



Kód studenta 28



8 Hranové obarvení (otázka studijního zaměření – 3 body)

Hranové k -obarvení grafu $G = (V, E)$ je přiřazení čísel (resp. barev) $1, 2, \dots, k$ 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.

Pozorování : OPT je alespoň Δ (jinak neobarvíme hrany incidentní s vrcholy s max. stupněm)

Máme tedy $2\Delta - 1$ barev. ✓

Postupujeme takto :

Barteme hrany v libovolném pořadí a barvíme libovolnou barvou, která rezponsabil kolizi.

Počít to funguje?

3

V nejhorším případě máme hranu vedoucí mezi dvěma vrcholy stupně Δ a všechny okolní hrany už jsou obarvené.

Kromě hrany UV je s vrcholy u něbo v incidentních režijních $2\Delta - 2$ bran (tedy pokud ur existuje) a tudíž nám jedna barva na UV zbyde. ✓

(Snad nemusím dokazovat, že to skutečně běží v polynomickém čase)



2 body

Kód studenta 28



7 Kódy (otázka studijního zaměření – 3 body)

Definujte pojem minimální vzdálenost 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

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

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

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

Minimální vzdálenost je minimální počet něn patiček na
znamku kterého kód má libovolný jiný kód se něn?

Co to znamená?

(Pro množinu kódů C je to $\min_{x,y \in C} d(x,y)$ pro všechnou definici d, co to je?

Vzdálenost dvou kódů $d: C \times C \rightarrow \mathbb{N}$ se běžně definovala jeho

a) počet odlišných bitů (default) zaměňte pojmy

b) Levenshteinova vzdálenost „kód ‘a’, kód ‘b’“ slovo

:

délka kódů je 8 (protože \mathbb{Z}_2^8)

velikost kódů je $\binom{4}{2}^2 = 6$ (vybitíme dve sudé a dvě liché pozice)

vzdálenost je 2

- flip jednoho bitu námě vzbudit podmínu na počet jedniček
- 11110000 a 11011000 se liší jen ve dvou bitech

✓ 5

"zelených" vektorů je $\binom{4}{2}^2 = 6^2 = 36$ (vybráno 2 z 4 lichých
+ 2 z 4 sudých pozic)

ale celková velikost C je $8^2 = 64$ proč?

protože jediné podmínka je ta, že parita sudých i parita
lichých pozic je 0. (tedy nula nuly a 4 jedničky)

Jemu zahrnují

neobsahují

Jsme tedy schopni detektovat jednu chybu
ale ne opravit.



Kód studenta 28

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

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

(1b) +

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

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

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 nějaké 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.)

$$(2k^2-1)(k-1) + (2k^3-1) = 2k^3 - 2k^2 - k + 1 + 2k^3 - 1 = 4k^3 - 2k^2 - k \cancel{\times} \leq 4k^3 - 1$$

(pro $k \geq 1$)

tudíž každá přímka je paralelna k jinému bodu
 $\rightarrow 4k^6$ incidencí ✓

$\left. \begin{array}{l} \text{+ porování o} \\ \text{celočíselných soustach} \\ \text{a koeficientech} \end{array} \right\}$

2. Každá přímka je jednoznačně určena dvoujicími bodů.

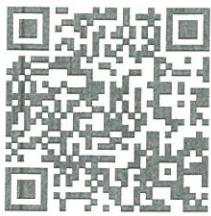
$\Theta(k^4)$ bodů $\rightarrow \Theta(k^8)$ dvoujic bodů

Máme $\Theta(k^5)$ přímek, tudíž každá z nich může procházet $\Theta(k^2)$ bodů (a tudíž zabrat $\Theta(k^3)$ dvoujic).

(Pro nerovnorovné rozložení přichodi to nevhodně k psp)
 (počet zabraných dvoujic roste superlineárně s počtem bodů na přímce)

\rightarrow určitě není možno víc ne $\Theta(k^{13/2})$ incidencí

↳ TO JE PŘÍLIŠ SLABÉ, NEODPOVÍDÁ NA OTÁZKU



Kód studenta 28

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

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

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

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

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?

" \Rightarrow "

pokud původní systém ně řešení x
tak \textcircled{I} je určitě splněno
levá část \textcircled{II} je $\leq x_1$
a pravá část \textcircled{II} je $\geq x_1$

" \Leftarrow "

pokud má nový systém řešení x'
tak najdeme x_1 takoví
aby splňovalo
 $a'_j x' - b_j \leq x_1 \leq b_k - a'_k x' \quad \forall j \in \{m_1+1, \dots, m_2\}$
 $\forall k \in \{m_2+1, \dots, m\}$

Opakováním kroků dojdeme k systému s jednou proměnnou.
Ten triviálně vyřešíme a následně začneme dopočítávat
jednotlivá x_1 . (Převod $a'_j x' - b_j \leq b_k - a'_k x'$ na $(a_j - a_k)x' \leq b_k + b_j$)
Vzhledem k tomu, že každé řešení "menšího" systému je "vzítovat"
implikuje řešení "většího", tak řešení systému s jednou proměnnou
nás dostane až k původnímu řešení. (Příme z " \Leftarrow ")
" \Rightarrow " naopak zárokyje, že pokud nějaký systém nemá řešení
tak jej nemá žádat z nás.

Cesová složitost postupu

Pořadujeme kolik iterací, kolik je proměnných.

V každé iteraci možná musíme provádět všechny koeficienty

Navíc nam v části \textcircled{II} může vzniknout $\binom{n}{2}^2$ nerovnic.

Ty musíme překouknout filtrovat, jinak můžeme dostat i $\Omega(n!)$ nerovnic

(Po $\frac{\log n}{2}$ iteracích by počet nerovnic mohl být $\geq \sqrt{n}^{\sqrt{n}}$)

Pokud budeme filtrovat nerovnice Gaussova eliminací, tak proč to nevyužít GE celé?

(Filtrování = hledání lineárně nezávislých vektorů)

Pro "hloupoč" implementaci tedy minimálně $O(n!)$

Pro inteligentní implementaci používáme n-krat filtrování ale nás rychle se zvýšující problém

→ zjistíme nás je první spustění na $O(n^2)$ nerovnic

Jak rychle umíme najít $O(n)$:lin. nezávislých vektorů mezi $O(n^2)$ vektory délky n?

Gauss. Elim. v $O(n^4)$, možná to jde i lepe?

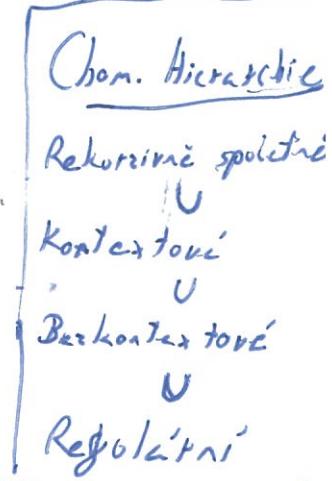
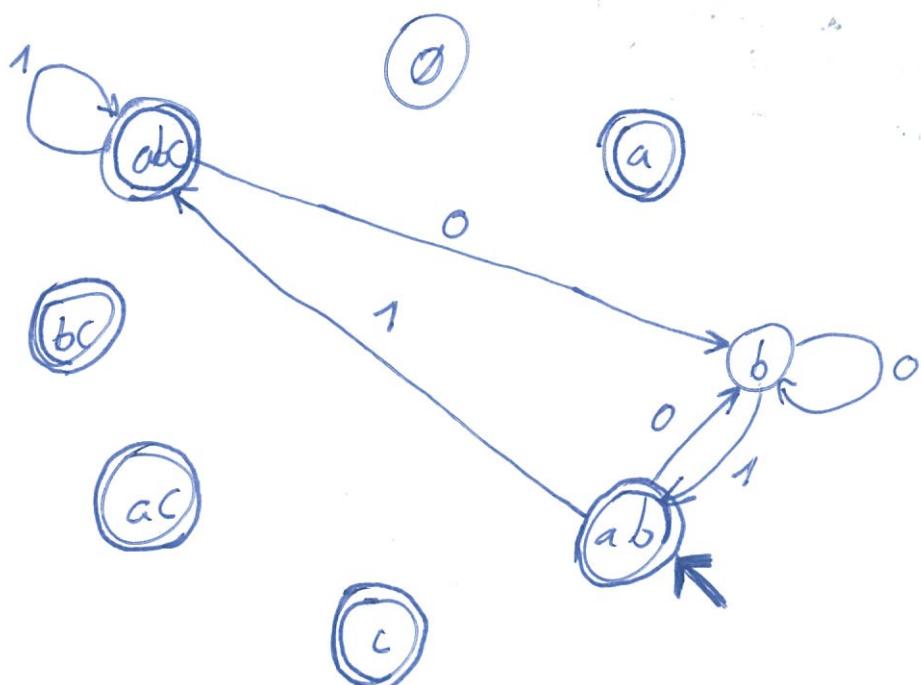
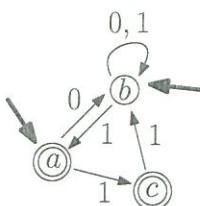
BN



Kód studenta 28

5 Automaty (3 body)

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



✓ (hranы pro redosažitelné stavы směšány pro přehlednost)

Tento jazyk patří do všech tříd v Chomského hierarchii
- je rozpoznatelný konečným automatem, což jej řadí např.
do reduktivních tříd. (Regulační jazyky)

0+
na



Kód studenta 28



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 přeče 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/slovnik (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 základní 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) popишete, jak bude vypadat implementace metod `add` a `dispatch` třídy `Server`. Pokud tyto metody potřebují přistupovat ke členským proměnným třídy, popишete 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.

Samostatný papír

```
private string dispatch(string serviceId, string methodId, ParametersCollection  
parameters)
```

```
    return map[serviceId, methodId](parameters)
```

takovou mapu v C++ něžná

a to je co?

map je tedy rejtak' typické mapování z $\overbrace{\text{String, String}}$ na
 $\text{string}^*(\text{ParametersCollection})$.

A ano, předpokládem, že implementuje díky [] operator.

Příklad je (asi) v C, ne v C++. Kdile se
v C++ opravdu reprogramuje.

Druho dítka chybí - nedostatkem yskytlu.

Student 28 Otázka 3

interface IService {

public:

table by methods + C++ napsal!

int numberOFMethods();

struct MethodInfo * listMethods();

string serviceName();

}

struct MethodInfo {

char * name;

string * (ParametersCollection)

methodp;

table je pointer
na funkci, jež
tím reprezentuje
metodu?

co já vymyslím, co výjde nejjednodušší abstraktní mapování z
dvojic stringů na pointery na funkce

std::map
by upřímně řešil

Trie?

Hashování? (název ještě perfektní)

public void add(IService service){

string s = service.serviceName();

int n = service.numberOFMethods();

struct MethodInfo * l = service.listMethods();

for (int i=0; i<n; i++) {

je self? jak funguje insert s tímto argumenty?

self.map.insert(s, [Ei].name, l[i].methodp);

? Co je map?

}

}

3 body
pěkné až na rek.



Kód studenta 28

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 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 zvláště). Pro jednoduchost předpokládejme, že alokační jednotka souborového systému je právě jeden sektor disku. Sektoru 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 (číselovány 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 načít si předem celou FAT tabulku do paměti? Nebo by bylo vhodnější před čtením každého ze souborů znova načít 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`).

Samoslaty papír

tradice ✓ *nichéji nové disky?*

Sektor bývá kolem 1kB, běžně 512B - 4kB (když tak sítan ta i-čka dny s lety)

FAT16 by 1GB rozdělil na 2^{16} datových sektorů, samotná tabulka by zbraňala 128kB (zanechává všechno) a každý sektor tedy zanechává.



- datový sektor by měl cca 16kB, což je trochu moc
- byly byly výsledně omezeny na $< 2^{16}$ souborů, což je trochu málo (pokud neexistuje tak velké sektoře, tak naopak ztrácíme kapacitu)

FAT32 by 1GB určitě zvládlo rozdělit na 512B sektory těch by bylo cca 2^{21} , tudíž by tabulka zbraňala $2^{23}B = 8MB$.

Tabulka si otevírá též i v. oddílu, aby ztráty pro FAT16 byly větší (bud nemůžeme použít celý disk, nebo zaokrouhlujeme velikost souboru na nejbližší výšší násobek 16kB)

2. Záleží na vlastnostech disku (a možnosti dostupné paměti)

Pro SSD je to též jedno a tudíž nemusíme zapráškovat 8MB paměti tabulkou.

Pro normální disk by bylo fajn remixer se periodicky vracet k tabulce a přejít celý oddíl najedro. Pokud tedy máme dost paměti (pro dnešní pořadí není 1GB až taklik, ale kdo mi dnes 1GB oddíl?) tak to raději přestene až a "uvolnime" až v paměti.

BTW: 100 000 souborů → takže FAT32 je spíšem odpověď! ☺

3)

```
void list (int start, char* fat) {
    if (start == 212-1) {
        return;
    }
    printf ("%d \n", start);
```

~~char a = fat [start-1]~~

offset = ((start - 1) >> 1) * 3; // -1 kompenzuje vyslovní od 1

if (start % 2 == 1) {

return list (fat [offset] * 16 + fat [offset+1] % 16, fat);

}

return

list ((fat [offset+1] >> 4) * 16 + fat [offset+2], fat);

}

rekurze? zvládne pr. tail recursion?

vhodna

fat je globální!

(tiskne čísla datových sektorů, NE logických)

předpokládám (mno jiné)
>12b int
include stdio.h
bez vazeb na bloky
rekurze

1+ b

Kop.

rozvrh je serializovatelný

Kop.

navržená oprava by domnělý konflikt neopravila



Kód studenta 28

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 $T_2: R(A) W(B) \cancel{W(C)}$. Je rozvrh $S: W_1(A) R_2(A) R_1(B) W_2(B) W_1(C) W_2(C)$ konfliktově serializovatelný (conflict serializable)? Vysvětlete proč a pokud není, navrhněte úpravu, aby byl.

1) Seznamem dvojic cizích klíčů - $T(\underline{FK_1}, \underline{FK_2})$

Pokud je klíč opatřen minimální, tak ano

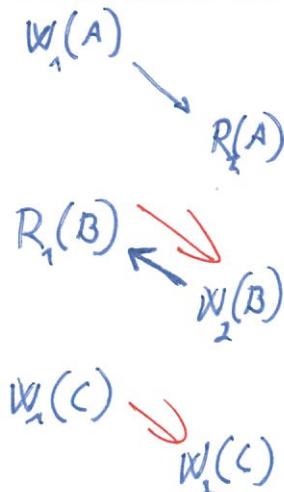
V případě vztahu 1:1 by byla definice $T(\underline{FK_1}, \underline{FK_2})$

(a asi by nebylo ještě potřeba separátní tabulkov)

V případě vztahu 1:N by byla definice $T(FK_1, FK_2)$

(a asi by taky nebylo ještě potřeba separátní tabulkov)

2)



Není - před $W_1(C)$ proběhlo $W_2(B)$
je
a tudíž databáze po $W_1(C)$
může být nekonzistentní
(ačkoliv to se spoavi po $W_2(C)$)

$W_2(B)$ musí počítat na $W_1(C)$

③ MM



Kód studenta 28



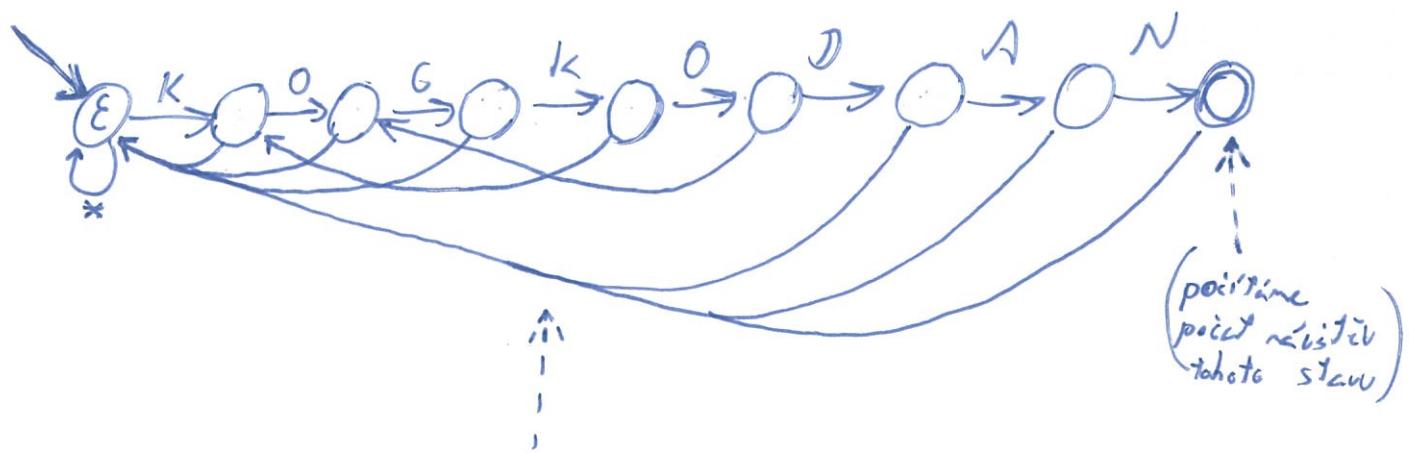
1 Algoritmy a datové struktury: vyhledávání v textu (3 body)

- 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) Vyhledávací automat algoritmu KMP pro m-znakové slovo obsahuje $m+1$ stavů odpovídajících všem prefixům hledaného slova (včetně celého slova a právního jízdyce).
- Dopředné hrany popisují přechod k nejbližšímu delšímu prefisu zatím co zpětné hrany vedou k nejbližšímu kratšímu prefisu, který je suffixem aktuálního stavu.
- Automat se snadí použít dopředné hrany, v případě selhání použije zpětnou a zkouší znova. Poze po návratu do stavu ϵ (tedy právního slova) je ochoten udělat krok ve vstupu bez použití dopředné hrany. ✓
- 2) Zpětné hrany se konstruují spojováním automatu na slovo bez prvního znaku a jsou spočítány v čase $O(n)$ pro m-znakové slovo.
- Pro vyhledávání všech výskytů je patička i v $O(n)$ pro n-znakový vstup, protože cena kroků zpět se amortizuje - než kroky upírád. → ✓

2)

KOCKODAN



tyto zpětné hranы reprezobují
zahodení znaku ze vstupu! ✓
(jen ta s hvězdičkou)