Domácí úkol 2:

Pro (neorientovaný, prostý) graf a dvojici jeho vrcholů u, v na vstupu navrhněte algoritmus, který spočítá počet různých nejkratších cest mezi u a v.

(nejkratších co do počtu hran, délky hran neuvažujeme)

Dvě cesty jsou různé, pokud se liší v alespoň jedné hraně (resp. vrcholu pro prosté grafy). Poznamenejme, že takových cest může být asymptoticky mnohem více než čas, který jsme ochotni na výpočtu strávit.

Plnohodnotné řešení by mělo obsahovat

- Stručný popis metody řešení
- Pseudokód (adekvátně vysokoúrovňový, př: ukázkové BFS,DFS)
- Ošetření speciálních případů
- Argumentace korektnosti
- Analýza časové složitosti (se zdůvodněním)

Popis: Použijeme upravené BFS spuštěné z vrcholu v. U každého vrcholu spočítáme počet cest z v. Počet cest do každého vrcholu určíme jako součet počtů cest do jeho předchůdců v předchozí vrstvě.

pseudokód (plný pseudokód, stačilo ukázat pseudokód vnitřního cyklu)

```
Function BFSCount(graf G, vrcholy u, v)
    stav[*] ← N ; aktVrstva = 0 ; seznam1 ← {v}; seznam2 ← ∅
    pocet[*] ← 0; pocet[v] ← 1
    while w ∈ seznam1 do
        stav[w] ← Z ; vrstva[w] ← aktVrstva
        for x soused w do
            if stav[x] = N then
                přidej x do seznam2
                stav[x] ← O
                vrstva[x] ← aktVrstva + 1
            end
            if vrstva[x] = aktVrstva + 1 then
                pocet[x] + = pocet[w]
            end
        end
        if seznam1 ≠ ∅ then
            ++aktVrstva ; swap seznam1,seznam2
        end
    end
    return pocet[u]
end
```

Speciální případy: Pokud cesta neexistuje (u a v jsou v různých komponentách), BFS se nikdy do u nedostane a vrátí se defaultní hodnota počtu cest 0, což je korektní. Pokud $u = v$, algoritmus správně odpoví 1.

Korektnost: Ukážeme indukcí podle vrstev BFS, že v poli pocet je na konci počet cest z v do příslušného vrcholu. Ve vrstvě 0 je pouze v, pro které je počet cest 1. Uvažme vrchol w v k-té vrstvě. Z I.P. předchůdci ve vrstvě $k-1$ mají správně určené počty. Z vlastností BFS je vzdálenost v a w právě k, a tedy každá nejkratší cesta musí každou hranou překračovat do následující vrstvy (ekv. používá pouze stromové a dopředné hrany), speciálně poslední hrana. Všechny cesty tedy vedou zkrze předchůdce v předchozí vrstvě. Součtem přes počty cest do předchůdců tedy žádnou cestu nevynecháme. Naopak nemůžeme žádnou cestu započítat dvakrát, protože cesty přes různé předchůdce se evidentně liší (a předchůdci sami už počítají cesty korektně). Zbývá uvážit nedosažitelné vrcholy, kde korektnost jistě platí (počet cest zůstane 0).

Složitost: Algoritmus se liší od BFS pouze počítáním hodnot navíc, což je pouze konstatně operací na každý vrchol a hranu. Celková složitost tedy jako BFS $\mathcal{O}(n+m)$.

Nejčastější problémy:

- Počítání cest jejich průchodem. Tento problém měl spoustu forem, princip je v myšlence prohledávat vrcholy opakovaně za každou cestu co do nich vede. To se může zdátrozumné na malých příkladech, kde je pouze pár nejkratších cest, ale v obecnosti může být cest až exponenticiálně v n. Tento postup by tedy v nejhorším případě vyžadoval exponenciální čas i prostor (což je ještě mnohem horší než exp. čas). Problém je ještě závažnější u řešení, která stále tvrdila lineární složitost, dle BFS.

- Určení složitosti. Ve spoustě řešení byla složitost určena jako $\mathcal{O}(n^2)$ počítáním přes počty provedení cyklů. To je sice pravdivý horní odhad, ale není to vhodný odhad. Tento způsob počítání je pouze triviální metoda, téměř vždy potřebujeme alespoň trochu sofistikovanější přístup. Př. BFS,DFS určijí složitost jako konstantně operací pro každý prvek grafu; Bellman-Ford počítá maximální počet relaxací vrcholu a tedy maximální počet zpracování každé hrany; atd. Dodejme ještě, že obecně $m = O(n^2)$ a tedy $n+m = \mathcal{O}(n^2)$, ale velmi často nemáme v grafu asymptoticky $\omega(n^2)$ hran. Závislost na m místo n^2 je vždy mnohem lepší. Třeba pro rovinné grafy (a spoustu dalších praktických tříd) platí $m = \mathcal{O}(n)$.

- Ukončování prohledávání při nalezení u. To není technicky problém jako spíše zbytečná komplikace. Pro praktickou implementaci je to samozřejmě dobrý nápad, pro naše účely to algoritmus nijak nezlepší, pouze zkomplikuje pseudokód.