0+

Kód studenta 79

☐ počet listů odpovědi (pokud více než tento jeden)

1 Algoritmy rozděl a panuj (společné okruhy)

0+

1) Mějme algoritmus A, který na datech velikosti n udělá $T(n)$ elementárních operací. Algoritmus A je rekurzivní a pracuje takto (a, c, x, y jsou přirozená čísla, $a > 0$, $c > 1$):

a) Udělá $\Theta(n^x)$ elementárních operací, aby ze vstupních dat vybral a podmnožin velikosti n/c.

b) Rekurzivně pustí sám sebe na každou z vybraných podmnožin dat (pokud má velikost alespoň c, pro data menší velikosti vyřeší úlohu v konstantním čase).

c) Udělá $\Theta(n^y)$ elementárních operací, aby všechna řešení z bodu b) spojil do řešení původní úlohy na datech velikosti n.

Určete (bez důkazu) asymptoticky těsný odhad funkce $T(n)$ v závislosti na parametrech a, c, x, y.

2) Nechť X a Y jsou pole délky n, každé obsahující *setříděnou* posloupnost n přirozených čísel. Navrhněte a popište algoritmus s časovou složitostí $O(\log n)$, který najde medián (jeden z mediánů) všech $2n$ čísel obsažených v polích X a Y. Dokažte metodou z bodu 1), že složitost Vašeho algoritmu je opravdu $O(\log n)$.

1) $\Theta(n^x)$

a podmnožin $\frac{n}{c}$

a) $\underbrace{\frac{n}{c}, \frac{n}{c}, \ldots, \frac{n}{c}}_{a}$ $\left(\log_a \frac{n}{c}\right)$

b) $\frac{n}{c}, \frac{n}{c}, \frac{n}{c}, \ldots, \frac{n}{c}, \left(> \frac{n}{c}\right)$ ← konstantním čase. $\left(\log_c \frac{n}{c}\right)$

c) $\left(\log_a \frac{n}{c}\right)$

$T(n) = \left(n^{x+y} \cdot \log_a\right) \left(n^{x+y}\right) + \left(\log_a \frac{n}{c}\right)$

nejsme

(2)

X : délka (n) } setříděné.
Y : délku (n) }

⊕ Tady v tom případě bych použil metodu Merge jako je MergeSortu. Protože už máme setříděné ty posloupnosti takže můžeme začít mergovat ~~a jak víme tak to trvá v čase log (n)~~

~~Poté když máme a mi zastavíme pokud budeme~~ → O(n)

a jak víme tak to trvá v čase log (n). Pak když mergeji 2 posloupnosti tak dokázu vrátit median na pozici n a ~~nebo~~ to ve konstantním čase. To znamená, že mi to vrátí median pokud velikost nového pole = n.

⇒ Celková složitost O(log n).

function Merge (X, Y):

index1 = 0 ; // ukazatel na pozici v X
index2 = 0 ; //——————— Y
mergearr = [ ]

Přidáváme všechny čísla z { X do mergearr pokud X[index1] ≤ Y[index2]
Y do mergearr pokud X[index1] ≥ Y[index2]

Indexy se aktualizují, když je pořád menší nebo větší tak se ~~zvyšuje~~ o 1 ~~jak je~~ jinak to druhá se pak zvýší pokud se obrátila podmínka.

1 bod

Kód studenta 79

2 počet listů odpovědi (pokud více než tento jeden)

2 Semafor (společné okruhy)

Knihovna `System.Threading` jazyka C# obsahuje třídu `Semaphore` s metodami odpovídajícími operacím klasického semaforu, *P* (`WaitOne`) a *V* (`Release`). Objekt vytvořený pomocí `new Semaphore(0, 1)` odpovídá binárnímu semaforu inicializovanému nulou. Pokud více než jedno vlákno čeká na tentýž semafor, pořadí, ve kterém budou vlákna spouštěna po uvolnění semaforu, není specifikováno.

Program vpravo má produkovat výstup `OneTwoThreeFourFive`, přitom liché části výstupu mají být vypisovány funkcí `f1` a sudé části funkcí `f2`, přičemž tyto funkce běží v různých vláknech.

Synchronizace, implementovaná pomocí semaforu, téměř funguje, ale obsahuje časově závislé chyby (race conditions).

1. Popište scénář, který vede k jinému výstupu než `OneTwoThreeFourFive`, nebo končí uváznutím (deadlock). Scénař zapište jako posloupnost čísel řádek, přičemž každé číslo reprezentuje *ukončení* příkazu na dané řádce. Pouze vnitřky funkcí `f1` a `f2` jsou relevantní.
2. Existovaly by v tomto kódu časově závislé chyby i v případě, že by knihovna C# garantovala FIFO pořadí spouštění vláken čekajících ve `WaitOne`?
3. Napište řešení, které neobsahuje časově závislé chyby a spolehlivě vypisuje požadovaný výstup `OneTwoThreeFourFive`. Použijte přitom dva semafory a pouze jejich funkce `Release()` and `WaitOne()`. Řešení nesmí používat žádné jiné proměnné nebo objekty sdílené mezi vlákny kromě těchto dvou semaforů. Volání `Console.Write` musejí samozřejmě zůstat tam, kde jsou.

(Jazyk C# je v této otázce použitý pouze jako generický zástupce běžných programovacích jazyků, otázka ani řešení s jazykem C# jako takovým nesouvisí.)

```
1  class Program
2  {
3      private static Semaphore sem;
4
5      private static void f1()
6      {
7          Console.Write("One");
8          sem.Release();
9          sem.WaitOne();
10         Console.Write("Three");
11         sem.Release();
12         sem.WaitOne();
13         Console.Write("Five");
14     }
15
16     private static void f2()
17     {
18         sem.WaitOne();
19         Console.Write("Two");
20         sem.Release();
21         sem.WaitOne();
22         Console.Write("Four");
23         sem.Release();
24     }
25
26     static void Main(string[] args)
27     {
28         sem = new Semaphore(0, 1);
29         Thread t1 = new Thread(f1);
30         Thread t2 = new Thread(f2);
31         t1.Start();
32         t2.Start();
33         t1.Join();
34         t2.Join();
35     }
36 }
```

(2) Kdyby knihovna C# garantovala FIFO pořadí ✓ spouštění vláken čekajících ve WaitOne tak by v tomto kódu stále existovaly časově závislé chyby.

(2) Semafor (79)

(1) 7, 8, 9, 10, 11, 18, 19, 20, 21, 22, (11), 12, 13, 23

ne

? již provedeno

(3)

```
f1() {
    console.Write("One");
    sem.waitOne();
    sem.release();
    console.write("Three")
    sem.waitone();
    sem.release();
    console.Write("Five")
}
```

nevyužíváte 2. semafor

```
f2() {
    sem.release();
    console.Write("Two");
sem.waitOne()
    sem.release();
    console.Write("four");
    sem.waitOne();
}
```

může hned pokračovat X

Kód studenta 79

2 počet listů odpovědi (pokud více než tento jeden)

3 Báze vektorových prostorů (společné okruhy)

1. Zformulujte Steinitzovu větu o výměně vhodných vektorů mezi lineárně nezávislou a generující množinou (čili nikoli nutně bázemi).

2. Mějme reálnou matici $\boldsymbol{A}$ takovou, že je podobná diagonální matici $\boldsymbol{D}$ prostřednictvím součinu $\boldsymbol{R}^{-1}\boldsymbol{A}\boldsymbol{R}$, kde

$$\boldsymbol{R} = \begin{pmatrix} -1 & -1 & -8 & -2 & -8 \\ 0 & 1 & -3 & 0 & 2 \\ -1 & -2 & 6 & -2 & 3 \\ -1 & -3 & 7 & -1 & -4 \\ 1 & 2 & 0 & 2 & 4 \end{pmatrix},$$

přičemž je známo, že prvky na diagonále $\boldsymbol{D}$ jsou čísla $17, 17, 23, 17, 23$ v uvedeném pořadí.

Rozhodněte, zdali některý ze sloupců $\boldsymbol{R}$ lze vyměnit za následující vektor $\boldsymbol{u}_i$, aby matici $\boldsymbol{A}$ bylo stále možné diagonalizovat i prostřednictvím výsledné matice $\boldsymbol{R}'$.

Pokud taková výměna je možná, určete všechny sloupce matice $\boldsymbol{R}$, které mohou být vyměněny.

(a) Řešte pro $\boldsymbol{u}_1 = (4, 2, 2, -3, -2)^{\mathrm{T}}$.

(b) Řešte pro $\boldsymbol{u}_2 = (7, -6, -2, 12, -6)^{\mathrm{T}}$.

Své odpovědi zdůvodněte.

$$D = \begin{pmatrix} 17 & 0 & 0 & 0 & 0 \\ 0 & 17 & 0 & 0 & 0 \\ 0 & 0 & 23 & 0 & 0 \\ 0 & 0 & 0 & 17 & 0 \\ 0 & 0 & 0 & 0 & 23 \end{pmatrix} \Rightarrow \text{Naše vlastní čísla:}$$

$\lambda_1 = 17$

$\lambda_2 = 23$

(1) Steinitzova věta o výměně vhodných vektorů mezi lineárně nezávislou a generující množinou:

Pokud lze sestrojit matici A takovou, že je podobná k nějaké matici D prostřednictvím součinu $R^{-1}AR$ tak existuje vektor, který

zle ze sloupců R vyměnit tak aby matice A byla stále diagonalizovat i prostřednictvím výsledné matice R´

⊕ Jiná definice:

- Pokud máme lineárně nezávislé vektory a vyměníme ~~nějakou~~ nějaký vložný vektor, který neporušuje tu nezávislost tak se nemění generující množina.

79 (3)

$$R = \begin{pmatrix} -1 & -1 & -8 & -2 & -8 \\ 0 & 1 & -3 & 0 & 2 \\ -1 & -2 & 6 & -2 & 3 \\ -1 & -3 & 7 & -1 & -4 \\ 1 & 2 & 0 & 2 & 4 \end{pmatrix}$$

$H_3 \leftarrow H_3 - H_1$
$H_4 \leftarrow H_4 - H_1$
$H_5 \leftarrow H_5 + H_1$

$$\begin{pmatrix} -1 & -1 & -8 & -2 & -8 \\ 0 & 1 & -3 & 0 & 2 \\ 0 & 1 & 14 & 0 & 11 \\ 0 & -2 & 15 & 1 & 4 \\ 0 & 1 & -8 & 0 & -4 \end{pmatrix}$$

$H_3 \leftarrow H_3 - H_2$
$H_4 \leftarrow H_4 + 2H_2$
$H_5 \leftarrow H_5 - H_2$

$$\begin{pmatrix} -1 & -1 & -8 & -2 & -8 \\ 0 & 1 & -3 & 0 & 2 \\ 0 & 0 & 17 & 0 & 9 \\ 0 & 0 & 9 & 1 & 8 \\ 0 & 0 & -5 & 0 & -6 \end{pmatrix}$$

$H_4 \leftarrow 17H_4 - 9H_3$
$H_5 \leftarrow 17H_5 + 5H_3$

$$\begin{pmatrix} -1 & -1 & -8 & -2 & -8 \\ 0 & 1 & -3 & 0 & 2 \\ 0 & 0 & 17 & 0 & 9 \\ 0 & 0 & 0 & 17 & 55 \\ 0 & 0 & 0 & 5 & -57 \end{pmatrix}$$

$H_5 \leftarrow 17H_5 - 5H_4$

$$\begin{pmatrix} -1 & -1 & -8 & -2 & -8 \\ 0 & 1 & -3 & 0 & 2 \\ 0 & 0 & 17 & 0 & 9 \\ 0 & 0 & 0 & 17 & 55 \\ 0 & 0 & 0 & 0 & 694 \end{pmatrix}$$

a) u_1 ~~lze~~ dokáže ~~nakradit~~ nakradit $(-1, 0, -1, -1, 1)^T$ a $(-1, 1, -2, -3, 2)^T$.

b) u_2 ~~nelze~~ nara ~~nakra~~ nic ~~nelo~~ nedokáže nakradit.

1+

Kód studenta 79

2 počet listů odpovědi (pokud více než tento jeden)

4 Model teorie (společné okruhy)

1. Nechť T je teorie jazyka (signatury) $L = \langle \mathcal{R}, \mathcal{F} \rangle$ v predikátové logice (kde $\mathcal{R}, \mathcal{F}$ jsou množiny relačních a funkčních symbolů s danými aritami). Uveďte definice pojmů *struktura jazyka* L a *model teorie* T.
2. Vyjádřete následující tvrzení formulemi $\varphi_1, \varphi_2, \varphi_3$ v jazyce $L = \langle Z, S, P \rangle$ predikátové logiky (s unárními predikáty pro *'složit zkoušku'*, *'mít štěstí'*, *'být připraven'*).
   (a) *Ne každý, kdo složí zkoušku, má štěstí, ale kdo má štěstí, zkoušku složí.*
   (b) *Štěstí přeje připraveným. (Kdo je připraven, má štěstí.)*
   (c) *Nějaký student byl připraven, ale zkoušku nesložil.*
3. Pro teorii $T_1 = \{\varphi_1, \varphi_2\}$, a také pro teorii $T_2 = \{\varphi_1, \varphi_2, \varphi_3\}$, buď najděte nějaký její model anebo formálně dokažte (pomocí tablo metody, rezoluce, či Hilbertovského kalkulu), že žádný model nemá.

(2) φ_1: $(\exists x)(Z(x) \wedge \neg S(x)) \wedge (\forall y)(S(y) \to Z(y))$

φ_2: $(\forall x)(P(x) \to S(x)) \sim (\forall x)(\neg P(x) \vee S(x)) \Rightarrow \{\neg P(x), S(x)\}$

φ_3: $(\exists x)(P(x) \wedge \neg Z(x)) \Rightarrow \{\{P(y)\}, \{\neg Z(x)\}\}$ ✓

(3) ~~Pomocí φ_2 a φ_3 sestavíme následující rezoluci pro (T_2)~~

$\{\neg P(x), S(x)\}$ $\{P(y)\}$

$\sigma = \{x/y\}$

□

Protože T_2 je ~~zamítnutelná rezolucí~~ $\Rightarrow$ ~~nemá žádný model~~

① Nechť T je teorie jazyka $L = \langle \mathbb{R}, \mathbb{F} \rangle$

tak model teorie T jsou všechny struktury, ~~které~~ ve kterém T je pravdivý.

co znamená, že teorie T je pravdivá ve struktuře?

struktura jazyka L ~~je~~ ~~nějaké~~ jsou ~~dodrženy~~ v teorii T ??

ve v predikátové logice

struktuře ??

③ Pro T_1 sestavíme tablo:

$T\big((\exists x)(Z(x) \wedge \neg S(x)) \vee (\forall y)(S(y) \wedge Z(y))\big)$

$T\big((\forall x)(P(x) \to S(x))\big)$

$T\big((\exists x)(Z(x) \wedge \neg S(x))\big)$

$T\big(P(t_1) \to S(t_1)\big)$

$T\big(Z(c_1) \wedge S(c_1)\big)$ c_1: nová konst.

$F\big(P(t_1)\big)$

$T\big(Z(c_1)\big)$

$T\big(S(c_1)\big)$

$F\big(P(c_1)\big)$

$T\big(S(t_1)\big)$

$T\big(Z(c_1)\big)$

$T\big(S(c_1)\big)$

$T\big((\forall y)(S(y) \wedge Z(y))\big)$

$T\big(P(t_1) \to S(t_1)\big)$ t_1: term

$T\big(S(t_2) \wedge Z(t_2)\big)$ t_2: term

$F\big(P(t_1)\big)$

$T\big(S(t_2)\big)$

$T\big(Z(t_2)\big)$

$T\big(P(t_1)\big)$

$T\big(S(t_2)\big)$

$T\big(Z(t_2)\big)$

④ Pomocí tabla vidíme, že T_1 má 2 modely a to:

$Z(x)$	$S(x)$	$P(x)$
1	1	1
1	1	0

← ~~z toho~~ Tady má být správně kanonický model.

$T_2 = \{\varphi_1, \varphi_2, \varphi_3\}$.

~~φ_1 zle přepsat jako $\{\neg Z(f()), S(f(x))\}$~~

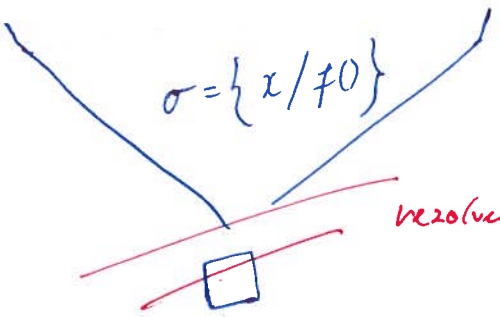
φ_2 zle přepat na $\{\neg P(x), S(x)\}$

φ_3 zle přepsat na $\{P(f())\}, \{\neg Z(f())\}$

tady u φ_3 jsem udělal skolemizaci. $x = f()$ (konst. symbol)

Dostáváme:

$\{\neg P(x), S(x)\}$ $\{P(f())\}$ $\{\neg Z(f())\}$



rezolventa je $\{S(f())\}$!

Pomocí rezolučního stromu je T_2 zamítnutelná

$\Rightarrow T_2$ nemá žádný model.

(3) Pro T_1 sestavíme tablo:

$T\left((\exists x)(Z(x) \wedge \neg S(x)) \wedge (\forall y)(S(y) \to Z(y))\right)$

$T\left((\forall x)(P(x) \to S(x))\right)$

~~$T((\exists x)(\forall y)((Z(x) \wedge \neg S(x)) \wedge \neg S(y) \vee Z(y)))$~~

$T((\exists x)(Z(x) \wedge \neg S(x))$

$T((\forall y)(S(y) \to Z(y))$

$T(Z(c_1))$ $\quad c_1$: nová konstanta.

$T(\neg S(c_1))$

$F(S(c_1))$

$F(S(c_1))$ $\qquad$ $T(Z(c_1))$

$F\,P(c_1)$ $\quad T(S(c_1))$ $\quad F(P(c_1))$ $\quad T(S(c_1))$

⊗ ⊗

Pomocí tabla vidíme, že T_1 má 1 model.

$\mathcal{A} = \{$ c_1 : konstanta.

$\{ Z(c_1) \}$

$\{ \overline{P(c_1)}, \overline{S(c_1)} \}$

$\}$

Není to struktura
(univerzum a 3 malé
relace)

2 - kop

Kód studenta 79

2 počet listů odpovědi (pokud více než tento jeden)

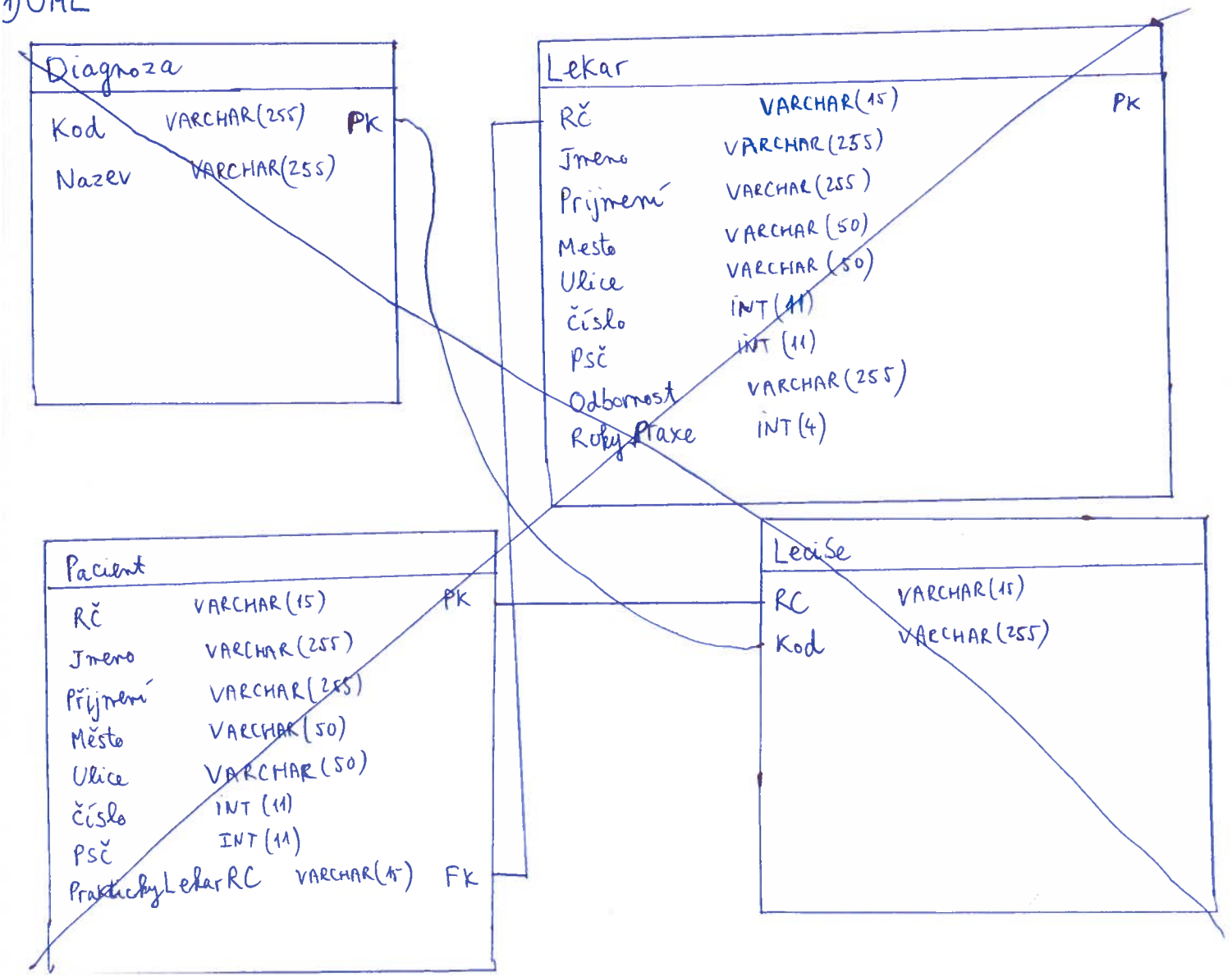
5 Datový model (specializace WDOP)

V informačním systému nemocnice je použit následující logický relační datový model, kde podtržením jsou vyznačeny klíče a italikou cizí klíče:

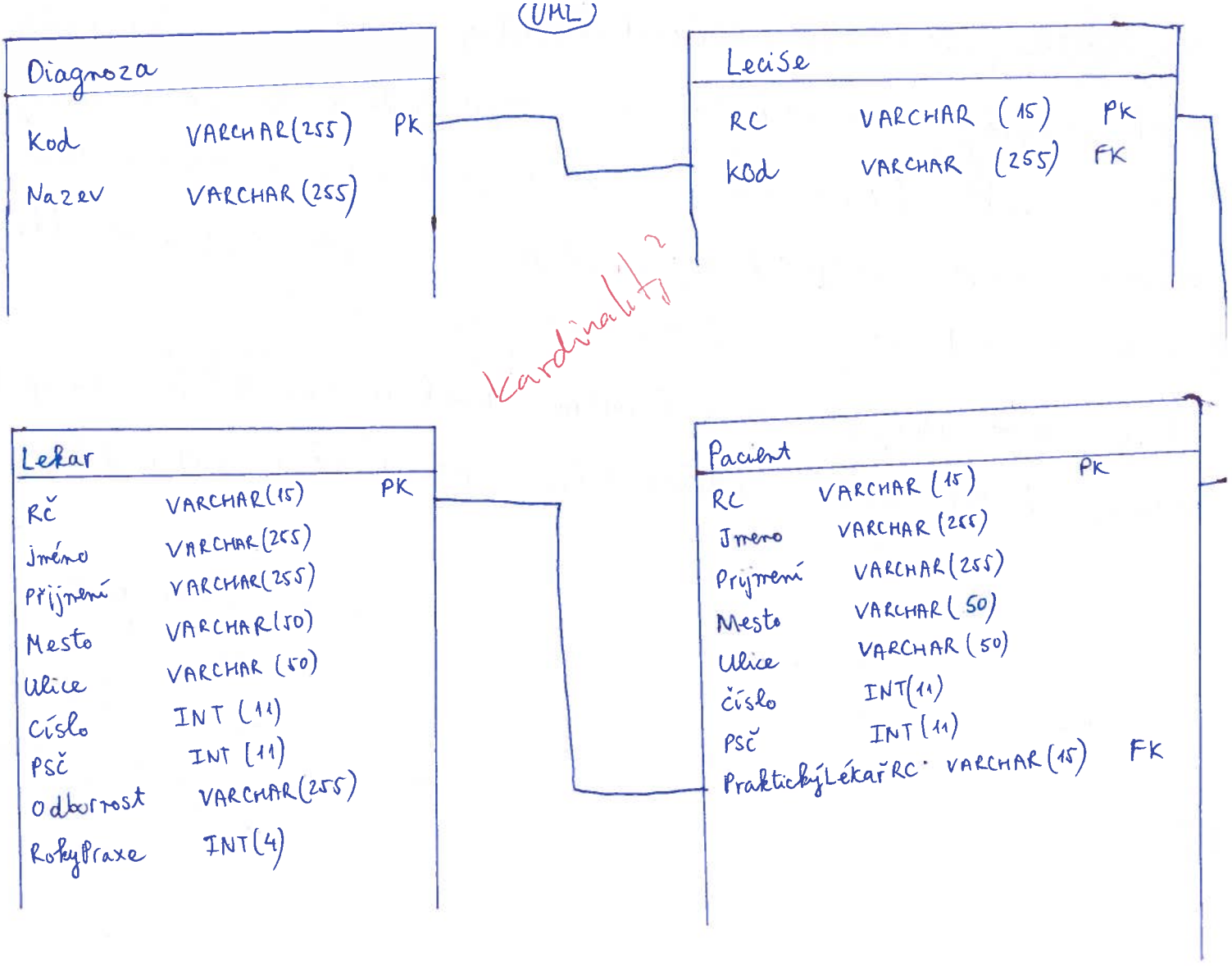
- Diagnóza(**Kód**, Název)
- Lékař(**RČ**, Jméno, Příjmení, Město, Ulice, Číslo, PSČ, Odbornost, RokyPraxe)
- Pacient(**RČ**, Jméno, Příjmení, Město, Ulice, Číslo, PSČ, *PraktickýLékařRČ*), PraktickýLékařRČ ⊆ Lékař[RČ]
- LéčíSe(***RČ***, *Kód*), RČ ⊆ Pacient[RČ], Kód ⊆ Diagnóza[Kód]

1. Znázorněte výše uvedený logický relační datový model pomocí diagramu ve zvoleném konceptuálním jazyce (ER, UML).
2. Pokud je to potřeba, rozšiřte konceptuální model a odpovídající logický relační datový model tak, aby:
   - každý pacient se mohl léčit na libovolný počet diagnóz,
   - každý pacient měl právě jednoho praktického a mohl mít libovolný počet dalších lékařů, → spojovací tabulka
   - každý pacient mohl podstoupit libovolný počet vyšetření v rámci nějaké diagnózy a naopak,
   - pacient, který je zároveň lékařem, neměl osobní údaje evidovány redundantně.

1) UML



(UML)



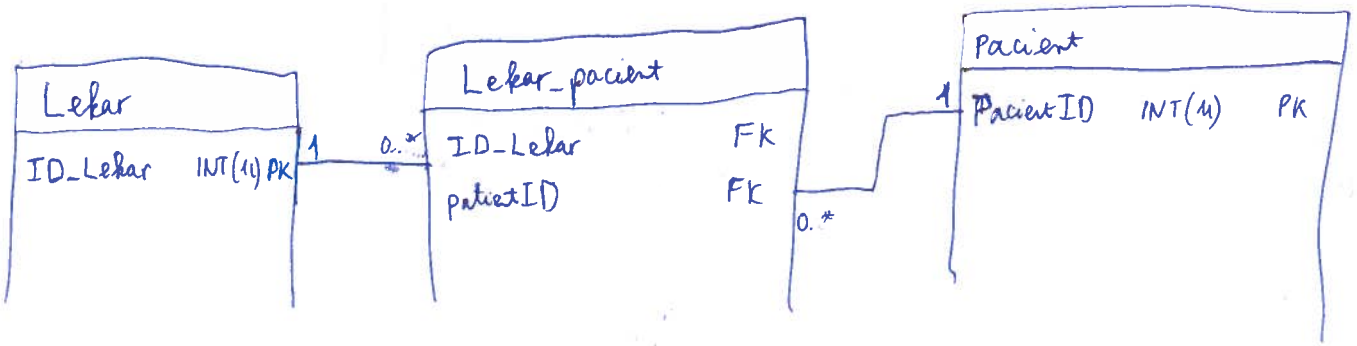
(2)

- Aby se pacient mohl léčit libovolným počet diagnóz, tak přidám do tabulky pacient "PacientID INT(11)", kt. bude další klíč. A do tabulky LeciSe přidám PacientID ⊆ Pacient.PacientID.

Proč nepoužít kód? Může kód ID odkazovat na různé diagnózy?

- Aby pacient měl právě jednoho praktického lékaře a mohl mít libovolný počet dalších lékařů tak přidám spojovací tabulku mezi "Lekar" a "Pacient" takto:

Přidám ID_Lekar do Lekar



- Aby každý pacient mohl postupovat libovolný počet vyšetření v rámci nějaké diagrózy a naopak tak to je stejné jako u té první ukázky. to že tabulka LeciSe má Kod ⊆ Diagroza.kod a ~~pacientID ⊆ Pacient~~ pacientID ⊆ Pacient.pacientID tak to zajišťuje (slouží jako spojovací tabulka pro Diagroza a Pacient

Vyšetření na X a léčení se na X. jsou dvě různé věci

- Aby pacient, který je zároveň lékařem, neměl osobní údaje evidovány redundantně tak bych to udělal tak, že pacient.RČ ⊆ Lekar.RČ.

⊇

ne každý pacient je lékař, ale každý lékař může být chápán jako pacient

5 Datový model

3 kop

Kód studenta 79

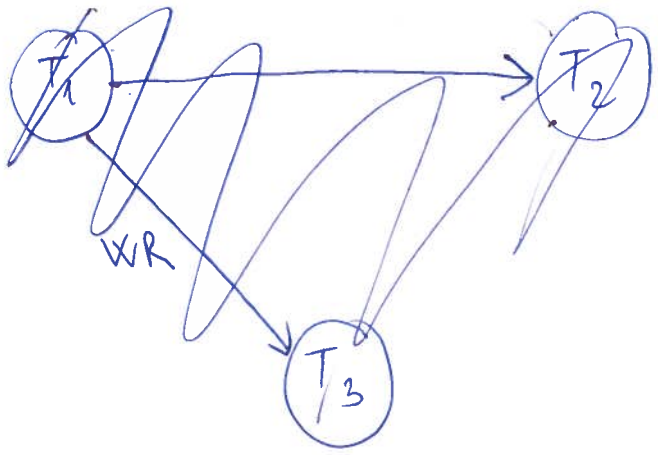
☐ počet listů odpovědi (pokud více než tento jeden)

6 Transakce (specializace WDOP)

Pro následující transakční rozvrh, kde R znamená operaci čtení a W operaci zápisu:

	T_1	T_2	T_3
1	W($Pacient_1$)		
2		???	
3		W($Pacient_1$)	
4		COMMIT	
5			R($Pacient_1$)
6			W($Pacient_1$)
7			COMMIT
8	R($Pacient_2$)		
9	COMMIT		

1. Jaká databázová operace čtení/zápisu v transakci T_2 v čase 2 místo ??? způsobí, že rozvrh nebude konfliktově uspořádatelný (conflict-serializable)? Své rozhodnutí zdůvodněte.
2. Jaká databázová operace čtení/zápisu v transakci T_2 v čase 2 místo ??? způsobí, že rozvrh nebude zotavitelný (recoverable)? Své rozhodnutí zdůvodněte.

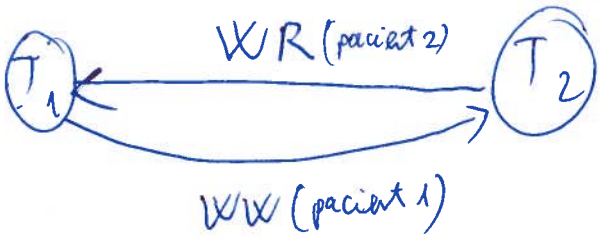


1) ~~- Rozvrh vždy bude konfliktově uspořádatelný ať už cokoliv dosadíme do ???, protože vždy nedokážeme sestrojit cyklický graf z této transakce (jinak řečeno, vždy dostaneme acyklický graf).~~

2) Rozvrh nebude zotavitelný pokud za ??? dosadíme R(Pacient 1), protože poté T_2 hned začala zapisovat a comitla smažu T_1 stihla W(Pacient1) ~~comit~~ commitnout. ✓

(1) za ??? dosadíme W(Pacient 2), aby rozvrh nebyl konfliktně uspořádaný.

Důvod: Protože máme smyčku, teda rozvrh tvoří cyklický graf



Kód studenta 79

3 počet listů odpovědi (pokud více než tento jeden)

7 API a skriptování (specializace WDOP)

1. Navrhněte a popište REST API pro nemocniční systém z dřívější otázky. API musí podporovat následující funkcionalitu:
   - získání pacientů, jejichž jméno odpovídá query patternu,
   - získání informací o konkrétním pacientovi,
   - vytvořit, aktualizovat a zrušit pacienta,
   - získat seznam diagnóz pacienta,
   - přidat a odebrat diagnózu pacienta,
   - získat seznam lékařů pacienta,
   - získat seznam pacientů, kteří mají aspoň jednu ze zadaných diagnóz (seznam diagnóz může být hodně dlouhý).
2. Mějme webovou aplikaci, která zajišťuje přístup k IS přes navržené API. Napište JavaScript fragment, který po zadání query patternu (první endpoint) a kliknutí na tlačítko vyvolá příslušný HTTP API request, získá všechny pacienty odpovídající dotazu a zobrazí je v seznamu. Když uživatel klikne na konkétního pacienta ze seznamu získaného prvním dotazem, tak se aktualizuje seznam jeho doktorů a diagnóz získáním dat z API a jejich následným zobrazením.

   Předpokládejte, že API vrací data v JSON formátu a tento krátce popište formou příkladu pro každý použitý z endpointů. Předpokládejte existenci funkcí `displayDoctors` a `displayDiagnoses` pro zajištění úpravy DOM stromu. Stejně tak můžete použít funkce `clearDoctors` a `clearDignoses`. Dále předpokládejte existenci všech potřebných HTML elementů.

   Dbejte na správné využití asynchronního zpracování požadavků. Zajistěte, aby i při nepříznivém souběhu asynchronních volání po načtení seznamu pacientů byl seznam doktorů a diagnóz v konzistentním stavu, tj. zobrazují se seznamy ke zvolenému pacientu a nezobrazují se seznamy v případě, kdy pacient vybrán nebyl. Není třeba řešit chybové stavy fetch operací ani autentikaci.

7) API a skriptování.

2) displayDoctors();
displayDiagroses();

clearDoctors();
clearDigroses();

Mám tlačítko, který provede tu akci a má id = 'btn'

var btn = document.getElementById('btn');

A mám textbox kam se zadává query pattern a má id = 'queryPattern'

var query = document.getElementById('queryPattern');

```
btn.onClick() {
    var query = document.getElementById('queryPatern').innerHTML;
```

pak se tady používá něco myslím, že je to fetch

~~feth~~ fetch(METHOD:GET; url: URL1,
parameter: {query = query})

Pokud success tak
DisplayPacient(res.data)

await res.json()

await / async ?

Pokud ne tak
Console.log(err);

fce DisplayPacient(data) {

for(i=0 → data.length) {

Tady se vypíšou ti pacienti ve způsobu:

pacient.contain.innerHTML += <button id=data[i]_pacientID> data[i].jméno </butta>

}

Pak když máme seznam tak pomocí ID toho pacienta pošleme GET request získání seznam diagnóz a jeho doktorů pomocí našeho API, když přes fci displayDoctors a displayDiagnoses. Pokud klikneme na jiného pacienta tak nejdříve se zavolá clearDoctors a clearDiagnoses.

Ukázka JSON, který vrací první endpoint může vypadat takto:

```
'Pacient' : {
"1" : { "RC" : "1234..." , "Jmeno" : "DUONG" , ..... },
"2" : { "RC" : "234..." , "Jmeno" : "DAVID" , ... }
...
}
```

(7) API a skriptování

(1) Třeba moje API bude na ~~URL1~~ https://URL1

(+) Získání pacientů, jejichž jméno odpovídá query patternu:

GET

http://URL1/jmeno='query patter'

↑ ? query=<...>

SELECT * FROM pacient WHERE jmeno LIKE '%jmeno%';

(+) Získání informací o konkrétním pacientovy (přes PacientID jak jsem ~~bez~~ jména by mělo být přidal jako součástka

GET

http://URL1/id=1

↑ id bude =1

url/pacient/<id>

SELECT * FROM pacient WHRE pacientID = id

(+) Vytvořit, aktualizovat a zrušit pacienta.

POST

- Vytvořit: http://URL1/createpacient?RC=123&jmeno='DUONG'

INSERT INTO PACIENT (RC, Jmeno, ...) Values (RC, Jmeno, ...)

- Aktualizovat: https://URL1/updatePacient/id

UPDATE

PUT

UPDATE podle RFC 7231 nemusí být v payload request query string

je o tě

ALERT TABLE Pacient

SET ....

WHERE .. pacientID = id

DELETE

- Zrušit: http://URL1/deletePacient/id

nemusí být v URL, je definováno typem requestu

ALERT TABLE Pacient

DELETE row

Where pacientID = id

- Získat seznam diagnóz pacienta.

http:// URL1 / getDiagnoze / id GET

```
SELECT d.kod, d.nazev
FROM Pacient AS p
JOIN LeciSe AS l
ON p.pacientID = l.pacientID
JOIN Diagnoza as d
ON l.kod = d.kod WHERE p.pacientID = id
```

(pacientID jsem přidal v předch. úlohách)

- Získat seznam lékařů pacienta

http:// URL1 / getDoctorList / pacientID GET

```
SELECT l.jméno, l.příjmení
FROM pacient AS p
JOIN Lekar AS l
ON p.praktickýlékař RC = l.RC
WHERE p.pacientID = pacientID
```

- Získat seznam pacientů, kteří mají aspoň jednu ze zadaných diagnóz.

http:// URL1 / getPacientListDiagnost / diagnost = 'A,B,C' GET

spa spracuji diagnost = 'A,B,C' do pole [A, B, C]

může být dlouhé → nutný post request

```
SELECT p.jméno, p.příjmení
FROM Pacient AS p
JOIN LeciSe AS l ON p.pacientID = l.pacientID
JOIN Diagnoza AS d ON d.kod = l.kod
WHERE d.název IN (A, B, C)
GROUP BY p.pacientID
```

Kód studenta 79

O+

☐ počet listů odpovědi (pokud více než tento jeden)

8 Získávání informací (specializace WDOP)

Uvažujte situaci, kdy je třeba obohatit systém z předchozích otázek tak, aby umožňoval stanovit diagnózu na základě zadaných symptomů. Mějme kolekci dokumentů, kde každý dokument popisuje konkrétní nemoc.

1. Jak by vypadal booleovský model pro danou úlohu? Jaké termy by obsahoval spcializovaný slovník pro danou úlohu? Jak jsou reprezentovány dotazy a dokumenty v booleovském modelu a jak lze dotazování v tomto modelu efektivně implementovat?
2. Řekněme, že chceme systém modifikovat tak, aby dotaz nebyl seznam symptomů, ale přímo zpráva lékařského vyšetření. Jakou modifikaci booleovského modelu je vhodné v takovém případě použít? Jak se změní reprezentace dotazu a dokumentu? Jak bude pak vyhodnocována podobnost mezi dotazem a dokumentem a proč?

nemoc | symptoty
klíč | hodnota

⊕ Dotazy a dokumenty v booleovském modelu jsou reprezentované jako klíč - hodnota. Dotazujeme v booleovském modelu pomocí klíčů.

R binární matice
R bitové operace nad vektorem termů

⊕ Můžeme modifikovat booleovský model tak, že prohodíme klíče a hodnoty. Což hodnotu se stane klíčem a klíč se stane hodnotou

R to je prostě obdobná reprezentace pomocí invertovaného seznamu

- modifikace → vektorový model → TF-IDF → → binární vzdálenost