Demonstration of networKit

```
1 !git clone https://github.com/networkit/networkit.git
```

```
Cloning into 'networkit'...
remote: Enumerating objects: 78077, done.
remote: Counting objects: 100% (399/399), done.
remote: Compressing objects: 100% (190/190), done.
remote: Total 78077 (delta 221), reused 298 (delta 190), pack-reused 77678
Receiving objects: 100% (78077/78077), 259.62 MiB | 30.03 MiB/s, done.
Resolving deltas: 100% (56668/56668), done.
```

```
1 !pip install networkit
```

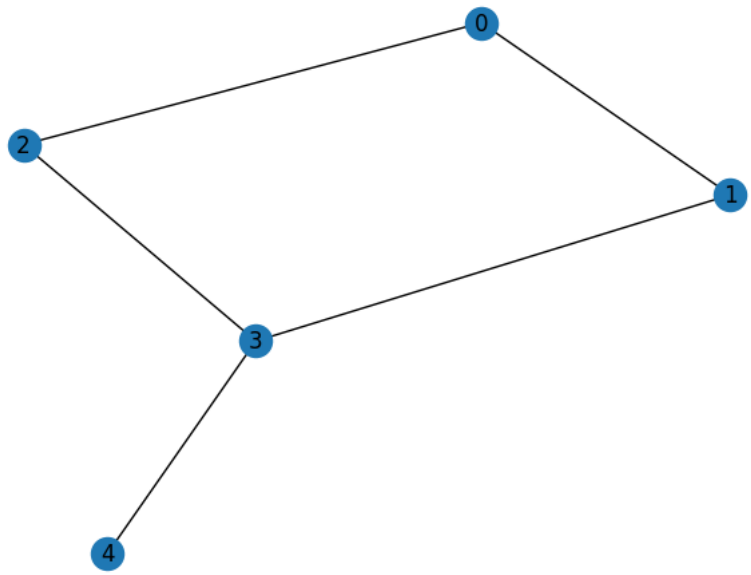
```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public
Collecting networkit
  Downloading networkit-10.1-cp39-cp39-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (9.9
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.9/9.9 MB 87.5 MB/s eta 0:00:00
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages (from netwo
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-packages (from netwo
Installing collected packages: networkit
Successfully installed networkit-10.1
```

```
1 import networkit as nk
```

```
1 # create a small undirected graph with 5 nodes and 5 edges
2 G = nk.Graph(5)
3 G.addEdge(0, 1)
4 G.addEdge(0, 2)
5 G.addEdge(1, 3)
6 G.addEdge(2, 3)
7 G.addEdge(3, 4)
8
```

```
True
```

```
1 import networkx as nx
2 import matplotlib.pyplot as plt
3
4 # create a graph
5 G = nx.Graph()
6 G.add_edges_from([(0, 1), (0, 2), (1, 3), (2, 3), (3, 4)])
7
8 # draw the graph
9 nx.draw(G, with_labels=True)
10 plt.show()
11
```



calculate degree centrality for each node

```
dc = nk.centrality.DegreeCentrality(G)
dc.run()
print("Degree centrality:", dc.ranking())

```

```
Degree centrality: [(3, 3.0), (0, 2.0), (1, 2.0), (2, 2.0), (4, 1.0)]
```

calculate betweenness centrality for each node

```

bc = nk.centrality.Betweenness(G)
bc.run()
print("Betweenness centrality:", bc.ranking())

```

```
Betweenness centrality: [(3, 7.0), (1, 2.0), (2, 2.0), (0, 1.0), (4, 0.0)]
```

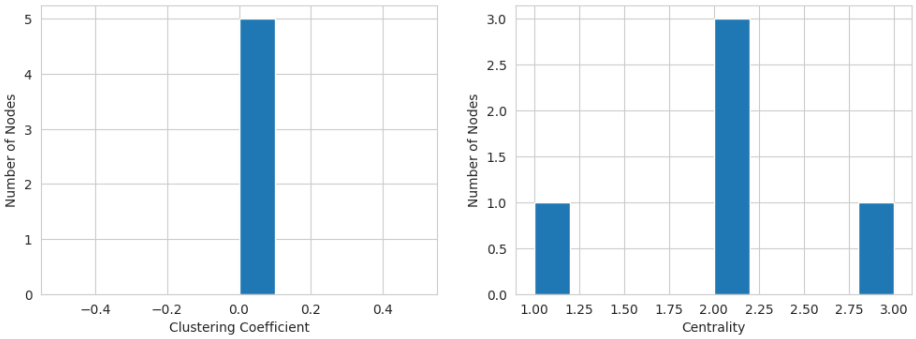
calculate shortest paths between all pairs of nodes

```
G = nk.Graph(5)

# add edges to the graph
G.addEdge(0, 1)
G.addEdge(0, 3)
G.addEdge(1, 2)
G.addEdge(1, 3)
G.addEdge(2, 3)
G.addEdge(2, 4)
apsp = nk.distance.APSP(G)
apsp.run()

# get the matrix of shortest path lengths
shortest_paths = apsp.getDistances()

# print the shortest path between node 0 and all other nodes
for i, distance in enumerate(shortest_paths[0]):
    print("Shortest path between nodes 0 and {}: {}".format(i, distance))

```

```
Shortest path between nodes 0 and 0: 0.0
Shortest path between nodes 0 and 1: 1.0
Shortest path between nodes 0 and 2: 2.0
Shortest path between nodes 0 and 3: 1.0
Shortest path between nodes 0 and 4: 3.0
```

```
import networkit as nk
import matplotlib.pyplot as plt

# create a small undirected graph with 5 nodes and 5 edges
G = nk.Graph(5)
G.addEdge(0, 1)
G.addEdge(0, 2)
G.addEdge(1, 3)
G.addEdge(2, 3)
G.addEdge(3, 4)

# calculate the clustering coefficient of each node
cc = nk.centrality.LocalClusteringCoefficient(G)
cc_values = cc.run().scores()

# calculate the centrality of each node
centrality = nk.centrality.DegreeCentrality(G)
centrality_values = centrality.run().scores()


# plot the clustering coefficient, centrality, and degree distribution
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12,4))

ax1.hist(cc_values, bins=10)
ax1.set_xlabel("Clustering Coefficient")
ax1.set_ylabel("Number of Nodes")
```

```
28
29 ax2.hist(centrality_values, bins=10)
30 ax2.set_xlabel("Centrality")
31 ax2.set_ylabel("Number of Nodes")
32
33
34
35 plt.show()
36
```



Community Detection

```
 1 import networkit as nk
 2 import matplotlib.pyplot as plt
 3
 4 # create a small undirected graph with 5 nodes and 5 edges
 5 G = nk.Graph(5)
 6 G.addEdge(0, 1)
 7 G.addEdge(0, 2)
 8 G.addEdge(1, 3)
 9 G.addEdge(2, 3)
10 G.addEdge(3, 4)
11
12 # perform community detection using the Louvain algorithm
13 cd = nk.community.detectCommunities(G, algo=nk.community.PLM(G))
14
15
```

```
Communities detected in 0.00455 [s]
solution properties:
-------------------  ----
# communities        2
min community size   2
max community size   3
avg. community size  2.5
imbalance            1
edge cut             2
edge cut (portion)   0.4
modularity           0.08
-------------------  ----
```

Parallelism

The degree of parallelism can be controlled and monitored in the following way:

```
1 print(nk.setNumberOfThreads(4)) # set the maximum number of available threads
```

```
None
```

```
1 print(nk.getMaxNumberOfThreads()) # see maximum number of available threads
```

```
4
```

```
1 print(nk.getCurrentNumberOfThreads()) # the number of threads currently executing
```

```
1
```

Clustering Coefficient

```
# create a small undirected graph with 5 nodes and 5 edges
G = nk.Graph(5)
G.addEdge(0, 1)
G.addEdge(0, 2)
G.addEdge(0, 3)
G.addEdge(1 ,2)
G.addEdge(3, 4)

# calculate the clustering coefficient of each node
cc = nk.centrality.LocalClusteringCoefficient(G)
print("Clustering coefficient of each node:", cc.run().scores())

```

```
Clustering coefficient of each node: [0.3333333333333333, 1.0, 1.0, 0.0, 0.0]
```

```
1
```


✓ 0s completed at 7:15 AM