Níže uvedené úlohy představují přehled otázek, které se vyskytly v tomto nebo v minulých semestrech ve cvičení nebo v minulých semestrech u zkoušky. Mezi otázkami semestrovými a zkouškovými není žádný rozdíl, předpokládáme, že připravený posluchač dokáže zdárně zodpovědět většinu z nich.

Tento dokument je k dispozici ve variantě převážně s řešením a bez řešení.

Je to pracovní dokument a nebyl soustavně redigován, tým ALG neručí za překlepy a jazykové prohřešky, většina odpovědí a řešení je ale pravděpodobně správně :-).

---------- BINARY SEARCH TREE ----------

1.
Je dán BVS s n uzly. Máme za úkol spočítat hodnotu součtu všech klíčů v tomto stromě. Když to uděláme efektivně, bude složitost této operace

a) $\Theta(n \cdot \log(n))$
b) $O(\log(n))$
c) $\Theta(n^2)$
d) $O(1)$
e) $\Theta(n)$

2.
The complexity of the INSERT operation in a balanced BST is (*n* denotes the number of nodes in BST)

a) $\Omega(n)$
b) $O(1)$
c) $\Theta(n)$
d) $O(\log n)$

3.
The complexity of the INSERT operation in a non-balanced BST is (*n* denotes the number of nodes in BST)

a) $\Omega(1)$
b) $O(\log n)$
c) $O(1)$
d) $\Theta(n)$

4.
The depth of a binary tree is 2 (root depth is 0 always). The number of leaves in the tree is:

a) minimum 0 a maximum 2
b) minimum 1 a maximum 3
c) minimum 1 a maximum 4
d) minimum 2 a maximum 4
e) unlimited

5.
Složitost operace Insert ve vyváženém BVS s *n* uzly je vždy

- o O(log(*n*))
- o O(log(hloubka BVS))
- o Θ(*n*)
- o O(1)
- ✓ Θ(log(*n*))
- o Θ(log(hloubka BVS))

6.
Složitost operace Delete v BVS s *n* uzly je vždy

- o O(log(*n*))
- o O(log(hloubka BVS))
- o Θ(*n*)
- ✓ O(*n*)
- o Θ(log(*n*))
- o Θ(log(hloubka BVS))

7.
Podobné otázky jako v předchozí úloze pro případ operací Find, Insert, Delete ve vyvážených nebo nevyvážených stromech jen opakují fakta z přednášky, resp. okamžitý jednoduchý náhled na celou věc.

8.
Binární vyhledávací strom:

a) musí splňovat podmínku haldy *(na tohle přijdeme později)*
b) udržuje v každém uzlu referenci na uzel s nejbližším větším klíčem
c) udržuje v každém uzlu referenci na uzel s nejbližším větším i s nejbližším menším klíčem
d) po každém vložení prvku do BVS musí proběhnout vyvážení stromu
e) po průchodu v pořadí inorder vydá seřazenou posloupnost klíčů

9.
Čísla ze zadané posloupnosti postupně vkládejte do prázdného binárního vyhledávacího stromu (BVS), který nevyvažujte. Jak bude vypadat takto vytvořený BVS?
Poté postupně odstraňte první tři prvky. Jak bude vypadat výsledný BVS?

Posloupnost a)
14 24 5 13 1 3 22 10 19 11

Posloupnost b)
10 16 5 17 4 15 3 1 23 13 2 11

Posloupnost c)
17 4 15 2 5 9 1 12 3 19 16 18

10.
Mějme klíče 1, 2, 3, ..., *n*. Číslo *n* je liché. Nejprve vložíme do BVS všechny sudé klíče v rostoucím pořadí a pak všechny liché klíče také v rostoucím pořadí. Jaká bude hloubka výsledného stromu? Změnil by se nějak tvar stromu, kdybychom lichá čísla vkládali v náhodném pořadí?

11.
Uzel binárního binárního vyhledávacího stromu obsahuje tři složky: Klíč a ukazatele na pravého a levého potomka. Navrhněte rekurzívní funkci (vracející `bool`), která porovná, zda má dvojice stromů stejnou strukturu. Dva BVS považujeme za strukturně stejné, pokud se dají nakreslit tak, že po položení na sebe pozorovateli splývají, bez ohledu na to, jaká obsahují data.

12.
Upravte sami (velmi snadno) předchozí funkci tak, aby zjistila, zda jsou dva BVS shodné, t.j. zda se shodují strukturou i svými daty.

13.
Napište rekurzivní verze operací: TreeMinimum, která vrátí referenci na uzel s nejmenší hodnotou v BVS.

14.
Je dána struktura popisující uzel binárního vyhledávacího stromu takto

```
struct node
{
      valType val;
      node * left, right;
      int count;
}
```

Nebo v Javě takto

```
class Node {
   valType val;    // na tom, co je valType, v tomto případě nesejde
   Node left;
   Node right;
   int count;
}
```

Navrhněte nerekurzívní proceduru, která do složky count v každém uzlu zapíše počet vnitřních uzlů v podstromu, jehož kořenem je tento uzel (včetně tohoto uzlu, pokud sám není listem).

15.
Napište funkci, jejímž vstupem bude ukazatel (=reference) na uzel X v BVS a výstupem ukazatel (=reference) na uzel s nejbližší vyšší hodnotou ve stromu.

16.
Napište funkci, která obrátí pořadí prvků v binárním vyhledávacím stromu. Obrácené pořadí znamená, že po výpisu v pořadí inorder (který neimplementujte!), budou prvky srovnány od největšího k nejmenšímu.

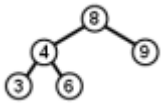
17.
Navrhněte algoritmus, který spojí dva BVS *A* a *B*. Spojení proběhne tak, že všechny uzly z *B* budou přesunuty do *A*, přičemž se nebudou vytvářet žádné nové uzly ani se nebudou žádné uzly mazat. Přesun proběhne jen manipulací s ukazateli. Předpokládejte, že v každém uzlu v *A* i v *B* je k dispozici ukazatel na rodičovský uzel.

18.
Napište funkci, která obrátí pořadí prvků v binárním vyhledávacím stromu. Obrácené pořadí znamená, že po výpisu v pořadí inorder (který neimplementujte!), budou prvky srovnány od největšího k nejmenšímu.

---------- AVL TREE ----------

1.
Do AVL stromu na obrázku vložíme klíč s hodnotou 5. Přitom musíme provést

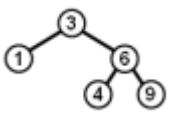
a) jednu R rotaci
b) jednu L rotaci
c) jednu LR rotaci
d) jednu RL rotaci
e) žádnou rotaci



2.

Do AVL stromu na obrázku vložíme klíč s hodnotou 7. Přitom musíme provést

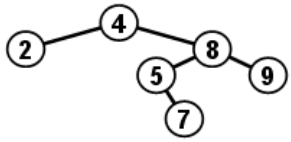
a) jednu R rotaci
b) jednu L rotaci



c) jednu LR rotaci
d) jednu RL rotaci
e) žádnou rotaci

3.

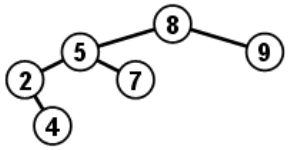
Daný AVL strom není vyvážený. Po provedení správné rotace se změní tvar stromu a potom bude pravý podstrom kořene obsahovat uzly s klíči:



a) 4 5 7 8 9
b) 5 7 8 9
c) 7 8 9
d) 8 9
e) 9

4.

Daný AVL strom není vyvážený. Po provedení správné rotace se změní tvar stromu a potom bude levý podstrom kořene obsahovat uzly s klíči:



a) žádný uzel
b) 2
c) 2 4
d) 2 4 5
e) 2 4 5 6

5.

V AVL stromu se algoritmus pro vyvažování musí rozhodnout, kterou z rotací pro vyvážení stromu použije po vložení uzlu E. Má k dispozici pouze hodnoty rozvážení ve všech uzlech, tj. vnitřních uzlech i listech. Musí se rozhodnout správně.

a) Použije levou rotaci v uzlu B, protože je nalevo od F a hodnota rozvážení je -1
b) Použije levou rotaci v uzlu B následovanou pravou rotací v uzlu F, protože rozvážení je v uzlu F rovno hodnotě 2 a rozvážení v B je záporné
c) Použije pravou rotaci v kořeni F, protože rozvážení je v tomto uzlu rovno hodnotě 2
d) Vůbec rotovat nemusí, protože strom vyhovuje kritériím AVL stromu

6.

V AVL stromu se algoritmus pro vyvažování musí rozhodnout, kterou z rotací pro vyvážení stromu použije po vložení uzlu C. Má k dispozici pouze hodnoty rozvážení ve všech uzlech, tj. vnitřních uzlech i listech. Musí se rozhodnout správně.

a) Použije pravou rotaci v kořeni F, protože rozvážení je v tomto uzlu rovno hodnotě 2
b) Použije levou rotaci v uzlu B, protože je nalevo od F a hodnota rozvážení je -1
c) Vůbec rotovat nemusí, protože strom vyhovuje kritériím AVL stromu
d) Použije LR rotaci v uzlu F, protože rozvážení je v tomto uzlu rovno hodnotě 2 a rozvážení v B je záporné.

7.

Jednoduchá levá rotace v uzlu *u* má operační složitost

a) závislou na výšce levého podstromu uzlu *u*
b) konstantní
c) mezi $O(1)$ a $\Omega(n)$
d) závislou na hloubce uzlu *u*

8.

Jednoduchá pravá rotace v uzlu *u* má operační složitost

a) $\Theta(1)$
b) závislou na hloubce uzlu *u*
c) mezi $O(1)$ a $\Omega(\log n)$
d) $\Omega(n)$

9.
Rotace ve vyváženém binárním vyhledávacím stromě

a) se provádí vždy, když je výška jednoho z podstromů váhově vyváženého stromu alespoň o dvě vyšší než ve druhém podstromu
b) se provádí při každém vkládání nového uzlu
c) je invariantní vůči procházení stromem v pořadí *inorder*
d) se provádí pouze při vkládání levého syna v pravém podstromu a pravého syna v levém podstromu

10.
Left rotation in node *u* (in the sub-tree with the root node *u*)

a) moves the right sun u_R of the root node *u* to the root. Node *u* becomes the left son of the node u_R and the left sub-tree of u_R becomes the right sub-tree of the node *u*.
b) moves the left sun u_L of the root node *u* to the root. Node *u* becomes the right son of the node u_L and the left sub-tree of u_L becomes the right sub-tree of the node *u*.
c) moves the right sun u_R of the root node *u* to the root. Node *u* becomes the left son of the node u_R and the right sub-tree of u_R becomes the left sub-tree of the node *u*.
d) moves the left sun u_L of the root node *u* to the root. Node *u* becomes the right son of the node u_L and the right sub-tree of u_L becomes the left sub-tree of the node *u*.

11.
Right rotation in node *u* (in the sub-tree with the root node *u*)

a) moves the right sun u_R of the root node *u* to the root. Node *u* becomes the left son of the node u_R and the left sub-tree of u_R becomes the right sub-tree of the node *u*.
b) moves the left sun u_L of the root node *u* to the root. Node *u* becomes the right son of the node u_L and the left sub-tree of u_L becomes the right sub-tree of the node *u*.
c) moves the right sun u_R of the root node *u* to the root. Node *u* becomes the left son of the node u_R and the right sub-tree of u_R becomes the left sub-tree of the node *u*.
d) moves the left sun u_L of the root node *u* to the root. Node *u* becomes the right son of the node u_L and the right sub-tree of u_L becomes the left sub-tree of the node *u*.

12.
The keys 40 20 10 30 were inserted one after another into an originally empty AVL tree. The process resulted in

a) one R rotation
b) one L rotation
c) one RL rotation
d) one LR rotation
e) no rotation at all

13.
The keys 10 30 20 40 were inserted one after another into an originally empty AVL tree. The process resulted in

f) one R rotation
g) one L rotation
h) one RL rotation

i) one LR rotation
j) no rotation at all

14.
Simple right rotation in node *u*

a) decreases the depth of the right son of *u*
b) swaps the contents of node *u* with the contents of its left son
c) decreases the depth of the left son of *u*
d) moves the inner node left from the right son of *u* to the root

15.
Simple left rotation in node *u*

a) decreases the depth of the left son of *u*
b) swaps the contents of node *u* with the contents of its left son
c) moves the inner node left from the right son of *u* to the root
d) decreases the depth of the right son of *u*

16.
Jednoduchá levá rotace v uzlu *u* má operační složitost
a) závislou na výšce levého podstromu uzlu *u*
b) mezi O(1) a $\Theta(n)$
c) závislou na hloubce uzlu *u*
d) konstantní

17a.
LR rotace v uzlu u lze rozložit na
a) levou rotaci v pravém synovi uzlu u následovanou pravou rotací v uzlu u
b) pravou rotaci v pravém synovi uzlu u následovanou levou rotací v uzlu u
c) levou rotaci v levém synovi uzlu u následovanou pravou rotací v uzlu u
d) pravou rotaci v levém synovi uzlu u následovanou levou rotací v uzlu u

17b.
RL rotace v uzlu u lze rozložit na
a) levou rotaci v pravém synovi uzlu u následovanou pravou rotací v uzlu u
b) pravou rotaci v pravém synovi uzlu u následovanou levou rotací v uzlu u
c) levou rotaci v levém synovi uzlu u následovanou pravou rotací v uzlu u
d) pravou rotaci v levém synovi uzlu u následovanou levou rotací v uzlu u

18.
Kolik jednoduchých rotací se maximálně provede při jedné operaci Insert v AVL stromu s *n* uzly?
a) jedna
b) dvě
c) log(*n*)
d) *n*

19.
LR rotace v uzlu *u* má operační složitost
a) závislou na hloubce uzlu *u*
b) Θ (log *n*)
c) konstantní
d) lineární

20a.
Levá rotace v uzlu *u*

a) v podstromu s kořenem *u* přemístí pravého syna u_p uzlu *u* do kořene. Přitom se uzel *u* stane levým synem uzlu u_p a levý podstrom uzlu u_p se stane pravým podstromem uzlu *u*
b) v podstromu s kořenem *u* přemístí levého syna u_l uzlu *u* do kořene. Přitom se uzel *u* stane pravým synem uzlu u_l a levý podstrom uzlu u_l se stane pravým podstromem uzlu *u*
c) v podstromu s kořenem *u* přemístí pravého syna u_p uzlu *u* do kořene. Přitom se uzel *u* stane levým synem uzlu u_p a pravý podstrom uzlu u_p se stane levým podstromem uzlu *u*
d) v podstromu s kořenem *u* přemístí levého syna u_l uzlu *u* do kořene. Přitom se uzel *u* stane pravým synem uzlu u_l a pravý podstrom uzlu u_l se stane levým podstromem uzlu *u*

20b.
Pravá rotace v uzlu *u*
a) v podstromu s kořenem *u* přemístí pravého syna u_p uzlu *u* do kořene. Přitom se uzel *u* stane levým synem uzlu u_p a levý podstrom uzlu u_p se stane pravým podstromem uzlu *u*
b) v podstromu s kořenem *u* přemístí levého syna u_l uzlu *u* do kořene. Přitom se uzel *u* stane pravým synem uzlu u_l a levý podstrom uzlu u_l se stane pravým podstromem uzlu *u*
c) v podstromu s kořenem *u* přemístí pravého syna u_p uzlu *u* do kořene. Přitom se uzel *u* stane levým synem uzlu u_p a pravý podstrom uzlu u_p se stane levým podstromem uzlu *u*
d) v podstromu s kořenem *u* přemístí levého syna u_l uzlu *u* do kořene. Přitom se uzel *u* stane pravým synem uzlu u_l a pravý podstrom uzlu u_l se stane levým podstromem uzlu *u*

21.
Vyvážení BVS některou z operací rotace
a) je nutno provést po každé operaci Insert
b) je nutno provést po každé operaci Delete
c) je nutno provést po každé operaci Insert i Delete
d) snižuje hloubku stromu
e) není pro udržování BVS nezbytné

22.
Napište proceduru *strom leváRotace(strom s).* Napřed si namalujte obrázek a rozmyslete si, kterých ukazatelů se změna dotkne.
Dtto pro LR rotaci. Domyslete, kde chybí testy typu if(root==null) return null;

---------- **B-TREE** ----------

1.

B-strom je řádu k, pokud každý jeho uzel, kromě kořene, musí obsahovat alespoň k klíčů a zároveň může obsahovat nejvýše 2k klíčů.
Vybudujte B-strom řádu 1 tak, že do prázdného stromu vložíte v uvedeném pořadí klíče
25, 13, 37, 32, 40, 20, 22.
Dále tento strom zrušte, a to tak, že jednotlivé klíče klíče odstraníte v pořadí
13, 25, 40, 22, 20, 37, 32.
Nakreslete strom po každé operaci Insert a Delete.

2.

Vybudujte B-strom řádu 1 tak, že do prázdného stromu vložíte v uvedeném pořadí klíče
32, 18, 31, 59, 20, 23, 24, 36, 60, 58, 15, 57, 51, 17, 16, 26, 42, 21, 43, 12.
Dále tento strom zrušte, a to tak, že jednotlivé klíče klíče odstraníte v pořadí
23, 31, 26, 15, 24, 42, 17, 36, 20, 43, 16, 32, 18, 59, 21, 51, 60, 12, 58, 57.
Nakreslete strom po každé operaci Insert a Delete.

3.

B-strom je řádu 5 a máme do něj umístit 1 000 000 klíčů. Jaký je maximální a minimální možný počet uzlů tohoto stromu? Jaká je maximální a minimální možná hloubka tohoto stromu?

Řešení Pokud jsou všechny uzly maximálně naplněny klíči, má každý uzel 10 klíčů a 11 potomků.
Počet klíčů v jednotlivých patrech je pak 10, 11*10, 11*11*10, 11*11*11*10 atd. Když má tento strom hloubku *h*, jeho počet klíčů je roven 10 * (1 + 11 +11*11 + ... + 11^*h*) = 10 * (11^(*h*+1) −1) /(11 −1) =
11^(*h*+1) −1.
Nejmenší *h*, pro které platí 11^(*h*+1) −1 ≥ 1000000, je *h*=5, strom tedy bude mít 6 pater včetně kořene.
Pokud by měly být všechny uzly minimálně zaplněny, tj. počet uzlů by se maximalizoval, pak zopakujeme předchozí výpočet s jinými parametry, jmenovitě:
Počet klíčů v jednotlivých patrech je pak 1, 6*5, 6*6*5, 6*6*6*5 atd. Když má tento strom hloubku *h*, jeho počet klíčů je roven 5 * (1 + 6 +6*6 + ... + 6^*h*) − 4 = 5 * (6^(*h*+1) −1) /(6 −1) − 4 =
6^(*h*+1) −5.
Nejmenší *h*, pro které platí 6^(*h*+1) −5 ≥ 1000000, je *h*=7, strom tedy bude mít 8 pater včetně kořene.

4.

Řešte předchozí úlohu pro obecnou hodnotu řádu B-stromu *k*. a pro daný počet klíčů *n*.