Sada: 3 Příklad: 1 IV003 Algoritmy a datové struktury II

Jméno: Karel Kubíček UČO: 408351

Jméno: Henrich Lauko UČO: 410438

Popis algoritmu: algoritmus podle nápovědy využívá dynamické programování a průchod do hloubky. Průchodu do hloubky je využito v topologickém řazení, algoritmus je popsán ve slidech k IB002 (slidy 9-10, strana 32). Pro naše potřeby otáčíme pořadí výsledného uspořádání, do seznamu uzlů vkládáme uzel od konce a vkládáme při ukončení jeho průzkumu.
Potom ještě před první vrchol přidáme nový počáteční vrchol, abychom neměli více počátečních vrcholů.
Prvkem dynamického programování je memoizace, ke které používáme matici velikosti $|x| \times |y|$ (už prodloužené seznamy vrcholů o počáteční vrchol). V této matici si budeme pamatovat délky nejdelší sekvence, osa x bude připadat topologicky uspořádaným vrcholům x, osa y pro vrcholy y. Na souřadnici $[i, j]$ uchováváme informaci o délce nejdelší společné sekvence pro dvojici i-tého vrcholu z x a j-tého vrcholu z y. Na začátku víme, že nejdelší společná sekvence dvou předřazených vrcholů je 0. Výsledná délka nejdelší sekvence je největší číslo v matici, nalezení sekvence provedeme pomocí backtrackingu do počátečního uzlu. Matici indexuji od 1.
For cykly v kódu tedy iterují přes všechny vrcholy obou grafů, důležité je následně porovnat, zdali vrcholy mají stejný klíč, což nám výrazně rozlišuje chování. V případě různého klíče musíme jen najít maximum v předchůdcích, první volání funkce MAX tedy obsahuje for cyklus, který iteruje přes všechny předchůdce jak vrcholu y (hledáme dvojici právě zpracovávaného i-tého vrcholu z x a předchůdce j-tého vrcholu z y). Obdobně vypadá hledání maxima z dvojic předchůdce $x[i]$ a vrcholu $y[j]$. Pro snazší hledání předchůdců si můžeme třeba vytvořit inverzní graf, což lze provést v lineárním čase vzhledem k součtu počtu vrcholů a hran. Na konci tohoto bloku vybereme větší ze dvou maxim a to přiřadíme na aktuální pozici v matici. Zajímavější je případ, kdy se klíče rovnají. To znamená, že budeme zvyšovat délku sekvence o jedna, ale vůči čemu? Musíme projít všechny dvojice předchůdců, což je až kvadratické množství dvojic (provedeno dvojicí vnořených for cyklů iterujících přes všechny předchůdce). Z těchto dvojic (tedy souřadnic v matici) musíme najít maximum, což zvýšeno o jedna ukládáme na aktuální pozici v matici. Důvodem hledání v kartézském součinu předchůdců je, že prodlužováním sekvence ji měníme, zatímco v případě nerovnosti k sekvenci jen přidáme další vrchol. Výstupem algoritmu je maximální hodnota v matici, kterou najdeme dalším vyhledáním maxima. Alternativně by se dal algoritmus upravit tak, aby maximum vždy bylo v pravém horním rohu matice a to tak, že bychom za všechny konce v topologicky uspořádaném grafy přiřadili nový konečný vrchol, ve kterém by nutně všechny sekvence končili.

Sada: 3 Příklad: 1 IV003 Algoritmy a datové struktury II

Jméno: Karel Kubíček UČO: 408351

Jméno: Henrich Lauko UČO: 410438

Procedura findLongestSequence(x, y)

vstup: x a y jsou acyklické orientované grafy zadané polem indexovaným od 0 a s klíčem *key*
výstup: nejdelší sekvence

```
1  topologicalSort (x) // uspořádáme topologicky x a y od nejnižších k nejvyšším uzlům
2  topologicalSort (y)
3  do grafů x a y vložíme vrchol ϵ, ze kterého vede hrana do všech počátečních vrcholů, tento vrchol do výsledné sekvence nepočítáme, ačkoliv to v pseudokódu není ošetřeno
4  m ← vynulovaná matice rozměrů |x| × |y|
5  for i = 1 to |x| do
6      for j = 1 to |y| do
7          if x[i].key ≠ y[j].key then
               // porovnání i-tého znaku z grafu x a j-tého z grafu y
8              m1 ← maxI (m[i, y[j].predeccessors) // projde m ve sloupci i a vyhledá max mezi předůdci
9              m2 ← maxI (m[x[i].predeccessors, j) // projde m na řádku j a vyhledá max mezi předůdci
10             m[i, j] ← max (m1, m2)
11         else
12             m[i, j] ← maxII (m[x[i].predeccessors, y[j].predeccessors) + 1
                   // projde všechny pozice m které odpovídají indexům předchůdců
13         fi
14     od
15 od
16 return (backtrack v m)
```

Procedura maxI(y, m, i, j)

vstup: Graf y (v prvním případě), pozice i, j a matice memoizace m
výstup: maximum z dvojic vrcholu i a předchůdců j

```
1  maxIndex ← Nil
2  maxValue ← ∞
3  for (v, j) ∈ y do
4      if m[i, v] > maxValue then
5          maxIndex ← v
6          maxValue ← m[i, v]
7  od
8  return (maxValue)
```

Symetricky pro předchůdce v x.

Sada: 3 Příklad: 1 IV003 Algoritmy a datové struktury II

Jméno: Karel Kubíček UČO: 408351

Jméno: Henrich Lauko UČO: 410438

Procedura MAXII(x, y, m, i, j)

vstup: Grafy y a x, pozice i, j a matice memoizace m
výstup: maximum z dvojic předchůdců vrcholu i a j

```
1  maxPosotion ← Nil
2  maxValue ← ∞
3  for (u, i) ∈ x do
4      for (v, j) ∈ y do
5          if m[u, v] > maxValue then
6              maxPosition ← v
7              maxValue ← m[i, v]
8      od
9  od
10 return (maxValue)
```

Korektnost: u korektnosti topologického řazení se odkážu na slidy, ze kterých jsem algoritmus čerpal, kde je korektnost popsána stručně na slidu 32, budu tedy popisovat jen korektnost dalších řádků.
Algoritmus je korektní, pokud na vstupech splňujících vstupní podmínku skončí a zároveň je parciálně korektní.
Konečnost algoritmu je dána použitím for cyklů iterujících přes konečný počet vrcholů.
Parciální korektnost algoritmu provedeme pomocí indukce (na následníky). Pro dvojici předřazených vrcholů grafů, tedy pozici v matici $m[1, 1]$ je nejdelší společná sekvence 0.
Předpokládejme, že algoritmus napočítá korektní délku nejdelší sekvence pro vrcholy na pozicích k z x a l z y. Pak algoritmus platí pro následníky k a l.
Pro všechny dvojice (následník k, l) a (k, následník l) provedeme kontrolu, zdali nejdelší společná sekvence k a l je nejdelší. V případě že je, pak prodloužením zpracované části grafu o jeden vrchol nic nezmění, protože prodlužujeme pouze jeden graf, což nový symbol do sekvence přidat nemůže. Na další pozici tedy uložíme hodnotu $m[i, j]$.
Pro všechny dvojice (následník k, následník l) provedeme kontrolu, zdali nejdelší společná sekvence k a l je nejdelší. V případě že je, pak ještě porovnáme, zdali následníci nemají stejný klíč. Pokud ano, pak se nám sekvence o jedna prodlouží a do nové pozice v matici uložíme hodnotu $m[i, j] + 1$.

Asymptotická časová složitost: složitost TOPOLOGICALSORT je stejně jako u DFS $\mathcal{O}(|V| + |E|)$. Složitost pro vytvoření inverzních grafů pro hledání předchůdce vrcholu také. Ve dvou for cyklech iterujících přes všechny vrcholy máme hledání maxima. To je v prvním případě lineární složitosti vzhledem k počtu vrcholů obou grafů, v druhé však počet vrcholů v grafech nesčítáme, ale násobíme (kartézský součin). Asymptoticky tedy převládá kvadrát v hledání maxima v případě rovnosti klíčů, což při $n \cdot m$ volání ve for cyklech znamená celkovou složitost $\mathcal{O}(|x|^2 \cdot |y|^2) = \mathcal{O}(n^4)$.