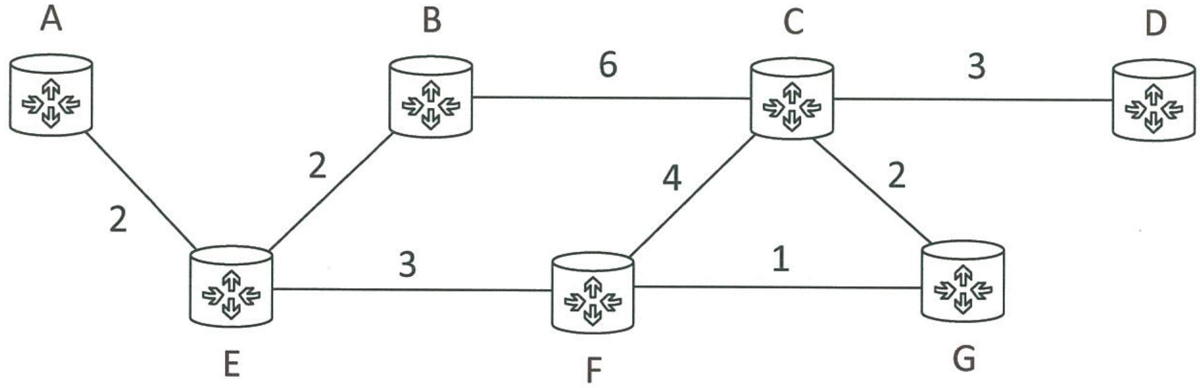
Houška Petr
houskape@gmail.com

3

Čas 10:28 - 10:43

6 Základy sítí: směrování (3 body)

Mějme sítě propojené směrovači (routery) dle obrázku. Routery jsou označeny písmeny, vzdálenosti k sousedním routerům jsou u hran grafu (hrany reprezentují sítě spojující routery a vzdálenost představuje „pomalost" daného spojení, tj. vyšší vzdálenost znamená pomalejší spojení). Routery na počátku znají pouze své sousedy a vzdálenosti k nim. K výpočtu směrovacích informací používají distribuovaný distance-vector algoritmus, který jako metriku používá součet vzdáleností mezi routery.



1. Jaké základní údaje musí obsahovat jednotlivé záznamy ve směrovací tabulce?
2. V kolikátém kroku algoritmu pro výpočet směrovacích informací znají pro síť na obrázku všechny routery úplnou optimální informaci o směrování? Počáteční stav je krok 0.
3. Napište obsah směrovací tabulky na routeru A po zjištění úplné informace.
4. Popište, jak budou reagovat ostatní routery, když vypadne router F. Změní to nějak vzdálenost z routeru A do routeru D?

1) adresy + masky sítí svých sousedů + metrika jak je to k nim daleko
+ časem adresy + masky cíl sítí + metrika + adresy dalšího routeru na ~~nejkratší~~ nejkratší cestě

2) 4, nejdelší optimální cesta: v každém kroku si ∀ router prodlouží informaci o cestách +1 hopů

3)

cíl síť	metrika	další router
E	~~4~~ 2	E
B	4	E
F	5	E
C	8	E
G	6	E
D	11	E

4) Ano změní, nová cesta bude mít stejný next hop, ale jinou metriku: 13
- sousední routery zjistí, že F vypadl, zpropag. informaci a začnou poč. nové cesty

např. z E už nejde do G přes F
- E od B zjistí, že B má do G za 8
- ~~aktualizuje~~ ↳ aktualizuje

↳ E taky invaliduje svoje cesty přes F, tj. např. A čímž invaliduje původní kratkou cestu do
↳ A si počká až E zjistí od B, další kolo pak aktualizuje

když to zjistí tedy rozešle, že z E do G např. se nedá dostat
↳ zatím nemá cestu
až další kolo rozešle, že našlo cestu přes B

3 -

Houška Petr
houskape@gmail.com

Čas .....

3 Databáze (3 body)

1. Stručně definujte význam pojmu *izolace* (isolation) v ACID transakčním modelu.
2. Způsob, kterým je implementována izolace transakcí v konkrétním databázovém systému, často způsobuje omezení jejich souběžného zpracování. SQL-92 definuje čtyři úrovně zajištění izolace (READ UNCOMMITTED, READ COMMITTED, REPEATABLE READS, SERIALIZABLE). Vyberte si libovolné dvě a stručně (nejlépe jednou větou) vysvětlete, jaké izolační vlastnosti zajišťují nebo naopak dovolují porušit.
3. Mějme tabulku uživatelů (`uziv`) se sloupci (`id`, `jmeno`, `plat`, `vek`). Tabulka obsahuje záznamy:

```
(1, "Franta", 40000, 42)
(2, "Pepa", 70000, 54)
(3, "Tonda", 100000, 63)
```

Dále uvažme dvě probíhající transakce. Pro jednoduchost si představme, že jednotlivé příkazy těchto transakcí byly postupně provedeny v pořadí, jaké je určeno v následujícím schématu. Výsledek každé operace SELECT je uveden pod dotazem v komentáři.

```
/* Transaction 1 */

SELECT plat FROM uziv WHERE vek =
    (SELECT MAX(vek) FROM uziv)
/* plat == 100000 */
UPDATE uziv SET plat = 50000 WHERE id = 1
```

```
/* Transaction 2 */

SELECT plat FROM uziv WHERE id = 1
/* plat == 40000 */
SELECT id FROM uziv WHERE vek =
    (SELECT MAX(vek) FROM uziv)
/* id == 3 */
UPDATE uziv SET plat = 120000 WHERE id = 3
COMMIT
```

```
SELECT AVG(plat) FROM uziv
/* AVG(plat) == 80000 */
COMMIT
```

Na základě příkazů v transakcích a uvedených hodnot určete, jakou úroveň izolace dle SQL-92 (tedy READ UNCOMMITED, READ COMMITTED, REPEATABLE READS, nebo SERIALIZABLE) splňuje databázový systém, který transakce zpracoval. Vaše rozhodnutí stručně zdůvodněte.

1) izolace znamená že pro jednotlivé transakce to vypadá, že právě je zpracovává právě jed tj. efekty transakcí jsem, až do atomického commitu pro ostatní neviditelné ✓

2) SERIALIZABLE: rozvrh je naprosto (z pohledu transakcí) ekvivalentní sériovému rozvrhu, tj. izolace je zajištěna dokonale. ✓
READ UNCOMMITED: je dovoleno i číst ještě necommitnuté změny cizí transakce tj nezajišťuje, že to nepřečte údaje, co nakonec o třeba commitnuté nebudou. ✓

3) Určitě není SERIALIZABLE, neodpovídá sériovému rozvrhu PROČ není REPEATABLE READS ?!
READ commited! očividně se čtení projeví až commitnuté změny. Zatímco u 1. transakce se projeví už commited změna platu z 2. i tak v 2. se neprojeví necommitnutá změna z 1. ✓ −

3 body + ● skoro excelentní!

Houška Petr
houskape@gmail.com

Čas 10:47-11:00

5 Architektura počítačů a operačních systémů (3 body)

Předpokládejte následující program zapsaný v jazyce C# (klíčové slovo `static` označuje globální proměnnou nebo globální funkci, datový typ `int` je 32-bitový celočíselný se znaménkem):

```
1  using System;
2
3  class Program {
4          static int a;
5          static int b;
6          static int c;
7
8          static void ComputeResult() {
9                  c = a / b;
10         }
11
12         static void Main(string[] args) {
13                 a = int.Parse(Console.ReadLine());
14                 b = int.Parse(Console.ReadLine());
15                 ComputeResult();
16                 Console.WriteLine(c);
17         }
18 }
```

Dále předpokládejte nějaký 32-bitový procesor s obecnou registrovou architekturou a s podporou pro hardwarové celočíselné dělení – dostačuje nějaká obecná představa procesoru, jehož instrukční sada by rámcově odpovídala typickému návrhu takových procesorů. Tedy můžete využít instrukce nějaké reálné procesorové architektury (např. x86), ale stejně tak je plně dostačující, pokud si nějakou přiměřeně realistickou instrukční sadu vymyslíte sami.

1. Zapište v assembleru uvažovaného procesoru posloupnost instrukcí, do které by se nejspíše přeložil jediný řádek C# kódu ve funkci `ComputeResult` (tj. řádek číslo 9 celého programu). Adresy proměnných si libovolně vhodně zvolte. Pro každou použitou instrukci napište stručný popis jejího chování.

2. Pokud výše uvedený program spustíme a na standardní vstup zadáme:

   ```
   20
   0
   ```

   tak vypíše:

   ```
   Unhandled Exception: System.DivideByZeroException: Attempted to divide by zero.
      at Program.ComputeResult() in Program.cs:line 9
      at Program.Main(String[] args) in Program.cs:line 15
   ```

   Vysvětlete kdo (jaký kód) nejspíše vypisuje text uvedené chybové hlášky „Unhandled Exception ...“.

   Vraťte se ke svému řešení první části otázky a vysvětlete na něm, jak je zařízeno, že když procesor bude vykonávat vámi uvedený strojový kód a dojde k pokusu o dělení nulou, tak dojde k výpisu zmíněné chybové hlášky. Stačí vyjmenovat a stručně popsat posloupnost jednotlivých hlavních celků operačního systému, uvedeného programu, atd., které se na ošetření dělení nulou budou v typické situaci podílet.

1) c = a / b ← vždy glob.

```
load R1 0xFE(....)      // načte do registru R1 hodnotu na adrese nativní
                        //   slovo procesoru "int 32" tedy 4
load R2 (....)
intdiv R1 R2 R3         // provede celočíselné dělení: první operand / druhý
                        //   & uloží výsledek do třetího
store R3 0x(....)       // uloží obsah registru (R3) na adresu
                        //   - protože registry 32 bit, uloží 32 bit číslo
```

(32 bit adresa 'a'; adresa 'b'; př.?)

✓

2) Ten kód vypíše handler CLR poté, co zachytí neošetřenou managed (v uživ. kódu) výjimku, která vznikne na 9. řádku po následující sérii událostí. ✓

1. Procesor zpracovává instrukci 'intdiv': narazí na to, že druhý operand je 0
2. Vyvolá (sám CPU) synchronní přerušení & do registru nastaví flag 'Division By Zero'
3. Podle tabulky přerušení zavolá obsluhu OS pro přerušení
4. OS v rutině pro přerušení zjistí, že šlo o div0 Exc, vyhodí (v případě Win) SH exception do programu, který právě běžel CLR (pro linux obdobně)
5. CLR zachytí výjimku, zabalí ji do managed objektu, nastaví metadata (ten text) a "vyhodí" ji na 9. řádku. Výjimka probublá user kódem, nikde není zachycena
6. Zachytí ji CLR handler, její message vypíše na konzoli & program ukončí

(3)

Houška Petr
houskape@gmail.com

Čas ...20... min

8 Matematická lingvistika: základní formalismy pro popis přirozených jazyků (otázka pro studijní zaměření – 3 body)

1. Podrobně popište základní fakta o tvorbě jazykových korpusů. Uveďte základní charakteristiky, jak by měl moderní korpus vypadat.
2. Rozdělte jazykové korpusy podle typu značkování, pro každý typ uveďte alespoň dva příklady.
3. Uveďte příklady paralelních korpusů a vysvětlete jejich využití. druhá strana

1) Jazykový korpus je nějaká soustava textů jednoho / více jazyků.
Aby byla užitečná, tak by měla daný jazyk dobře reprezentovat: ✓
- být dost velká: ~ desítky milionů slov
- obsahovat dostatečně různorodé texty
  - např. TreeBank byly jen texty z WSJ
    ⇒ nereprezentuje obecný jazyk ale jen podmnožinu používanou v bankovním světě (lepší reprezentace „průměrného jazyka", nejde jen o různorodost, ale i reprez. poměr)
  - ~ naopak Český národní korpus: noviny, časopisy, knihy, ...
- ideálně co nejvíce anotovaný: 15 českých morfo. značek
2)
  - anotace: značkování
    - morfologické: slova v něm mají morfologické značky (slovní druh, pád, rod, v případě češtiny)
    - syntaktické: věty jsou parsované, tj mají závislostní / složk. stromy nebo ekviv. informaci
    - tektogramatické: nad syntaktickou info ještě např. valence, anafory, ...
  - ~ i neotagovaný korpus stále užitečný např. pro učení statistického modelu jazyka

~ morfologický, syntaktický: tektogramatický: Pražský korpus ~ oanotovaná podmnožina českého národního

~ Treebank: jen syntakticky oanotovaný
~ Český národní: jen syntaktický anotovaný ✓

(přijde mi že v předmětu PCLing se neuvádělo dost příkladů korpusů abych měl 2 příklady k danému typu)

3) Paralelní korpusy:

- korpusy se stejnými texty ve více jazycích
  ~ většinou ustanovené tím, že někdo vezme korpus jednoho jazyka a přeloží jej
- např. byla snaha přeložit korpus TreeBank (texty WSJ)
  - těžké: třeba lidi co umí nějen dobře přeházdat, ale rozumí bank. světu
- užitečné např. pro učení rozdílů mezi jazyky, učení překladačů...

Houška Petr
houskape@gmail.com

3

Čas . . 10. . min

7 Matematická lingvistika: formální jazyky a automaty (otázka pro studijní zaměření – 3 body)

1. Vysvětlete pojem neprojektivity v závislostních stromech.
2. Uveďte příklad nějaké neprojektivní věty.
3. Vysvětlete, do jaké třídy jazyků podle Chomského hierarchie patří přirozené jazyky, které obsahují neprojektivní konstrukce.

se nepodařilo otevřít
soubor

2) Soubor se nepodařilo otevřít.

1) Neprojektivita je jev, když slovo závisí na slově, které není před ním ani za ním, ale někde dále. V závislostních stromech se projevuje závislostní "šňůrou" napříč grafem. Např. u příkladě (2) "soubor" - na začátku závisí na slově "otevřít" (na konci)

3) Není že jít o jazyky bezkontextové. ~~Tedy z~~ Nepoj. konstrukce totiž nelze bezkontextovými gramatikami generovat. U přirozených jazyků jde tedy o jazyky kontextové.

U případě složitějších (nezvyklé u přívor jazyků) by mohlo jít o jazyky vyšší úrovně.

3

Houška Petr

houskape@gmail.com

Čas . 20. min. .

9 Matematická lingvistika: základy teorie informace (otázka pro studijní zaměření – 3 body)

1. Předpokládejme, že jednoduchý jazyk L obsahuje pouze 8 znaků (p, t, k, s, h, a, i, u) a že tento jazyk je jejich náhodnou posloupností s následujícími pravděpodobnostmi:

p	t	k	s	h	a	i	u
1/8	1/8	1/16	1/8	1/16	1/4	1/8	1/8

Vypočítejte entropii znaků jazyka L.

2. Při podrobnějším zkoumání jazyka L bylo odhaleno používání slabik. Předpokládejme, že jazyk je náhodnou posloupností nezávisle generovaných slabik souhláska-samohláska s následujícími pravděpodobnostmi:

	a	i	u	
p	1/8	1/8	0	1/4
t	1/8	1/16	1/16	1/4
k	1/16	0	1/16	1/8
s	1/8	1/16	1/16	1/4
h	1/16	0	1/16	1/8
	1/2	1/4	1/4	

Dále pracujeme s nově zjištěnými pravděpodobnostmi podle uvedené tabulky. Tabulka pravděpodobností slabik může být interpretována jako sdružená pravděpodobnost dvou náhodných proměnných C a V:

- C: slabika začíná souhláskou $c \in \{p, t, k, s, h\}$
- V: slabika ~~začíná~~ končí samohláskou $v \in \{a, i, u\}$

Vypočítejte vzájemnou informaci $I(C;V)$.

s dovolením nebudu dopočítávat

1) $\sum_{x \in \{p,t,\dots\}} p(x) \cdot \log\left(\frac{1}{p(x)}\right) = \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot 3 + \frac{1}{16} \cdot 4 + \frac{1}{8} \cdot 3 + \frac{1}{16} \cdot 4 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot \dots$

2)

$I(C;V) = E(C) - E(C|V) = E(V) - E(V|C)$

$= \frac{3}{2} - \left(\frac{1}{8} \cdot 1 + \frac{1}{8} \cdot 1 + \frac{1}{16} \cdot 1 + \frac{1}{8} \cdot 1 + \frac{1}{16} \cdot 1 + \frac{1}{8} \cdot 1 + \frac{1}{16} \cdot 2 + 0 + \frac{1}{16} \cdot 2 + 0 + 0 + \frac{1}{16} \cdot (2, 1, 2, 1) \right) =$

$x \in \{a, i, u\}$, $y \in \{p, t, k, s, h\}$

$p(a|p) \neq 1$

$p(a|p)$

s dovolením nebudu dopoč.

2

Houška Petr
houskape@gmail.com

Čas 9:30 – 10:03

3 - body

1 Automaty a gramatiky (3 body)

1. Definujte *zásobníkový automat* a popište všechny složky definice.
2. Definujte přijímání jazyka prázdným zásobníkem a přijímání jazyka koncovým stavem.
3. Zkonstruujte *deterministický* zásobníkový automat přijímající jazyk $L = \{u\#v \mid u, v \in \{a, b\}^*, |u| \geq |v|\}$.

druhá strana

1) Zás. automat: $(X, Q, Y, \delta, q_0, y_0, F)$

X: množina znaků slova (tj. čtených)

Q: množina stavů automatu

q_0: výchozí stav automatu

F: množina stavů ukončujících automat

(pokud stavíme automat, aby přijímal dočtením s prázdným zás., tak může být prázdná)

Y: množina znaků, které mohou být na zásobníku

y_0: výchozí znak na zásobníku

δ: přechodová fce

- deterministická: $X \times Q \times Y \to Q \times Y^*_{FINITE} \times \{0, 1\}$

(X – čtené písmeno, $\cup \{\lambda\}$; Q – starý stav; Y – znak, co pop-ne / sebere ze zásobníku; Q – nový stav; Y^* – znaky co dá na zásobník, konečná pos. znaků z Y; {0,1} – posune / neposune se dál)

- nedeterm. ~~stejně silný~~, & přechod stejná jen pravá strana 2^...

det. zás. autom. je slabší než nedetermin.

2) Prázdným zás: $w \in L$: $\delta^*(w, q_0, y_0) \to (q, \{\emptyset\}, 1)$ (nosná přechod. fce) ? ?

- tj. pokud automat přečte celé slovo & má prázdný zásobník, na stavu nezáleží

Koncovým stavem: $w \in L$: $\delta(w, q_0, y_0) \to (q_F, y^*, -)$ & $q_F \in F$?

- tj. pokud automat přečte slovo & dostane se do stavu, který náleží do množiny koncových stavů

~~3) 2 dva stavy: před # pro 1 znak přidán na zás. 2x ... po #~~

je potřeba uvést, že poč. zás. symbol je y_1

3) $u \neq v : |u| \geq |v|$ nedal zás. správa

Houška Petr
houskape@gmail.com

3

Čas 10:10 – 10:25

2 Algoritmy a datové struktury: minimální kostra (3 body)

1. Definujte minimální kostru (souvislého neorientovaného) grafu.
2. Popište libovolný algoritmus pro hledání minimální kostry.
3. Charakterizujte hrany, které lze přidat k podgrafu minimální kostry tak, že vznikne opět podgraf minimální kostry.

1) Kostra grafu je takový podgraf, který obsahuje všechny vrcholy, je stromem, je spojitý.

Pro ohodnocený graf (hrany hodnoty) je min. kostra taková kostra, že # kostra taková jejíž součet cen hran by byl ostře nižší.

2) Baruvkův algoritmus:
- začnu s lesem jednotlivých vrcholů bez hran
- v každém kole projdu všechny stromečky a vždy spojím daný stromeček s lib. jiným nejlevnějším možným hranou ! co předpokládá
- takto v každém kroku alespoň zpoloviční počet stromů ⇒ nakonec jen jeden ⇒ kostra - $\mathcal{O}(m \cdot \log(n))$
- je havelská kostra určitě, vždy spojují nespojené stromy, minimalnost z 3.

3) Jde právě o nejmenší hranu, která z daného podgrafu vychází. { nejlevná / nejmenší
Platí totiž - tzv. řezové lemma. Pro každý hranový řez grafu patří minimální hrana daného řezu do min. kostry. Důkaz je sporem, pokud ne, lze nějakou z nahradit tou nejmenší spor s minimalitou.

Δ v Baruvkově alg. přidáme právě hrany z řezů.

3

Houška Petr
houskape@gmail.com

Čas 11:15 – 11:54

4 Programovací jazyky: principy objektového návrhu (3 body)

Mějme „tradiční" souborový systém se soubory a hierarchickými adresáři. Soubory i adresáře mají svoje jméno, vlastníka, skupinu a přístupová práva ala UNIX (práva `rwx` pro vlastníka, skupinu a ostatní) a základní atributy (velikost a čas modifikace). Kromě souborů a adresářů souborový systém podporuje i symbolické odkazy (jak na soubory, tak i na adresáře).

1. Navrhněte sadu tříd, které budou objektovým způsobem reprezentovat adresářovou strukturu tohoto souborového systému. Instance navržených tříd mají umožňovat získávat informace o souborech/adresářích, tj. získat jejich jméno a jméno včetně celé cesty od kořene souborového systému, práva, atributy, u adresářů získat seznam položek tvořící obsah adresáře. (Do návrhu ale nezahrnujte metody pro další manipulaci se souborovým systémem jako vytváření, mazání, čtení či zápis souborů.)

   Při tvorbě tříd respektujte pravidla objektového návrhu. Čeká se návrh objektového modelu, tedy v tomto bodě není třeba psát těla metod či privátní atributy.

2. Do návrhu přidejte metodu `walk()` (včetně těla), která, pokud je zavolána na adresáři, projde do hloubky celý adresář včetně podadresářů a aplikuje na všechny navštívené soubory/adresáře *operaci*, která je jako parametr předána metodě `walk()`. Operace by měla být ideálně předávána jako *lambda výraz*. Ukažte zavolaní metody `walk()` s operací, která vytiskne jméno souboru/adresáře.

   Ve vašem řešení dejte pozor na možnou existenci symbolických odkazů na adresáře – operace se na každý navštívený element smí provést právě jednou.

Pro řešení úlohy si vyberte jeden z jazyků Java, C++, C#. Při hodnocení odpovědi nebudou uvažovány drobné syntaktické chyby, ale obecně by použité konstrukce měly odpovídat zvolenému programovacímu jazyku.

1) abstract
FSNode : IFsNode
string Name
Path Path // full path
[flags] RightsEnum Rights // bitové flagy práv
Common Attributes Attributes // struct spol. atributů
}

FSFile : FsNode {
Stream<byte> Content
FileAttributes FileAttributes // specifické souborové atributy
}
FSFolder : FSNode {
FolderAttributes FolderAttributes
IEnumerable<IFSNode> Content // soubory / file uvnitř adresáře
}

interface IFsNode {
string Name
Path Path
}

FSSymbolLink<T> : T : IFSNo
{
Path Path
T RefevencedObj
string Name // getter na name ref. obj
}

2) [illegible]

```
do FSFolder publicWalk (Func <FSNode> fun){

    HashSet <FSNode> Visited = new HashSet();
    Visited.Add(this);
    fun(this)
    this.Content.For
    walkThisFolder(fun, visited);

}

private walkThisFolder(Func<IFSNode fun, Hashset <IFSNode> Visited){
  foreach (var node in this.Content){
    [illegible]
    if(f is FSSymbolLink m
    if(n is FSSymbolLink <FSSymbolLin
    if(n is FSSymbolLink<FSFile>f) {fun(f.Referenced Obj)}   Visited.Add(f.Referenced Obj)
    if(n is FSSymbolLink <FSFolder> {fun(f.ReferencedObj); walkThisFolder(f.RefObj);   Visited
    if(n is FsFile

}

private walkThisFolder(Func<FSNode>, HashSet <FSNode> Visited){
 foreach(var node in this.Content){
  FSNode obj;
     if( node is FSSymbolLink<FSFile>f){ obj = f.RefObj}
     if(node is FS SL <FSFolder> f) { obj = f.RefObj}
     if(node is FSFolder f) {obj=f}
     if(node is FSFile f) {obj=f}

 if Pobj is FSFol
      if(Visited.Contains(obj)){ continue;}
      Visited.Add(obj)
      fun(obj)
      if(obj is FSFolder fol) {walkThisFolder(fun, Visited);

  }
            nejde zavolání s výpisem
}  Walk(item => Console.WriteLine(item.Name));
```