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

Jméno: Karel Kubíček UČO: 408351

Jméno: Henrich Lauko UČO: 410438

Datovou strukturou, která splňuje podmínky stanovené zadáním je B+ strom se stupněm 4. Každý uzel bude mít tyto atributy: $p[x]$ je ukazatel na podstrom x, pp je ukazatel na rodiče a $small[x]$ je klíč odpovídající nejmenší hodnotě v podstromu x.

1. MINIMUM hledáme vždy v nejlevějším podstromu. Rekurzivně se tedy volá minimum na nejlevější podstrom, dokud se nedosáhne úrovně listů, kde nejlevější potomek je minimum našeho stromu.

   Procedura minimum(T)

   vstup: B+ strom T, ve kterém máme minimum hledat
   výstup: ukazatel na minimum
   if T je list **then**
       return T // *nalezeno minimum, vracím ukazatel*
   else
       return `minimum` ($T.p0$) // *firstSubTree je nejlevější podstrom*

   Časová složitost hledání minima je logaritmická, jelikož výška stromu je logaritmická (bude dokázáno níže u insert) a rekurze se v každém zanoření volá na strom s výškou o jedna menší.

2. INSERT(k) se skládá ze 2 částí. První částí je nalezení správného místa, na které prvek patří a zařazení na toto místo, druhou fází je v případě překročení limitu 4 potomci na strom rozdělení uzlu s 5 potomky na 2 uzly, jeden se dvěma, druhý se třema potomky a vypropagování kontroly velkého počtu potomků o úroveň výše.

   Procedura insert(T, k)

   vstup: B+ strom T, do kterého máme přidat klíč k
   výstup:
   $x \leftarrow$ `selectKey` (k) // *určí, do kterého podstromu patří zadaný klíč*   *($k > small[x] \wedge k \leq small[x+1]$)*
   if T je list **then**
       vlož k na místo za $small[x]$ a posuň zbylé klíče
       `controlNode` (T)
   else
       `insert` ($T.p[x], k$)

Procedura controlNode(T)

vstup: B+ strom T, jehož kořenový uzel kontrolujeme
výstup:
if počet klíčů v T je 5 **then**
    rozděl uzel na uzel se 3 a 2 klíči
    vytvoř nový klíč v rodičovském uzlu
    `controlNode` ($T.pp$) // *zkontroluj rodičovský uzel, pokud neexistuje, tak ho vytvoř*
if počet klíčů v T je 1 **then**
    spoj uzel se sousedním uzlem s méně klíči
    odstraň z rodičovského uzlu klíč pro uzel T
    `controlNode` ($T.pp.p[x]$) // *zkontroluj uzel vzniklý spojením*

I insert je logaritmické asymptotické časové složitosti. První část algoritmu udělá maximálně $\log_2(n)$ kroků při hledání správného listu, do kterého má klíč zařadit, následující procedura `controlNode` v případě plné zaplněnosti všech rodičovských uzlů udělá maximálně $\log_4(n)$ (ale pokud by se dělení uzlů vypropagovalo až ke kořeni, pak by i první část musela proběhnout v $\log_4(n)$ krocích). To dohromady patří do $\mathcal{O}(\log(n))$

3. Delete(x)

Procedura delete(T)

vstup: B+ strom T, který máme vymazat
výstup:
odstraň z rodičovského uzlu klíč pro uzel T
`controlNode` ($T.pp$)
uvolni paměť T

Podobná procedura, jako insert, tentokrát nutno kontrolovat, jestli není v rodičovském uzlu klíčů málo (1). To, že operace patří do $\mathcal{O}(\log(n))$

4. Decrease Key(x, k) se dá realizovat posloupností procedur `delete` (x) a následného `insert` (k). To v součtu znamená složitost $2 \cdot \log(n) \in \mathcal{O}(\log(n))$.

5. Extract Minse dá realizovat složením procedur `delete` (`minimum`). To v součtu znamená složitost $2 \cdot \log(n) \in \mathcal{O}(\log(n))$.

U důkazů složitosti jsem vždy předpokládal, že strom bude mít výšku $\log_4(n)$ až $\log_4(n)$. Pokud by tato vlastnost neplatila, pak by složitost operací neodpovídala zadání. Vlastnosti stromu tedy stojí na proceduře `controlNode`.