Cycle Similar to the Image's Outline

CPS 19-079 이지원

Problem

Input : pixel data / Output : point path

원래의 outline image의 픽셀들을 연결하여 가장 비슷한 Cycle을 만드는 Algorithm 제작

"가장 비슷하다" 란?

모든 점을 지나는 Cycle중에서 Edge 길이의 합이 최소

1. 중복되는 edge의 길이를 모두 더함
2. 중복되는 edge의 길이는 한번만 더함, 길이가 같다면 중복되는 edge 개수를 최소

Problem

$$V = \{p_i | 1 \leq i \leq n\} \qquad P = (p_{\sigma(1)}, p_{\sigma(2)}, \cdots, p_{\sigma(m)})$$

1. edge의 길이를 모두 더함

$$W = \sum_{1 \leq i < n} w((p_{\sigma(i)}, p_{\sigma(i+1)}))$$

→ W를 최소화 하는 P를 찾는다

2. 중복되는 edge의 길이는 한번만 더함.
   길이가 같다면 중복되는 횟수를 최소

$$S_P = \{(p_{\sigma(k)}, p_{\sigma(k+1)}) | 1 \leq k < m\}$$

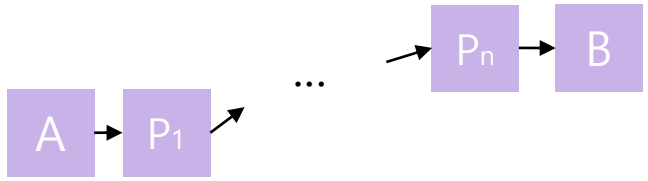
$$W = \sum_{e \in S_p} w(e)$$

→ W를 최소로 하는 P를 찾는다
→ W가 같다면 $m - |S_p|$를 최소로 한다

Definition

1-1. 중복되는 edge의 길이를 모두 더하는 경우

Path(A,B) = A에서 B까지 픽셀을 따라갔을 때의 거리



$$path(A,B) = \overline{AP_1} + \overline{P_1P_2} + \cdots + \overline{P_nB}$$

Properties

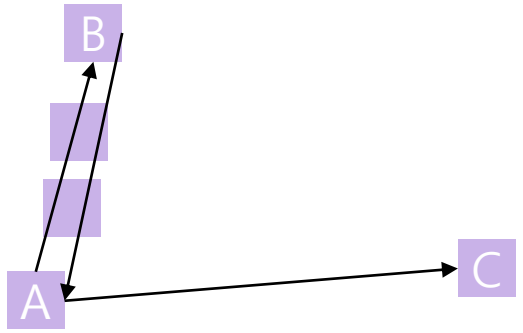
$$\overline{AB} \leqq path(A,B)$$

$$\overline{AB} + \overline{BC} \geqq \overline{AC}$$

∵ 최단거리 / 삼각부등식

과정

1-1. 중복되는 edge의 길이를 모두 더하는 경우



Let. A-B의 path가 중복

$$d = 2 \times path(A, B) + \overline{AC}$$

$$d > path(A, B) + \overline{AB} + \overline{AC} > path(A, B) + \overline{BC}$$

A→B→A→C 보다 A→B→C 으로 갈 때 거리가 더 작음

Edge가 중복되지 않을 경우가 optimal하다.

TSP(traveling salesman problem, NP-hard)와 같은 문제이다

과정

1-2. 중복되는 edge의 길이를 한번만 더하는 경우

모든 node를 지나므로 path는 spanning tree를 포함한다

Path distance ≥ spanning tree의 edge weight의 합

≥ minimum spanning tree(MST) edge weight 합

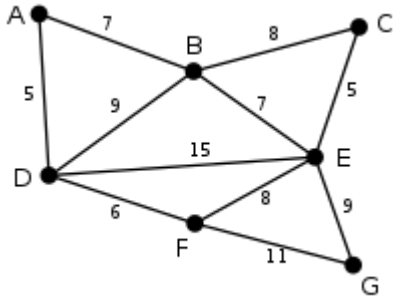
→ Spanning tree는 모든 node들에 연결이 되어있으므로,
MST edge만들 따라서 path를 만들 수 있다.

S_p가 MST가 되는 경우 optimal한 해이다.

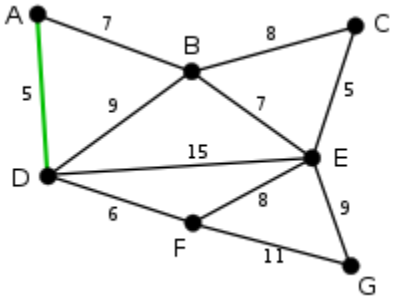
알고리즘

MST를 구하는 방법 – Kruskal Algorithm

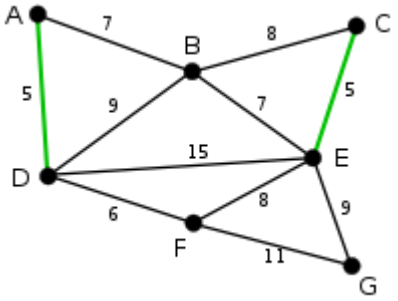
1



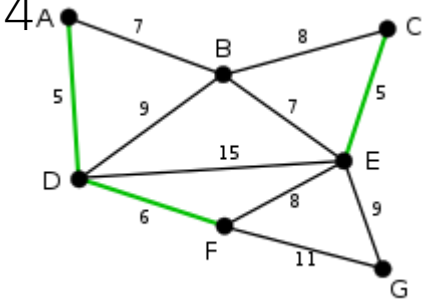
2



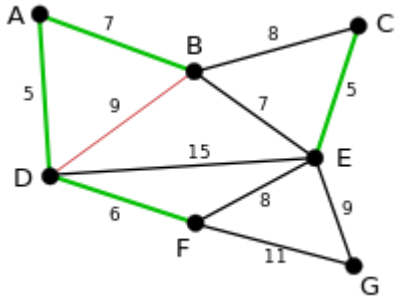
3



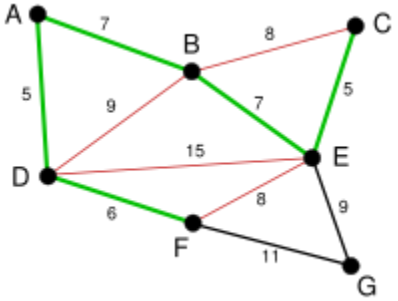
4



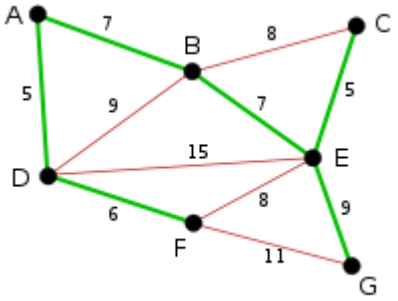
5



6



7



알고리즘

MST를 구하는 방법 – Kruskal Algorithm

```
Kruskal(N, E, cost) :
    sort edges in E by increasing cost
    while |T| < |N| − 1:
        let (u, v) be the next edge in E
        if u and v are on different components:
            join the components of u and v
            T = T ∪ {(u, v)}
    return T
```

Time Complexity

$E = n^4, V = n^2$

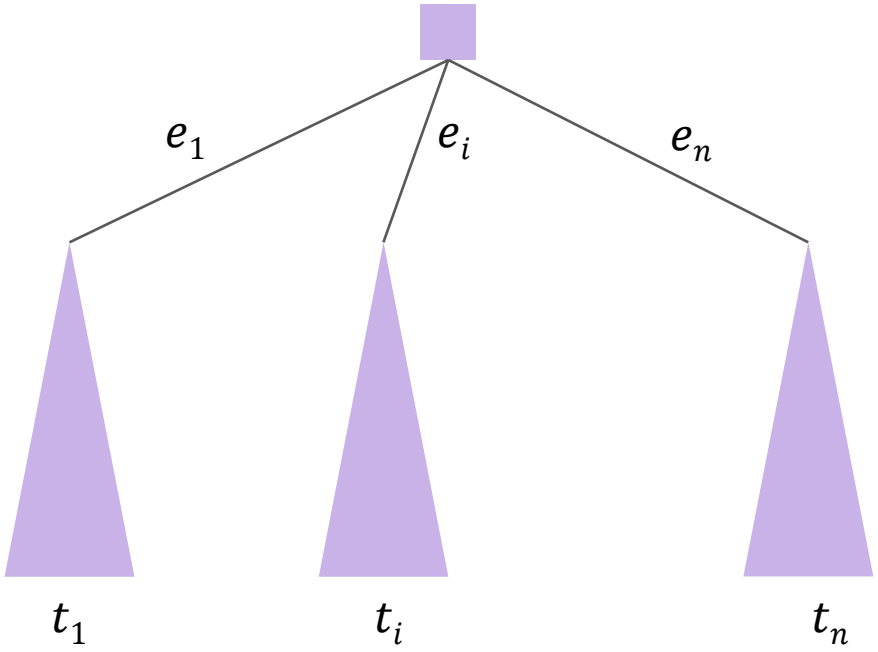
$O(ElogE) = O(ElogV) = O(n^4logn)$

알고리즘

MST에서 어떤 path가 만들어 지는가

MST는 임의의 start node를 root로 가지는 tree로 볼 수 있다.

Sub tree로 가는 길은 e_i edge밖에 없으므로 해당 edge를 통해서 갔다가 와야 한다.



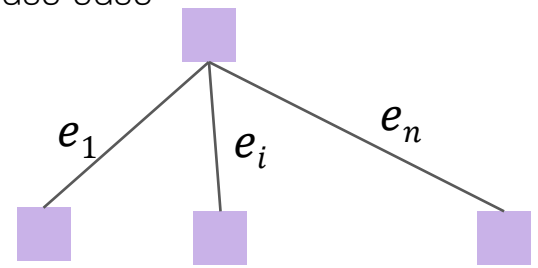
$$s \rightarrow e_1 \rightarrow t_1 \rightarrow e_1 \rightarrow \ldots \rightarrow e_m \rightarrow t_m \rightarrow e_m \rightarrow s$$

$$d(T) = 2\sum_{1}^{m} e_i + d(t_i)$$

알고리즘

MST에서 어떤 path가 만들어 지는가

Base case



Mathematical induction

Assume. $d(t) = 2|Et|$ (t = sub tree)

Base case – sub tree가 없는 경우

$$d(t) = \sum_{1}^{n} 2e_i = 2|E|$$

Step

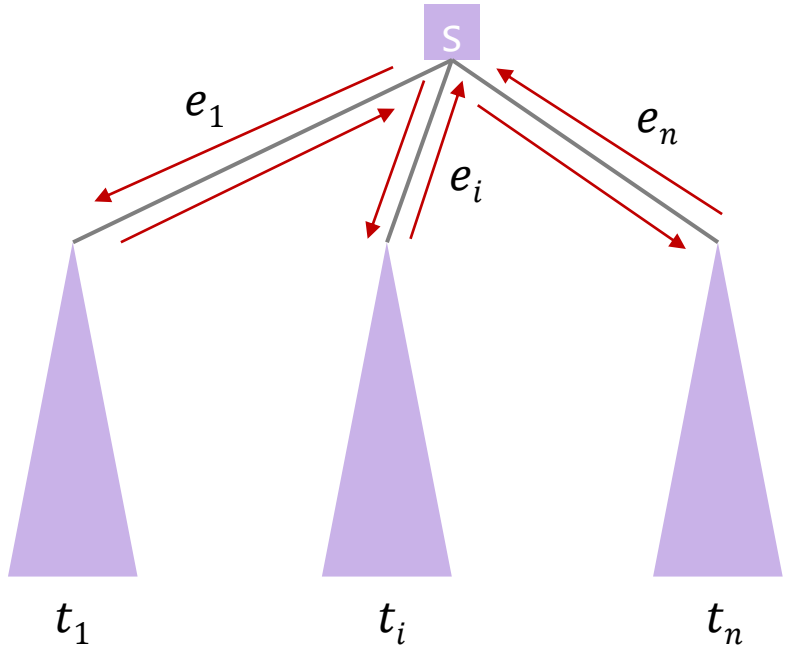
$$d(T) = 2\sum_{i=1}^{m} e_i + d(ti) = 2\sum_{i=1}^{m} e_i + 2|E_{ti}|$$

$$= 2|E_T| \quad (\because |E_T| = \textstyle\sum_{i=1}^{m} e_i + |E_{ti}|\,)$$

알고리즘

MST에서 어떤 path가 만들어 지는가 – 탐색 경로 구현

$$s \rightarrow e_1 \rightarrow t_1 \rightarrow e_1 \rightarrow \ldots \rightarrow e_m \rightarrow t_m \rightarrow e_m \rightarrow s$$



임의의 node에서 DFS 탐색을 하는 경로

```
def DFSvisit(node):
    color[node] = 1
    for adj in graph[node]:
        if color[adj] == 0:
            path.append(adj)
            pred[adj] = node
            DFSvisit(adj)
            path.append(pred[adj])
    color[node] = 2
```

새로운 node를 탐색할 때 append &
탐색이 끝날 때 parent node를 append

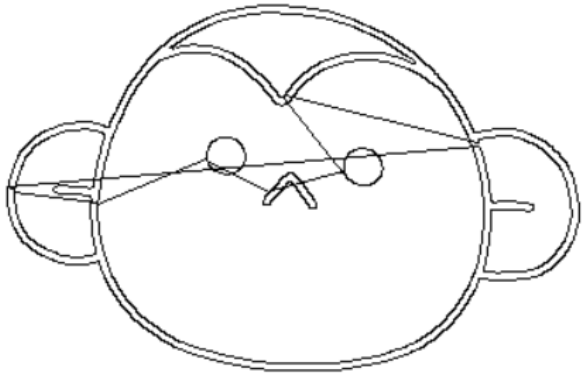
구현

구현 및 결과

Input : N=2934 / cluster = 7개

가까운 node를 따라갈 때

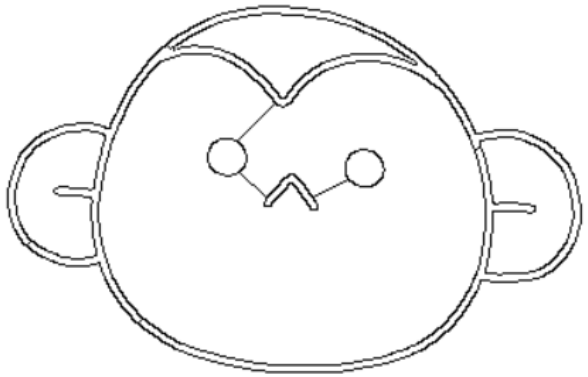
시간 : 0.22sec
Total distance : 3692.85



MST를 구할 때

Cluster 및 탐색시간 : 0.32sec
MST 시간 : 558.94sec

MST distance : 3130.04
Total distance : 6260.08



알고리즘

Time Complexity를 줄이는 방법

현재 Time Complexity = O(n^4logn)

n^4개의 길이를 정렬 + 경로 만들기

N = 2934 → MST 생성 : 558sec
N = 4000 → MST 생성 : 15min 이상

Kruskal을 실행하는 과정에서 모든 edge들을 체크하니 마지막에서 시간이 매우 많이 걸렸음
→ Kruskal을 돌리기 전에 이 수를 줄일 수는 없을까?

Outline은 여러 line의 합으로 이루어져 있다

Line들을 cluster로 만들어서 cluster끼리 연결하는 방법을 찾는다
→ cluster간의 MST를 보면 time도 줄어들 것

알고리즘

Time Complexity를 줄이는 방법

line을 따라가기 위해서는 가까운 pixel을 골라야 한다.

→ 해당 픽셀에서 픽셀까지의 거리를 모두 구해서 가장 가까운 픽셀을 그 다음점으로 지정

400*400 image 1~2min O(n^4)

가장 가까운 점들을 찾기 위해서 해당 점에서 탐색을 한다.

400*400 image / N=4200 → 1~2sec

일정 거리 이상이 되면 다른 cluster 라고 판단

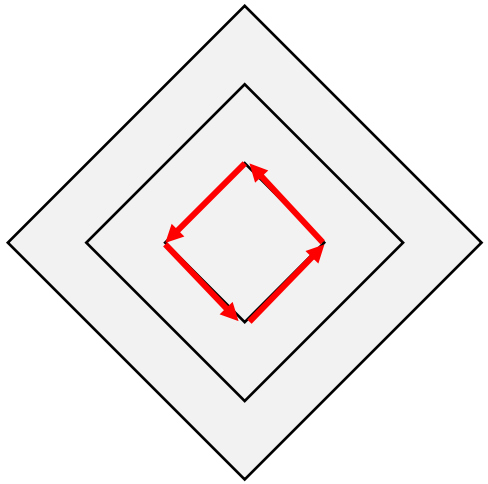
알고리즘

Time Complexity를 줄이는 방법 – 탐색하는 방법

탐색하는 방법

마름모의 형태로 탐색

```
def diamond(n):
    direction = []
    for i in range(n):
        direction += [(n-i, i), (-i, n-i), (-n+i, -i), (i, -n+i)]
    return direction
```



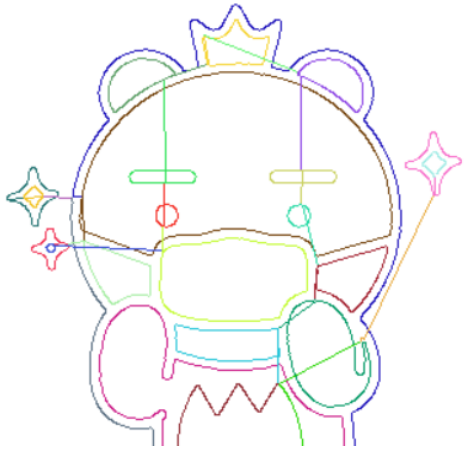
원의 형태로 탐색

```
def circle(n):
    direction = []
    for i in range(n+1):
        if i==n:    a = 0
        else :
            a = ((n-0.5)**2-i**2)**0.5
        b = ((n+0.5)**2-i**2)**0.5
        for j in range(int(a)+1, int(b)+1):
            direction += [(i,j), (j,-i), (-i,-j), (-j,i)]
    return direction
```

알고리즘

Time Complexity를 줄이는 방법 – 탐색하는 방법

마름모의 형태로 탐색



cluster 수 : 27
시간 : 1.937
total distance : 5686.127

원의 형태로 탐색

cluster 수 : 24
시간 : 2.187
total distance : 5558.136

가까운 거리에 있어도 마름모는 더 먼 거리에 있는 것으로 생각
→ 그 거리가 5보다 커지면 cluster의 수가 커짐

알고리즘

Time Complexity를 줄이는 방법 – Cluster 구분 길이

N = 3
cluster 수 : 25
걸리는 시간 : 0.596

N = 5
cluster 수 : 24
걸리는 시간 : 0.607

N = 10
cluster 수 : 17
걸리는 시간 : 0.573

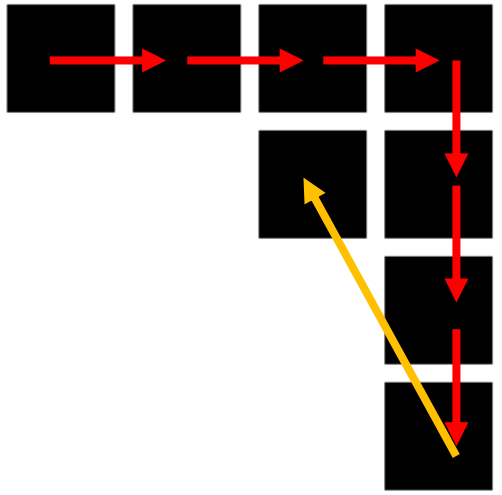
같은 색 = 같은 cluster

알고리즘

Time Complexity를 줄이는 방법 – 탐색하는 방법

Cluster를 만드는 과정

가까운 점들을 선택하는 식으로 하다가 지나친 점들이 있을 수 있다.
→ Cluster의 개수가 늘어남
→ 실제 그림과 달라지고 time도 늘어남



"주변에 선택된 pixel 이 얼마나 있는가" 를 거리와 함께

조건으로 넣어준다

알고리즘

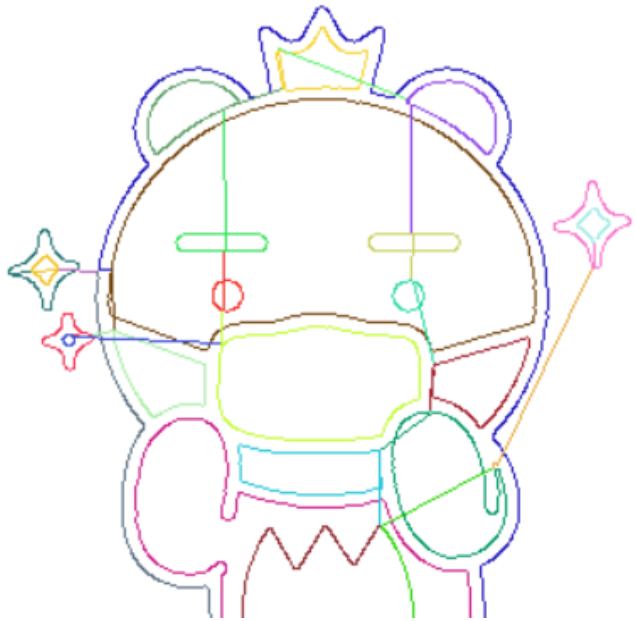
Time Complexity를 줄이는 방법 – 탐색하는 방법

주변의 점을 확인하지 않은 경우

Cluster 개수 : 49
total distance : 6184.54

주변의 점을 확인하는 경우

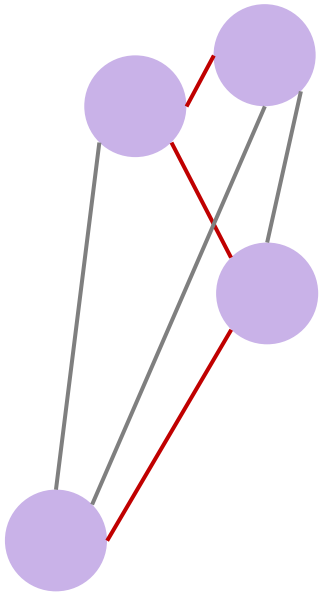
cluster 수 : 27
total distance : 5686.12



알고리즘

Time Complexity를 줄이는 방법 – Cluster간 MST

$$C = \{c_i | 1 \leq i \leq k\}$$



각각의 cluster 내부 path는 그대로 MST_result에 넣어준다.

Cluster 간의 mst를 구하여서 MST_result에 넣어준다

→ Cluster 간의 거리를 정의하고 구해야 한다

Cluster 사이의 거리
= 두 cluster의 점들 중에 가장 가까운 거리

```
def clusterDis(c1, c2):
    min = math.inf
    for i in range(len(c1)-1):
        for j in range(len(c2)-1):
            x, y = c1[i], c2[j]
            d = x.getDistance(y)
            if d <= min:
                min = d
                pair = (x,y)
    return min, pair
```

Cluster간 거리 계산 구현

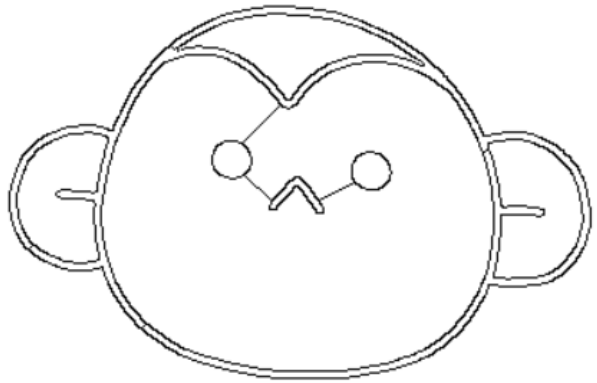
알고리즘

Time Complexity를 줄이는 방법 – Cluster간 MST

N : 2934 / Cluster 수 : 7

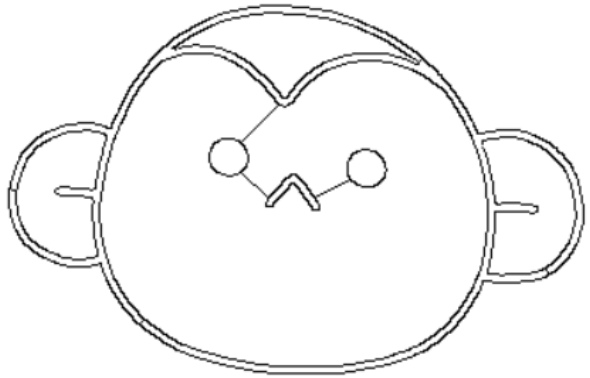
전체 점에 대한 MST

MST 시간 : 558.94sec
total distance : 6260.08



Cluster간 MST

MST 시간 : 3.67sec
total distance : 6276.09

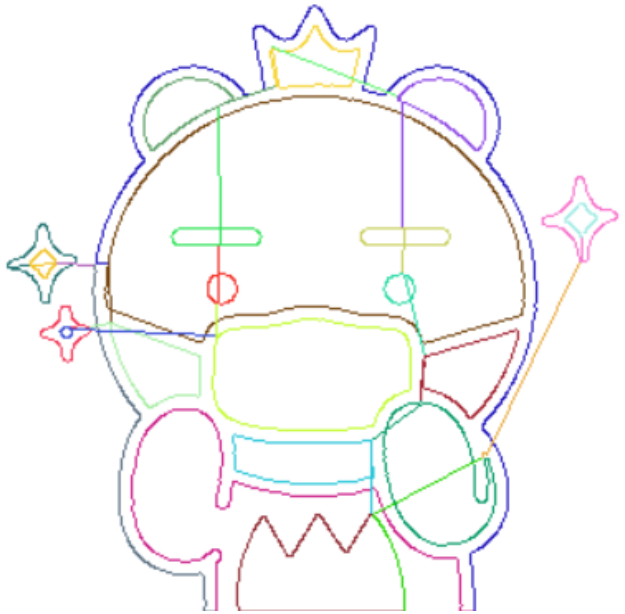


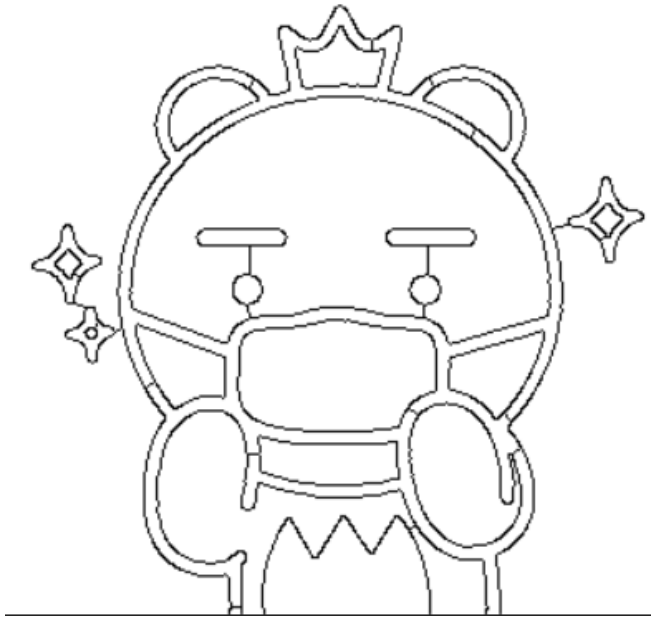
알고리즘

Time Complexity를 줄이는 방법 – Cluster간 MST

Cluster 수 : 24개
시간 : 15분 이상

cluster 걸리는 시간 : 0.569
MST 시간 : 8.582sec
total distance : 9343.437





알고리즘

Time Complexity를 줄이는 방법 – Cluster간 distance

Cluster들은 가까운 점들끼리 모여서 어느정도 continuous 하게 존재를 하고 있다.

각 cluster의 10n번째 점들끼리 길이 비교 비교한 후, 최소거리의 근처 점들끼리 다시 비교

```
def clusterDis2(c1, c2):
    min = math.inf
    n, m = 10, 10
    for i in range((len(c1)-1)//n):
        for j in range((len(c2)-1)//m):
            x, y = c1[i*m], c2[j*m]
            d = x.getDistance(y)
            if d <= min:
                min = d
                pair = (x,y)
    return min, pair
```

```
for i in range((pairIdx[0]-1)*n, (pairIdx[0]+1)*m):
    for j in range((pairIdx[1]-1)*n, (pairIdx[1]+1)*m):
        if 0 <= i < len(c1)-1 and 0 <= j < len(c2)-1:
            x, y = c1[i], c2[j]
            d = x.getDistance(y)
            if d <= min:
                min = d
                pair = (x,y)

return min, pair
```

알고리즘

Time Complexity를 줄이는 방법 – Cluster간 distance

N=4290 / cluster 수 : 24

10n 번째 점들끼리 비교

MST 시간 : 3.345sec
Total distance : 9400.89

비교 후 근처 점들끼리 비교

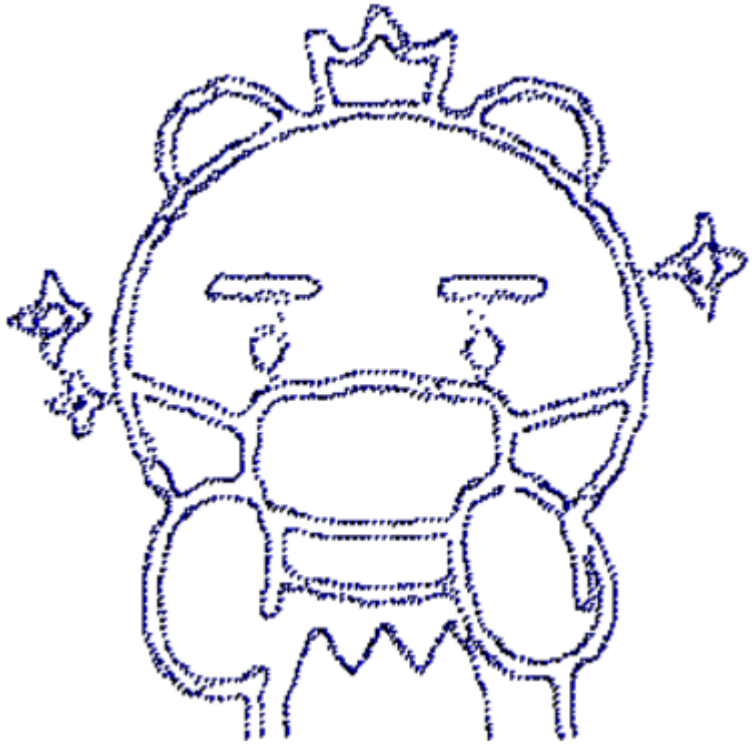
MST 시간 : 4.307sec
total distance : 9351.03

전체 점들 비교

MST 시간 : 8.582sec
total distance : 9343.43

Result

과거의 FFT 결과



현재의 FFT 결과

감사합니다