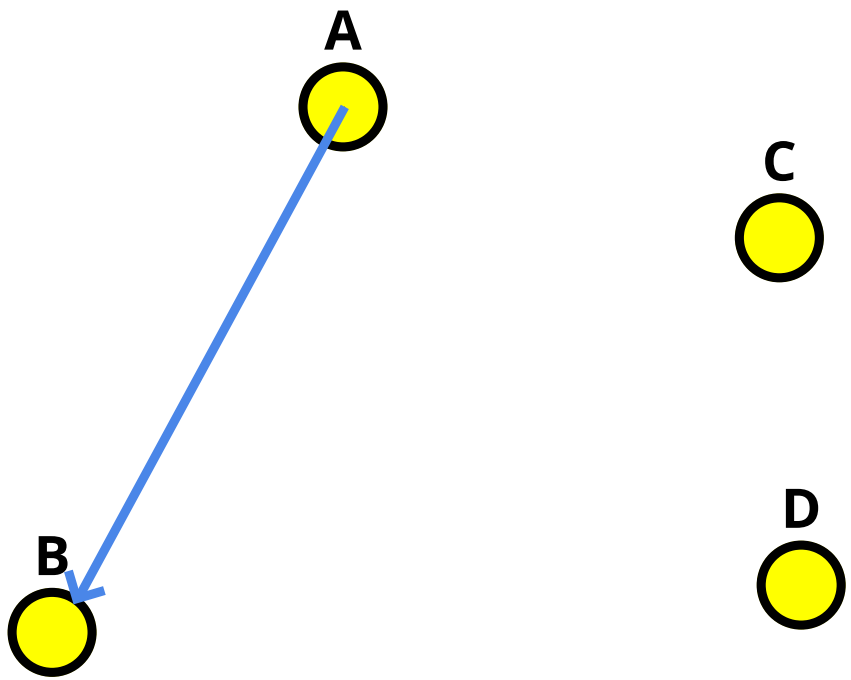


1. 랜덤하게
2. 그리디 알고리즘
3. 투옵트?방식으로 경로를 줄여가기
-> 트레블링 세일즈맨 프로브의 minimum condition

다음 스텝_+a로 추가할 수 있는 아이디어
: simulated annealing
: 기타 독창적인 아이디어

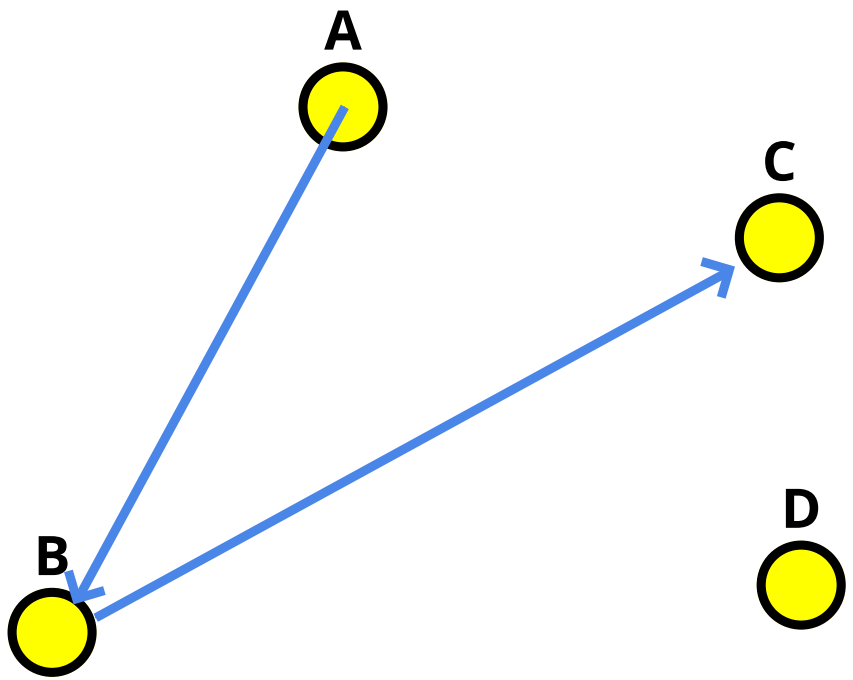
Traveling Salesman Problem

Path: A-B



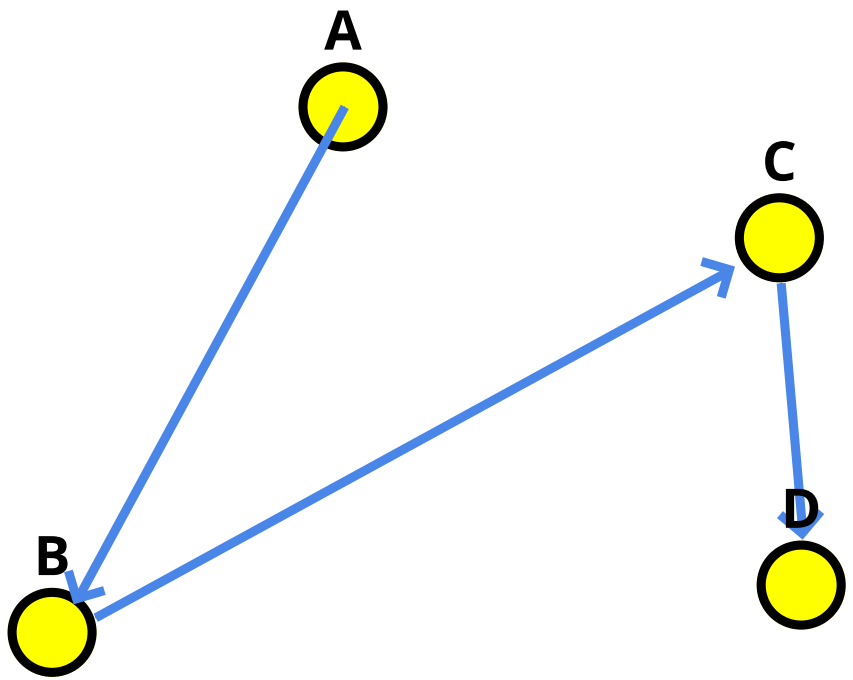
Traveling Salesman Problem

Path: A-B-C



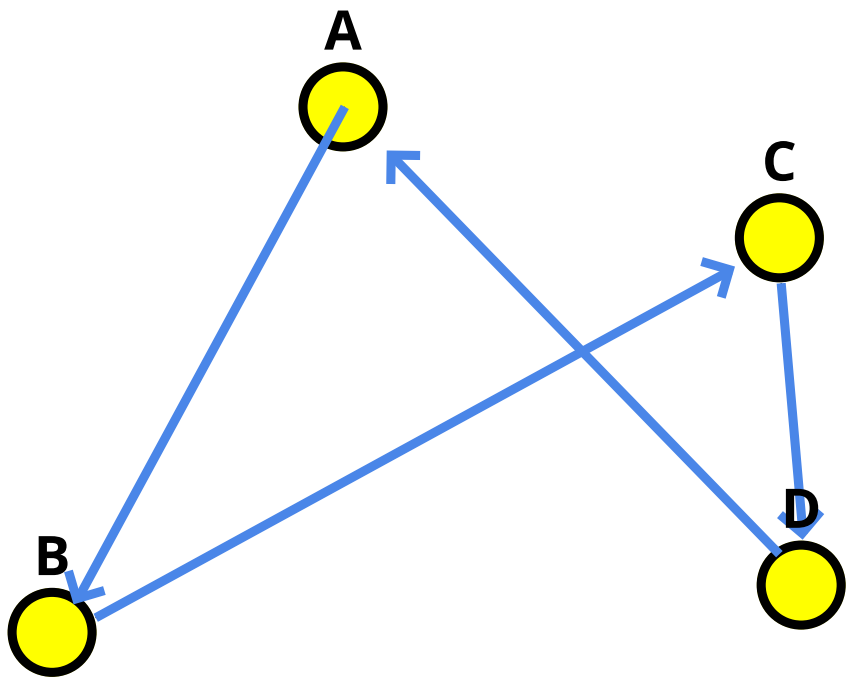
Traveling Salesman Problem

Path: A-B-C-D



Traveling Salesman Problem

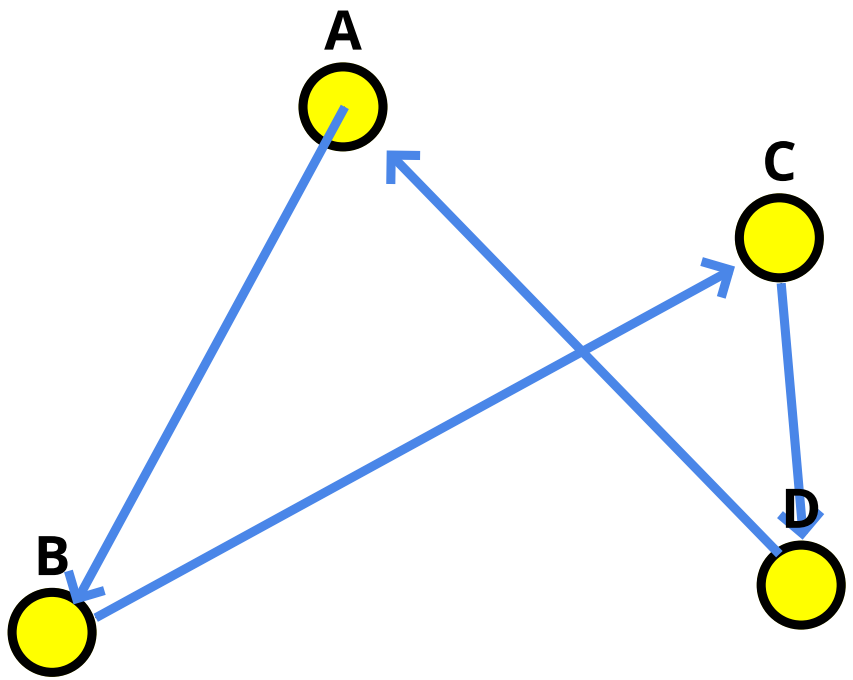
Path: A-B-C-D-A



Traveling Salesman Problem

Path: A-B-C-D-A

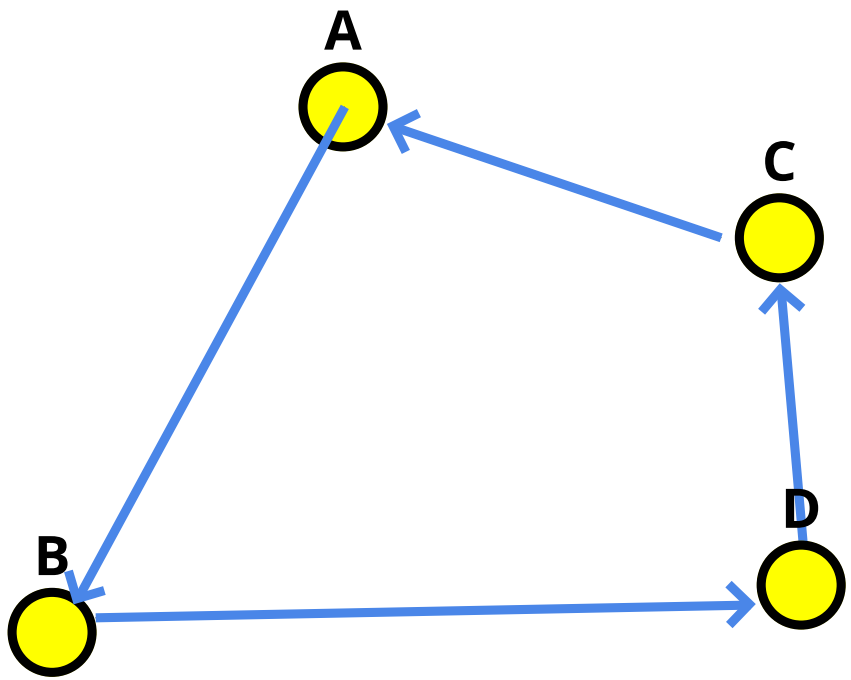
Shortest path?

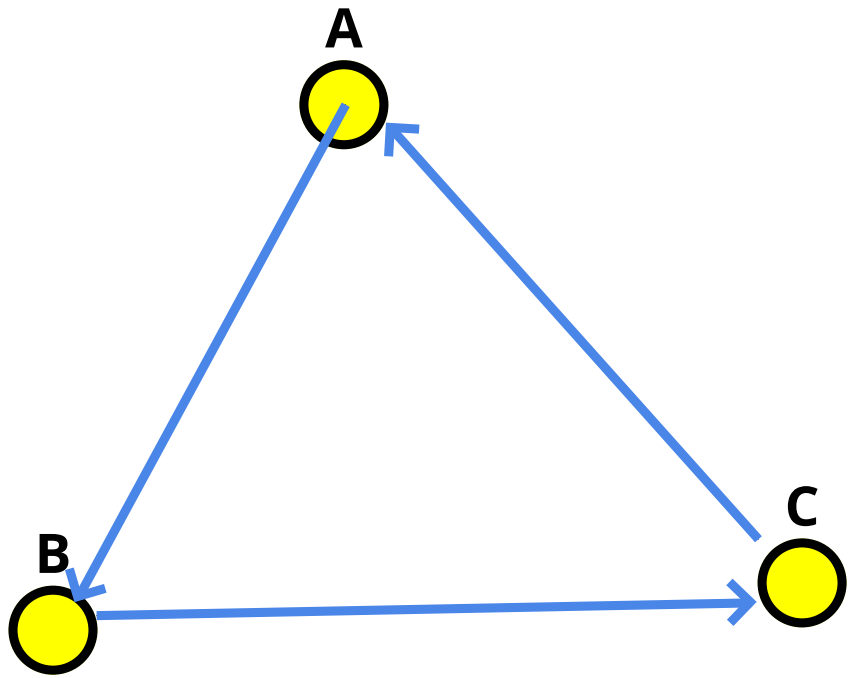


Traveling Salesman Problem

Shortest path?

Path: A-B-D-C-A





A	B	C	A
A	C	B	A
B	A	C	B
B	C	A	B
C	A	B	C
C	B	A	C

$$3 \times 2 \times 1 = 3!$$

3 cities = 3! = 6

4 cities = 4! = 24

5 cities = 5! = 120

50 cities = 50! = 3.0414093e+64

50 cities = 50! = 3.0414093e+**64**

100,000,000,000 lightyears = 9.46e+**26** meters

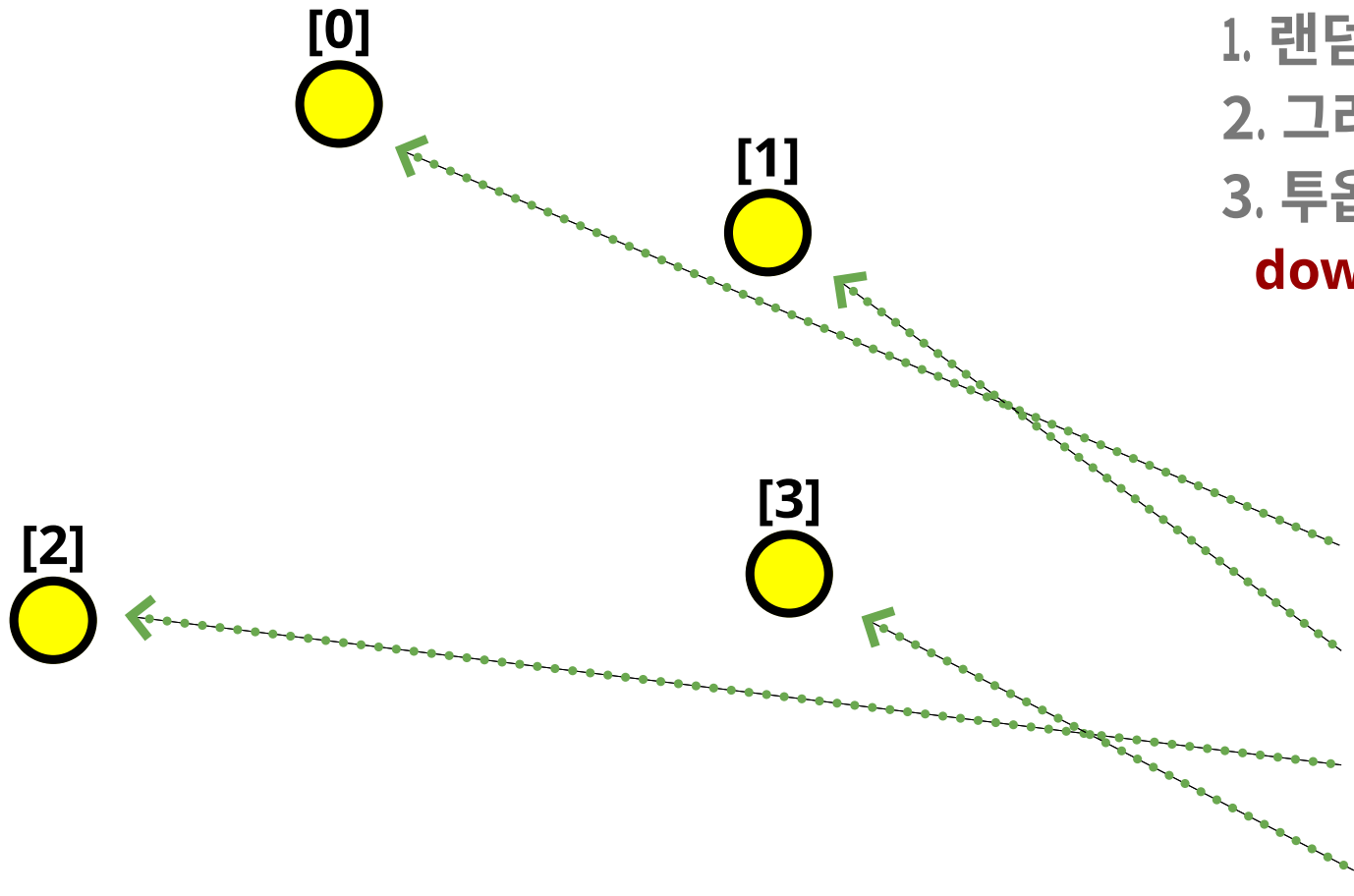
It is simply impossible.

Traveling Salesman Problem Visualization

Implementation

노드 내 각 인덱스에 도시정보가 저장되어있음

```
self.path = []
```



1. 랜덤으로 만들기
2. 그리디 쓰기
3. 투옵트로 개선하기

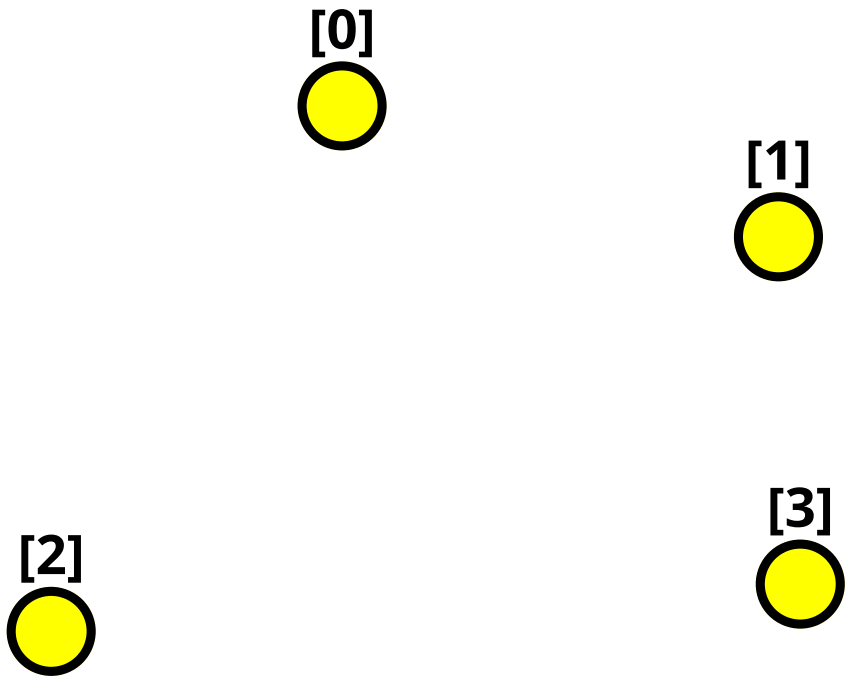
downloaded list of cities

클래스 내 속성

```
self.nodes
```

인덱스 내 도시 정보

[0]	[0, -301.0, 187.0]
[1]	[1, 302.0, 14.0]
[2]	[2, 343.0, -138.0]
[3]	[3, -445.0, 136.0]

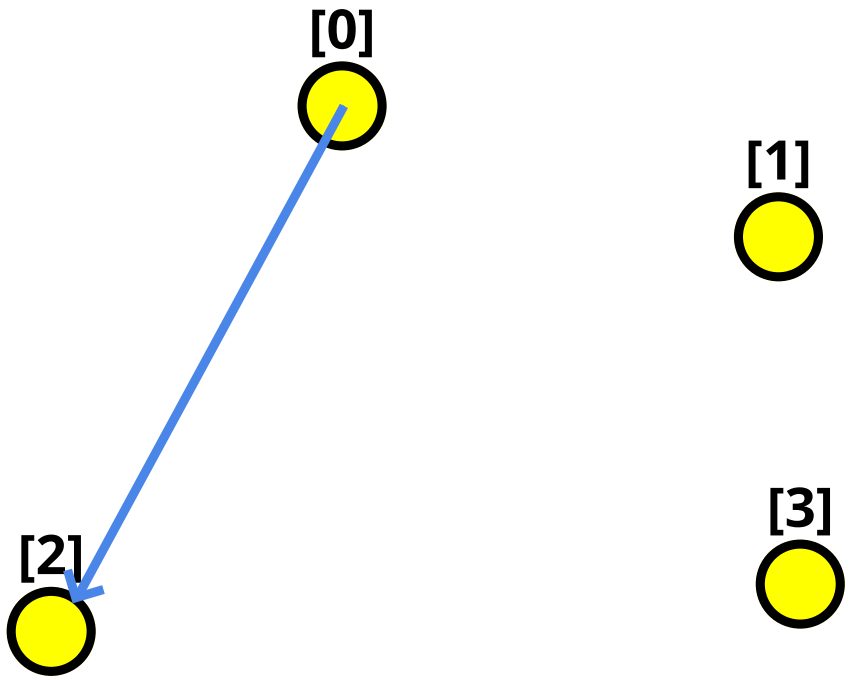



```
self.path = [0]
```

0

self.nodes

[0]	[0, -301.0, 187.0]
[1]	[1, 302.0, 14.0]
[2]	[2, 343.0, -138.0]
[3]	[3, -445.0, 136.0]

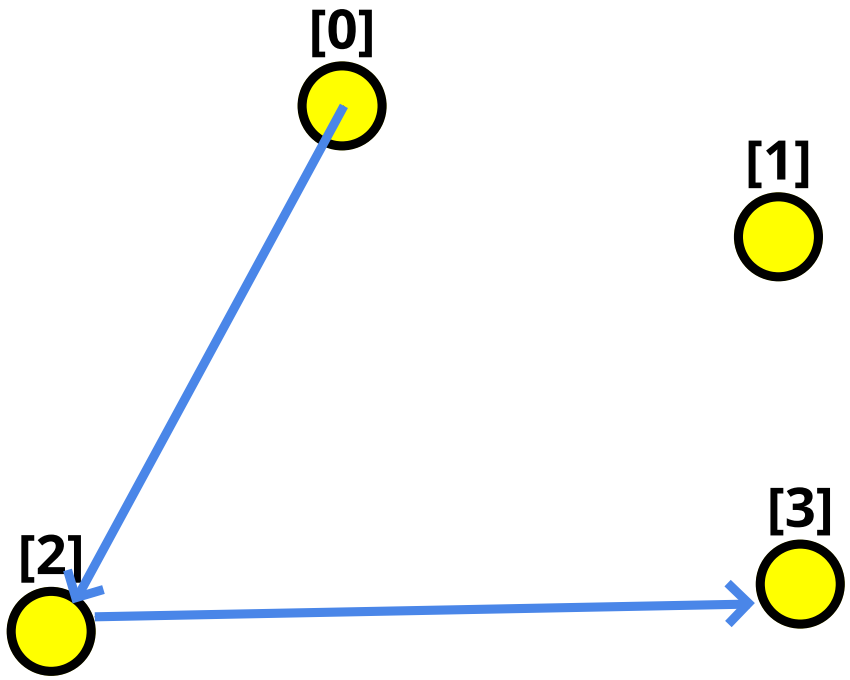



```
self.path = [0,2]
```

0
2

self.nodes

[0]	[0, -301.0, 187.0]
[1]	[1, 302.0, 14.0]
[2]	[2, 343.0, -138.0]
[3]	[3, -445.0, 136.0]

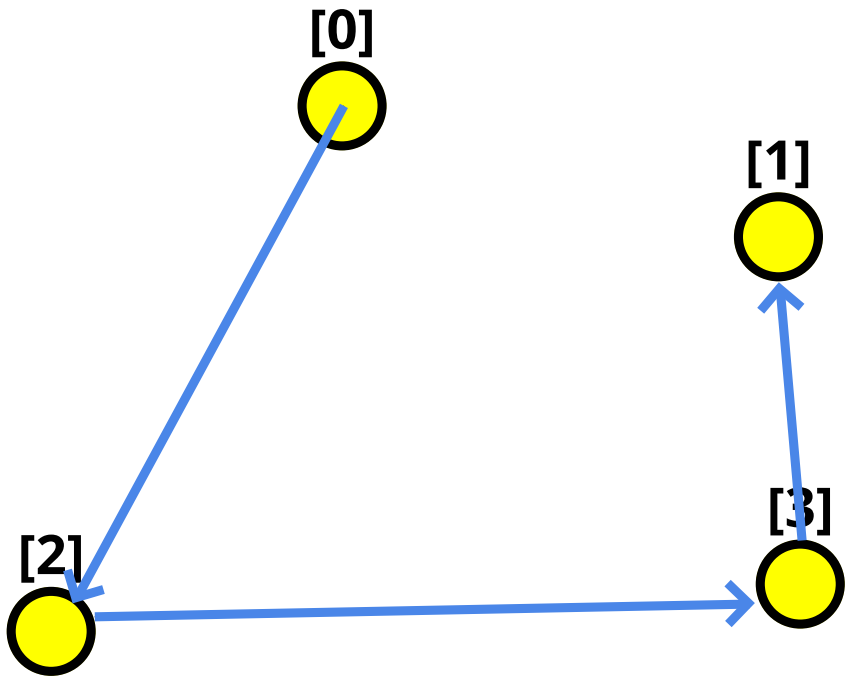


`self.path = [0,2,3]`

0
2
3

`self.nodes`

[0]	[0, -301.0, 187.0]
[1]	[1, 302.0, 14.0]
[2]	[2, 343.0, -138.0]
[3]	[3, -445.0, 136.0]

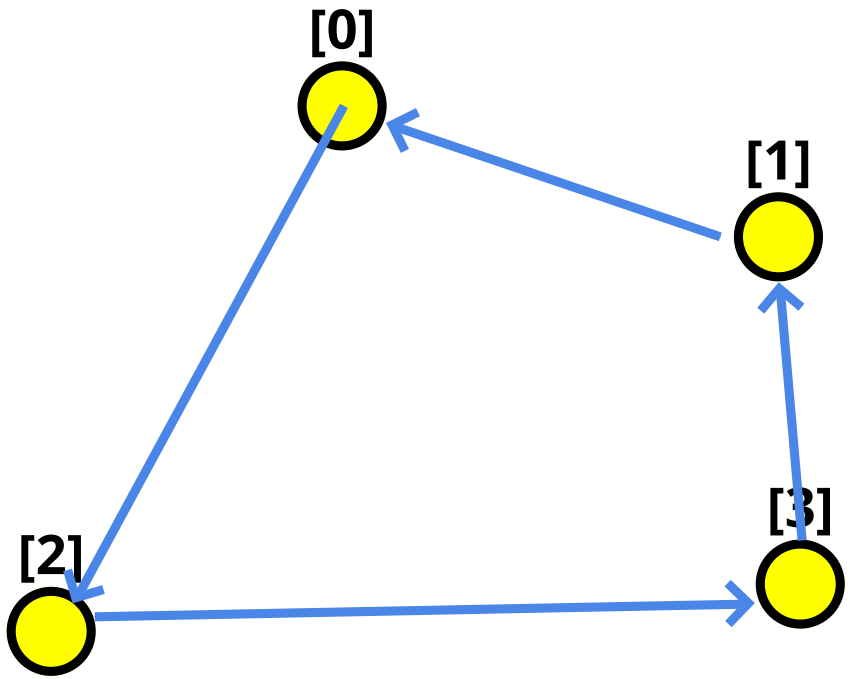


`self.path = [0,2,3,1]`

0
2
3
1

`self.nodes`

[0]	[0, -301.0, 187.0]
[1]	[1, 302.0, 14.0]
[2]	[2, 343.0, -138.0]
[3]	[3, -445.0, 136.0]



`self.path = [0,2,3,1,0]`

0
2
3
1
0

다시 처음 도시로
돌아오는 방식으로
path가 형성됨

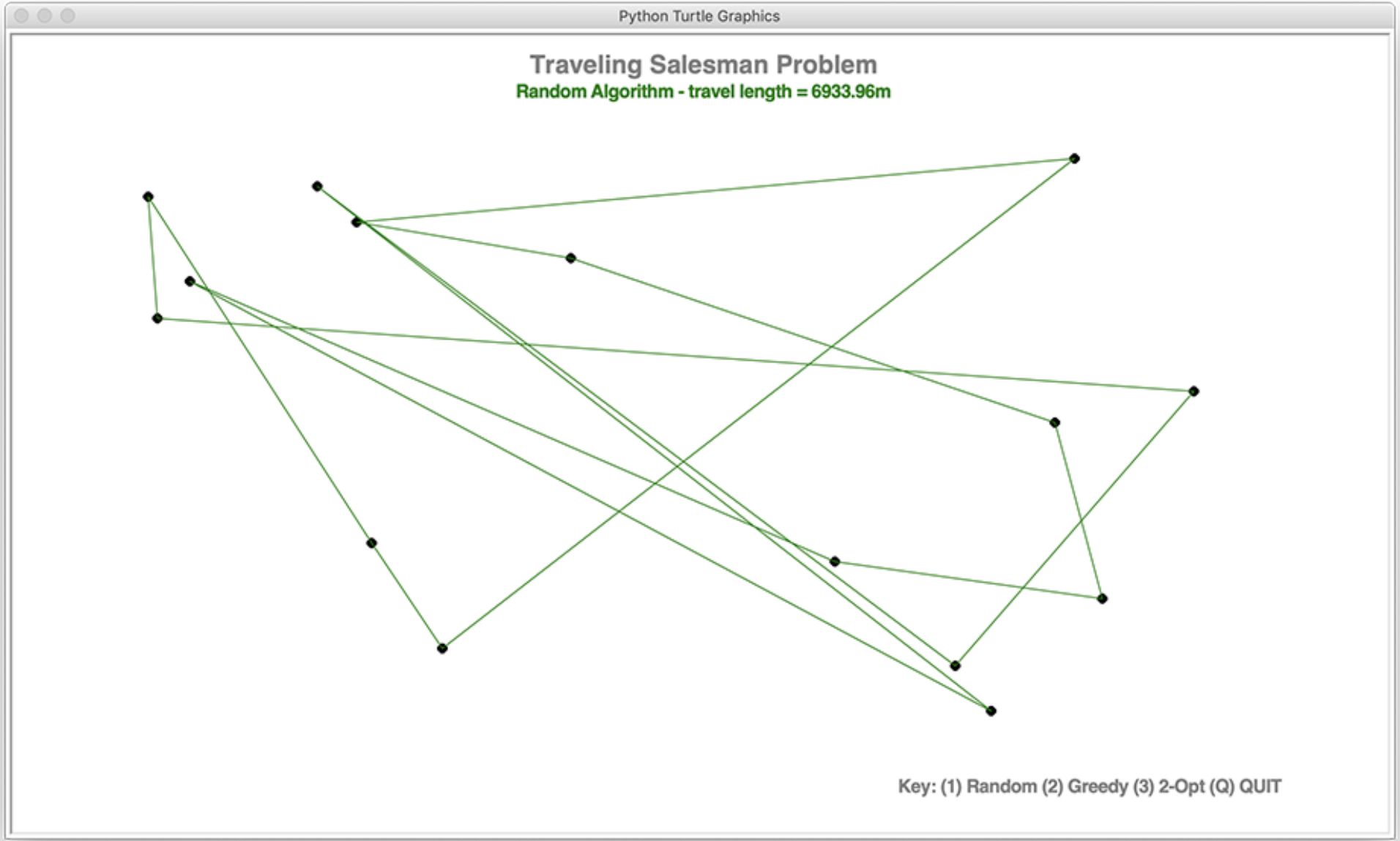
`self.nodes`

[0]	[0, -301.0, 187.0]
[1]	[1, 302.0, 14.0]
[2]	[2, 343.0, -138.0]
[3]	[3, -445.0, 136.0]

Strategies for TSP

- Random Algorithm
- Greedy Algorithm
- 2-Opt Algorithm

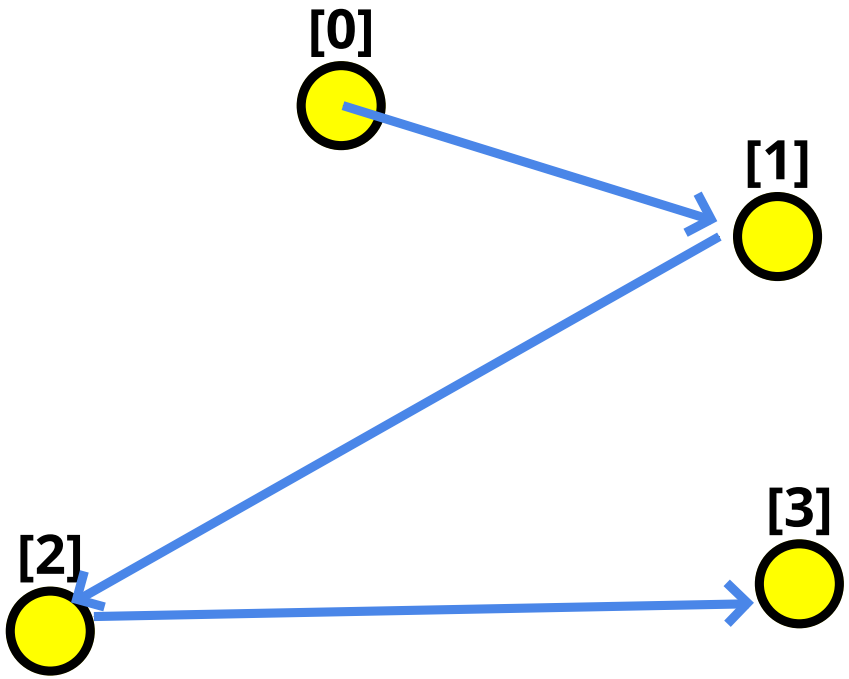
Random Algorithm


Python Turtle Graphics
Traveling Salesman Problem
Random Algorithm - travel length = 6933.96m
Key: (1) Random (2) Greedy (3) 2-Opt (Q) QUIT

```
self.path = [0,1,2,3]
```

0
1
2
3

list(range(n)) can make a path

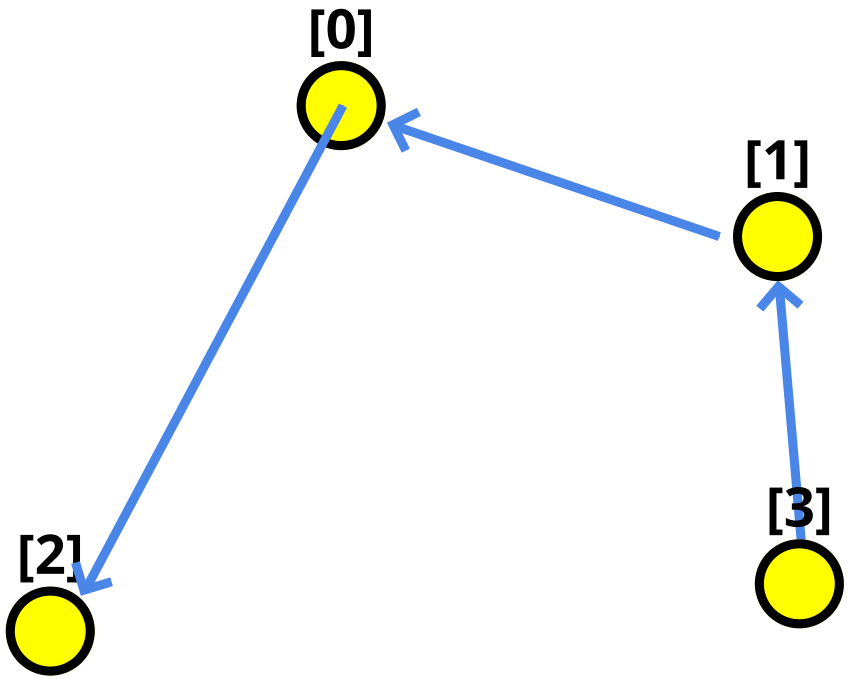


```
path = [3,1,0,2]
```

3
1
0
2

랜덤하게 경로를 섞어줌

shuffle the path using random.shuffle(path)

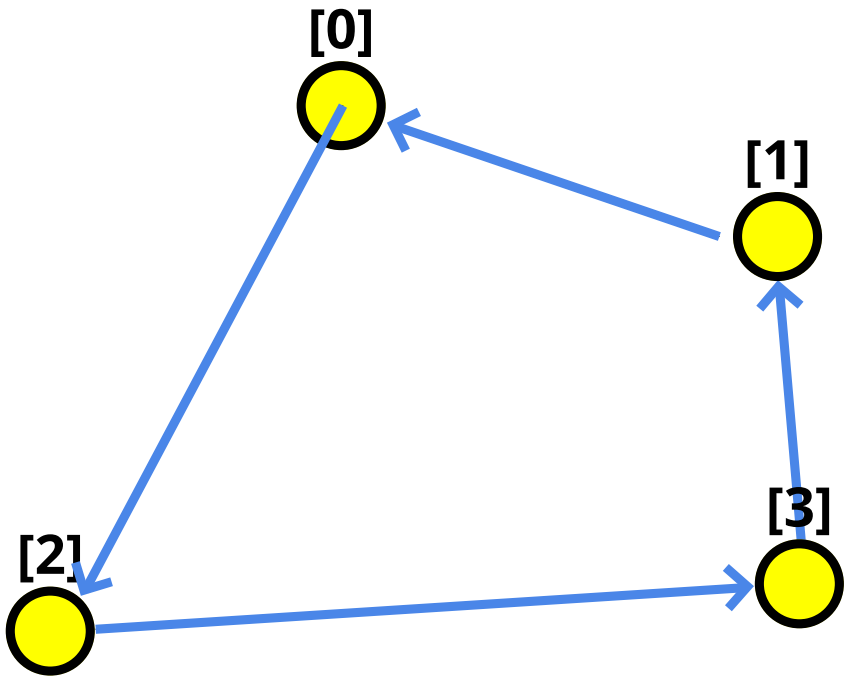


```
path = [3,1,0,2,3]
```

3
1
0
2
3

처음 도시로 돌아오게_처음 엘리먼트를 맨 뒤로 append 시켜주면 됨

Lastly, add path[0] to close the path



코드를 짜면 이렇게 된다

```
path = list(range(10))
print('path:', path)
```

```
path: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

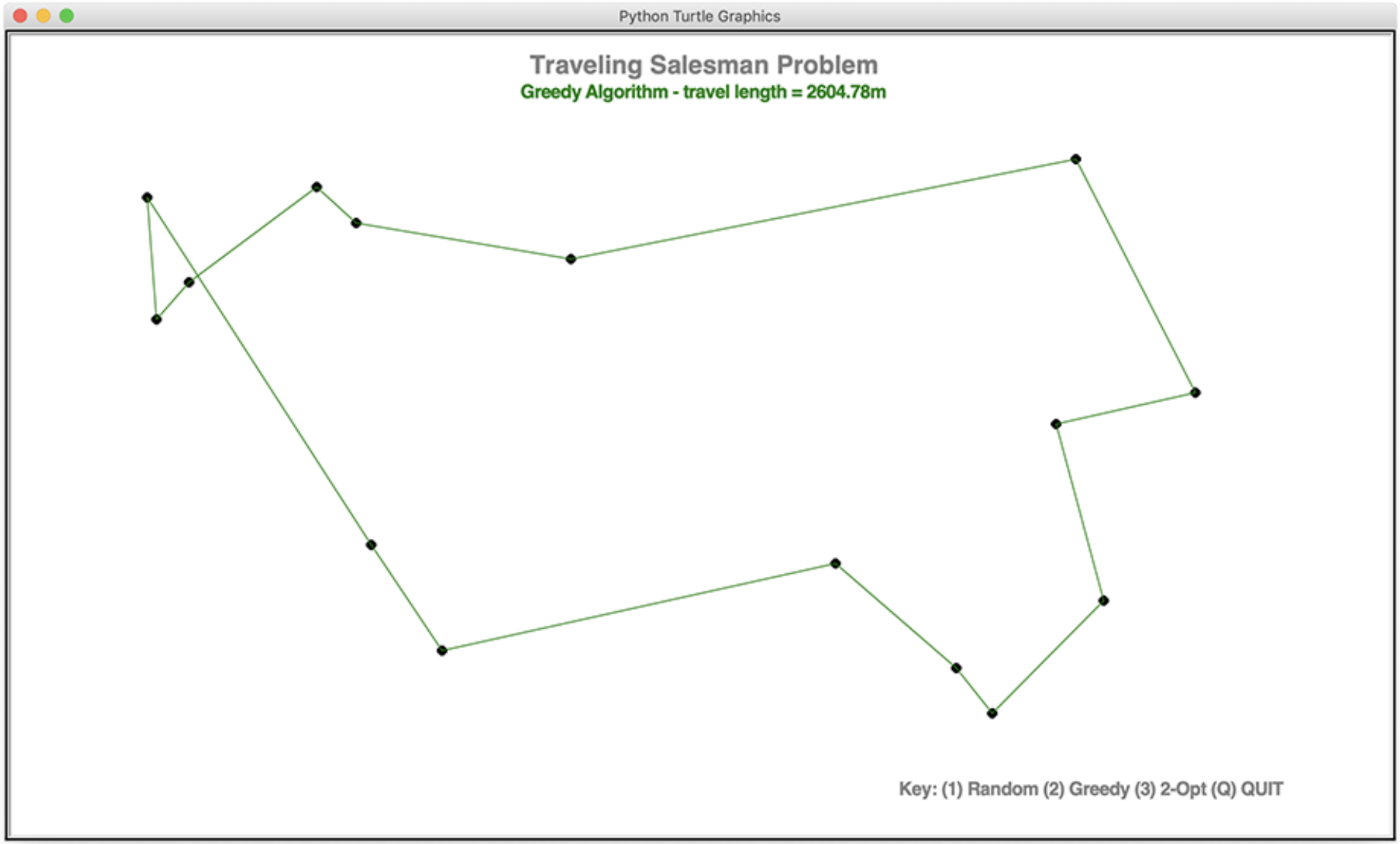
```
import random

random.shuffle(path)
print('path:', path)
```

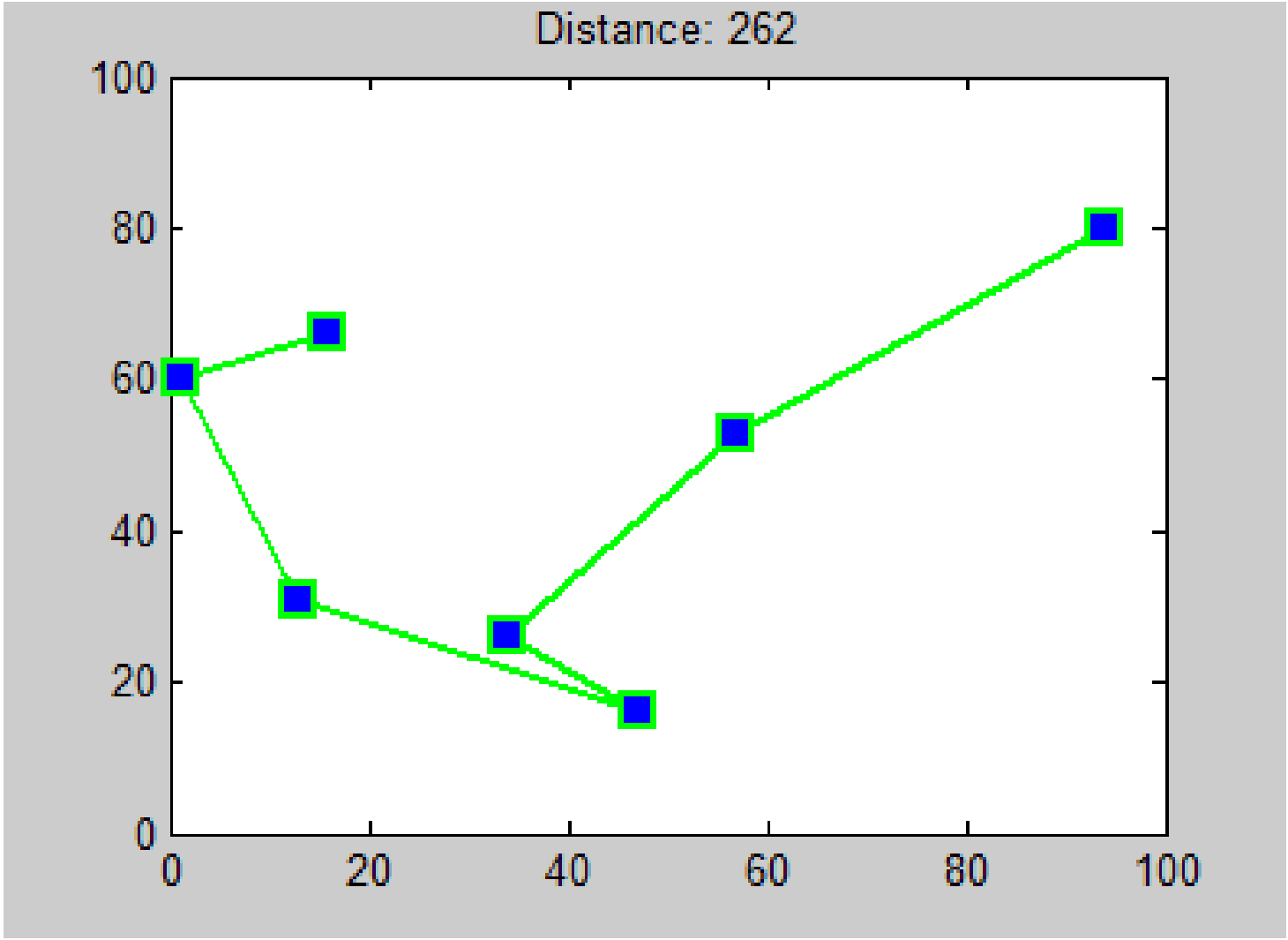
```
path: [3, 4, 9, 0, 6, 2, 1, 7, 5, 8]
```

```
path.append( path[0] ) # return to the first city
print('path:', path)
```

```
path: [3, 4, 9, 0, 6, 2, 1, 7, 5, 8, 3]
```

Greedy Algorithm

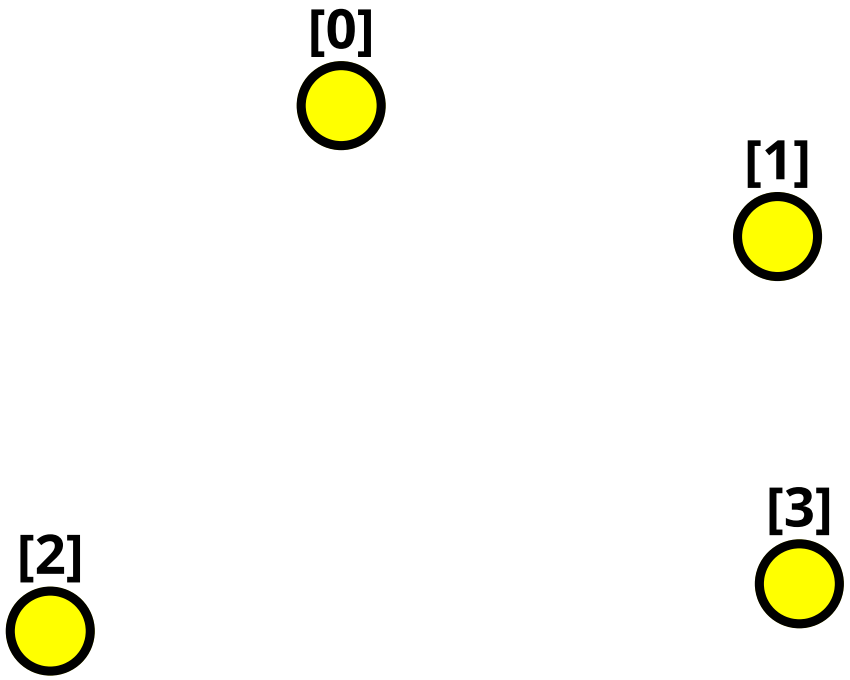

Python Turtle Graphics
Traveling Salesman Problem
Greedy Algorithm - travel length = 2604.78m
Key: (1) Random (2) Greedy (3) 2-Opt (Q) QUIT



https://upload.wikimedia.org/wikipedia/commons/2/23/Nearestneighbor.gif

```
pool = [0,1,2,3]
path = []
```

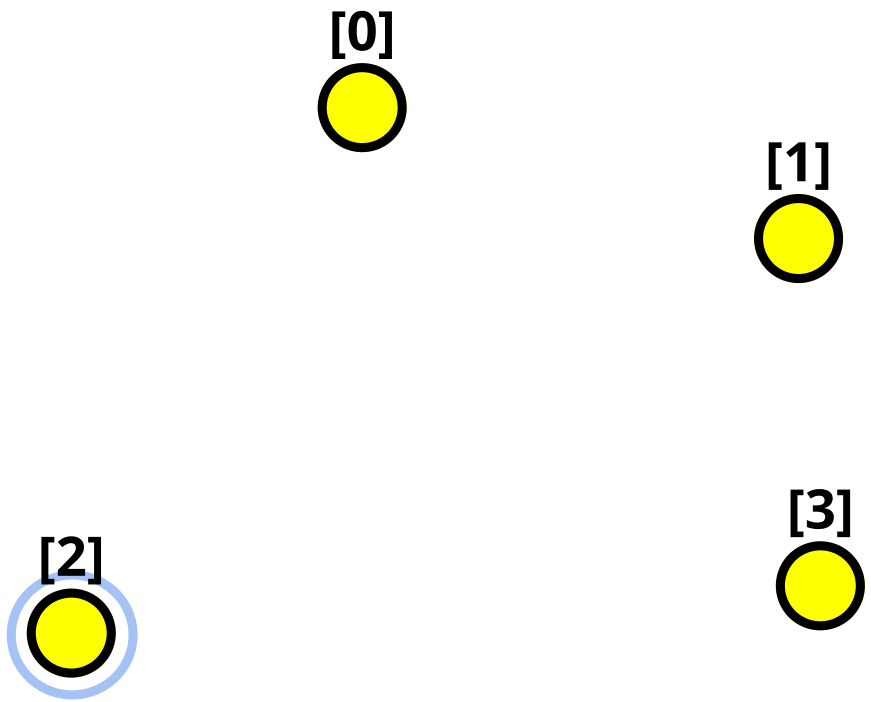
Initial path is **empty**



```
pool = [0,1,3]
path = [2]
```

2

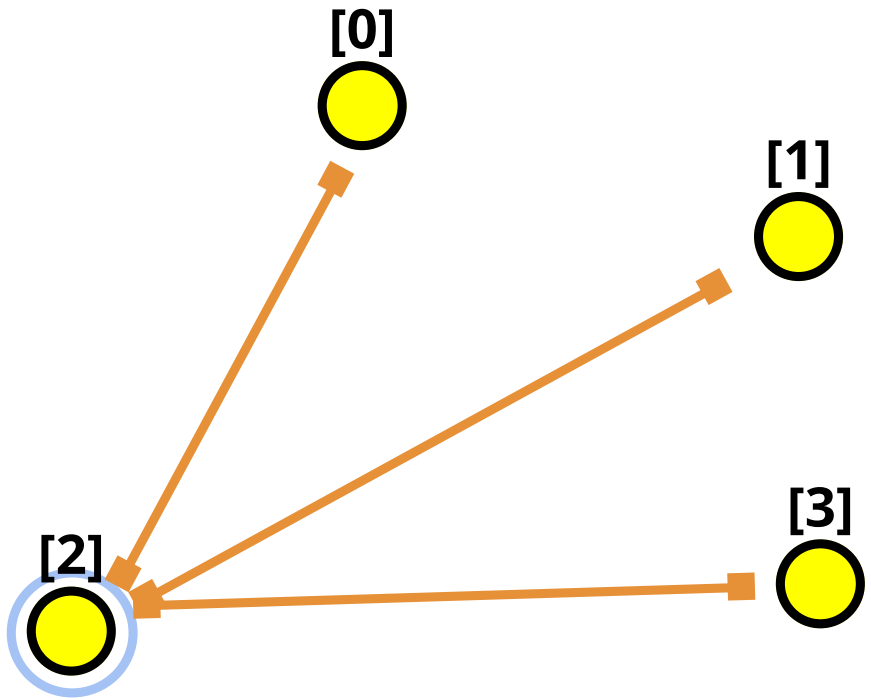
Pop a city randomly from the pool
and **append** to the path



```
pool = [0,1,3]
path = [2]
```

2

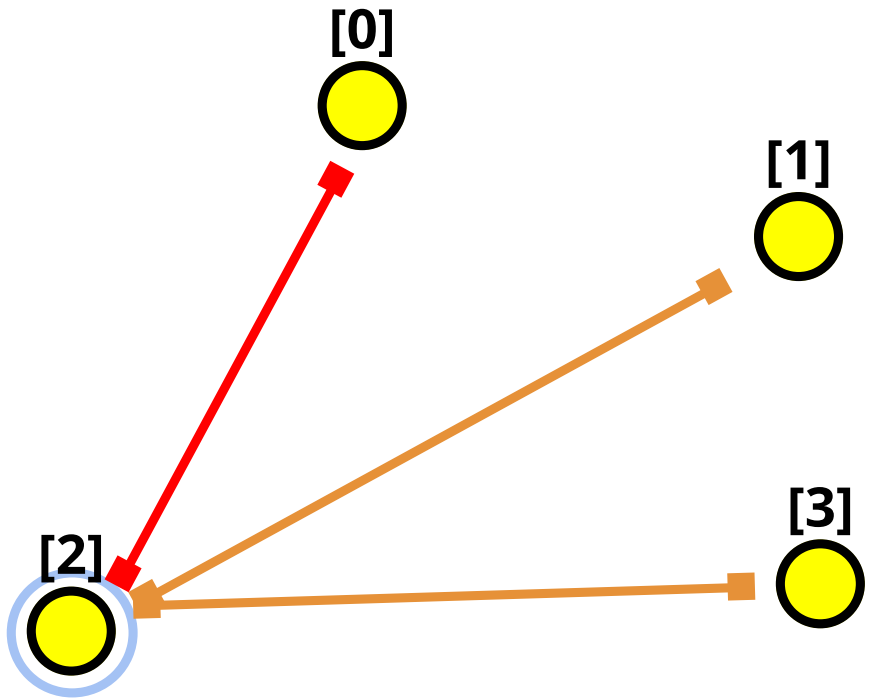
Find the **nearest city** from the **last city** in the **pool**



```
pool = [0,1,3]
path = [2]
```

2

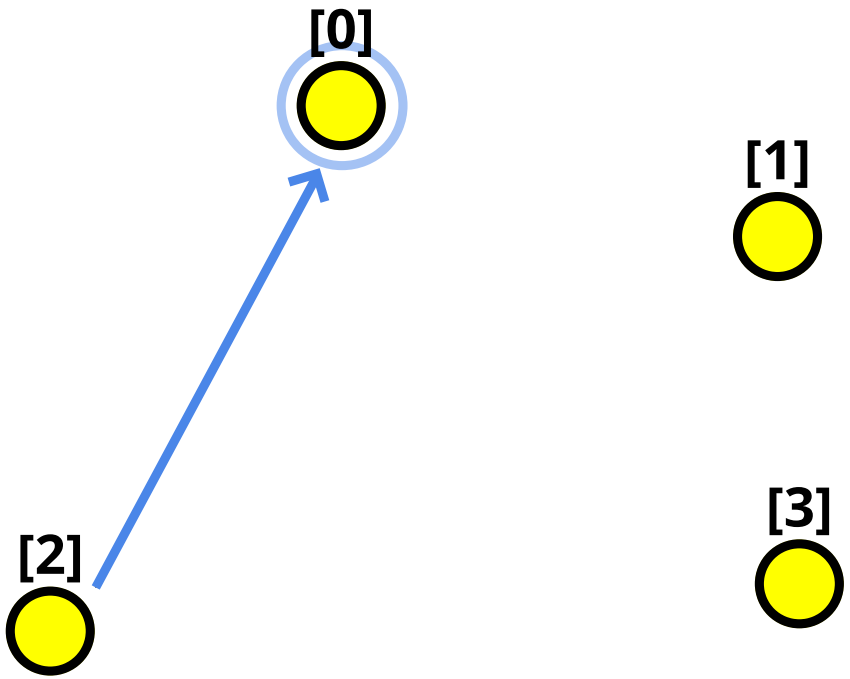
This is the **nearest one**



```
pool = [1,3]
path = [2,0]
```

2
0

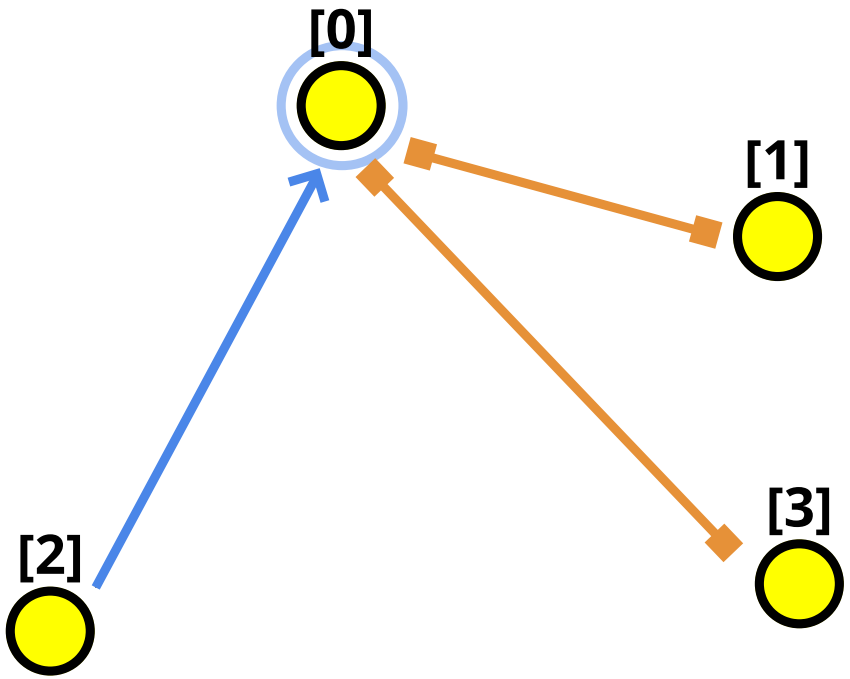
Pop the nearest city from the pool and **append** to the path



```
pool = [1,3]
path = [2,0]
```

2
0

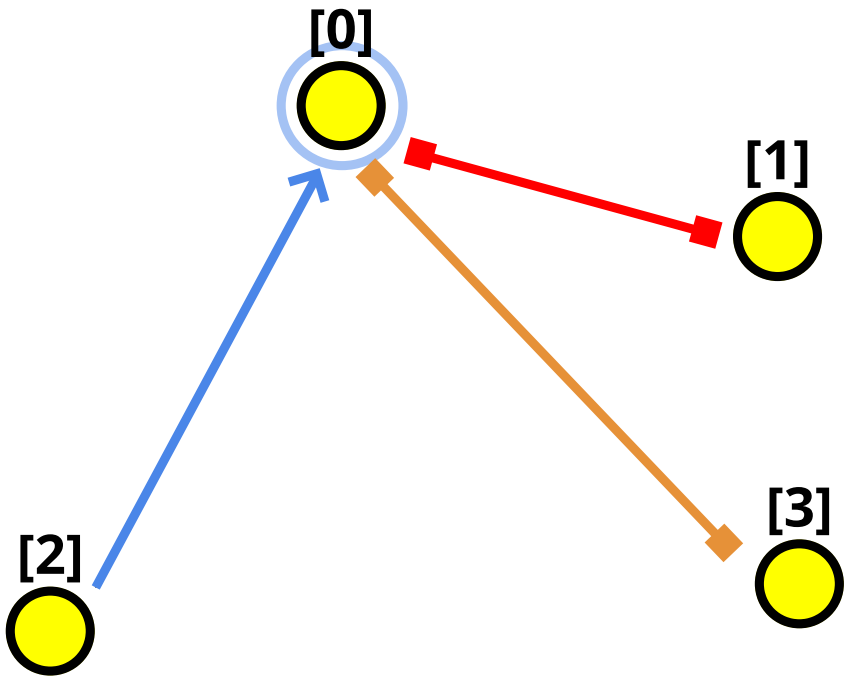
Find the **nearest city** from the **last city** in the **pool**



```
pool = [1,3]
path = [2,0]
```

2
0

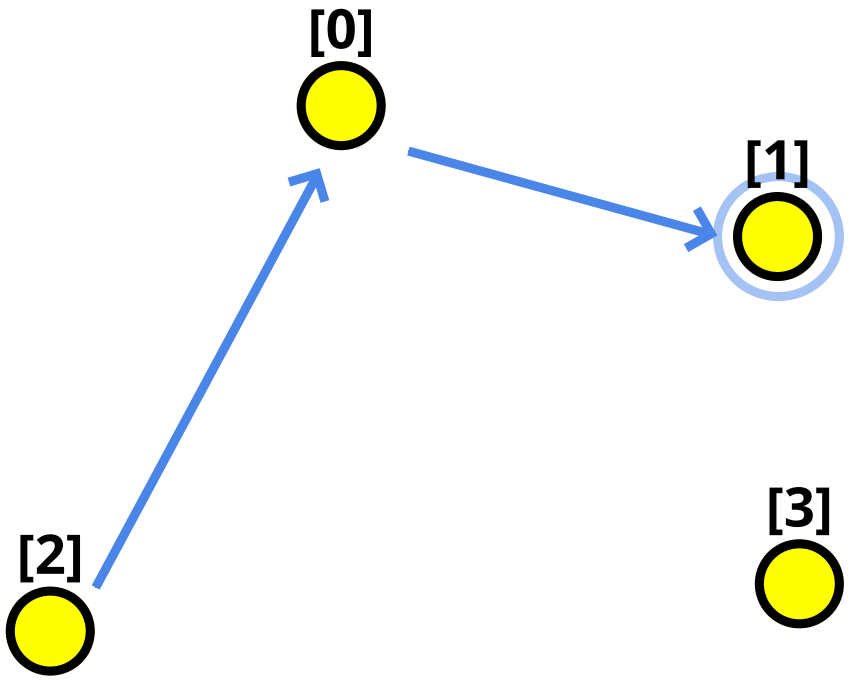
This is the **nearest one**



```
pool = [3]
path = [2,0,1]
```

2
0
1

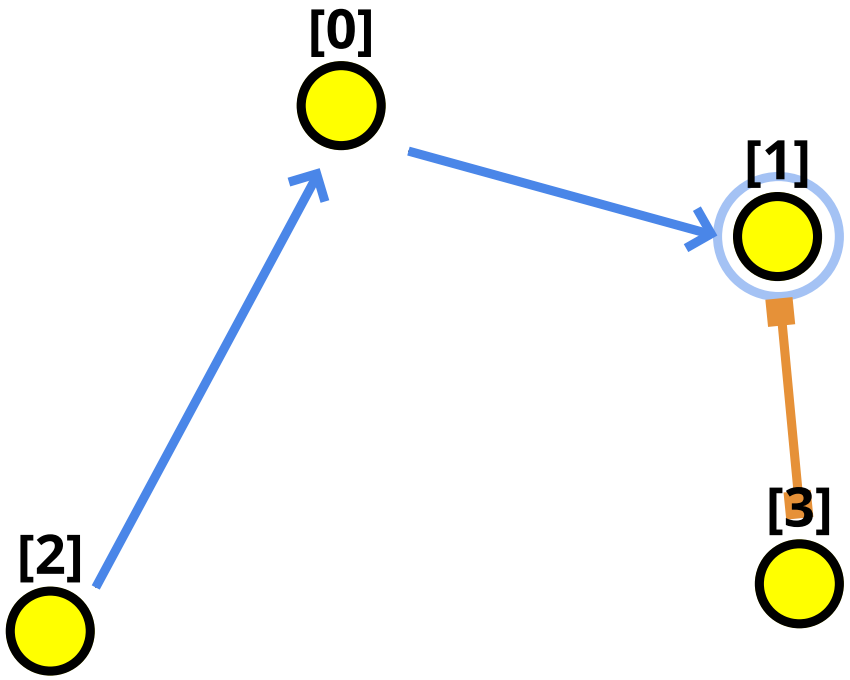
Pop the nearest city from the pool and **append** to the path



```
pool = [3]
path = [2,0,1]
```

2
0
1

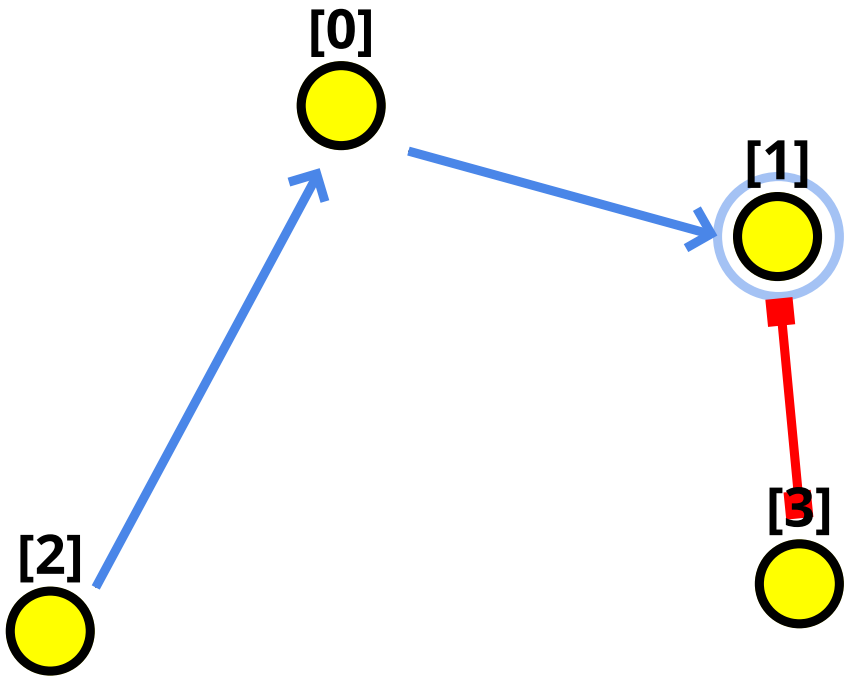
Find the **nearest city** from the **last city** in the **pool**



```
pool = [3]
path = [2,0,1]
```

2
0
1

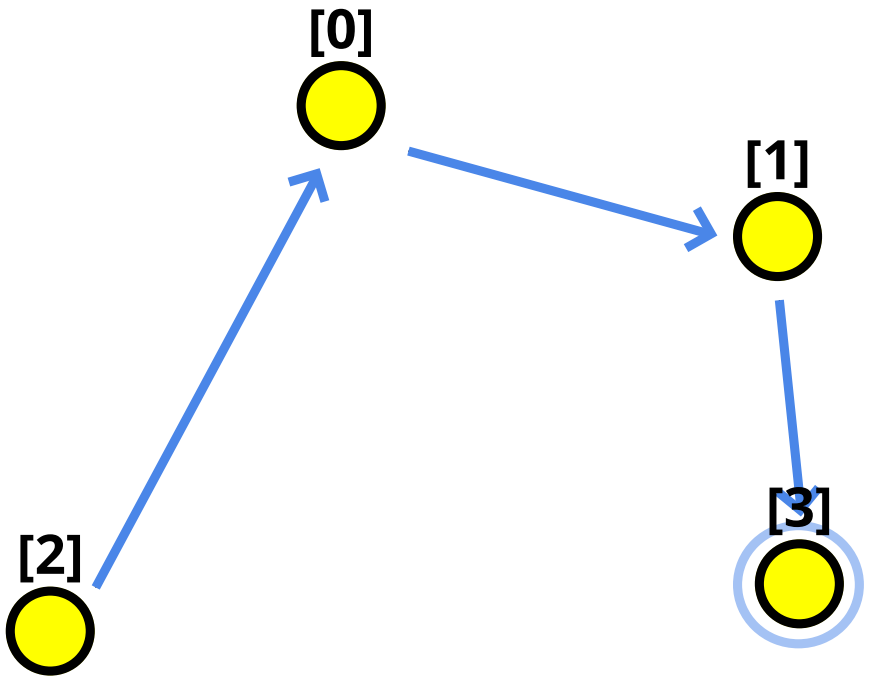
This is the **nearest one**



```
pool = []
path = [2,0,1,3]
```

2
0
1
3

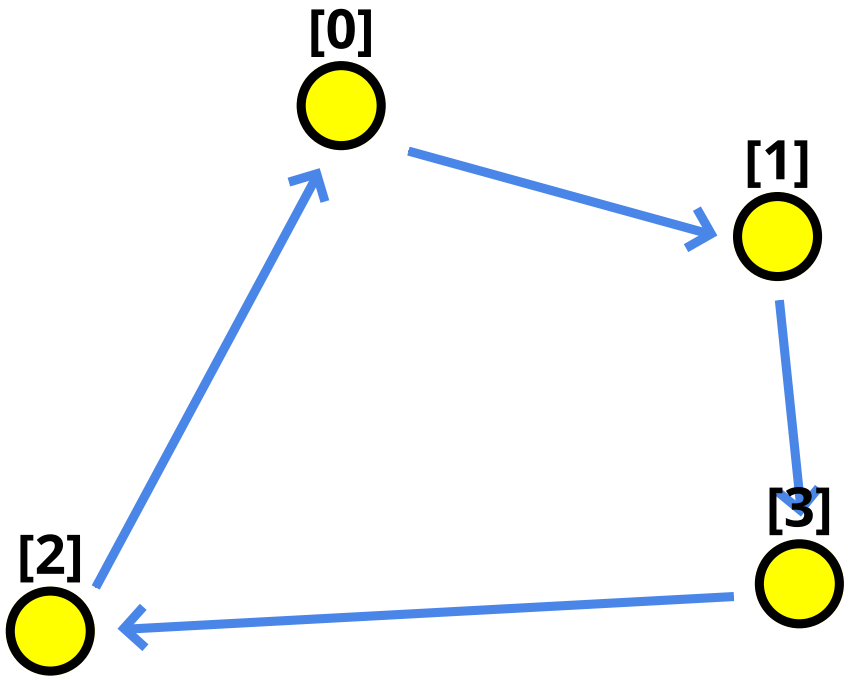
Pop the nearest city from the pool and **append** to the path



```
pool = []
path = [2,0,1,3,2]
```

2
0
1
3
2

Append the first city to the last to close the path



while문을 써서, pool이 0 될때까지 반복

```
pool = list(range(10))
path = []

print('pool:', pool)
print('path:', path)
```

```
pool: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
path: []
```

```
city = pool.pop(3) # the first city
path.append( city )

print('pool:', pool)
print('path:', path)
```

```
pool: [0, 1, 2, 4, 5, 6, 7, 8, 9]
path: [3]
```

```
city = pool.pop(5) # the second city
path.append( city )

print('pool:', pool)
print('path:', path)
```

```
pool: [0, 1, 2, 4, 5, 7, 8, 9]
path: [3, 6]
```

```
city = pool.pop(0)
path.append( city )

print('pool:', pool)
print('path:', path)
```

```
pool: [9]
path: [3, 6, 0, 1, 2, 4, 5, 7, 8]
```

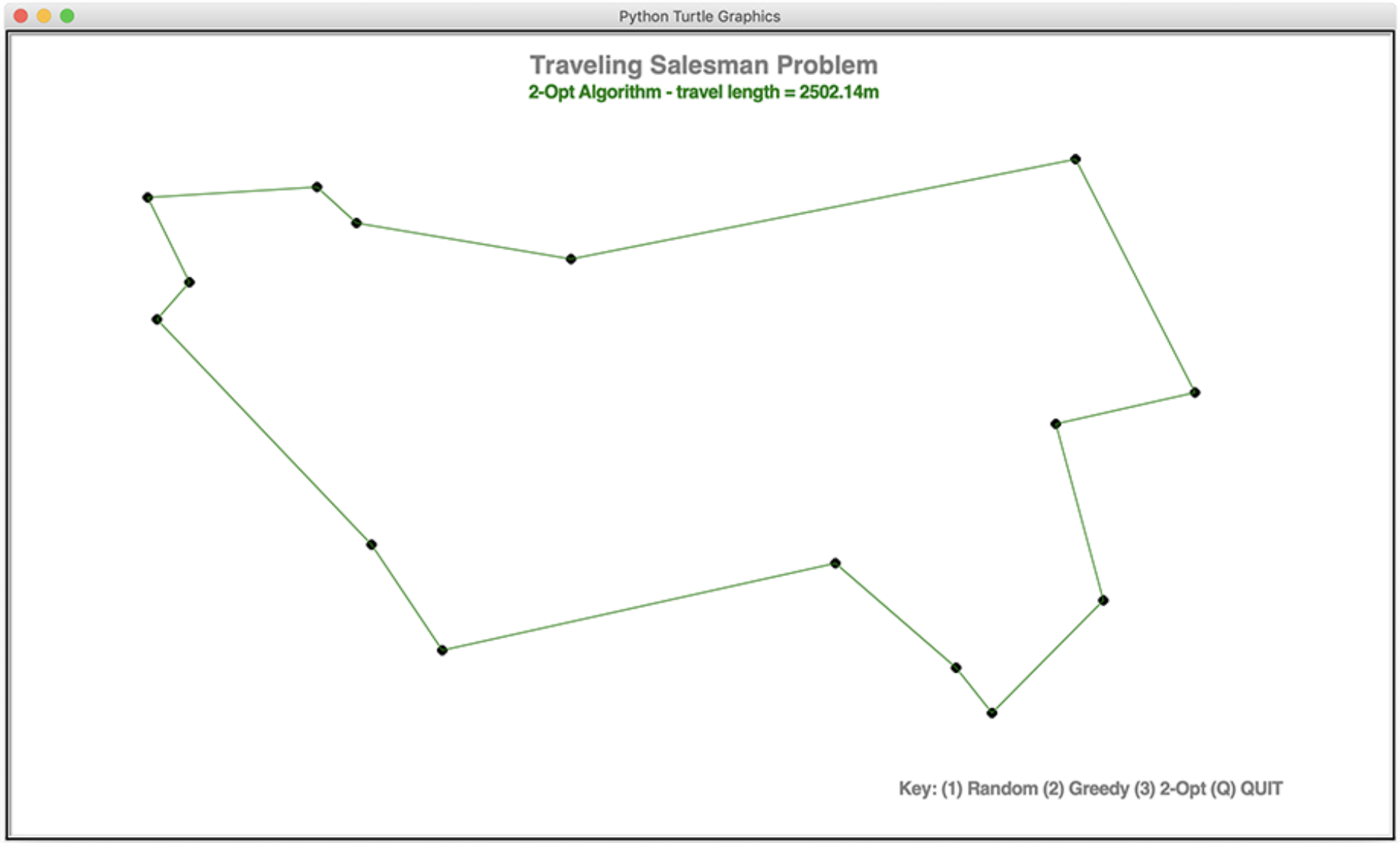
```
city = pool.pop(0) # the last city
path.append( city )

print('pool:', pool)
print('path:', path)
```

```
pool: []
path: [3, 6, 0, 1, 2, 4, 5, 7, 8, 9]
```

```
path.append( path[0] ) # return to the first city

print('path:', path)
```

```
path: [3, 6, 0, 1, 2, 4, 5, 7, 8, 9, 3]
```

2-Opt Algorithm


Python Turtle Graphics
Traveling Salesman Problem
2-Opt Algorithm - travel length = 2502.14m
Key: (1) Random (2) Greedy (3) 2-Opt (Q) QUIT

2-Opt Algorithm

1. Make a complete path (random / greedy)
2. Apply a '**swap**'
3. Keep the path if the changed version is shorter.
4. Repeat the process 2-3

두 지점을 선택 후 바꾸고 -> 평가 후 반영

It works because it removes crossings

2-Opt Swap

1. take **start** to **path[i-1]** and add them in order to new_route
2. take **path[i]** to **path[k-1]** and add them in **reverse** order to new_route
3. take **path[k]** to **end** and add them in order to new_route

example route: A ==> B ==> C ==> D ==> E ==> F ==> G ==> H ==> A

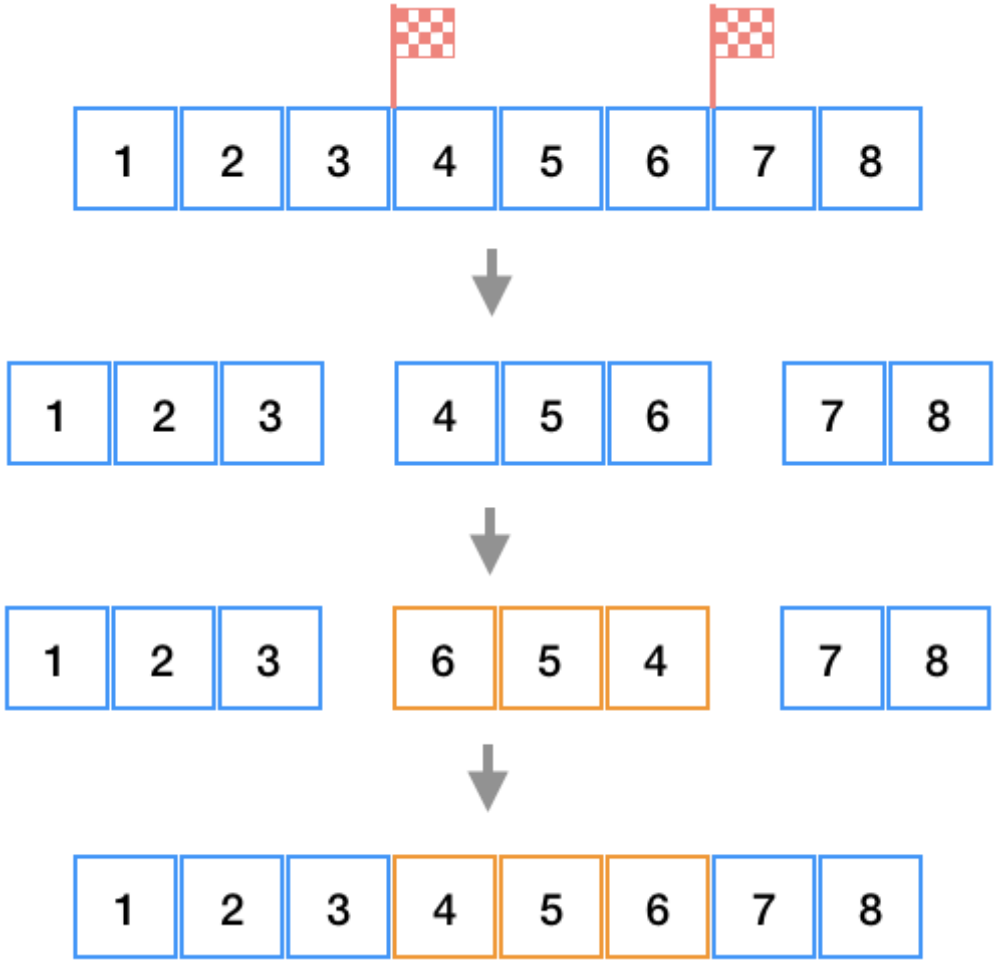
example i = 3, example k = 7

new_route:

1. **A ==> B ==> C** D,E,F,G였던 걸 swap해준 부분
2. A ==> B ==> C ==> **G ==> F ==> E ==> D**
3. A ==> B ==> C ==> G ==> F ==> E ==> D ==> **H ==> A**

2-Opt Swap

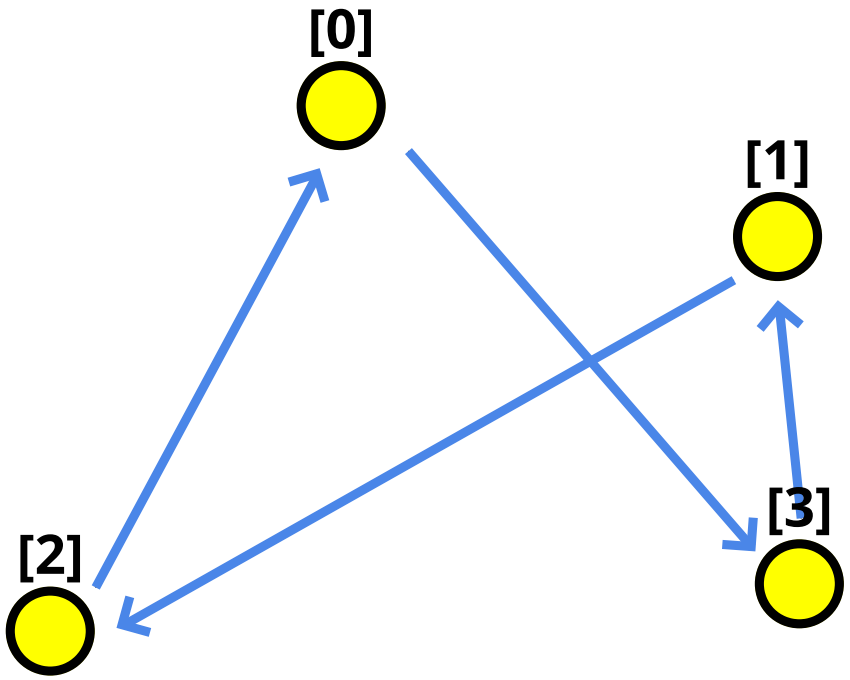
1. Slice a path to **path1**, **path2**, **path3**
2. Reverse the **path2**
3. Concaternate path1 + **path2** + path3



```
path = [2,0,3,1,2]
```

2
0
3
1
2

an Initial path

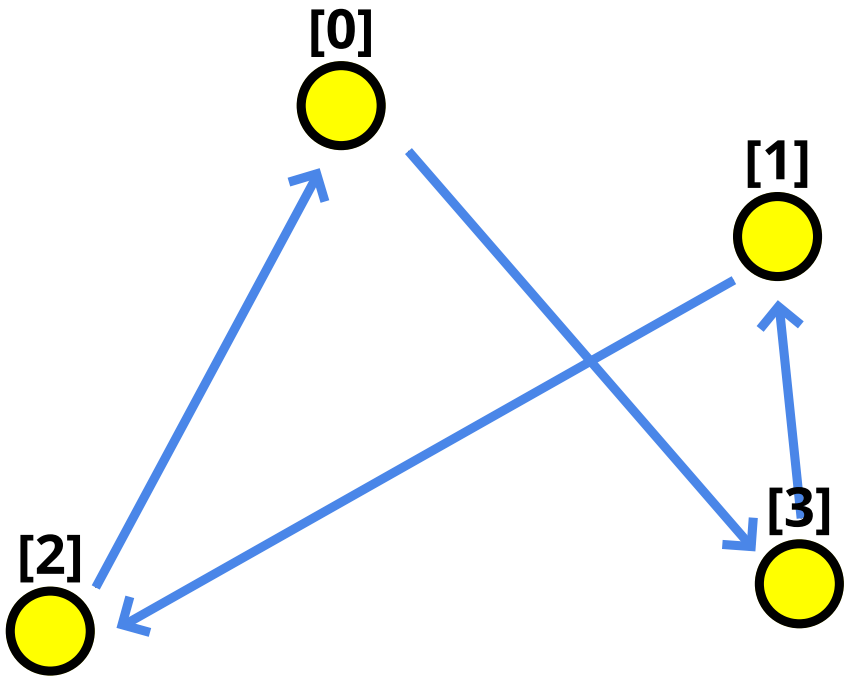


```
path1 = [2,0], path2 = [3,1], path3 = [2]
path = [2,0,3,1,2]
```

2
0
3
1
2

set **i**, **k** randomly (**i** = 2, **k** = 4)

slice the path into path1, **path2**, path3

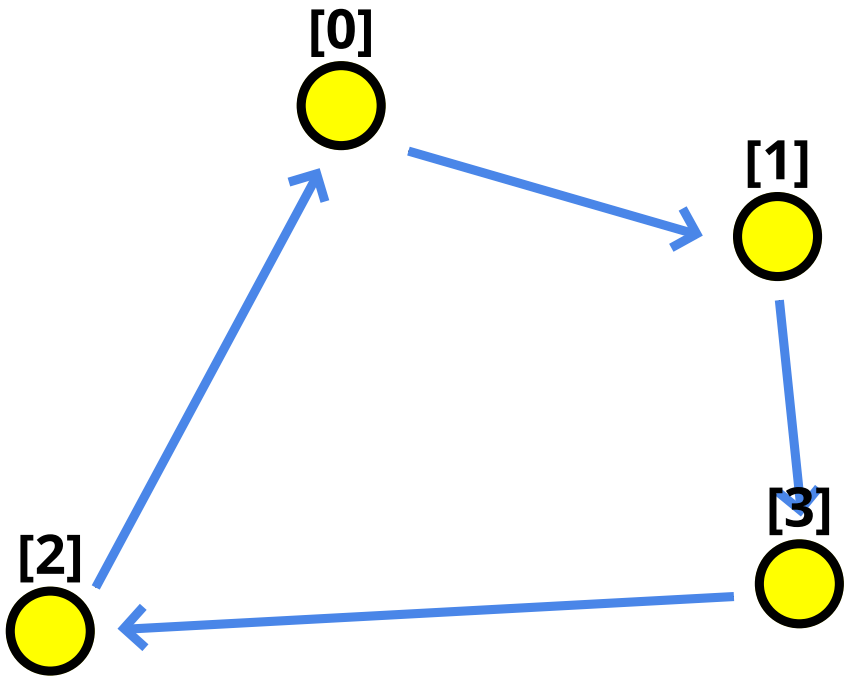


```
path1 = [2,0], path2 = [1,3], path3 = [2]
path = [2,0,1,3,2]
```

2
0
1
3
2

reverse **path2**

new_path = path1+**path2**+path3



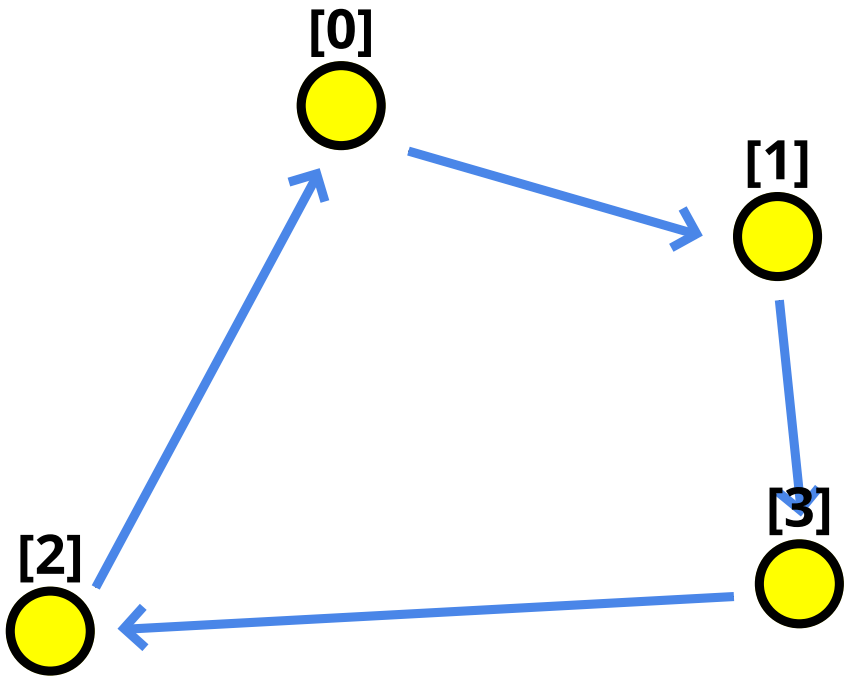
```
calc_path_length(path) > calc_path_length(path_new)
```

path = [2,0,1,3,2]

2
0
1
3
2

path = new_path

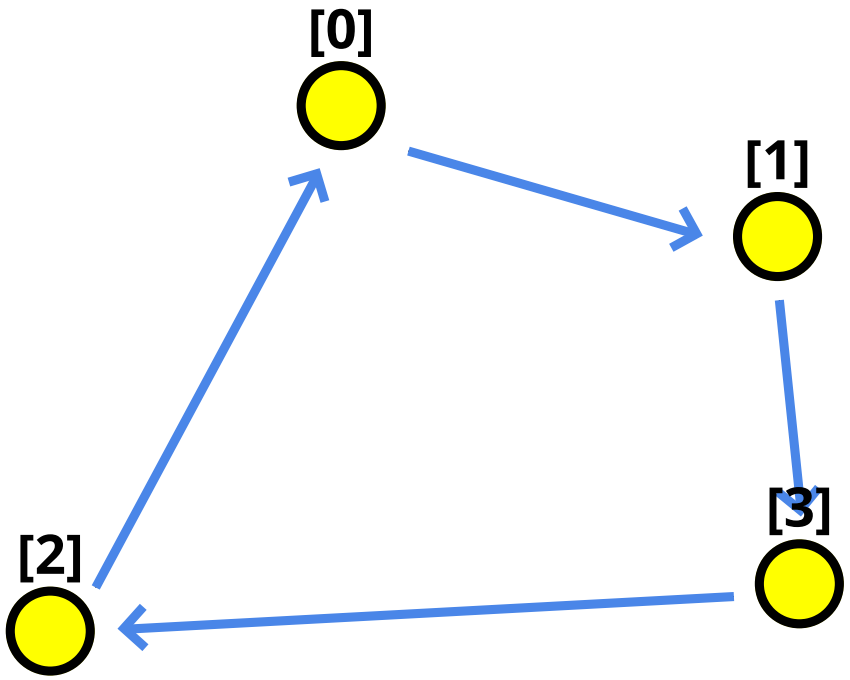
if the new_path is shorter than the path



```
path = [2,0,1,3,2]
```

2
0
1
3
2

Repeat N times



```
1 path = list(range(10))
2 path.append( path[0] )
3
4 print('path:', path)
```

```
path: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
```

```
1 i = random.randint(1, len(path)-2) # Edit this
2 k = random.randint(i, len(path)-2) # Edit this
3 print(i, k)
```

```
1 7
```

```
1 path1 = path[:i]
2 path2 = path[i:k]
3 path3 = path[k:]
4
5 print('path1:', path1)
6 print('path2:', path2)
7 print('path3:', path3)
```

```
path1: [0]
path2: [1, 2, 3, 4, 5, 6]
path3: [7, 8, 9, 0]
```

```
1 print('path2:', path2)
2
3 path2.reverse()
4
5 print('path2:', path2)
```

```
path2: [1, 2, 3, 4, 5, 6]
path2: [6, 5, 4, 3, 2, 1]
```

```
1 path = path1 + path2 + path3
2
3 print('path:', path)
```

```
path: [0, 6, 5, 4, 3, 2, 1, 7, 8, 9, 0]
```

Code

Random Algorithm

```
1       # Random Algorithm
2       def make_random_path(self):
3
4           # make a path containing all nodes
5           result = [] # EDIT THIS
6
7           # shuffle
8               # EDIT THIS
9
10          # return to the first city
11          if len(result) > 0:
12              result.append( result[0] )
13
14          # update with the new path
15          self.set_new_path( result, 'Random Algorithm' )
```

Greedy Algorithm

```
1     # Greedy Algorithm
2     def make_greedy_path(self):
3
4         # make a pool containing all nodes
5         pool = list(range(len(self.nodes)))
6
7         result = []
8
9         # pop the first node from the pool and append to the result
10            # EDIT THIS
11
12        # while there are any remaining node
13        while len( pool ) > 0:
14
15            # initialize min_dist and min_index
16            min_dist = None
17            min_index = None
18
19            # for every remaining nodes
20            for i in range(len(pool)):
21
22                # get the most recently added node
23                node1 = 0 # EDIT THIS
24                # the current remaining node
25                node2 = 0 # EDIT THIS
26                # calculate the distance between two ( using self.calc_dist() )
```

pool에 있는 도시가 남아 있으면 반복

2-Opt Algorithm

```
    # 2-Opt Algorithm
    def make_2opt_path(self):

        # make a path
        self.make_random_path()

        # loop 100 times
        for i in range(100):

            # modify the current path

            # select random two node indices
            i = 0 # EDIT THIS
            k = 0 # EDIT THIS

            # slice the path with three segments (path1, path2, path3)
            path1 = []  # EDIT THIS
            path2 = []  # EDIT THIS
            path3 = []  # EDIT THIS

```

Submit

- **Deadline: 9 December (Mon) 23:55 pm**
- Modify the given code and submit as **tsp_201812345.py**
- Put comments in your codes

- Visual competition(optional): 비주얼적인 측면도 업그레이드 할 수 있는 거(옵셔널)
  - submit a screen recording (e.g. **tsp_201812345.mp4**)
    - Screen recording tools
      - Windows: Bandicam (반디캠) 핵심적인 비주얼 컨셉만 10-15sec 가량 레코딩 및 제출
      - Mac OS: shift-cmd-5

TSP Competition

- **10 December (Tue) class time**
- **Performance Competition**
  - 3 rounds
  - round time: **60 sec**
- **Visual Competition**
  - Audience evaluation

Q&A