Dynamický datový typ binární vyhledávací strom

1 Uzel

ADT *binární vyhledávací strom* (dále BVS) patří mezi dynamické datové typy, protože do něj můžeme uložit libovolný, předem neznámý počet hodnot (jsme omezeni jen velikostí dostupné paměti).

Podobně jako u seznamu je i strom tvořen prvky, které jsou vzájemně provázané pomocí ukazatelů. Teď však nepůjde o lineární, ale o stromovou datovou strukturu (strom je speciální typ grafu).

Prvkům teď budeme říkat uzly. Každý uzel teď obsahuje ukazatele na levý a pravý podstrom – proto je *binární* (má dva podstromy).

Každý uzel nese dvojici hodnot, klíč a data. Uzly ukládáme a hledáme podle klíčů, ale zajímají nás data, která jsou s nimi asociována – proto strom využíváme jako *asociativní paměť*.

Všechny uzly stromu musí dodržovat pravidlo, že každý klíč v levém podstromu je menší a každý klíč v pravém podstromu je větší, než klíč aktuálního uzlu. Díky této vlastnosti se v BVS snadno vyhledává – proto je to strom vyhledávací.

V úloze pro toto cvičení budeme používat strom, jehož uzly ponesou dvojici klíč–data typu int–float dvojici klíč–data typu int–float.[1]

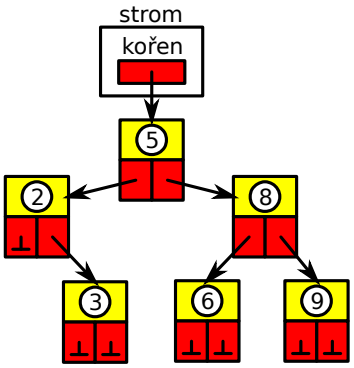
```
typedef struct _uzel Tuzel;

struct _uzel {
  int klic;
  float data;
  Tuzel* levy;
  Tuzel* pravy;
};
```

2 Binární vyhledávací strom

Podobně jako u ostatních ADT budeme i pro strom používat zastřešující datovou strukturu. Ta bude obsahovat ukazatel na počáteční uzel stromu, kterému budeme říkat kořen, a další užitečné informace, jako třeba váhu stromu. Obecně půjde o informace, které jsme schopni snadno aktualizovat při běžných operacích a jejich uchování zde urychlí práci se stromem, protože nebude potřeba samostatný algoritmus pro jejich výpočet.

```
typedef struct {
  Tuzel* koren;
  int vaha;
} Tstrom;
```

Grafické znázornění našeho BVS může vypadat takto:



Tento ukázkový strom má váhu 6, protože obsahuje 6 uzlů. Zároveň je tzv. *vyvážený*, protože pro všechny uzly platí, že váha jejich podstromů se liší maximálně o jedničku. V takovém ideálním případě budeme každý klíč ve stromu vyhledávat nejhůře v logaritmickém čase O(log(n)).

BVS však nemusí být perfektně symetrický. Dokonce se může stát, že při nevhodném pořadí vkládání uzlů bude vypadat jako lineární seznam, protože se vůbec nebude větvit. I když i takový strom může splňovat základní vlastnost BVS, říkáme mu *degenerovaný*. V takovém případě se časová složitost vyhledávání zhorší až na lineární O(n).

2.1 Modul strom.h

Pro toto cvičení už máte připraven modul, který obsahuje částečnou implementaci BVS.

Opět je zde použit koncept obalovacích funkcí, realizujících rozhraní pro uživatele modulu a vnitřních funkcí, které realizují operace

1 Trváte-li na příkladu z reálného světa, mohou být klíči například čísla závodníků a daty jejich výsledky, jako třeba čas, výška, délka atd.

nad stromem. Funkce rozhraní typicky pracují s datovým typem `Tstrom*`. Implementační funkce, které nyní jsou prakticky všechny rekurzivní, musí pracovat s typem `Tuzel*`.

2.2 Inicializace stromu

Inicializace stromu je obdobná, jako u předchozích ADT. V programu budeme pracovat s typem Tstrom*, tedy ukazatelem na strom. Tato pomocná struktura bude během inicializace alokována dynamicky. Je potřeba rozlišovat mezi neinicializovaným stromem (pomocná struktura není alokována) a prázdným stromem (struktura je alokována, ale ukazatel na kořen nese hodnotu NULL).

```
Tstrom* bvsInit(void);
void bvsZrus(Tstrom *strom);
```

2.3 Vkládání dvojice klíč–hodnota

Při vkládání nového uzlu do BVS je potřeba nejprve najít vhodné místo pro jeho vložení. V tomto typu stromu se nový uzel vždy vkládá jako list, tedy úplně na konec. Nový uzel nebude mít žádné podstromy.

V modulu je připravená obalovací funkce `bvsVloz`, která pracuje se stromem jako celkem. Samotný algoritmus vložení nového uzlu pak realizuje rekurzivní funkce `_bvsVloz`. V modulu je již hotová funkce pro alokaci a inicializaci nového uzlu `_novyUzel`, která bude volána z funkce `_bvsVloz`.

```
bool bvsVloz(Tstrom *strom,
  int klic, float data);

bool _bvsVloz(Tuzel **u,
  int klic, float data);

Tuzel* _novyUzel(int klic,
  float data);
```

Funkce budou vracet logickou hodnotu, která slouží pro detekci nepovedené operace. Operace se nemusí povést v případě, že vkládaný klíč se už ve stromu nachází nebo když dojde paměť pro dynamickou alokaci.

Všimněte si, že rekurzivní funkce má jako parametr dvojitý ukazatel na `Tuzel`. Je to proto, že jde o funkci, která modifikuje tvar stromu a musí tedy vracet novou hodnotu ukazatele získanou dynamickou alokací. Jinými slovy, jde zde o parametr typu `Tuzel*`, který je předávaný odkazem (proto druhá *).

Samotný algoritmus vkládání uzlu je rekurzivní. Je potřeba najít místo, kam je možné nový uzel vložit – do levého nebo pravého podstromu podle základního pravidla BVS. Toto vyhledávání je analogií známého algoritmu binárního vyhledávání, jen se teď nepracuje s polem. Díky tomu, že strom je sám o sobě rekurzivní datová struktura, má vkládání nových hodnot průměrně logaritmickou složitost, což je výhoda oproti poli, kde šlo u této operace o složitost lineární.

2.4 Odebírání klíče

Při odebírání je opět potřeba nejprve rekurzivně najít uzel podle zadaného klíče. Teprve když se ukáže, že se takový uzel ve stromu nachází, dojde k jeho odstranění.

Odstranění klíče ze stromu řeší obalovací funkce `bvsOdeber`, která volá rekurzivní funkci `_bvsOdeber`. Ta provede rekurzivní hledání uzlu podle zadaného klíče a teprve v případě, že jej najde, volá funkci `_zrusUzel`.

```
bool bvsOdeber(Tstrom *strom,
  int klic);

bool _bvsOdeber(Tuzel **u, int klic);

void _zrusUzel(Tuzel **uzel);
```

Funkce opět vracejí logickou hodnotu indikující úspěšné nebo neúspěšné odebrání uzlu.

Všimněte si, že rekurzivní funkce opět mají jako parametr dvojité ukazatele. Důvod je stejný jako při vkládání uzlu – operace mění tvar stromu a potřebují vrátit změněný ukazatel. Zde se bude někdy měnit ukazatel na uzel na prázdný ukazatel NULL.

Samotné zrušení uzlu je jednoduché, když má uzel jeden z podstromů prázdný. Pak stačí na jeho místo připojit druhý podstrom a uzel samotný uvolnit.

Problém nastává, když má uzel podstromy dva. Pak pouhé přemostění nejde provést. Řeší se to tak, že se buďto v levém podstromu najde nejpravější, nebo v pravém podstromu nejlevější uzel. Hodnotami takto nově nalezeného uzlu se přepíše původně rušený uzel a tento nově nalezený uzel se zruší nyní již známým

způsobem. Tato finta zajistí dodržení základní vlastnosti BVS.

Proč hledáme nejpravější uzel v levém podstromu (nebo opačně)? Protože tento uzel bude mít v daném podstromu určitě největší hodnotu klíče. Když se takový klíč přesune na místo původně rušeného uzlu, vlastnost BVS nebude porušena. Dále tento uzel určitě nemá žádný pravý podstrom (protože by tam musel být uzel s ještě větším klíčem), takže jej jde snadno zrušit výše popsaným způsobem.

2.5 Tisk stromu

V modulu se ještě nachází funkce bvsTisk, která tiskne strom na obrazovku po jednotlivých úrovních. To je užitečné při ladění stromu, protože si pak jde lépe představit prostorové uspořádání jednotlivých uzlů.

Pomocné algoritmy použité pro realizaci této funkce přesahují hranice toho, co jsme v této kapitole probrali, ale může pro vás být užitečné se s nimi seznámit. Doporučuji si jejich fungování vyzkoušet krokováním pomocí debuggeru.

2.6 Hlavní program s menu

Součástí kódu k tomuto cvičení je i hlavní program realizující nabídkové menu pro testování jednotlivých operací nad implementovaným BVS. V tomto menu je prostor pro přidávání vlastních testovacích operací.

3 Úloha – BVS

Součástí tohoto cvičení je soubor BVStromcviko.zip, který obsahuje částečnou implementaci BVS a obslužný testovací program s nabídkovým menu.

1. Seznam se s rozhraním a implementací BVS a nabídkového menu. Použij k tomu debugger.
2. V modulu chybí implementace funkcí `_bvsVloz` a `_zrusUzel`. Doplň je. Jejich funkčnost ověř pomocí testovacího programu a debuggeru.
3. V souboru `main.c` jsou připraveny funkce `operace1`–`operace5`. V komentářích těchto funkcí je popsáno 5 malých dílčích úkolů. Úkol č. 6 je bonusový. Napiš a otestuj tyto operace. Pro jejich implementaci použij pouze funkce z rozhraní modulu `strom.h`. Pokud myslíš, že ti v modulu nějaká funkce chybí, dopiš si ji.