Bakalářské zkoušky (příklady otázek)

2023-09-14

1 Architektura počítačů a operačních systémů (společné okruhy)

Mějme následující třídu v pseudokódu:

```
 1  class LimitedStack {
 2  public:
 3          LimitedStack(int md) {
 4              currentDepth = 0;
 5              maxDepth = md;
 6              stack = new int[maxDepth];
 7          }
 8
 9          int getDepth() {
10              return currentDepth;
11          }
12
13          bool pushValue(int v) {
14              if (currentDepth >= maxDepth)
15                  return false;
16              stack[currentDepth] = v;
17              currentDepth++;
18              return true;
19          }
20
21  private:
22          int    maxDepth;
23          int    currentDepth;
24          int[]  stack; // int *stack; pro C++
25  };
```

Dále předpokládejme, že máme multiprocesorový systém a program si spustí několik vláken, která budou mít všechna přístup ke sdílené proměnné typu `LimitedStack`. Tato vlákna budou volat podle své potřeby funkce `getDepth` a `pushValue`.

1. Je možné, že v programu nastane jev zvaný *race condition*? Pokud ano, napište rozsahy řádek nebo vyznačte přímo v uvedeném kódu řádky, které představují kritickou sekci.
2. Je možné, aby nějakým způsobem nastala situace, kdy při volání funkce `pushValue` přeteče zásobník (program se bude pokoušet zapisovat mimo alokované pole)? Zdůvodněte.
3. Pokud se domníváte, že v uvedeném kódu může nastat race condition, pokuste se tento problém odstranit úpravou programu (včetně možného přidání nějakých nových řádek).

Kde je to důležité, ve své odpovědi uvažujte jazyk C#, C++ nebo Java (a vaši volbu vyznačte).

Nástin řešení

1. Ano, 10 a 14-17.

2. Ano, T1 i T2 vykonají řadek 14 souběžně. Přečtou shodnou proměnnou `currentDepth`, která má hodnotu `maxDepth`-1, takže test selže. Následně po testu je T2 plánovačem zastaveno, T1 funkci dokončí a zvětší `currentDepth`. Poté je T2 znovu naplánovano a při zápisu hodnoty použije zvětšený index sahající za pole.
3. Záleží na jazyku. V Javě triviálně použitím `synchronized` metod. C# buď použije `lock` na nějaký vedlejší objekt nebo se použije `[MethodImpl(MethodImplOptions.Synchronized)]`. C++ použije třeba `Mutex` a buď explicitní `lock` a `unlock` nebo lépe `lock_guard`. Pozor na odemknutí před každým `return`.

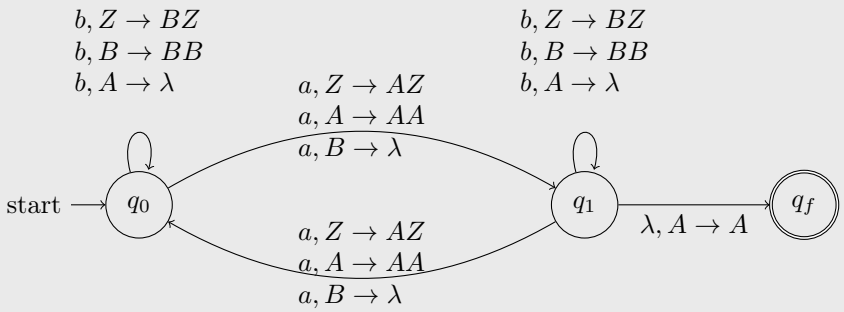
2 Zásobníkový automat (společné okruhy)

1. Uveďte definici *zásobníkového automatu*.
2. Sestrojte zásobníkový automat A, který přijímá právě řetězce $w \in \{a,b\}^*$, které mají více znaků a než b a zároveň počet a je lichý, tj.
$$L(A) = \{w \mid w \in \{a,b\}^* \ \&\ |w|_a > |w|_b \ \&\ (\exists k \in \mathbb{N}_0)|w|_a = 2k+1\}.$$
Za částečné řešení se uznává i automat rozpoznávající „více a než b“.

Nástin řešení

1. *Zásobníkový automat* je sedmice $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, kde
   - Q je konečná množina stavů,
   - Σ je konečná vstupní abeceda,
   - Γ je konečná zásobníková abeceda,
   - δ je přechodová funkce $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to \mathcal{P}_{FIN}(Q \times \Gamma^*)$,
   - $q_0 \in Q$ je počáteční stav,
   - $Z_0 \in \Gamma$ je počáteční zásobníkový symbol,
   - $F \subseteq Q$ je množina přijímacích stavů.

   V případě přijímání prázdným zásobníkem bez množiny přijímajících stavů.
2. Úkol řeší například zásobníkový automat A se třemi stavy: přijímacím q_f a q_0, q_1 rozlišujícími sudý a lichý počet a. Přebytečné znaky a resp. b si ukládáme na zásobník, adekvátně odebíráme.
   Formálně: $A = (\{q_0, q_1, q_f\}, \{a,b\}, \{Z, A, B\}, \delta, q_0, Z, \{q_f\})$, s přechodovou funkcí δ popsanou grafem



3 Souvislost grafů (společné okruhy)

Mějme graf G na 64 vrcholech, přičemž jeho vrcholy jsou označeny různými šesticemi nul a jedniček. Hrany tvoří:

- vrcholy, které se liší pouze v první souřadnici, např. 111010 a 011010 (tyto hrany tvoří perfektní párování),
- vrcholy, které mají první dva znaky totožné, např. 111010 a 110001, a také
- dvojice vrcholů $(111010, 101100)$, $(101100, 010110)$ a $(010110, 111011)$.

Jiné dvojice vrcholů spojeny nejsou.

1. Vyslovte Mengerovu větu (o vrcholové souvislosti a řezech) včetně formálního zavedení potřebných pojmů.
2. Určete, kolik hranově disjunktních cest vede mezi vrcholy 000000 a 111111.
3. Nalezněte nejmenší hranový řez a nejmenší vrcholový řez oddělující 000000 a 111111.

4 Geometrická řada (společné okruhy)

1. Definujte geometrickou řadu: řada $\sum_{n=0}^{\infty} a_n$ je geometrická, právě když $a_n = \dots$.
2. Uveďte, jaké součty má geometrická řada (v závislosti na příslušných parametrech).
3. Sečtěte řadu $\frac{4}{9} + \frac{8}{27} + \frac{16}{81} + \ddots$. Odpověď přiměřeně zdůvodněte.

Nástin řešení

1. ... právě když $a_n = q^n$ pro nějaké reálné číslo q.
2. Pro $|q| < 1$ má součet $1/(1-q)$, pro $q \geq 1$ má součet $+\infty$ a pro $q \leq -1$ součet nemá (součet neexistuje).
3. Součet je $(2/3)^2 \sum_{n\geq 0}(2/3)^n = (2/3)^2/(1-2/3) = 4/3$.

5 Databáze (specializace DW)

Je dán následující transakční rozvrh S:

T_1	T_2	T_3
$R(A)$		
	$W(B)$	
		$W(A)$
$W(B)$		
	$R(A)$	
		$W(B)$
COMMIT		
	COMMIT	
		COMMIT

1. Je rozvrh S konfliktově uspořádatelný (conflict–serializable)? Své rozhodnutí vysvětlete.
2. Je rozvrh S zotavitelný? Své rozhodnutí vysvětlete. Pokud ne, bylo by možné zotavitelnosti dosáhnout bez toho, že by se změnilo současné vzájemné pořadí operací R a W? Jak?

Do rozvrhu S byly doplněny operace pro uzamykání L_X (exkluzivní zámek) a L_S (sdílený zámek) s cílem získat rozvrh odpovídající požadavkům pro striktní dvoufázový uzamykací protokol (strict 2PL).

T_1	T_2	T_3
$L_S(A)$ $R(A)$		
	$L_X(B)$ $W(B)$	
		$L_X(A)$ $W(A)$
$L_X(B)$ $W(B)$		
	$L_S(A)$ $R(A)$	
		$L_X(B)$ $W(B)$
COMMIT		
	COMMIT	
		COMMIT

3. Bylo cíle dosaženo? Své rozhodnutí vysvětlete.

Nástin řešení

1. Ano, je. Precedenční graf obsahuje hrany $T_1 \to T_1$, $T_2 \to T_1$, $T_2 \to T_3$ a $T_1 \to T_3$. V grafu není orientovaný cyklus. Je tedy topologicky uspořádatelný.
2. Ne, není. T_2 čte A, zapsané v T_3, ale dělá $COMMIT$ dříve než T_3. Pro opravu musí být $COMMIT$ v T_3 pozdržen a proveden až po $COMMIT$ v T_3.
3. Ne, nebylo. Obecně to nejde, protože rozvrh, který odpovídá strict 2PL protokolu, je jak konfliktově uspořádatelný, tak zotavitelný. Doplnění vytvoří nekorektní rozvrh. Zde m.j. $L_X(A)$ v T_3 tuto transakci zablokuje do $COMMIT$ v T_1.

6 Datový management (specializace DW)

Do netříděného sekvenčního souboru s délkou bloku 4096 B je uloženo 8192 záznamů, každý s pevnou délkou 500 B.

1. Kolik pater bude mít hierarchický index nad sloupcem reprezentujícím klíč jednotlivých záznamů, pokud se do jednoho bloku indexu vejde 64 položek? Jedná se o přímý nebo nepřímý index? Svoje odpovědi zdůvodněte.
2. Jak se odpověď změní (pokud vůbec) v případě, že bude primární soubor uložen jako tříděný sekvenční soubor seřazený podle hodnot klíče? Své rozhodnutí vysvětlete.
3. Popište vkládání hodnot do neredundandního B-stromu s řádem m. Znázorněte, jak bude vypadat neredundandní B-strom (m=3) po vložení prvků 9, 3, 7, 6, 5 v tomto pořadí.

Nástin řešení

1. Bude potřeba adresovat všech 8192 položek primárního souboru v první úrovni, což zabere 128 bloků. 128 položek druhé úrovně zabere 2 bloky. 2 položky ve třetí úrovni se vejdou do jednoho kořenového bloku. Jedná se o přímý (odkazy adresují přímo primární soubor) index.
2. Ano, změní. Bude potřeba adresovat jen 1024 bloků v indexu první úrovně, což zabere 16 bloků. 16 položek se vejde do jednoho bloku druhé úrovně. Stále se jedná o přímý (odkazy adresují přímo primární soubor) index.
3. V případě vkládání 3. prvku do plného uzlu se dvěma prvky se v uzlu ponechají krajní hodnoty a prostřední se vloží o patro výše. Strom tedy bude vypadat:

7 Web app: vyhledávání článků (specializace DW)

Mějme webovou aplikaci pro vyhledávání článků. Uživatel má k dispozici textový vstup (input element s ID `searchInput`) a vyhledávací tlačítko (button element s ID `searchButton`). Při stisku tlačítka se uživateli zobrazí všechny články, které obsahují text ze vstupního pole jako podřetězec.

Aplikace používá REST API, z něhož potřebujete dva endpointy:

`/api/articles?search=<text-fragment>` vrací články, které obsahují daný řetězec, jako JSON seznam:

```
[ { "id": <id článku>, "content": <html fragment s obsahem>, "author": <id autora>, ... }, ... ]
```

`/api/user/<id>` vrací údaje o jednom uživateli jako JSON kolekci:

```
{ "id": <id uživatele>, "name": <celé jméno>, ... }
```

Pro jednoduchost již máte připravené funkce pro úpravu DOM: `clearArticles()` odstraní aktuálně zobrazované články a `addArticle(content, author)` přidá na stránku nový článek od daného autora.

1. Napište fragment JavaScriptu, který zajistí popisovanou funkcionalitu. Tedy při stisku tlačítka se provedou HTTP požadavky na příslušné endpointy z REST API a odpovídajícím způsobem se upraví DOM model (zobrazí se stažené články). Ve vašem řešení máte především demonstrovat práci s asynchroními požadavky (pomocí promises nebo `async`/`await`). Rovněž je nutné zajistit, aby se opakované stisky tlačítka během načítání článků ignorovaly.
2. Naznačte, jak by mohla vypadat implementace funkce `addArticle`. Není třeba psát celý kód, důležité je především nastínit, jak se do stránky vloží HTML fragment z položky `content` zaslané z REST API.

Ve vašem kódu nemusíte řešit okrajové situace (selhání sítě, chyby v datech, apod.). Pokud si nejste jisti přesnými názvy standardních funkcí a metod JS nebo DOM API, doplňte do komentáře, co očekáváte, že daná funkce dělá. Drobné chyby v syntax budou tolerovány.

Nástin řešení

1. Příklad řešení s použitím async/await. Hlavními očekávanými body řešení jsou správné použití async/await (nebo promises) a pak lock guard, naparsování JSONu a správné navázání event handleru.

```
var lock = false; // guard, ktery zajisti, aby tlacitko nefungovalo, kdyz probiha async operace
document.getElementById('searchButton').addEventListener('click', async () => {
  const query = document.getElementById('searchInput').value();
  if (query && !lock) {
    lock = true;
    clearArticles();
    const articlesResponse = await fetch(`/api/articles?search=${encodeURIComponent(query)}`);
    const articles = await articlesResponse.json(); // presne parsovani JSON neni dulezite
    for (const article of articles) {
      const authorResponse = await fetch(`/api/user/${encodeURIComponent(article.author)}`);
      const author = await authorResponse.json();
      addArticle(article.content, author.name);
    }
    lock = false;
  }
});
```

2. Stačí vyrobit element `<article>` a do property `innerHTML` mu přiřadit řetězec `content`. Alternativa je použít `DOMParser`, ale to je zbytečně složité. Přesný název není důležitý, ale je potřeba ukázat, že nastavit HTML fragment jako text nestačí (pak se to skutečně vloží jako text element, bez ohledu na to, že ve stringu jsou tagy).

8 Bezpečnostní tokeny (specializace DW)

Webová aplikace používá bezpečnostní tokeny JWT pro interní udržování autentizace (tedy autentizaci HTTP dotazů po té, co se uživatel přihlásil loginem a heslem). Token má hlavičku

```
{ "alg": "HS256", "typ": "JWT" }
```

kde `HS256` je identifikátor metody *Hash-Based Message Authentication Code* (HMAC) s použitím SHA256 jako hašovací funkce (pro připomenutí: HS256 je výchozí metoda, kterou musí podporovat všechny implementace JWT knihoven, a se kterou se potkáme nejčastěji).

1. Stručně popište (nejlépe algoritmicky pomocí funkcí) jak je zajištěna bezpečnost tohoto typu tokenu — tedy jak probíhá **vydání** a **ověření** tokenu bezpečnostní entitou (serverem).
2. Navrhněte, jaké položky musí mít tělo (payload) JWT tokenu, aby plnil autentizační funkci a byly zajištěny základní zásady bezpečnosti. Váš návrh by neměl by obsahovat zbytečné položky (tokeny mají být malé). Jednotivé položky stručně (jednou větou) vysvětlete/zdůvodněte.
3. Popište dva nejčastější mechanismy pro perzistentní uchování tokenu na straně klienta (tj. v prohlížeči) a stručně porovnejte jejich výhody nevýhody.

Nástin řešení

1. Viz `https://jwt.io/introduction`. Token se skládá ze 3 částí – Header, Payload, Signature – zakódovaných pomocí Base64. Server má tajný klíč (Secret), kterým tokeny podepisuje a ověřuje. Přičemž Signature = SHA256(Header + Payload + Secret). Tj. při vydávání tokenu se správně spočítá Signature, která je díky Secret neduplikovatelná, při ověření se úplně stejným způsobem spočítá znovu a porovná se Signature uloženou v tokenu.
2. Minimální payload obsahuje:
   - Nějaký identifikátor uživatele (ideálně databázové ID, neměl by to být osobní identifikátor jako RČ nebo email, ani login, už vůbec by to nemělo být něco, co je možné měnit).
   - Čas expirace, jinak by token platil navždy, což není žádoucí. Alternativně je možné místo exp uvádět iat (čas vydání), nebo dokonce oboje (v některých scénářích se to může hodit).

   Rozumné další položky jsou třeba role (pokud jich uživatel může mít více) nebo omezení autorizačního scope (např. read-only tokeny).
3. Existují právě dva rozumné a všeobecně podporované mechanismy – cookies a local storage (alternativně session storage). Hlavní rozdíl je, že cookies se spravují automaticky prohlížečem (může je rovnou nastavit server přes HTTP hlavičky) a automaticky se posílají s každým požadavkem (nemusíme se o ně starat). Cookies jsou také jediná cesta, pokud potřebujeme podporu SSR pro SPA aplikace. Token v local storage se musí přidávat do hlaviček každého async. requestu (HTTP hlavička `Authorization: Bearer <token>`), ale zase nad tím má programátor plnou kontrolu (hodí se např. pokud aplikace spravuje více tokenů pro přepínání rolí).

9 Kombinatorika (specializace OI-G-PADS, OI-G-PDM, OI-G-O, OI-O-PADS, OI-O-PDM, OI-PADS-PDM)

Nechť G je graf s n vrcholy, který neobsahuje žádný trojúhelník.

1. Kolik může mít G nejvýše hran?
2. Ukažte, že G obsahuje nezávislou množinu velikosti alespoň $\Omega(\sqrt{n})$.

Nástin řešení

1. $n^2/4$ (Turánova věta, maximum nastává pro $G = K_{n/2,n/2}$).
2. Okolí vrcholu maximálního stupně je nezávislá množina, tvrzení proto zřejmě platí, má-li G maximální stupeň více než $\sqrt{n} - 1$. Jestliže G má maximální stupeň nejvýše $\sqrt{n} - 1$, pak můžeme hladovým algoritmem (opakovaně vybereme libovolný vrchol v, přidáme ho do výsledku, a smažeme ho i s okolím) nalézt nezávislou množinu velikosti $\sqrt{n}$.

   Případně lze tento výsledek odvodit pomocí odhadu na Ramseyovo číslo $R(3, k)$ ze standardního důkazu Ramseyovy věty pro dvě barvy.

10 Optimalizace (specializace OI-O-PADS, OI-O-PDM, OI-G-O)

1. Uveďte přesné znění slabé a silné větě o dualitě lineárního programování.
2. Nelekněte se dlouhého zadání!

   Pro jednoduchost zápisu uv i vu označuje stejnou hranu $\{u, v\}$. Pro neorientovaný graf $G = (V, E)$ s nezáporným ohodnocením hran $d : E \to R^+$ a pro vybraný vrchol $r \in V$ uvažme následující lineární program (s proměnnými W_S

pro každou podmnožinu vrcholů S a dále s proměnnou α)

$$
\begin{aligned}
(P) \qquad \max & \sum_{S:S\subseteq V\setminus\{r\}} 2\cdot w_S & \\
& \sum_{S\subseteq V:|S\cap\{u,v\}|=1} w_S \le d_{uv} & \forall uv\in E \\
& \sum_{S\subseteq V:v\in S} w_S = \alpha & \forall v\in V \\
& w_S \ge 0 & \forall S\subseteq V
\end{aligned}
$$

a jeho duál (s proměnnými x_{uv} pro každou hranu $\{u,v\}\in E$, a y_u pro každý vrchol $u\in V$)

$$
\begin{aligned}
(D) \qquad \min & \sum_{uv\in E} d_{uv}\cdot x_{uv} & \\
& \sum_{e\in\delta(S)} x_e + \sum_{u\in S} y_u \ge 2 & \forall S\subseteq V \text{ t.ž. } r\notin S \\
& \sum_{e\in\delta(S)} x_e + \sum_{u\in S} y_u \ge 0 & \forall S\subseteq V \text{ t.ž. } r\in S \\
& \sum_{u\in V} y_u = 0 & \\
& x_{uv} \ge 0 & \forall uv\in E
\end{aligned}
$$

kde $\delta(S)=\{uv\in E\ :\ |S\cap\{u,v\}|=1\}$.

Vezměme nyní za $G=(V,E)$ následující graf: $V=\{1,2,3,4,5,6,7,8\}$, $E=\{12,23,34,45,56,67,78,81\}$ s ohodnocením hran $d_{12}=1$, $d_{23}=2$, $d_{34}=3$, $d_{45}=4$, $d_{56}=5$, $d_{67}=6$, $d_{78}=7$, $d_{81}=8$. Nechť $F=\{12,23,34,45,56,67,78\}$ a $T=(V,F)$.

Dále nechť vrchol $r=1$ a uvažme vektory w,x,y definované níže uvedenými předpisy:

Vektor w:

S	1	2	3	4	5	6	7	8	1,2	1,2,3	1,2,3,4	1,2,3,4,5	1,2,. . . , 6	1,2,. . . ,7	1,2,. . . ,8
w_S	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{3}{2}$	2	$\frac{5}{2}$	3	$\frac{7}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$

Pro každou podmnožinu S neuvedenou v tabulce máme $w_S=0$.

Vektor x:

e	12	23	34	45	56	67	78	81
x_e	1	1	1	1	1	1	1	0

Vektor y:

u	1	2	3	4	5	6	7	8
y_u	-1	0	0	0	0	0	0	1

Úkol: S využitím věty o dualitě a vektorů w,y dokažte, že T je minimální kostra grafu G. Nezapomeňte ověřit vše potřebné. Můžete využít (tj. není třeba dokazovat) následující lemma.

Lemma. Ke každé kostře $T=(V,F)$ grafu $G=(V,E)$ existuje celočíselný vektor y_T tak, že (χ_T,y_T) je přípustné řešení lineárního programu (D), kde χ_T označuje charakteristický vektor kostry T (tj. $\chi_T(e)=1$ pro $e\in F$, jinak $\chi_T(e)=0$).

11 Geometrie (specializace OI-G-PADS, OI-G-PDM, OI-G-O)

Uvažujme množinu $P=\{(1,0),(0,1),(-1,0),(0,-1)\}$. (Tedy P je množina čtyř bodů v rovině, které jsou vrcholy pootočeného čtverce.)

1. Explicitně popište duální množinu P^*.

2. Explicitně popište druhou duální množinu P^{**}.

Nástin řešení Je-li X množina, tak pak duální množina X^* je definovaná jako $\{y : \forall x \in X < x, y > \leq 1\}$. Výrazem $< x, y >$ myslím skalární součin.

1. Lze řešit přímo z definice. Například bod $(1, 0)$ dává omezení $y_1 \leq 1$, kde $y = (y_1, y_2)$, tedy poloprostor jehož hraniční přímka je rovnoběžná s osou y a prochází bodem $(1, 0)$. Podobně další body dávají analogická omezení a průnik takto vzniklých poloprostorů je čtverec s vrcholy $(1, 1)$, $(1, -1)$, $(-1, 1)$ a $(-1, -1)$.
2. Lze řešit i bez vyřešení předchozího bodu, pokud si vzpomeneme na lemma, které říká, že $X^{**} = conv(X \cup \{0\})$. V našem případě počátek patří do konvexního obalu P, takže $P^{**} = conv(P)$.

   I bez znalosti tohoto lemmatu lze řešit elementárně s pomocí výsledku první úlohy. Tentokrát se získají omezující poloprostory z vrcholů čtverce $(1, 1)$, $(1, -1)$, $(-1, 1)$ a $(-1, -1)$. Pak je potřeba si rozmyslet, že žádný další bod (v konvexním obalu) už nepřidává nové omezení.

12 Pokročilá diskrétní matematika (specializace OI-G-PDM, OI-O-PDM, OI-PADS-PDM)

Graf G je nakreslený na ploše tak, že má 4 stěny délky tři a 2 stěny délky čtyři. Všechny vrcholy grafu G mají stupeň 4. Nakreslení grafu G je buňkové.

1. Kolik má graf G hran a vrcholů?
2. Na které ploše je G nakreslen?
3. Nakreslete graf tak, aby byly splněny výše popsané podmínky.

Nástin řešení

1. Součet délek stěn je dvojnásobkem počtu hran, G má tedy $\frac{1}{2}(4 \times 3 + 2 \times 4) = 10$ hran. Součet stupňů je také roven dvojnásobku počtu hran, G má tedy $2 \cdot 10/4 = 5$ vrcholů. Lze nahlédnout, že se jedná o graf K_5.
2. Ze zobecněné Eulerovy formule plyne, že plocha, na níž je G nakreslen, má Eulerovský rod $e+2-n-f = 10+2-5-6 = 1$, je to tedy projektivní rovina.
3. $G = K_5$ lze nakreslit s předepsanými délkami stěn např. tak, že K_5 − párování nakreslíme do roviny s jednou stěnou délky čtyři, do této stěny dáme křižítko, a přes něj vedeme obě hrany párování.

13 Fourierova transformace (specializace OI-G-PADS, OI-O-PADS, OI-PADS-PDM)

1. Definujte diskrétní Fourierovu transformaci (DFT) pro vektory komplexních čísel.
2. Ukažte, jak se DFT dá použít k násobení polynomů.
3. Uveďte, jakou časovou složitost má algoritmus FFT pro efektivní výpočet DFT. Jaká z toho plyne složitost násobení polynomů? Samotnou FFT nemusíte popisovat.

Nástin řešení

1. Pro n zvolenou dimenzi prostoru a ω primitivní n-tou odmocninu z 1 je Fourierův obraz vektoru $\mathbf{x} \in \mathbb{C}^n$ vektor $\mathcal{F}(\mathbf{x}) \in \mathbb{C}^n$, pro který platí:
$$\mathcal{F}(\mathbf{x})_j = \sum_{k=0}^{n} \mathbf{x}_k \omega^{jk}.$$
2. Především si všimneme, že DFT se dá použít na převod mezi reprezentací polynomu vektorem koeficientů a vektorem hodnot: Je-li p polynom stupně menšího než n s vektorem koeficientů $\mathbf{p}$, pak $\mathcal{F}(\mathbf{p})$ je vektor funkčních hodnot

$(p(\omega^0), \ldots, p(\omega^{n-1}))$. Navíc je polynom těmito funkčními hodnotami určen jednoznačně, takže DFT je bijekce mezi vektory koeficientů a vektory funkčních hodnot. Nyní si stačí uvědomit, že v reprezentaci funkčními hodnotami stačí vektory násobit po složkách. Takže násobené polynomy převedeme do druhé reprezentace, tam je vynásobíme, a pak zase převedeme zpátky.

3. FFT pracuje v čase $\Theta(n \log n)$. Algoritmus na násobení polynomů se skládá ze dvou dopředných DFT, jedné zpětné, a násobení po složkách v lineárním čase, takže dohromady běží také v čase $\Theta(n \log n)$.

14 Pokročilá matematická analýza (specializace OI-G-PADS, OI-G-PDM, OI-G-O, OI-O-PADS, OI-O-PDM, OI-PADS-PDM)

1. Definujte metrický prostor (M, d).
2. Kdy je množina $X \subset M$ otevřená?
3. Nechť (M, d) je jednotkový reálný interval $[0, 1]$ s obvyklou metrikou $d(x, y) = |x - y|$. Je jeho podmnožina $X = [0, \frac{1}{2})$ otevřená? Odpověď zdůvodněte.

Nástin řešení

1. M je (neprázdná) množina, metrika $d : M \times M \to \mathbb{R}$ splňuje, že (i) vždy $d(x, y) \geq 0$ a $d(x, y) = 0 \iff x = y$, (ii) vždy $d(x, y) = d(y, x)$ a (iii) vždy $d(x, z) \leq d(x, y) + d(y, z)$.
2. $X \subset M$ je otevřená $\iff$
$$\forall\, x \in X\, \exists\, r > 0 \left(y \in M \,\&\, d(x,\, y) < r \Rightarrow y \in X\right).$$
3. Ano, je, splňuje podmínku uvedenou v předchozí části.

15 Poloprůhlednost (specializace PGVVH-PG, PGVVH-VPH)

Budeme se zabývat částečnou průhledností (poloprůhledností). Pro jednoduchost se omezíme na rastrové 2D obrázky.

1. Jakým způsobem se obvykle reprezentuje poloprůhlednost? Popište technicky, jaký údaj se musí do obrázku přidat a pokuste se definovat jeho sémantiku. Podporují tento systém současné grafické karty?
2. Vymyslete alespoň dva příklady aplikace poloprůhlednosti v rastrové 2D grafice. Navrhněte vzorečky, které by se při implementaci použily a v případě potřeby ještě upřesněte formát dat pixelů.
3. Napadá vás nějaké omezení, se kterým se při použití poloprůhledné grafiky potýkáme? Můžete se zamyslet i v oboru 3D grafiky (renderingu, GPU renderingu). Popište podrobně problém a jeho případné řešení, i když by třeba nebylo dokonalé.

Nástin řešení Alfa kanál (α) znamená neprůhlednost, je to jedno číslo navíc do každého pixelu. Rozsah 0.0 až 1.0 pro HDR, jinak obvykle 0 až 255.

Často se používá tzv. přednásobený formát, kde se neprůhledností α pronásobí všechny barevné kanály pixelu. Je to výhodné, protože ve většině operací by se takové násobení stejně muselo provádět.

GPU tento formát plně podporují, dokonce umí při zápisu do frame-bufferu pracovat se všemi běžnými binárními operacemi a využívat přednásobený formát.

Aplikace 1: operace "C = A OVER B". Je to skládání vrstev za sebe (viz PhotoShop, GiMP, kde jsou data obrázku uložena ve vrstvách). Pro přednásobený formát se použije vzorec $X_C = X_A + (1 - \alpha_A) * X_B$ (X reprezentuje jakákoli barevný kanál, i neprůhlednost α).

Aplikace 2: operace "C = A ATOP B". Napodobuje lepení poloprůhledné fólie A na povrch předmětu B. Pro přednásobený formát se použije vzorec $X_C = \alpha_B * X_A + (1 - \alpha_A) * X_B$.

Omezení 1: ve 3D grafice se musí při poloprůhledném vykreslování postupovat odzadu-dopředu, což znamená 3D třídění scény a je zcela proti koncepci Z-bufferu (normálně se data třídit nemusí).

Omezení 2: hodnota α reprezentuje jenom souhrnnou informaci o pokrytí plochy pixelu danou barvou, jakékoli geometrické informace (třeba z rasterizace) jsou nenávratně ztraceny. Kvůli tomu se v některých případech při anti-aliasingu objevují nechtěné artefakty a interference. Například při dvojím nakreslení identického objektu s různými barvami přes sebe se nedosáhne dokonalého zakrytí, protože na obrysu prosvítá spodní barva...

16 Homogenní souřadnice a maticové transformace (specializace PGVVH-PG, PGVVH-VPH)

1. Jak se pomocí matic transformují objekty v 3D počítačové grafice?
2. Proč zavádíme homogenní souřadnice a matice 4×4? Co nám umožňují realizovat?
3. Uveďte několik elementárních maticových transformací (translace, rotace, škálování – pokuste se matice napsat prvek po prvku) a jeden praktický příklad složené transformace (nemusíte konstrukci dotáhnout do konce, stačí naznačit postup).

17 Rotace v prostoru (specializace PGVVH-PG, PGVVH-VPH)

Jde nám o metody, jak matematicky reprezentovat rotaci pevného tělesa v 3D prostoru. Středem rotace bude pro jednoduchost počátek souřadnic. Těleso se kolem počátku bude jenom otáčet, to znamená, že nebude měnit svůj tvar ani velikost („transformace tuhého tělesa“ nebo anglicky „rigid body transform“).

1. Navrhněte první metodu, kterou byste zvolili pro případ, že bude třeba často rotaci interpolovat – tj. animovat dané těleso (později se budeme snažit vyjádřit plynulý a přirozený animační pohyb).
2. Navrhněte ještě jinou metodu, jak rotaci vyjádřit. Může mít proti té první některé výhody (rychlost, jednoduchost, snadnost GPU implementace), ale i nevýhody. Výhody a nevýhody uveďte a alespoň stručně zdůvodněte.
3. Jak byste postupovali při animaci rotace v čase? Vyberte si první nebo druhou metodu a dostatečně přesně popište matematický postup animace (i u této podotázky se zabývejte jen rotacemi).

Nástin řešení

1. Jednotkový kvaternion (uvést aspoň základní vzorečky, imaginární jednotky, konjugovaný kvaternion, pravidla pro sčítání a násobení (nemusí být), inverze, mocnina – bude potřeba pro SLERP). Otočení bodu x dvojnásobným úhlem kvaternionu q se spočte jako qxq^* (tj. dvě kvaternionová násobení, bod se převede do kvaternionu tak, že se vezme jeho homogenní souřadnice). Reprezentace: čtveřice reálných čísel. Nevýhoda: není praktické přímo kvaternionem transformovat množství 3D bodů (proto se převádí na 3x3 nebo 4x4 matici a ta se pak používá k masovým transformacím (např. na GPU)).
2. Eulerovy úhly - postupné otáčení kolem třech souřadných os. Existuje mnoho konvencí (pořadí jak se berou osy, zda se souřadný systém točí spolu s tělesem). Reprezentace: tři reálná čísla. Výhody: kompaktní, jednoduchá implementace. Nevýhody: „gimbal lock“, nemožnost rozumně interpolovat orientaci, singularity (podobně jako ve sférickém souřadném systému).
3. SLERP jednotkových kvaternionů: $SLERP(q, r, t) = q(q^*r)^t$ kde $0 <= t <= 1$ je reálné číslo (čas) a q resp. r jsou počáteční resp. koncový kvaternion. Obecná mocnina se implementuje goniometricky (lze uvést vzorec). Ještě lepší (spojitější) interpolování je možné dosáhnout pomocí klíčových snímků (klíčové kvaterniony) a pak na ně aplikovat spline křivku pomocí De Casteljau algoritmu (třeba nakreslit schémátko), který používá jen SLERP. Tak získáme hladký kvaternion pro libovolný bod na časové ose a bude splněna požadovaná spojitost C^N animace.

18 Distribuované sledování paprsku (specializace PGVVH-PG)

Tyto techniky se také nazývají „Monte-Carlo ray tracing“ nebo „Monte-Carlo integrace“.

1. Vyjmenujte několik příkladů, kde je užitečné do rekurzivního sledování paprsku zapojit stochastický výpočet (Monte-Carlo integraci).
2. Vyberte si jedno konkrétní použití Monte-Carlo a popište ho detailně. Která veličina se integruje? Který nedostatek původního přístupu se odstraňuje nebo potlačuje?
3. Navrhněte vzorkování (sampling), které by se dalo dobře použít v předchozím příkladu. Stačí popsat metodu vzorkování rámcově nebo nakreslit obrázek. Šlo by toto vzorkování implementovat adaptivně?

Nástin řešení

1. Měkké stíny, anti-aliasing, hloubka ostrosti objektivu, rozmazání pohybem, měkké odrazy světla, disperze světla.
2. Měkké stíny - plošný zdroj světla, integruje se osvětlení přes plochu zdroje. V každém průsečíku se odhaduje, kolik procent světla dopadá ze zdroje. Prakticky se odhady provádí vzorkováním plochy zdroje, tj. Monte-Carlo integrací. Původní velmi ostré hranice světla a stínu se nahradí příjemnějšími a realističtějšími měkkými přechody, současně se zohlední vzdálenost světelného zdroje, překážky a příjemce stínu.
3. Světelný zdroj ve tvaru obdélníku by se dobře dal vzorkovat metodou jittering (rozdělení plochy na MxN stejně velkých dílků a náhodný výběr jednoho vzorku v každém dílku). Adaptivní jittering je možný, lépe se implementuje v omezených podmínkách (tj. ne neomezený), např. systémem „quadtree“ nebo zjemňováním faktorem 2 (rozdělením již existujícího dílku na poloviny).

19 3D scéna pro kreslení na GPU (specializace PGVVH-VPH)

1. Jak byste v paměti počítače reprezentovali 3D scénu, která se pak bude v reálném čase vykreslovat na grafické kartě (GPU)? Nemusíte psát přímo deklarace dat, ale popište dostatečně podrobně svůj návrh slovy.
2. Uvažujte systém hierarchické reprezentace tak, aby se komponenty (objekty, modely) daly jednoduše ve scéně opakovat bez toho, aby se musela data duplikovat.
3. Jak technicky implementovat animaci (pohyb) jednotlivých objektů ve scéně? Omezte se na nedeformovatelná tělesa („rigid body animation“), která se jen ve scéně přemisťují otáčením a translací.

Nástin řešení Jedna možná varianta řešení, blízká HW (GPU). Scéna se skládá celá z trojúhelníků, ty jsou sdružovány do skupin (objektů), každý objekt může být ve scéně odkazován víckrát (instancing pomocí maticových transformací), to celé se pak dá zobecnit do 3D hierarchie:

1. data rozdělit na geometrii a topologii. Geometrie = pole vrcholů (vertex-buffer) s relevantními přidanými daty (3D souřadnice v object space, normálový vektor, texturové souřadnice pro N textur, příp. i barva...). Topologie = index-buffer (pole trojic celých čísel odkazujících se na vrcholy trojúhelnků). Jednotlivé objekty mohou mít své vlastní buffery nebo se mohou definovat pomocí úseků ve velkém sdíleném bufferu.

2. hierarchie - systém odkazů na jednotlivé objekty, který má podobu stromu (nebo DAG). Na každý vnitřní uzel se dá odkazovat vícekrát, definice celé scény bude spočívat v průchodu takovým grafem od kořene. Hrana grafu má asociovanou homogenní transformační matici 4x4, tím se dají vyvolat různé instance stejného tvaru (objektu).

3. požadovaná animace částí scény lze realizovat pouze změnou transformačních matic (a v každém snímku se musí opakovat průchod grafem scény). Pro pohodlnější a hladší reprezentace pohybu je vhodné rozdělit transformaci na translační část (vektor posunutí) a rotační část (např. kvaternion). Pro každý konkrétní čas snímku se vyčíslí posunutí pomocí interpolace translace (A) a interpolace orientace (B).

(A) nejlépe použít nějaký hladký interpolační systém, třeba Catmull-Rom nebo TCB spline.

(B) po částech lineární interpolace by používala SLERP interpolaci kvaternionů, při potřebě hladší komplikovanější animace by se pomoci mnoha SLERP operací implementovaly některé interpolační spliny, např. spojitě napojené Bezier křivky (viz algoritmus de Casteljau - detaily jsou příliš rozsáhlé, není je potřeba uvádět).

Jiná možná řešení se mohou lišit v detailech, měl by však být zachován princip instancingu a hierarchie bez zbytečných duplicit dat. Dále - pohyb pouze pomocí transformačních matic, výpočty provádí GPU (vlastní data (template) se nemění, mění se jen transformace).

20 Implementace konstrukcí objektově orientovaných jazyků (specializace PVS)

1. Vysvětlete rozdíl mezi běžnými a virtuálními metodami v C++. Může se rychlost volání běžných a virtuálních metod lišit? Proč?

Nástin řešení Při volání běžné metody se implementace volí podle statického typu cíle volání. Při volání virtuální metody se implementace volí podle skutečného typu cíle volání. Rychlost volání se zpravidla mírně liší, protože nalezení cílové adresy virtuální metody vyžaduje přístup do tabulky virtuálních metod. Hardware může cílovou adresu volání méně úspěšně předvídat.

2. Může nebo musí mít v C++ třída virtuální konstruktor nebo virtuální destruktor? Pokud ano, v jaké situaci se používá a proč / pokud ne, proč? Velmi stručně (stačí několik řádek) popište základní princip alespoň jednoho návrhového vzoru, který s pojmy virtuální konstruktor nebo virtuální destruktor úzce souvisí.

Nástin řešení Virtuální konstruktor není, protože konstruovaný objekt má staticky určený typ. Virtuální destruktor se musí používat při uvolňování objektů v polymorfním kontextu. Factory method slouží k vytvoření instance třídy, jejíž typ je dynamicky určen parametry volání. Lze uvést i Abstract Factory.

3. Uvažujte následující fragment kódu v C++ (předpokládejte potřebné hlavičkové soubory).

```
class A {
  public:
    virtual void f () { std::printf ("A::f "); }
};

class B : public A {
  public:
    virtual void f () { std::printf ("B::f "); }
};

int main (void) {
  A *u = new A ();
  A *v = new B ();
  B *w = new B ();

  u->f ();
  v->f ();
  w->f ();
  ((A*) w)->f ();
  ((A)(*w)).f ();
  static_cast<A*>(w)->f ();
  static_cast<A>(*w).f ();
  dynamic_cast<A*>(w)->f ();
  dynamic_cast<A>(*w).f ();
}
```

Jaké funkce se zavolají, případně jakou chybu (a proč) jednotlivé výrazy vyvolají? V případě, že by některé z výrazů způsobily kompilační chybu, pro vyhodnocení chování ostatních výrazů je neuvažujte (jako by byly zakomentované).

Nástin řešení `dynamic_cast<A>(*w).f` vyvolá kompilační chybu (dynamic cast lze aplikovat pouze na ukazatel nebo referenci). `u->f`, `((A)(*w)).f` a `static_cast<A>(*w).f` zavolají `A::f`. Ostatní volání zavolají `B::f`.

21 Stránkování (specializace PVS)

Uvažujte architekturu procesoru s podporou stránkování a délkou virtuální a fyzické adresy 32 bitů. Předpokládejte, že k překladu adres procesor používá jednoúrovňovou stránkovací tabulku. Velikost stránek je 4 KiB.

1. Kolik položek bude mít stránkovací tabulka a kolik paměti bude tato tabulka zabírat pro každý spuštěný proces? Položka stránkovací tabulky musí obsahovat všechny nutné informace a musí být pro procesor efektivně přístupná (maximálně 1 čtení z paměti).
2. Přístup na virtuální adresu `0x00020748` byl přeložen na fyzickou adresu `0x882F9748`. Dále předpokládejte, že stránkovací tabulka procesu je uložena v souvislém bloku fyzické paměti od adresy `0x00800000`. Na jaké fyzické adrese leží položka stránkovací tabulky použitá při překladu a jaký je její obsah? Uveďte hodnoty a význam všech částí záznamu.
3. Předpokládejte, že při přístupu procesu na virtuální adresu `0x00020748` nedošlo k výpadku stránky. Bezprostředně poté proces přečetl 8-bajtový blok paměti z adresy `0x00020FF8`. Můžete usoudit, zda dojde nebo může dojít k výpadku stránky u druhého přístupu? Odpověď zdůvodněte.

V odpovědích podle potřeby používejte hexadecimální (případně binární) zápis čísel.

Nástin řešení

1. Řešení by mělo odvodit potřebnou velikost položky (4B => 20 bitů na číslo rámce + 6 bitů P A D X RW UC + 6 bitů zbývá), počet bitů na číslo stránky udává počet položek, vynásobením dostaneme počet bajtů nutných pro uložení tabulky.
2. Na základě informace o velikosti stránky by mělo řešení určit číslo stránky a rámce obou adres. Číslo stránky udává číslo položky stránkovací tabulky. Vynásobeno velikostí položky dává offset od začátku stránkovací tabulky ve fyzické paměti, součet obou pak dává fyzickou adresu, ze které musel procesor položku přečíst. Položka musí obsahovat číslo rámce a minimálně flag P (present).
3. Oba přístupy jsou do stejné stránky. Pokud proces nebyl přerušen/přeplánován, lze očekávat, že výpadek nenastane. V případě přeplánování mohlo být mapování odstraněno a k výpadku dojít může.

22 Jazyk SQL (specializace PVS)

Uvažujte následující databázové schéma:

- Student (sid: integer, jméno: string, ročník: string) s klíčem sid,
- Předmět (pid: string, název: string, konání: string, místnost: string, uid: integer) s klíčem pid,
- Zápis (sid: integer, pid: string) s klíčem sid a pid.

Stejně pojmenované atributy reprezentují cizí klíče.

1. Napište příkazy pro vytvoření tabulek v jazyce SQL, včetně naznačených integritních omezení.
2. Uložte do každé tabulky alespoň jeden korektní záznam.
3. Vypište abecedně seřazený seznam jmen všech studentů, kteří si nezapsali žádnou přednášku od vyučujícího s identifikátorem (uid) 33458.
4. Zrušte všechny tabulky.

Nástin řešení

1. ```
   CREATE TABLE student (sid INT PRIMARY KEY, jmeno VARCHAR(255), rocnik INT);
   CREATE TABLE predmet (pid INT PRIMARY KEY, ...);
   CREATE TABLE zapis (sid integer, pid INT, PRIMARY KEY (sid, pid) );
   ALTER TABLE zapis ADD FOREIGN KEY (sid) REFERENCES student(sid);
   ALTER TABLE zapis ADD FOREIGN KEY (pid) REFERENCES predmet(pid);
   ```

2. ```
   INSERT INTO student VALUES (123, 'Karel', 1);
   ...
   ```

3. ```
   SELECT s.jmeno, s.sid FROM student s WHERE s.sid NOT IN
      ( SELECT z.sid FROM zapis z, predmet p WHERE z.pid = p.pid AND p.uid = 33458 )
      ORDER BY jmeno;
   ```
   ```

4.
```
DROP TABLE zapis;
DROP TABLE student;
DROP TABLE predmet;
```

23 MapReduce (specializace PVS)

Uvažujte CSV soubory obsahující informace o motocyklech. Každý řádek obsahuje značku, typ, nejvyšší rychlost, cenu a hmotnost.

1. Pomocí programovacího modelu MapReduce vypočítejte pro každou značku motocyklu maximální a průměrnou cenu. Specifikujte příslušné funkce pomocí libovolného (pseudo)kódu.
2. Můžeme použít funkci Combine? Proč?

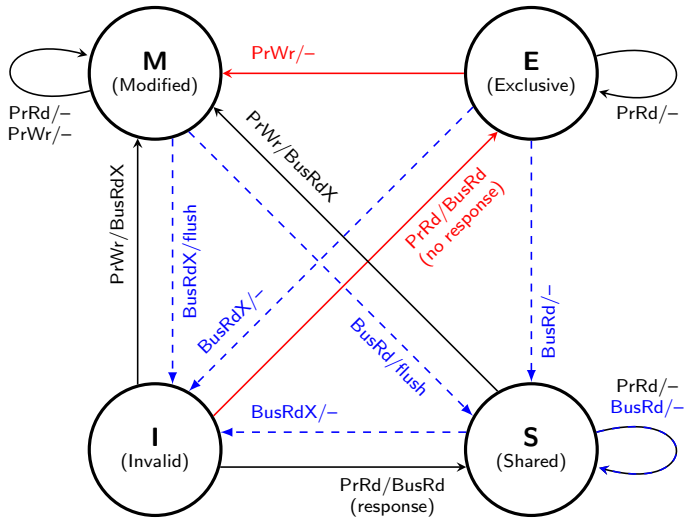
Nástin řešení

1. Funkce Map může mít na vstupu např. obsah jednoho souboru. Zpracuje jednotlivé řádky tak, že pro každý řádek bude vracet dvojici (značka, cena). Funkce Reduce bude mít na vstupu dvojice (značka, kolekce všech příslušných cen), z nich spočítá a vrátí maximum a průměr.
2. Funkci Combine je možné použít pro výpočet maxima, ale ne pro výpočet průměru.

24 Koherence cache (specializace SP)

Předpokládejte víceprocesorový systém s několika fyzickými procesory, z nichž každý má svou lokální cache. Všechny cache jsou *write-back* a systém implementuje *koherenční protokol MESI* založený na snoopingu.

1. Vysvětlete, jaký problém řeší a jaké záruky poskytuje výše uvedený mechanizmus pro zajištění koherence.
2. Pro jednotlivé stavy a přechody v následujícím stavovém diagramu vysvětlete co daný stav vypovídá o dané cache line a za jakých okolností a proč se vykoná daný přechod.



3. Předpokládejte, že program běžící na výše uvedeném systému způsobil níže uvedenou posloupnost přístupů různých procesorů do *stejné* cacheline. Pro každou operaci uveďte, jak budou jednotlivé procesory reagovat (a komunikovat na sdíleném médiu) a jak se bude měnit stav cacheline na jednotlivých procesorech. Předpokládejte rovněž, že na počátku jsou všechny cacheline neplatné.

   (1) **P2** read
   (2) **P2** write
   (3) **P1** read

(4) **P0** read

(5) **P1** write

Nástin řešení

1. Konzistence (potenciálně více) lokálních stavů (cache procesorů) a globálního stavu (hlavní paměť) programu. Koherenční protokol zajišťuje, že každý procesor "vidí" vlastní zápisy do paměti v pořadí určeném programem (program order), že zápisy do paměti budou nakonec (eventually) "viditelné" pro všechny procesory, a že zápisy do stejného místa v paměti budou uspořádány a všechny procesory je "uvidí" ve stejném pořadí.
2. Stavy se týkají jednotlivých cachelines, vedle poměrně zřejmých stavů Invalid a Modified (exclusive dirty) je důležité odlišit stavy Shared (shared clean) a Exclusive (exclusive clean). Posledně uvedený stav umožňuje tichý přechod do stavu Modified (bez ohlášení bus read exclusive).

   Ohodnocení přechodů reprezentuje podmínky, za kterých se přechod uskutečňuje, tj. lokální přístup k cacheline (plné čáry) či přístup jiného procesoru k cacheline (čárkované čáry) a odpovídající vnější aktivitu (např. flush) nebo transakci na sdíleném médiu (např. bus read nebo bus read exclusive/read for update), jejichž sledování (snooping) umožňuje ostatním procesorům udržovat správný stav lokální cache.

   Očekává se vysvětlení operací a skupin přechodů. Přechod do stavu Exlusive je možný pouze pokud žádný z ostatních procesorů (v časovém limitu) nezareaguje na transakci bus read.
3. Jedná se v podstatě o mechanickou interpretaci přechodového diagramu. Pro každý přístup se očekává řádek se stavem cacheline na každém procesoru a popis souvisejících aktivit procesorů.

25 Principy objektového návrhu (specializace SP)

Předpokládejte, že jste dostal(a) za úkol udržovat níže uvedený kód spolu s požadavkem rozšířit jej o podporu pro vykreslování trojúhelníků. Rovněž můžete předpokládat, že v blízké době bude nutné přidávat podporu pro další typy geometrických útvarů.

```
1 public record Point(double x, double y);
2
3 public enum ShapeType { SQUARE, CIRCLE }
4
5
6 public abstract class Shape {
7   ShapeType type;
8
9   public Shape(ShapeType type) {
10    this.type = type;
11  }
12 }
13
14
15 public class Square : Shape {
16   Point topLeft;
17   double side;
18
19   public Square(Point topLeft, double side)
20     : base(ShapeType.SQUARE) { ... }
21 }
22
```

```
23 public class Circle : Shape {
24   Point center;
25   double radius;
26
27   public Circle(Point center, double radius) :
28     base(ShapeType.CIRCLE) { ... }
29 }
30
31
32 public class Graphics {
33   public static void DrawShape(Shape shape) {
34     if (shape.type == ShapeType.SQUARE) {
35       DrawSquare(shape as Square);
36     } else if (shape.type == ShapeType.CIRCLE) {
37       DrawCircle(shape as Circle);
38     } else {
39       throw new ArgumentOutOfRangeException();
40     }
41   }
42
43   static void DrawSquare(Square square) { ... }
44   static void DrawCircle(Circle circle) { ... }
45 }
```

S ohledem na potřebu rozšiřování je zřejmé, že kód v daném kontextu porušuje tzv. *open-closed principle* (OCP), tedy jeden z hlavních principů objektového návrhu.

1. Popište, co je podstatou OCP, a vysvětlete, jak se jeho aplikace odráží v nějakém vámi zvoleném návrhovém vzoru (např. Strategy nebo Template Method).
2. Identifikujte místa, kde v uvedeném kódu dochází k porušování OCP, a vysvětlete proč.

3. Upravte objektový návrh v uvedeném kódu tak, aby v daném kontextu vyhovoval požadavkům OCP. Řešení nemusí zachovat stávající veřejné třídy.

Přestože je uvedený kód zapsán v jazyce C#, ekvivalentní kód v jazyce Java by byl až na volání konstruktorů předka a přetypování na potomky v podstatě identický. Tyto rozdíly nejsou pro zodpovězení otázky podstatné a odpověď na poslední část otázky je možné zapsat v libovolném z jazyků C# nebo Java.

Nástin řešení

1. Pro vybrané typy změn by měl být návrh *open for extension*, tedy podporovat rozšiřování o nové chování, které umožní změny realizovat, ale bez nutnosti modifikovat stávající kód, tedy *closed for modification*. K tomuto se používají abstrakce ve formě rozhraní či abstraktních tříd, se kterými mohou ostatní části systému manipulovat, ale které neomezují možná chování implemetací či odvozených tříd.

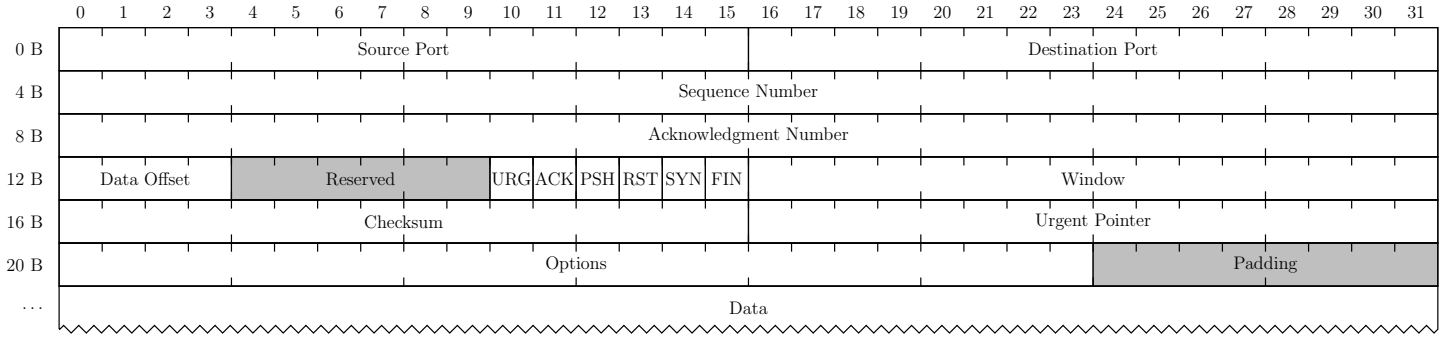
   V případě návrhového vzoru *Strategy* je důsledkem aplikace OCP zavedení třídy reprezentující abstraktní operaci, kterou může klientská třída použít bez znalosti konkrétní třídy, která ji implementuje.

2. Jde o místa v kódu, která by bylo nutné změnit při každém přidání nového potomka Shape. Specificky jde tedy o výčtový typ ShapeType (je nutné přidat novou výčtovou hodnotu/instanci), metodu Graphics.DrawShape() (přidání větve pro novou hodnotu ShapeType spolu s přetypováním a voláním statické metody) a třídu Graphics (přidání statické metody pro každého nového potomka).

3. Za dané situace je vhodné zbavit se výčtového typu ShapeType, ve třídě Shape zrušit atribut type (a s ním související konstruktor a jeho volání v odvozených třídách) a naopak přidat abstraktní metodu Draw(), kterou musí jednotlivé třídy implementovat. Statické metody pro kreslení specifických typů přesunout z třídy Graphics do příslušných potomků Shape jako overriding implementace abstraktní metody Draw() a třídu Graphics ideálně zrušit (včetně metody DrawShape()).

26 Řízení toku a předcházení zahlcení v TCP (specializace SP)

Následující diagram popisuje hlavičku TCP paketu podle RFC 793:



1. Na příkladech konkrétních situací vysvětlete účel mechanismů flow control (řízení toku) a congestion control (předcházení zahlcení) v TCP protokolu (tedy popište situace, ve kterých by absence těchto mechanismů způsobovala problémy, a vysvětlete, jaké problémy to jsou).

2. S použitím konkrétních polí z hlavičky TCP paketu popište podmínku, kterou mechanismus flow control (řízení toku) používá pro rozhodnutí, zda je možné odeslat paket.

3. Vysvětlete, která pole z hlavičky TCP paketu se podílejí na mechanismu congestion control (předcházení zahlcení) a naznačte jak (tedy na rozdíl od předchozího bodu není nutné popsat konkrétní pravidla, ale očekává se, že vysvětlíte, jak se z hodnot polí detekují relevantní situace a jak v nich TCP rámcově reaguje).

Nástin řešení

1. Flow control řeší situace, kdy by odesilatel poslal více dat, než je příjemce schopen přijmout, například kvůli nedostatku paměti na uložení přijatých dat. Absence flow control by znamenala, že příjemce musí přijatá data zahazovat. Congestion control řeší situace, kdy odesilatel poslal více dat, než je prostředí mezi odesilatelem a příjemcem schopno přenést, například kvůli překročení celkové přenosové kapacity. Absence congestion control by znamenala, že se data budou doručovat s rostoucím zpožděním a nakonec zahazovat.

2. Odesilatel může odeslat nanejvýš `Window` bajtů dat po datech potvrzených příjemcem, tedy musí platit, že `Acknowledgment Number + Window >= Sequence Number + Data Length`, přitom délka dat se nepřenáší explicitně, ale vypočítá z celkové délky paketu.
3. Odesilatel detekuje výpadek přenosu pozorováním opakovaného potvrzení se stejným `Acknowledgment Number`, reaguje posláním ztraceného segmentu a úpravou pravidel pro odesílání dalších segmentů (fast retransmit a fast recovery).

Přesné detaily se mohou lišit podle konkrétní implementace TCP protokolu, skica odpovědi představuje očekávaný základ.

27 Struktura UNIX-like systému souborů (specializace SP)

Předpokládejte jednoduchý UNIX-like systém souborů (například podobný ext2), který alokuje po jednotlivých blocích (bez podpory extentů), používá standardní alokační bitmapy a strukturu inodes a dentries (bez podpory indexování) podle následujících definic (zjednodušeno ze zdrojového kódu jádra Linuxu):

```
#define EXT2_NDIR_BLOCKS        12
#define EXT2_IND_BLOCK          EXT2_NDIR_BLOCKS
#define EXT2_DIND_BLOCK         (EXT2_IND_BLOCK + 1)
#define EXT2_TIND_BLOCK         (EXT2_DIND_BLOCK + 1)
#define EXT2_N_BLOCKS           (EXT2_TIND_BLOCK + 1)

#define EXT2_NAME_LEN 255

struct ext2_inode {
    __u16   i_mode;                 /* File mode */
    __u16   i_uid;                  /* Low 16 bits of Owner Uid */
    __u32   i_size;                 /* Size in bytes */
    __u32   i_atime;                /* Access time */
    __u32   i_ctime;                /* Creation time */
    __u32   i_mtime;                /* Modification time */
    __u16   i_gid;                  /* Low 16 bits of Group Id */
    __u16   i_links_count;          /* Links count */
    __u32   i_blocks;               /* Blocks count */
    __u32   i_flags;                /* File flags */
    __u32   l_i_reserved1;
    __u32   i_block[EXT2_N_BLOCKS]; /* Pointers to blocks */
};

struct ext2_dir_entry {
    __u32   inode;                  /* Inode number */
    __u16   rec_len;                /* Directory entry length */
    __u8    name_len;               /* Name length */
    __u8    file_type;
    char    name[EXT2_NAME_LEN];    /* File name */
};
```

1. Popište krok po kroku postup přečtení jednoho bajtu na pozici 123456 v souboru, pokud máte k dispozici jeho dentry. U každého kroku musí být uvedeny všechny informace (odkud se vezmou a případně co znamenají), ze kterých se vypočítá pozice pro čtení z disku a délka čteného úseku (aktualizaci atime neřešte). Pokud nějaké informace nejsou zadané, rozumně je doplňte podle svého uvážení.
2. Spočítejte kompletní prostor obsazený na disku souborem o délce 123456 bajtů, včetně všech metadat týkajících se souboru. Předpokládejte, že soubor leží v kořenovém adresáři disku a má jméno největší možné délky. Očekává se údaj v bajtech, chybějící informace opět rozumně doplňte podle svého uvážení.

Nástin řešení

1. Z dentry se přečte číslo inode. Z rozměrů systému souborů (přečtených ze superbloku) se vypočítá pozice dané části tabulky inodů (superblok říká velikost bloku, velikost group, velikost inode table per group a velikost inode, tedy číslo inode děleno velikost inode table per group určí group, číslo inode modulo velikost inode table per group krát velikost inode určí pozici v group inode table). Přečte se inode. Z pozice a velikosti bloku se určí požadované číslo

bloku (pro pozici 123456 a velikost bloku 4096 je číslo bloku 31). Blok bude vyžadovat jednu úroveň indirekce (číslo je větší než `EXT2_NDIR_BLOCKS`), tedy se přečte blok odkazovaný `i_block[EXT2_IND_BLOCK]` a v něm se vezme příslušné číslo cílového bloku (`__u32` integer na pozici `31 - EXT2_NDIR_BLOCKS`), z něj se pak přečte `__u8` integer na pozici 123456 modulo velikost bloku. Převody čísla bloku na číslo sektoru podobně jako určení pozice v tabulce inode, čtení samozřejmě musí probíhat nejméně po celých sektorech (ale u dat spíše po celých blocích, protože se plní block cache).

2. Vedle samotných dat souboru jsou metadata v indirect bloku, v inode, v dentry, striktně vzato také ve free block bitmap a inode bitmap. U všech lze očekávat zarovnání, u dat na velikost bloku, u inode na rozumnou mocninu dvou (128 bajtů).

28 Problém splňování podmínek (CSP) (specializace UI-SU, UI-ZPJ)

Definujte problém splňování podmínek (CSP z anglického Constraint Satisfaction Problem). Popište prohledávání s navracením (backtracking) jako způsob řešení CPS. Vysvětlete pojmy „hranová konzistence“ (arc consistency), „kontrola dopředu“ (forward checking) a „pohled dopředu“ (look ahead).

Pomocí CSP modelujte problém Sudoku, tj. problém umístění čísel $1, \ldots, 9$ do mřížky 9×9 takovým způsobem, že se v žádném sloupci, řádku ani boxu velikosti 3×3 žádná cifra neopakuje více než jednou. Předpokládejte, že některá čísla jsou už v mřížce zadaná.

Nástin řešení

- Definice CSP – trojice (proměnné, domény, podmínky), obecně může být pro každou proměnnou jiná doména, podmínek je konečně mnoho. Podmínka je relace mezi proměnnými, podmnožina Kartézského součinu domén.
- Popis backtracking, arc consistency, forward checking a look ahead.
  - backtracking – přiřadíme hodnotu nějaké proměnné a opakujeme dokud nedostaneme validní řešení nebo spor. V případě sporu backtrackujeme a učiníme jiné rozhodnutí.
  - arc consistency – podmínka (arc, pokud máme pouze binární podmínky) je konzistentní, pokud pro každou hodnotu z domény existuje hodnota pro ostatní proměnné, které tuto podmínku splní. Hodnoty pro které neexistuje ohodnocení ostatních proměnných můžeme z domény vyloučit.
  - forward checking – po ohodnocení proměnné x zkontrolujeme ostatní proměnné, které se vyskytují v podmínkách spolu s x. Odstraníme nekonzistentní hodnoty.
  - look ahead – podobné jako forward check, ale kontrolujeme postupně všechny podmínky, dokud se nějaké proměnné mění doména. Tj. jde o silnější techniku než forward check. Obdobně můžeme říct, že po ohodnocení proměnné se problém udělá arc consistent.
- Model sudoku – každé políčko je jedna proměnná, každá proměnná má doménu $1, \ldots, 9$. Podmínky jsou „All Different“ (případně nerovnost po dvojicích) na vhodné podmnožiny proměnných (tj, sloupce, řádky, 3x3 políčka). Typicky máme v sudoku už zadaná nějaká čísla, ta můžeme přiřadit pomocí domén nebo pomocí podmínek.

29 chí-kvadrát test (test dobré shody) (specializace UI-SU)

1. Popište χ^2 test (test dobré shody). Jaká je jeho nulová a alternativní hypotéza? Jak se spočítá hodnota testové statistiky? Jaké má tato testová statistika pravděpodobnostní rozdělení?
2. Majitel obchodu měl během pěti pracovních dnů jednoho týdne celkem 100 zákazníků. Počty zákazníků v jednotlivých dnech (pondělí až pátek) byly postupně 25, 30, 15, 14 a 16. Pomocí χ^2 testu rozhodněte, zda se počty zákazníků v jednotlivých dnes statisticky významně liší. *Nápověda:* Kritické hodnoty testové statistiky na hladině významnosti $\alpha = 0.05$ pro 3-6 stupňů volnosti (df) jsou (postupně) 7,8 (df=3), 9,5 (df=4), 11,1 (df=5) a 12,6 (df=6).

Nástin řešení

1. Nulová hypotéza je, že pozorované četnosti jednotlivých kategorií odpovídají jejich očekávaným četnostem. Alternativní

hypotéza je, že neodpovídají. Testová statistika se vypočítá jako

$$\sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

, kde O_i jsou pozorované četnosti a E_i jsou očekávané četnosti. Pokud nulová hypotéza platí, má testová statistika χ^2 rozdělení s $n-1$ stupni volnosti, kde n je počet různých kategorií.

2. Zajímá nás, jestli počet zákazníků je každý den stejný. Očekávané četnosti jsou tedy $E_i = 20$. Dosazením do vzorce dostaneme hodnotu testové statistiky $(25-20)^2/20 + (30-20)^2/20 + (15-20)^2/20 + (14-20)^2/20 + (16-20)^2/20 = (25+100+25+36+16)/20 = 202/20 = 10.1$. Máme 5 kategorií, kritická hodnota testové statistiky se 4 stupni volnosti je 9,5, rozdíl v počtu zákazníků tedy je statisticky významný.

30 Kombinace více modelů (specializace UI-SU)

1. Popište hlavní myšlenky technik bagging a boosting. Popište podrobněji jeden vybraný konkrétní algoritmus pro boosting.
2. Jak funguje klasifikátor založený na náhodných lesech (random forest)? Jakým způsobem se trénuje?
3. Porovnejte rozhodovací stromy a náhodné lesy. Jaké jsou výhody a nevýhody těchto modelů?

Nástin řešení

1. Podstatou baggingu je vytvoření více modelů trénovaných na různých podmnožinách (bootsrap) trénovací množiny. Tyto podmnožiny jsou typicky samplovány s opakováním. Hlavním myšlenkou boostingu je trénování několika modelů postupně s tím, že další modely jsou trénovány na vážených datech, kde instance nesprávně klasifikované aktuálním klasifikátorem mají větší váhu. Příkladem může být např. algoritmus AdaBoost.
2. Náhodné lesy obsahují množinu mnoha rozhodovacích stromů, které hlasují o celkovém výsledku klasifikace/regrese. Jednotlivé rozhodovací stromy jsou trénované na podmnožině instancí trénovací množiny a na podmnožině příznaků.
3. Výhodou rozhodovacích stromů je jejich intrepretabilita a rychlost trénování. Výhodnou náhodných lesů je jejich (typicky) větší přesnost a schopnost generalizace.

31 Evaluační metriky (specializace UI-SU)

1. Definujte základní evaluační metriky pro binární klasifikátor – úspěšnost (accuracy), citlivost (sensitivity, recall), přesnost (precision), F1 skóre.
2. Chceme vytvořit systém pro detekci podvodných plateb kreditní kartou. Máme data, kde jsou miliony plateb, z nichž jen několik stovek jsou platby podvodné. Jaké metriky použijete pro vyhodocení takového systému? Jaké metriky naopak nebudou moc vhodné? Odpovědi zdůvodněte.
3. Pro výše zmíněnou úlohu jsme vytvořili dva systémy. Jeden z nich má citlivost 0,99 a přesnost 0,2. Druhý má citlivost 0,9 a přesnost 0,95. Vysvětlete, co tyto hodnoty prakticky znamenají. Který ze systémů byste použili? Své rozhodnutí zdůvodněte.

Nástin řešení

1. Definujme TP/TN/FP/FN jako počty true/false positive/negative. Správnost potom je (TP+TN)/(TP+FP+TN+FN), citlivost je TP/(TP+FN), přesnost je TP/(TP+FP) a F1 skóre je harmonický průměr citlivosti a přesnosti – 2TP/(2TP+FP+FN).
2. Pro vyhodnocení systému je vhodné použít dvojici citlivost a přesnost. Dá se také použít F1 skóre nebo AUC. Nevhodná naopak bude správnost vzhledem k výrazně nevyváženému množství instancí v jednotlivých třídách.
3. První model odhalí 99 % podvodných plateb, na druhou stranu ale jen 20 % z ohlášených podezřelých plateb budou platby podvodné. Druhý model sice odhalí jen 90 % podvodných plateb, ale 95 % z podezřelých plateb opravdu podvody

budou. V případě použití prvního modelu tedy bude potřeba prošetřit cca 5krát více případných podvodů, což může vést ke zbytečnému obtěžování zákazníků a i vyšším nákladům na prošetřování. Může tedy být vhodnější použít model druhý. Na druhou stranu, pokud bychom chtěli mít větší jistotu, že podvody odhalíme a výše zmíněné nevýhody nám nevadí, je vhodnější model první. Neexistuje tu jednoznačně správná odpověď, důležitá je především argumentace.