

Mahire Zühal Özdemir

19360859015

PROJE FİNAL 7

SNAP, Stanford Ağ Analizi Projesi'nin kısaltılmış halidir.

Grafik ve Ağ Türleri

SNAP; grafikleri ve ağları destekler. Graflar topolojileri tanımlar. Ağlar, ağın düğümlerinde ve/veya kenarlarında veri bulunan grafiklerdir. Düğümlerde ve kenarlarda bulunan veri türleri basitçe şablon parametreleri olarak geçirilir, bu da düğümler ve kenarlar üzerinde zengin verilere sahip çeşitli ağ türlerini uygulamak için çok hızlı ve kullanışlı bir yol sağlar.

SNAP'ta grafik türleri:

TUNGraph: yönlendirilmemiş çizge (sıralanmamış düğüm çifti arasında tek kenar)

TNGraph: yönlendirilmiş çizge (sıralı bir düğüm çifti arasında tek yönlendirilmiş kenar)

TNEGraph: yönlendirilmiş çoklu grafik (bir çift düğüm arasında birden fazla yönlendirilmiş kenar)

SNAP'ta ağ türleri:

TNodeNet<TNodeData>: TNGraph gibi, ancak her düğüm için TNodeData nesnesi ile

TNodeEDatNet<TNodeData, TEdgeData>: TNGraph gibi, ancak her düğümde TNodeData ile ve her kenarda TEdgeData

TNodeEdgeNet<TNodeData, TEdgeData>: TNEGraph gibi, ancak her düğümde TNodeData ile ve her kenarda TEdgeData

TNEANet: TNEGraph gibi, ancak düğümler ve kenarlar üzerinde özniteliklerle. Öznitelikler çalışma zamanında tanımlanabildikleri için dinamikler.

TBigNet<TNodeData>: TNodeNet'in bellek açısından verimli uygulaması bellek parçalanması ve milyarlarca kenarın yeterli RAM ile işlenmesi

Öncelikle SNAP kütüphanesini indirip import ediyoruz.

```
PS C:\Users\zuhal\Desktop\bioinformaticProjects> python -m pip install snap-stanford
```

Daha sonra kodumuzu python dosyamıza yazıyoruz.

Graph Oluřturma:

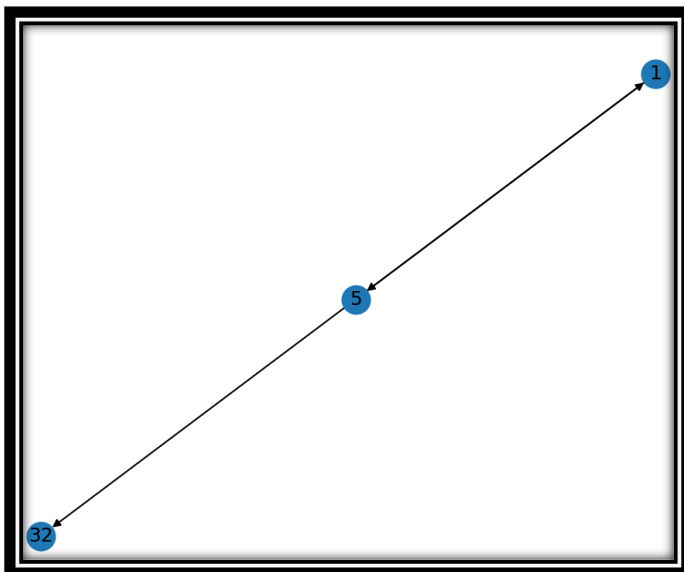
```
import snap  
  
Graph = snap.TNGraph.New()  
Graph.AddNode(1)  
Graph.AddNode(5)  
Graph.AddNode(32)  
Graph.AddEdge(1,5)  
Graph.AddEdge(5,1)  
Graph.AddEdge(5,32)
```

yazıyoruz. İlk olarak Snap kütüphanesinden TNGraph sınıfını kullanarak Graph adında yönlendirilmiş graph oluřturuyoruz .

- İlk olarak graph'a id'si 1 olan düğüm ekliyoruz.
- Graph'a id'si 5 olan düğüm ekliyoruz.
- Graph'a id'si 32 olan düğüm ekliyoruz.
- Graph'a 1 numaralı düğümünden 5 numaralı düğüme yönlendirilmiş kenar ekliyoruz.
- Graph'a 5 numaralı düğümünden 1 numaralı düğüme yönlendirilmiş kenar ekliyoruz.(çift yönlü)
- Graph'a 5 numaralı düğümünden 32 numaralı düğüme yönlendirilmiş kenar ekliyoruz.

Bu graph'ı networkx ve matplotlib kütüphanesi kullanarak görselleřtirebiliriz.

```
import networkx as nx  
import matplotlib.pyplot as plt  
G2 = nx.DiGraph()  
for node in Graph.Nodes():  
    G2.add_node(node.GetId())  
for edge in Graph.Edges():  
    G2.add_edge(edge.GetSrcNId(), edge.GetDstNId())  
nx.draw(G2, with_labels=True)  
plt.show()
```



İteratörler:

- 100 düğüm ve 1000 kenardan oluşan yönlü bir graph oluşturulur.
- Herbir düğümün önce id'si sonra giren ve çıkan kenar sayısı yazdırılır.

```
import snap

Graph = snap.GenRndGnm(snap.PNGraph, 100, 1000)

for NI in Graph.Nodes():
    print("node id %d with out-degree %d and in-degree %d" % (NI.GetId(), NI.GetOutDeg(), NI.GetInDeg()))

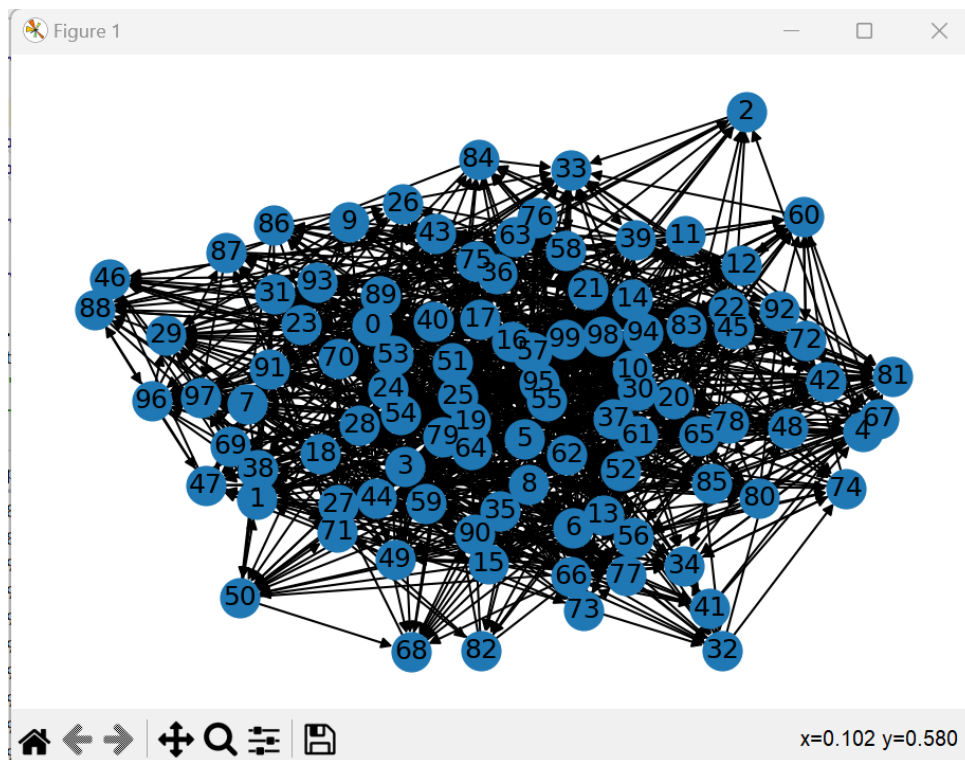
for EI in Graph.Edges():
    print("edge (%d, %d)" % (EI.GetSrcNId(), EI.GetDstNId()))

for NI in Graph.Nodes():
    for e in NI.GetOutEdges():
        print("edge (%d %d)" % (NI.GetId(), e))
```

```
C:\Users\zuha\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuha/Desktop/bioinformaticProjects/proje7.py
node id 0 with out-degree 13 and in-degree 13
node id 1 with out-degree 9 and in-degree 12
node id 2 with out-degree 4 and in-degree 8
node id 3 with out-degree 11 and in-degree 14
node id 4 with out-degree 8 and in-degree 10
node id 5 with out-degree 10 and in-degree 8
node id 6 with out-degree 11 and in-degree 7
node id 7 with out-degree 7 and in-degree 8
node id 8 with out-degree 10 and in-degree 12
node id 9 with out-degree 7 and in-degree 8
node id 10 with out-degree 15 and in-degree 16
node id 11 with out-degree 7 and in-degree 9
node id 12 with out-degree 12 and in-degree 11
node id 13 with out-degree 11 and in-degree 9
node id 14 with out-degree 9 and in-degree 13
node id 15 with out-degree 10 and in-degree 14
node id 16 with out-degree 11 and in-degree 12
node id 17 with out-degree 16 and in-degree 12
node id 18 with out-degree 10 and in-degree 7
node id 19 with out-degree 20 and in-degree 12
node id 20 with out-degree 17 and in-degree 10
node id 21 with out-degree 9 and in-degree 13
node id 22 with out-degree 11 and in-degree 8
```

```
edge (0, 20)
edge (0, 26)
edge (0, 43)
edge (0, 44)
edge (0, 56)
edge (0, 59)
edge (0, 62)
edge (0, 70)
edge (0, 72)
edge (0, 86)
edge (0, 89)
edge (0, 95)
edge (0, 96)
edge (1, 9)
edge (1, 19)
edge (1, 30)
edge (1, 34)
edge (1, 53)
edge (1, 56)
edge (1, 59)
```

```
edge (0 20)
edge (0 26)
edge (0 43)
edge (0 44)
edge (0 56)
edge (0 59)
edge (0 62)
edge (0 70)
edge (0 72)
edge (0 86)
edge (0 89)
edge (0 95)
edge (0 96)
edge (1 9)
edge (1 19)
edge (1 30)
edge (1 34)
edge (1 53)
edge (1 56)
edge (1 59)
```



Genel olarak graph veri tipleri çeşitli iteratörleri döndürmek için aşağıdaki fonksiyonları kullanır:

BegNI(): ilk düğüme giden iterator

EndNI(): son düğümden bir önceki düğüme giden iterator

GetNI(u): u kimliğine sahip düğüme giden iterator

BegEI(): ilk kenara giden iterator

EndEI(): son kenardan bir önceki kenara giden iterator

GetEI(u,v): (u,v) kenarına giden iterator

GetEI(e): e id'li kenara giden iterator (yalnızca çoklu ağlar için)

GetId(): düğüm kimliğini döndürür

GetOutDeg(): bir düğümün dış derecesini döndürür

GetInDeg(): bir düğümün derece içi değerini döndürür

GetOutNId(e): e-inci dış kenarın uç noktasının düğüm kimliğini döndürür

GetInNId(e): e-inci kenarın uç noktasının düğüm kimliğini döndürür

IsOutNId(int NId): n düğüm kimliğine mi işaret ediyoruz

IsInNId(n): n düğüm kimliği bize işaret ediyor mu

IsNbhNId(n): n düğümü komşumuz mu

GetDat(): düğümle ilişkili TNodeData veri türünü döndürür

GetOutNDat(e): e-inci dış kenarın uç noktasındaki düğümle ilişkili verileri döndürür

GetInNDat(e): e-inci kenarın uç noktasındaki düğümle ilişkili verileri döndürür

GetOutEDat(e): e-inci çıkış kenarı ile ilişkili verileri döndürür

GetInEDat(e): e-inci kenar ile ilişkili verileri döndürür

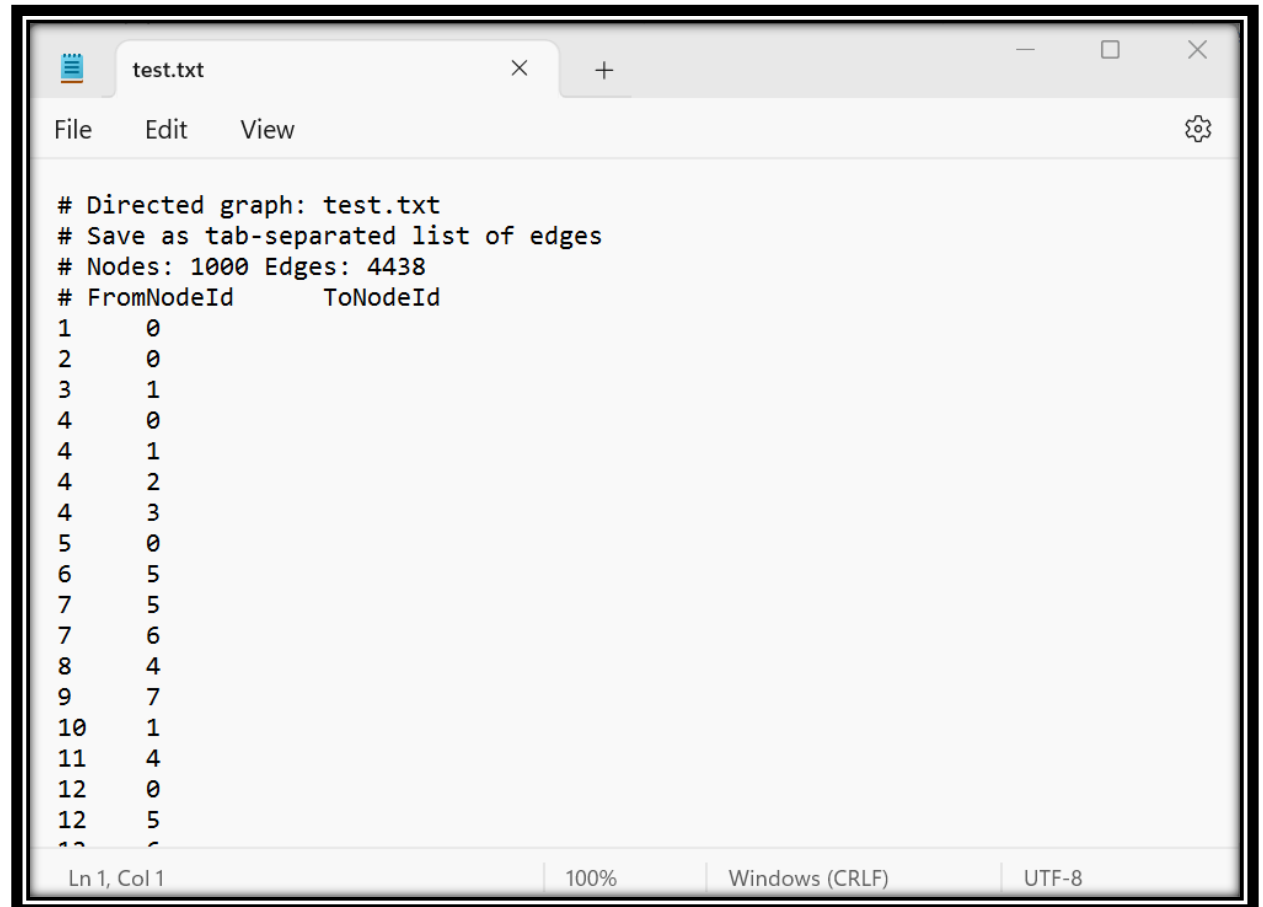
Input/Output:

GenForestFire fonksiyonu kullanılarak, 1000 düğümlü ve düğümler arasında ileri geri yönlü 0.35 olasılıkla gerçekleşen bir ağ oluşur.

```
Graph = snap.GenForestFire(1000, 0.35, 0.35)

# Ağ bilgilerini yazdırma
FOut = snap.TFOut("test.graph")
Graph.Save(FOut)
# Dosyadan grafi yükle
FIn = snap.TFIn("test.graph")
G2 = snap.TNGraph.Load(FIn)

# Grafi text dosyasına kaydet
snap.SaveEdgeList(Graph, "test.txt")
# Text dosyasından grafi yükle
G2 = snap.LoadEdgeList(snap.PNGraph, "test.txt", 0, 1)
```



```
# Directed graph: test.txt
# Save as tab-separated list of edges
# Nodes: 1000 Edges: 4438
# FromNodeId      ToNodeId
1      0
2      0
3      1
4      0
4      1
4      2
4      3
5      0
6      5
7      5
7      6
8      4
9      7
10     1
11     4
12     0
12     5
```

Grafikleri ve Ağları Manipüle Etme:

```
import snap

G = snap.GenForestFire(1000, 0.35, 0.35)

FOut = snap.TFOut("test.graph")
G.Save(FOut)
FOut.Flush()

FIn = snap.TFIn("test.graph")
G2 = snap.TNGraph.Load(FIn)
```

```
import snap

# Yeni bir graf oluşturun
G = snap.GenForestFire(1000, 0.35, 0.35)
# Yönlü grafi yönsüz grafa dönüştür
UG = snap.ConvertGraph(snap.PUNGraph, G)
# En büyük bağlı bileşeni al
WccG = snap.GetMxWcc(G)
# {0,1,2,3,4,5} düğüm kümesine göre alt grafi al
SubG = snap.GetSubGraph(G, snap.TIntV.GetV(0, 1, 2, 3, 4))
# G'nin 3-core'unu al
Core3 = snap.GetKCore(G, 3)
# Derecesi 10 olan düğümleri sil
snap.DelDegKNodes(G, 10)
```

```
C:\Users\zuhal\Desktop\bioinformaticProjects\venv\Scripts\python.exe C:/Users/zuhal/Desktop/bioinformaticProjects/proje7.py

***ForestFire: GeoFire Nodes:1000 StartNodes:1 Take2AmbProb:0
                FwdBurnP:0.35 BckBurnP:0.35 ProbDecay:1 Orphan:0
(1000, 4230) burned: [4,9,1] [0.01s]

Process finished with exit code 0
```

Yapısal Özelliklerin Hesaplanması:

SNAP, ağların yapısal özelliklerini verimli bir şekilde hesaplamak için zengin işlevsellik sağlar. Fonksiyonlar TSnap isim alanının bir parçası olarak uygulanmaktadır.

```
import snap

G = snap.GenPrefAttach(1000, 3)
CntV = snap.TIntPrV()
snap.GetWccSzCnt(G, CntV)
snap.GetOutDegCnt(G, CntV)
EigV = snap.TFltV()
snap.GetEigVec(G, EigV)
snap.GetBfsFullDiam(G)
snap.GetTriads(G)
snap.GetClustCf(G)
```