Nejlevější maximální podstrom s vyhledávací vlastností

Datové struktury

Úloha je zaměřena na procvičení manipulace s binárním stromem. Bude proto nejvýhodnější, když pro reprezentaci zvolíme co nejjednodušší datovou strukturu, abychom s ní mohli přehledně pracovat. Máme dvě zhruba rovnocenné možnosti.

A. Každý uzel bude představovat samostatný objekt, který bude obsahovat své veškeré sledované údaje (klíč, výšku, hloubku atd.) včetně reference na levého a pravého bezprostředního potomka.

B. Pro každý sledovaný údaj (klíč, výšku, hloubku atd.) zřídíme samostatné pole, přičemž uzel sám bude představován pouze indexem těchto polí.

Ukázkové řešení obsahuje obě možnosti, je patrno, že v programu tohoto rozsahu je čitelnost i efektivita obou přístupů celkem srovnatelná.

Kromě uvedené datové struktury nebudeme pro celé zpracování stromu potřebovat žádné další pomocné struktury, veškerou práci zastane rekurze. Zadání říká, že hloubka stromu je nejvýše 30, takže se nemusíme obávat přetečení zásobníku pro nějaké okrajové nebo degenerované případy.

Postup řešení

Nabízí se rozdělit řešení do třech koncepčně oddělených fází: Načtení stromu, výpočet nezbyttných charakteristik jednotlivých uzlů a nalezení hledaného podstromu.

Interní reprezentace podle varianty A nebo B výše je odlišná od formátu zadání a proto v první fázi pouze načteme strom do zvolené datové struktury. V obou ukázkových řešeních k tomu slouží metody `buildTreeFromLineExpr` a `buildTreeFromInPre`, které se ve variantě A a B nepatrně liší díky tomu, že při práci s poli je vhodné dopředu znát velikost dat, abychom před samotným načítáním alokovali pole potřebné velikosti.

Ve druhé fázi vypočteme a do každého uzlu uložíme jeho jednotlivé vlastnosti, které budeme potřebovat ve fázi třetí, kdy jen budeme hledat podstrom s žádanými parametry. Vypočítávání těchto vlastností je jednoduchým cvičením na průchod stromem ve vhodném pořadí. Důležité je určit, jestli daný uzel je kořenem vyhledávacího podstromu. Rekurzivně je tato vlastnost charakterizována: Uzel *u* je kořenem vyhledávacího podstromu, pokud oba jeho bezprostřední potomci jsou kořeny (menšího) vyhledávacího podstromu a pokud navíc maximální klíč v levém podstromu *u* je menší než klíč uzlu *u* a ten je dále menší než minimální klíč v pravém podstromu *u*. Pro jednoduchost je to v ukázkovém řešení zachyceno pomocí tří složek uzlu: Maximální a minimální hodnotu klíče v podstromu, jehož je daný uzel kořenem, a dále také logickou hodnotu toho, zda je tento podstrom vyhledávací. Pro každou z těchto vlastností je připravena rekurzivní metoda, která každému uzlu přislušnou hodnotu přiřadí.

Alternativním postupem, který by navíc šetřil paměť, by bylo napsat funkci, která pomocí vhodného mechanizmu vrací najednou všechny tři potřebné hodnoty pro daný uzel. Zájemcům to ponecháváme jako snadné cvičení, stávající altrnativa je méně elegantní, ale z hlediska kódu snad přehlednější.

Zbývající výpočty hloubky, výšky a velikosti podstromu jednotlivých uzlů jsou snadné a mimo kód je není třeba komentovat.

Poslední fází celého řešení je nalezení požadovaného maximálního podstromu. Každý strom v zadaném formátu obsahuje vždy nějaký nejlevější podstrom s vyhledávací vlastností a tím je přinejmenším nejlevější list stromu. Při hledání dalších větších podstromů proto vyjdeme právě od něj. Nejprve jej separátní metodou `leftmostLeaf` nalezneme a pak jen předáme rekurzivní metodě `findLeftmostMaxBSt`, která začíná svou práci v kořeni a pro každý uzel určí, zda je tento uzel hledaným kořenem nejlevějšího maximálního vyhledávacího stromu. Protože jsou obě metody krátké a přehledné, zájemcům lépe poslouží kód než zdlouhavý popis.