1

Kód studenta 3

☐ počet listů odpovědi (pokud více než tento jeden)

1 Algoritmy rozděl a panuj (společné okruhy)

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) ~~$\Theta(n^x) = \Theta(\log_a n/c)$~~

$\Theta(n^x) \cdot \Theta(\log_a n/c) + \Theta(n^y)$?? X

2) mám setříděné posloupnosti $X = x_1 \dots x_n$

$Y = y_1 \dots y_n$

~~jedna hledá~~

pořídím si ukazatele i na pole X a j na pole Y a procházím prvky polí X a Y od $i=0$ a $j=0$ porovnávám prvky a pokud je $x_i < y_j$ tak zvýším i o 1 a pokračuje dál, pokud $x_i > y_j$ tak zvýším j o 1 a pokračuje dál, pokud $x_i = y_j$ i i j zvýším o 1

– pole postupně rozdělují na poloviny a porovnám vždy první prvky z X_i s posledními z Y a naopak, takto zjistím zda nějaké celé kusy nemůžu slepit za sebe, pokud ne, noříím se hlouběji a opět posloupnosti rozdělím na půl, a celou dobu se snažím vyřadit pouze posloupnost délky n, kde poslední prvek bude medián, v případě, že

→ ~~vždy víme jistě~~, že nějaká část posloupnosti bude na n+1 polích tak ji zahazuji

Ukázka na příkladu:

$X = [1, 3, 7, 8, 9, 10]$
$Y = [1, 2, 3, 4, 5, 6]$

porovnám $1 < 10$ $6 > 1$
$1 < 1$ $10 > 1$
nic nemůžu zahodit

→ rozdělím na poloviny

$X_1 = [1, 3, 7]$
$Y_1 = [1, 2, 3]$

$1 = 1$ $1 < 7$
$1 < 3$ $3 < 7$
nemůžu nic zahodit

~~$X_2 = [8, 9, 10]$~~
$Y_2 = [4, 5, 6]$

$4 < 8$
$6 < 8$

mohu zahodit X_2
→ jsme na délce $2m - X_2.length = 12 - 3 =$
$\Rightarrow 9 > n/2 = 6$,
pokračuji v dělení

$X_1 = [1, 3, 7]$ → $X_{11} = [1, 3]$ $X_{12} = [7]$
$Y_1 = [1, 2, 3]$ $Y_{11} = [1, 2]$ $Y_{12} = [3]$
$Y_2 = [4, 5, 6]$ $Y_{21} = [4, 5]$ $Y_{22} = [6]$

porovnám a vyhazuji X_{12} → $9 - 1 = 8$
$8 > n/2 = 6$
→ pokračuje dál

porovnám a vyhazuji Y_{22} → $8 - 1 = 7$
$7 > n/2 = 6$
pokračuje dál

→ Y_{21} již vyhodit nemůžu → dál rozděluje
$n/2 = 6 < 7 - 2 = 5$?

→ $X_{111} = [1]$ $X_{112} = [3]$ $Y_{12} = [3]$
$Y_{111} = [4]$ $Y_{112} = [2]$
$Y_{211} = [4]$ $Y_{212} = [5]$

→ vyhodím 5 a dostávám medián 4

→ defacto se snažíme posloupností procházet jako binárním stromem, abychom docílili složitosti $\Theta(\log n)$

→ na každé úrovni provedu konstantní počet operací složitost je tedy $\Theta(\log n)$

hloubka stromu

popis algoritmu bez uvádění na jednotlivých konkrétních datech není úplně srozumitelný ...

Kód studenta 3

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 }
```

Otázka 2. Kód studenta 3

1) (18), 7, 8, 18, 19, 20, 21, 22, 23, 9, ...

→ vypíše se One, Two, Four OK

2) ne FIFO nic nevyřeší, ~~ok~~ OK.

3)
```
private static Semafor semafor1
private static Semafor semafor2
private static void f1()
{
    Console.WriteLine("One");
    semafor2.Release();
    semafor1.WaitOne();
    Console.WriteLine("Three");
    semafor2.Release();
    semafor1.WaitOne();
    Console.WriteLine("Fine");
}

private static void f2()
{
    semafor2.WaitOne();
    Console.WriteLine("Two");
    semafor1.Release();
    semafor2.WaitOne();
    Console.WriteLine("Four");
    semafor1.Release();
}
```
OK

Kód studenta 3

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.

1) Pokud vektory $x_1, x_2, \dots x_m$ generují prostor X a $y_1, \dots y_n$ jsou lineárně nezávislé vektory, tak vektor x_i můžeme vyměnit za vektory y_j, které jsou lineární kombinací x_i. To by fungovat!

Nelze určit platnost

2) $$D = \begin{pmatrix} 17 & 0 & 0 & 0 & 0 \\ 0 & 17 & 0 & 0 & 0 \\ 0 & 0 & 23 & 0 & 0 \\ 0 & 0 & 0 & 17 & 0 \\ 0 & 0 & 0 & 0 & 23 \end{pmatrix}$$

vlastní čísla podobných matic jsou shodná

vlastní čísla D

$\lambda_1 = \lambda_2 = \lambda_4 = 17$

$\lambda_3 = \lambda_5 = 23$

$D = R^{-1} A R$

$DR = R^{-1} A$

$R^{-1} D R = A$

Odeźka 3. kod studenta 3

$$\left(\begin{array}{ccccc|c} -1 & -1 & -8 & -2 & -8 & 1\ 0\ 0\ 0\ 0 \\ 0 & 1 & -3 & 0 & 2 & 0\ 1\ 0\ 0\ 0 \\ -1 & -2 & 6 & -2 & 3 & 0\ 0\ 1\ 0\ 0 \\ -1 & -3 & 7 & -1 & -4 & 0\ 0\ 0\ 1\ 0 \\ 1 & 2 & 0 & 2 & 4 & 0\ 0\ 0\ 0\ 1 \end{array}\right) \sim \left(\right.$$

$$\left(\begin{array}{ccccc|c} -1 & -1 & -8 & -2 & -8 & 1\ 0\ 0\ 0\ 0 \\ 0 & 1 & -3 & 0 & 2 & 0\ 1\ 0\ 0\ 0 \\ 0 & 1 & -14 & 0 & -11 & 1\ 0\ 1\ 0\ 0 \\ 0 & & & & & \end{array}\right)$$

$$\left(\begin{array}{ccccc|cc} -1 & -1 & -8 & -2 & -8 & 4 & 7 \\ 0 & 1 & -3 & 0 & 2 & 2 & -6 \\ -1 & -2 & 6 & -2 & 3 & 2 & -2 \\ -1 & -3 & 7 & -1 & -4 & -3 & 12 \\ 1 & 2 & 0 & 2 & 4 & -2 & -6 \end{array}\right) \sim$$

$$\left(\begin{array}{ccccc|cc} -1 & -1 & -8 & -2 & -8 & 4 & 7 \\ 0 & 1 & -3 & 0 & 2 & 2 & -6 \\ 0 & 0 & 6 & 0 & 7 & 0 & -8 \\ 0 & -1 & 7 & 1 & 0 & -5 & 6 \\ 1 & 2 & 0 & 2 & 4 & -2 & -6 \end{array}\right) \sim \left(\begin{array}{ccccc|cc} -1 & -1 & -8 & -2 & -8 & 4 & 7 \\ 0 & 1 & -3 & 0 & 2 & 2 & -6 \\ 0 & 0 & 6 & 0 & 7 & 0 & -8 \\ 0 & -1 & 7 & 1 & 0 & -5 & -6 \\ 0 & 1 & -8 & 0 & -4 & 2 & 1 \end{array}\right)$$

$$\sim \left(\begin{array}{ccccc|cc} -1 & -1 & -8 & -2 & -8 & 4 & 7 \\ 0 & 1 & -3 & 0 & 2 & 2 & 6 \\ 0 & 0 & 6 & 0 & 7 & 0 & -8 \\ 0 & 0 & 4 & 1 & 2 & 3 & 0 \\ 0 & 0 & 5 & 0 & 11 & -2 & -9 \end{array}\right) \sim \left(\begin{array}{ccccc|cc} -1 & -1 & -8 & -2 & -8 & 4 & 7 \\ 0 & 1 & -3 & 0 & 2 & 2 & 6 \\ 0 & 0 & 6 & 0 & 7 & 0 & -8 \\ 0 & 0 & 0 & -6 & 16 & -18 & -32 \\ 0 & 0 & 0 & 0 & -31 & -12 & 14 \end{array}\right)$$

$-31 x_5 = -12 \rightarrow x_5 = \frac{12}{31}$

$-6x_4 + 14x_5 = -18$

$-6x_4 + \frac{14 \cdot 12}{31}$

$x_4 = \frac{14 \cdot 12^2}{31} \cdot \frac{1}{6} = \frac{28}{31}$

$6x_3 + 7x_5 = 0$

$6x_3 = -\frac{12}{31}$

$x_3 = -\frac{2}{31}$

$1x_2 - 3x_3 + 2x_5 = 2$

$x_2 - 3 \cdot (-\frac{2}{31}) + 2 \cdot \frac{12}{31} = 2$

$x_2 + \frac{6}{31} + \frac{24}{31}$

$40 \qquad 54$

$-8 \cdot 5 + (+9) \cdot 6$

$-2 \cdot 6 +$

$-$

$-12 + 4 \cdot 7 =$

$-12 + 28 = 16$

$-8 \cdot 4 = -32$

$-66 + 35 = -31$

$det(R) =$

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

$$\sim \begin{pmatrix} 1 & 1 & 8 & 2 & 8 \\ 0 & 1 & -3 & 0 & 2 \\ 0 & 0 & 6 & 0 & 3 \\ 0 & 0 & 0 & 3 & 21 \\ 0 & 0 & 0 & 5 & 6 \end{pmatrix} \sim \begin{pmatrix} 1 & 1 & 8 & 2 & 8 \\ 0 & 1 & -3 & 0 & 2 \\ 0 & 0 & 6 & 0 & 3 \\ 0 & 0 & 0 & 1 & 7 \\ 0 & 0 & 0 & 5 & 6 \end{pmatrix} \sim \begin{pmatrix} 1 & 1 & 8 & 2 & 8 \\ 0 & 1 & -3 & 0 & 2 \\ 0 & 0 & 6 & 0 & 3 \\ 0 & 0 & 0 & 1 & 7 \\ 0 & 0 & 0 & 0 & -29 \end{pmatrix} \sim$$

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

$det R = 6$

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

$$-1x_1 - 1x_2 - 8x_3 - 2x_4 - 8x_5 = 7$$

$$x_2 - 3x_3$$

$\pi_{11}\ q_i$

(2)

Kód studenta 3

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.)

a) $\varphi_1 = \exists x \, (Z(x) \,\&\, \neg S(x)) \,\&\, (S(y) \Rightarrow Z(y))$

b) $\varphi_2 = \forall x \, P(x) \Rightarrow S(x)$

c) $\varphi_3 = \exists x : (P(x) \,\&\, \neg S(x))$

1) struktura jazyka $\mathcal{A} = \langle A, R, F \rangle$, kde A jsou proměnné $x, y, \ldots$, R jsou relační symboly a F jsou funkční symboly, můžeme ji nazývat model $M(P)$

Model teorie T je struktura $\mathcal{A}$ taková, že $T \models A$, která platí v teorii T.

T platí v A

co to znamená?

kód studenta 3

$T((\exists x)(Z(x) \& \neg S(x)) \& (\forall y)(S(y) \Rightarrow Z(y)))$

$\vdash P(x) \Rightarrow S(x)$

Otázka 4.

Převedeme do PNF

γ_2: $P(x) \Rightarrow S(x) \sim \neg P(x) \vee S(x)$

γ_1: $\exists x (Z(x) \& \neg S(x)) \& (\forall y (S(y) \Rightarrow Z(y)) \sim (\exists x)(Z(x) \& \neg S(x)) \& (\neg S(y) \vee Z(y))$

γ_3: $\exists x (P(x) \& \neg Z(x))$

Skolemizace:

γ_1: $(\exists x)(Z(x) \& \neg S(x)) \& (\neg S(y) \vee Z(y)) \sim$

$\sim (Z(c) \& \neg S(c)) \& (\neg S(y) \vee Z(y))$ kde c je nový konstantní symbol

γ_2: $\neg P(x) \vee S(x)$

γ_3: $P(d) \& \neg Z(d)$ kde d je nový konstantní symbol

$T_1 = \{\gamma_1, \gamma_2\} = \{\{Z(c)\}, \{\neg S(c)\}, \{\neg S(y), Z(y)\}, \{\neg P(d), S(d)\}\}$?

důkaz rezolucí

c/y

$\neg S(c), Z(c)$

d/c

$\neg P(c)$

→ ~~model~~ musí platit $Z(c)$, $\neg S(c)$, $\neg P(d)$?

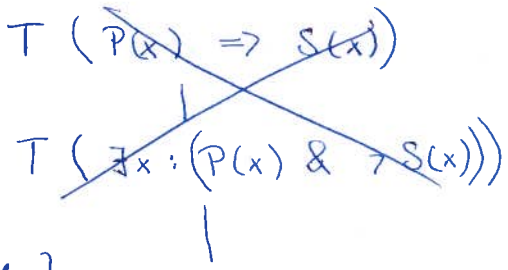
model: (c, d), kde platí:

$Z(c)$ – c složil zkoušky

$\neg S(c)$ – c neměl štěstí

$\neg P(d)$ – d nebyl připraven

to není struktura



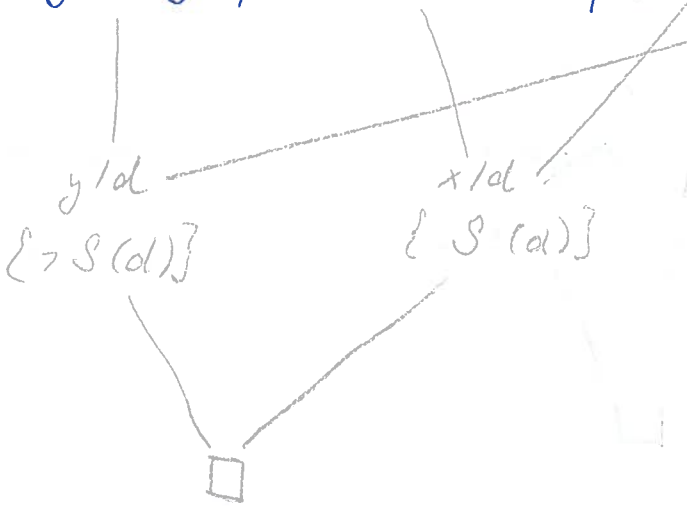
$T_2 = \{\varphi_1, \varphi_2, \varphi_3\}$

φ_1: $(Z(c) \;\&\; \neg S(c)) \;\&\; (\neg S(y) \vee Z(y))$

φ_2: $\neg P(x) \vee S(x)$

φ_3: $P(a) \;\&\; \neg Z(a)$

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



T_2 nemá žádný model.

3

kop

Kód studenta 3

2 počet listů odpovědi (pokud více než tento jeden)

5 Datový model (specializace WDOP)

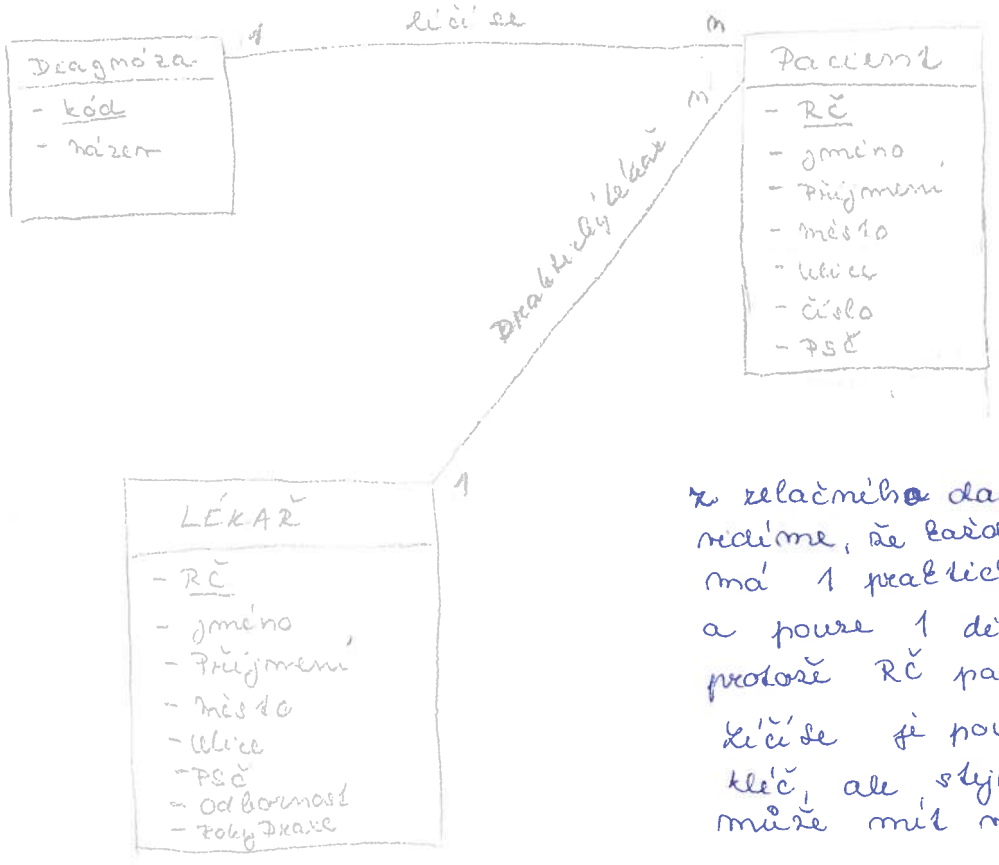
V informačním systému nemocnice je použit následující logický relační datový model, kde podtržením jsou vyznačeny klíče a italikou cizí klíče:

- 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,
   - každý pacient mohl podstoupit libovolný počet vyšetření v rámci nějaké diagnózy a naopak,
   - pacient, který je zároveň lékařem, neměl osobní údaje evidovány redundantně.

primární klíče

1)



z relačního dat. modelu vidíme, že každý pacient má 1 praktického lékaře, a pouze 1 diagnózu, v tabulce ~~tabulce~~ prostože RČ pacienta se léčíše je použito jako primární klíč, ale stejnou diagnózu může mít více pacientů

Otázka 5. kód studenta 3

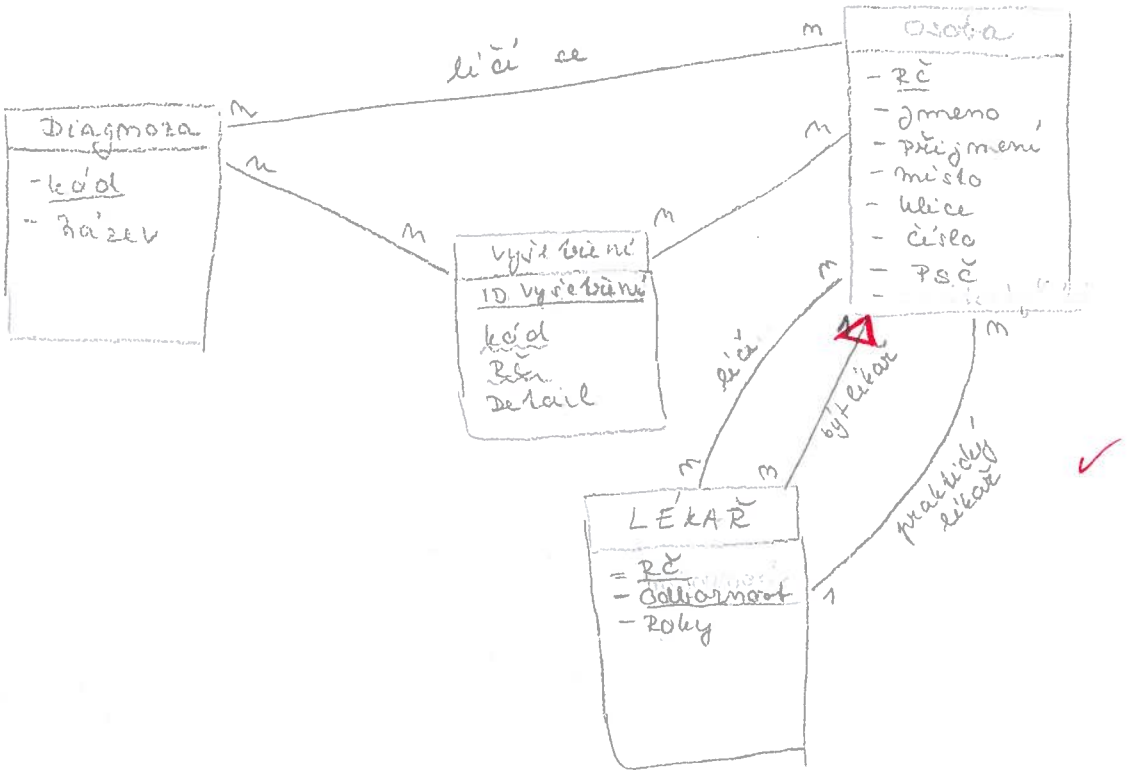
2) a) každý pacient může mít libovolný počet diagnoz
→ v relačním modelu, změníme primární klíč
v ~~tab~~ Léčí se na RČ a kód, každý záznam
je tedy určen složeným primárním klíčem
✓ Léčí se (RČ, kód), kde RČ ⊆ Pacient [RČ]
kód ⊆ Diagnóza [kód]

b) každý pacient má 1 praktického lékaře
ale libovolný počet dalších lékařů
~~tab~~ → do relačního modelu musíme přidat vztah,
✓ Léčí (RČ lékař, RČ Pacient), kde RČ lékař ⊆ Lékař [RČ]
RČ Pacient ⊆ Pacient [RČ]

c) každý pacient mohl podstoupit libovolný počet vyšetření
v rámci nějaké diagnózy a naopak
→ přidáme vztah vyšetření mezi pacientem
a diagnozou Vyšetření (RČ, kód, ~~Typ~~ ID Vyšetření, Detail vyšetření)
✓ RČ ⊆ Pacient [RČ]
kód ⊆ Diagnoza [kod]

d) pacient, který je zároveň lékařem nemíl evidovány
osobní údaje redundantně
→ změníme režimu modelu, vytvoříme vztah
Osoba (RČ, Jméno, Příjmení, Město, Ulice, PSČ, Praktický lékař RČ
Praktický lékař RČ ⊆ Lékař [RČ], Praktický lékař RČ ≠ RČ
Lékař (RČ, Odbornost, Roky Praxe),
✓ RČ ⊆ Osoba [RČ]
každý lékař může být odborník
nejvíce ve 2 oblastech, proto ho
~~→ v konkrétním modelu podle~~ je identifikujeme pomocí RČ a odbor
Léčí (~~RČ Osoba~~ RČ Pacient, RČ lékař) RČ Pacient ⊆ ~~RČ~~ Osoba [RČ]
RČ lékař ⊆ Osoba [RČ]
RČ Pacient ≠ RČ Lékař
→ zároveň jsme přidali podmínku, že lékař nesmí léčit
sám sebe, (přijde mi to neetické)
→ musíme změnit i Léčí (RČ, kód) RČ ⊆ Osoba [RČ]
kód ⊆ Diagnoza
a Vyšetření (~~Detail~~ ID Vyšetření, kód, RČ, Detail vyšetření)
kód ⊆ Diagnoza [kod] ✓
RČ ⊆ Osoba [RČ]

Změněný UML model



2 - kop

Kód studenta 3

2 počet listů odpovědi (pokud více než tento jeden)

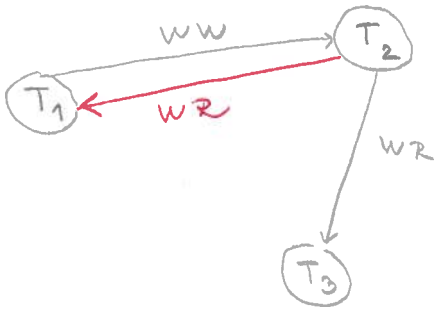
6 Transakce (specializace WDOP)

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 je konfliktově uspořádatelný pokud jeho precedenční graf neobsahuje žádný cyklus.



přidáme W(Pacient$_2$) poté náš graf obsahuje cyklus, tedy není konfliktově uspořádatelný ✓

2) opět W(Pacient$_2$) protože pak budeme požadovat aby byl COMMIT ~~T_1 před COMMIT T_2~~ aktuálně požadujeme aby byl COMMIT T_2 před COMMIT T_1 protože T_1 čteme z Pacient$_2$ do kterého předtím T_2 zapisoval, požadujeme tedy aby se pro

kód studenta 3

6. otázka

2) aktuálně potřebujeme aby COMMIT T_1 proběhl před COMMITem T_2 protože v T_2 přepisujeme data co jsme zapsali v T_1 a potřebujeme, aby byla zachována konzistence databáze i při chybě v T_1

→ přidáním $W(Pacient_2)$ tedy docílíme toho, že T_1 musí mít COMMIT před T_2 a T_2 musí mít ~~ke~~ COMMIT před T_1 což nelze, rozvrh nebude zotavitelný

ne, rozvrh bude konfliktově neuspořádatelný ale nebude obsahovat žádné nepotřebné čtení, které by mohlo T_2 COMMITovat až po T_1

nutno přidat $R(Pacient_1)$

Kód studenta 3

2+

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.

1) (pracuji s původním nijak neupraveným modelem z otázky 5)

a) získání pacientů se jménem odpovídajícím patternu
GET /pacienti/pacient/{jmeno = $pattern} → bez pacient? jm< >
(za rovná se je uveden pattern) nebo pacient/<jm>
vrací odpovídající pacienty jako JSON

b) získání informací o konkrétním pacientovi
GET /pacienti/pacient/{rc = $id}
vrací JSON konkrétního pacienta

c) vytvořit, aktualizovat, zrušit pacienta
POST /pacienti/pacient se vstupem: JSON nového pacienta
POST /pacienti/pacient/{rc = $id} se vstupem JSON s upravenými údaji a daným id pacienta
DELETE /pacienti/pacient/{rc = $id}

d) získat seznam diagnoz pacienta
GET /pacienti/pacient/{rc = $id} /diagnoza

2)

```
async function displayPacients (queryPattern)

    document.getElementById("btnDisplayPacients").addEventListener("click", {
        clearPacients("seznamPacientu");
        var resultPacients = await fetch("/pacienti/pacient/jmeno=${queryPattern}");
        var Pacients = await resultPacients.json();
        var seznam = document.getElementById("seznamPacientu");
        for (let i = 0; i < pacients.length; i++)
        {
            // pro pacienty zobrazujeme jen jmeno a prijmeni,
               pro přehlednost, kvůli zabezpečení osobních údajů neukazujeme RODNÉ ČÍSLO
            var li = document.createElement(li);
            li.id = pacients[i].rc;
            li.innerHTML = pacients[i].jmeno + " " + pacients[i].prijmeni
            li.addEventListener("click", async() => {
                var doctorsResult = await fetch("/pacienti/pacient/{rc=${pacients[i].rc}}/lekari");
                var doctors = await doctorsResult.json();
                var diagnoseResult = await fetch("/pacienti/pacient/{rc=${pacients[i].rc}/diagnoza}");
                var diagnose = await diagnoseResult.json();
            * pokračování na str. 4.

function clearPacients (idSeznam)        HTML <ul> se kterým pracujeme
    var seznam = document.getElementById(idSeznam);
    seznam.innerHTML = "";
}
        vyčistí původní seznam pacientů
```

/pacienti/pacient/{jmeno=$pattern} vrací JSON ve formátu:

```
{
    pacienti : [
        {
            "rc" : 2,
            "jmeno" : "Pavel",
        }
        {"rc" : 3,
        ...
    ]
}
```

Otázka 7. kód studenta 3

e) přidat, odebrat diagnozu

POST /pacienti/pacient/{rc=$id}/diagnoza s JSONEM obsahující danou diagnozu

DELETE /pacienti/pacient/{rc=$id}/diagnoza

podle původního modelu mohl mít 1 pacient jen 1 diagnozu, v případě, že by diagnoz mohl mít více, uvedeme kod diagnozy

f) získat seznam lékařů pacienta

GET /pacienti/pacient/{rc=$id}/lékaři

g) získat seznam pacientu, kteří mají alespoň 1 ze zadaných diagnoz

GET /pacienti/pacient/diagnoza/{kod = $kod}

2

↑ to je ale jenom 1 diagnóza
je třeba jich zadat více potenciálně hodně
⇓
POST

Ukázka 7.

Kód Studenta 3

```
*
            clearDoctors();
            clearDiagnoses();
            displayDoctors();
            displayDiagnoses();
        }
        seznam.addElement(li);
    }
}
```

} → pokud to dolním zde, tak k tomu, že po každé query pattern bude mít seznam pacientů, zdvojí vyselectovaného a přitom naplněný seznam doktorů a diagnóz z předchozích "dělitelů" na prázdno

```
data diagnoz:
  diagnozy: [
    {
      "nazev": "chřipka",
      ⋮
    }
    {
      "nazev": "malárie",
      ⋮
    }
    ⋮
  ]
}
```

```
data doktoru
  lékaru: [
    {
      "rc": 2,
      "jmeno": "martin"
      ⋮
    }
    {
      ⋮
    }
    ⋮
  ]}
```

Kód studenta 3

2—

☐ 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?

1) booleovský model by byl reprezentován vektory obsahujícími 0/1 (pro ~~nemá~~ nemá symptom/má symptom), slovník by tedy byl reprezentován vždy, pro každou nemoc bychom měli vektor který by obsahoval všechny existující příznaky a u příznaků ~~které~~ které by patřili ke konkrétní nemoci by byla 1 a jinak 0.

kašel rýma teplota bolest hlava
např. chřipka: [1 , 1 , 1 , 1 , ...]

→ ~~dotazy reprezentuji~~
dotazy i dokumenty reprezentujeme vektory obsahujícími daní symptomy
a dotazujeme se pomocí logiky ~~nebo podobnosti vektorů~~

efektivní dotazování – inverted seznam nebo bit-like queries (pro AND)

2) můžeme použít reprezentaci celého textu, který nejprve očistíme od často používaných slov bez významu (a, ale, aby ...) a slovům, která jsou pro nás naopak důležitá můžeme zvýšit váhu o nějakou konstantu, dotazování pak bude fungovat přes podobnost vektorů

proč? → booleovský model

vektorový model