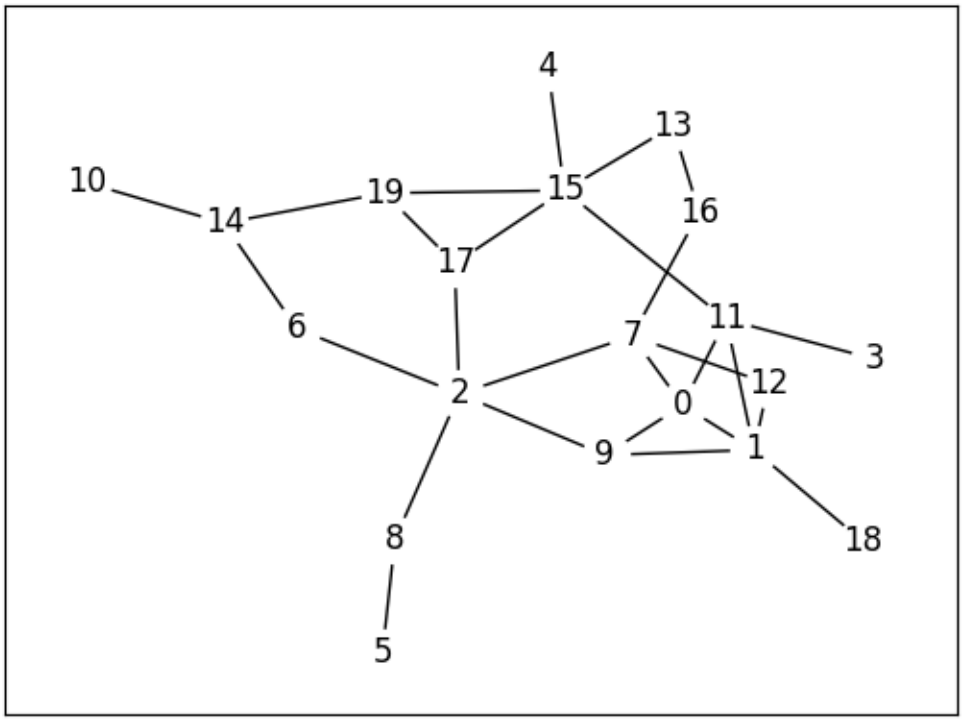
LISTA 2 – TECHNOLOGIE SIECIOWE

PIOTR MACIEJOŃCZYK

1. Stworzenie topologii grafu G:



Żaden wierzchołek nie jest izolowany oraz liczba krawędzi jest mniejsza niż 30.

2. Stworzenie macierzy natężeń N:

Poniższa funkcja tworzy macierz natężeń.

```python
# creating the intensity matrix
def create_intensity_matrix():
    N = [[0] * NODES_COUNT for _ in range(NODES_COUNT)]
    for i in range(0, NODES_COUNT):
        j = i+1
        while j < NODES_COUNT:
            # generate intensity
            N[i][j] = randint(MIN_CAPACITY, MAX_CAPACITY)
            j += 1
    return N
```

Wartości MIN_CAPACITY oraz MAX_CAPACITY określają liczbę przesyłanych pakietów od jednego węzła „i” do drugiego węzła „j”. Najpierw ustawiłem te wartości w ten sposób:

```python
MAX_CAPACITY = 1000
MIN_CAPACITY = 100
```

3. Stworzenie funkcji „c” oraz „a”:

a) funkcja c(e):

```python
# creating the "c" matrix where every number is the same
self._efficiency = [[0] * NODES_COUNT for _ in range(NODES_COUNT)]
for i in range(0, NODES_COUNT):
    for j in range(0, NODES_COUNT):
        if self._G[i][j] == 1:
            self._efficiency[i][j] = EFFICIENCY_DEFAULT
            self._efficiency[j][i] = EFFICIENCY_DEFAULT
```

Jak można zauważyć, moja funkcja przepustowości c(e) przyjmuje dla każdego kanału komunikacyjnego taką samą wartość, którą określiłem na:

```python
# efficiency in Gbps of one edge
EFFICIENCY_DEFAULT = 2 * (10 ** 9)
```

(czyli 2 Gbps)

b) funkcja a(e):

```
# creating the "a" function based on intensity matrix
def _create_flow_function(self, N):
    self._flow = [[0] * NODES_COUNT for _ in range(NODES_COUNT)]
    for i in range(0, NODES_COUNT):
        j = i + 1
        while j < NODES_COUNT:
            intensity = N[i][j]
            # creating list of paths between all nodes
            nodes = ds.find_path(self._G_ds, i, j).nodes
            for k in range(0, len(nodes) - 1):
                current = nodes[k]
                next = nodes[k + 1]
                self._flow[current][next] += intensity
                self._flow[next][current] += intensity
                # making sure a(e) < c(e) for each edge
                if self._efficiency[current][next] < self._flow[current][next] * PACKET_SIZE:
                    raise NetworkOverload()
            j += 1
```

Powyższa funkcja przepływu a(e) równocześnie sprawdza czy aby na pewno dla każdej krawędzi jest mniejsza niż wartości przepustowości.

4. Mierzenie niezawodności otrzymanej sieci:

a) obliczenie średniego opóźnienia pakietów:

```
# calculating T
def avg_delay(self, N):
    T = 0
    for i in range(0, NODES_COUNT):
        j = i + 1
        while j < NODES_COUNT:
            if self._G[i][j] == 1:
                # using the given equation: T = T + (a(e) / (c(e) / m - a(e)))
                T += self._flow[i][j] / ((self._efficiency[i][j] / PACKET_SIZE) - self._flow[i][j])
            j += 1
    N_sum = reduce(lambda so_far, curr: so_far + sum(curr), N, 0)
    # finally dividing it by G
    T /= N_sum
    return T
```

b) funkcja odpowiedzialna za obliczanie niezawodności:

```python
# function responsible for measuring the reliability of a given network
def measure(self, intensity, edge_probability, T_max):
    successes = 0
    network_overloads = 0
    nopath_errors = 0
    timeouts = 0
    total_count = 500
    # repeating the test a lot
    for _ in range(0, total_count):
        # perhaps damaging some edges
        for i in range(0, NODES_COUNT):
            for j in range(0, NODES_COUNT):
                if self._G[i][j] == 1 and random() >= edge_probability:
                    self._G[i][j] = 0

        # creating the shortest paths and adding "c" function
        self._prepare_for_testing()
            self._prepare_for_testing()
            # creating the "a" function
            try:
                self._create_flow_function(intensity)
            except ds.NoPathError:
                # an important edge has been destroyed, hence resetting the topology
                self._reset()
                nopath_errors += 1
                continue
            except NetworkOverload:
                # the connection has been overloaded and is unable to sustain traffic
                network_overloads += 1
                self._reset()
                continue

            # comparing T to T_max
            T = self.avg_delay(N)
            if T < T_max:
                successes += 1
            else:
                timeouts += 1
            self._reset()

        return successes / total_count, network_overloads, nopath_errors, timeouts
```

Powyższa funkcja przyjmuje jako parametry macierz natężeń sieci, prawdopodobieństwo niezniszczenia się kanału komunikacyjnego oraz maksymalne opóźnienie pakietów. W pierwszej

pętli na podstawie podanego prawdopodobieństwa program „uszkadza” kanały komunikacyjne w sieci. Następnie powstają funkcje ‘c’ oraz ‘a’ i najpierw sprawdzane jest czy żaden z węzłów nie jest izolowany. Jeśli wszystkie węzły mają przynajmniej jednego sąsiada, to program sprawdza, czy dla każdej krawędzi c(e) > a(e). Na sam koniec po wykonaniu się głównej pętli ustaloną liczbę razy funkcja zwraca miarę niezawodności; liczbę przypadków, kiedy kanał komunikacyjny został przeciążony; liczbę sytuacji, kiedy odizolowano węzeł; liczbę przypadków, kiedy $T >= T_{max}$.

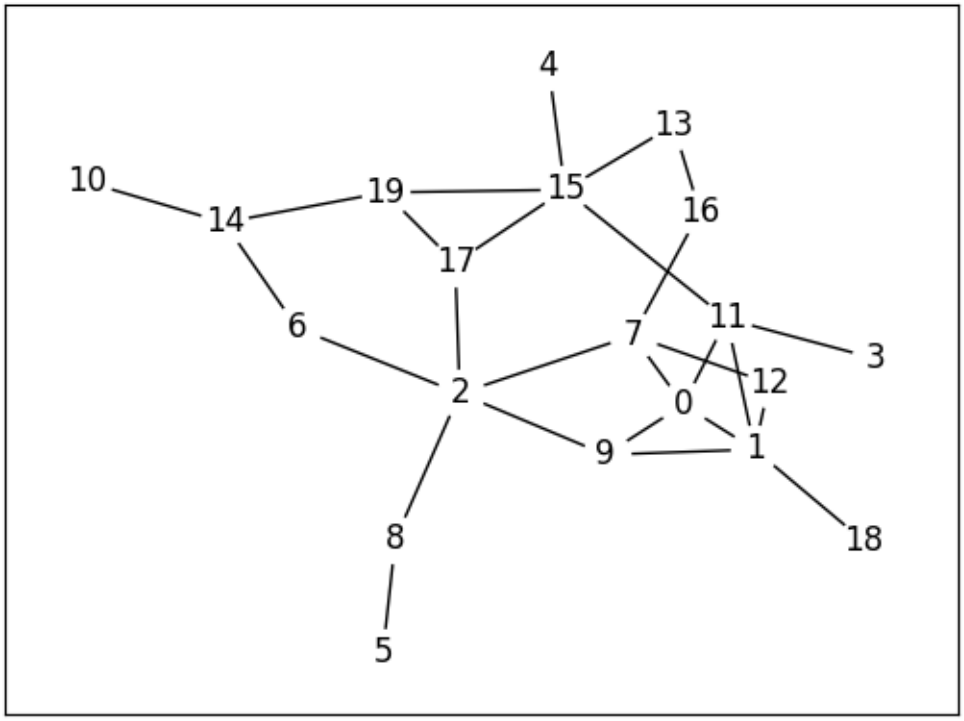
Oto przykładowe wywołanie funkcji:

```
rel = Reliability(G)

reliability, network_overloads, no_path_errors, timeouts = rel.measure(N, 0.99, 0.001)

print("reliability      :", reliability)
print("network overloads:", network_overloads)
print("nopath errors    :", no_path_errors)
print("timeouts         :", timeouts)
```

5. Stopniowe zwiększanie wartości w macierzy natężeń:

Przyjęte parametry:

a) topologia G:



b) prawdopodobieństwo p = 99%

c) maksymalne opóźnienie T_{max} = 0.001s

Macierz natężeń w każdym przypadku była tworzona tymi dwoma stałymi:

```
# maximum and minimum capacity of packets traversing one edge
MAX_CAPACITY = 1000
MIN_CAPACITY = 100
```

W testach zmieniałem stałą i mnożyłem przez zakres MIN_CAPACITY do MAX_CAPACITY.

```
for i in range(0, NODES_COUNT):
    j = i+1
    while j < NODES_COUNT:
        # generate intensity
        N[i][j] = randint(MIN_CAPACITY, MAX_CAPACITY) # * chosen constant
        j += 1
```

Dla const = 1:

```
reliability      : 0.946
network overloads: 0
nopath errors    : 27
timeouts         : 0
```

Dla const = 5:

```
reliability      : 0.938
network overloads: 0
nopath errors    : 31
timeouts         : 0
```

Dla const = 10:

```
reliability      : 0.924
network overloads: 0
nopath errors    : 38
timeouts         : 0
```

Dla const = 15:

```
reliability      : 0.932
network overloads: 6
nopath errors    : 28
timeouts         : 0
```

Dla const = 20:

```
reliability      : 0.886
network overloads: 23
nopath errors    : 34
timeouts         : 0
```

Dla const = 25:

```
reliability      : 0.04
network overloads: 452
nopath errors    : 28
timeouts         : 0
```

Dla const = 30:

```
reliability      : 0.0
network overloads: 472
nopath errors    : 28
timeouts         : 0
```

WNIOSEK: Przy zwiększaniu wartości w macierzy natężeń rośnie niewydolność sieci i zwiększa się liczba przeciążeń krawędzi. Dlatego miara niezawodności spada, aż osiąga 0% wydajności, gdyż kanały komunikacyjne są przeciążone.

6. Stopniowe zwiększanie przepustowości:

Przyjęte parametry:

a) topologia G: taka jak punkt wyżej

b) macierz natężeń: taka jak punkt wyżej dla const = 1

c) prawdopodobieństwo p = 99%

Dla T_{max} = 0.000001s:

```
reliability      : 0.0
network overloads: 0
nopath errors    : 33
timeouts         : 467
```

Dla T_{max} = 0.00001s:

```
reliability      : 0.944
network overloads: 0
nopath errors    : 28
timeouts         : 0
```

Dla T_{max} = 0.0001s:

```
reliability      : 0.946
network overloads: 0
nopath errors    : 27
timeouts         : 0
```

Dla T_{max} = 0.001s:

```
reliability      : 0.954
network overloads: 0
nopath errors    : 23
timeouts         : 0
```

Dla T_{max} = 0.01s:

```
reliability      : 0.928
network overloads: 0
nopath errors    : 36
timeouts         : 0
```

Dla T_{max} = 0.1s:

```
reliability      : 0.938
network overloads: 0
nopath errors    : 31
timeouts         : 0
```

WNIOSEK: Po stopniowym zwiększaniu T_{max} przestały występować sytuacje, w których pakiet nie zdążył dostać się do celu, przez co miara niezawodności od progu T_{max} = 0.000001s znacznie się zwiększyła i udowodniła, iż dobra jest sieć z dużą przepustowością.

7. Dodawanie nowych krawędzi:

Przyjęte parametry:

a) topologia G: na początku tak jak punkt wyżej

b) macierz natężeń: taka jak punkt wyżej

c) prawdopodobieństwo p = 99%

d) maksymalne opóźnienie T_{max} = 0.001s

Dla |E| < 20:

```
reliability      : 0.844
network overloads: 0
nopath errors    : 78
timeouts         : 0
```

Dla |E| < 25:

```
reliability      : 0.87
network overloads: 0
nopath errors    : 65
timeouts         : 0
```

Dla |E| < 30:

```
reliability      : 0.956
network overloads: 0
nopath errors    : 22
timeouts         : 0
```

Dla |E| < 35:

```
reliability      : 0.978
network overloads: 0
nopath errors    : 11
timeouts         : 0
```

Dla |E| < 40:

```
reliability      : 0.986
network overloads: 0
nopath errors    : 7
timeouts         : 0
```

Dla |E| < 45:

```
reliability      : 0.992
network overloads: 0
nopath errors    : 4
timeouts         : 0
```

Dla |E| < 50:

```
reliability      : 1.0
network overloads: 0
nopath errors    : 0
timeouts         : 0
```

WNIOSEK: Im więcej krawędzi w sieci, tym mniejsza szansa na nie znalezienie drogi docelowej oraz wyższa miara niezawodności całej sieci.