



3 body



Kód studenta 22

7 Kódy (otázka studijního zaměření – 3 body)

Definujte pojem minimální vzdálenost pro samooprávné kódy.

Uvažme binární lineární kód $C \subseteq \mathbb{Z}_2^8$ generovaný slovy, která mají právě 2 jedničky na sudých pozicích a právě 2 jedničky na lichých pozicích, tedy

$$C = \text{span}\{(x_1, \dots, x_8) \in \mathbb{Z}_2^8 : x_1 + x_3 + x_5 + x_7 = 2, x_2 + x_4 + x_6 + x_8 = 2\}.$$

(Uvedené sčítání jednotlivých prvků kódového slova je bráno v celých číslech.)

Určete parametry tohoto kódu (délka, velikost, vzdálenost) a sestrojte jeho kontrolní matici.

Délka kódu, pak jeho min. vzd. $\Delta(C) = \min_{x \neq y \in C} d(x, y)$,
 kde $d(x, y)$ je Hammingova vzdálenost, tj. #pozic, kde se $x \neq y$ lze x, y jsou nějaké řádky nebo řádky abecedy.

Zadaný kód C je lineární, tedy $\Delta(C) = \min_{x \in C} d(x, 0)$,
 tj. minimální počet nenulových pozic prvků C .

Kód prvek C vznikne součtem několika generátorů. Na které pozici má 1 \Leftrightarrow lichý počet gen. ji tam měl. Zdrojem plakr, že součet počtu 1 na lichých pozicích i na sudých pozicích je sudý. To znamená (pro nenulový prvek), že obsahuje zálespon dvě jedničky.

Na druhou stranu $10100000 + 01010000 = 11110000$

Tedy $\Delta(C) = 2$. Dle definice $\Delta(C) = \min_{x \in C} d(x, 0)$
 a velikost je $|C| = 2^{\dim C}$. Protože C je rekt. prostor, stává namájet bázi C .



Ukážeme, že bází C tvoří vektory se dvěma jedničkami, které jsou buď na první a jedné další liché, nebo na druhé a jedné další sudé pozici, tj. $\begin{cases} (10100000), \\ (10001000), \\ (10000001) \end{cases}$ v $\begin{cases} (01010000), \\ (01000100), \\ (01000001) \end{cases}$. (pozor na "litchy" a "sude")

$\xrightarrow{\text{Episyn je do souboru vložen}}$
 $\xrightarrow{\text{v tom retej}}$
 $\xrightarrow{\text{(pisyn je pod sebe, aby se v tom nacházelo.)}}$

Všimněme si, že $B \subseteq C$: chceme-li dostat na vektor s jedničkou na pozicích rekneme 1 a ; liche), vložit vektor s jedničkami na pozicích 1, 2, 4, i, a 2, 4, i, j; kde j je liche a $j \notin \{1, 3\}$. Analogicky se to udělá pro sudé.

Nyní si všimněme, že B generuje C. Na to stačí aby generovala zadání generátory C. A na to stačí aby generovala všechny vektory s právě dvěma jedničkami, oběma buď na liche, nebo na sudé pozici (potom se setkou pravděpodobnosti liche a „sudé“ vektory). Rekneme, že chceme dostat vektor s jedničkami na pozicích i, j, i, j, liche. Pokud $i=1$, je prýmo v bázi B. Jinak seťeme balzové vektory s dvěma jedničkami na pozici i resp. j. Pro sudé analogicky.

Nakonec dokážeme, že B je lin. rez. Předpokládejme, že máme několik nulařů součet prvků z B. Když jsme v \mathbb{Z}_2 , tj. koeficienty jsou 0 a 1. Je několik, čili jistěm alespoň jeden prvek, bude liche. Když ve výsledku je na první pozici 0, čili jsou tam právě dva ze tří lichých vektorů z B. A to je srovnání spor s nulařem součtu. $\dim(C) = |B| = 6$, tj. $|C| = 2^6$. Kontrolní matici je treba

$$K = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} : \text{V prvním řádku kontrolujeme paritu lichých pozic, ve druhém sudých.}$$

Zajíždí $\dim(K) = 2 = 8 - \dim(C)$ a jednoduše ověříme $C = \text{Ker } K$, protože $Kb = 0 \forall b \in B$, a fakty K je skutečně kod. matice.



3 body



Kód studenta 22

9 Množinové systémy (otázka studijního zaměření – 3 body)

1. Nechť A je systém podmnožin množiny $\{1, \dots, n\}$ takový, že každé dvě množiny v A mají neprázdný průnik. Ukažte, že $|A| \leq 2^{n-1}$.

2. Najděte systém podmnožin množiny $\{1, \dots, n\}$ velikosti 2^{n-1} takový, že každé dvě množiny v A mají neprázdný průnik.

② Stav vztah $F = \{A \subseteq \{n\} \mid 1 \in A\}$. Dostatečné $\forall A, B \in F: 1 \in A \cap B$, tj. $A \cap B \neq \emptyset$. A $|F| = |\mathcal{P}(\{n\} \setminus \{1\})| = 2^{n-1}$, jde o klasické sítě.

① Označme $B = \mathcal{P}(\{n\}) \setminus A$, tj. všechny podmn. $\{1, \dots, n\}$, které nejsou $\in A$. Najdeme prostor zobrazení $A \rightarrow B$, z čehož bude plynout $|A| \leq |B|$ (viz úloha 8).
 $A \cap B = \emptyset$, Tak $2^n = |\mathcal{P}(\{n\})| \stackrel{(*)}{=} |A \cup B| = |A| + |B| \geq 2|A|$, tedy $|A| \leq 2^{n-1}$.
 Zobrazení $\xrightarrow{\text{jednoduché}} A \rightarrow B$ je definováno posláním každé množiny na její doplněk. Jelikož doplněk je involuce (tj. $A = (A^c)^c$), tak pokud $A = B$ (tj. $x^c = y^c$, tedy $x = y$), tedy fólie zobrazení je skutečně prostor. Nově je-li $x \in A$, potom $x^c \notin A$, tj. $x^c \in B$, jelikož $x \cap x^c = \emptyset$. Tedy doplněk $\in \{n\}$ je hledaný prostor zobrazení $A \rightarrow \mathcal{P}(\{n\}) \setminus A$.



Kód studenta 22



8 Mohutnost množin (otázka studijního zaměření – 3 body)

(3b)

- ① Napište definici "Množina x má menší mohutnost než množina y ".
- ② Dokažte, že pro každou množinu x platí, že x má menší mohutnost než potenční množina x (v teorii ZF, bez použití axiomu výběru).

① x má menší mohutnost než y , znamená třeba $x \not\leq y$, pokud existuje prosté zobrazení $f: x \rightarrow y$, tj. $\forall f \subseteq x \times y$ takové, že $(\forall a \in x)(\exists! b \in y)((a, b) \in f) \wedge (\forall a \in x)(\forall b \in x)(f(a) = f(b) \Rightarrow a = b)$.

x má (osobně) menší mohutnost než y , pokud $x \not\leq y$ a $y \not\leq x$ ~~což je rozdíl mezi mohutnostmi~~

② Jistě $x \in P(x)$, vezměme fci $x \mapsto \{x\}$.
Pro spor předpokládejme, že existuje prosté $f: P(x) \rightarrow x$, tj. že schématu axioma vydělení (2) existuje množina

$$M = \{f(A) \mid A \in P(x) \wedge f(A) \notin A\}.$$

HODÍLO BY SE
ZDÍJETI POUŽITÍ PRVOSTY $P(x) \not\in x$.

Jistě $M \in P(x)$. Podívejme se nyní na $f(M)$. Když by

$f(M) \in M$, tak by to byl spor s definicí M .

Dále $f(M) \notin M$, jenže k z definice M implikuje $f(M) \in M \rightarrow \perp$. Tedy f nemůže existovat, proto $P(x) \not\leq x$, a tedy $x \not\leq P(x)$.



Kód studenta 22

6 Optimalizační metody (3 body)

Nechť $Ax \leq b$ je systém lineárních nerovnic o n proměnných. Vynásobením každé nerovnosti kladnou konstantou můžeme docílit, že první sloupec matice A je vektor obsahující pouze složky 0, -1 and 1. Systém $Ax \leq b$ můžeme tudíž ekvivalentně zapsat jako

$$\begin{aligned} a'_i x' &\leq b_i \quad (i = 1, \dots, m_1), \\ -x_1 + a'_j x' &\leq b_j \quad (j = m_1 + 1, \dots, m_2), \\ x_1 + a'_k x' &\leq b_k \quad (k = m_2 + 1, \dots, m), \end{aligned} \quad (*)$$

krok konstanty $\leftarrow b_i$ je toto pravidlo bude

(1) kde $x' = (x_2, \dots, x_n)$ a kde a'_1, \dots, a'_n jsou řádky A bez první složky. Pak se můžeme zbavit x_1 : dokažte, že systém $Ax \leq b$ má řešení právě když systém

$$\begin{aligned} a'_i x' &\leq b_i \quad (i = 1, \dots, m_1), \\ a'_j x' - b_j &\leq b_k - a'_k x' \quad (j = m_1 + 1, \dots, m_2, k = m_2 + 1, \dots, m) \end{aligned} \quad (***)$$

(2) má řešení. Ukažte, že opakováním použitím tohoto kroku je možné vyřešit systém lineárních nerovnic (nebo dokázat, že systém nemá řešení). Jaká bude časová složitost takového postupu?

(1) Pokud $Ax \leq b$ má řešení, pak jistě i $(*)$ má řešení - totožně x . Pokud $(*)$ má řešení, tak i $(***)$ má řešení, jelikož $(***)$ vznikne z $(*)$ seřazením nerovnic (a jistě po nich pokud $a'_i u \leq v \wedge a'_j w \leq z$, tak $i u+w \leq v+z$) a úpravou. Naopak pokud $(***)$ má řešení x' , tak pravidlo znadíme zvolíme $x_1 = a'_j x' - b_j$. Potom jistě $x_1 + a'_k x' \leq b_k$, jelikož levá strana se rovná b_k .
 ~~$x_1 + a'_j x' \leq b_j$, jelikož levá strana se rovná b_j .~~
~~A z $(***)$ vzní, že $x_1 - a'_j - b_j \leq b_k - a'_k x'$, tedy $x_1 + a'_k x' \leq b_k$, což je chybou.~~
zvolíme $x_1 = \min_{m_2+1 \leq k \leq m} (b_k - a'_k x')$. Z definice pro všechna $m_2+1 \leq k \leq m$ platí $x_1 = \min_k (b_k - a'_k x') \leq b_k - a'_k x'$, tedy pro všechna rounic v $(*)$ má řešení. První řada má řešení $x^{(x_1+1)}_{(x_1+1)}$ triviálně ✓

Druhá sada rovnic specifikuje, že $k_{m_1+1} \leq j \leq m_2$

$$a_j^T x^* - b_j \leq \min_k (b_k - a_k^T x^*) \quad (\text{ještě tady})$$

Tedy $a_j^T x^* - b_j \leq x_1^* \leq x_1$, tedy $-x_1 + a_j^T x^* \leq b_j \leq k_j$,
takže jsou všechny.

D

② Myžíme soustavu m rovnic o n neznámých.

Přímý souborovým řešením $O(mn)$ času

→ do dalšího kroku nejsou posíleny

$$\begin{array}{c} (m_0 + m_1 + \dots + m_n) \\ \text{různice o n-1 neznámých} \\ \text{kde } \\ \text{koef. } -x_1 \\ = 0 \end{array}$$

$\#x_1 \quad \#-x_1$
 $\# \# + x_1$

za každou dvojici využívají různost.

Když se dostaneme na 0 neznámých, tak budeme mít
jen nějaké různosti porovnávající konstanty b_i , tj.
bude platit, anebu ne → můžeme pak odpropagovat na začátku
→ zjistíme (najdeme), zda $Ax \leq b$ má řešení. V každém
kroku řešíme lineární průsečík (vzhledem ke vstupu), pokud
vystřídeme rovnice pro další krok nainicujeme jinou.

Platí, že pokud $x, y > 0$, $x+y$ je parní, tak mat $x+y$ základ pro
pro $x=y$. Analogicky si rozmyslíme, že v nejhorském případě $m_0=0$,
tj. mítme, že zároveň lze m rovnici o n neznámých,
dalším kroku dostaneme $\leq \frac{m^2}{4}$ rovnic o n-1 neznámých, t.j.
stručně $O(mn) + O\left(\frac{m^2}{4}(n-1)\right) + O\left(\frac{m^4}{76}(n-2)\right) + \dots + O\left(\frac{m^{2i}}{4^{2i+1}}(n-i)\right) + \dots + O\left(\frac{m^{2n}}{4^{2n+1}}\right)$
takže "zároveň" m, dostaneme
 ~~$\min O\left(\sum_{i=0}^{n-1} O\left(mn \sum_{j=0}^i \left(\frac{m}{4}\right)^{2j}\right)\right)$~~ \rightarrow dojde papír

Kód studente 22

6 Opt. metody

List 2/2

Víme, že užíváme n kroků, v každém

podproblém v i-tém z nich bude mít velikost

$$4 \frac{m^{2^i}}{4^{2^i}} \quad \left(\text{indexy jsou od nuly až po } \frac{m^{2^i}-1}{4^{2^i-1}} \text{ indukce} \right)$$

(rozbalíme vztah $\left(\frac{m^{2^i}}{4^{2^i-1}}\right)^2 / 4 = \frac{m^{2^{i+1}}}{4^{2^{i+1}-1}}$)

Pro $m > 4$ je toto rostoucí funkce, nejdříve něnic fády budeme mít v posledním kroku, konkrétně $4\left(\frac{m}{4}\right)^{2^n}$

(horní odhad). Hodně hrubé návizení avšak, že v každém z n kroků jste užíváli $O(n\left(\frac{m}{4}\right)^{2^n})$ práce, tj. složitost je $O\left(n^2\left(\frac{m}{4}\right)^{2^n}\right)$ (alespoň $\Omega\left(\left(\frac{m}{4}\right)^{2^n}\right)$), což je drojíde exponenciální v n , tj. opravdu návizení, a tak se mi ani nechce počítat, jaký polynom koeficient tam užije.

Právěpodobně jsem ale někde udělal chybu nebo něco přehledl, třeba složitost byla vypočtena opravdu návizení (i když je makorec sepsaný dle této struktury), mnohem víc, než bych u státnic řekal.

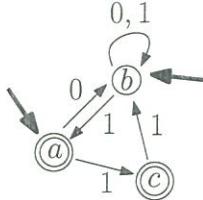




Kód studenta 22

5 Automaty (3 body)

1. Převeďte nedeterministický konečný automat $A = (\{a, b, c\}, \{0, 1\}, \delta, \{a, b\}, \{a, c\})$ z obrázku na ekvivalentní deterministický automat.
2. Do jakých tříd jazyků v Chomského hierarchii patří jazyk $L(A)$?



② $L(A)$ je jazyk rozpoznatelný (R)M konečným automatem, tj. je regulární. Obj. znacení 2, jsem - li rek. spoj. jazyky uveden 0) (1... kontek., 2... bezkontek.). Tedy je samořejmě i bezkontextový, kontextový a nekonzistentní spočetný.

③ Použijeme podmnožinovou konstrukci; nezakreslené řípky vedou do fail stavu značeného \emptyset .

$$B = \left(P(\{a, b, c\}), \{0, 1\}, \delta', \{\{a, b, c\}\}, P(\{a, b, c\}) \setminus \{\{b\}, \emptyset\} \right) \text{ slouží}$$

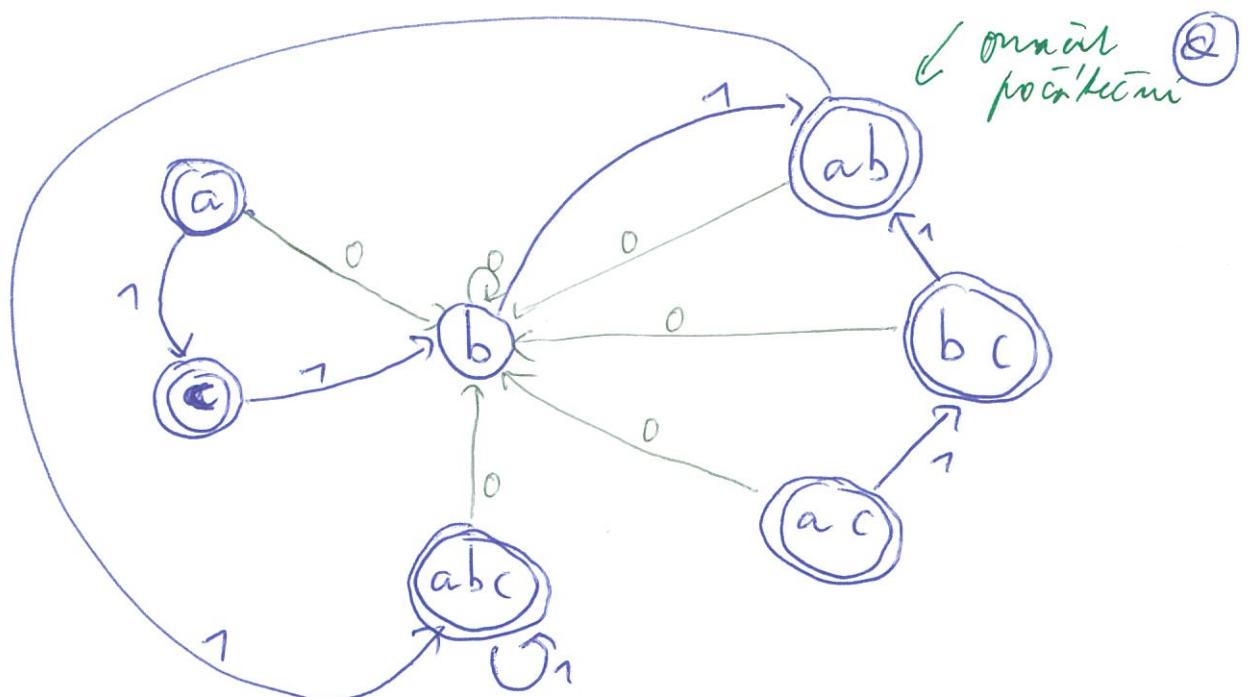
$$\delta'(a, 0) = b \quad \delta'(a, 1) = c \quad \delta'(b, 0) = b \quad \delta'(b, 1) = ab$$

$$\delta'(c, 0) = \emptyset \quad \delta'(c, 1) = b \quad \delta'(ab, 0) = b \quad \delta'(ab, 1) = abc$$

$$\delta'(ac, 0) = b \quad \delta'(ac, 1) = bc \quad \delta'(bc, 0) = b \quad \delta'(bc, 1) = ab$$

$$\delta'(abc, 0) = b \quad \delta'(abc, 1) = abc \quad \delta'(\emptyset, 0) = \delta'(\emptyset, 1) = \emptyset$$

obr. na druhý
straně



3b
čB



Kód studenta 22

3 Objektový interface pro REST API server (3 body)

Mějme specializovaný web server pro REST API implementovaný v mainstreamovém objektově orientovaném staticky typovaném jazyce (C++ C# nebo Java) s následujícími vlastnostmi. Server umožňuje, aby se do něho dynamicky vkládaly služby, přičemž služba je objekt třídy implementující rozhraní `IService`, který zapozdruje několik REST metod. Služby jsou identifikovány řetězci, které musí být unikátní v rámci serveru, REST metody jsou rovněž identifikovány řetězci, které musí být unikátní v rámci dané služby. Součástí rozhraní třídy `IService` je i schopnost publikovat seznam svých REST metod. Souvislost mezi REST metodami a HTTP metodou volání (GET, POST) neřešíme, v dalším kontextu je metoda bez přívlastku chápána jako členská funkce třídy.

Zpracování požadavku přes REST API probíhá tak, že server přečte URL a další data z HTTP protokolu a z nich dekóduje identifikátor služby, identifikátor REST metody a kolekci parametrů. Dle identifikátorů najde ve vnitřních záznamech požadovanou službu a na ní zavolá metodu implementující požadovanou REST metodu, přičemž metoda dostane kolekci parametrů jako vstupní argument a vrátí řetězec. Pro naše účely si představme kolekci parametrů jako mapu/slovík (konkrétní typ si upravte dle zvoleného jazyka), kde klíče jsou řetězce (názvy parametrů) a hodnoty jsou objekty typu `ParameterValue` (další detaily jsou pro naši úlohu nezajímavé).

Server je reprezentován objektem třídy `Server`. Tato třída má (mimo jiné) dvě podstatné metody, které by mohly být symbolicky zapsány přibližně takto (přesný zápis závisí na použitém jazyce):

```
public void add(IService service)  
private string dispatch(string serviceId, string methodId, ParametersCollection parameters)
```

Metoda `add` zaregistrouje novou službu v rámci serveru. Tato metoda může být relativně pomalá a volá se typicky jen při startu aplikace (např. když jsou načítány zásvuné moduly). Metoda `dispatch` najde a zavolá cílovou metodu příslušné služby a je volána výhradně z vnitřní implementace serveru při zpracování požadavku ze sítě. Metoda `dispatch` by neměla mít velký overhead.

1. Navrhněte, jak by měl vypadat interface `IService` a stručně (např. komentářem) vysvětlete význam jednotlivých metod (pokud není naprostě zřejmý z názvu metody).
2. Představte příklad konkrétní třídy implementující rozhraní `IService`. Zaměřte se zejména na realizaci části rozhraní zodpovědné za předání informací o nabízených metodách.
3. Stručně (případně v pseudokódu) popište, jak bude vypadat implementace metod `add` a `dispatch` třídy `Server`. Pokud tyto metody potřebují přistupovat ke členským proměnným třídy, popište tyto proměnné také. Připomeňme, že metoda `dispatch` by měla být rozumně efektivní.

Drobné chyby v syntaxi budou tolerovány, avšak celkový návrh a logika rozhraní by měly obecně odpovídat principům a zvyklostem zvoleného jazyka. V jazycích, které to umožňují, je povoleno rozumné použití reflexe, nicméně reflexe není nutně vyžadována.

1. `// Funkce get-methods vrátí mapu klíč=Lambda, která způsobí volání vlastní metody. Pojde mi to rychleji než páčivo pojítory na fce.`

```
class IService {  
public:  
    virtual ~IService(); // abstr. třída -> virtual /n, / dest.  
    virtual map<string, function<string(ParametersCollection)>> get-methods();  
    string serviceId; // můžou zajistit unikátnost metody
```

3. `typedef map<string, ParameterValue> ParametersCollection;` →

②

```

using namespace std;
class Namaluj : public virtual Service {
private: using PC = ParametersCollection;
    string sloboda(const PC & parameters);
    string hrocha(const PC & parameters);
public:
    string name serviceId = "NAMALUJ";
    virtual map<...> getMethods() override {
        map<...> ret;
        ret["SLOBA"] = [this](const PC & pars){ return this->sloboda(pars); };
        ret["HROCHY"] = [this](const PC & pars){ return this->hrocha(pars); };
    }
}

```

Výklad

Mohou být Service pro každou metodu mohla poskytnout i seznam povinných/volitelných parametrů. Nebo nabídnout tri, které by dostala několik metod a indexem parametrů mohly být využity, nebo když to v pořádku neplatí, mohlo to dobařit a mohlo vyplývat.

Teď je potřeba upravit referenci, když se this má využít a oddělit se před tím, že je tam ještě bude OK.

return sloboda(ret);

}

|| konstruktor, ...

|| osobně by mi patřilo rozumět, aby se Server nejakej Service rostal o API prostře příjat

③ class Server { || serviceId → mapa funkce lambda. OK
 private: map<string, map<string, function<string(PC)>>> serviceMethods;

|| služby add (map<string, unique_ptr<IService>> services);

string dispatch(...){

|| zkontroluj validitu parametrů (existence služby a metody..)

[return serviceMethods[serviceId][methodId](parameters);]

};

public:

← checeme virtuální metody

void add(unique_ptr<IService> service) {

|| zkontroluj unikátnost serviceId add, kdyžže už máme services

serviceMethods[service->serviceId] = service->getMethods();

services[service->serviceId] = move(service);

}

};

3 body
pékne



Kód studenta 22

4 Souborový systém (3 body)

Tato otázka obsahuje zjednodušený popis souborového systému FAT. Pokud chcete, můžete v odpovědi uvažovat i skutečný souborový systém FAT (pokud ano, výslově to napište).

Oddíl naformátovaný (zjednodušeným) souborovým systémem FAT12/16/32 je rozdelený na 3 části:

1. boot sektor (nultý logický sektor oddílu),
2. tzv. tabulka FAT, viz dále (1. až N. sektor oddílu),
3. datové sektory (N+1. sektor až poslední sektor oddílu), které obsahují samotná data souborů (nebo adresářů, nicméně jelikož ty se z pohledu alokace sektorů od souborů v souborovém systému neliší, tak se jimi dále nebudeme zabývat zvlášť). Pro jednoduchost předpokládejme, že alokační jednotka souborového systému je právě jeden sektor disku. Sektory disku mají jednu z obvyklých velikostí – označme ji jako X bytů.

Každý datový sektor oddílu je přiřazený právě jednomu souboru, nebo je označený jako volný. Každý soubor na disku zabírá určité množství celých sektorů, přičemž ale tyto sektory nemusí být konkrétnímu souboru přiděleny kontinuálně (soubory mohou být na disku fragmentované). V adresářovém záznamu pro konkrétní soubor je zapsáno číslo datového sektoru (číslovány od 1), který obsahuje data prvních X bytů souboru (pro soubory s velikostí menší nebo rovnou X bytů je to zároveň poslední sektor souboru). Pro soubory s velikostí větší než X bytů nalezneme informaci o dalších sektorech přidělených souboru ve FAT tabulce oddílu – pro každý soubor (který zabírá např. M sektorů) obsahuje FAT tabulka „zakódovanou“ obdobu jednosměrně vázaného seznamu posloupnosti čísel 2. až M. sektoru souboru. Přesný obsah FAT tabulky je následující:

Pro každý datový sektor obsahuje FAT tabulka právě jeden Z bitový záznam ($Z = 12$ bitů pro FAT12, 16 bitů pro FAT16, 32 bitů pro FAT32) – tento záznam je bezznaménkové Z bitové celé číslo uložené v pořadí little endian. V tabulce FAT je tedy právě tolik záznamů, kolik oddíl obsahuje datových sektorů, a žádné jiné informace FAT tabulka neobsahuje (FAT tabulka je tedy fakticky jen pole celých čísel). Záznam na offsetu 0 v prvním sektoru FAT tabulky obsahuje informaci o 1. datovém sektoru oddílu, hned za ním (bez paddingu) následuje záznam pro 2. datový sektor oddílu, pak záznam pro 3. datový sektor oddílu, atd. Pokud záznam pro datový sektor A obsahuje číslo 0, tak je tento datový sektor volný a není přidělen žádnému souboru. Pokud záznam pro datový sektor A obsahuje číslo B, tak je datový sektor A přidělený nějakému souboru, a po X bytech dat toho souboru uložených v datovém sektoru A je následujících X bytů souboru uložených v datovém sektoru B (kde pro fragmentované soubory nemusí být B rovno A+1, a obecně může být B být větší i menší než A). Pokud záznam pro datový sektor A obsahuje maximální hodnotu Z bitového čísla, pak je sektor A posledním sektorem nějakého souboru (M. sektorem přiděleným souboru o velikosti M sektorů). Pokud tedy např. data souboru A.TXT budou ležet v datových sektorech 10, 7, 8, 15, tak v adresářovém záznamu pro A.TXT bude zapsáno číslo 10 a ve FAT tabulce bude v záznamu pro sektor 10 uloženo číslo 7, v záznamu pro sektor 7 číslo 8, v záznamu pro sektor 8 číslo 15, a v záznamu pro sektor 15 maximální hodnota Z bitového čísla.

1. Jaká může být obvyklá hodnota X? Pokud je velikost oddílu právě 1 GiB, je vhodné tento oddíl naformátovat souborovým systémem FAT16 nebo FAT32? Vysvětlete proč.
2. V takovém 1 GiB oddílu máme uloženo 100 000 souborů, kde každý má velikost 1 KiB. Pokud nyní chceme přečíst obsah všech těchto souborů, bylo by pro typický disk vhodné načíst si předem celou FAT tabulku do paměti? Nebo by bylo vhodnější před čtením každého ze souborů znova načíst do paměti pouze ty sektory FAT tabulky, o kterých zjistíme, že obsahují informace potřebné pro právě čtený soubor? Vysvětlete proč.
3. Předpokládejte, že v „globální proměnné“ (resp. statické proměnné nějaké třídy) `fat` typu pole bytů máme načtené veškeré záznamy celé FAT tabulky oddílu naformátovaného souborovým systémem FAT12 – jeden záznam FAT tabulky má velikost 12 bitů, tedy každé 3 byty pole `fat` obsahují právě 2 záznamy o 2 datových sektorech (záznam pro sektor s nižším číslem je vždy v 1. bytu a ve spodních 4 bytech 2. bytu; záznam pro sektor s vyšším číslem je ve vyšších 4 bytech 2. bytu a ve 3. bytu). V jazyce C# nebo Java nebo C++ naprogramujte proceduru, která jako svůj jediný argument dostane číslo prvního datového sektoru přiděleného nějakému souboru, a má na standardní výstup vypsat čísla všech datových sektorů tomuto souboru přiřazených (dle informací v proměnné `fat`).

1. Myslím, že jednotky KiB (neplatné 4KiB), FAT tabulka zabere řádky řádky, což je snesitelné a malé soubory nezabírají zase řádky místy navíc. Max. velikost může být max. číslo je $2^7 \cdot 2^{16} \sim 64 \cdot 10^3$ LK $\sim 16\text{GiB} / 4\text{KiB} = \# \text{sektorů}$. Je tedy třeba FAT32, obvyklou už uvedenovali?

② FAT tabulka bude asi obsahovat disku (fj. ~1 MiB).
 Chceme načíst asi 100 MiB (neplatnost správ 600 MiB, soubory jsou malé a sektory na ně zbyt velké).

Výsledek by tedy mělo smysl si stahovat celou tabulkou do paměti a od disku si nechat (burstovat) poslat všechny relevantní sektory v rovném pořadí a že si sladit u sebe v RAM - disk je mnohem pomalejší a navíc mu vyhovuje sekvenční čtení, skákání do FAT a tabulky a zpět by strávily hrozivě moc času.

③ Tady hráje roli little-endian vs. big-endian kódování.
 Je možné, že little-endian známené je $1101 = 1+2+4$ (fj. první bit je kon. lsb.). Ale možné je to naopak, takže pro jistotu napíšu obě varianty. EDIT: vložené nahraje, protože stejně bity tak, jak je doslova, a budou předepsané.
 EDIT2: zde je to sam. LE je pro mě, že první bit je lsb. Takže ve FAT tabulce máme 3 bity $\boxed{x \ 1 \ 0 \ 1}$, x je lsb parná základna, 2 lsb dvojka.
~~uint32_t read_sector(uint32_t i) {~~ // pomocná funkce, vrátila obsah FAT tabulky, // takže potřebuju
 ~~uint32_t fat_i = 3 * (i >> 2);~~ // mít druhou fat základnu začínající // v poli na pozici $3\lfloor\frac{i}{2}\rfloor$
 ~~bool higher = i & 1;~~ // obecně spodní, nebo horní základna // z důvodu trojice?
 ~~uint32_t ret = fat[fat_i];~~ // uint32_t w = fat[fat_i + 1], md = fat[fat_i + 2], hg = fat[fat_i + 3];
 ~~if(!higher) {~~
 ~~ret = (ret & 0b1111) & 0b1111;~~
 ~~ret = (ret << 4) | (fat[fat_i + 1] & 0b1111);~~
 ~~return (w | ((md & 0b1111) << 8))~~
 ~~& spodní bity & posunuto o 8;~~

(3) na dalším papíře.

Úloha 4

(soub. systém) práv 2/2

③ Pokud si plánu LE a BE, tak to rádi zvládneš rovnou. Předpokládat, že LE znamená least-significant bitem. Pro jistotu mohou být implementovány oběma způsoby, ale vždy předpokládat, že programový jazyk (C++) interně reprezentuje čísla stejně jako je možné na disku. Když máme totéž rozložení, tak by se bylo nejjednodušší:

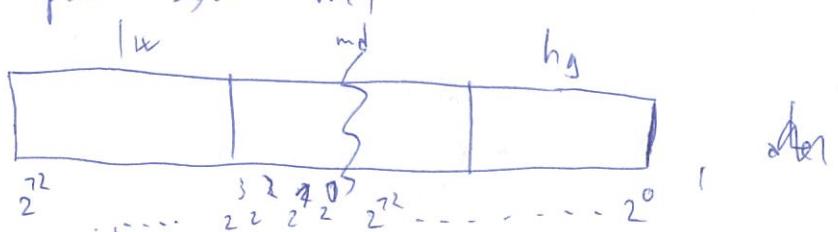
```
void vypis(uint32_t zápisobek) {
    while (i != 0xFFFFFFFF) { // dokud i nejsou samé 1, tj. konec
        cout << i << " "; // ob 111111111111;
        i = next_sector(i); // druhým ještěk, koncový
    }
    cout << endl;
}
```

```
uint32_t next_sector_LSB(uint32_t i) {
    uint32_t fat_i = 3 * (i >> 2); // index nejpr. ze dvou relevantních
    // zápisů pole fat. = 3 * (i / 2)
    bool higher = i & 1; // chceme vysívat zápisom?
    uint32_t l = (w = fat[fat_i]), md = fat[fat_i + 1], hg = fat[fat_i + 2];
    // předpokladejme, že bitové posuny níže neplatí
    if (!higher) { // (w | md | hg)
        return l | ((md & 0b1111) << 8); // využíváme spodní 4 bity md
    } else
        return (md >> 4) | (hg << 4); // deje využíváme posunem
}
```

```
uint32_t next_sector_MSB(uint32_t i) {
    ... if (higher) return (lw << 4) | (md >> 4);
    else return ((md & 0b1111) << 8) | hg;
```



Poznámka: Bojujte s tím, že si nejsou jistí, co je LE vs big endian. Zároveň zadání říká, že ve spodních 4 bitech druhého byte "akč.", spodní charakter jako reprezentující menší čísla. Cesta v tom je, že představují, že mají lsb vpravo (a tedy a >> mají geometrický smysl), zatímco fakty moc nejsou protože je pořad charakter jen násoben možností drojkou. Kdyby LE bylo přirozené, že lsb je vpravo, pak bych měl



akč. fakt je spodní sektor by byl
 $(lw \ll 4) | (md \gg 4)$ a horní sektor
 $((md \& 0b1111) \ll 8) | hg$, přičemž samozřejmě je potřeba delat poskyt a dostatečně velkých beznaménkových rozdílných typů.

35

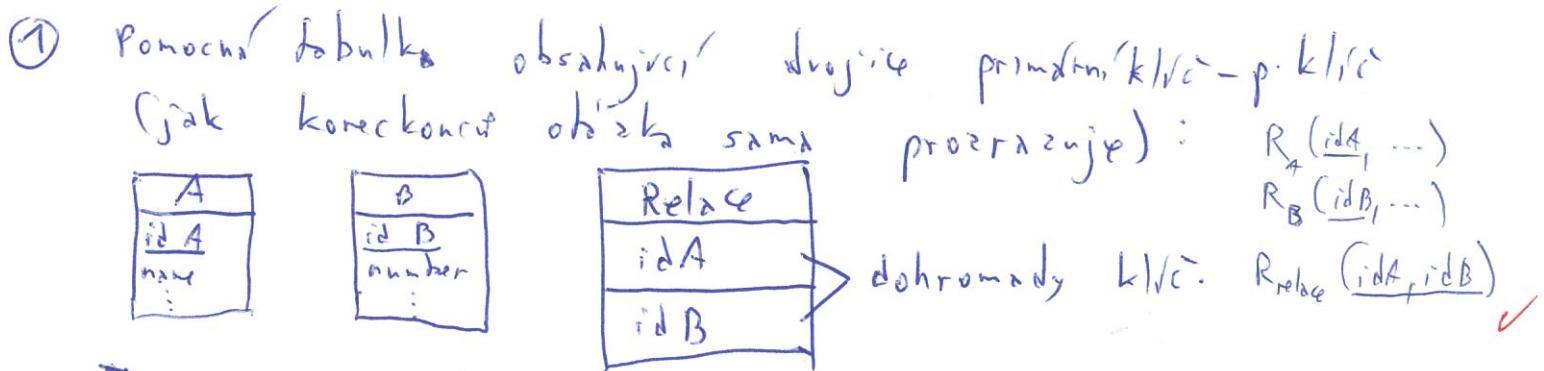
Kyr



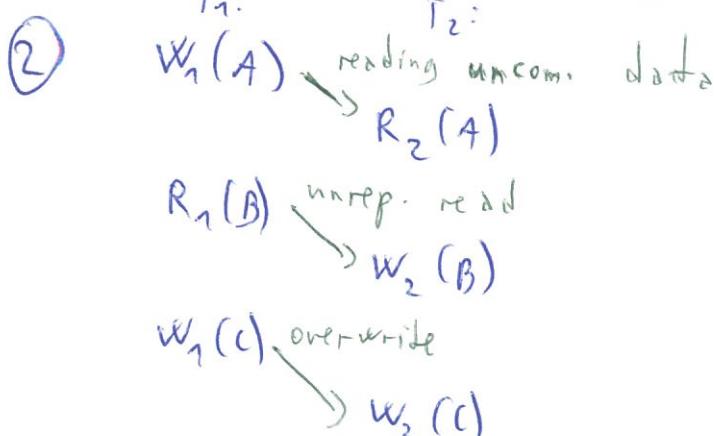
Kód studenta 22

2 Databáze (3 body)

- Načrtněte, jak byste v databázové tabulce reprezentovali binární vztah M:N. Lze z definice tabulky $T(\underline{FK_1}, \underline{FK_2})$, vzniklé převodem binárního vztahu, určit jeho původní kardinalitu (1:1, 1:N, M:N)? Vysvětlete.
- Uvažujte transakce $T_1: W(A) R(B) W(C)$ a $T_2: R(A) W(B) R(C)$. Je rozvrh $S: W_1(A) R_2(A) R_1(B) W_2(B) W_1(C) W_2(C)$ konfliktové serializovatelný (conflict serializable)? Vysvětlete proč a pokud není, navrhněte úpravu, aby byl.



Na druhém Pro kardinalitu 1:1 by bylo $T(\underline{FK_1}, \underline{FK_2})$, protože každý klíč určí jen druhý. Pro 1:N by bylo $T(\underline{FK_1}, \underline{FK_2})$, všechny, že každý člověk má několik mobilů, ale každý mobil patří jen jednomu člověku, tak všechny mobily, určené fo člověku, jsou patřiv. Je-li klíčem až dvojice klíčů, znanecky, že je znalošť jednoho souboru nemůže nic vše určit, t.j. relace je M:N. ✓



všechny konfliktové řípky vedené $T_1 \rightarrow T_2$, tj. graf konfliktů je acyklický, tj. rozvrh je serializ. a konfliktové ser. a odpovídají společnemu rozvrhu T_1 T_2 .

③ MM

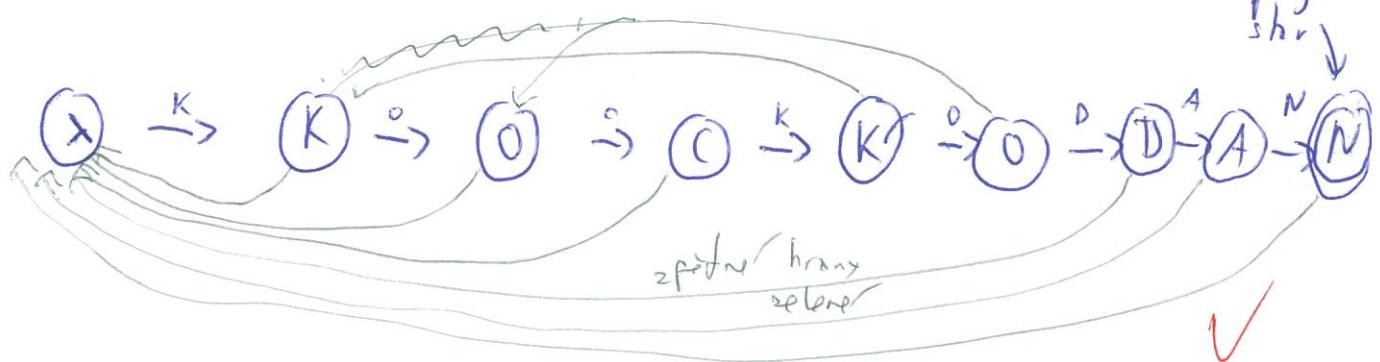


Kód studenta 22

1 Algoritmy a datové struktury: vyhledávání v textu (3 body)

- Definujte vyhledávací automat algoritmu KMP (Knuth-Morris-Pratt). Jak vypadají jeho stavy, dopředné a zpětné hrany?
- Nakreslete vyhledávací automat pro slovo KOCKODAN.
- Jakou časovou složitost má konstrukce automatu a jakou jeho použití k vyhledání všech výskytů slova v textu?

① Buť β jehla (tj. hledané slovo). Stavy automatu jsou odpovídající prefiksu β (vč. λ), ze stánu α do stánu αx , $x \in \Sigma$ večeří hrana s šířkou x , zpětná hrana vedoucí ze stánu α do stánu β takového, že β je nejdéle vlastní/suffix α . (β je rozšířený prefiks) ✓



③ Konstrukce automatu trvá $O(|\beta|)$, protože vlastně jen hledáme bez provádění přesného hledání slova v němž hledáme. Pokud předpokládáme, že máme automat sesetřeny někde v pořadí. ✓