NTIN060 | Homework 04/8

Milan Wikarski (milan@wikarski.sk)

Zadanie

Príklad 8. Počet všech cest.Pro libovolné dva vrcholy v DAGu zjistěte počet všech cestmezi nimi.

Riešenie a dôkaz správnosti

Majme acyklický orientovaný graf (DAG) `G = (V, E)` a nejaké vrcholy `v0,u0 ∊ V`. Chcem zistiť počet ciest z `v0` do `u0`

Riešenie spočíva v dvoch krokoch. Najprv použijeme algoritmus DFS, pomocou ktorého zistíme topologické usporiadanie vrcholov v grafe `G`. DFS spustíme na vrchole `v0`. Topologické usporiadanie vrcholov je opačné poradiu, v kotrom vrcholy DFS opúšťajú. Toto využijeme tak, že si vrchol uložíme do zásobníka vždy, keď opustí DFS.

Následne budeme počítať počet ciest z vrcholu `v0` do všetkých dosiahnuteľných vrcholov. Označme si `P[w]` počet ciest z vrchola `v0` do vrchola `w`. Na začiatku budeme uvažovať, že existuje práve jedna cesta z `v0` do `v0`, a teda `P[v0] = 1`. Majme vrchol `v` taký, že do neho vedú hrany `e1 = (u1,v), e2 = (u2,v), ... ek = (uk,v)`. Uvažujme (indukčný predpoklad), že už poznáme hodnoty `P[u1], P[u2], ..., P[uk]`. Potom zrejme platí `P[v] = P[u1] + P[u2] + ... + P[uk]`, pretože počet ciest z vrcholu `v0` do vrcholu `v` musí prechádzať cez práve jeden z vrcholov `u1, u2, ... uk`. Dôležité je, že všetky vrcholy `u1, u2, ..., uk` sa nachádzajú v indukčnom usporiadaní pred vrcholom `v`, a teda už poznáme počet ciest, ktorý do nich vedie.

Algoritmus na konci vráti hodnotu `P[u0]`, ktorá obsahuje buď počet ciest z `v0` do `u0` alebo `0`, ak žiadna takáto cesta neexistuje.

Algoritmus

```
ALGORITMUS: DFS
---------------------------------------------------------------
 INPUT: graf G = (V, E) a vrchol v

for (∀ u; u is neighbour of v):      ◅ Každý vrchol u spojený hranou s vrcholom v
  if (D[u] is False):
    D[u] <- True
    DFS(u)
    order.push(u)                    ◅ Keď opúšťame vrchol, vložíme ho do zásobníka
```

```
ALGORITMUS: PocetCiest
---------------------------------------------------------------
 INPUT: graf G = (V, E) a vrcholy v0, u0
OUTPUT: Počet ciest z v0 do u0

order <- new Stack()               ◅ vytvoríme si prázdny zásobník
for ∀ v ∊ V:
  D[v] <- False                    ◅ Objavené vrcholy
  P[v] <- 0                        ◅ Počet ciest z v0 do v
DFS(v0)
order.push(v0)                     ◅ Začneme vo v0
P[v0] <- 1
while (order is not empty):
  v <- order.pop()                 ◅ Vyberieme v zo zásobníka
  for (∀ u; u is neighbour of v)   ◅ Každý vrchol u spojený hranou s vrcholom v
     P[u] <- P[u] + P[v]
return P[u0]
```

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 `D[v]` pre všetky `v ∈ V`.

2. DFS

DFS, počas ktorého zistíme topologické poradie prebehne v čase `O(n + m)`, pretože každý vrchol otvoríme práve raz a každú hranu preskúmame práve raz.

3. Počet ciest

Počet všetkých vrcholov v zásobníku bude najviac `n`. (Môže byť aj menej, ak sú vrcholy, do ktorých sa z `v0` nedá dostať). Každý vrchol zo zásobníka vyberieme práve raz a pozrieme sa na všetky jeho hrany. Dokopy to je najviac `O(n + m)` operácií.

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

Priestorová zložitosť

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