2 body

Kód studenta 30

1 Automaty (3 body)

1. Definujte (generativní) gramatiku, jazyk generovaný gramatikou a Chomského hierarchii gramatik.
2. Zkonstruujte gramatiku generující jazyk

$$L = \{uvu^R \mid u \in \{a,b\}^*, v \in \{a\}^*\},$$

kde u^R označuje zrcadlový obraz slova u.

3. Zařaďte jazyk L do Chomského hierarchie, tj. určete nejmenší třídu Chomského hierarchie, do které jazyk náleží, a dokažte, že žádná gramatika ze slabší třídy Chomského hierarchie nemůže generovat L.

① gramatika je čtveřice (V_N, V_T, S, P); kde
V_N je množina neterminálních symbolů
V_T je množina terminálních symbolů
S je startovní neterminál
P je soubor tzv. „přepisovacích pravidel" ~~tvar~~
v tvaru $\alpha \to \beta$ kde $\alpha, \beta \in V_N^* \cup V_T^*$
$\forall \alpha$ obsahuje aspoň 1 neterminál

jazyk generovaný gramatikou je jazyk, pro jehož slova platí, že jsou tvořena znaky z V_T a pokud začínu v S, jsem jej je schopná nějakou posloupností přepisovacích pravidel z P získat. $\forall$ eko se přepisuje podle pravidla

jazyky dělíme na čtyři skupiny $L_3 \subset L_2 \subset L_1 \subset L_0$,
kde L_0 .... rekurzivně spočetné jazyky — ~~přijímá~~ bez omezení pravidel gramatika, které je přijímají
L_1 .... kontextové (nezkracující) jazyky — ~~pravidla ve~~ tv. levá strana pravidla musí být kratší ($|\alpha| \leq |\beta|$) nebo stejně dlouhá
L_2 .... bezkontext. jazyky
– levá strana může obsahovat jen 1 neterminál, pravá bez omezení
L_3 .... ~~lineární~~ (regulární) jazyky
– ~~pravá strana~~ pravidla jen tvaru $X \to z$
nebo $X \to wZ$
kde $X, Z \in V_N^* \cup \{\varepsilon\}$ a $w \in V_T^*$

$\frac{2}{3}$ bodu

$X \in V_N$ $X \to wZ$ $Z \in V_N^*$
$X \to w$ $w \in V_T^*$

vedá se vygenerovat prázdné slovo
$\alpha A \beta \to \alpha\gamma\beta$
$\alpha, \beta \in (V_N \cup V_T)^*$
$\gamma \in (V_N \cup V_T)^+$
pravidla $S \to \lambda$
ale pokud S nesmí být na levé straně

② $L = \{ uvu^R \mid u \in \{a,b\}^*, v \in \{a\}^* \}$

příklady slov: baaab, aaaaaaa, abaaaaaba

$u\ v\ u^R$

— musím umět „najednou" „prostředek a věty" a kolem nich „symetricky zároveň" přidávat buď a/b, nebo běžná

$S \to ASA \mid BSB \mid P$

$A \to a$

$B \to b$

$P \to Pa \mid a \mid \lambda$

negeneruje $\lambda \in L$ — 1 bod

③ — jazyk je zjevně bezkontextový, ale je regulární?

Dle pumping lemmatu, kdyby byl, tak $\exists n$: $\forall z \in L, |z| \geq n$ jdou napsat jako $z = uvw$, $|uv| \leq n$, $|v| \geq 1$ a $\forall i \geq 0$ všechny slova tvaru uv^iw taky patří do L. Kdyby to neplatilo, regulární není. Jenže to tady zřejmě zjevně funguje, co dál? NIE VŽDY!

Dle Myhill-Nerodovy věty je regulární jazyk V konečným počtem tříd, kongruence nad ~~daným abecedou~~ X^*.

X... abeceda
X^*... množina slov

Kongruence ~ je ekvivalence nad X^*, že $u \sim v \Rightarrow uw \sim vw \ \forall u,v,w \in X^*$.

Dále mějme tady ~~kongruenci třídu~~ nějaká dvě slova, ~~která~~ u, v, že $u \not\sim v$ (jazyk je to zjevně nekonečný, tříd je omezeně — podle holubníkového principu musí existovat).

Uvažme slovo $w = bab$, které zjevně leží v L.

Má však platit $u \sim v \Rightarrow uw \sim vw$, jenže to je spor, protože $uw, vw \notin L$ (bab na konci rozbíjí tu symetrii), takže jazyk není regulární. toto nie je spor, ak obidve uw a vw nepatria do L — 1/3 bodu

1

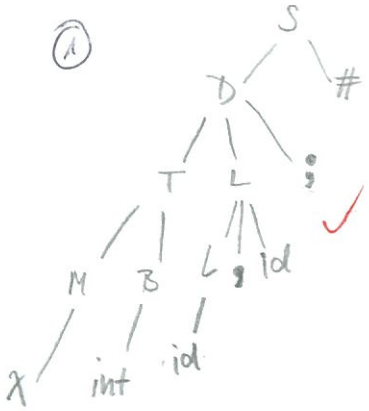
Kód studenta 30

6 Překladače (3 body)

Gramatika má počáteční neterminál S a terminály **id**, ",", ";", **long**, **int**, **double** a #. Terminál **id** představuje libovolný identifikátor (různý od terminálů - klíčových slov) tvořený sekvencí písmen a číslic začínající písmenem:

$$
\begin{aligned}
S &\rightarrow D\ \# \\
D &\rightarrow T\ L\ \mathbf{;} \\
T &\rightarrow M\ B \\
M &\rightarrow \lambda \\
M &\rightarrow \mathbf{long} \\
B &\rightarrow \mathbf{int} \\
B &\rightarrow \mathbf{double} \\
L &\rightarrow \mathbf{id} \\
L &\rightarrow L\ \mathbf{,}\ \mathbf{id}
\end{aligned}
$$

1. Nakreslete derivační strom pro řetězec **int i, j;** #.
2. Určete množiny $FIRST(A)$ a $FOLLOW(A)$ pro $A \in \{D, T, M, B, L\}$.
3. Určete, zda je gramatika LL(1), a stručně vysvětlete proč.



✓

2 FIRST(D) AT

„A"	FIRST	FOLLOW
D	T	Lj
T	M	B
M	—	λ, long
B	—	int double
L	L	L, id

✗ FIRST?

FIRST?

3 nevím ✗

1-

Kód studenta 30

5 Sítě (3 body)

1. Stručně vysvětlete význam protokolu IP z pohledu vrstevnaté architektury. Jaké funkce nabízí vrstvám nad sebou a co požaduje od vrstev pod sebou?
2. Protokol IPv4 v sobě nativně podporuje fragmentaci datagramů. Vysvětlete, kdy k fragmentaci může dojít a jak je možné jí předcházet.
3. Uvažme použití protokolu IP nad Ethernetovou linkou (1000BASE-T). Vysvětlete, jak v tomto případě síťový uzel přeloží IPv4 a IPv6 adresy na adresy linkové vrstvy (MAC) pro lokální doručení v rámci dané sítě.

Mix poloprav. vyjadřování z contextu.

(1) Protokol IP pokrývá síťovou vrstvu, tedy stará se o komunikaci mezi sítěmi a adresací jejich uzlů. Musí umět „zabalit" ~~data~~ do IP paketu data z linkové a fyzické vrstvy a ty nabízet ke komunikaci transportní vrstvě (nad kterou pak je ještě aplikační. Nedává smysl.

Díky IP / (díky kajemné komunikaci mezi sítěmi) IP nemusíme navázat spojení / mezi dvěma koncovými uzly a dál skrze aplikační vrstvu zařídit konkrétní službu (např. já ze své IP adresy pošlu HTTP GET požadavek na nějakou adresu, -nějaký server, a ten mi pošle zpět na moji adresu HTTP odpověď.) např. TCP

(2) Technické možnosti každé sítě se liší, v každé síti je možno přenášet v linkové vrstvě jinak velké rámce. Routery mezi sítěmi tedy na síťové vrstvě musí případně datagramy zmenšovat resp. rozdělovat ~~do menších rámců~~, aby se daly linkovou vrstvou přenášet (aby se vešly). ✓ 1-

Dá se tomu předejít, že zjistím MTU po cestě, - jak? / nejmenší „průchodnost" dílčích cest po cestě do cíle a u sebe rovnou udělám takhle malé IP datagramy. (Trochu jako Sedyx zjišťuji min. rezervu v síti v teorii grafů)

směrování s tím nesouvisí.

③ pomocí směrovacích záznamů — OS si v nastavení
síťě pamatuje, na jakém portu (MAC adrese) má
jakou IP adresu, pokud ji nemá přímo, přesměruje
požadavek na stroj „hierarchicky" výše (brána gateway).

port je TCP/UDP věc.

to je věc směrování, ne překlad IP → MAC.

Chybí IPv6

3 body

Kód studenta 30

4 Zámky (3 body)

Pro řešení otázky je dostačující následující zjednodušená představa o konceptech jazyka C# (pokud ale skutečné chování využitých konceptů jazyka C# znáte přesněji, tak to není na závadu):

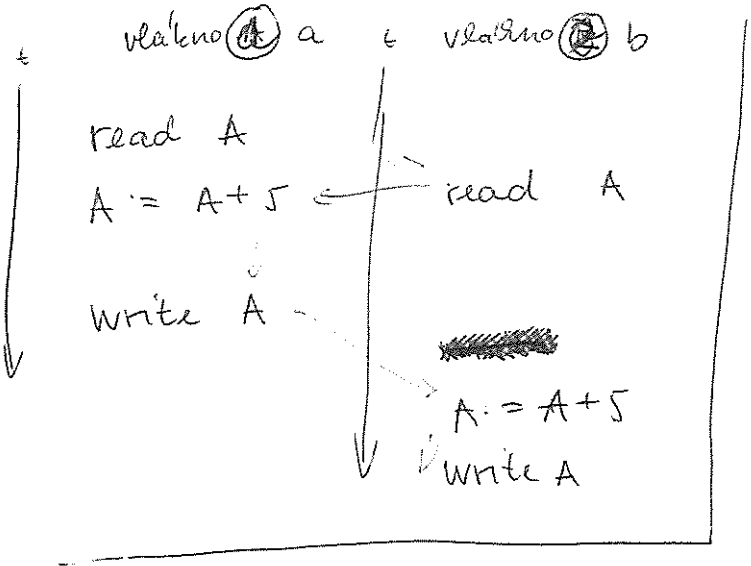
Každá instance třídy `object` v sobě obsahuje právě jeden unikátní zámek. Tento zámek je možno zamknout pomocí operace `lock` (viz řádek 13 kódu níže) – jedná se o běžnou implementaci zámků s podporou pro rekurzivní zamykání a s pasivním čekáním na uvolnění zámku. Otevírací složená závorka za příkazem `lock` odpovídá žádosti o zamčení zámku (řádek 13), párová zavírací složená závorka tohoto bloku pak odpovídá operaci odemčení zámku (řádek 18).

Metoda `Parallel.For(int fromInclusive, int toExclusive, Action<int> body)` vytvoří nová vlákna, jejichž počet odpovídá počtu logických procesorů na cílovém systému. Tato vlákna poté volají metodu předanou jako parametr `body` metody `Parallel.For` (v uvedeném příkladu níže tedy metodu `ProcessFilename`) tak, že pro každé `i` z rozsahu <`fromInclusive`, `toExclusive`) je metoda `body` zavolána právě jednou. Nicméně není definované, ve kterém z vyrobených vláken bude metoda `body` pro konkrétní `i` zavolána, stejně tak není definované pořadí, v jakém budou jednotlivé hodnoty `i` zvoleny. Návrat do volajícího vlákna z metody `Parallel.For` je bezpečně zaručen až v situaci, kdy všechna vytvořená vlákna dokončila všechna volání metody `body` pro všechny hodnoty `i`.

Předpokládejte následující třídu zapsanou v jazyce C#:

```
1  class JpegCounter {
2          public JpegCounter(string[] filenames) { _filenames = filenames; }
3
4          string[] _filenames;
5          int _count;
6          object _globalLock = new object();
7
8          public int CountJpegs() {
9                  _count = 0; Parallel.For(0, _filenames.Length, ProcessFilename); return _count;
10         }
11
12         void ProcessFilename(int i) {
13                 lock (_globalLock) {
14                         string ext = Path.GetExtension(_filenames[i]).ToUpper();
15                         if (ext == ".JPG" || ext == ".JPEG") {
16                                 _count++;
17                         }
18                 }
19         }
20 }
```

1. V kontextu vícevláknového programu stručně vysvětlete, co to je race condition, a jak může vznikat.
2. Vysvětlete, zda je v uvedeném kódu důležité použití zámku `_globalLock`, a zda by bez jeho použití mohlo docházet k nějakým race conditions. Mohla by metoda `ProcessFilename` bez využití zámku `_globalLock` (tj. po smazání řádků 6, 13 a 18) vracet pro stejný vstup předaný třídě `JpegCounter` jinou hodnotu, než verze uvedená zde v zadání?
3. Předpokládejte, že vytvoříme program, který vyrobí novou instanci třídy `JpegCounter`, která dostane odkaz na seznam 10.000.000 jmen souborů, kde rámcově desetina z nich má příponu `.jpg` nebo `.jpeg`. Dále předpokládejte nějaký typický dvoujádrový procesor, na kterém takový program spustíme. Proveďte hrubý odhad, kolikrát bude asi uvedená implementace metody `CountJpegs` rychlejší (nebo pomalejší) než její sekvenční implementace (kdybychom napsali běžný `for` cyklus a všechna volání metody `ProcessFilename` provedli postupně v jednotlivých iteracích `for` cyklu rovnou z volajícího vlákna). Vysvětlete, zda (a případně jak) by se dala metoda `ProcessFilename` co nejjednodušeji upravit tak, aby byla paralelní verze rychlejší než verze uvedená v zadání.



(1) Race (z anglického „závod") je situace, kdy kvůli špatné synchronizaci a „neohlížení se na sebe" dvě vlákna způsobují nekonzistenci výpočtu.

Na příkladu — v okamžiku, kdy vlákno b přičítá ke A pětku, toto A „už neplatí" — mezitím co ho b načetlo a pracovalo s ním, vlákno a ho změnilo. Tedy místo aby bylo A celkově zvýšeno o 10, je zvýšeno jen o 5. ✓

Spolu. To by se mohlo stát v zadání — jestliže se nemýlím, že count++ je sled read(_count); _count := _count+1 a write(_count) pak by bez toho zámku mohla dvě vlákna pro dva JPEG soubory „stejným stylem" nahodit počítadlo jen o jedničku místo o 2. Takže ANO, bez něj by mohla vracet o něco nižší, špatný výsledek. ✓

(3) Podle mě bude +- stejná, protože zámek je kolem celého kódu, v němž se něco děje, takže ve výsledku bude vždy pracovat (dělat něco zajímavého) jen jedno vlákno, takže z toho bude vlastně sekvence. ✓

Přitom „sahání do stejných dat" je jen _count++ (řádek 16), takže by stačilo dát „pod zámek jen to, a vyhodnocování, jestli má daný soubor příponu JPEG, by mohlo být skutečně paralelně zpracováno. Odhad, o kolik by to pak bylo rychlejší, však nemám, ale aspoň čas potřebný na řádky 14 a 15 by se snížil až na 1/2. ✓

STUDENT 30

otázka 4

2 BODY

Kód studenta 30

7 Databáze a Web: Validita XML dat (otázka studijního zaměření – 3 body)

1. Vysvětlete pojem "validní XML schéma".
2. Uveďte alespoň 3 rozdíly mezi jazykem DTD a XML Schema.
3. V jazyce XML Schema vyjádřete element adresa, který obsahuje podelementy ulice, číslo popisné, město a PSČ v libovolném pořadí. Lze takové schéma vyjádřit i v jazyce DTD? Pokud ano, jak. Pokud ne, proč.

(1) = takové schéma, které dodržuje veškerá / v podstatě syntaktických pravidel

- párové tagy jsou vždy uzavřené
  <jmeno> Jan Novák </jmeno>
- případně nepárové jsou správně zapsané
  <tag /> (např.
 nebo v XHTML
- tagy se navzájem „nekříží"
  <a> Bla | NE <a> Bla
- atributy jsou zapsány v uvozovkách za rovnítkem („HTMLS zápis" bez uvozovek není problém)

Tato je správná sformovanost, ne validita, je to jen část odpovědi

(2) • DTD je starší
- mají jiný zápis
- XML schema je validní XML ~~se~~ jazyk (namespace xs:), je to tedy „XML, jímž se definuje jiné XML", zatímco DTD ~~nedefinuje~~ ne
- XML Schema má širší vyjadřovací schopnosti; např. podporuje dědičnost typů

DTD: <! ELEMENT ... <! ATTRIBUTE ... >
<xs:simpleType name="...">

OK

(3)
```
<xs:complexType name="address">
  <xs:string name="street" />
  <xs:int name="streetNumber" />
  <xs:string name="city" />
  <xs:int name="ZIP" />
</xs:complexType>
```

NEPŘESNÉ, CHYBÍ UKÁZKOVÝ KONSTRUKT

Kdyby to chtěla uspořádaný, nikdy by nebyl „type"

* ... číslo popisné, teď si nejsem jistá anglickým ?

V DTD by nešlo, NENÍ ÚPLNĚ PRAVDA (RESP. PŘESNÉ) tam je dané pořadí ve výčtu

- v DTD by nešlo udělat lib. pořadí

1 BOD

Kód studenta 30

8 Databáze a Web: XSLT (otázka studijního zaměření – 3 body)

1. Krátce popište jak pracuje XSLT procesor. Vysvětlete pojem "implicitní XSLT šablona".
2. Jaký bude výstup aplikace prázdného XSLT skriptu na dokument obsahující pouze element `<h1 c="blue">Hello world!</h1>`?
3. Napište XSLT skript, jehož výstupem je seznam názvů a hodnot všech atributů libovolného vstupního XML dokumentu.

(1) • XSLT je XML jazyk pro transformaci XML do jiného XML, typicky se používá pro "převod" XML do HTML

• jednotlivé šablony* mají pomocí XPath vyjádřeno, na co má být daná transformace aplikovat, funguje tak, že procesor, XPath "matchuje" elementy/podstromy ve vstupním dokumentu a na ty pak aplikuje příslušné transformace, dají se však "volat i přímo" ze vnitř jiné šablony

MOJE DOBRÉ PŘEDSTAVY

• „implicitní XSLT" šablona" — ta, co se na element(y) aplikuje na základě „matchnutí" s XPath nějaké šablony — ty, co nejsou „zavolané" MOJE PRAVDA

(2) prázdný ME — aby něco, musela bych aspoň aplikovat transformaci „identita"

(3) ~~<xslt:template match="/">~~ / vím jen, že tagy jsou <xslt:template .../>, <xslt:for-each atp.)

Nepamatuji si přesnou syntaxi. Nicméně — jak jsem psala, XSLT ~~umí~~ aplikuje transformace buď na základě „matchnutí" nebo „zavolání". Tento skript by tedy vypadal schematicky tak, že by měl šablonu na "/" (tedy začal by v kořeni", ~~a v podstatě prohledáváním do~~ vždy by prohledal děti, vypisoval by názvy a hodnoty, a na každého by rekurzivně zavolal šablonu, která by dělala to samé — „zarážka" rekurze by byly listy v celém stromě.

IDEA OK, KONKRÉTNÍ SYNTAXE BY NEMUSELA

2 BODY

Kód studenta 30

9 Databáze a Web: Relační úplnost, spojení tabulek a SQL (otázka studijního zaměření – 3 body)

1. Na vhodném jednoduchém příkladu vysvětlete rozdíl mezi vnitřním a vnějším spojením tabulek v jazyce SQL. INNER JOIN × OUTER JOIN
2. Vysvětlete pojmy "relace" a "relačně úplný jazyk". Je jazyk SQL relačně úplný? Proč?

(1) ~~[illegible]~~ vnější spojení tabulek ~~[illegible]~~ k jedné tabulce „z vní" přilepuje k tabulce tabulku druhou – zachová všechny řádky první tabulky, a kde je shoda, tam doplní buňky z druhé, jinak tam je řádek doplněn prázdnými buňkami ✗ OK

~~vnitřní spojení – zachová pouze ty řádky kde je shoda, tzv. inner join je také nazýván „natural" (přirozený), u pravého/levého to hraje roli, kdyby je shoda více – pak při pravém inner joinu mohu „zapikovat" levou tabulku a u levého pravou~~ místo „doplňování výsledků prázdnými buňkami řádek vynechá"

(2) • relace matematicky – mám-li množiny X, Y, pak relace je libovolná podmnožina jejich kartézského součinu ANO

• v databázích se jedná o vztahy mezi ~~[illegible]~~ objekty (např. „Oddělení má ~~nejvýše~~ alespoň jednoho zaměstnance a zaměstnanec patří právě do jednoho oddělení") NENÍ TAK ÚPLNĚ PRAVDA – D VŽ JE JINÝ VÝZNAM?

relační ~~popis dat~~ vrstva návrhu databáze je ta „nejvyšší", obsahuje „popis světa" daného ~~[illegible]~~ systému, je možné jej popsat např. ER diagramy [Oddělení] 1 —— 1..N [Zaměstnanec]

• relačně úplný jazyk je takový jazyk, který umí vyjádřit všechny druhy relací typu [objekt] (x,y) —— vztah —— (z,w) [objekt] NENÍ PRAVDA

↳ samotný jazyk SQL by neuměl vyjádřit např. max. počet („Oddělení má nejvýše 5 zaměstnanců") – k tomu již potřebuji nějaké jeho relačně úplné rozšíření, které umí triggery, co ty constraints OK INSERT validuje 0,5

Mějme tabulku lidí a PSČ

Jméno a příjmení (JP)	a zde bydlí (město)
Jan Novák	Praha
Filip Hanák	Hanušovice
Petr Novotný	Pardubice

Město	PSČ
Praha	120 00
Pardubice	530 02
Sezemice	533 04

~~budeme joinovat přes město (vynechat tabulku JP – město – PSČ)~~

příklady

① LEFT OUTER JOIN

„lepím ~~levou~~ „matchli" řádky levé tabulky k pravé"

hlavička

Jan Novák	Praha	120 00
Petr Novotný	Pardubice	530 02
—	Sezemice	533 04

② RIGHT OUTER JOIN

„k levé lepím pravou"

Jan Novák	Praha	120 00
Filip Hanák	Hanušovice	.....
Petr Novotný	Pardubice	530 02

③ FULL OUTER JOIN

zde jako ② (dělám to z obou stran – ~~zde není match~~) ještě + řádek

—	Sezemice	533 04

~~LEFT INNER JOIN~~ ~~RIGHT INNER JOIN~~

④ (NATURAL) FULL INNER JOIN

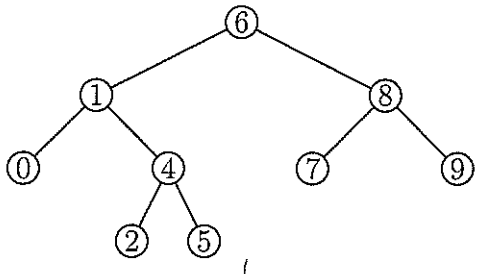
Jan Novák	Praha	120 00
Petr Novotný	Pardubice	530 02

Kód studenta 30

2 Algoritmy a datové struktury (3 body)

3 - body

1. Definujte AVL stromy a jejich základní vlastnosti (invarianty).
2. Popište průběh operace vložení prvku 3 do následujícího AVL stromu (s dodržením všech invariantů). Vysvětlete jednotlivé kroky.
3. Proveďte diskusi časové složitosti operací (1) vyhledání hodnoty a (2) vložení nového prvku, zejména ve srovnání se základní variantou binárního vyhledávacího stromu.



levý podstrom obsahuje menší a pravý větší prvky než on

vyhledávací

① AVL je binární a hloubkově vyvážený strom.

↳ binární strom = strom, pro jehož každý uzel s výjimkou listů platí, že jeho [illegible]

↳ hloubkově vyvážený := strom, pro jehož každý uzel platí, že [illegible]

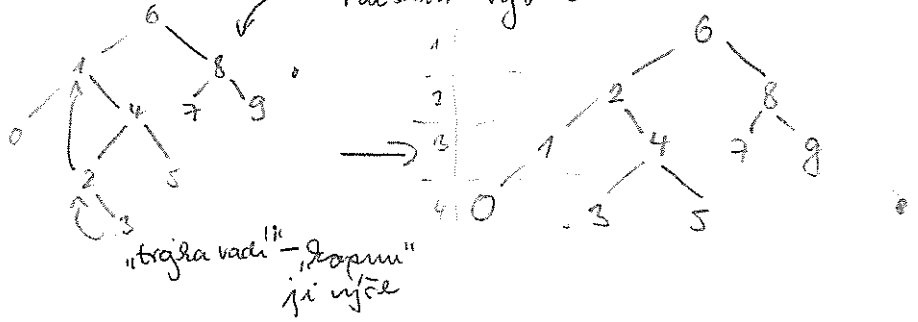
výšky (hloubky) jeho podstromů se liší nejvýše o 1. DK

② $3 < 6$ & $3 > 1$ & $3 < 4$ & $3 > 2$ → připojím

③ k ② : [illegible]

Mám:

Musím vyvážit — zrotovat levý podstrom kořene:



DK

③ • Hloubková vyváženost AVL garantuje $\Theta(\log n)$ čas pro nalezení prvku — oproti BVS, který se zrušili absenci požadavku na vyváženost může „zvrhnout" v zřetězený seznam prvků, v němž pak hledání může trvat až $\Theta(n)$. OK

• Vkládání v první fázi trvá stejně, co hledání, ale u AVL musím ještě navíc vyvažovat — a s rotováním se můžu klidně dostat až do kořene a někdy musím tím pádem přeuspořádat všechny prvky — a to může být $\Theta(n)$.

ok, vždy dělím v nejhorším případě od prvního listu ke kořeni hledání vždy $\leq O(1)$

2—

Kód studenta 30

3 Databáze (3 body)

1. Uvažujte následující relační schéma: AUTOR(ID, Jméno, DatumNarození), KNIHA(Název, PočetStran, IDAutora).
   Vysvětlete pojem "integritní omezení". Doplňte chybějící integritní omezení do uvedeného schématu.
   Zapište následující dotaz v SQL: jména autorů, kteří napsali alespoň deset stostránkových knih.
2. Vysvětlete pojem "funkční závislost na atributech relace". Rozšiřte některou tabulku z prvního příkladu tak, aby v ní byla porušena třetí normální forma.

(1) — integritní omezení — omezení hodnot atributů, resp. hodnot, např. horní limit pro čísla; to, že daný atribut musí být cizí klíč do jiné tabulky, musí být unikátní (UNIQUE)..

(FOREIGN KEY) # IDAUTORA JE CIZÍ KLÍČ

— (SELECT Jméno FROM AUTOR, KNIHA WHERE PočetStran = 100) ~~count(Jméno) ≥~~

NESTAČÍ JMÉNO

zde se provede implicitní natural join

POTŘEBUJÍ STEJNÝ NÁZEV ATRIBUTŮ

SELECT Jméno FROM (

GROUP BY HAVING

) WITH count(Jméno) ≥ 10

(2) • atribut a je funkčně závislý na b, jestliže jsou atributu na sobě tímto směrem závislé jejich hodnoty

→ např. PSČ udává obec, ~~ze jména~~ kde je sídlo firmy záleží na tom, jaká firma to je atd.

• Armstrongova pravidla

$Y \subseteq Z \Rightarrow Y \rightarrow Z$ (triviální) (speciálně $a \rightarrow a$)

$X \rightarrow Y \wedge Y \rightarrow Z \Rightarrow X \rightarrow Z$ (tranzitivita)

$X \rightarrow Y \wedge X \rightarrow Z \Leftrightarrow X \rightarrow YZ$ ((de)kompozice)

3NF → žádný atribut není tranzitivně závislý na žádném klíči

↖ spojením tabulek na kaž...

KNIHA(Název, PočetStran, IDautora, DatumNarození)

je podmínka narušena (DatumNarození ekvivalentní IDAutora závisí na Názvu