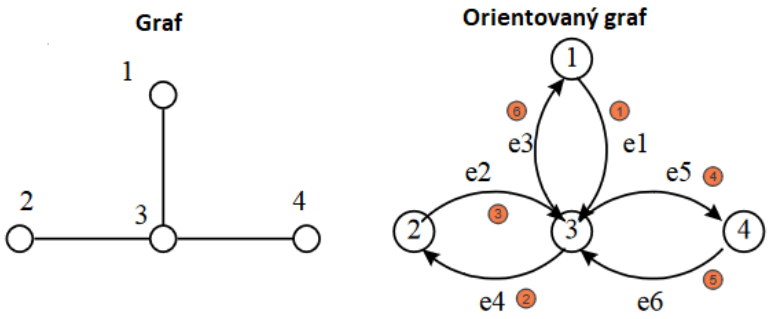
Přiřazení pořadí preorder vrcholům

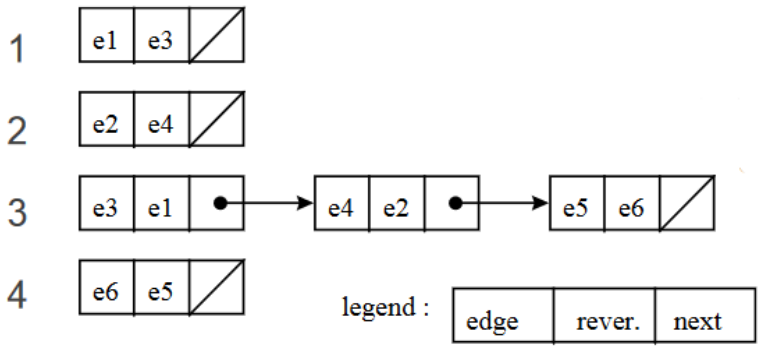
Pro přiřazení pořadí preorder vrcholům je nejprve nutno provést několik přípravných kroků:

1. **Vytvoření Eulerovi cesty**

   Pro vytvoření Eulerovi cesty je třeba nejdříve provést modifikací původního grafu $T = (V, E)$ na orientovaný graf $T' = (V, E')$ tak, že každá hrana (u, v) je nahrazena dvěma orientovanými hranami $< u, v >$ a $< v, u >$.



Z orientovaného grafu lze poté vytvořit seznam sousednosti, ve kterém je pro každý vrchol vytvořen provázaný seznam všech jeho odchozích hran a jejich reverzních hran.



Se seznamem sousednosti lze vytvořit Eulerovu cestu následujícím algoritmem:

```
Algorithm
- constructing Euler tour
Input: adjacency list of T
Output: Array Etour with 2n-2 entries
        Etour(e) is a edge following e

for i = 1 to 2n-2 do in parallel      { e_i = (u, v) }
  if next(e_R) <> nil   then Etour(e) = next(e_R)
    else Etour(e) = AdjList(v)        {first item of adj. list of  vertex v }
  endif
endfor
```

Tento algoritmus má třídu složitosti $O(c)$. Samotná Eulerova cesta potom udává každé hraně jejího následníka. Pro správné fungování vypočtu sumy suffixů je třeba zaslepit. To se provede tak, že hraně vedoucí do kořene přiřadíme jako následníka ji samou.

2. **Vytvoření pole hodnot**

Každé hraně se přiřadí hodnota 1 nebo 0 podle toho, zda se jedná o dopřednou hranu nebo ne. Tento krok má opět složitost $O(c)$.

3. **Vypočtení sumy suffixů nad polem hodnot**

Nad vytvořeným polem hodnot se vypočte suma suffixů. Protože vstupní graf obecně není reprezentován lineárním polem, tak se využije Eulerovi cesty pro získání následníků.

```
Algorithm
V = [v_{n-1}, ... v_1, 0]
for i = 1 to n do in parallel
        if Succ[i] = i then Val[i] = 0 /* neboli neutrální prvek operace ⊕ */
                       else Val[i] = v_i
        for k = 1 to log n do
                Val[i] = Val[i] ⊕ Val[Succ[i]]
                Succ[i] = Succ[Succ[i]]
        end for
        if Val[last] <> 0 then Val[i] = Val[i] ⊕ Val[last]

end for
```

Časová složitost tohoto algoritmu je $O(log\ n)$ a cena je $O(n \cdot log\ n)$.

4. **Provedení korekce sumy suffixů pro preorder**

Výsledná suma suffixů se musí přepočítat na hodnoty, které lze použít na určení preorder pořadí vrcholů. Tento krok má složitost $O(c)$.

```
Algorithm
1) for each e do in parallel
        if e is forward edge  then weight = 1
                else weight = 0
        endif
2) weight = SuffixSums(Etour, Weight)
3) for each e do in parallel
        if e=(u, v) is forward edge then
          preorder(v) = n - weight(e) + 1
        endif
   preorder(root) = 1
```

Časová složitost pro provedení všech těchto kroků je $O(log\ n)$. Cena těchto kroků závisí na implementaci sumy suffixů.

Implementace

Pro n stavů grafu používám $2 * n - 1$ procesů. Procesy 1 až poslední pro provádění výpočtů a proces 0 pro kontrolu výsledků/synchronizaci. Pracovní procesy posílají své výsledky kontrolnímu procesu, který rozesílá tyto výsledky ostatním pracovním procesům. Protože vstupní graf se indexuje od 1, tak v celém programu také používám indexování od 1. Elementy s indexem 0 ignoruji. Pro zapnutí/vypnutí debugovacích výpisů je třeba upravit konstantu DEBUG_ON v hlavičkovém souboru pro.h.

Závěr

Implementováním kontrolního procesu jsem navýšil složitost tohoto algoritmu. Pro přiblížení se k teoretické složitosti by bylo nutné implementovat komunikaci přímo mezi pracovními procesy.