Lista 3

termin wykonania: 2021-05-23

Zadanie 1.
Zakładamy, że mamy graf n wierzchołków, w którym krawędzie są nieskierowane.
Krawędź między wierzchołkami i a j oznaczamy: $\{i, j\}$.
Listę sąsiadów wierzchołka i oznaczamy: $N(i)$.
Podobnie jak w poprzednich zadaniach zakładamy, że w grafie istnieje ścieżka Hamiltona złożona z krawędzi postaci $\{v,\ v+1\}$ (dzięki czemu graf jest spójny), oraz pewna liczba d dodatkowych krawędzi (*skrótów*).
Należy zaimplementować wykonywanie protokołu routingu podobnego do znanego protokołu RIP, zgodnie z poniższymi wskazówkami.

- Każdy wierzchołek i zawiera zmienną reprezentującą tzw. *routing table* (oznaczaną przez R_i), która dla każdego wierzchołka j, różnego od i, zawiera następujące dane:
  - $R_i[j].nexthop$ - wierzchołek ze zbioru $N(i)$ (tj. sąsiad i) leżący na najkrótszej, znanej wierzchołkowi i, ścieżce p od i do j, oraz
  - $R_i[j].cost$ - długość tej ścieżki p.
- Początkowo każdy wierzchołek i zna swoich bezpośrednich sąsiadów $N(i)$ i wie o istnieniu krawędzi postaci $\{v, v+1\}$. Zatem,
  - dla $j \in N(i)$, początkowo $R_i[j].cost = 1$ i $R_i[j].nexthop = j$, a
  - dla $j \notin N(i)$, $R_i[j].cost = |i-j|$ oraz
    - $R_i[j].nexthop = i+1$, jeśli $i < j$, albo
    - $R_i[j].nexthop = i-1$, jeśli $j < i$.
- Ponadto, dla każdego$R_i[j]$, istnieje flaga $R_i[j].changed$ (początkowo ustawiona na $true$).
- W każdym wierzchołku i działają dwa współbieżne wątki:
  - $Sender_i$ oraz
  - $Receiver_i$
- Oba te wątki mają współbieżny dostęp do routing table R_i. W Go można zaimplementować R_i jako *stateful goroutine* a w Adzie jako zmienną *protected*.
- Co pewien czas $Sender_i$ budzi się i jeśli istnieją jakieś j, gdzie $R_i[j].changed = true$, to tworzy pakiet z ofertą, do którego dodaje pary $(j,\ R_i[j].cost)$ dla wszystkich takich j, ustawiając $R_i[j].changed$na $false$, a następnie wysyła ten pakiet do każdego swojego sąsiada z $N(i)$.
- Wątek $Receiver_i$ oczekuje na pakiet z ofertą od jakiegoś sąsiada z $N(i)$. Gdy taki pakiet otrzymuje od jakiegoś sąsiada l, to dla każdej pary $(j,\ cost_j)$z takiego pakietu:
  - wylicza $newcost_{i,j} = 1 + cost_j$,
  - jeśli $newcost_{i,j} < R_i[j].cost$ to ustawia nowe wartości:

        - $R_i[j].cost = newcost$,
        - $R_i[j].nexthop = l$,
        - $R_i[j].changed = true$,
- Oba wątki drukują stosowne komunikaty o wysyłanych i otrzymywanych pakietach oraz zmianach w w routing table.

Zwróć uwagę aby $Sender_i$ nie blokował dostępu do R_i w czasie gdy rozsyła pakiet z ofertami do sąsiadów oraz tak zmieniał $R_i[j].changed$ na $false$ aby nie "zagłuszyć" żadnej nowej zmiany.

Punktacja:
- implementacja w Adzie: 2.5 p.
- implementacja w Go: 3 p.