Sprawozdanie nr 2

Technologie sieciowe

Laboratorium piątek 9:15

Weronika Jasiak
236733

1. Cele

Zadanie polega na przetestowaniu napisanego wcześniej programu, który symuluje działanie sieci komputerowej oraz sprawdzenie parametrów takich jak niezawodność rozumianej jako m. in. spójności. Program został napisany w języku Java, a do przedstawienia sieci jako graf została użyta biblioteka `JGraphT`. W grafie węzły symbolizują grupy użytkowników, natomiast krawędzie oznaczają połączenia między nimi. Głównymi celami są:

- Sprawdzenie jak liczba krawędzi wpływa na spójność sieci biorąc pod uwagę różne prawdopodobieństwo zerwania połączenia dla różnych krawędzi,
- Sprawdzanie opóźnienia pakietu, przekraczania przepływu.

2. Realizacja

2.1. Badanie spójności sieci

Do badania spójności został napisany program, który najpierw dodaje do grafu węzły oraz krawędzie, następnie z konkretnym prawdopodobieństwem zrywa dane połączenie, a następnie sprawdza spójność grafu.
Została wykorzystana biblioteka `JGraphT`, która zawiera funkcje takie jak:

- `addVertex` – dodawanie węzła,
- `addEdge` – dodawanie krawędzi,
- `removeEdge` – usuwanie krawędzi,
- `isGraphConnected` – sprawdzanie czy graf jest spójny.

Dane potrzebne do stworzenia grafu tj. wierzchołki, krawędzie oraz prawdopodobieństwo nie uszkodzenia konkretnych połączeń znajdują się we wcześniej zaimplementowanych klasach. Poza samym obiektem grafu zostały również wykorzystane dwa obiekty typu `ArrayList` – jedna lista zawiera wszystkie grafy, natomiast druga przechowuje odpowiadające im krawędzie.

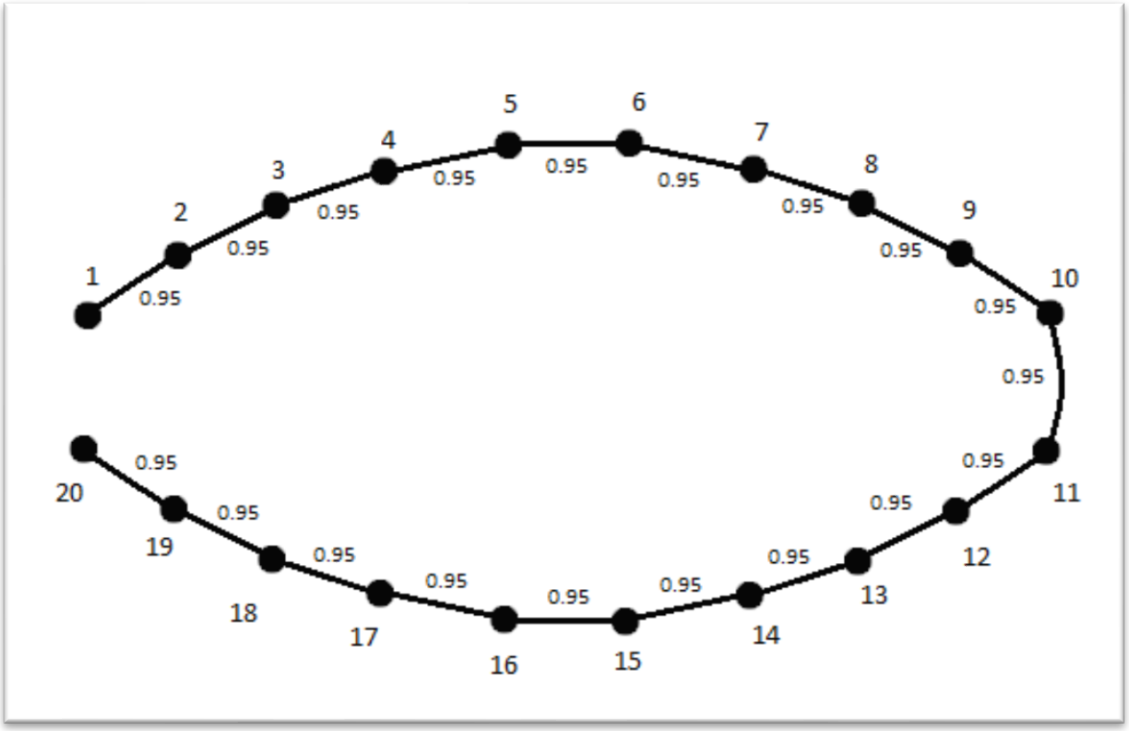
Do symulowania zerwania połączenia (usuwania losowych krawędzi) został użyty generator liczb pseudolosowych oraz prosta instrukcja warunkowa sprawdzająca czy wylosowana liczba jest większa od wagi krawędzi grafu symbolizującej odległość pomiędzy wierzchołkami.

```
double waga = graf.getEdgeWeight(krawedz);
double numer = random.nextDouble();
if (numer > waga) {
	// usuwamy krawędź
	graf.removeEdge(krawedz);
}
```

Program po uruchomieniu wczytuje wcześniej zaimplementowane przykłady grafów i z wykorzystaniem metody Monte Carlo sprawdza spójność grafu. Na koniec wypisywane są następujące dane:

- nazwa grafu,
- niezawodność,
- ilość pomyślnych prób,
- średnia liczba zerwanych połączeń.

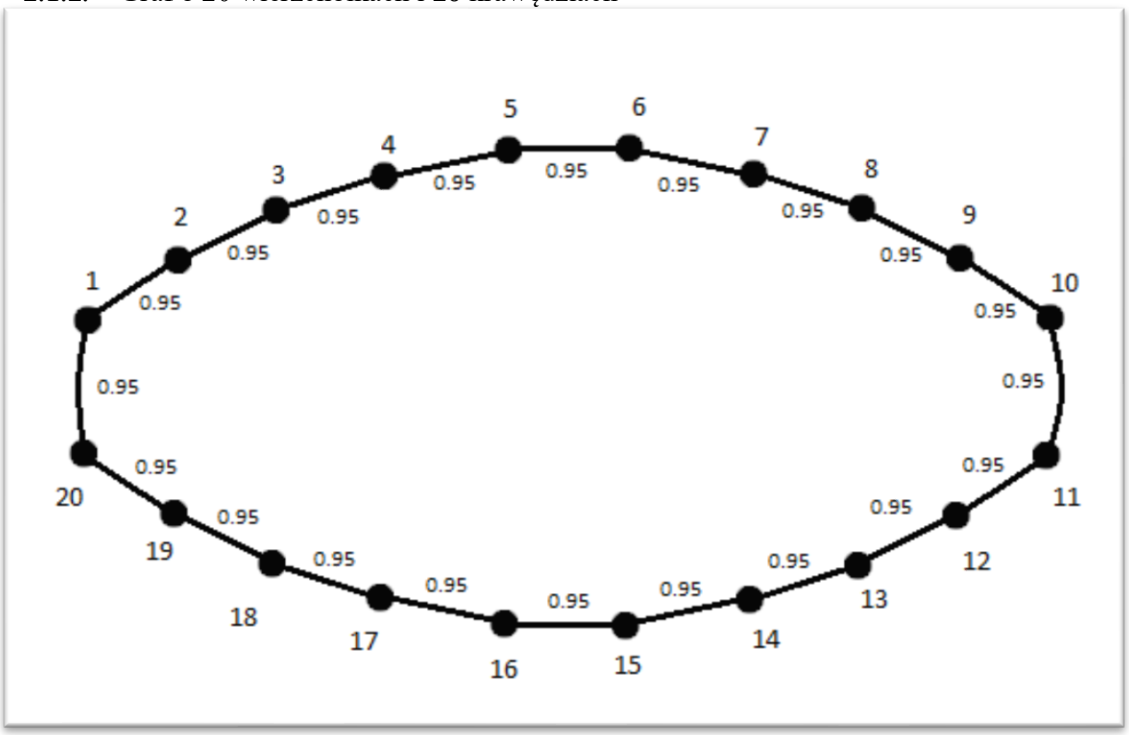
2.1.1. Graf o 20 wierzchołkach i 19 krawędziach



Graf 1.

Pierwszym rozpatrywanym przypadkiem będzie sieć zawierająca 20 wierzchołków oraz 19 krawędzi, o prawdopodobieństwie zachowania spójności połączenia równym 95 %. Jest to najprostszy możliwy spójny model sieci

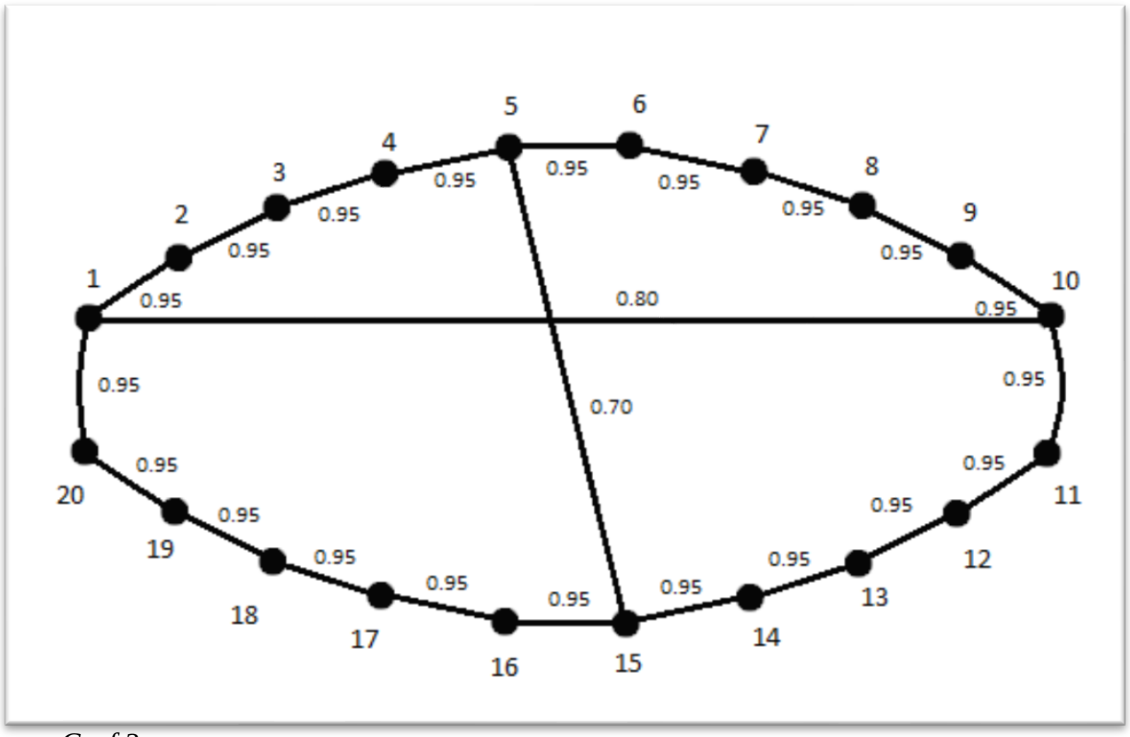
2.1.2. Graf o 20 wierzchołkach i 20 krawędziach



Graf 2.

W drugim przypadku do poprzedniego grafu została dodana dodatkowa krawędź `{1,20}`, o prawdopodobieństwie zachowania spójności równym 95 %, a więc taki samym jak w przypadku pozostałych krawędzi.

2.1.3. Graf o 20 wierzchołkach i 22 krawędziach



Graf 3.

W trzecim przypadku dodatkowo dodajemy krawędź `{1,10}`, o prawdopodobieństwie zachowania spójności równym 80 %, oraz krawędź `{5,15}`, o prawdopodobieństwie zachowania spójności równym 70 %.

2.1.4. Graf o 20 wierzchołkach i 26 krawędziach

W ostatnim przypadku do *Grafu 3.* dodatkowo dodajemy 4 losowo wybrane krawędzie o prawdopodobieństwie zachowania spójności równym 40 %.

2.2. Badanie opóźnień przepływu sieci

Do badania opóźnień przepływu sieci został napisany program w języku `Java` wykorzystujący bibliotekę `JGraphT`. Wszelkie testy zostaną przeprowadzone na proponowanych grafach o 10 wierzchołkach oraz mniej niż 20 krawędziach.

Dodatkowo została zaimplementowana klasa `GraphEdge` zawierająca daną krawędź oraz jej przepustowość i przepływ.

```java
public class GraphEdge extends DefaultEdge
    int capacity;
    int flow = 0;
    public GraphEdge(int capacity) {
        this.capacity = capacity;
    }
    public int getCapacity() {
        return capacity;
    }
    public int getFlow() {
        return flow;
    }
    public void setFlow(int flow) {
        this.flow = flow;
    }
}
```

Listing 2.

Dane potrzebne do stworzenia grafu wczytane są z plików, który zawierają:

- *graf.txt* - w pierwszej linijce znajdują się informacje o: niezawodności, maksymalnym opóźnieniu i ilość prób, a w następnych linijkach znajdują się informację o krawędziach: wierzchołki początkowy i końcowy oraz maksymalny transfer.
- *transfery.txt* – w pliku znajdują się informcje o transferach: wierzchołek początkowy, końcowy i ilość pakietów.

Ponownie poza samym obiektem grafu został również wykorzystany `ArrayList`, który przechowuje obiekty klasy `GraphEdge`, oraz macierze odzwierciedlające macierze odczytane z pliku.

Zostały również zaimplementowane następujące funkcję:

a)

```java
private static boolean CapacityTest(Graph<Integer, GraphEdge>
graph) {
        for (GraphEdge edge : graph.edgeSet()) {
                if (edge.getFlow() > edge.getCapacity()) {
                        return false;
                }
        }
        return true;
}
```

Listing 3.

Powyższa funkcja sprawdza, czy w którejś z krawędzi grafu nie następuje nadmiarowy przepływ (przepustowość mniejsza od przepływu). Jeżeli występuje taka sytuacja to zwracana jest wartość `true`, w przeciwnym przypadku zwracane jest `false`.

b)

```
// jeśli przepustowość jest większa od przepływu
if (CapacityTest graph)) {
        // wyliczanie opoznienia
        int sumOfPackages =
        Arrays.stream(flows2).flatMapToInt(Arrays::stream).sum();
        final Zadanie2 finalData = data;
        double sumFromEdges =
        graph.edgeSet().stream().mapToDouble(GraphEdge ->
        ((GraphEdge.getFlow() * finalData.largeOfPackage) /
        GraphEdge.getCapacity() - GraphEdge.getFlow())).sum();
        int late = (int) (sumFromEdges / sumOfPackages);
        lates.add(late);
        // jeśli opoznienie mniejsze od dopuszczalnego
        if (late < data.maxlate) {
                success++;
        }
}
```

Listing 4.

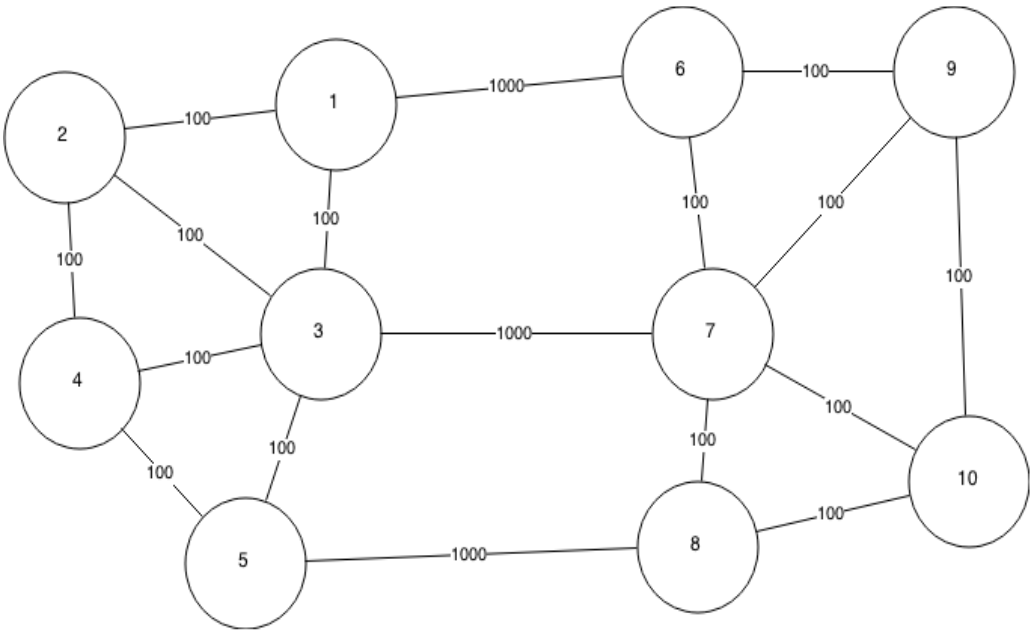
Powyższa warunek dla przepustowości większej od przepływu liczy opóźnienie pakietu.

Na koniec swojego działania program wypisuje:

- prawdopodobieństwo sukcesu,
- liczbę prób,
- liczbę porażek,
- średnie opóźnienie.

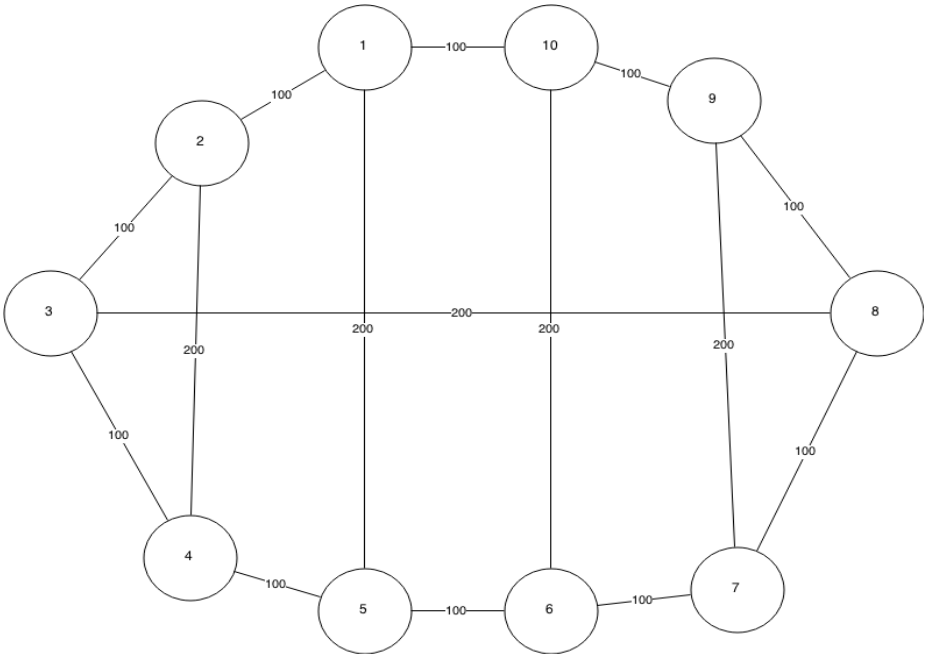
2.2.1. Proponowana topologia grafu

Zaproponowany model sieci zawiera 10 wierzchołków oraz 17 krawędzi.



Graf 4.

Zaproponowany model sieci zawiera 10 wierzchołków oraz 15 krawędzi.



Graf 5.

3. Wyniki

3.1. Badanie spójności

3.1.1. Wyniki dla grafu o 20 wierzchołkach i 19 krawędziach

```
Liczba prób: 1000
Liczba spójnych sieci: 383
Niezawodność: 38%
```

Listing 5.

W przypadku *Grafu 1.* zerwanie chociażby jednego połączenia skutkuje utratą spójności grafu. Mimo tego, że prawdopodobieństwo niezawodności każdego z połączeń jest wysokie i wynosi 95%, to prawdopodobieństwo niezawodności sieci wynosi zaledwie 38%, ponieważ graf ten nie posiada żadnych dodatkowych połączeń.

3.1.2. Wyniki dla grafu o 20 wierzchołkach i 20 krawędziach

```
Liczba prób: 1000
Liczba spójnych sieci: 763
Niezawodność: 76%
```

Listing 6.

W przypadku *Grafu 2.* została dodana tylko jedna dodatkowa krawędź, a prawdopodobieństwo niezawodności sieci wzrosło o połowę. Powodem tego jest to, że w grafie powstał cykl, więc zerwanie tylko jednego połączenia nie powoduje utraty spójności całej sieci.

3.1.3. Wyniki dla grafu o 20 wierzchołkach i 22 krawędziach

```
Liczba prób: 1000
Liczba spójnych sieci: 863
Niezawodność: 86%
```

Listing 7.

W *Grafie 3.* zostały dodane dwie krawędzie o mniejszym prawdopodobieństwie niezawodności niż już istniejące krawędzie. Prawdopodobieństwo niezawodności sieci wzrosło o 10% względem poprzedniego przypadku, ponieważ jest więcej ścieżek pomiędzy dwoma konkretnymi wierzchołkami.

3.1.4. Wyniki dla grafu o 20 wierzchołkach i 26 krawędziach

```
Liczba prób: 1000
Liczba spójnych sieci: 909
Niezawodność: 90%
```

Listing 8.

Dodanie 4 losowych krawędzi o niskim prawdopodobieństwie niezawodności zwiększyło o około 4% prawdopodobieństwo niezawodności sieci względem poprzedniego przypadku.

Wnioski:

Im, więcej krawędzi dodamy do grafu, tym większe jest prawdopodobieństwo że graf pozostanie spójny. Nawet jeżeli do już spójnej sieci będziemy dodawać połączenia o dużym prawdopodobieństwie zerwania, to i tak będzie to skutkowało lepszą niezawodnością tej sieci. Opłaca się mieć jak najwięcej połączeń, ale dobrym rozwiązaniem jest zapewnienie pewnego cyklu między wierzchołkami, który będzie posiadać niskie prawdopodobieństwo zerwania krawędzi.

3.2. Badanie opóźnień przepływu sieci

3.2.1. Wyniki dla proponowanej topologii sieci

a) opóźnienie

Graf 4

```
Średnie opóźnienie: 582
```

Listing 9.

Graf 5

```
Średnie opóźnienie: 456
```

Listing 10.

b) niezawodność

Graf 4

```
Liczba prób: 1000
Liczba sukcesów: 479
Niezawodność: 47%
```

Listing 11.

Graf 5

```
Liczba prób: 1000
Liczba sukcesów: 479
Niezawodność: 57%
```

Listing 12.

Wnioski:

Jak można zauważyć na powyższych przykładach, niezawodność danej sieci maleje im większe wymagania odnośnie szybkości jej działania mamy. Znaczący wpływ ma również prawdopodobieństwo nie uszkodzenia połączeń. Jeżeli zadbamy więc o dobrej jakości połączenia, pamiętając również o ich dobrej przepustowości, oraz nie będziemy mieli przesadnie wygórowanych oczekiwań odnośnie szybkości działania sieci, to jej niezawodność powinna być satysfakcjonująca.

4. Podsumowanie

Konstruując sieć komputerową należy zadbać o połączeniach o jak najmniejszym prawdopodobieństwie uszkodzenia. Na niezawodność pozytywnie wpływają również cykle, jak i duża ilość nadmiarowych połączeń. Ważnymi aspektami są również przepustowości oraz przepływ danych połączeń – jeżeli wartości te będą wysokie to opóźnienie przesyłu danych będzie ledwie zauważalne. Jeżeli powyższe warunki zostaną spełnione, to powinniśmy być w stanie zbudować sieć o wysokiej niezawodności oraz niskim średnim opóźnieniem.