Lista 3

**termin wykonania: 2021-05-23**

**Zadanie 1.**

Zakładamy, że mamy graf wierzchołków, w którym krawędzie są nieskierowane.

Krawędź między wierzchołkami a oznaczamy: .

Listę sąsiadów wierzchołka oznaczamy: .

Podobnie jak w poprzednich zadaniach zakładamy, że w grafie istnieje ścieżka Hamiltona złożona z krawędzi postaci (dzięki czemu graf jest spójny), oraz pewna liczba dodatkowych krawędzi (*skrótów*).

Należy zaimplementować wykonywanie protokołu routingu podobnego do znanego protokołu RIP, zgodnie z poniższymi wskazówkami.

* Każdy wierzchołek zawiera zmienną reprezentującą tzw. *routing table* (oznaczaną przez ), która dla każdego wierzchołka , różnego od , zawiera następujące dane:
  + - wierzchołek ze zbioru (tj. sąsiad ) leżący na najkrótszej, znanej wierzchołkowi , ścieżce od do , oraz
  + - długość tej ścieżki .
* Początkowo każdy wierzchołek zna swoich bezpośrednich sąsiadów i wie o istnieniu krawędzi postaci . Zatem,
  + dla , początkowo i , a
  + dla , oraz
    - , jeśli , albo
    - , jeśli .
* Ponadto, dla każdego, istnieje flaga (początkowo ustawiona na ).
* W każdym wierzchołku działają dwa współbieżne wątki:
  + oraz
* Oba te wątki mają współbieżny dostęp do routing table . W Go można zaimplementować jako *stateful goroutine* a w Adzie jako zmienną *protected*.
* Co pewien czas budzi się i jeśli istnieją jakieś , gdzie , to tworzy pakiet z ofertą, do którego dodaje pary dla wszystkich takich , ustawiając na , a następnie wysyła ten pakiet do każdego swojego sąsiada z .
* Wątek oczekuje na pakiet z ofertą od jakiegoś sąsiada z . Gdy taki pakiet otrzymuje od jakiegoś sąsiada , to dla każdej pary z takiego pakietu:
  + wylicza ,
  + jeśli to ustawia nowe wartości:
    - ,
    - ,
    - ,
* Oba wątki drukują stosowne komunikaty o wysyłanych i otrzymywanych pakietach oraz zmianach w w routing table.

Zwróć uwagę aby nie blokował dostępu do w czasie gdy rozsyła pakiet z ofertami do sąsiadów oraz tak zmieniał na aby nie "zagłuszyć" żadnej nowej zmiany.

Punktacja:

* implementacja w Adzie: 2.5 p.
* implementacja w Go: 3 p.