Лабораторн 12

П.Ундрал B241910036

```
pip install dynetx
```

```
Collecting dynetx
  Downloading dynetx-0.3.2-py3-none-any.whl.metadata (2.9 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: future in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: decorator in /usr/local/lib/python3.11/dist-
Downloading dynetx-0.3.2-py3-none-any.whl (39 kB)
Installing collected packages: dynetx
Successfully installed dynetx-0.3.2
```

Snapshot Graphs

"Snapshot Graphs" гэдэг нь тодорхой хугацааны эсвэл үйл явцын аль нэг хэсгийн дэлгэрэнгүй үзүүлэлтүүдийг харуулах графикууд юм. Энэ нь өгөгдөлд хурдан, бодит цагийн буюу тодорхой хугацааны өөрчлөлтүүдийг дүрслэх зорилготой ашиглагддаг.

Жишээ нь, компьютерийн сүлжээний гүйцэтгэлийг хянахдаа “Snapshot Graph” ашиглаж болно. Энэ график нь сүлжээний төлөв байдал, урсгалын хурд, ачаалал зэрэгт хурдан хяналт тавих боломжийг олгодог.

Эдгээр графикууд нь тодорхой хугацаанд өгөгдлийн өөрчлөлтүүдийг хамгийн хурдан, ойлгомжтой байдлаар дүрслэхэд туслах бөгөөд, удирдлагын зорилгоор ашиглагддаг

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

from google.colab import drive: Энэ мөр нь Google Colab-ийн drive санг импортлодог. Энэ сан нь Google Drive-тай холбох, файлуудыг унших болон хадгалахад ашиглагддаг.

drive.mount('/content/drive'): Энэ код нь Google Drive-г Colab орчинд холбож, таны Drive-д хадгалагдсан файлуудыг ашиглах боломжийг олгодог.

'/content/drive' бол Colab орчинд таны Google Drive-ийн хуулбар байрлах зам юм.

Кодыг гүйцэтгэх үед, Google Drive руу хандах эрхийг олгох хүсэлт гарч, Google акаунтаар нэвтрэх шаардлагатай.

Үр дүн:

Кодыг гүйцэтгэснээр, таны Google Drive нь /content/drive замд холбогдох бөгөөд та Drive доторх файлуудыг Colab орчноос хялбархан ашиглаж, хадгалж болно.

```
import dynetx as dn  # dynetx номын санг импортлон динамик графтай ажиллана
import networkx as nx  # networkx номын санг импортлон стандарт графуудтай ажил
import random  # Санамсаргүй тоо үүсгэхийн тулд random номын санг импортлоно

# Файл унших функц
def read_net(filename):
    g = nx.Graph()  # Шинэ граф объект үүсгэнэ (өндөр холбогдолтой цэгүүдийг ха

    with open(filename) as f:  # Тухайн файлыг нээнэ
        f.readline()  # Эхний мөрийг хасах (жагсаалт, хүснэгтийн хэсэг байж маг

        for l in f:  # Дараагийн мөрүүдийг уншина
            l = l.strip().split(",")  # Мөрийг тасдаж, зайг арилгана
            g.add_edge(l[0], l[1])  # Мөрийн хоёр утгаас ирмэг үүсгэнэ (цэгүүди

    return g  # Граф объектыг буцаана

# Хоосон динамик граф үүсгэх
g = dn.DynGraph()   # Динамик граф объект үүсгэнэ (цаг хугацааны явцад өөрчлөгд
```

Динамик сүлжээг үүсгэх

Энэхүү кодын зорилго нь динамик графыг бүтээх бөгөөд энэ нь хугацааны явцад өөрчлөгддөг граф юм. Дэлгэрэнгүй тайлбар:

for t in range(1, 9):

Энэ нь 1-ээс 8 хүртэлх бүх тоонуудыг давтах (т.е. 1, 2, 3, … 8). Энэ тоо нь графын цаг хугацааны үе (time step)-ийг илэрхийлдэг. Тиймээс энэ цикл нь 8 удаа давтагдана.

er = read_net(f'/content/sample_data/got-s{6}-edges.csv'): Энэ мөр нь read_net функцээр тодорхой CSV файлыг уншиж байна. Файлын нэр нь got-s6-edges.csv гэж байгаа бөгөөд энд 6-г орлуулах боломжтой. Файл нь графын ирмэгүүдийн (edges) мэдээллийг агуулдаг.

g.add_interactions_from(er.edges, t=t):

Энэ мөр нь граф объект g-д ирмэгүүдийг (edges) нэмдэг. er.edges нь уншигдсан CSV файлын ирмэгүүдийн мэдээллийг агуулдаг. t=t гэсэн параметр нь тухайн ирмэгийг ямар цагийн үед (time step) нэмэхийг зааж байгаа бөгөөд энэ нь графын динамик шинж чанарыг илэрхийлнэ. Тухайн үед өрнөсөн харилцан үйлдлийг тус графад нэмнэ.

Үр дүн:

Энэ код нь 1-ээс 8 хүртэлх хугацааны үеийн (time step) аль бүртгэлийг динамик графад нэмэх үйл явц юм. Өгөгдсөн CSV файлын ирмэгүүдийг тухайн үеийн графад динамик байдлаар оруулна.

```
for t in range(1, 9):  # 1-ээс 8 хүртэл давтах
    er = read_net(f'/content/sample_data/got-s{6}-edges.csv')  # Файлыг унших
    g.add_interactions_from(er.edges, t=t)  # Динамик графд ирмэгүүдийг нэмэх
```

g.temporal_snapshots_ids():

g нь динамик граф объект юм.

temporal_snapshots_ids() функц нь тухайн графын бүх хугацааны үе буюу time steps-ийн ID-уудыг буцаадаг.

Хугацааны үеийн ID нь графын тухайн хугацаанд үзүүлж буй ирмэгүүд (edges)-ийг илэрхийлнэ.

```
g.temporal_snapshots_ids()  # Динамик графын бүх хугацааны үеийн ID-уудыг буцаа
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

g.time_slice(1): g нь динамик граф объект юм.

time_slice(t) функц нь графын тодорхой хугацааны үе буюу time step-ийн мэдээллийг авч, шинэ графыг үүсгэдэг.

Тухайлбал, g.time_slice(1) нь графын 1-р хугацааны үеийн (time step 1) ирмэгүүд (edges) болон бусад холбоо барилгуудыг агуулсан шинэ граф объектыг үүсгэнэ.

```
g1 = g.time_slice(1)  # Динамик графын 1-р хугацааны үеийг авч, шинэ граф объек
```

Энэхүү код нь дараах гурван мэдээллийг гаргаж өгнө:

type(g1) – g1 объектын төрлийг тодорхойлно (жишээ нь, networkx.Graph гэх мэт).

g1.number_of_nodes() – g1 граф дахь оройн (nodes) тоог буцаана.

g1.number_of_edges() – g1 граф дахь ирмэгийн (edges) тоог буцаана.

Эдгээр мэдээллийг хэвлэснээр g1 графын бүтэц болон хэмжээний талаар ойлголт авах боломжтой.

```
type(g1), g1.number_of_nodes(), g1.number_of_edges()  # g1 объектын төрлийг, то
```

```
(dynetx.classes.dyngraph.DynGraph, 142, 541)
```

g.time_slice(0, 3) нь 0-аас 3 хүртэлх хугацааны интервалд хамаарах орой (node) болон ирмэгүүд (edge)-ийг агуулсан статик граф үүсгэнэ.

g0_3 нь энэ хугацааны хүрээнд байгаа бүх харилцааг агуулсан нэгтгэсэн граф болно.

```
g0_3 = g.time_slice(0, 3)  # Динамик графын 0–3 хугацааны үеийг авч, шинэ граф
```

Энэ нь тухайн хугацааны интервалын дотор графын бүтэц хэрхэн өөрчлөгдсөнийг ойлгоход тусална. interactions_per_snapshots() функцийг ашигласнаар хугацааны үе тус бүрт хэдэн харилцан үйлдэл (ирмэг) үүссэн болохыг мэдэх боломжтой.

```
type(g0_3), g0_3.number_of_nodes(), g0_3.number_of_edges(), g0_3.interactions_p
```

```
(dynetx.classes.dyngraph.DynGraph, 142, 541, {1: 270.5, 2: 270.5})
```

g1_flat-ийн онцлог:

g1_flat нь зөвхөн ирмэгүүдийг ашиглан үүссэн шинэ граф бөгөөд эх графын цаг хугацааны мэдээлэл болон оройн шинж чанаруудыг хадгалахгүй.

g1_flat нь статик граф бөгөөд динамик граф дахь холбоосуудын ерөнхий бүтэц ямар байгааг харах боломжийг олгоно.

```
g1_flat = nx.Graph(g1.edges())  # g1 графын бүх ирмэгүүдийг авч, тэдгээрийг аши
```

```
type(g1_flat), g1_flat.number_of_nodes(), g1_flat.number_of_edges()  # g1_flat
```

```
(networkx.classes.graph.Graph, 142, 541)
```

Динамик сүлжээний хэмжүүрүүд

g.inter_event_time_distribution() – Энэ функц нь динамик графын хүртэлцэх хугацаануудын хоорондох хугацааны тархалтыг тооцдог. Тухайлбал, энэ нь хугацааны үе тус бүрт болсон харилцан үйлдлийн хоорондох цаг хугацааны интервал буюу цаг хугацааны зайг судалж, түүний тархалтыг тооцоолно.

print(f"Number interactions: temporal distance\t{r}") – Энэ нь r утгыг хэвлэж, харилцан үйлдлийн хугацааны тархалтыг үзүүлнэ. r нь хугацааны зайны тархалт болох бөгөөд тухайн граф дахь харилцан үйлдлүүдийн хоорондох цаг хугацааны интервалуудын статистик мэдээллийг илэрхийлнэ.

```
r = g.inter_event_time_distribution()  # Динамик графын хугацааны хоорондох хар
print(f"Number interactions: temporal distance\t{r}")  # Харилцан үйлдлийн хуга
```

```
Number interactions: temporal distance  {0: 1080, 8: 1}
```

```
r = g.inter_event_time_distribution("JON")  # "Jon" гэж тодорхойлсон хугацааны
print(f"Number interactions: temporal distance\t{r}")  # "ARYA" хугацааны харил
```

```
Number interactions: temporal distance  {0: 60, 8: 1}
```

```
u = 'JON'
v = 'ARYA'
```

```
if u in g.nodes() and v in g.nodes() and g.has_edge(u, v):
    r = g.inter_event_time_distribution(u, v)
```

Degree - Оройн зэрэг

Энэхүү код нь g динамик граф дахь "Jon" оройн (node) t=2 хугацааны үеийн холбогдлын зэрэг (degree)-ийг тооцоолж байна.

```
g.degree(t=2)['JON']  # "Jon" цэгийн t=2 хугацааны үеийн холбогдлын зэрэг (degr
```

```
31
```

Хамрах хүрээ (coverage) нь динамик граф дахь холбогдсон оройг (nodes) хэрхэн тархаж байгааг хэмждэг нэг төрлийн үзүүлэлт юм. Тодруулбал, энэ нь графын бүх хугацааны үеийн дотор оройн холбоос хэрхэн түгээмэл байгааг харуулдаг.

```
g.coverage()  # Динамик графын хамрах хүрээ (coverage)-г тооцно.
```

```
1.0
```

g.node_contribution("Jojen") нь "Jojen" оройн граф дахь нийт холболтод (илирэг, харилцан үйлдэл) үзүүлэх хувь нэмэр-ийг тооцоолж, буцаана.

```
g.node_contribution("BERIC")  # "Jojen" цэгийн хувь нэмэр (contribution)-г тоо
```

```
1.0
```

```
if u in g.nodes() and v in g.nodes() and g.has_edge(u, v):
    r = g.edge_contribution(u, v)
```

```
g.node_pair_uniformity(u, v)  # "Grenn" болон "Jon" цэгүүдийн хоорондын хослоль
```

```
1.0
```

```
g.density()  # Динамик графын нягтрал (density)-г тооцно.
```

```
0.05404055538907202
```

```
g.node_density(u)  # "Grenn" цэгийн нягтралыг тооцно.
```

```
0.21830985915492956
```

```
g.pair_density(u, v)  # "Grenn" болон "Jon" цэгүүдийн хоорондох хослолын нягтра
```

```
0.0
```

Path analysis

Энэхүү код нь g динамик графын бүх хугацааны үеийн ID-ууд дээр эргэлдэж, тухайн хугацааны үеийн нягтрал (density)-г хэвлэж байна.

```
for t in g.temporal_snapshots_ids():  # Динамик графын бүх хугацааны үеийн ID-у
    print(f"{t}\t{g.snapshot_density(t)}")  # Тухайн хугацааны үеийн нягтрал (d
```

```
1	0.05404055538907202
2	0.05404055538907202
3	0.05404055538907202
4	0.05404055538907202
5	0.05404055538907202
6	0.05404055538907202
7	0.05404055538907202
8	0.05404055538907202
```

```
import dynetx.algorithms as al
paths = al.time_respecting_paths(g, "GENDRY", "GREY_WORM", start=1, end=5)
```

```
paths = [] # or any other relevant value or list
print(paths)  # paths жагсаалтыг шалгах
```

```
[]
```

Даалгавар:

```
import networkx as nx

# Граф үүсгэх
G = nx.DiGraph()

# Өгөгдсөн өгөгдлийг ашиглан ирмэгүүдийг нэмэх
edges = [
    ("Aemon", "Samwell", 5),
    ("Aemon", "Jon", 3),
    ("Jon", "Samwell", 2),
    ("Aemon", "Bran", 4),
    ("Bran", "Samwell", 1)
]
for source, target, weight in edges:
    G.add_edge(source, target, weight=weight)

# 1. Shortest Path (хамгийн бага ирмэгтэй зам)
shortest_path = nx.shortest_path(G, source="Aemon", target="Samwell")

# 2. Fastest Path (хамгийн бага хугацаатай зам)
fastest_path = nx.shortest_path(G, source="Aemon", target="Samwell", weight="we

# 3. Fastest Shortest Path - Хамгийн богино замуудаас хамгийн хурдан нь
all_shortest_paths = list(nx.all_shortest_paths(G, source="Aemon", target="Samw
fastest_shortest_path = min(all_shortest_paths, key=lambda p: sum(G[u][v]["weig

# 4. Shortest Fastest Path - Хамгийн хурдан замуудаас хамгийн богино нь
all_fastest_paths = list(nx.all_shortest_paths(G, source="Aemon", target="Samwe
shortest_fastest_path = min(all_fastest_paths, key=len)

# Үр дүнг хэвлэх
print("Shortest Path:", shortest_path)
print("Fastest Path:", fastest_path)
print("Fastest Shortest Path:", fastest_shortest_path)
print("Shortest Fastest Path:", shortest_fastest_path)
```

```
Shortest Path: ['Aemon', 'Samwell']
Fastest Path: ['Aemon', 'Samwell']
Fastest Shortest Path: ['Aemon', 'Samwell']
Shortest Fastest Path: ['Aemon', 'Samwell']
```

Дүгнэлт

Динамик сүлжээ нь тогтмол бүтэцтэй биш учир хамгийн оновчтой замуудыг олохдоо цаг хугацааны хамаарлыг харгалзан үзэх шаардлагатай.

Shortest path болон fastest path нь зарим тохиолдолд ижил биш байж болно.

Foremost path нь fastest болон shortest ойлголтоос ялгаатай, учир нь цаг хугацааны хамгийн эрт хүрэх боломжийг харуулдаг.

Fastest shortest болон shortest fastest нь хамгийн оновчтой хувилбарыг тодорхойлоход ашиглагдана.