

Ukázková řešení a komentář k cvičné úloze hloubka stromu

Nejprve uvádíme komentář k autorským řešením, předpokládáme, že se hodí spíše pro ty, kteří si potřebují postup řešení ujasnit, ostatním nabízíme zamyšlení nad kódem na konci tohoto dokumentu.

Kód ukázkových řešení je uložen v příslušných podadresářích, obsahuje řešení všech pěti řešitelů, kteří psali kód Javě a úspěšně během cvičení vyřešili všechny instance. Dále je připojeno na konci tohoto dokumentu řešení v C čitelné i pro Javisty z klávesnice řešitele, který dodatečnou domácí optimalizací dokázal zkrátit kód na minimum při zachování čitelnosti a přitom kód podstatně zrychlil.

Autorské řešení, objektová varianta

Datové struktury

Každý uzel je představován jedním objektem. Uzel obsahuje

- hodnotu klíče,
- hodnotu klíče v levém a pravém potomku (pokud existují, jinak je to nula),
- referenci na levého a pravého potomka a na svého rodiče.

Protože jednotlivé uzly načítáme ze vstupu v náhodném pořadí, nemůžeme strom budovat již během čtení, a musíme proto všechny uzly načíst do vhodné struktury a teprve potom strom vybudovat. Vhodnou a nejjednodušší strukturou je v tomto případě pole uzlů.

Postup řešení je tento:

1. Načteme počet uzlů ve stromu a inicializujeme pole uzlů na tuto délku.
2. Jeden za druhým načteme mechanicky veškeré uzly do pole.
3. Seřadíme načtené pole uzlů podle velikosti jejich klíčů.
4. Projdeme pole od začátku a ke každému uzlu X , který navštívíme, určíme jeho levého a pravého potomka a u těchto potomků zaregistrujeme X coby jejich přechůdce. Určení potomka probíhá tak, že v poli hledáme uzel, který má svůj klíč roven hodnotě klíče levého (resp. pravého) potomka aktuálně navštíveného uzlu X . Díky tomu, že je pole seřazené, můžeme s výhodou využít hledání půlením, které má výrazně nižší časovou složitost než lineární hledání.
5. Jedním průchodem pole najdeme kořen stromu (nemá přechůdce).
6. Kořen stromu předáme jako parametr rekurzivní funkci určující hloubku stromu a vytiskneme výsledek.

Složitost

1. Kupodivu lineární. Pole uzlů se musí v paměti alokovat a inicializovat, to v konstantním čase stihnout nelze :-).
 2. Zřejmě lineární v počtu uzlů.
 3. Pomalé (kvadratické) řazení by zde nic nepomohlo, protože bez použití řazení má krok 4. beztak kvadratickou složitost. Je nutno využít rychlejší řazení se složitostí $\Theta(n \cdot \log n)$, kde n je délka pole.
 4. U každého uzlu určíme jednoho jeho potomka nejdéle v čase úměrném $\log n$, každému uzlu musíme určit dva potomky a uzlů je n , celkem tedy potřebujeme čas nejvýše úměrný $2 \cdot n \cdot \log n$, t.j. $O(n \cdot \log n)$.
 5. Zřejmě lineární v počtu uzlů.
 6. Rekurzivní funkce určující hloubku stromu navštíví každý uzel jen jednou (tělo funkce se pro každý uzel vykoná právě jednou), takže i zde je složitost lineární v počtu uzlů.
- Celkem tedy jsou rozhodující body 4. a 5., z nichž vyplývá celková složitost postupu $\Theta(n \cdot \log n)$, kde n je délka pole.

Autorské řešení, varianta bez objektů

Struktura a postup řešení jsou zcela identické jako ve variantě s objekty. Pouze uzel přestal být jedním objektem, jeho data se „rozptýlila“ do více polí, čímž je dosaženo mírné úspory paměti a zřetelnější úspory času, odpadá veškerá režie vytváření objektů.

Datové struktury

Jednotlivé hodnoty uzlů ukládáme do separátních polí. Uzel sám je reprezentován indexem pole, podle indexu v jednotlivých polích najdeme/uložíme příslušné hodnoty. Máme pole pro:

- hodnotu klíče,
- hodnotu klíče v levém a pravém potomku uzlu (pokud existují, jinak je to nula),
- referenci na levého a pravého potomka uzlu a na rodiče uzlu.

Reference jsou celá čísla, představují indexy do pole klíčů a pole klíčů v levých a pravých potomcích.

Protože jednotlivé uzly načítáme ze vstupu v náhodném pořadí, nemůžeme strom budovat již během čtení, a musíme proto všechny údaje o jednotlivých uzlech načíst do odpovídajících polí a teprve potom strom vybudovat nastavením referencí v dalších polích.

Postup řešení je tento:

1. Načteme počet uzlů ve stromu a inicializujeme všechna pole na tuto délku.
2. Jeden za druhým načteme mechanicky veškeré údaje o jednotlivých uzlech uzly do připravených polí.
3. Seřadíme pole klíčů společně s poli klíčů levých a pravých potomků.
4. Projdeme pole od začátku a ke každému uzlu X, který navštívíme – v tomto případě je uzel určen aktuálním indexem pole -- určíme jeho levého a pravého potomka (což jsou opět pouze indexy pole) a u těchto potomků zaregistrujeme X coby jejich přechůdce. Určení potomka probíhá tak, že v poli klíčů potomků hledáme hodnotu rovnou hodnotě klíče uzlu X, t.j. hodnotě pole klíčů na pozici X. Díky tomu, že je pole seřazené, můžeme s výhodou využít hledání půlením, které má výrazně nižší časovou složitost než lineární hledání.
5. Jedním průchodem pole rodičů najdeme kořen stromu (nemá rodiče).
6. Kořen stromu předáme jako parametr rekurzivní funkci určující hloubku stromu a vytiskneme výsledek.

Odvození složitosti je identické s objektovou variantou.

Rychlé a úsporné posluchačské řešení v C:

Používá pomocné pole hloubek (čtvrtý sloupec 2D pole), které plní postupně během určování hloubky jednotlivých uzlů, přičemž postupuje od listů směrem vzhůru. Pro neznalé: `scanf ("%d",&a);` načte celé číslo na vstupu do proměnné `a`.

```
#include <stdlib.h>
#include <stdio.h>

using namespace std;

int main(int argc, char** argv) {
    int pocet;
    scanf ("%d",&pocet);
    int hloubka = 0;

    int pole[pocet][4];
    int pred[81000];
    for(int i = 0; i<pocet; i++){
        int a;
        scanf ("%d",&pole[i][0]);
        scanf ("%d",&a);
        pole[i][1]=a;
        pred[a]=i+1;
        scanf ("%d",&a);
        pole[i][2]=a;
        pred[a]=i+1;
    }
    for(int i = 0; i<pocet; i++){
        if(pole[i][1]==0 && pole[i][2]==0 ){

            int h=0;
            int akt = i;
            while(pred[pole[akt][0]]!=0){
                h++;
                akt=pred[pole[akt][0]]-1;
                if(h>pole[i][3]){
                    pole[i][3]=h;
                }else{
                    continue;
                }
            }
            if(h>hloubka){hloubka = h;}
        }
    }
    printf ("%d",hloubka);

    return 0;
}
```

Určení asymptotické složitosti tohoto řešení ponecháváme čtenářům jako jednoduché cvičení.