Bakalářské zkoušky (příklady otázek)

2025-09-08

1 Třídy složitosti (společné okruhy)

1. Definujte *třídy složitosti* **P** a **NP**.
2. Definujte *převod* (redukci) mezi rozhodovacími problémy (jazyky).
3. Definujte **NP**-*úplnost* rozhodovacího problému.
4. Rozhoďte, zda následující problém leží v třídě **NP**, a odpověď zdůvodněte: Je dáno přirozené číslo x zapsané v desítkové soustavě. Existují přirozená čísla a a b taková, že $1 < a, b < x$, $x = a \cdot b$ a desítkové zápisy a i b bez počátečních nul jsou stejně dlouhé?

Nástin řešení Definice viz Průvodce labyrintem algoritmů, kapitoly 19.1 a 19.3. Problém leží v třídě **NP**, jako certifikát stačí použít číslo a. Verifikátor pak vypočte $b = x/a$ (pokud by při dělení vyšel zbytek, zamítne) a zkontroluje, že $1 < a, b < x$ a desítkové zápisy a a b jsou stejně dlouhé. Délka certifikátu je lineární a časová složitost verifikátoru polynomiální, obojí vzhledem k délce vstupu (počtu číslic x).

2 Ukládání dat v binárním souboru (společné okruhy)

Aplikace ukládá svá data do binárního souboru, což je ilustrováno následujícím (částečným) hexadecimálním výpisem:

```
0000:  00 06 48 68  53 53 4C 4C  00 22 03 E8  FC 18 00 05  ..Hh SSLL ."..  ....
0010:  48 65 6C 6C  6F 00 05 57  6F 72 6C 64  00 00 00 00  Hell o..W orld  ....
0020:  00 00 00 2C  00 00 00 00  00 00 12 34  00 88 01 F4  ...,  ....  ...4  ....
```

Logicky soubor obsahuje *uzly*. Každý uzel obsahuje stejnou sadu *atributů*. Typy atributů jsou určeny *hlavičkou* souboru umístěnou na jeho začátku, která se skládá ze dvou bajtů udávajících počet atributů (v našem příkladu 6), následovaných typy atributů, kódovanými jedním bajtem pro každý atribut, odpovídající tomuto výčtu:

```
enum AttrType { TUInt16 = 0x48, TLink = 0x4C, TString = 0x53, TSInt16 = 0x68};
```

Vícebajtová čísla uložená v souboru jsou vždy v big-endian pořadí. Bezprostředně za hlavičkou se nachází *kořenový uzel*. Každý uzel začíná dvěma bajty obsahujícími délku dat uzlu v bajtech (v kořenovém uzlu našeho příkladu 34). Data uzlu obsahují hodnoty atributů, kódované podle typů atributů deklarovaných v hlavičce. Pro typy `TUInt16` a `TSInt16` je atribut uložen jako dva bajty obsahující 16bitové celé číslo bez znaménka nebo se znaménkem (dvojkový doplněk). Atributy typu `TString` jsou ASCII řetězce proměnné délky kódované jako dva bajty určující počet znaků, následované znaky, každý uložený v jednom bajtu. Pro `TLink` je uloženo 8 bajtů obsahujících pozici jiného uzlu v souboru. Částečný výpis ukazuje, že kořenový uzel nese hodnoty `{1000, -1000, "Hello", "World", 0x2C, 0x1234}`.

Binární soubor je reprezentován třídou `BinaryFile`, která poskytuje následující metodu, jež načte `len` bajtů do bufferu `buf`, začínajícího na absolutní pozici `filepos` v souboru (znaménkové typy se používají kvůli kompatibilitě s Javou):

```
void readBytes (long filepos, byte[] buf, int len);                 // Java, C#
void readBytes (std::int64_t filepos, unsigned char* buf, int len);    // C++
```

1. Napište deklaraci třídy `Reader`, včetně jejích privátních dat a implementace následujících metod: Konstruktor obdrží otevřený objekt `BinaryFile` jako parametr; okamžitě načte hlavičku. Metoda `attributes` vrací počet atributů. Metoda `getType` vrací typ `i`-tého atributu, číslováno od nuly. Metoda `getRoot` vrací pozici kořenového uzlu. Metoda `readNode`

vrátí nový objekt `Node` reprezentující uzel uložený na pozici `filepos`. Objekt je vracen odkazem v Java/C#, ale hodnotou v C++. Data uzlu se načítají ze souboru pouze touto funkcí.

2. Deklarujte a implementujte privátní datové prvky a konstruktor třídy `Node`. Třída má ukládat data uzlu jako pole bajtů, tj. tak, jak jsou uložena v souboru. Navíc má obsahovat strukturu, která umožní lokalizovat libovolný atribut v konstantním čase. Konverze bajtů na celá čísla nebo řetězce se provádí pouze při volání odpovídajících metod `get...` třídy `Node`.

3. Napište implementaci následujících metod třídy `Node` pro získání hodnoty celočíselných atributů se znaménkem a bez znaménka (16bit), použijte přitom pouze vestavěné operátory zvoleného jazyka:

```
int getUInt16(int i);
int getSInt16(int i);
```

Každá metoda čte `i`-tý atribut; je odpovědností volajícího zajistit, že volání odpovídá typu atributu. Typ `int` se považuje za dostatečně velký, aby obsahoval rozsah $\langle -32768, 65535 \rangle$.

3 Základy analýzy (společné okruhy)

Nechť $p \in (0, +\infty)$ je kladné reálné číslo. Označme U_p rovinný útvar, který je zleva ohraničen přímkou $x = p$, zprava přímkou $x = p + 1$, zdola osou x a shora grafem funkce $f(x) = x + \frac{1}{x}$. Jinými slovy,

$$U_p = \left\{ (x, y) \in \mathbb{R} \times \mathbb{R};\ p \leq x \leq p + 1 \wedge 0 \leq y \leq x + \frac{1}{x} \right\}.$$

1. Ukažte, že U_p má obsah $p + \ln\left(1 + \frac{1}{p}\right) + \frac{1}{2}$ čtverečních jednotek.

2. Existuje v intervalu $(0, 100)$ nějaká hodnota p, pro kterou má U_p obsah větší než 10 000 čtverečních jednotek?

3. Najděte hodnotu $p \in (0, +\infty)$, pro niž obsah U_p nabývá svého minima, případně ukažte, že taková hodnota neexistuje.

Nástin řešení

1. Označme $u(p)$ obsah útvaru U_p. Tento obsah je roven integrálu $\int_p^{p+1} f(x)dx$. Funkce $f(x) = x + \frac{1}{x}$ má primitivní funkci $F(x) = \frac{x^2}{2} + \ln(x) + c$ pro libovolné $c \in \mathbb{R}$. Hledaný obsah je tedy roven

$$\begin{aligned} u(p) &= F(p+1) - F(p) \\ &= \frac{(p+1)^2}{2} + \ln(p+1) - \frac{p^2}{2} - \ln(p) \\ &= \frac{2p+1}{2} + \ln(p+1) - \ln(p) \\ &= p + \frac{1}{2} + \ln\left(\frac{p+1}{p}\right) \\ &= p + \ln\left(1 + \frac{1}{p}\right) + \frac{1}{2}. \end{aligned}$$

2. Zde si stačí všimnout, že funkce $u(p)$ má pro p jdoucí k nule zprava limitu $+\infty$, a tedy na libovolném pravém okolí nuly nabývá libovolně velkých hodnot. Pro dostatečně malé $p > 0$ má tedy U_p obsah větší než 10 000 čtverečních jednotek.

3. Pomocí aritmetiky derivací a pravidel pro derivování složené funkce zjistíme, že funkce $u(p)$ má na uvažovaném intervalu $(0, +\infty)$ derivaci

$$u'(p) = 1 + \frac{1}{1 + \frac{1}{p}} \cdot \frac{-1}{p^2} = 1 - \frac{1}{p^2 + p}.$$

Vyřešením kvadratické rovnice zjistíme, že jediné kladné p_0 splňující $u'(p_0) = 0$ je $p_0 = \frac{-1+\sqrt{5}}{2}$. Zároveň vidíme, že $u'(p)$ je rostoucí funkce – to lze vidět přímo ze vzorce pro $u'(p)$, případně výpočtem druhé derivace

$$u''(p) = \frac{2p+1}{(p^2+p)^2},$$

která je pro $p \in (0, +\infty)$ zjevně kladná. Z toho plyne, že $u'(p)$ je záporná pro $p \in (0, p_0)$ a kladná pro $p > p_0$, a tedy funkce $u(p)$ je klesající na intervalu $(0, p_0]$ a rostoucí na intervalu $[p_0, +\infty)$.

Funkce $u(p)$ tedy v bodě p_0 nabývá své jediné minimum.

4 Binární relace (společné okruhy)

1. Definujte podmínky, které musí splňovat binární relace R na neprázdné množině X, aby byla ekvivalencí. Podmínky formulujte pomocí matematických zápisů s použitím logických spojek, kvantifikátorů a podobně.
2. Pro $X = \{a, b, c, d\}$ určete, kolik různých ekvivalencí na množině X existuje.
3. Pro $X = \{a, b, c, d\}$ najděte příklad binární relace, která je tranzitivní, ale není symetrická ani (slabě) antisymetrická. Zdůvodněte, že nalezená relace má požadované vlastnosti.

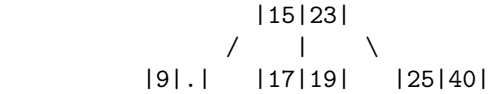
   (Antisymetrií míníme $\forall x, y \in X : ((x, y) \in R \wedge (y, x) \in R) \Rightarrow x = y$.)

Nástin řešení

1. Ekvivalence je relace, která je
   - reflexivní $\forall x \in X : (x, x) \in R$
   - symetrická $\forall x, y \in X : (x, y) \in R \Rightarrow (y, x) \in R$
   - tranzitivní $\forall x, y, z \in R : ((x, y) \in R \vee (y, z) \in R) \Rightarrow (x, z) \in R$
2. Budeme počítat rozklady X na třídy ekvivalence, rozborem případů dle velikostí rozkladových tříd: jedna třída (velikosti 4) 1x, dvě třídy $1 + 3$ 4x, $2 + 2$ 3x, tři třídy $(2 + 1 + 1)$ 6x, čtyři jednoprvkové třídy 1x, celkem 15 možností.
3. Například $R = \{(a, a), (a, b), (b, a), (b, b), (c, d)\}$. Tranzitivita rozborem případů (předpoklady splňují volby $x = a, y = b, z = a$ a $x = b, y = a, z = b$), volba $x = a, y = b$ porušuje antisymetrii, volba $x = c, y = d$ porušuje symetrii.

5 Hierarchické indexy (specializace WDOP)

1. Definujte datovou strukturu B-strom.
2. Uvažujte následující neredundantní B-strom stupně 3. Vložte do něj hodnotu 45 a následně smažte hodnotu 9. Nakreslete stav po každé z operací.



3. V relačních databázových systémech se ovšem pro indexaci sloupců využívají především B+ stromy. Vysvětlete, jak se liší od B-stromu, a co tyto modifikace v tomto kontextu přinášejí za výhody.

Nástin řešení Odpověď 1.

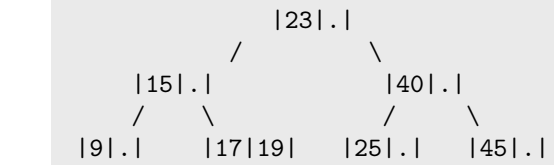
B-stromem stupně m rozumíme strom splňující následující podmínky:

- Kořen má alespoň 2 potomky, pokud není list.
- Každý uzel (kromě kořene) má alespoň $\lceil m/2 \rceil$ potomků.
- Každý uzel má nejvýše m potomků (stupeň).

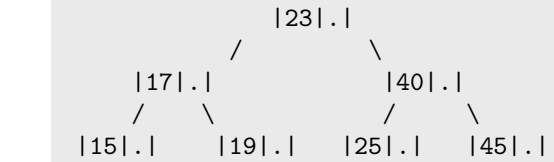
- Každý uzel obsahuje $k-1$ klíčů, kde k je počet potomků (tedy počet klíčů = počet potomků – 1).
- Klíče v uzlu jsou uspořádány vzestupně a rozdělují intervaly hodnot pro podstromy:
  - klíče v prvním podstromu < první klíč,
  - klíče mezi i-tým a $(i+1)$-ním potomkem jsou mezi i-tým a $(i+1)$-ním klíčem,
  - klíče v posledním podstromu > poslední klíč.
- Všechny listy se nacházejí na stejné úrovni (strom je vyvážený).

Odpověď 2.

Operace 1:



Operace 2:



Odpověď 3.

B+ strom:

- Vnitřní uzly obsahují pouze klíče (ne data).
- Všechna data jsou uložena až v listech.
- Listy jsou navíc mezi sebou propojené (lineární seznam).
- Výhody: rychlejší sekvenční procházení, efektivnější rozsahové dotazy.

6 Moderní databázové systémy (specializace WDOP)

Uvažujte následující situaci: Chceme vhodně reprezentovat entity *kniha*, *autor* a *vydavatelství* a vztahy mezi nimi zachycující kde byla kniha vydána, kdo ji napsal a zda jsou dva autoři přátelé. Dále chceme vyhodnocovat např. následující dotaz: "Které knihy napsal Dan Brown s někým, kdo nepatří do jeho přátel?"

1. Demonstrujte, jak byste taková data reprezentovali v grafové databázi jako např. neo4j, a jak by bylo možné dotaz vyjádřit např. v jazyce Cypher. (Je možné využít i jinou grafovou databázi podobného typu nebo jazyk. Pak specifikujte, o které jde. Není požadována absolutně přesná syntaxe jazyka, ale je třeba demonstrovat správnou znalost jeho základních principů.)
2. Naznačte (stačí schematicky, ne nutně přímo v jazyce SQL), jak by bylo možné stejnou situaci reprezentovat v tradiční relační databázi. Porovnejte, jak se tyto dva přístupy liší, jaké mají výhody a nevýhody (v tomto případě, popř. i obecně).

Nástin řešení Odpověď 1.

Reprezentace:

- Uzly: `(Autor {jmeno})`, `(Kniha {nazev})`, `(Vydavatelstvi {nazev})`
- Hrany:
  - `(Autor)-[:NAPSAL]->(Kniha)`

- `(Kniha)-[:VYDAL]->(Vydavatelstvi)`
- `(Autor)-[:PRITEL]-(Autor)`

Dotaz:

```
MATCH (dan:Autor {jmeno: "Dan Brown"})-[:NAPSAL]->(k:Kniha)<-[:NAPSAL]-(spoluautor:Autor)
WHERE dan <> spoluautor
  AND NOT (dan)-[:PRITEL]-(spoluautor)
RETURN DISTINCT k.nazev;
```

Odpověď 2.

Reprezentace:

- **Autor** (id_autor **PK**, jméno)
- **Vydavatelství** (id_vydavatelstvi **PK**, název)
- **Kniha** (id_kniha **PK**, název, id_vydavatelstvi **FK → Vydavatelství**)
- **Napsal** (id_autor **FK → Autor**, id_kniha **FK → Kniha**, **PK** = (id_autor, id_kniha))
- **Přítel** (id_autor1 **FK → Autor**, id_autor2 **FK → Autor**, **PK** = (id_autor1, id_autor2))

V relační databázi by se dotaz skládál z několika drahých spojení tabulek. Komplikované by bylo, pokud bychom chtěli vyjádřit např. přátele přátel, nebo obecně rekurzi. Složité je i přidávání nových typů vztahů.

7 Vyhledávání na webu (specializace WDOP)

Do databáze dokumentů bylo zaindexováno prvních pět dokumentů. Data byla uložena v podobě matice, kde sloupce reprezentují jednotlivé termy a řádky dokumenty.

	A	**B**	**C**	**D**	**E**
D_1	1	0	1	1	0
D_2	**0**	**1**	**1**	**0**	**1**
D_3	**1**	**1**	**1**	**0**	**0**
D_4	0	1	0	1	1
D_5	1	0	1	0	1

Nad touto databází byly implementovány dvě klientské aplikace. První z nich podporuje boolský model vyhledávání a druhá model vektorový. Uživatel, pro něhož databáze obsahuje dva relevantní dokumenty D_2 a D_3, položil prostřednictvím první aplikace dotaz "A and (B or C)"a prostřednictvím druhé aplikace dotaz "(1, 1, 1, 0, 0)".

1. Vysvětlete, jak se ve vektorovém modelu vyhodnocuje Kosinová míra podobnosti. Určete, jaká ohodnocení přidělí dokumentům první aplikace, a jaká ohodnocení jim přidělí aplikace druhá, pokud používá právě Kosinovu míru. Jaké dokumenty jednotlivé aplikace vrátí jako odpověď a v jakém pořadí?
2. Popište, co znamenají pojmy přesnost a úplnost pro vyhodnocení efektivity vyhledávacího modelu a spočtěte je pro odpovědi poskytnuté výše uvedenými aplikacemi pro položené dotazy.
3. Lze nad těmito zaindexovanými dokumenty prostřednictvím jednotlivých klientských aplikací položit takový dotaz, který by vrátil právě a pouze oba relevantní dokumenty? Pokud ano, jak by dotaz vypadal? Pokud ne, proč?

Nástin řešení

1. Kosinova míra podobnosti dvou k-rozměrných vektorů $\vec{D}$ a $\vec{q}$ se spočte jako $(\vec{D} \cdot \vec{q})/(|\vec{D}| \cdot |\vec{q}|) = \sum_{n=1}^{k}(D_i * q_i)/\sqrt{\sum_{n=1}^{k} D_i^2}/\sqrt{\sum_{n=1}^{k} q_i^2}$. Boolský model ohodnotí dokumenty vyhovující podmínce hodnotou 1 (true), ostatní hodnotou false (0). V daném případě budou ohodnocení po řadě 1, 0, 1, 0, 1 a aplikace vrátí dané tři dokumenty v neurčeném pořadí. Vektorový model ohodnotí dokumenty po řadě hodnotami $2/\sqrt{3}/\sqrt{3}$, $2/\sqrt{3}/\sqrt{3}$, $3/\sqrt{3}/\sqrt{3}$, $1/\sqrt{3}/\sqrt{3}$, $2/\sqrt{3}/\sqrt{3}$, tedy hodnotami 2/3, 2/3, 3/3, 1/3, 2/3. Model vrátí všech pět dokumentů. D_3 bude první, D_4 bude poslední, pořadí ostatních nebude určeno.

2. Přesnost P se získá jako podíl počtu vrácených relevantních dokumentů a počtu vrácených dokumentů. Uplnost R se získá jako podíl počtu vrácených relevantních dokumentů a počtu všech relevantních dokumentů. První vyjadřuje pravděpodobnost, že vrácený dokument je relevantní, druhá pravděpodobnost, že relevantní dokument je vrácený. V případě Boolského modelu získáme hodnoty $P = 1/3$ a $R = 1/2$. V případě vektorového modelu získáme hodnoty $P = 2/5$ a $R = 2/2$.
3. V případě Boolského modelu by šlo napsat podmínku například "B and C". V případě vektorového modelu jednička na libovolné pozici do odpovědi zařadí i nějaký nerelevantní dokument. Dotaz "(0, 1, 1, 0, 0)" by vrátil požadované dokumenty na prvních dvou místech, ale nasledované i zbylými dokumenty.

8 Transakční rozvrhy (specializace WDOP)

Jsou dány dva transakční rozvrhy S_1 a S_2:

S_1:	T_1	T_2	T_3
1)	$R(A)$		
2)		$W(B)$	
3)			$W(A)$
4)	$R(B)$		
5)	COMMIT		
6)		$R(A)$	
7)		COMMIT	
8)			$W(B)$
9)			COMMIT

a

S_2:	T_1	T_2	T_3
1)	$R(A)$		
2)	$R(B)$		
3)	COMMIT		
4)		$W(B)$	
5)		$R(A)$	
6)		COMMIT	
7)			$W(A)$
8)			$W(B)$
9)			COMMIT

1. Definujte konfliktovou ekvivalenci rozvrhů a rozhodněte, zda jsou rozvrhy S_1 a S_2 konfliktově uspořádatelné. Své rozhodnutí zdůvodněte.
2. Je rozvrh S_1 zotavitelný? Své rozhodnutí zdůvodněte. Pokud ne, navrhněte, zda a jak by šel zotavitelným učinit, aniž by se změnilo vzájemné pořadí čtení a zápisů v rozvrhu.
3. Je zaručeno, že oba rozvrhy S_1 a S_2, pokud by byly spuštěné na stejných datech, povedou ke shodnému výslednému stavu databáze? Své rozhodnutí zdůvodněte.

Nástin řešení

1. Dva rozvrhy jsou konfliktově ekvivalentní, pokd v obou všechny dvojice konfliktních operací (stejná proměnná, jiná transakce, ne dvojice čtení) probíhají ve stejném relatviním pořadí. S_1 je konfliktově uspořádatelný. Dvojice operací 1-3, 3-6, 2-4 a 2-8 mají stejné relativní pořadí, jaké by měly v sériovém rozvrhu T_2 T_1 T_3. Lze ukázat i na precedenčním grafu, který nebude obsahovat cykly. Rozvrh S_1 je sám o sobě sériový, a je tedy konfliktově ekvivalentní sám se sebou.
2. Není. Operace 4 čte nepotvrzenou hodnotu, zapsanou v 2, ale T_1 potvrzuje dříve, než T_2. Podobně čte operace 6 nepotvrzenou hodnotu, zapsanou v 3, ale T_2 potvrzuje dříve, než T_3. Pokud se nemá prohodit pořadí čtení a zápisů v rozvrhu, je potřeba s COMMITem v T_2 počkat, až skončí T_3. S COMMITem v T_1 je třeba počkat, až opožděně skončí T_2.

3. Není. S_1 je konfliktově ekvivalentní s $T_2\ T_1\ T_3$, ale ne s $T_1\ T_2\ T_3$. V prvním rozvrhu čte T_1 hodnotu A zapsanou v T_2. Ve druhém čte původní hodnotu v databázi. Zbytek transakce a celkový výsledek tedy může být jiný.

9 Ramseyova věta a odhady (specializace OI-G-O, OI-G-PADS, OI-G-PDM, OI-O-PADS, OI-PADS-PDM)

1. Pro přirozená čísla $n \geq k \geq 2$ uvažujme náhodné obarvení hran kliky na n vrcholech dvěma barvami; každá hrana je nezávisle na ostatních obarvena s pravděpodobností 50% červeně a 50% modře. Jako m označme počet k-prvkových podmnožin M vrcholů této kliky takových, že všechny hrany mezi vrcholy M jsou modré. Ukažte, že střední hodnota náhodné veličiny m je méně než
$$2^{k \log_2 n - \binom{k}{2}}.$$
Co z toho plyne pro velikost Ramseyova čísla $R(k,k)$?
2. Nechť $n \geq 3$ a $k \geq 3 + 2\log_2 n$ jsou přirozená čísla. Nechť $q(n,k)$ je počet grafů na vrcholech $\{1,\ldots,n\}$ neobsahujících žádnou kliku velikosti k. Nechť $p(n)$ je počet permutací $\lfloor \frac{n^2}{10 \log_2 n} \rfloor$ prvků. Které z čísel $q(n,k)$ a $p(n)$ je větší?

Nástin řešení

1. Pro každou k-prvkovou podmnožinu M je pravděpodobnost, že všechny hrany mezi jejímy vrcholy jsou modré, rovna $2^{-\binom{k}{2}}$. Z linearity střední hodnoty je tedy
$$\mathbb{E}[m] = \binom{n}{k} \cdot 2^{-\binom{k}{2}} < n^k \cdot 2^{-\binom{k}{2}} = 2^{k \log_2 n - \binom{k}{2}}.$$
Ramseyovo číslo $R(k,k)$ je nejmenší přirozené číslo takové, že v každém obarvení hran kliky na $R(k,k)$ vrcholech dvěma barvami najdeme monochromatickou kliku velikosti k. Zde uvažujeme nejenom modře obarvené kliky, ale i červené. Střední hodnota počtu monochromatických klik velikosti k v náhodném obarvení kliky na n vrcholech tedy je
$$2\mathbb{E}[m] < 2^{k \log_2 n - \binom{k}{2} + 1}.$$
Jestliže $n \leq 2^{k/2-1}$, pak je tato střední hodnota menší než 1, a proto existuje nějaké obarvení, které žádnou monochromatickou kliku velikosti k neobsahuje. Z toho plyne, že $R(k,k) > 2^{k/2-1}$.
2. Uvažme náhodně zvolený graf G na vrcholech $\{1,\ldots,n\}$. Počet klik velikosti k v tomto náhodném grafu odpovídá náhodné veličině m z předchozí úlohy. S využitím omezení $k \geq 3 + 2\log_2 n$ tedy dostáváme, že střední hodnota počtu těchto klik je méně než
$$2^{k \log_2 n - \binom{k}{2}} \leq 2^{\frac{k(k-3)}{2} - \binom{k}{2}} = 2^{-k} < 1/2.$$
Z Markovovy nerovnosti je tedy pravděpodobnost, že G má alespoň jednu kliku velikosti k, menší než $1/2$. Počet všech grafů s vrcholy $\{1,\ldots,n\}$ je $2^{\binom{n}{2}}$, proto $q(n,k) > 2^{\binom{n}{2}-1}$.

   Nechť $x = \lfloor \frac{n^2}{10 \log_2 n} \rfloor$; zjevně $x < n^2$. Počet permutací x prvků je
$$p(n) = x! \leq x^x = 2^{x \log_2 x} < 2^{2x \log_2 n} \leq 2^{\frac{n^2}{5}} < 2^{\binom{n}{2}-1} < q(n,k).$$
   Číslo $q(n,k)$ je tedy větší než $p(n)$.

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

1. Formulujte přesné znění silné věty o dualitě lineárního programování.
2. Je dán neorientovaný graf $G = (V,E)$ a vrcholy $s_1, s_2, t_1, t_2 \in V$. Pro $i = 1,2$ označme P_i množinu všech cest mezi vrcholy s_i a t_i v daném grafu a nechť $P = P_1 \cup P_2$. Uvažme následující lineární program, ve kterém máme proměnnou

x_p pro každou cestu z množiny P:

$$\begin{aligned} \max \sum_{p \in P} x_p & \\ \sum_{p: e \in p} x_p \leq 1 & \quad \forall e \in E \\ x_p \geq 0 & \quad \forall p \in P \end{aligned}$$

Napište duální program (duální proměnné označujte y, s patřičnými indexy).

3. Uvažme graf G daný množinou vrcholů $V = \{s_1, s_2, t_1, t_2, a, b, c, d\}$ a množinou hran

$$E = \{s_1a,\ s_2a,\ s_1b,\ s_2b,\ ac,\ bd,\ t_1c,\ t_2c,\ t_1d,\ t_2d\}$$

(pro stručnost zápisu používáme pro neorientovanou hranu mezi vrcholy a a c označení ac).

Pokud existuje, najděte optimální řešení výše uvedené primární úlohy a pomocí silné věty o dualitě dokažte, že je opravdu optimální.

Nástin řešení

1. Silná dualita pro LP: Pro dvojici lineárních programů P a D nastává jedna z těchto čtyř možností:
   (a) Oba P i D jsou neřešitelné.
   (b) P je neomezený a D je neřešitelný.
   (c) D je neomezený a P je neřešitelný.
   (d) Oba P i D jsou řešitelné a existují optimální řešení x, y pro P a D taková, že $c^T x = b^T y$.
2. V duálním LP je proměnná y_e pro každou hranu $e \in E$, a jeho účelová funkce a podmínky jsou následující:

$$\begin{aligned} \min \sum_{e \in E} y_e & \\ \sum_{e \in p} y_e \geq 1 & \quad \forall p \in P \\ y_e \geq 0 & \quad \forall e \in E \end{aligned}$$

3. LP1 je formulace maximálního toku mezi s_1–t_1 a s_2–t_2. Jediné cesty mezi s_1–t_1 a s_2–t_2 jsou

$$p_1 : s_1{-}a{-}c{-}t_1, \quad p_2 : s_1{-}b{-}d{-}t_1, \quad p_3 : s_2{-}a{-}c{-}t_2, \quad p_4 : s_2{-}b{-}d{-}t_2.$$

Možné primární přípustné řešení je $x_{p_1} = 1$, $x_{p_3} = 1$, $x_{p_2} = 0$, $x_{p_4} = 0$ s hodnotou účelové funkce 2.

Duální přípustné řešení je $y_{ac} = 1$, $y_{bd} = 1$ a $y_e = 0$ pro ostatní hrany; hodnota účelové funkce je 2. Podle věty o silné dualitě je výše uvedené primární řešení (stejně jako duální) optimální.

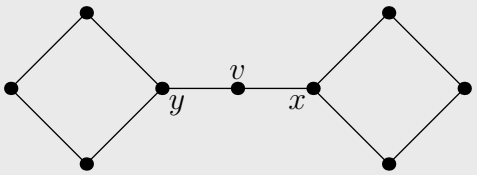
11 Hladový algoritmus pro barevnost grafu (specializace OI-G-PDM, OI-O-PDM, OI-PADS-PDM)

Uvažujme hladový algoritmus na obarvení grafu G barvami 1, 2, ..., fungující následovně: Má-li G jen jeden vrchol, přiřadíme tomuto vrcholu barvu 1 a skončíme. Jinak vybereme libovolný vrchol v nejmenšího stupně, rekurzivním voláním nalezneme obarvení grafu $G - v$, a nakonec vrcholu v přiřadíme nejmenší barvu, která není přiřazena žádnému sousedovi vrcholu v.

1. Ukažte, že tento algoritmus obarví každý rovinný graf G použitím nejvýše 6 barev.
2. Najděte bipartitní graf, pro který tento algoritmus nemusí nutně nalézt obarvení dvěma barvami.
3. Ukažte, že tento algoritmus obarví každý chordální graf G optimálně, tj. s použitím nejvýše $\chi(G)$ barev.

Nástin řešení

1. Minimální stupeň rovinného grafu je nejvýše 5 (a každý indukovaný podgraf rovinného grafu je rovinný). Každý vrchol v, který popsaný algoritmus vybere, tedy má vždy stupeň nejvýše 5, a v okamžiku, kdy ho barvíme, proto na jeho sousedech nebude použita alespoň jedna z barev $1, \ldots, 6$. Každému z vrcholů tedy bude přiřazena jedna z těchto barev.
2. Uvažme například následující graf:



   Nejprve algoritmus může jako vrchol v nejmenšího stupně vybrat označený prostřední vrchol. Při rekurzivním volání na podgraf $G - v$ pak algoritmus nalezne nějaké obarvení obou zbývajících 4-cyklů pomocí barev 1 a 2. Jelikož tyto 4-cykly jsou rotačně symetrické, jistě se ale může stát, že vrchol x dostane barvu 1 a vrchol y barvu 2. Pro vrchol v pak budeme muset použít barvu 3.
3. Každý indukovaný podgraf chordálního grafu je chordální, obdobně jako v úloze s rovinným grafem tedy stačí ukázat, že každý chordální graf G má minimální stupeň menší než $\chi(G)$. Každý chordální graf má simpliciální vrchol u, tj. vrchol, jehož sousedé tvoří kliku. Jelikož barevnost grafu je vždy větší nebo rovna velikosti $\omega(G)$ největší kliky, dostáváme tedy $\deg u < \omega(G) \leq \chi(G)$.

12 Komparátorové sítě (specializace OI-G-PADS, OI-O-PADS, OI-PADS-PDM)

1. Definujte komparátor.
2. Popište konstrukci třídicí komparátorové sítě o n vstupech s hloubkou $\mathcal{O}(\log^2 n)$. Správnost konstrukce nemusíte dokazovat, ale hloubku vypočtěte.
3. Dokažte, že nemůže existovat třídicí komparátorová síť s hloubkou asymptoticky menší než logaritmickou.

Nástin řešení Definice a konstrukce viz Průvodce labyrintem algoritmů, kapitola 15.3. Ve 3. části si stačí uvědomit, že žádná vrstva komparátorové sítě o n vstupech nemůže obsahovat víc než n komparátorů. Síť hloubky $h(n)$ tedy provede nejvýše $n \cdot h(n)$ porovnání. To by pro $h \in o(\log n)$ bylo ve sporu s dolním odhadem složitosti třídění, který říká, že k setřídění n prvků je v nejhorším případě potřeba provést $\Omega(n \log n)$ porovnání.

13 Vlastnosti funkcí více proměnných (specializace OI-G-O, OI-G-PADS, OI-G-PDM, OI-O-PADS, OI-PADS-PDM)

Definujme funkci $f\colon \mathbb{R}^2 \to \mathbb{R}$ následovně:

$$f(x,y) = \begin{cases} 0 & \text{pro } y = 0 \\ \frac{\exp(-x^2)}{y} & \text{pro } y \neq 0, \end{cases}$$

kde $\exp(x)$ označuje exponenciální funkci e^x.

1. Je funkce f spojitá v bodě $(0,0)$?
2. Definujme množinu $M \subseteq \mathbb{R}^2$ takto:

$$M = \left\{ (x,y) \in \mathbb{R}^2;\ x > 0 \wedge y \geq \frac{1}{x} \right\}.$$

   Je množina M uzavřená? Je množina M kompaktní?
3. Jakou největší hodnotu nabývá funkce f na množině M a v kterém bodě množiny M se tato hodnota nabývá?

Nástin řešení

1. Funkce f není spojitá v bodě $(0,0)$. To je vidět třeba z toho, že funkce $f(0,y) = \frac{1}{y}$ není na žádném prstencovém okolí nuly omezená, tudíž ani nemá vlastní limitu pro $y \to 0$.
2. Množina M je uzavřená (kolem každého bodu $\mathbb{R}^2 \setminus M$ existuje okolí disjunktní s M), ovšem není kompaktní, neboť není omezená.
3. Své maximum na množině M nabývá funkce f v bodě $\left(\frac{1}{\sqrt{2}}, \sqrt{2}\right)$, kde má hodnotu $\frac{e^{-1/2}}{\sqrt{2}} = \frac{1}{\sqrt{2e}}$. Funkce f nemůže nabývat extrémy ve vnitřních bodech M, neboť obě její parciální derivace jsou na M záporné. Z toho plyne, že funkce f je na M klesající v obou proměnných, a tedy speciálně pro každý vnitřní bod (x,y) množiny M platí $f(x, \frac{1}{x}) > f(x,y)$. To ukazuje, že maximum f může být jen na hranici množiny M, tedy v bodech tvaru $\left(x, \frac{1}{x}\right)$. Funkce $f(x, \frac{1}{x}) = xe^{-x^2}$ nabývá svého jediného maxima v bodě $x = \frac{1}{\sqrt{2}}$, což odpovídá i hledanému maximu funkce f na množině M.

14 Poloprůhlednost (specializace PGVVH-PG, PGVVH-VPH, PGVVH-PV)

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 jakýkoli 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...

15 Scanline vyplňování (specializace PGVVH-PG, PGVVH-VPH, PGVVH-PV)

Budeme se zabývat vybarvením vnitřku rovinného mnohoúhelníka v rastrovém prostředí (čtvercová mřížka pixelů), obrys útvaru není potřeba obkreslovat.

1. Jak by měl být mnohoúhelník zadán a jaké možnosti definice jeho vnitřku mají smysl?

2. Popište základní princip algoritmu řádkového vyplňování 2D mnohoúhelníka (uveďte i případné pomocné datové struktury používané v průběhu vyplňování).

3. Jak by se algoritmus zjednodušil, kdyby měl vyplňovat pouze konvexní útvary?

Nástin řešení 1. Mnohoúhelník je zadán uzavřenou posloupností vrcholů, nezáleží, ve kterém vrcholu se začne ani na orientaci (CW nebo CCW). Každý vrchol má dvojici souřadnic $[x, y]$. Dvě nejčastější definice vnitřku:

1. Podle parity (odd-even rule) - každá hrana mění příslušnost do vnitřku polygonu, přejdu přes jednu hranu ... jsem uvnitř, přejdu přes další ... jsem opět venku.
2. Podle stupně obtočení (winding rule) - jakoby byl obvod polygonu vyroben z provázku, pokud do daného bodu zapíchnu prst, jsem venku, pokud lze provázek odstranit, jinak jsem uvnitř.

2. Řádkové vyplňování: reprezentuji si všechny nevodorovné hrany a budu udržovat jejich průsečíky s vodorovnou vykreslovanou řádkou (scanline). Tou řádkou postupuji shora dolů po jednom pixelu a na ní vždy jednoduše vybarvím vnitřní pixely polygonu. K tomu mi poslouží setřídění průsečíků zleva doprava (podle souřadnice x - viz seznam C níže) a aplikace příslušného pravidla z 1. Po řádkách se postupuje indukcí: řádka se posune o jeden pixel dolů, seznam průsečíků C se musí upravit:

1. Pokud se na nové řádce nachází některý z vstupních vrcholů polygonu (viz seznam I), tak vzniká nový průsečík (průsečík zanikne automaticky, když se vynuluje jeho čítač)
2. Ostatní průběžné průsečíky se posunou v souřadnici x o diferenci (směrnici = konstantu, kterou jsme si dopředu u každé hrany spočítali dělením dx/dy) a dekrementuje se jejich čítač
3. Protože se horizontální polohy průsečíků mění, musí se přetřídit seznam C

Algoritmus končí, když zanikne poslední průsečík (tj. jeho hrana skončí). Pomocná data: setříděný seznam průsečíků C ("current": x, dx/dy, čítač = za kolik řádků hrana skončí). Vstupní data se převedou na vstupní seznam I "input", kde jsou dosud nezpracované hrany ($[x, y]$, dx/dy, výška hrany v pixelech = iniciální hodnota pro čítač); tento seznam je nejlepší udržovat setříděný podle y (v každé řádce se v něm bude hledat, jestli nějaký nový průsečík nevzniknul).

3. Konvexní mnohoúhelník: seznam C se zredukuje na dvojici průsečíků (začátek a konec vyplňování), nemusí se třídit. Přechod na další řádku (scanline) bude jednodušší – pokud některá z těchto dvou hran končí, nahradí se sousední hranou (v rámci pořadí hran v obrysu polygonu)

16 Hierarchie 3D scény (specializace PGVVH-PG, PGVVH-VPH, PGVVH-PV)

Bez ohledu na konkrétní technickou stránku reprezentace 3D scény v paměti se často v praxi používá koncept "hierarchie". Vzpomeňte si na situace, kdy jste se s hierarchickými přístupy setkali.

1. Popište principy hierarchické reprezentace, zejména s ohledem na maticové transformace a případné atributy objektů. Definujte sémantiku transformací (transformačních matic) a svoje rozhodnutí zdůvodněte.
2. Je možné do konceptu zahrnout možnost opakovaného použití shodných objektů ("instancing")? Může hierarchie scény pomoci při editování scény pomocí předem připravené databáze modelů nebo komponent? Naznačte princip implementace.
3. Uveďte jeden konkrétní případ hierarchické reprezentace, může být z 3D nebo i 2D grafiky. Svůj příklad doplňte popisem jednoho klíčového algoritmu, který se nad danou datovou strukturou musí implementovat.

Nástin řešení **Princip hierarchie:** základní myšlenkou hierarchických datových struktur je dekompozice scény na části, které se opět mohou dělit na menší části, apod. V počítačová grafice je důležitý koncept "B je částí A", což s sebou nese definici geometrické transformace, kterou musí B podstoupit, než se stane součástí A. Transformace = maticová transformace homogenní maticí 4x4. Pokud ukládáme takové relativní transformace lokálně, dostaneme nakonec strom nebo DAG, kde jsou v hranách uloženy právě tyto transformace. Uzly grafu jsou objekty v širším slova smyslu, nějaké součásti světa, na které má smysl se odkazovat.

Pokud pracujeme s hierarchickým **stromem**, lze ještě zavést pojem "atribut" a "dědičnost atributu". Každý uzel grafu může mít přiřazeny atributy definující některé vlastnosti objektu: barvu, materiál, texturu ... nebo i negrafické atributy (metadata).

Dědičnost umožňuje definovat atributy jen v některých vnitřních uzlech, do podřízených částí stromu se přenesou dědičností. DAG graf má použití atributů problematičtější, ale též se dá vymyslet smysluplná sémantika.

Shrnutí **lokálních transformací**: vyjadřují převod souřadné soustavy podřízeného do souřadné soustavy nadřízeného uzlu. Pokud požadované algoritmy potřebují i opačnou transformaci, může se i ona (inverzní matice) ukládat/cachovat u každé hrany.

Instancing: pokud máme dovoleno použít DAG, je instancing snadný: prostě se použije tolik odkazů na daný uzel (objekt), kolik je potřeba. Každý odkaz je opatřen svou transformační maticí. Pokud bychom chtěli i modifikovat atributy, musela by se v interní implementaci systému místo prostého ukazatele používat cesta od kořene k uzlu objektu (aby se daly vyhodnotit atributy).

Databáze modelů, součástek, prefabů: jednoduchá koncepce, kde uzlem v grafu může být odkaz do databáze. Implikuje to použití systému DAG (není to strom). Uživatel např. grafického 3D editoru si pak může své vlastní objekty sestavovat i s pomocí dílů z databáze, výsledky zpět do privátní databáze ukládat a vlastně i celková finální 3D scéna bude jedním takovým objektem. Při přesunování celých součástek/modelů se modifikují jen matice spojené s jejich odkazy, vlastní položky databáze zůstávají neměnné. Pozn.: komplikovanější editory 3D scén musí umožňovat i koncept "Copy on Change" dávající volbu použitý modul modifikovat, pokud by bylo potřeba v něm udělat změnu...

Příklad: 3D scéna pro ray-tracing. Vnitřní uzly mohou (ale nemusí) reprezentovat množinové operace (pak to je "CSG strom"), odkazy na podřízené uzly obsahují obě transformační matice, ta původní (zdola nahoru) se používá pro editaci nebo sestavování scény, ta inverzní (shora dolů) je zase potřeba při výpočtu průsečíku paprsku se scénou. To je taky náš důležitý algoritmus:

Paprsek je původně definován ve **světovém systému souřadnic** (kořen stromu scény). Protože je mnohem jednodušší transformovat paprsek ($P_0 + t \cdot \vec{p_1}$) než tělesa, postupně se při průchodu grafem od kořene k listům transformuje paprsek až do souřadné soustavy jednoduchého tělesa v listu. Tam se provede výpočet průsečíku a výsledek je už pouze v podobě parametru t přenesen do původního prostoru.

Kdybychom potřebovali scénu **animovat**, tak přesouvání nebo otáčení jednotlivých objektů nebo větších celků se snadno udělá pouhou modifikací transformačních matic ve hranách. Změna polohy/orientace jednoho objektu = změna matice u jedné hrany.

17 Urychlování ray tracingu (specializace PGVVH-PG)

Předpokládejme, že se scéna pro ray tracing skládá pouze z trojúhelníků. Počet trojúhelníků označíme N, může to být číslo v řádu milionů až miliard. Půjde nám o formulaci metody, která bude mít logaritmickou složitost výpočtu průsečíku.

Zvolte si libovolný přístup, který povede k časové složitosti $O(\log N)$ výpočtu průsečíku jednoho paprsku s celou scénou. Můžete předpokládat, že nás zajímá jen nejbližší průsečík od počátku paprsku.

O jiných metodách nepište, pouze o té jedné, kterou jste si zvolili.

1. Popište dostatečně přesně vstupní data: reprezentaci 3D scény a reprezentaci paprsku. Snažte se Vaše deklarace příliš nekomplikovat, můžete upozornit na další doplňující data, která by se mohla pro rendering hodit. Jaká bude reprezentace výsledku (průsečíku)?
2. Popište dostatečně přesně pomocnou datovou strukturu, která se pro akceleraci bude používat. Popište algoritmus její konstrukce, naznačte, které postupy by vedly k co nejefektivnějšímu pozdějšímu výpočtu průsečíků (viz 3.). Nemusíte psát formální důkaz složitosti $O(\log N)$, ale pokuste se alespoň naznačit principy, které k ní vedou. Uveďte podmínky (charakter scény), které by mohly vylepšit nebo naopak pokazit Vaši konstrukci. Dala by se konstrukce paralelizovat?
3. Formulujte algoritmus hledání průsečíku jednoho paprsku se scénou (opatřenou Vaší akcelerační strukturou). Definujte přesně, za jakých podmínek dostaneme negativní výsledek, a jaké jsou podmínky ukončení prohledávání v případě, že průsečík existuje (nezapomeňte, zajímá nás jen nejbližší průsečík). Diskutujte možnost masového paralelismu při výpočtu mnoha paprsků s jednou scénou.

Nástin řešení Dané podmínky splňuje jakákoli urychlovací struktura založená na stromech, ať dělíme prostor (Quad-tree nebo KD-tree) či scénu (BVH = hierarchie obalových těles). Při vhodném algoritmu konstrukce urychlovacího stromu bude dosaženo logaritmické složitosti výpočtu nejbližšího průsečíku.

Na ukázku použijeme hierarchii obalových těles BVH, obaly budou osově orientované kvádry (AABB = Axis Aligned Bounding Boxes). To je volba jednoduchá na použití, ale vhodná jen pro statické tvary těles, při případné animaci se musí strom přepočítávat v každém snímku.

1. Vstupní scéna je neuspořádaná množina trojúhelníků (triangle soup), každý trojůhelník má své identifikační (pořadové) číslo id (implicitní údaj, nemusí se ukládat) a trojici vrcholů V_1, V_2 a V_3. Každý vrchol musí mít 3D souřadnice $[x, y, z]$, ale může obsahovat i další údaje pro rendering, například normálový vektor, texturovou souřadnici či barvu, ty se ovšem nebudou při výpočtu průsečíků uvažovat. Celý trojúhelník může mít odkaz na těleso, ze kterého pochází (implicitně, pomocí svého id), materiál, textura či barva by pak mohly být uložené u tělesa. Paprsek je jednoduše reprezentován svým počátkem P_0 a směrovým vektorem $\vec{p_1}$ (pozn. předpokládáme, že paprsek již byl předem transformován do souřadného systému scény). Průsečík P bude možné reprezentovat jediným reálným číslem t ($P = P_0 + t \cdot \vec{p_1}$).
2. BVH bude binární strom, kde ve všech uzlech budou obalové AABB kvádry a v listech navíc seznamy odkazů na trojúhelníky (jen seznamy číselných id). Tzv. Bucket tree umožňuje uložit do listů více trojúhelníků, aby se zbytečně nemusely dělit menší množiny trojúhelníků. Konstrukce stromu se bude implementovat metodou top-down, kdy se dělí počáteční celá množina na menší části. Po provedení několika dělení se může další výpočet provádět paralelně (rozděl a panuj). Oblíbená suboptimální heuristika SAH (Surface Area Heuristics) rozhoduje, jak přesně se v každém vnitřním uzlu rozdělí množina trojúhelníků na dvě menší tak, aby byl minimalizován očekávaný čas výpočtu průsečíků (pozn. zde mohou být uvedeny podmínky a náznak odvození SAH ... pravděpodobnostní vzorce). Celá konstrukce BVH stromu může být formulována tak, že povede k vyváženému stromu, což umožňuje jeho implicitní uložení (bez pointerů). Pro dobře strukturovanou scénu, která neobsahuje příliš extrémně protáhlých trojúhelníků (daly by se i uměle rozdělit...) bude dosaženo průměrné časové složitosti $O(\log N)$ výpočtu průsečíku paprsku se scénou, protože hloubka stromu je logaritmická a potenciál nutnosti procházet všechny větve je malý.
3. Pro daný paprsek postupujeme od kořene BVH stromu směrem do listů. Vždy zkontrolujeme, zda paprsek protíná obalový kvádr podstromu: když ne, nemusíme do té větve vůbec vstupovat. Pokud paprsek protne obaly obou potomků daného uzlu, musíme navštívit obě větve; jako první projdeme podstrom s bližším průsečíkem. Můžeme též použít algoritmus procházející do šířky ty větve, jejichž obaly byly protnuty. Prioritou jejich zpracování je opět pořadí průsečíků s jejich obaly (blíže protnuté obaly mají větší potenciál, že v nich bude nalezen nejbližší průsečík). V každém navštíveném listu se musí otestovat paprsek proti všem uloženým trojúhelníkům. Negativní výsledek nastane, jestli jsme zpracovali všechny protnuté obaly (= jejich podstromy) a paprsek neprotnul žádný z trojúhelníků. Pozitivní výsledek můžeme ohlásit tehdy, když paprsek protnul trojúhelník A a současně byly otestovány všechny ostatní trojúhelníky v daném listu (a byly dál než průsečík s A). Dále musí platit, že ve frontě dosud nezpracovaných vnitřních uzlů není žádný obal, jehož průsečík leží blíže než průsečík s A. Paralelismus se implementuje snadno, protože urychlovací BVH strom se po své konstrukci již nemění a může se předávat jednotlivým výpočetním jednotkám jako sdílená R/O data.

18 Osvětlení a stínování v počítačových hrách (specializace PGVVH-VPH)

Budeme se zabývat rolí osvětlení při zobrazování 3D scén v počítačových hrách.

1. Z hlediska realističnosti zobrazení 3D objektů je možné použití několika rozdílných přístupů, přitom všechny mohou mít své opodstatnění ve vývoji her. Seřaďte podle věrnosti zobrazení ("fotorealističnosti") následující technologie od méně věrných k těm nejvěrnějším: Phongovo stínování, konstantní stínování ("flat shading"), Ray-tracing, drátový model ("wireframe"), Gouraudovo stínování. Která z nich mají podporu v moderních GPU? Alespoň u dvou uveďte přibližný princip, jak pracují.
2. Vlastnosti povrchu (materiál) se dají v realtime grafice reprezentovat mnoha různými způsoby. Vyjmenujte co nejvíce metod, jak se pomocí dodatečných dat nebo sofistikovanějších algoritmů dá vylepšovat vzhled 3D objektů ve scéně.
3. Popište alespoň rámcově, jak se v realtime enginu mohou počítat tzv. "vržené stíny", tedy stíny, které vzniknou díky překážkám šíření světla ve 3D scéně. Vzpomenete si, jestli nám zde může pomoci technologie GPU? Zkuste popsat alespoň princip.

Nástin řešení 1. od nejméně realistických k nejrealističtějším

- drátový model ("wireframe")
- konstantní stínování ("flat shading")
- Gouraudovo stínování

- Phongovo stínování
- Ray-tracing

Konstantní stínování: model odrazu světla se spočítá ve vrcholech (Vertex shader) a výsledná barva se pošle dál do pipeline jako ("flat") (např. flat out vec4 VertexColor;). Barvu trojúhelníka určuje barva jeho posledního vrcholu (to je default, lze to změnit).

Goraudovo stínování: model odrazu světla se spočítá ve vrcholech (Vertex shader) a výsledná barva se pošle dál do pipeline, použije se default nastavení. GPU automaticky provede perspektivně korektní interplaci této veličiny.

Phongovo stínování: atribut normálového vektoru se musí propagovat až do Fragment shaderu (default je perspektivně korektní interpolace, což nám vyhovuje). Model odrazu světla se spočítá ve Fragment shaderu. Je třeba dát pozor na vstupní veličiny modelu odrazu světla, musí být všechny ve stejné souřadné soustavě (např. světový systém souřadnic) a vektory musí být normalizovány (rasterizer mění délku normálového vektoru!).

2. Základní sada atributů pro vzhled (appearance) jsou materiálové konstanty použitého modelu odrazu světla (např. pro Phonga je to k_A, k_D, k_S, h a základní barva povrchu C_D). Některé se obvykle předávají jako ("shader uniform"), některé jako atributy vrcholů (barva, normálový vektor). Nejsilnější prostředek je použití textury (nejčastěji barva, normála "normal map")...

Dále by se mohl zmínit "environment mapping" (globální HDR textura reprezentující okolí vykreslovaného objektu, poslouží pro dobrou aproximaci odlesku), "parallax mapping" – ten ovšem už předpokládá komplikovanější Fragment shader (musí se hledat přesné místo dopadu paprsku)...

3. "Shadow mapping" je hardwarově implementovaný na GPU, znamená to použití více průchodů scénou, každý pro jeden bodový/směrový zdroj světla. GPU umí porovnávat hloubku uloženou v textuře s aktuální vzdáleností fragmentu od kamery - podle toho se určí, zda je fragment ve stínu nebo ne.

"Volumetric shadows" je komplikovanější a věrnější metoda pro výpočet vržených stínů pracující správně i ve členitých scénách. Je potřeba hardwarově podporovaný "Stencil buffer" s operacemi inkrement/dekrement. Dále se scéna musí rozdělit na stěny přivrácené a odvrácené od zdroje světla (to znamená dva průchody pro každý zdroj na výpočet šablony + jeden průchod z pohledu kamery pro rendering). Více detailů viz literatura (např. slajdy k přednášce NPGR019).

19 Překlad adres (specializace PVS)

Předpokládejme hypotetický little-endian RISC procesor s paměťovou architekturou, kde virtuální i fyzický prostor používá 16-bitové adresování s klasickou load-store instrukční sadou.

Jediným zásadním omezením našeho hypotetického procesoru je, že obsahuje pouze load/store instrukce pro načtení/zápis přesně 16bitového slova, které musí být přirozeně zarovnané (tj. na sudých adresách).

Pro překlad adres se používá 1-úrovňové stránkování s 512 B stránkami. Pro ušetření místa se ve stránkovací tabulce ukládá pouze číslo fyzického rámce (obvykle označované jako PFN). Díky tomu se adresa pro překlad jedné stránky vejde přesně do jednoho bajtu, protože nejnižší bit (bit 0) je využit jako příznak platnosti záznamu (valid nebo též present bit). Stránkovací tabulka se tedy vejde do 128 B. Procesor (resp. MMU) vyžaduje, aby počátek stránkovací tabulky byl zarovnán právě na 128 B.

Napište kód funkce, která nastaví mapování ve stránkovací tabulce pro jednu konkrétní stránku.

- Začátek stránkovací tabulky je připraven v parametru `root` (jde o fyzickou adresu).
- Virtuální adresa (právě mapované) stránky je připravena v parametru `virt`.
- Fyzická adresa (právě mapovaného) rámce je připravena v parametru `phys`.

Přímý přístup k datům stránkovací tabulky je možný pouze pomocí speciálních funkcí, které zajistí interně překlad adres a také zabrání optimalizacím překladače (které by mohly vést k nekonzistentnímu stavu záznamů). Signatury těchto funkcí jsou uvedeny níže, zadávané (fyzické) adresy musí být přirozeně zarovnané (tj. na 16 bitů) podle omezení zmíněného výše (nejde tedy změnit jedinou položku, vždy dochází k atomické aktualizaci dvou položek zároveň).

```
uint16_t phys_read(uint16_t phys);
void phys_write(uint16_t phys, uint16_t value);
```

Pro zápis (pseudo)kódu zvolte vhodný C-like jazyk. Kód pište, prosím, jen tiskacím (klidně hůlkovým) písmem. Při kontrole vašeho řešení se budeme velmi zaměřovat na korektní výpočet offsetů a indexů, nezapomeňte si též zkontrolovat správné směry bitových posunů.

Můžete předpokládat, že vstup v proměnných `root`, `virt` a `phys` je korektní a není ho nutné ověřovat. Nezapomeňte, že `virt` a `phys` jsou adresy (nikoliv VPN a PFN).

Nástin řešení Stránkovací tabulka je tedy tvořena 64 dvojicemi překladových záznamů (každá dvojice má velikost 16 bitů a je nejmenší položkou, se kterou můžeme pracovat pomocí `phys_read` a `phys_write`).

```
// cislo/index stranky a ramce
vpn := virt >> 9
pfn := phys >> 9

// adresovat muzeme jen celou dvojici, takze musime ziskat adresu
// sude (spodni) polozky
adresa_polozky := root + (vpn & ~0x1)
existujici := phys_read(adresa_polozky)

je_sude := vpn & 1
// nastaveni valid bitu na novem zaznamu
nove_mapovani := (pfn << 1) | 1

// podle pozici ve dvojici vymazeme nahrazovanou cast
if je_sude:
    existujici := existujici & 0xff00
else
    existujici := existujici & 0x00ff
    nove_mapovani := nove_mapovani << 8
end

novy := existujici | mapovani
phys_write(adresa_polozky, novy)
```

20 Virtuální metody (specializace PVS)

1. Jaký je výstup následujícího programu v C++? Odpověď stručně (jednou nebo dvěma větami) odůvodněte.

```
class XA {
public:
  int f() { return 1; }
  virtual int g() { return 2; }
};

class XB : public XA {
public:
  int f() { return 3; }
  virtual int g() { return 4; }
};

int main() {
        XA* x = new XB();
        std::cout << x->f() << x->g();
}
```

2. Napište definice tříd `YA` a `YB` z následujícího fragmentu kódu tak, aby výsledný výstup byl shodný s výstupem první části, avšak bez použití klíčového slova `virtual`. Jinými slovy, implementujte mechanismus volání virtuálních metod vlastními

prostředky. Třídy `YA` a `YB` můžete doplnit libovolnými dalšími konstrukcemi (kromě `virtual`) nutnými pro implementaci zadání.

```
class YA {
  .... { return 1; }
  .... { return 2; }
  ....
};

class YB : public YA {
  .... { return 3; }
  .... { return 4; }
  ....
};

int main()
{
  YA* y = new YB();
  std::cout << y->f() << y->g();
}
```

Princip vašeho řešení stručně vysvětlete.

Kód pište čitelně, pokud byste ve vašem řešení měli větší než zanedbatelný počet škrtání, vsuvek, přepisů apod., přepište řešení do čitelné a přehledné podoby. Drobné syntaktické nepřesnosti nebudou považovány za chybu, výsledný kód však musí obsahovat všechny nutné součásti řešení a respektovat viditelnost identifikátorů a typovou kompatibilitu.

Nástin řešení Výstup má být `14`, metoda `f` je určena staticky (kompilačně) dle typu proměnné, metoda `g` je volána dynamicky dle skutečného objektu pomocí tabulky virtuálních metod.

V druhé části otázky je třeba implementovat mechanismus volání metody dle skutečného objektu nějakou variantou VMT. Samotná metoda `g` je potom jen proxy pro volání výkonné funkce přes ukazatel uložený při vytváření objektu. Jelikož jde o jedinou metodu, stačí pouze jeden ukazatel, tabulka není nutná. Implementace volání nevirtuálních metod je beze změn.

Možné řešení:

```
class YA {
protected:
  int gi() { return 2; }
  using y_method = int(YA::*)();
  y_method gm;
public:
  YA( y_method m = &gi) : gm(m) {}
  int f() { return 1; }
  int g() { return (this->*gm)(); }
};

class YB : public YA {
protected:
  int gi() { return 4; }
public:
  YB() : YA(static_cast<y_method>(&YB::gi)) {}
  int f() { return 3; }
};
```

21 Rezervace ubytování (specializace PVS)

Předpokládejme následující relační schéma **databáze rezervací ubytování**:

```
CREATE TABLE customer (
  email VARCHAR PRIMARY KEY,
  name VARCHAR NOT NULL,
  country CHAR(3)
);

CREATE TABLE room (
  building CHAR(1),
  number INT,
  type VARCHAR,
  PRIMARY KEY (building, number)
);

CREATE TABLE booking (
  id INT PRIMARY KEY,
  check_in DATE NOT NULL,
  check_out DATE NOT NULL,
  building CHAR(1),
  number INT,
  customer VARCHAR REFERENCES customer (email),
  FOREIGN KEY (building, number) REFERENCES room (building, number),
  CHECK (check_in < check_out)
);
```

Navrhněte SQL SELECT výrazy pro následující dotazy:

1. Pokoje s konfliktními rezervacemi aneb pokoje, pro které existují alespoň dvě časově se překrývající rezervace.
2. Jména zákazníků z Austrálie nebo Nového Zélandu (kódy `AUS` a `NZL`), kteří nikdy nerezervovali pokoj typu `double`.

Kód pište, prosím, jen tiskacím (klidně hůlkovým) písmem.

Nástin řešení

```
SELECT DISTINCT B1.building, B1.number
FROM booking AS B1 JOIN booking AS B2 ON
  (B1.id < B2.id) AND
  (B1.building = B2.building) AND (B1.number = B2.number) AND
  (B1.check_in < B2.check_out) AND (B1.check_out > B2.check_in);

SELECT name
FROM customer
WHERE (country IN ('AUS', 'NZL')) AND NOT EXISTS (
  SELECT *
  FROM room NATURAL JOIN booking
  WHERE (type = 'double') AND (booking.customer = customer.email)
);
```

22 Analýza uživatelských požadavků a návrh architektury (specializace PVS)

Pracujete na nové aplikaci pro rezervaci městských sportovišť – tělocvičny, tenisové kurty, bazény. Ze schůzky se stakeholdery jste si přinesli následující poznámku: „Obyvatelé města chtějí mít možnost snadno najít volné sportoviště a rovnou si jej zarezervovat. Rezervace by měla být přehledná a rychlá, protože uživatelé často rezervují na poslední chvíli. Také se může stát, že nestíhají, a tak potřebují rezervaci zrušit. Do budoucna chceme, aby byla aplikace snadno rozšiřitelná o další typy sportovišť."

1. Zahrnuje poznámka pouze funkční požadavky? Odpovězte ano/ne a poté vysvětlete.
2. Identifikujte v poznámce alespoň 3 různé funkční požadavky a vyjádřete je jako user stories. (Pokud potřebujete doplňující informace nad rámec poznámky, domyslete si je.)
3. Navrhněte první draft architektury aplikace vycházející z identifikovaných požadavků jako *modularizovaný monolit* (angl. *Modularized Monolith*).

4. Stručně vysvětlete, jak jste při návrhu uplatnili princip *vysoké koheze* (angl. *high cohesion*).

Nástin řešení

1. Poznámka obsahuje funkční i kvalitativní (ne-funkční) požadavky. Funkční např. vyhledání sportoviště, rezervace sportoviště. Ne-funkční např. přehlednost, rychlost, snadná rozšiřitelnost.
2. Požadavky by měly být zapsány ve formátu user stories "Who, What, Why", např. „Jako uživatel chci mít možnost vyhledat volný tenisový kurt, abych si mohl okamžitě rezervovat hru s kamarádem." „Jako uživatel chci mít možnost potvrdit si rezervaci tenisového kurtu s kamarádem, protože nemá smysl dělat rezervaci, pokud on nemůže." (Požadavky mohou mluvit o konkrétních typech sportovišť i obecně o sportovištích - obojí je správně.) User story by měla být ve formátu Who, What, Why. Who a What by mělo cca sedět na zadání, Why tam být musí a je potřeba si nějaké věcně rozumné vymyslet, v zadání nejsou - jde o to, vymyslet něco, co věcně zdůvodňuje ten požadavek.
3. *Modularizovaný monolit* má být rozdělen do několika modulů s jasně vymezenými věcnými (doménovými) odpovědnostmi. Typické věcné moduly:
   - **Sport Facility Catalog Management** - správa seznamu sportovišť a jejich dostupnosti.
   - **Sport Facility Search** - vyhledávání sportovišť, procházení jejich detailů.
   - **Reservation Management** - vyhledávání volných slotů v rámci sportoviště, vytváření, potvrzení a rušení rezervací.
   - **User Management** – evidence uživatelů a jejich účtů, autentizace a autorizace.

   Dále je možné členit moduly po technické stránce (presentation, business/application, database), ale neměly by být primárně členěny přes technický pohled bez věcného dělení. Moduly jsou součástí jedné aplikace (monolit) a jsou nasazeny jako jeden celek, ale jsou navrženy tak, aby jejich rozhraní byla jasně definovaná a vnitřní implementace zapouzdřená.
4. Princip *vysoké koheze* je uplatněn tak, že každý modul obsahuje funkce a datové struktury, které spolu úzce souvisejí a řeší jednu oblast odpovědnosti. Například modul **Reservation Management** řeší pouze práci s rezervacemi a neobsahuje logiku týkající se správy uživatelů nebo katalogu sportovišť. U modularizovaného monolitu by koheze měla být vysvětlena především věcnými souvislostmi, ale jako částečně správné je i vysvětlení přes technické souvislosti (např. dám dohromady prezentační věci a vedle aplikační logiku) – částečně správné to je proto, že samo o sobě je to správně, ale v kontextu modularizovaného monolitu cílíme na to věcné – doménové členění logiky.

23 Adresace (specializace SP)

Uvažujte následující program:

```
1  int data [1024];
2
3  int summarize () {
4      int sum = 0;
5      for (int i = 0 ; i < 1024 ; i ++) {
6          sum += data [i];
7      }
8      return sum;
9  }
```

Tento program jsme přeložili pro procesory Intel x86-64 a RISC-V rv32i.

Intel x86-64:

```
0000: 488d0519000000 lea 0x19(%rip),%rax
0007: 31d2           xor %edx,%edx
0009: 488d8800100000 lea 0x1000(%rax),%rcx
0010: 0310           add (%rax),%edx
0012: 4883c004       add $0x4,%rax
0016: 4839c8         cmp %rcx,%rax
0019: 75f5           jne 0x10
001b: 89d0           mov %edx,%eax
001d: c3             ret
```

RISC-V rv32i:

```
0000: 000007b7 lui  a5,0x0
0004: 03078793 addi a5,a5,0x30
0008: 000016b7 lui  a3,0x1
000c: 00d786b3 add  a3,a5,a3
0010: 00000513 li   a0,0
0014: 0007a703 lw   a4,0(a5)
0018: 00478793 addi a5,a5,4
001c: 00e50533 add  a0,a0,a4
0020: fed79ae3 bne  a5,a3,0x14
0024: 00008067 ret
```

1. Stručně vysvětlete, co to je "pozičně nezávislý kód" (position-independent code, PIC).
2. V obou verzích programu určete, ve kterém registru je uchovávána reference do pole během cyklu.
3. Pro obě verze programu rozhodněte, zda je přeložená podoba pozičně závislá nebo pozičně nezávislá.

Odpovědi zdůvodněte s odkazem na konkrétní místa obou programů, podle kterých jste se rozhodovali.

Poznámky k výpisu pro procesor Intel x86-64. Operandy instrukcí jsou uváděny v pořadí `src,dst`, názvy registrů používají prefix `%`, adresový výraz `offset(%reg)` sečte offset s obsahem registru `reg`. Instrukce `lea` zapíše adresu `src` do `dst`, instrukce `mov` zapíše obsah `src` do `dst`, instrukce `add` přičte obsah `src` k `dst`, instrukce `xor` analogicky, instrukce `jne` skočí, pokud obsahy `src` a `dst` předchozí instrukce `cmp` nebyly stejné, adresa skoku je zakódována relativně vůči registru `rip` (program counter).

Poznámky k výpisu pro procesor RISC-V rv32i. Operandy instrukcí jsou uváděny v pořadí `dst,src` nebo `dst,src1,src2`, u skoku `src1,src2,addr`, názvy registrů používají prefix `a`, adresový výraz `offset(reg)` sečte offset s obsahem registru `reg`. Instrukce `lui` nastaví obsah `dst` na `src << 12`, instrukce `li` nastaví obsah `dst` na `src`, instrukce `add(i)` nastaví obsah `dst` na `src1 + src2`, instrukce `lw` nastaví obsah `dst` na obsah adresy `src`, instrukce `bne` skočí, pokud obsahy `src1` a `src2` nejsou stejné, adresa skoku `addr` je zakódována relativně vůči registru `pc` (program counter).

Nástin řešení

1. Viz `https://en.wikipedia.org/wiki/Position-independent_code`.
2. Pro Intel x86-64 je to `rax`, na adrese 0x10 je vidět čtení `data[i]`, inicializace na 0x0, inkrement na 0x12, atd. Pro RISC-V rv32i je to `a5`, na adrese 0x14 je vidět čtení `data[i]`, inicializace na 0x0 a 0x4, inkrement na 0x18, atd.
3. Ve verzi pro Intel x86-64 je reference do pole inicializována relativně k adrese program counteru, tedy je pozičně nezávislá. Ve verzi pro RISC-V rv32i je reference do pole inicializována absolutní konstantou, tedy je pozičně závislá. V obou verzích je jediný přímý skok, který kóduje cílovou adresu relativně, tedy je pozičně nezávislý. Jiná relevantní místa v kódu nejsou. Verze pro Intel x86-64 je tedy position independent, verze pro RISC-V rv32i je tedy position dependent.

24 Mapované soubory (specializace SP)

Toto je rozhraní funkcí `mmap` a `munmap` poskytovaných operačním systémem Linux pro mapování souborů:

```
void *mmap (void *addr, size_t length, int prot, int flags, int fd, off_t offset);
int munmap (void *addr, size_t length);
```

Uvažujte následující příklad použití tohoto rozhraní. Předpokládejte, že konstanty `FILENAME` a `SIZE` obsahují jméno a velikost běžně čitelného textového souboru.

```
1 int file = open (FILENAME, O_RDONLY);
2 char *data = (char *) mmap (NULL, SIZE, PROT_READ, MAP_SHARED, file, 0);
3 for (int i = 0 ; i < SIZE ; i ++) {
4     putchar (data [i]);
5 }
6 munmap (data, SIZE);
7 close (file);
```

1. Odhadněte význam konkrétních hodnot argumentů volání `mmap` použitých v příkladu.
2. Bude v příkladu docházet k výpadkům datových stránek, a pokud ano, kdy a proč ?
3. Bude adresa v proměnné `data` zarovnaná, a pokud ano, jak a proč ?

4. Pokud se tentýž příklad spustí současně ve více procesech, budou adresy v proměnných `data` souviset, jak a proč ?
5. Načrtněte alternativní příklad, který místo volání `mmap` načte obsah souboru pomocí volání `read`. Bude se tento příklad lišit v efektivitě přístupu k obsahu souboru, jak a proč ?

Jako pomůcka ještě rozhraní funkce `read`:

```
ssize_t read (int fd, void *buf, size_t count);
```

Nástin řešení

1. Viz `man mmap`.
2. Obecně ano, pokaždé, když čtení z paměti `data [i]` poprvé přistoupí na novou stránku s obsahem souboru. Teoreticky může být výpadků méně, konkrétní implementace může mapovat více než jednu stránku při jednom výpadku, případně mapovat stránky přímo uvnitř `mmap`. Další místa možných datových výpadků, která ale nejsou relevantní z pohledu otázky, jsou při přístupu k `FILENAME`, při používání zásobníku, uvnitř implementace vstupních a výstupních operací, při lazy binding symbolů a podobně.
3. Ano, na velikost stránky, protože mapování stránek funguje právě jen po celých stránkách.
4. Adresy nikterak souviset nebudou, každý proces bude mít vlastní adresový prostor, při absenci ASLR je ale možné, že adresy budou vybírány deterministicky a tedy budou ve více procesech (shodou okolností) stejné.
5. Nahrazení `mmap` pomocí `read` vyžaduje nějakým způsobem alokovat buffer. Implementace bude nutně méně efektivní v tom, že obsah souboru se bude muset do tohoto bufferu kopírovat.

25 Immutable typy v OOP (specializace SP)

1. Vysvětlete, co jsou to *immutable* typy v objektově orientovaném programování a pro jaké druhy objektů se hodí. Uveďte alespoň dvě významné výhody a alespoň jednu nevýhodu, které souvisejí s použitím *immutable* typů a objektů při návrhu a vývoji software.
2. V níže uvedeném kódu bylo cílem vytvořit immutable třídu `Configuration`, ale návrh třídy je špatně a zamýšlenému účelu neodpovídá.
   (a) Identifikujte konkrétní místa v kódu, kvůli kterým není možné třídu `Configuration` považovat za immutable, a vysvětlete, proč je kód v těchto místech problematický.
   (b) Navrhněte dvě možná řešení, jak návrh třídy `Configuration` opravit, a porovnejte výhody a nevýhody obou řešení (např. s ohledem na výkon a nároky na kód používající opravenou třídu).

Následující kód je psán v jazyce Java. V jazyce C# by vypadal obdobně, s tím rozdílem, že třída by byla definována jako `public sealed class` a instanční proměnné jako `private readonly`. Typ `List` představuje rozhraní a v C# by se jednalo o `IList`. Třída `String` je v obou prostředích immutable.

```
1  public final class Configuration {
2      private final int workerCount;
3      private final List<String> serviceEndpoints;
4
5      public Configuration(int workerCount, List<String> endpoints) {
6          this.workerCount = workerCount;
7          this.serviceEndpoints = endpoints;
8      }
9
10     public int workerCount() {
11         return this.workerCount;
12     }
13
14     public List<String> endpoints() {
15         return this.serviceEndpoints;
16     }
17 }
```

V odpovědi použijte libovolný z jazyků Java, C#, nebo C++. Detaily syntaxe nejsou podstatné, pokud nezpůsobí nejednoznačnost, která by zásadně ovlivnila význam kódu v řešení.

Nástin řešení

1. Správná odpověd by měla uvést, že stav immutable objektu nelze po jeho vytvoření měnit, tj. hodnoty všech jeho atributů jsou typicky nastaveny v konstruktoru a žádné metody by neměly umožnit změnu stavu objektu. Jsou vhodné zejména pro reprezentaci hodnotových typů a jako klíče asociativních datových strukturách.

   Mezi výhody patří jednodušší rozhraní, možnost objekty předávat metodám s jistotou, že nedojde k jejich změně, možnost sdílení mezi vlákny bez nutnosti synchronizace, představují ideální klíče v asociativních datových strukturách, instance je možné cachovat.

   Za použití se „platí“ potenciálně vyšší běhovou režií v situacích, kdy je potřeba měnit stav programu reprezentovaný immutable objekty, což vede na vyšší spotřebu paměti a potenciálně vyššímu tlaku na garbage collector v managed běhových prostředích. Vytváření složitějších objektů s mnoha atributy může vyžadovat větší množství kódu nebo složitější návrhové vzory.

2. (a) Hlavní problém je v metodě `endpoints()`, která vrací interní referenci na (modifikovatelný) kontejner. Volající může skrz tuto referenci kontejner modifikovat a měnit tak stav instance `Configuration`. Druhý problém spočívá v tom, že konstruktor ukládá referenci na kontejner předaný volajícím. Pokud si volající referenci ponechá, může skrz ni opět modifikovat stav objektu.

   (b) Konstruktor by měl vytvořit defenzivní kopii předaného seznamu. Metoda `endpoints()` by měla vracet defenzivní kopii (což s sebou nese režii v důsledku kopírování seznamu při každém zavolání, ale na druhou stranu metoda vrací volajícímu seznam, se kterým si může dělat co chce), nebo vracet nemodifikovatelný pohled (read-only view) na interní seznam (to snižuje režii, ale při pokusu o modifikaci pohledu dojde k výjimce—volající tedy musí zajistit, že seznam např. nepředá nikomu, kdo by se mohl pokoušet jej modifikovat).

26 Překlad adres v IP sítích (specializace SP)

1. Definujte, co je Network Address Translation (NAT) v IPv4 sítích. Vysvětlete rozdíly mezi statickým NAT, dynamickým NAT a Port Address Translation (PAT), a k čemu se používají. Jaký vliv má použití NAT na konektivitu v síti a komunikační protokoly?

2. Předpokládejte, že firma má vnitřní síť s adresním prostorem 192.168.1.0/24, která je připojena k Internetu přes NAT bránu s veřejnou adresou 203.0.113.10. Všichni uživatelé na počítačích v interní síti se dostanou na Internet, tj. mohou se připojit k libovolnému počítači ve vnější síti.

   (a) Přepokládejte, že uzel ve vnitřní síti s adresou 192.168.1.20 komunikuje pomocí protokolu TCP s webovým serverem ve vnější síti na adrese 93.184.216.34 (port 80). Popište, jaké změny, a na základě jakých informací bude router dělat v paketech od uzlu směrem k serveru a naopak.

   (b) Firma ve vnitřní síti zprovoznila ještě vlastní webový server (TCP port 80) s interní adresou 192.168.1.100. Klienti z vnitřní sítě se na něj mohou připojit, ale spojení z vnější sítě (na veřejnou adresu brány) nefunguje.

   Vysvětlete, proč v současné konfiguraci TCP pakety od klienta z vnější sítě, směrované na port 80 na adrese 203.0.113.10, nedorazí na interní webový server, a jaké z omezení NAT tato situace ilustruje. Navrhněte úpravu konfigurace brány, která klientům z vnější sítě umožní se na interní webový server připojit.

Nástin řešení

1. V odpovědi by mělo být uvedeno, že v rámci NAT router překládá privátní IPv4 adresy na veřejné IPv4 adresy na hranici sítě. V případě statického NAT se jedná o trvalé mapovaní adres 1:1, v případě dynamického NAT se jedná od mapování z privátních adres na nějakou množinu veřejných adres. Pokud více uzlů na vnitřní síti sdílí jednu veřejnou adresu, přichází do hry PAT (Port Address Translation), kdy jsou součástí mapování také čísla portů.

   Výhodou je úspora IPv4 adres a částečné skrytí vnitřní struktury sítě, nevýhodou je ztráta obousměrné (end-to-end) konektivity a z toho plynoucí nutnost speciálních mechanizmů pro podporu příchozích spojení, a problémy s protokoly, které vkládají IP adresy a porty do dat.

(a) Předpokládáme, že uzel ve vnitřní síti komunikuje např. z portu 12345. Původní paket od klienta má tedy zdrojovou adresu 192.168.1.20:12345 a cílovou adresu 93.184.216.34:80 a NAT brána před odesláním paketu směrem k serveru přepíše zdrojovou adresu na 203.0.113.10:55001 (číslo portu je alokováno před odesláním prvního paketu). Při příjmu paketu od serveru s cílovou adresou 203.0.113.10:55001 změní NAT brána cílovou adresu na adresu 192.168.1.20:12345. Pro překlad musí NAT brána udržovat překladovou tabulku, jejíž záznamy zachycují mapování vnitřní adresou uzlu, vnější adresou brány a vnější adresou serveru (včetně portů).

(b) NAT brána automaticky vytváří překladové záznamy pouze pro odchozí spojení. Pokud přijme paket z vnější sítě, pro který neexistuje záznam v překladové tabulce, tak neví, zda a kterému uzlu na vnitřní síti jej má předat. Zachází s ním tedy jako s normálním paketem a pokud brána neposlouchá na TCP portu 80, tak spojení odmítne. Situace ilustruje narušení oboustranné konektivity v důsledku NAT, kdy brána blokuje příchozí spojení, pokud nejsou explicitně nakonfigurována.

Úprava konfigurace musí zavést pravidlo, které bráně umožní takové pakety předat uzlu na vnitřní síti. Je tedy potřeba nakonfigurovat statické mapování/port forwarding z 203.0.113.10:80 na 192.168.1.100:80.

27 Jednotahové hry (specializace UI-SU, UI-ZPJ, UI-ROB)

Uvažte jednotahové hry, kdy hráči volí tahy naráz a jejich odměna je přiřazena na základě kombinace zvolených tahů.

(A) Pro jednotahové hry dvou hráčů napište definice pojmů *dominantní strategie*, *Nashovo ekvilibrium* a *Pareto optimální výsledek*.

(B) V následujících maticích každé políčko obsahuje dvě čísla, která udávají zisk hráčů A a B po vykonání akcí *Right* nebo *Left*. Hráči zisk maximalizují. Pro tyto tři jednotahové hry napište všechny dominantní strategie, Nashova ekvilibria a Pareto optimální výsledky.

(C) Pomocí těchto příkladů vysvětlete rozdíly mezi všemi dvojicemi pojmů definovaných v části (A).

		B	
		Left	Right
A	Left	A = 10, B = 5	A = 7, B = 8
	Right	A = 0, B = 6	A = 15, B = 7

Tabulka 1: Hra (1)

		B	
		Left	Right
A	Left	A = 100, B = 100	A = 0, B = 0
	Right	A = 0, B = 0	A = 50, B = 50

Tabulka 2: Hra (2)

		B	
		Left	Right
A	Left	A = 20, B = 20	A = 0, B = 40
	Right	A = 40, B = 0	A = 10, B = 10

Tabulka 3: Hra (3)

28 Evaluace binárního klasifikátoru (specializace UI-SU, UI-ZPJ)

Představte si, že vaším úkolem je vybrat nástroj pro analýzu počítačových logů a vyhledávání anomálií v nich (ty mohou být způsobené například selháním hardware, kybernetickým útokem, apod.). Máte možnost si vybrat ze dvou nástrojů. Z

dostupných marketingových materiálů jste se o prvním nástroji dozvěděli, že má úspěšnost (accuracy) 99,5 % a umí správně odhalit 90 % anomálií. U druhého nástroje se uvádí, že odhalí 81 % anomálií a jen 10 % nahlášených anomálií jsou falešně pozitivní.

Předpokládejte, že oba nástroje byly vyhodnoceny na stejných datech obsahujících 10 000 instancí, z nichž pouze 1 % jsou anomálie.

1. Spočítejte chybovou matici (matice konfuze, confusion matrix) obou nástrojů a tyto metriky pro binární klasifikaci: úspěšnost (accuracy), přesnost (precision), úplnost (recall), F1-skóre.
2. Popište výhody a nevýhody jednotlivých nástrojů při praktickém nasazení. Který z nich byste použili? Zdůvodněte. (*Zde není nutně jediná správná odpověď.*)

Nástin řešení O datech víme, že obsahují celkem 9 900 negativních instancí (nejsou anomálie) a 100 pozitivních instancí.

1. První nástroj správně detekuje 90 ze 100 anomálií. Tedy TP = 90, z toho plyne, že FN = 10. Tento nástroj také správně určí 9 950 z 10 000 instancí. Tedy musí správně určit 9 950 - TP = 9860 negativních instancí. Tedy TN = 9 860 a FP = 40. Recall je 0.9 (ze zadání), precision je TP/(TP+FP) = 90/130 (cca 0.69). F1-skóre je 0.78, accuracy (ze zadání) 0.995.

   Druhý nástroj správně detekuje 80 ze 100 anomálií. Tedy TP = 81, a FN = 19. Ze zadání jen 10 % nahlášených jsou FP, tedy precision je 0.9 = TP/(TP+FN) a tedy FN = 81/9 = 9. Zbytek instancí (9 891) je TN. Recall je tedy 0.81 (ze zadání), precision je 0.9 (ze zadání), F1-skóre je 0.85, accuracy 0.997.
2. Druhý nástroj je lepší ve všech metrikách (kromě recall) než nástroj první. Jeho výhodou může být to, že hlásí mnohem méně falešně pozitivních výsledků, nezaměstnává tedy správce systému zbytečnými hlášeními. Na druhou stranu, pokud jsou anomálie kritické, může být lepší použít první nástroj, který jich detekuje více za cenu toho, že okolo 30 % detekcí jsou falešně pozitivní.

29 Predikce pomocí tabulárních dat (specialization UI-SU, UI-ZPJ)

Máte k dispozici následující zemědělský dataset pro predikci výnosu plodin a na základě tohoto datasetu chcete natrénovat model strojového učení.

pH půdy	Srážky (mm)	Průměr. teplota (°C)	Typ hnojiva	Odrůda osiva	Velikost farmy (ha)	Zavlažování	Nadm. výška (m)	Výnos (t/ha)
6,8	450	22	Organické	Hybrid-A	48,6	Kapkové	150	4,2
7,2	380	25	Chemické	Standard	34,4	Postřikovač	200	3,8
6,5	520	18	Organické	Odolná	121,4	Dešťová	850	3,2
7,0	320	28	Bio-hnojivo	Hybrid-B	60,7	Kapkové	75	5,1
6,9	680	20	Chemické	Vysoký výnos	25,9	Záplavové	300	6,8

Na základě ukázkového datasetu odpovězte na následující otázky:

1. Jaký typ úlohy strojového učení to je?
2. Navrhněte vhodný model strojového učení pro tuto úlohu a stručně vysvětlete, proč by byl vhodný.
3. Pokud to navržený model vyžaduje, popište kroky předpracování, které byste na tento dataset aplikovali před trénováním modelu. Jak byste zacházeli s různými typy příznaků (features)?
4. Jaké evaluační metriky byste použili k posouzení výkonnosti modelu?

Nástin řešení Jedná se o *regresi* a *učení s učitelem*, protože cílová proměnná (Výnos) je spojitá číselná hodnota a máme k dispozici trénovací data s cílovými hodnotami.

Nejlepší volba modelu: *Random Forests* nebo *Gradient Boosting (XGBoost/LightGBM)*, vhodné pro tabulární data. *Lineární regrese* nebo *MLP* by také mohly fungovat s pečlivým předpracováním příznaků.

Kroky předpracování:

- *Kategorické příznaky:* Aplikovat one-hot enkódování enkódování pro typ hnojiva, odrůdu osiva a metodu zavlažování.
- *Číselné příznaky:* Zkontrolovat odlehlé hodnoty a aplikovat standardizaci/normalizaci (např. StandardScaler), pokud používáme algoritmy citlivé na škálu. V datasetu: pH půdy, srážky, teplota, velikost farmy, nadmořská výška.
- (volitelně) *Chybějící hodnoty:* Dodat chybějící hodnoty (průměr/medián pro číselné, modus pro kategorické).
- (volitelně) *Feature engineering:* Zvážit vytvoření kombinovaných příznaků (např. srážky vs. teplota) nebo polynomiálních příznaků, pokud doménové znalosti naznačují nelineární vztahy.

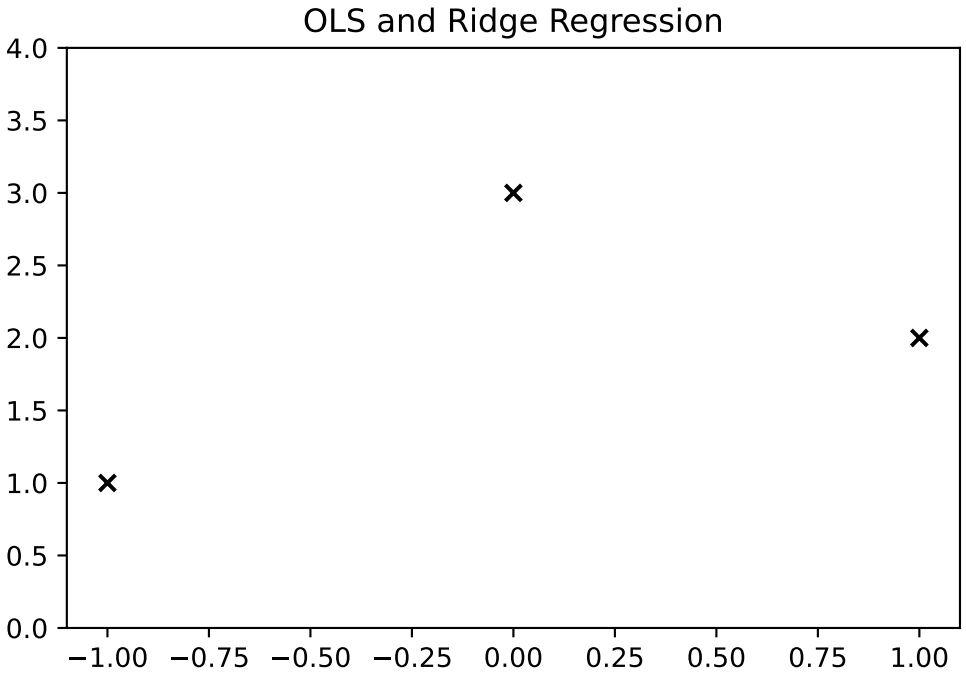
Evaluační metriky: Mean Absolute Error, (Root) Mean Square Error,

30 Lineární regrese, L2 regularizace (specializace UI-SU)

Mějme data s nezávislou proměnnou x a predikovanou proměnnou y zadaná v tabulce níže. Budeme pro ně hledat lineární model bez regularizace a s L2 regularizací.

x	y
-1	1
0	3
1	2

1. Popište lineární model a metodu nejmenších čtverců. Jakým způsobem se odhadují koeficienty modelu?
2. Uvažujte model bez regularizace (OLS). Vypočítejte koeficienty modelu pro data zadaná v tabulce a do grafu zaneste predikované hodnoty pro $x \in \langle -1, 1 \rangle$.
3. Popište optimalizovanou funkci s $L2$ regularizací. Jak se změní koeficienty při odhadu modelu s L2 regularizací? Stačí určit směr změny koeficientu u x oproti OLS a přibližně zanést do grafu predikované hodnoty pro $x \in \langle -1, 1 \rangle$.



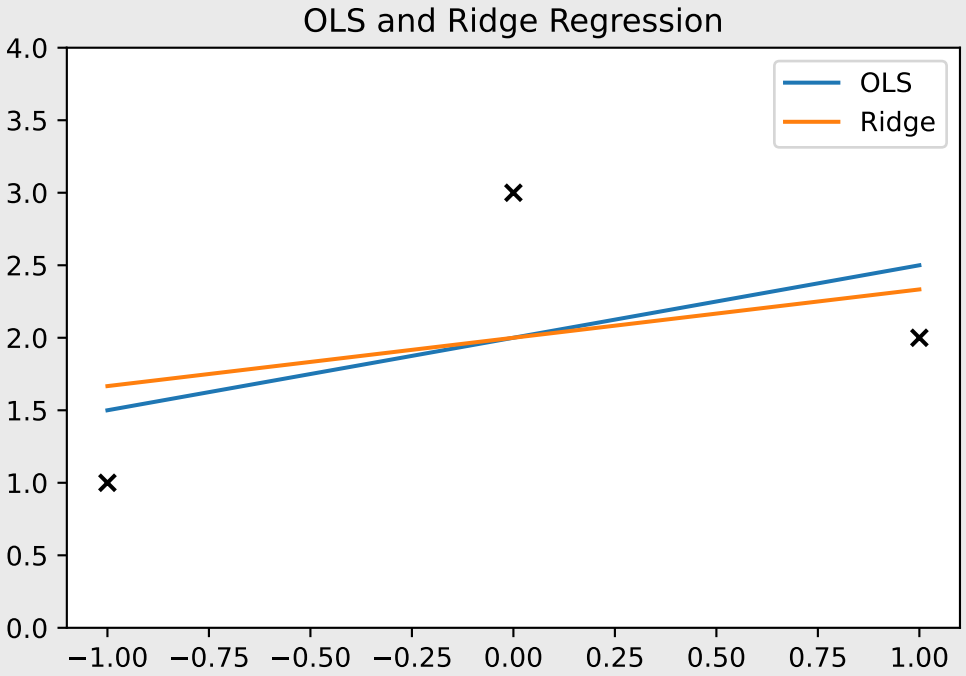
Nástin řešení

1. Lineární model s jednou nezávislou proměnnou odhaduje dva parametry β_0, β_1 modelu $f(x) = \beta_0 + \beta_1 \cdot x$, minimalizujeme funkci $\sum_{i=1,2,3}(f(x_i) - y_i)^2$. Pro odhad parametrů k tabulce přidáme sloupec jedniček a řešíme $\beta = (X^T X)^{-1} X^T y$.

2. Díky jediné nezávislé proměnné a centrovaným datům lze i snadněji, zde uvádíme obecné řešení:

$$\begin{aligned}\beta &= (X^T X)^{-1} X^T y \\ &= \left(\begin{bmatrix}1,1,1\\-1,0,1\end{bmatrix}\cdot\begin{bmatrix}1,-1\\1,0\\1,1\end{bmatrix}\right)^{-1}\cdot\begin{bmatrix}1,1,1\\-1,0,1\end{bmatrix}\cdot\begin{bmatrix}1\\3\\2\end{bmatrix} = \left(\begin{bmatrix}3,0\\0,2\end{bmatrix}\right)^{-1}\cdot\begin{bmatrix}6\\1\end{bmatrix} = \begin{bmatrix}\frac{6}{3}\\\frac{1}{2}\end{bmatrix} = \begin{bmatrix}2\\\frac{1}{2}\end{bmatrix}.\end{aligned}$$

3. S L2 regularizací přidáváme do cílové funkce součet kvadrátů β_j bez β_0, tj. minimalizujeme $\sum_{i=1,2,3}(f(x_i)-y_i)^2+\alpha\beta_1^2$, $\alpha > 0$. Koeficient β_1 bude menší než v předchozím případě, ale zachová si znaménko. Intercept β_0 není penalizován, v tomto případě se nezmění, obecně se změnit může. (V grafu je R2 regularizace nazvaná Ridge).



31 TF-IDF (specializace UI-ZPJ)

1. Popište, jak se používá metoda pro vyhledávání informací TF-IDF. Uveďte vzorec pro výpočet TF-IDF, popište jeho součásti a intuici, proč tato metoda funguje.
2. Uveďte, jaká data jsou k výpočtu potřeba, a proč a jakým způsobem mají být předzpracována.
3. Popište, jak se TF-IDF používá pro vyhledávání dokumentů podle dotazu (query).

Nástin řešení

1. $\text{TF-IDF}(t,d) = \text{TF}(t,d) \times \text{IDF}(t)$, kde $\text{TF}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$ a $\text{IDF}(t) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$.
2. Kombinuje lokální důležitost termu v dokumentu (TF) s globální vzácností termu v kolekci (IDF). Vzácné termy mají vyšší diskriminační sílu než časté. *Dodatečná pozorování:* Logaritmus v IDF koriguje Zipfův zákon – neutralizuje velmi časté slova. Součet TF-IDF dává mutual information mezi termy v dokumentu a kolekcí dokumentů.
3. Potřebujeme: frekvence termů v dokumentech a celkový počet dokumentů. Předzpracování: tokenizace, normalizace (např. malá/velká písmena), odstranění stopslov.
4. Pro vyhledávání: dokumenty i dotazy reprezentovány jako TF-IDF vektory, podobnost měřena kosinovou vzdáleností. Vrátíme dokumenty s nejvyšší podobností k dotazu.

32 Pohyb všesměrového podvozku (specializace UI-ROB)

Tříkolový podvozek robota je vybavený "omni" koly. Podvozek je symetrický, pozice kol $i \in \{1, 2, 3\}$ jsou charakterizovány vzdáleností r od referenčního bodu robota a úhly θ_i vůči ose x, orientace kol příslušnými jednotkovými vektory $\hat{u_i}$ kolmými na posunutí kola vůči referenčnímu bodu. Uvažujte rovnoměrný pohyb robota v rovině, zadaný translací $\vec{v_T}$ a rotační rychlostí ω referenčního bodu robota.

Vysvětlete princip pohybu. Nakreslete schéma robota s vyznačením jeho výše uvedené konfigurace a zadaného pohybu včetně vlivu tohoto pohybu na body určující polohu jednotlivých kol. Odvoďte vzorec pro výpočet rychlostí jednotlivých kol ze zadaného pohybu robota, tj. zjištění S_i, pokud známe $\vec{v_T}$ a ω, a vzorce pro inverzní výpočet, tj. zjištění $\vec{v_T}$ a ω, pokud známe S_1, S_2, S_3 (např. z odometrie).

Nástin řešení Princip pohybu: každé kolo přispívá pohybu celého robota ve směru svého odvalování, tj. ve směru jednotkového vektoru otáčení kola. Složením těchto vektorů od všech kol dostaneme okamžitý směr pohybu referenčního bodu. Pro zadaný rovnoměrný pohyb se bude jednat buď o otáčení na místě (translační rychlost nulová, rotační nenulová), po přímce (translace nenulová, rotace nulová) nebo po kružnici (obě rychlosti nenulové), kde výše zmíněný směr pohybu referenčního bodu je tečný k této kružnici. Pohyb každého jednotlivého kola je složen ze dvou složek, jedna je dána vlastním otáčením kola, druhá (kolmá k otáčení) je vynucena otáčením ostatních kol, přičemž díky použití omni kol tato kolmá složka otáčení tohoto konkrétního kola neovlivní, pouze umožní jeho pohyb v tomto směru.

Vzorce pro pohyb a postup odvození: Pohyb $\vec{w_i}$ kola i je dán posunutím od referenčního bodu a zadanou translací a rotací. K výpočtu rychlosti kola S_i tento vektor promítneme na jednotkový vektor daného kola $\hat{u_i}$:

$$\vec{w_i} = \vec{v_T} + \vec{\omega} \times \vec{r_i} \tag{1}$$

$$S_i = \vec{w_i} \cdot \hat{u_i} \tag{2}$$

$$= v_{T_x} cos(\varphi_i) + v_{T_y} sin(\varphi_i) + \omega r \tag{3}$$

kde vektor $\vec{\omega}$ určuje rotaci, takže jeho směr je kolmý na rovinu pohybu a velikost je dána rychlostí ω; vektor $\vec{r_i}$ určuje polohu kola i vůči referenčnímu bodu, jednotkový vektor $\hat{u_i}$ pohybu kola i je dán úhlem φ_i natočení kola i vůči ose x.

Vzorce pro inverzi získáme tak, že sestavíme soustavu rovnic pro výpočet rychlostí všech tří kol S_i v závislosti na zadaném pohybu $\vec{v_T}$ a ω dle konfigurace podvozku robota, danou umístěním a orientací kol $\vec{r_i}$, a tuto soustavu vyřešíme pro $\vec{v_T}$ a ω. Vzhledem k tomu, že se jedná o soustavu tří rovnic pro tři neznámé a rovnice nejsou závislé (dáno zadanou konfigurací robota), řešení bude existovat právě jedno.

Je zadán symetrický podvozek, bez újmy na obecnosti zvolíme konfiguraci, kde úhly θ_i jsou postupně 0°, 120°, 240°(resp. 0, $\frac{2}{3}\pi$, $\frac{4}{3}\pi$) a tedy směry jednotkových vektorů kol φ_i 90°, 210°, 330°($\frac{\pi}{2}$, $\frac{7}{6}\pi$,$\frac{11}{6}\pi$). Dosazením konstant daných konfigurací robota do rovnice 3 dostaneme

$$S_1 = v_{T_y} + \omega r \tag{4}$$

$$S_2 = -\frac{\sqrt{3}}{2} v_{T_x} - \frac{1}{2} v_{T_y} + \omega r \tag{5}$$

$$S_3 = \frac{\sqrt{3}}{2} v_{T_x} - \frac{1}{2} v_{T_y} + \omega r \tag{6}$$

Tuto soustavu vyřešíme pro v_{T_x}, v_{T_y}, ω a dostaneme

$$v_{T_x} = -\frac{\sqrt{3}}{2} S_2 + \frac{\sqrt{3}}{2} S_3 \tag{7}$$

$$v_{T_y} = \frac{2}{3} S_1 - \frac{1}{3} S_2 - \frac{1}{3} S_3 \tag{8}$$

$$\omega = \frac{1}{3r}(S_1 + S_2 + S_3) \tag{9}$$

33 Denavit-Hartenberg systém (specializace UI-ROB)

Popište systém Denavit-Hartenberg pro popis kinematického řetězce a uveďte transformační matice jednotlivých podkroků a výslednou transformační matici pro jeden krok v řetězci. Uveďte postup konstrukce popisu systému a základní vlastnosti výsledné popisné tabulky, shrnující parametry a proměnné jednotlivých podkroků.

Nástin řešení Viz `https://w.wiki/EJDQ`.

34 Segmentace obrazu (specializace UI-ROB)

Otázka se týká metod segmentace obrazu.

1. Definujte segmentaci obrazu. V definici je 5 vlastností segmentace. Uveďte příklad logického predikátu, který můžeme použít k segmentaci.
2. Popište prahovací metody segmentace a detailně popište Otsuho prahování.
3. Popište kroky K-means algoritmu. Uveďte jednu metodu určení parametru K. Jak použijeme K-means pro segmentaci?

Nástin řešení

1. Definujte segmentaci obrazu. V definici je 5 vlastností segmentace. Uveďte příklad logického predikátu, který můžeme použít k segmentaci.

   Definition Let R denote the spatial region of the whole image. Then segmentation is the division of R into n regions $R_1, R_2, \ldots, R_n$ such that:

   (a) $\bigcup_{i=1}^{n} R_i = R$

   (b) R_i is a connected set, $\forall i = 1, 2, \ldots, n$

   (c) $R_i \cap R_j = \emptyset, \forall i, j; i \neq j$

   (d) $Q(R_i) = \text{TRUE}, \forall i = 1, 2, \ldots, n$

   (e) $Q(R_i \cup R_j) = \text{FALSE}$ for adjacent R_i, R_j,

   where $Q(R_k)$ is a logical predicate defined over points of R_k, and the regions R_i and R_j are adjacent if $R_i \cup R_j$ is connected.

   Example Predicate

   $$|f(i,j) - f(\tilde{i},\tilde{j})| \leq t, \quad (i,j) \in R_k, (\tilde{i},\tilde{j}) \in R_k$$

2. Popište prahovací metody segmentace a detailně popište Otsuho prahování.

   Thresholding is the simplest technique, computationally inexpensive and fast

   Thresholding maps an input image $f(i,j)$ to the output image $g(i,j)$ such that:

   $$g(i,j) = \begin{cases} 1, & \text{if } f(i,j) \geq T \text{object} \\ 0, & \text{if } f(i,j) < T \text{ background} \end{cases}$$

   The image variance is defined as a constant for each threshold:

   $$\sigma_i^2 = \sigma_b^2(t) + \sigma_w^2(t)$$

   Intra-class (within) Variance – Minimize

   $$\sigma_w^2(t) = P_0(t)\sigma_0^2(t) + P_1(t)\sigma_1^2(t)$$

Inter-class (between) Variance – Maximize

$$\sigma_b^2(t) = P_0(t)(\mu_0(t) - \mu_I)^2 + P_1(t)(\mu_1(t) - \mu_I)^2 = P_0(t)P_1(t)(\mu_0(t) - \mu_1(t))^2$$

3. Popište kroky K-means algoritmu. Uveďte jednu metodu určení parametru K. Jak použijeme K-means pro segmentaci?

K-Means Algorithm

(a) Randomly distribute the initial means (K).

(b) Determine the assignment C for the given means:

$$C(i) = \underset{1 \le k \le K}{\arg\min} \|x_i - m_k\|^2, \quad i = 1, \ldots, N$$

(c) For a given assignment C, calculate the means m_k:

$$m_k = \frac{\sum_{x_i \in C_k} x_i}{N_k}$$

(d) Repeat steps 2 and 3 until stopping criteria are met:

- Mean Squared Error (MSE) < threshold, or
- No change in means.

Properties

- Will converge
- Not necessarily to the global optimum
- Sensitive to noise and outliers
- Sensitive to initial mean
- Convex clusters

K parameter Compute WCSS for different K, find Inflection (elbow) point

K-Means for segmentation Feature space: Brightness / color / texture + position (to have connected components)