Bakalářské zkoušky (příklady otázek z informatiky)

podzim 2021

1 Chomského hierarchie (3 body)

1. Uveďte definici bezkontextové gramatiky a jazyka generovaného bezkontextovou gramatikou.
2. Zařaďte následující jazyk do Chomského hierarchie, tj. určete nejmenší třídu (nejvyšší typ) Chomského hierarchie, do které náleží, a dokažte, že náleží do Vámi uvedené třídy a nepatří do třídy menší.

$$L = \{a^i b^j \mid 0 \leq i \leq j \leq 2i\}$$

2 Minimální kostry (3 body)

1. Definujte problém minimální kostry a popište algoritmus pro tento problém o co nejlepší časové složitosti. Nemusíte detailně popisovat použité datové struktury, ale uveďte složitost operací na datových strukturách.
2. Dokažte korektnost popsaného algoritmu.
3. Popište co nejrychlejší algoritmus pro následující problém. Je zadán graf $G = (V, E)$ s nezápornými váhami na hranách $(w_e)_{e \in E}$, minimální kostra T, a zmenšení ceny nějaké hrany $f \in E$, což je nová váha w'_f menší než w_f. Úkol je nalézt minimální kostru T' vůči aktualizovaným vahám, tedy váha hrany $e \neq f$ je stále w_e, ale váha f je nyní w'_f místo w_f.

   (Jinými slovy, váha jedné hrany se libovolně sníží a úkol je najít minimální kostru za předpokladu, že známe minimální kostru vůči starým vahám.)

3 SQL dotazování (3 body)

Uvažujte tabulky PERSON a CAR se schématy PERSON(ID, Name, Age) a CAR(ID, Color, OwnerID). Tabulka PERSON obsahuje záznamy ([1, Tom, 25], [2, Jane, 30], [3, Adam, 29], [4, Max, 36]). Tabulka CAR obsahuje záznamy ([1, Red, 4], [2, Blue, 1], [3, Red, 2], [4, White, 4]).

Zapište výsledek následujících dotazů, v případě chyby v SQL dotazu místo s chybou vyznačte.

1. Select Name From PERSON Inner Join CAR On (ID = ID)
2. Select Max(Age) From PERSON Where Age < 30
3. Select Name From PERSON Where Age = Max(Age)
4. Select Name From PERSON Where ID Not In (Select OwnerID From CAR)
5. Select Color, Count(ID) From CAR Where ID = 1 Group By Color Having Count(ID) > 1
6. Select * From CAR Where OwnerID >= ALL (Select ID From PERSON)

4 Objektový návrh karetní hry (3 body)

Předpokládejte, že chceme implementovat tahovou hru pro více hráčů jako počítačovou verzi karetní hry ze světa, kde vládne magie. Základní kartou této hry je karta typu **bytost** – každá bytost má: **a) sílu (power)** = celé nezáporné číslo, **b) seznam schopností**, který může být prázdný, nebo obsahuje řádově jednotky položek. Síla i seznam schopností každé karty přítomné ve hře se může v průběhu hry měnit různými kouzly.

Schopnost (ability) je boolovská vlastnost charakterizovaná pouze tím, zda ji daná bytost má nebo nemá. Konkrétní schopnost může mít bytost v seznamu vícekrát – pro případné vyhodnocování efektů schopnosti je to ale ekvivalentní s tím, když ji má v seznamu právě jednou.

Pro řešení otázky si zvolte jazyk C#, C++ nebo Java, vaši volbu vyznačte a ve zvoleném jazyce napište všechny části řešení.

Část A

Napište implementaci třídy `Creature`, která reprezentuje kartu typu bytost ve hře. Napište pouze deklaraci veřejného kontraktu takové třídy a neřešte implementační detaily jednotlivých metod a privátních datových položek. Zároveň navrhněte a popište veškeré další (netriviální) typy potřebné pro návrh kontraktu třídy `Creature`. Vás návrh musí podporovat minimálně: **a) přidání** schopnosti dané bytosti, **b) odebrání** jedné instance schopnosti, tj. pokud má bytost schopnost v seznamu vícekrát, tak se jen o 1 sníží počet výskytů, **c) zjištění**, zda bytost danou schopnost má, či nikoliv.

Podpora pro schopnosti musí být navržena rozšiřitelným způsobem, a to tak, abychom mohli nové schopnosti do hry přidávat bez úpravy původního kódu hry (např. v knihovně s rozšířením). Zároveň by systém schopností měl být navržený co nejvíce silně typovaný, aby případný překlep ve jménu schopnosti při programování algoritmů vyhodnocujících efekty schopností vedl k překladové, nikoliv běhové chybě programu.

Pomocí navržené infrastruktury napište implementaci/deklaraci schopnosti „*defender*“ a schopnosti „*lifelink*“

Dále napište příklad vytvoření instance `Creature` reprezentující bytost „*Gwynevere*“ s počáteční sílou 2 a počátečními schopnostmi „*defender*“ a „*lifelink*“.

Část B

Jiný druh karty v implementované hře je karta **enchantment** (okouzlení). Kartu enchantment je možné v průběhu hry kouzlem připojit ke vhodné kartě typu bytost – s cílovou kartou bytosti je karta enchantment spojená tak dlouho, dokud není jiným kouzlem odebrána. Součástí každého druhu karty enchantment je „algoritmus“, který u zvolené karty bytosti kontroluje, zda je na ni daná karta enchantment aplikovatelná.

Rozšiřte implementaci třídy `Creature` o podporu enchantmentů (a doplňte další k tomu potřebné typy) a napište příklad implementace enchantment karty „*Angelic Gift*“, kterou lze aplikovat pouze na bytosti se silou menší než 10, a pouze pokud má bytost současně schopnosti „*defender*“ a „*lifelink*“.

Část C

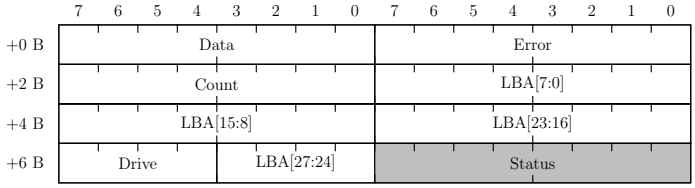
Pokud je karta enchantment připojena k nějaké kartě bytosti, tak má akce připojení na bytost nějaký efekt daný druhem enchantmentu – tento efekt může principiálně libovolně modifikovat vlastnosti dané bytosti. Doplňte implementaci tříd tak, aby po připojení enchantmentu k bytosti došlo k **aplikaci jeho efektu**, a po odpojení enchantmentu došlo ke **zrušení efektu**.

Rozšiřte implementaci enchantment karty „*Angelic Gift*“ tak, aby cílové bytosti přidala 3 body síly, a zároveň ji přidala schopnost „*flying*“ (předpokládejte, že schopnost „*flying*“ již máte definovanou podobným způsobem, jako jste definovali schopnosti „*defender*“ a „*lifelink*“). Po odpojení „*Angelic Gift*“ od bytosti tato bytost 3 body síly opět ztrácí, a stejně tak ztrácí schopnost „*flying*“ (pokud ji neměla i z jiného zdroje, nebo jako svoji počáteční schopnost).

Dále napište **a)** příklad kódu, který kartu „*Angelic Gift*“ připojí ke kartě bytosti „*Gwynevere*“, **b)** příklad kódu, který kartu „*Angelic Gift*“ od karty bytosti „*Gwynevere*“ odebere.

5 Architektura počítačů (3 body)

Pro práci s diskem poskytuje řadič disku rozhraní ve formě sady 8-bitových registrů, které jsou namapovány do paměti od adresy `0xE00007F8`. Rozložení (včetně offsetu od bázové adresy) a význam jednotlivých registrů ilustruje následující obrázek:



Registr	Význam
Data	Registr pro přenos dat z disku/na disk.
Error	Kód chyby, pokud zařízení indikuje chybu.
Count	Počet sektorů v požadavku na čtení/zápis.
LBA	Číslo prvního sektoru v požadavku na čtení/zápis (28-bitová logická adresa sektoru). Rozsah [m:n] indikuje bity LBA, které jsou přes registr přístupné.
Drive	Číslo zařízení, se kterým bude řadič komunikovat.
Status	Indikuje stav zařízení a zpracování požadavku.

Význam relevantních bitů v registru *Status* ilustruje následující obrázek:

Bit	Význam
BUSY	Indikuje, že zařízení pracuje (a tedy nepřijímá příkazy).
DRQ	Indikuje připravenost k přenosu dat jednoho sektoru (Data Request Ready).
ERROR	Indikuje chybu při zpracování požadavku.

V dalších částech předpokládejte, že veškerá komunikace probíhá v režimu **PIO** (*programmed I/O*), tedy **bez využití** přerušení nebo přímého přístupu do paměti (DMA). Pro bezpečný přístup k registrům řadiče máte k dispozici následující funkce:

```
uint8_t io_read_byte(uintptr_t address);
void io_write_byte(uintptr_t address, uint8_t value);
```

Parametry `address` a `value` odpovídají adrese registru a hodnotě, která má být do registru zapsána. Typ `uintptr_t` reprezentuje celá nezáporná čísla v rozsahu nutném pro uložení adresy.

Část A

Předpokládejte, že k řadiči je možné připojovat pouze disky se sektory o velikosti **512** bajtů. Na základě informací o rozhraní řadiče zodpovězte následující otázky:

1. Jaká je maximální velikost požadavku na čtení/zápis?
2. Jaká je maximální kapacita disku, se kterou je možné pracovat?

Odpovědi zdůvodněte a hodnoty uveďte v bajtech (s předponou odpovídající vhodnému dvojkovému řádu).

Část B

Napište funkci (pseudokód), která nakonfiguruje řadič pro zpracování jednoho požadavku na přístup k disku. Požadovaná funkce má následující signaturu:

```
void setup_request(uint8_t drive, uint32_t lba, uint8_t count);
```

Parametry `drive`, `lba`, a `count` odpovídají číslu zařízení, číslu počátečního sektoru požadavku a počtu sektorů. Předpokládejte, že zařízení je připraveno ke komunikaci, a že aktivace požadavku se provádí v jiné funkci (a nic z toho tedy funkce `setup_request` nemusí řešit).

Část C

Napište funkci, která přenese data z právě dokončeného požadavku na čtení z disku. Disk přenáší data po sektorech a připravenost k přenosu dat jednoho sektoru je indikována nastavením bitu `DRQ` na hodnotu `1` a bitu `BUSY` na hodnotu `0`. Pokud nedošlo při čtení z disku k chybě, bude bit `ERROR` nastaven na hodnotu `0`, v opačném případě bude nastaven na hodnotu `1`.

Samotný přenos probíhá v režimu PIO, data je tedy nutné vyčíst po bajtech z registru *Data*. Po přenesení posledního bajtu každého sektoru je bit `DRQ` nastaven na hodnotu `0` a disk může dočasně indikovat činnost pomocí bitu `BUSY`. Před přenosem dalšího sektoru je tedy nutné počkat, než bude disk připraven k přenosu dat a ověřit, zda nedošlo k chybě. Požadovaná funkce má následující signaturu:

```
unsigned int read_sectors(unsigned int count, uint8_t * buffer);
```

Parametr `count` reprezentuje **počet sektorů**, které mají být přečteny. Parametr `buffer` obsahuje ukazatel na pole bajtů, do kterého mají být data (všech sektorů) uložena. Návratová hodnota indikuje počet přečtených sektorů. Ten může být menší, než počet požadovaných sektorů, pokud při čtení dojde k chybě. Přítomnost chyby stačí testovat před začátkem přenosu každého sektoru.

6 Transportní protokoly a jejich vlastnosti (3 body)

Vysvětlete následující pojmy. Stručně také vyjmenujte, čeho jejich prostřednictvím chceme dosáhnout/jaké problémy v rámci jejich zajištění musíme řešit.

1. Proudový přenos (stream transmission)
2. Spojovaný přenos (connection-oriented transmission)
3. Spolehlivý přenos (reliable transmissions)

Navrhněte a pečlivě popište, jak byste tyto vlastnosti zajistili u hypotetického (přesto smysluplně a efektivně fungujícího) protokolu na transportní vrstvě L4. Inspirovat se můžete existujícími protokoly v architektuře TCP/IP, stejně tak můžete navrhnout vlastní řešení.

7 Optimalizační metody (3 body)

Řezbářská dílna Pech a syn se během prvního dubnového týdne plánuje soustředit na dva výrobky: obra a vlka.

Na výrobu jednoho obra je třeba hranol lipového dřeva o rozměrech 250 x 120 x 340 mm prodávaný za 800 Kč, na výrobu jednoho vlka je potřeba hranol olšového dřeva o rozměrech 250 x 100 x 270 mm prodávaný za 600 Kč. Vyřezání jednoho obra trvá 4 hodiny (jedno jakému řezbáři z dílny, všichni jsou stejně zruční), vyřezání jednoho vlka 5 hodin. Obry i vlky je před prodejem třeba povrchově upravit, což u jednoho obra trvá dvě hodiny, u jednoho vlka hodinu. V dílně pracují na plný úvazek celkem čtyři řezbáři, každý 35 hodin týdně. Řezbářství má k dispozici na nákup dřeva 20 tis. Kč, na vyřezávání týden, na povrchovou úpravu zajišťovanou externími spolupracovníky 48 hodin. Jeden obr má prodejní cenu 2050 Kč, jeden vlk 2100 Kč.

Cílem je naplánovat výrobu tak, aby celková (souhrnná) prodejní cena vyrobených obrů a vlků byla co nejvyšší a přitom bylo výrobu možno realizovat za výše uvedených omezení (tj. omezení na cenu nakupového materiálu, časové možnosti řezbářů a externích spolupracovníků).

1. Formulujte problém jako úlohu celočíselného LP.
2. Vyřešte lineární relaxaci formulace z bodu 1; doporučujeme využít grafickou metodu. (Jakou užitečnou a nesamozřejmou vlastnost má polytop všech přípustných řešení lineární relaxace problému?)
3. Najděte celočíselné optimum.

8 Kódy (otázka studijního zaměření – 3 body)

Definujte perfektní kód a uveďte příklad.

Existuje perfektní binární kód délky 9 a vzdálenosti 3? Pokud ano, uveďte příklad, pokud ne, ukažte proč.

9 Kombinatorická geometrie (otázka studijního zaměření – 3 body)

Uvažujme V-mnohostěn $M \subseteq \mathbb{R}^3$ zadaný jako konvexní obal devíti bodů:

$(0,0,1), (0,1,0), (1,0,0), (0,-1,0), (-1,0,0), (1,1,-1), (1,-1,-1), (-1,-1,-1), (-1,1,-1)$

1. Popište M jako H-mnohostěn.
2. Popište duální mnohostěn M^* (pokud možno geometricky, ale pokud se vám to geometricky nepodaří, tak alespoň kombinatoricky).
3. Jsou M a M^* kombinatoricky ekvivalentní? (Zdůvodněte.)

10 Aproximační algoritmy (otázka studijního zaměření – 3 body)

1. Zaveďte pojem "aproximační faktor algoritmu".
2. Na problému rozvrhování úloh na identických strojích ilustrujte techniku lokálního prohledávání (tj. popište problém a algoritmus, který problém předepsanou technikou řeší, uveďte aproximační poměr algoritmu).
3. Analyzujte výše popsaný algoritmus (tj. dokažte správnost, odhad aproximačního poměru a časové složitosti).

11 Relační databáze (otázka studijního zaměření – 3 body)

Mějme online obchod, který vede evidenci prodejů v relační databázi v tabulce *sales*. Tabulka *sales* má následující strukturu:

- **sale_id** (int): primární klíč
- **date** (date): datum uskutečnění prodeje
- **price** (int): celková cena prodeje

– **customer** (varchar): login uživatele, který nákup provedl

– **customer_full_name** (varchar): plné jméno uživatele ve tvaru „jméno příjmení“, který nákup provedl

– **address** (varchar): adresa pro doručení ve formátu „Město;Ulice;Číslo popisné;Poštovní Směrovací číslo“.

Jedná se o jedinou tabulku v systému.

1. Upravte schéma databáze (tabulka *sales*) tak, aby splňovalo 3. normální formu.
2. Pro upravené schéma z předchozího bodu napište SQL dotaz, který pro každou ulici v databázi vrátí součet všech prodejů (sloupeček **price**). V dotazu je třeba ošetřit situaci, kdy je v databázi uložena ulice, ale není k ní uveden žádný prodej. V takovém případě bude pro danou ulici vrácena hodnota 0.
3. Při používání upraveného schématu z 1. bodu se ukázalo, že je třeba optimalizovat vyhledávání prodejů pro daný den. Jak je možné tuto optimalizaci realizovat s využitím SQL serveru?

Při řešení můžete použít SQL dialekty běžně známých databází (Oracle, MySQL, PostgreSQL, MSSQL, ...).

12 Webové aplikace (otázka studijního zaměření – 3 body)

Mějme webovou aplikaci pro zobrazení hierarchického seznamu úkolů:

```
<ul id="item-list">
  <li><span class="item-title" data-ref="001">Výroba skříně</span>
    <ul>
      <li><span class="item-title" data-ref="002">Nákup materiálu</span>
        <ul>
          <li><span class="item-title" data-ref="003">Dřevo</span></li>
          <li><span class="item-title" data-ref="004">Konfirmáty</span></li>
        </ul>
      </li>
      <li><span class="item-title" data-ref="005">Formátování DTD</span></li>
    </ul>
  </li>
  <li><span class="item-title" data-ref="006">Připravit oběd</span>
    <ul>
      <li><span class="item-title" data-ref="007">Uvařit těstoviny</span></li>
      <li><span class="item-title" data-ref="008">Nastrouhat sýr</span></li>
    </ul>
  </li>
<ul>

<script>
 document.getElementById("item-list")
   .addEventListener("click", event => event.target.style.color = "red" );
</script>
```

Třída *item-title* není jinde použita.

1. Výše uvedenou webovou aplikaci je třeba rozšířit: uživatel může kliknout na „text“ libovolné položky (element **span**), tím dojde ke zvýraznění textu. Pokud byl předtím zvýrazněný jiný text, jeho zvýraznění se zruší. Zvýraznění je realizováno změnou barvy textu na červenou. O tuto implementaci se neúspěšně již pokoušel jiný kolega, viz obsah elementu **script** výše. Cílem je modifikovat, či nahradit, tento kód tak, aby implementoval požadovanou funkcionalitu.
2. Dalším požadavkem je zaznamenávat interakci uživatele s webovou aplikací – na jaké „texty“ (elementy **span**) uživatel klikl. Při kliknutí je třeba odeslat zprávu serveru na URL adresu **/api/v1/event**. Zpráva bude ve formátu JSON s následujícím obsahem:

```
{
  "action": "click",
  "ref": "003"
}
```

Hodnota pro klíč **ref** je hodnota *data-ref* atributu **span** elementu, na který uživatel klikl.

3. Webový server, na kterém běží aplikace z předchozích bodů, podporuje šifrování přenosu (HTTPS protokol) a je dostupný na adrese **click-business.com**. Dnes, 8.9.2021, vrátil server certifikát s následujícím obsahem:

```
Issuer:     CN = Let's Communicate
Valid from:  1. 1.2020  0:00:01
Valid to:   31.12.2020 23:59:59
Subject:    CN = service.click-business.com
...
```

   Na straně klienta je dále dostupný pouze následující certifikát:

```
Issuer: CN = Let's Communicate
Subject: CN = Let's Communicate
Valid from:  1. 1.2020  0:00:01
Valid to:   31.12.2030 23:59:59
...
```

   Tento certifikát náleží certifikační autoritě a prohlížeč jej považuje za důvěryhodný.

   Pro každý certifikát jsou uvedeny pouze vybrané hodnoty, chybějící hodnoty jsou z pohledu naší úlohy validní.

   Je bezpečné na základě serverem poskytnutého certifikátu zaslat zprávu z předchozí úlohy (tedy provést POST na URL **https://click-business.com/api/v1/event**)? Svou odpověď zdůvodněte.

Pokud v řešení použijete funkce poskytnuté webovým prohlížečem, krátce popište jejich význam (zejména pokud si nejste jisti přesným názvem).

13 XML (otázka studijního zaměření – 3 body)

Mějme následující XML dokument:

```
<data>
  <items>
    <item active="1">
      <price>27</price>
      <title>Banány</title>
      <unit>kg</unit>
      <price-history>
        <price>23</price>
        <price>25</price>
        <price>24</price>
      </price-history>
    </item>
    <item active="1">
      <price>12</price>
      <title>Rajčata</title>
      <unit>kg</unit>
      <price-history>
        <price>12</price>
        <price>12</price>
      </price-history>
    </item>
    <item active="0">
      <price>16</price>
      <title>Okurka</title>
      <unit>kus</unit>
      <price-history>
        <price>23</price>
        <price>20</price>
        <price>25</price>
```

```
      </price-history>
      </item>
  </items>
</data>
```

1. Co bude výsledek dotazu

   ```
   //item[price-history/price < 20 and unit = "kg"]/title
   ```

2. Co bude výsledek dotazu

   ```
   data//*[self::price][position() > 2]/../..[@active="1"]/unit
   ```

3. Napište dotaz, který vybere elementy s názvy produktů, které jsou aktuálně levnější než kdy dříve. Tedy hodnota v `price` je nižší než všechny hodnoty v `price` pod elementem `price-history`.

14 Kódy (otázka studijního zaměření – 3 body)

Definujte perfektní kód a uveďte příklad.

Existuje perfektní binární kód délky 9 a vzdálenosti 3? Pokud ano, uveďte příklad, pokud ne, ukažte proč.

15 Hranová barevnost grafů (otázka studijního zaměření – 3 body)

Zadefinujte hranovou barevnost grafu.

Nechť G je kubický (3-regulární) souvislý graf. Která z následujících tvrzení platí a proč?

1. Má-li G perfektní párování, pak je hranově 3-obarvitelný.
2. Je-li G hranově 3-obarvitelný, pak má perfektní párování.
3. Má-li G Hamiltonovskou kružnici, pak je hranově 3-obarvitelný.
4. Je-li G hranově 3-obarvitelný, pak má Hamiltonovskou kružnici.

16 Dobré uspořádání (otázka studijního zaměření – 3 body)

1. (a) Napište definici dobrého uspořádání. (Pojem *uspořádání* definovat nemusíte).
   (b) Napište definici ordinálního čísla. Pokud použijete pojem *tranzitivní množina*, definujte jej také.
2. Které z následujících množin jsou ordinální čísla? Odpovědi krátce zdůvodněte.
   (a) 2021
   (b) $\bigcup 2021$
   (c) $20 \cap 21$
   (d) ω
   (e) $\{\omega\}$
   (f) $\omega \cup \{\omega\}$
   (g) $\omega \cap \{\omega\}$
   (h) $\omega \cup \{\omega\} \cup \{\{\omega\}\}$

17 Morfologická, syntaktická a sémantická analýza přirozeného jazyka (otázka studijního zaměření – 3 body)

1. Vysvětlete pojmy *stemming*, *lemmatizace*, *tagování* a *morfologická analýza*. Uveďte příklady.
2. Vysvětlete pojem *Universal Dependencies*.

18 Základní formalismy pro popis přirozeného jazyka (otázka studijního zaměření – 3 body)

1. Jak se má Pražský závislostní korpus (PDT) k teorii Funkčního generativního popisu?
2. Pro vámi (vhodně) zvolenou větu zakreslete její syntaktický a tektogramatický strom dle PDT a popište jejich rozdíly.

19 Základy teorie informace (otázka studijního zaměření – 3 body)

X a Y jsou dvě náhodné veličiny a H je označení entropie. Uveďte, zda níže uvedené nerovnosti **platí** nebo **neplatí**. Pokud **platí**, dokažte. Pokud **neplatí**, uveďte protipříklad, nebo dokažte platnost opačné nerovnosti. Nápověda: použijte vztah mezi vzájemnou informací a entropií.

1. $H(X) \geq H(X|Y)$
2. $H(X) + H(Y) \leq H(X, Y)$

20 Rastrová a vektorová grafika (otázka studijního zaměření – 3 body)

1. Uveďte základní princip rastrové a vektorové grafiky, uveďte příklady grafických formátů, vektorových i rastrových.
2. Jaké jsou výhody vektorové grafiky a pro které operace bychom ji preferovali? Uveďte konkrétní příklady.
3. Naznačte principy komprese rastrových obrázků (nemusíte uvádět detaily).

21 Textury ve 3D grafice (otázka studijního zaměření – 3 body)

1. Jaký je rozdíl mezi 2D a 3D texturou? Uveďte typické aplikace těchto dvou přístupů.
2. Jaké jsou hlavní výhody a nevýhody 3D textur při použití v ray-tracingu?
3. Jak je možné napodobit vnitřní strukturu materiálu (dřevo, mramor) pomocí textury nebo procedurální definice v paprskovém zobrazovači (ray-tracing, path-tracing)? Pozn.: pište o simulaci struktury materiálu, ne o zobrazovači!

22 Urychlování ray tracingu (otázka studijního zaměření – 3 body)

Rekurzivní sledování paprsku je algoritmus, který je ve své naivní podobě velmi pomalý, zejména při zobrazování rozsáhlých scén. Pro jednoduchost budeme dále uvažovat jen scény složené pouze z trojúhelníků.

1. Identifikujte, která část algoritmu je zodpovědná za velmi dlouhý výpočet u rozsáhlých scén. Jaká by byla asymptotická složitost v té nejjednodušší/naivní variantě algoritmu? (Asymptotickou složitost se budeme snažit v dalších bodech vylepšit.)
2. Navrhněte jeden konkrétní přístup pro urychlení výpočtu. Není potřeba popisovat algoritmy do detailů, stačí dostatečně podrobné slovní vysvětlení. Nemusíte popisovat ostatní principy urychlení, popište jen ten, který jste si zvolili!

   Zřetelně oddělte dvě fáze: a) co je potřeba udělat při předzpracování dat, a b) co se používá při výpočtu jednotlivých pixelů obrázku.
3. Odhadněte asymptotickou složitost vašeho návrhu, stručně odůvodněte. Uveďte podmínky (charakter nebo struktura scény), za kterých je možné (sub)optimálního urychlení dosáhnout.

23 Sockety (otázka studijního zaměření – 3 body)

Mějme následující pseudokód jednoduchého socketového serveru nad TCP/IP.

```
socket = create_socket(AF_INET, SOCK_STREAM)
socket.bind('127.0.0.1', 8080)
socket.listen()
while True:
    cl = socket.accept()
    data = cl.recv(2048)
    cl.send(md5sum(data))
```

```
cl.close()
```

Tento server spustíme na stroji `test.example.com` (s veřejnou IP adresou `1.2.3.4`).

1. Nepředpokládáme-li další nastavení, z jakých strojů bude tento server dostupný? Proč? Jakou cílovou adresu a port musí klient použít pro připojení k tomuto serveru? Mohl by být server provozován i na jiném portu (jaká jsou obecná omezení/doporučení pro výběr čísla portu)?
2. Po připojení k serveru klient odeslal jako svůj požadavek 2048 znaků `A`. Jaká bude odpověď serveru (vysvětlete, nepočítejte skutečný výsledek)? Jak se odpověď změní, odešle-li klient znaků 4096? Předpokládejte, že interní buffery mají určitě velikost větší než 4 KiB.
3. Předpokládejme, že se k tomuto serveru připojí najednou více uživatelů. Jakým způsobem operační systém, potažmo samotný server, rozliší jednotlivá spojení?
4. Jaký transportní protokol bude použit k přenosu (TCP, UDP, ...)? Jaké má vlastnosti (spojovaný/nespojovaný, spolehlivý/nespolehlivý, atd.)?
5. Jak by se změnilo chování serveru, pokud bychom místo `SOCK_STREAM` použili `SOCK_DGRAM` (a případně odpovídajícím způsobem upravili zbývající část kódu se zachováním aplikační logiky)?

24 Metoda zpětného učení (otázka studijního zaměření – 3 body)

Na úrovni linkové vrstvy L2 detailně vysvětlete, k čemu se používá a jak funguje metoda zpětného učení.

Mějme směrovač (switch) mající vlastní adresu `CC-DD-EE-11-11-11` a porty symbolicky označené jako S, T, U a V, jehož předávací (forwardovací) tabulka aktuálně vypadá takto:

S	`CC-DD-EE-22-22-22`
S	`CC-DD-EE-33-33-33`
T	`CC-DD-EE-44-44-44`
U	`CC-DD-EE-55-55-55`
U	`CC-DD-EE-66-66-66`

Jak přesně budou zpracovány následující L2 rámce (obsahující standardní IPv4 datagram) postupně přijaté na portu S a jak se případně změní předávací tabulka? Odůvodněte.

1. Rámec od odesilatele `CC-DD-EE-33-33-33` určený pro příjemce `CC-DD-EE-55-55-55`
2. Rámec od odesilatele `CC-DD-EE-22-22-22` určený pro příjemce `CC-DD-EE-33-33-33`
3. Rámec od odesilatele `CC-DD-EE-22-22-22` určený pro příjemce `CC-DD-EE-77-77-77`
4. Rámec od odesilatele `CC-DD-EE-88-88-88` určený pro příjemce `CC-DD-EE-22-22-22`
5. Rámec od odesilatele `CC-DD-EE-33-33-33` určený pro příjemce `FF-FF-FF-FF-FF-FF`
6. Rámec od odesilatele `CC-DD-EE-22-22-22` určený pro příjemce `CC-DD-EE-11-11-11`

25 Překlad IP adres (otázka studijního zaměření – 3 body)

V rámci malé firemní sítě mějme jediný směrovač (router) s dynamickým překladem mezi privátními a veřejnými adresami (PAT nebo také NAPT), jeho vnější rozhraní (označme jej jako A) má veřejnou IP adresu 191.1.1.111 s CIDR prefixem 16 a vnitřní rozhraní (B) privátní adresu 10.0.0.1 s maskou sítě 255.255.255.0. Veškerý provoz z vnitřní sítě mimo ni je směrován do vnější. Předpokládejte, že překladová tabulka aktuálně obsahuje následující záznamy:

Protokol	Vnitřní adresa	Vnější adresa
TCP	10.0.0.11:50333	191.1.1.111:60777
UDP	10.0.0.22:50444	191.1.1.111:60888
TCP	10.0.0.22:50555	191.1.1.111:60999

Jak přesně budou zpracovány následující IPv4 datagramy postupně přijaté směrovačem na vnitřním rozhraní B? Zaměřte se i na případné změny v hlavičce a těle datagramu nebo změny v překladové tabulce. Vysvětlete a odůvodněte.

1. IP datagram s TCP segmentem od 10.0.0.11:50333 pro 202.2.2.222:80
2. IP datagram s UDP datagramem od 10.0.0.22:50444 pro 255.255.255.255:22 (lokální broadcast)

3. IP datagram s TCP segmentem od 10.0.0.99:50555 pro 144.4.4.44:443
4. IP datagram s TCP segmentem od 10.0.0.11:50888 pro 10.0.0.1:80

Analogicky a ve stejném rozsahu popište, jak bude zpracován následující IPv4 datagram přijatý na vnějším rozhraní *A*.

5. IP datagram s TCP segmentem od 202.2.2.222:80 pro 191.1.1.111:60777

26 Návrhový vzor Model-View-Controller (MVC) (otázka studijního zaměření – 3 body)

Popište základní princip návrhového vzoru „Model-View-Controller (MVC)“ a role hlavních komponent (elementů) aplikace navržené podle tohoto vzoru.

Nakreslete diagram interakce těchto komponent, ve kterém zachytíte směr komunikace, význam předávaných informací mezi komponentami, a také interakci uživatele s danou aplikací (tedy v diagramu nějak vyjádřete, se kterými komponentami uživatel interaguje a jak primárně).

Soustřeďte se na podstatné aspekty a zanedbejte méně významné technické detaily. Vhodně doplňte jednotlivé prvky diagramu textovým komentářem.

Vytvořte jednoduchý konkrétní příklad z oblasti webových aplikací, se zaměřením na jejich serverové části. Stručně popište činnosti, které se typicky provádějí v jednotlivých komponentách aplikace navržené podle vzoru MVC – tedy popište, jakou činnost v rámci takové webové aplikace provádí komponenta „Model“ a co má na starosti, a potom to samé ještě pro komponenty „Controller“ a „View“.

Vytvořte jednoduchý příklad rozhraní hlavních komponent ve smyslu kostry nějaké jednoduché webové aplikace. Použijte váš oblíbený programovací jazyk (C#, Java, C++, Python, apod.). Soustřeďte se na klíčové prvky těchto rozhraní s ohledem na interakci komponent (názvy akcí, předávaná data), a zanedbejte nepodstatné technické detaily.

27 Dependency injection (otázka studijního zaměření – 3 body)

Stručně vysvětlete koncept „dependency injection (DI)“. Zaměřte se přitom na základní princip, dále stručně popište, jaký problém koncept řeší, a okomentujte hlavní výhody použití tohoto způsobu konstrukce softwarových systémů oproti alternativám, a to zejména s ohledem na modularitu návrhu a testování jednotlivých komponent systému.

Vytvořte jednoduchou ukázku použití DI ve vybraném objektově-orientovaném programovacím jazyce (C#, Java, C++, Python, apod.). Uvažujte například aplikaci pro zpracování objednávek v online obchodě, kde hlavní proces komunikuje s moduly pro sklad zboží, platební bránu, a přepravní službu. Stačí jenom hrubá kostra aplikace se 4-5 třídami (komponentami) a k tomu kostra testu pro jednu vybranou metodu z jedné třídy (komponenty), která má závislost aspoň na jedné další třídě.

28 Fork-join model a vytváření korektních programů s několika vlákny (otázka studijního zaměření – 3 body)

Popište fork-join model jako způsob řešení paralelizace výpočtu, respektive jako způsob návrhu programů s několika paralelními výpočty. Stručně diskutujte hlavní výhody modelu fork-join oproti řešení založeném na systémových (low-level) vláknech, a to především z pohledu čitelnosti kódu a eliminace chyb v synchronizaci vláken. Vytvořte krátkou ukázku použití fork-join na jednoduché paralelizovatelné úloze, jako například třídění (mergesort) nebo hledání souboru na disku v rozsáhlém stromu adresářů. Použijte na vytvoření ukázky váš oblíbený programovací jazyk (C#, Java, C++, apod.).

Stručně vysvětlete koncept „immutability“ (objektů, datových struktur) a jeho použití (výhody) za účelem vývoje korektních a bezpečných programů (ve smyslu „thread-safety“) s několika paralelními výpočty. Rozšiřte ukázku použití fork-join tak, aby se v daném programu v maximální rozumné míře používaly „immutable“ objekty a datové struktury (tam kde to dává smysl).

Při vytváření ukázkových programů se soustřeďte na klíčové aspekty vzhledem k zadání otázky. Zanedbejte nepodstatné detaily.

29 Synchronizační primitiva (otázka studijního zaměření – 3 body)

Uvažujme následující triviální implementaci semaforu uvnitř operačního systému. Implementace předpokládá hardware s jediným procesorem, pro ohraničení kritické sekce využívá prosté zakázání přerušení.

Sémantika volaných funkcí je popsána níže.

```
typedef struct {
  int value;
  list_t queue;
} sem_t;
```

```
void sem_init(sem_t* sem, int value) {
  list_init(&sem->queue);
  sem->value = value;
}
```

```
1  void sem_wait(sem_t* sem) {
2    bool ipl = interrupts_disable();
3    while (sem->value <= 0) {
4      list_append(&sem->queue, thread_current());
5      interrupts_restore(ipl);
6      thread_suspend();
7      ipl = interrupts_disable();
8    }
9    sem->value--;
10   interrupts_restore(ipl);
11 }
```

```
12 void sem_post(sem_t* sem) {
13   bool ipl = interrupts_disable();
14   sem->value++;
15   while (!list_is_empty(&sem->queue)) {
16     thread_t* thread = list_pop(&sem->queue);
17     thread_wakeup(thread);
18   }
19   interrupts_restore(ipl);
20 }
21
22
```

Funkce pro práci se seznamem (`list_*`) by typicky zahrnovaly ještě nutná přetypování apod., která pro jednoduchost vynecháváme (sémantika `append` je zřejmá, `pop` odstraní (a vrátí) ze seznamu první prvek).

Funkce `interrupts_disable` zakáže na procesoru přerušení a vrátí aktuální stav (povoleno/zakázáno). `interrupts_restore` obnoví stav přerušení (povoleno/zakázáno).

Funkce `thread_suspend` uspí aktuální vlákno. Implicitně tím dojde k přeplánování; součástí kontextu vlákna je i stav přerušení (přerušení jsou povolena nebo zakázána); tj. při přeplánování je stav přerušení (povoleno/zakázáno) obnoven podle kontextu nově běžícího vlákna.

Funkce `thread_wakeup` přepne vlákno do stavu `READY` (tj. může být naplánované); okamžik, kdy se vlákno skutečně rozběhne je ale řízen plánovačem (a nastane tedy typicky později). V případě, že je vlákno ve stavu `READY`, neudělá funkce nic.

1. Popište zamýšlenou sémantiku synchronizačního primitiva semafor (tj. funkcí `sem_init`, `sem_wait` a `sem_post`). Zaměřte se na funkci/účel z hlediska uživatele nikoliv na implementační detaily (tedy např. *`sem_init` slouží ke vstupu do kritické sekce a vpustí do ní maximálně jedno vlákno*, nikoliv, že *`sem_init` zakáže přerušení*).
2. Uvedená implementace obsahuje race-condition. Popište, za jakých okolností k ní dojde a jaké bude mít důsledky, využijte čísla řádků pro lepší popis (např. *plánovač přeplánuje vlákno A na řádku 34 a přepne na vlákno B*).
3. Implementace výše má ještě jeden nedostatek: nezaručuje, že některé z čekajících vláken nebude „čekat navždy“ (nesplňuje tedy tzv. *bounded waiting*). Navrhněte, jak by šlo tento nedostatek odstranit.

30 Verzovací systémy (otázka studijního zaměření – 3 body)

Předpokládejme projekt, který používá distribuovaný systém pro správu verzí (např. Git) a v repozitáři je pro jednoduchost jediný soubor (`data.txt`) s tímto obsahem:

```
1  aa
2  bd
3  cc
4  dd
5  ee
6  ff
7  gg
8  hh
9  ii
10 ij
```

K projektu mají přístup vývojáři Alice, Bob a David. Do hlavní vývojové větve (dále označována jen jako `master`) mohou zapisovat jen Alice a Bob, všichni vývojáři ale mají možnost vytvářet libovolné další větve a nahrávat do nich své změny.

1. Uvažujme, že Alice chce do `master` dostat změnu (zachycenou záplatou níže), kterou potřebuje prodiskutovat s Bobem.

```
--- a/data.txt
+++ b/data.txt
@@ -1,6 +1,6 @@
-aa
+AA
 bd
-cc
+CC
 dd
 ee
 ff
```

Popište, co jsou tzv. *feature branch* a jak je možné je využít ve výše uvedeném případě.

Napište, které operace nad repozitářem bude potřebovat Alice, aby navrhovanou změnu neprováděla přímo v `master` větvi a které Bob, aby mohl změnu vyzkoušet a poté zaintegrovat do hlavní větve (přesná syntaxe příkazů není vyžadována, důležité je zachytit hlavní koncepty).

2. Nezávisle na předchozím bodu proběhla následující změna: Bob si všiml překlepu na posledním řádku a opravu (na `jj`) ihned zanesl do hlavní větve (`master`) jako nový commit.

   Jaké informace budou k tomuto commitu přidruženy (tj. co by zobrazil třeba příkaz `git show ...`)? Přesná syntaxe/-formát není důležitý, zaměřte se na hlavní části a stručně naznačte, proč jsou součástí commitu (tj. kdo a kdy je může využít).

3. Předpokládejme, že ve stejné době, kdy Alice začala pracovat na změně z první části otázky vytvořil vlastní větev i David a opravil překlep na druhém řádku na `bb` a vytvořil nový commit s touto opravou.

   Bob zaintegroval Alicinu změnu a nyní by rád zaintegroval i opravu od Davida. Ta se ale dostala do tzv. konfliktu.

   Vysvětlete, co v tomto případě konflikt znamená a jakým způsobem budou Bob a David pokračovat, aby se Davidova změna mohla zaintegrovat. Předpokládejte, že reálně by úpravy byly náročnější a tíhu vyřešení by měl nést David.

   Pokud budete uvádět konkrétní příkazy, není nutné dodržet přesnou syntaxi, důležité je zachytit hlavní akce, které musí David provést (vlastní integrace do `master` od Boba by měla být stejná jako v předchozím bodě).

31 Návrhové vzory (otázka studijního zaměření – 3 body)

Následující fragment zobrazuje třídu `InputStream` z Javy, která reprezentuje proud bajtů pro čtení. Metoda `read` načte až `len` bajtů do pole `buf` (zápis začne na indexu `off`); metoda vrací počet skutečně načtených dat (nebo `-1` při dosažení konce vstupu).

Metoda `close` ukončuje práci se vstupem a je zde uvedena jako zástupce dalších metod z třídy `InputStream`.

Třída `FilterInputStream` nepřidává žádnou novou funkčnost a jen deleguje jednotlivá volání.

```
public abstract class InputStream {
    public abstract int read(byte[] buf, int off, int len);
    public abstract void close();
}

public class FilterInputStream extends InputStream {
    protected InputStream inp;
    public FilterInputStream(InputStream in) { inp = in; }
    public int read(byte[] buf, int off, int len) { return inp.read(buf, off, len); }
    public void close() { inp.close(); }
}
```

Výše uvedené třídy využijte jako základ pro vyřešení následujících úloh.

1. Využijte návrhový vzor *dekorátor* a napište implementaci třídy `DecompressingInputStream`, která bude dekomprimovat dodaný `InputStream`.

   Pro vlastní dekompresi využijte následující rozhraní, jehož implementaci dodá uživatel (v rámci volání `new`).

```
public interface Decompressor {
    public int getDecompressedBytes(byte[] buf, int off, int len);
    public void addCompressedBytes(byte[] buf, int off, int len);
}
```

Data k dekompresi dodá uživatel postupně voláním `addCompressedBytes`, dekomprimovaná data jsou čtena pomocí `getDecompressedBytes`. V obou případech se používá sémantika práce s částí pole `buf` od indexu `off` a maximální délce `len`.

Návratová hodnota `getDecompressedBytes` reprezentuje skutečnou délku *dekomprimovaných* dat. V případě, že nelze dekomprimovat další data, vrací metoda hodnotu `0` a uživatel by měl zavolat metodu `addCompressedBytes`, která dodá další blok komprimovaných dat. Jinými slovy: nedává smysl volat metodu `addCompressedBytes` dokud metoda `getDecompressedBytes` nevrátí hodnotu `0` (čímž signalizuje, že nemá další data k dekompresi).

V rámci implementace není nutné řešit chybové stavy, pro jednoduchost můžete předpokládat, že komprimovaná data jsou korektní. Implementace ale musí řešit dosažení konce vstupu (kdy `read` vrací `-1`).

2. Rozšiřte vaši implementaci tak, aby uživatel mohl volitelně spočítat kontrolní součet dekomprimovaných dat. Pro vlastní výpočet kontrolního součtu můžete předpokládat, že uživatel dodá instanci následujícího rozhraní (vlastní získání kontrolního součtu je v režii konkrétní implementace a není třeba ji v rámci zadání vůbec řešit).

   ```
   public interface Checksum {
       public void update(byte[] buf, int off, int len);
   }
   ```

3. Jak bude nutné vaši implementaci změnit, pokud by uživatel chtěl také spočítat kontrolní součet původních (komprimovaných) dat?