Rozdělení BVS

1 Popis řešení

Vytvoříme rekurzivní funkci, která pro daný strom a danou hodnotu vrátí uspořádanou dvojici rozdělených stromů. Naše funkce začne v kořeni a bude z něj vyhledávat daný prvek. Iterace se pro vrchol v bude skládat ze 3 kroků:

1 Zjistíme, ve kterém podstromu v se nachází daná hodnota. Druhý podstrom obsahuje pouze prvky menší/větší než k, tedy se jím prozatím nemusíme dále zabývat.

2 Spustíme rozdělující funkci rekurzivně na podstrom, ve kterém je hodnota.

3 Když dostaneme výsledek z rekurzivního volání (dvojice (a, b)), nejprve změníme syna v v prohledávaném směru za složku výsledku v opačném směru. (Pokud jsme prohledávali pravý podstrom, nastavíme jako pravého syna v strom a.) Funkce pak vrátí dvojici (v, b), pokud se v rekurzi prohledával pravý podstrom, respektive (a, v), pokud se prohledával levý podstrom.

Rekurze se zastaví, když najdeme vrchol s hledanou hodnotu nebo když prohledávaný vrchol neexistuje (je roven null). V prvním případě vrátíme dvojici (a, b), kdy a je levý podstrom vrcholu a b je vrchol a jeho pravý podstrom. V druhém případě vrátíme dvojici $(null, null)$.

2 Pseudokód

```
<node, node> Split(node root, int k){
   if (root == null) return <null, null>;
   if (root.val == k) return <root.L, root+root.R>;
   if (root.val > k){
      x = Split(root.L, k);
      root.L = x.2;
      return <x.1, root>;
   }
   x = Split(root.R, k);
   root.R = x.1;
   return <root, x.2>;
}
```

3 Důkaz správnosti

Nejprve ověříme, že krajní případy rekurze vrací správné hodnoty. V situaci, kde jsme našli k, je navrácená dvojice stromů triviální.

Pokud jsme funkci zavolali na null, znamená to, že jsme se snažili v předchozím kroku hledat k ve směru, ve kterém v nemělo syna (BÚNO vpravo). Leví potomci v i v jsou tedy menší než k, tudíž po vrácení <null, null> z našeho volání se o řád výš správně vrátí $<v$, null>, jelikož strom vycházející z v nemá žádné prvky $\geq k$. Pro druhou stranu probíhá zdůvodnění analogicky.

Nyní tedy můžeme přejít k indukčnímu kroku. BÚNO root.val $< k$. Zavoláním funkce na pravý podstrom dostaneme dvojici stromů obsahující prvky z tohoto podstromu a odpovídající zadání. Jelikož jsme použili prvky z pravého syna rootu, jsou všechny větší než root, tedy můžeme levý z dvojice vrácených stromů nastavit jako pravého syna a stále dostaneme validní BVS.

Protože navíc víme, že root.val $< k$, víme, že levý podstrom rootu obsahuje také pouze prvky menší než k. Když tedy přidáme levý z dvojice vrácených stromů jako pravého syna, dostaneme strom, ve kterém jsou všechny prvky menší než k.

Když tento strom použijeme jako levý do návratové dvojice a pravý strom zachováme z předchozí iterace, dostaneme dvojici stromů, která obsahuje root a všechny jeho potomky a splňuje zadání. Toto je přesně požadovaná návratová hodnota funkce.

Pro root.val $> k$ zdůvodníme stejně, pouze s opačnými směry. Indukční krok je tedy v pořádku a algoritmus tím pádem validní.

4 Časová složitost

Každá iterace provede krom rekurzivního volání pouze konstantní množství operací. Při rekurzivním volání se vždy dostaneme do o jedna nižší úrovně stromu. Časová složitost je tedy $O(vkastromu)$.

5 Prostorová složitost

Iterace si pamatují pouze konstantní množství dat a iterací je nejvýše n pro úplně nevyvážený strom. Dále si pamatujeme konstantní množství dat ke každému vrcholu. Prostorová složitost algoritmu je tedy $O(n)$.