Programování pro matematiky
5. cvičení - Datové struktury

Peter Kovács

Doporučuje si promyslet řešení každého z úkolů, pro vaše vlastní ujasnění učiva. Navíc pokud některé z úvah sepíšete můžete získat body

Každý úkol obsahuje za názvem maximální počet bodů, které lze za úkol získat. Vaším úkolem je si vybrat úkoly, které chcete řešiť. Klidně všechny. Úkoly, které jste si vybrali sepište do jednoho souboru a odvzdejte do recodexu. Za celý úkol můžete získat maximálne **2 body**. Po odvezdání ohodnotím každou odevzdanou úlohu. Body sečtu a přidelím vám $max(body, 2)$. Úkoly řešte sami. Pokud spolupracujete v skupině dostanete své body podělené počtem lidí v skupině.

Uzavorkovani - implementace (1 bod):

Do kódu z cvičení doimplementujte kontrolu uzávorkovaní do funkce *skontroluj_ uzavorkovani(zavorky)*. Na vstupu dostanete list s hodnotami. Kde kladné číslo i značí otevření závorky typu i a $-i$ značí uzavření závorky typu i. Při řešení můžete použít předimplmentované kusy kódy v zdrojáku. V zdrojáku jsou za funkcí testy, které odskoušejí zda implementace funguje. Pokud se rozhnodnete řešit tento úkol odevzdejte zdroják do recodexu spolu s řešením jiných úloh. Kód se nevyhodnotí automaticky a budu ho bodovat ručne.

Řešení: Implementaci naleznete v zdrojáku na stránce

Seřazení v grafu (1 bod):

Popište algoritmus, který by dostal dostal na vstup souvislý graf a jeden vrchol z grafu a vypsal všechny vrcholy grafu seřazené dle vzdálenosti od zadaného vrcholu. V každém vrcholu grafu víme zavolat funkci $sousedi(v)$, která vrátí list, všech sousedů vrcholu v v grafu. Graf není ohodnocen a teda délka cesty mezi vrcholy je rovna počtu hran. Pokud jsou 2 vrcholy stejně vzdálené je jedno v jakém pořadí jich vypíšeme. Dále popište jak by sme postupovali pokud by bol graf ohodnocený a zároveň acyklický. Algoritmy popište slovně nebo v pseudokódu.

Řešení: V skratce popíšu myšlenku algoritmu. Na vstupu dostanu vrchol. Ten zařadím do fronty. Teď až kým fornta nezůstane prázdná budu opakovat následovné kroky: Vyberu první prvek z fronty. Vypíšu ho a zavolám na nej funcki sousedi, která mi vráti jeho sousedy. Pak v libovolném pořadí přidám do fronty sousedy,které jsem ještě nenavštívil. Nenavštívené sousedy si musím držet někde bokem, třeba v dictionary, nebo mít list kde na i-tém indexu je 0 pokud nebyl vrchol i dosud navštíven 1 pokud byl. Pro ohodnocený acyklický graf je vhodné použít prioritní frontu, implementovanou například jako haldu. Klíč bude vzdálenost, kde v acyklické grafu je to prostě vzdálenost jeho předchůdce + velikost hrany. V acyklické grafu nám odpadá povinnost kontrolovat, jestli sme prvek navštívili.

Poloha prvků v haldě (0 bod) - není za body ale určite doporučuje si to přemyslet:

1. Kde všude se může nachádzet druhý největší prvek?
2. Kde všude se může nachádzet třetí největší prvek?
3. Kde všude se může nachádzet nejmenší prvek?
4. Kde všude se může nachádzet druhý nejmenší prvek?

Řešení: Ať se jedná o minimovou haldu, tedy na vrchu je vždy minimum. Nejmenší prvek je tedy určitě na vrchu. Druhý nejmenší musí být v druhém najvrchnejším patře. Třetí nejmenší může být v druhém nebo třetím. Nejmenší prvek je na spodním patře, jinými slovy jako list stromu. Druhý největší prvek může být v nejnižším nebo druhém nejnižším patře. Druhá možnost vzniká pokud má pouze jednoho syna. Třetí

může být na posledním nebo předposledním patře.

(ne)porušení haldové podmínky (1 bod):

Podívejme se na co se stane po odstránení prvku z haldy. Navrch vyhodíme poslední prvek z haldy a ten bude bublat dolu (BubbleDown). Na cvičení sme si ukázali, že prvek může porušovat haldovou podmínku při cestě nadol. My tuto podmínku upravujeme tak, že pokud je nějaký ze synů menší jako otec, tak vyměníme otce a menšího z dvou synů. Tím sme urovnali haldovou podmínku pro náš prvek a možná porušili haldovou podmínku o úroveň níže. Ukažte, že syn, který si s otcem vyměnil místo neporušuje haldovou podmínku.

Řešení: Stačí si uvědomit, že když otec zbublal o jedno podlaží vyměnil jse se synem, nad kterým byly půdovně rodiče otce. Před bubláním otce platila podmínka, musí tedy platit i teď.

Prioritní fronta (1 bod):

Prioritní fronta se někdy definuje tak, že prvky se stejnou prioritou vrací v pořadí, v jakém byly do fronty vloženy. Ukažte, jak takovou frontu realizovat pomocí haldy. Dosáhněte časové složitosti O(log n) na operaci.

Řešení: Klíč v haldě budeme reprezentovat jako dvojici tvořenou skutečným klíčem prvku a časovou značkou. Časová značka značí, kolikátý v pořadí byl prvek přidán. Potom porovnání prvků ďeláme tak, že porovnáme klíče a pokud by se rovnali porovnávame časvou značku. S takto předefinovaným porovnáváním můžeme požít klasickou haldu.

Nekonečné vyhledávaní (1 bod):

Popište algoritmus, který v nekonečné posloupnosti $x_1 < x_2 < \dots$ najde pro zadané y pozici i takovou, že $x_i \leq y < x_{i+1}$. Počet kroků hledání by neměl řádově přesáhnout $log_2 i$.

Řešení: Stačí exponencionálně zvětšovat koeficient j. Vždy se podívám jestli $y < x_j$. Pokud ne zdvojnásobím j. Pokud y bylo menší tak sme omezili interval, kde hledáme na $[x_{j/2}, x_j]$. Lehko už ověríme, že nelezení horní meze spotřebuje zhruba $log_2 j$, co je řádově $log_2 i$ a pak už jendom vyhledáme binárně v intervalu $[x_{j/2}, x_j]$, co nemůže přesáhnout $log_2 j$, co je opět řádově $log_2 i$