Kód studenta 47

8 Hranové obarvení (otázka studijního zaměření – 3 body)

Hranové k-obarvení grafu $G = (V, E)$ je přiřazení čísel (resp. barev) $1, 2, ..., k$ hranám grafu tak, že žádné dvě hrany se společným vrcholem nemají stejnou barvu. Problém hranového barvení spočívá v nalezení hranového obarvení s nejmenším možným počtem barev.

Navrhněte $(2 - 1/\Delta)$-aproximační algoritmus pro problém hranového barvení, kde Δ označuje maximální stupeň ve vstupním grafu.

Pořídíme si $2\Delta-1$ barev $k_1 \dots k_{2\Delta-1}$. Položíme $\delta(v_i)$ počet neobarvených hran vrcholu $v_i \in V$. Každý vrchol si pamatuje, jaké barvy smí použít. Pokud chceme obarvit hranu $(u,v) \in E$, znamená to, že $\Delta \geq \delta(u) > 0$ & $\Delta \geq \delta(v) > 0$, pak ale máme ve vrcholu u alespoň $2\Delta-1-\Delta+\delta(u) \geq \Delta$ volných barev a stejně tak u vrcholu v. Protože máme $2\Delta-1$ barev celkem, má u i v alespoň jednu barvu společnou, kterou použijeme na obarvení (u,v).

Optimum OPT je očividně $\geq \Delta$ z definice hranového obarvení.

Potom: $OPT \geq \Delta$ & $OPT' \leq 2\Delta-1$ $\rightarrow OPT' \leq (2 - 1/\Delta)\,OPT$

A náš algoritmus je tedy $2 - 1/\Delta$ aproximační.

Složitost: $O(|E| \cdot \Delta^2)$.

3

0 bodů

Kód studenta 47

7 Kódy (otázka studijního zaměření – 3 body)

Definujte pojem minimální vzdálenosti pro samoopravné 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

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

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

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

Kód studenta 47

9 Incidence bodů a přímek (otázka studijního zaměření – 3 body)

2+b

Mějme k přirozené číslo a mějme dánu množinu $4k^4$ bodů v rovině

$$B = \{0, 1, \ldots, k-1\} \times \{0, 1, \ldots, 4k^3 - 1\}.$$

Dále mějme množinu $4k^5$ přímek P obsahující všechny přímky s rovnicí $y = ax + b$, kde $a \in \{0, 1, \ldots, 2k^2 - 1\}$ a $b \in \{0, 1, \ldots, 2k^3 - 1\}$.

1. Určete počet incidencí B a P.
2. Může mít systém $4k^4$ bodů a $4k^5$ přímek asymptoticky více incidencí než systém (B, P) výše? (Pokud k odpovědi potřebuje nějake tvrzení (větu) ze sylabu přednášky Kombinatorická a výpočetní geometrie I, tak toto tvrzení formulujte a odvoďte, jak z něho řešení plyne. Nemusíte ale tvrzení dokazovat.)

~~$O(m^{3/2} n^{3/2} + m + n)$~~ $O(mn)$

1. Body tvoří obdélníkovou mřížku. Výška $y \in \{0, \ldots, 4k^3 - 1\}$, šířka $x \in \{0, \ldots, k-1\}$.

b určuje posun přímky po ose y, a určuje náklon přímky.

Poté počet incidencí I je roven:

$$I = \sum_{b=0}^{2k^3-1} \sum_{a=0}^{2k^2-1} \min\left(\lim_{x \to a^+} \frac{4k^3 - 1 - b}{a}, k \right) ?$$

Tedy jdeme přes všechny posuny a náklony přímek a počítáme, kolik bodů protneme, než překročíme výšku nebo šířku mřížky.

Pro $b=0$ až $b = 4k^3 - 2k^3 + k - 1$ platí: CO ZNAČÍ TENTO ZÁPIS?

$\forall a \in \{0, \ldots, 2k^2 - 1\}$: # incidencí $J(a,b) \geq J(2k^2-1, 4k^3 - 2k^3 + k - 1) = \min\left(\frac{2k^3 - k}{2k^2 - 1}, k \right) = k$

Tedy $I = (2k^2 - 1) \cdot (4k^3 - 2k^3 + k - 1) \cdot k + z$ kde z značí zbytek. ZBYTEK ČEHO?

$I = O(k^6) + z$. Ve zbytku budeme omezeni výškou mřížky m tedy

$I = O(k^6)$. NENÍ SPOČÍTÁNO PŘESNĚ

2. Nemůže, protože z tvrzení z KVGI víme, že počet incidencí m bodů a n přímek v rovině je $O(m^{2/3} n^{2/3} + m + n) = O((4k^4)^{2/3} (4k^5)^{2/3} + 4k^4 + 4k^5) =$

$= O(k^{8/3} k^{10/3} + k^4 + k^5) = O(k^6)$. ✓

3

Kód studenta 47

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

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

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

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

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

Máme-li první zápis, můžeme jej přepsat jako:

$$\begin{array}{lll} a'_i x' \leq b_i & & \forall i \\ -x_1 + a'_j x' \leq b_j & \rightarrow \quad a'_i x' \leq b_i & \forall i \\ x_1 + a'_k x' \leq b_k & -x_1 \leq b_j - a'_j x' & \forall j \\ & x_1 \leq b_k - a'_k x' & \forall k \end{array} \tag{c}$$

~~Tyto nerovnice musí platit současně. Jsou to tedy omezující podmínky na hodnoty x_1~~

$\rightarrow$: Mějme x takové, že splňuje (a). Poté platí:

$0 \leq b_j - a'_j x' + x_1 \quad \forall j \in \{m_1+1, \ldots m_2\}$ ~~a zároveň~~

$0 \leq b_k - a'_k x' - x_1 \quad \forall k \in \{m_2+1, \ldots m\}$

Musí tedy platit i $0 \leq b_j + b_k - a'_j x' - a'_k x' + x_1 - x_1 \quad \forall j \forall k$.

A tedy i $a'_j x' - b_j \leq b_k - a'_k x' \quad \forall j \forall k$. ✓

'←': ~~Mějme Pokud Mějme x takové, že splňuje (b).~~

~~Poté, pokud by neexistovalo x nesplňovalo (a), zrovna to, že~~

~~$\exists j \in \{m_1+1, \dots m_2\}\ \exists k \in \{m_2+1, \dots m\}$:~~

~~$-x_1 + a'_j x' \leq b_j \ \wedge\ x_1 + a'_k x' > b_k$~~ nebo

~~$-x_1 + a'_j x' > b_j \ \wedge\ x_1 + a'_k x' \leq b_k$~~

~~x porušuje jednu z podmínek (a)~~ pro $i \in \{m_1+1, \dots m\}$.

~~BÚNO x porušuje podmínku $i = m_1+1$. Poté $\forall k \in \{m_2+1, \dots m\}_k$ platí podle (c):~~

~~$-x_1 > b_i - a'_i x'$~~

~~$x_1 \leq b_k - a'_k x'$~~ ~~$x_1 \leq b_k - a'_k x' \leq$~~

~~$\rightarrow x_1 > b_i - a'_i x' \geq b_k - a'_k x' \geq x_1 \rightarrow -x_1 > x_1$~~

Předpokládáme existenci x' splňujícího (b).

Hledáme x_1 takové, aby: $-x_1 + a'_j x' \leq b_j \quad \forall j \in \{m_1+1, \dots, m_2\} \ \wedge$

$x_1 + a'_k x' \leq b_k \quad \forall k \in \{m_2+1, \dots, m_2\}$.

$\rightarrow \quad x_1 \geq b_j - a_j x' \quad \forall j \quad \wedge \quad x_1 \leq b_k - a_k x' \ \forall k$

$\rightarrow \quad b_j - a'_j x' \leq x_1 \leq b_k - a'_k x' \quad \forall j \forall k$

$x_1 \in \left[\underbrace{\sup \{ b_j - a'_j x' \mid \forall j \in \{m_1+1, \dots, m_2\} \}}_{S}, \ \underbrace{\inf \{ b_k - a'_k x' \mid \forall k \in \{m_2+1, \dots m\} \}}_{I} \right]$

Kdyby byl interval prázdný, pak $S > I \rightarrow$

$\exists j, k: \ b_j - a'_j x' > b_k - a'_k x'$, což porušuje (b) ↯. □

Zbytek v příloze (1).

(1)

Kód studenta 47

6 Optimalizační metody

Systém nerovnic můžeme řešit iterativně:

ⓐ má řešení <-> ⓑ má řešení <->

$a_i' x' \leq b$ $\forall i$

ⓓ $(a_j' + a_k')x' \leq b_k + b_j$ $\forall j,k$ má řešení

ⓓ má $(m_1) + (m_2 - m_1) \cdot (m - m_2)$ nerovnic. Nejhůř

$O + \frac{m^2}{4}$.

ⓓ je tedy nová soustava nerovnic $A'x' \leq b'$.

Tu můžeme opět upravit na

ⓔ
$a''_i x'' \leq b'_i$ $\forall i \in \{1, \dots, m'_1\}$

$a''_j x'' - b'_j \leq b'_k - a''_k x''$ $\forall j \in \{m_1+1, \dots m'_2\}$ $\forall k \in \{m'_2+1, \dots m'\}$

ⓔ má nejhůře

$\frac{m^4}{16}$ nerovnic.

Nakonec nám zůstane jen 1 proměnná. Pokud pro ni existuje řešení, můžeme dohledat ostatní členy x.

Časová složitost roste s počtem nerovnic exponenciálně:

Po α krocích máme $O\left(\frac{m^{2^\alpha}}{2^{2\alpha}}\right)$ nerovnic.

✓

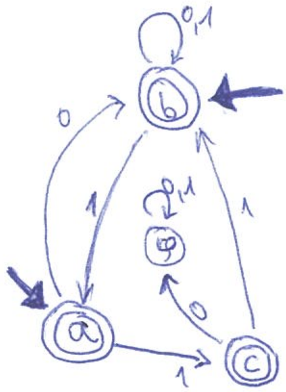
0N

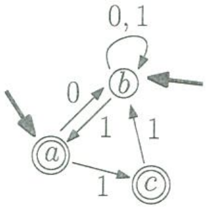
Kód studenta 47

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 $L(A)$? ✗

1.





φ uprostřed je 'stok', kde skončí slova nevyhovující automatu A.

není deterministický KA ; (b, 1) = {a, b} ...

3 -

Kód studenta 47

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 zapouzdřuje 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/slovník (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 třídá 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` zaregistruje novou službu v rámci serveru. Tato metoda může být relativně pomalá a volá se typicky jen při startu aplikace (např. když jsou načítány zásuvné 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 naprosto 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.

```
public interface IService {
    string ServiceId {get;}
    string CallMethod (string methodId, ParametersCollection parameters);
    IEnumerable<string> ContainedMethods {get;}
}
```
Všechny services budou muset implementovat dispatching.

```
public class GreetingService : IService {
    public string ServiceId => nameof(GreetingService);
    public string delegate Method(ParametersCollection parameters);
    private readonly Dictionary<string, Method> _methods;

    public GreetingService(){
        _methods = new Dictionary<string, Method>();
        _methods.Add("Greet", Greet); // Dalo by se zlepšit pomocí reflexe.
    }

    public IEnumerable<string> ContainedMethods => _methods.Keys;
    private string Greet(ParametersCollection _){
        return "Hello World!";
    }

    public string CallMethod(string methodId, ParametersCollection parameters){
        return _methods[methodId].Invoke(parameters);
    }
}
```
↳ co když tam není?

```
public Server ...{
    :
    private readonly Dictionary<string, IService> _services; // Init in constructor
    :
    public void Add(IService service){
        _services.Add(service.ServiceId, service);
    }
    private string Dispatch(string serviceId, string methodId, ParametersCollection parameters){
        return _services[serviceId].CallMethod(methodId, parameters);
    }
}
```
co když tam není?

Pozn.: V kódu záměrně neřeším výjimky z důvodu přehlednosti a časového presu. Mělo by se ošetřit: Neexistence servisy, metody. Duplicity záznamů.

Metoda Dispatch obsahuje 2 volání vytažení informací ze slovníků. Dalo by se to zkrátit na 1 volání, pokud by si server pamatoval namísto servis rovnou metody. Avšak 2 volání mi připadají jako malá daň za přehlednost.

2 body

Kód studenta 47

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, výslovně to napište).

Oddíl naformátovaný (zjednodušeným) souborovým systémem FAT12/16/32 je rozdělený 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 infromace 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 znovu 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 bitech 2. bytu; záznam pro sektor s vyšším číslem je ve vyšších 4 bitech 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. V závislosti na velikosti oddílu O : $\frac{O}{2^y - 1}$, kde y je 16 pro FAT16, 12 pro FAT12 a 32 pro FAT32.

$2^y - 1$ je ~~počet~~ maximální počet záznamů v tabulce (z počtu čísel vytvořitelných z y bitů pomocí kterých můžeme odkazovat na datové sektory. 2^y je rezervováno na ukončení čtení souboru.

O je velikost oddílu po odečtení velikosti tabulky a boot sektoru.

nelze zvolit – daná HW

✗

FAT16. 1GiB = 2^{30} B. Tedy FAT 32 by měl ~~víc~~ záznamů v tabulce než je počet bajtů, které máme k dispozici.

2. Máme-li FAT16 systém, je velikost jednoho ~~sk~~ sektoru $\geq \frac{2^{30}}{2^{16}} = 2^{14} = 2^4$ kiB = 16kiB

Velikost tabulky je 2^{16} * . 16b = 2^{17} B = 2^7 kiB = 128kiB.

Načítáme 100 000 souborů, v tabulce máme záznamy pro 64. 1024 sektorů * (= 65 536). Jeden soubor má velikost 1kiB a vejde se tedy vždy do jednoho sektoru. Jestliže máme k dispozici jen 65536 záznamů, budeme načítat některé soubory vícekrát. ~~a tedy se vypla~~ *

Pokud načítáme opakovaně jen pár souborů, vyplatí se načítat každý záznam zvlášť. Každopádně velikost tabulky je 128KiB a tedy by se měla do paměti vejít celá * bez problémů * a ušetřit to naram ušetří časově drahé přístupy na disk.

⇒ z toho vyplývá, že řešení 1 je špatně ;-)

3.
```
public void IterateFat( int index){
  while (index < fat.Length){
    Console.WriteLine (index);
    var a = ~~index / 2~~ ((index - 1)/2) * 3 ;
    if ( index % 2 == 1){ //Indexováno od 1
        index = fat[a] << 4 ;
        index += fat[a+1] & 0xOF ;
    }else {        // 2, 4, 6, 8, ...
        index = (fat[a +1] & 0xF0) << 8 ;
        index += fat[a+2] ;

    }
  }
}
```

✓

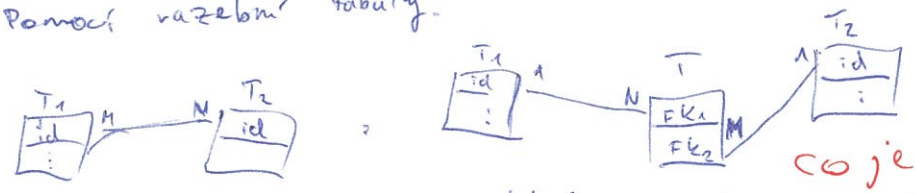
1 kp.

Kód studenta 47

2 Databáze (3 body)

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

1. Pomocí vazebních tabulky.



co je lépe ?

Podle kardinality hodnot v tabulce. Pokud má vazební tabulka kardinalitu [(1:1), (1:1)], byla původní kardinalita (1:1).
[(1:1), (1:N)] (1:N).
[(1:M), (1:N)] (M:N).

toto najdu v definici té vazební tabulky

2. ~~Předpokládám, že~~

~~Není, protože nemůžeme říci, zda se má~~ ~~nejprve provést~~ ~~$R_1(B)$ či $W_2(B)$~~

Není, jelikož v transakci 1 se zapisovaná hodnota C může odvíjet od čtené hodnoty B, která se mezi $R_1(B)$ a $W_1(C)$ změní v $W_2(B)$. Úprava:

$W_1(A)\ R_2(A)\ R_1(B)\ W_1(C)\ W_2(B)\ R_2(C)$

je, pokud by T_2 běžela za T_1 dopadlo by to stejně

0 MM

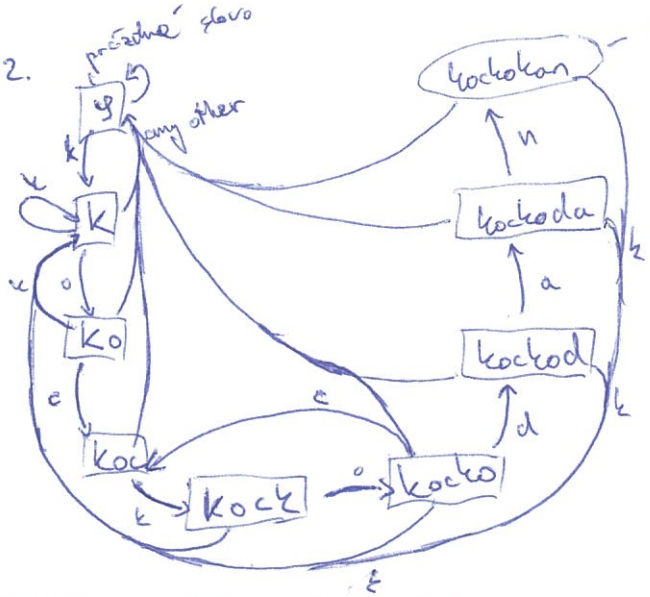
Kód studenta 47

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

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

1. Automat algoritmu KMP je množina stavů a přechodů mezi nimi. Stavy dělíme na počáteční (prázdné slovo), mezistavy a koncové přijímající stavy.
Dopředné ~~hra~~ Hrany (přechody) mezi stavy dělíme na dopředné a zpětné. Každá hrana má asociovaný character znak abecedy.
Zpětná hrany zamrtají? slovo a pokud naše slovo obsahovalo podsekvenci, která sedí na začátek automatu, pokračujeme ve zpracování textu z ~~tohoto~~ stavu. co to přesně znamená? nejdelší takovou patřičného přijímací stav

2.



v tabulce:

q \	K	O	C	D	A	N	?
0	1	0	0	0	0	0	0
1	1	2	0	0	0	0	0
2	1	0	3	0	0	0	0
3	4	0	0	0	0	0	0
4	1	5	0	0	0	0	0
5	1	0	3	6	0	0	0
6	1	0	0	0	7	0	0
7	1	0	0	0	0	8	0
8	1	0	0	0	0	0	0

lib. - jiný znak abecedy

kockodan přijímací stav

Tohle je klasický DKA, nikoliv KMP

3. Čas konstrukce nejhůře $O(m^2 n)$ (délka abecedy, délka slova)

Pro každé písmeno slova, pro každé písmeno abecedy projdeme celé slovo.

Ale řekněme se. Dá se lépe odhadnout jako $O(m^2 \cdot \min\{m,n\})$.

Vyhledávání v textu poté $O(t \cdot \min\{m,n\})$, (délka textu)

Pro každé písmeno z textu projdu řádek tabulky.

nikoliv, je to $O(m)$ nezávisle na velikosti abecedy

tohle má být $O(t)$