Obsah

Programátorská dokumentace

Program je určen na vytváření a provádění operací s datovou strukturou AVL strom jako s objektem třídy. AVL strom je datová struktura pro uchovávání údajů a jejich vyhledávání. Pracuje v logaritmicky omezeném čase. Jedná se o samovyvažující se binární vyhledávací strom.

Vlastnosti AVL-stromu:

- V levém podstromu vrcholu jsou pouze vrcholy s menší hodnotou klíče.
- V pravém podstromu vrcholu jsou pouze vrcholy s větší hodnotou klíče.
- Délka nejdelší větve levého a pravého podstromu každého uzlu se liší nejvýše o 1.

Knihovny a hlavičkový soubory:

iostream	hlavičkový soubor s třídami, funkcemi a proměnnými pro organizaci I/O v programovacím jazyce C++.	https://learn.microsoft.com/en-us/cpp/standard-library/iostream?view=msvc-170
stdio.h	hlavičkový soubor standardní knihovny jazyka C, obsahující definice maker, konstanty a deklarace funkcí a typů používaných pro různé standardní vstupní a výstupní operace	https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/stdio.h.html
stdlib.h	hlavičkový soubor standardní knihovny jazyka C, který obsahuje funkce, které se zabývají alokací paměti, řízením provádění program a konverzí typů	https://pubs.opengroup.org/onlinepubs/009695399/basedefs/stdlib.h.html
cmath	hlavičkový soubor <cmath> deklaruje sadu funkcí pro výpočet běžných matematických operací a transformací	https://cplusplus.com/reference/cmath/
Linked_list.h	hlavičkový soubor pro prací s datovou strukturou lineární spojový seznam	

Základní metody:

- Třída AVL_node:

```
class AVL_node {
    int data;
    int height;
    AVL_node* left;
    AVL_node* right;
```

Atributy:

- data: **int**
  číselná hodnota vrchlu
  modifikátor: private
- height: **int**
  výška stromu s kořenem v odpovídajícím vrholu
  modifikátor: private
- left: **AVL_node***
  odkaz na levého syna
  modifikátor: private
- right: **AVL_node***
  odkaz na pravého syna
  modifikátor: private

- Konstruktory:
- Bez parametrů:

```
AVL_node() {
    data = 0;
    height = 1;
    left = NULL;
    right = NULL;
};
```

  modifikátor: public
- S parametry:

```
AVL_node(int d) {
    data = d;
    height = 1;
    left = NULL;
    right = NULL;
};
```

  parametry: d: **int**
  hodnota atributu *data*
  modifikátor: public

`void insert(int a)`

Přidava k AVL stromu nový vrchol s zadanou hodnotou **a**. Nejdřív se program rekurzivně projde stromem a «najde» kam by se měla vložit hodnota **a** (pokud ve stromu ještě není). Nová hodnota se vždy přidává jenom k listu. Pak se vrcholům přiřádí nové vyšky a program «půjde pozpatky», ověřující, jestli jsou odpovídající podstromy sbalancované (délka nejdelší větve levého a pravého podstromu každého uzlu se liší nejvýše o 1). Pokud nějaký podstrom není sbalancovaný, tak se provede balancování (viz **`void balance(int a)`** v pomocných metodách).

parametry: `a`: **`int`**
hodnota, přidáváného vrcholu

modifikátor: public

Příklad:

Vytváření AVL stromu s hodnotami 1, 2, 3

```
int main() {

    AVL_node vrchol(1);
    vrchol.insert(2);
    vrchol.insert(3);

    vrchol.show(); //vypíše do konzole strukturu stromu
}
```

Output:

```
    ,--(3)
    |
(2)
    |
    '--(1)
```

`void del(int a)`

Vypouští z AVL stromu vrchol s zadanou hodnotou **a**. Metoda nejprve rekurzivním průchodem od kořene směrem k listům najde vrchol s hodnotou **a** (nebo její předchůdce). Pokud je ten vrchol listem, vrchol se smaže, provede se přepočet vyšek postromů a směrem pozpatky provede se balancování. Jinak, atribut data tohoto vrcholu se nahradí buď minimální hodnotou pravého podstromu, nebo maximální hodnotou levého. Dále se přepočtou výšky postromů a směrem pozpatky se provede balancování.

parametry: a: `int`
hodnota vypouštěného vrcholu
modifikátor: public

Příklad:

```
int main() {

    AVL_node vrchol(1);
    vrchol.insert(2);
    vrchol.insert(3);

    vrchol.del(2);

    vrchol.show();
}
```

Output:

```
(3)
   |
   '--(1)
```

`int find(int a)`

Rekurzivně prohledává strom. Pokud hodnota byla nelezena, vratí 1 a vypíše do konzole «Hodnota a byla nalezena». Jinak, vratí 0 a vypíše «Hodnota a nebyla nalezena».

parametry: a : int
hledaná hodnota
modifikátor: public
vrátí: int
1 – hodnota byla nalezena
0 – hodnota nebyla nalezena

Příklad:

```
int main() {

    AVL_node vrchol(1);
    vrchol.insert(2);
    vrchol.insert(3);

    int a = vrchol.find(2);
    cout << a << endl;
    vrchol.find(7);
}
```

Output:

```
Hodnota 2 byla nalezena
1
Hodnota 7 nebyla nalezena
```

`int findmin()`

Dokud nedosáhne listu, prujde rekurzivně od kořene k levému následníku. Vratí hodnotu listu s minimální hodnotou.

modifikátor: public

vrátí: int

minimální hodnota stromu

Příklad:

```
int main() {

    AVL_node vrchol(1);
    vrchol.insert(2);
    vrchol.insert(3);

    cout << vrchol.findmin() << endl;
}
```

Output:

```
1
```

`int findmax()`

Dokud nedosáhne listu, prujde rekurzivně od kořene k pravému následníku. Vratí hodnotu listu s maximální hodnotou.

modifikátor: public

vrátí: int

maximální hodnota stromu

Příklad:

```
int main() {

    AVL_node vrchol(1);
    vrchol.insert(2);
    vrchol.insert(3);

    cout << vrchol.findmax() << endl;
}
```

Output:

```
3
```

`int next()`

Je to iterator, který vrací postupně hodnoty od nejmenší k největší. Pokud dojdou hodnoty, vrátí stále největší ale k tomu navíc vypíše do konzole «`Další hodnoty nejsou`». Používá se pomocná globální proměnna *previous*, které během prvního volání next( ) přiřadí hodnota vystupu funkci findmin( ). V dalších voláních se najde předchůdce vrcholu s hodnoutou previous. V přídadě, že hodnota předchůdce je větší než *previous* a není pravý podstrom vrcholu s hodnotou *previous*, tak hodnota předchodce je hledaná další nejmenší hodnota. Pokud má vrchol s hodnotou *previous* pravý podstrom pak hledanou hodnotou je nejmenší hodnota toho podstromu. Obdobně pro případ, že hodnota předchůdce je menší než *previous*.

modifikátor: public

vrátí: int

další nejmenší hodnota

Příklad:

```
int main() {

    AVL_node vrchol(1);
    vrchol.insert(2);
    vrchol.insert(3);

    cout << vrchol.next() << endl;
    cout << vrchol.next() << endl;
    cout << vrchol.next() << endl;
    cout << vrchol.next() << endl;
}
```

Output:

```
1
2
3
Dalsi hodnoty nejsou
3
```

Pomocné metody:

- `void show(bool fromleft = 0, bool prev = 0, linked_list* prev_state = NULL)`

  Rekurzivní funkce, vypisující do konzole strukturu AVL stromu.

  parametry: `fromleft`: `bool`

  proměnna pomocna k určení jestli funkce byla rekurzivně zavolána z levé větví.

  0 - funkce byla zavolána z levé větví

  1 - funkce byla zavolána z pravé větví

  `prev: bool`

  proměnna pomocna k určení parametru fromleft

  `prev_state: linked_list*`

  parametr, ve kterém je uložen předchozí řadek, vitisknutý v konzoli

  modifikátor: public

Příklad:

```
int main() {

    AVL_node vrchol(1);
    vrchol.insert(2);
    vrchol.insert(3);

    vrchol.show();

}
```

Output:

```
    ,--(3)
    |
(2)
    |
    '--(1)
```

`int is_leaf()`

Rozhoduje jestli je AVL strom listem nebo ne.

modifikátor: public

vrátí: int

1 – strom je listem

0 – strom není listem

Příklad:

```
int main() {

    AVL_node vrchol(1);

    cout << vrchol.is_leaf() << endl;

    vrchol.insert(2);
    vrchol.insert(3);

    cout << vrchol.is_leaf() << endl;

}
```

Output:

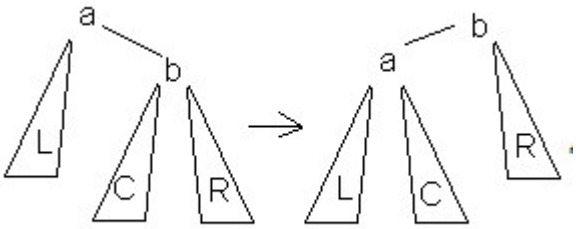
```
1
0
```

`void check_height()`

Projde rekurzivně AVL stromem a spravně ohodnotí atribut height. Nejdřív výškám listů přiřadí 1. Potom půjde pozpatky a přiřadí výškám vrcholů max(vyška levého následníka, vyška pravého následníka).

`void left_rotation()`

Provede levou rotaci AVL stromu (viz. obr.1)

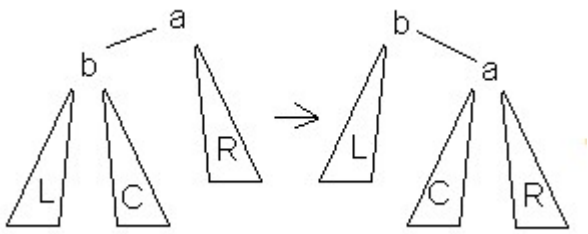


Obr.1

modifikátor: private

- **`void right_rotation()`**

  Provede pravou rotaci AVL stromu (viz. obr.2)

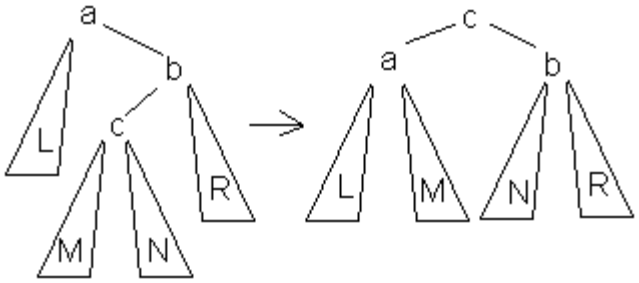


Obr.2

modifikátor: private

- **`void b_left_rotation()`**

  Provede levou velkou rotaci AVL stromu (viz. obr.3)

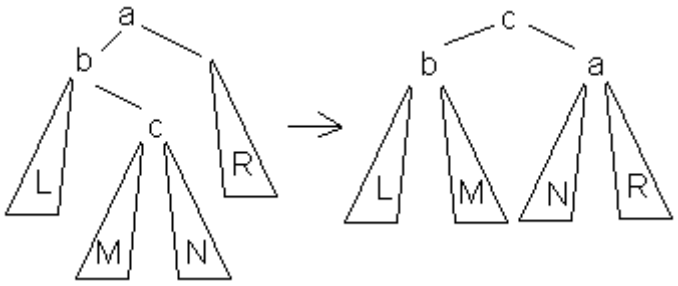


Obr.3

modifikátor: private

- **`void b_right_rotation()`**

  Provede pravou velkou rotaci AVL stromu (viz. obr.4)



Obr.4

modifikátor: private

`void balance()`

V případě, že rozdíl mezi vyškami pravého a levého podstromů je ostře větší, než 1, provede balancování stromu. Balancování se provádí tak, že pokud vyška levého (resp. pravého) podstromu pravého (resp. levého) podstromu je menší nebo rovna vyšce pravého (resp. levého) podstromu pravého (resp. levého) podstromu, provede se malá levá (resp. pravá) rotace stromu. Pokud naopak vyška levého (resp. pravého) podstromu pravého (resp. levého) podstromu je větší, než vyška pravého (resp. levého) podstromu pravého (resp. levého) podstromu, provede se velká levá (resp. pravá) rotace stromu. Po provedení uvedených operací rozdíl mezi vyškami pravého a levého podstromů je menší nebo rovný 1.

modifikátor: private