# NTIN060 | Homework 03/1

Milan Wikarski (milan@wikarski.sk)

## Zadanie

1. Nejkratší objížďka.(4. př 2. stránka z cvičení.) Pro neorientovaný graf G a hranu e nalezněte nejkratší cyklus obsahující hranu e.

## Riešenie

Uvažujme neorientovaný graf `G = (V, E)`. Ďalej uvažujme nejakú hranu `e = {v0, u0}`. Definujme graf `G' = (V, E \ e)` (graf `G` bez hrany `e`). V grafe `G'` bude najkratšia cesta `P = (v0, ..., u0)` medzi vrcholmi `v0` a `u0` spolu s hranou `e` tvoriť kružnicu `C = (v0, ..., u0, v0)`.

Stačí teda trochu poupraviť algoritmus DFS tak, aby začínal vo vrchole `v0`, ignoroval hranu `e`, a keď nájde hranu, medzi nejakým vrholom a vrcholom `u0`, zastaví sa a vráti hľadanú kružnicu. Algoritmus si bude počas behu držať pole predchodcov `P`, aby sme spätne vedeli nájsť vrcholy tvoriace kružnicu. Okrem toho si bude držať pole `S`, kde `S[v] = 0`, ak vrchol `v` ešte nebol objavený a `S[v] = 1`, ak vrchol `v` už objavený bol.

Ak algoritmus prejde všetky vrcholy, a cestu medzi `v0` a `u0` nenájde, znamená to, že hrana `e` v grafe `G` neležala na kružnici. V takom prípade vráti hodnotu `null`.

```
ALGORITMUS: NejkratsiObjizdka
---------------------------------------------------------------
 INPUT: graf G = (V, E) a hrane e = {v0, u0}
OUTPUT: Postupnosť vrcholov C = (u0, ..., v0, u0)

queue <- new Queue()                      ◁ vytvoríme si prázdnu frontu
for ∀ v ∊ V:
  P[v] <- null
  S[v] <- 0
S[v0] <- 1
queue.enqueue(v0)                         ◁ Začneme vo v0
while (queue is not empty):
  v <- queue.dequeue()                    ◁ Vyberieme v z fronty
  for (∀ u; u is neighbour of v)          ◁ Každý vrchol u spojený hranou s vrcholom v
    if ({v, u} != e and S[u] = 0):        ◁ Ignorujeme hranu e a objavené vrcholy
      S[u] <- 1                           ◁ Označíme vrchol u ako objavený
      P[u] <- v                           ◁ Uložíme predchodcu u
      if (u == u0):
        return C(u, P)                    ◁ Vrátime kružnicu
      else:
        queue.enqueue(u)                  ◁ Pridáme u na koniec fronty
return null                               ◁ e neleží na kružnici
```

### Kružnica z predchodcov

Nasledujúci algoritmus popisuje, ako dostaneme kružnicu z pola predchodcov:

```
ALGORITMUS: KruznicaPredchodci
---------------------------------------------------------------
 INPUT: Pole predchodcov P a vrchol u = u0
OUTPUT: Kružnica C = (u, ..., v0, u)

C = (u)
v <- u
while (P[v] is not null):
```

```
  C <- (C, P[v])                   ⊲ Pridaj v na koniec C
  v <- P[v]
C = (C, u)                         ⊲ Pridaj u na koniec C
return c
```

### Fronta

Neuvádzam tu pseudokód pre implementáciu fronty, ale uvediem jej očakávané správanie. Fronta bude mať 2 funkcie:

- `enqueue(item)` - vloží `item` na koniec fronty.
- `dequeue()` - vyberie prvok zo začiatku fronty a vráti ho

Ďalej predpokladáme, že existuje spôsob, ako overiť, či je front prázdna.

## Dôkaz správnosti

Najkratšia cesta z `v0` do `u0` v grafe `G' = (V, E \ e)` spolu s hranou `e` tvoria najkratšiu kružnicu `C`, ktorej hrana `e` patrí. Zrejme, ak `e` leží na kružnici, tak po odobratí sa graf nemôže rozpadnúť na komponenty súvislosti. Existuje teda nejaká cesta z `v0` do `u0`, ktorá spolu s `e` tvorí kružnicu. Najkratšia takáto cesta musí tvoriť najkratšiu kružnicu.

Cesta, ktorú náš algoritmus nájde z vrchola `v0` do vrchola `u0` v grafe `G'` bude tou najkratšou, pretože sme použili upravený algoritmus BFS.

## Časová a priestorová zložitosť

Označme si `n = |V|` počet vrcholov a `m = |E|` počet hrán.

### Časová zložitosť

Rozdelme si algoritmus na 3 fázy:

#### 1. Inicializácia

Inicializácia prebehne v čase `O(n)`, pretože inicializujeme hodnoty `P[v]` a `S[v]` pre všetky `v ∈ V`.

#### 2. Prehľadávanie grafu

Algoritmus prehľadáva graf `G`, ale ignoruje hranu `e`. Každý vrchol navštívime práve raz, a teda pre každý vrchol preskúmame všetky jeho hrany práve raz. Každý hrana spája práve 2 vrcholy. Spolu to máme `O(n + 2m)`.

#### 3. Budovanie kružnice

Počas budovania kružnice (viz. algoritmus `KruznicaPredchodci`) musíme prechádzať pole predchodcov `P`, dokým nenarazíme na taký vrchol, ktorý predchodcu nemá. To bude zrejme vrchol, v ktorom sme začali, a teda vrchol `v0`. V najhoršom prípade prejdeme všetky vrcholy (to by sa stalo, ak by graf G bol kružnica). V najhrošom prípade teda máme `O(n)` prechodov while cyklom.

Po súčte všetkých troch fáz dostaneme `O(3n + 2m) = O(n + m)`.

### Priestorová zložitosť

Algoritmus pracuje s pomocnými polami `S` a `P`, ktoré majú dĺžku `n`. Okrem toho potrebuje pamäť pre frontu, kde v jeden moment nemôže byť viac, ako `n` prvkov. Spolu teda vyžaduje `O(3n) = O(n)` pamäte.