Bakalářské zkoušky (příklady otázek)

2025-06-23

1 Deterministické konečné automaty (společné okruhy)

1. Uveďte definici jazyka $L(A)$ přijímaného daným deterministickým konečným automatem $A = (Q, \Sigma, \delta, q_0, F)$ včetně definice jeho rozšířené přechodové funkce δ^*.
2. Nechť $S \subseteq \Sigma^*$ je neprázdná konečná množina slov, Q je množina všech prefixů slov z S (včetně prázdného slova ε a slov z S) a $A = (Q, \Sigma, \delta, \varepsilon, S)$ je deterministický konečný automat s množinou stavů Q. Definujte jeho přechodovou funkci δ tak, aby přijímal jazyk

$$L(A) = \{w \in \Sigma^* \mid w \text{ obsahuje nějaké } s \in S \text{ (jako podslovo)}\}.$$

3. Sestrojte deterministický konečný automat přijímající jazyk

$$\{w \in \{0,1\}^* \mid w \text{ obsahuje 010 nebo } \underline{\text{končí}} \text{ na 10}\}.$$

Nástin řešení

1. $L(A) := \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$, kde $\delta^* \colon Q \times \Sigma^* \to Q$ je definována induktivně

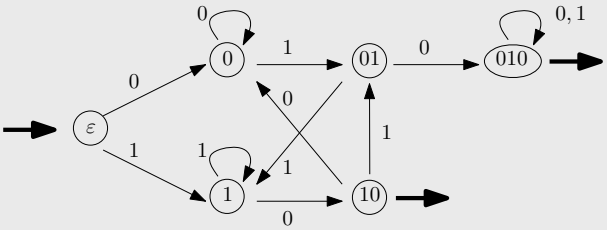
$$\begin{aligned} \delta^*(q, \varepsilon) &:= q, \\ \delta^*(q, wx) &:= \delta(\delta^*(q, w), x) \end{aligned}$$

   pro každé $q \in Q$, $w \in \Sigma^*$ a $x \in \Sigma$.

2. Pro každé $w \in Q$ a $x \in \Sigma$,

$$\delta(w, x) := \begin{cases} w, & \text{pokud } w \in S, \\ s, & \text{pokud } w \notin S \text{ a } wx \text{ končí na nějaké } s \in S \text{ (vezmi např. nejkratší)}, \\ v, & \text{kde } v \text{ je nejdelší sufix slova } wx, \text{ který je v } Q, \text{ v ostatních případech.} \end{cases}$$

3. Automat $A = (\{\varepsilon, 0, 1, 01, 10, 010\}, \{0,1\}, \delta, \varepsilon, \{10, 010\})$ je na obrázku níže.



2 Jednoduchý správce paměti (společné okruhy)

Předpokládejte v celé otázce pouze kontext 32-bitového ***little endian*** procesoru, ***bitem 0 rozumíme LSb.***

Naším cílem je v souvislém bloku paměti implementovat jednoduchý heap pro dynamickou alokaci podbloků paměti (neplést s haldou jako datovou strukturou). Celý náš heap bude mít pevnou velikost danou při inicializaci a nebude používat žádné

optimalizace nad rámec zde popsaného. Paměť našeho heapu si chceme organizovat jako souvislou posloupnost podbloků, kde každý podblok má být vždy ***zarovnaný na 2 byty***. Každý podblok heapu je buď *volný* nebo *obsazený* (naalokovaný).

Volné podbloky mají následující strukturu:
`+0`: `Size` [16-bit] = velikost podbloku ***včetně této hlavičky***. Jelikož z výše uvedeného vyplývá, že bit 0 této položky by byl vždy 0, tak tuto nulu neukládáme. Místo toho pro nás bude mít bit 0 položky `Size` speciální význam – bude označovat, zda je podblok volný nebo obsazený. `0` = *volný*, `1` = *obsazený*. Tedy pro volný podblok:
`bit 0`: `0` = volný
`+2`: `Next` [16-bit] = offset dalšího volného podbloku od začátku celého heapu – tedy volné podbloky tvoří vlastně jednosměrně vázaný seznam. Poslední volný podblok má v této položce vyplněnou hodnotu `-1`.
`+4`: zbytek podbloku je nevyužitý a je vyplněn nulovými byty.

Obsazené podbloky mají následující strukturu:
`+0`: `Size` [16-bit] = má stejný význam jako u volného podbloku, s tím rozdílem, že zde:
`bit 0`: `1` = obsazený
`+2`: `Payload` – samotná data podbloku (náš alokátor vrací všechny tyto byty vynulované).
Pozor: obsazené podbloky tedy nemají položku `Next` – místo v paměti, které u volného podbloku náleželo položce `Next`, se alokací podbloku stane součástí `Payload`u a musí být alokátorem na haldě vynulováno.

Pro celý heap si pamatujeme offset prvního volného podbloku od začátku celého heapu. Na počátku se heap skládá právě z 1 volného podbloku, jehož velikost odpovídá velikosti celého heapu. Při požadavku na alokaci nového podbloku dostaneme požadovanou velikost payloadu v bytech a použijeme první dostatečně velký volný podblok na nejnižším offsetu od začátku heapu (tj. strategie first-fit).

Předpokládejte následující kostru třídy v C#, kterou budete doplňovat (požadovanou implementaci můžete napsat v jazyce C# nebo Java nebo C++ – pro Javu nebo C++ předpokládejte ekvivalentní implementaci kostry v daném jazyce s tím, že parametry konstruktoru a metod si můžete vhodně upravit dle zvyklostí v daném jazyce) (`ushort` je 16-bitové bezznaménkové celé číslo):

```
class Heap {
        private ushort _firstFreeOffset = 0;
        private byte[] _heap;

        public Heap(byte[] availableMemory) {
                _heap = availableMemory;
                // TODO
        }
        private ushort FindFirstFree(ushort payloadSize) { // TODO }
        private void Mark(ushort offset, bool isFree) { // TODO }

        public void SetUshort(ushort offset, ushort value) { ... }
        public ushort GetUshort(ushort offset) { ... }
        public ushort Alloc(ushort payloadSize) { ... }
        public void Free(ushort payloadOffset) { ... }
}
```

Metody `SetUshort` a `GetUshort` **ne**programujte, ale naopak je vhodně využijte ve svém kódu. Metoda `SetUshort` na `offset` od začátku heapu uloží `value` v little endian pořadí. Metoda `GetUshort` vrátí 16-bitovou hodnotu uloženou na heapu od daného `offset`u (byty na heapu chápe v little endian pořadí).

Metody `Alloc` a `Free` **ne**programujte – tyto metody bude programovat někdo jiný s využitím vašich metod. Pro úkol 4 níže se hodí vědět: metoda `Alloc` naalokuje na heapu místo pro `payloadSize` velký payload a vrátí offset pro payload; metoda `Free` bere offset payloadu vráceného nějakým voláním metody `Alloc` a podblok, do kterého payload náleží, uvolní.

Úkoly:

1. Doplňte implementaci konstruktoru, který inicializuje prázdný heap.
2. Napište implementaci metody `FindFirstFree`, která vrátí offset prvního volného podbloku vhodné velikosti. Stav podbloku tato funkce nijak nemění.
3. Napište implementaci metody `Mark`, která očekává, že na `offset`u je platný podblok haldy, a změní jeho stav na „obsazený“ nebo „volný“ dle parametru `isFree`. Funkce nebude manipulovat s položkou `Next`.
4. Předpokládejte, že jsme pro náš heap dostali k dispozici 22 bytů – tato paměť je předem vyplněna nulovými byty. Předpokládejte, že provedeme s heapem následující operace:

A = `Alloc(2)`
B = `Alloc(4)`
C = `Alloc(2)`
`Free(B)`
Napište hexdump 22 bytů paměti konečného stavu heapu po provedení všech těchto operací – řešení vyplňte do jedné z níže uvedených tabulek (je jich zde redundantně více, kdybyste své řešení potřebovali opravit).

offset	+0	+1	+2	+3	+4	+5	+6	+7
0x0000								
0x0008								
0x0010								

offset	+0	+1	+2	+3	+4	+5	+6	+7
0x0000								
0x0008								
0x0010								

Nástin řešení

offset	+0	+1	+2	+3	+4	+5	+6	+7
0x0000	05	00	00	00	06	00	0E	00
0x0008	00	00	05	00	00	00	08	00
0x0010	FF	FF	00	00	00	00	–	–

3 Skalární součin (společné okruhy)

Uvažujme zobrazení ve tvaru $\langle x, y\rangle = x^T A y$ pro symetrickou matici $A \in \mathbb{R}^{n\times n}$.

1. Rozhodněte o platnosti výroků:
   (a) Je-li $\langle x, y\rangle = x^T A y$ skalární součin na $\mathbb{R}^n$, pak $\det(A) > 0$.
   (b) Je-li $B \in \mathbb{R}^{n\times n}$ symetrická regulární matice, pak $\langle x, y\rangle = x^T B^2 y$ je skalární součin na $\mathbb{R}^n$.
2. Uvažujme skalární součin na $\mathbb{R}^n$ zadaný ve tvaru $\langle x, y\rangle = x^T A y$ pro nějakou vhodnou matici $A \in \mathbb{R}^{n\times n}$. Zapište Cauchyho–Schwarzovu nerovnost tak, aby místo normy a skalárního součinu byly použity maticové a aritmetické operace.
3. Rozhodněte, zda pro všechna přirozená $n \geq 1$ platí, že matice
$$A = \begin{pmatrix} 1 & \dots & \dots & 1 \\ \vdots & 2 & \dots & 2 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 2 & \dots & n \end{pmatrix}$$
(tj. $A \in \mathbb{R}^{n\times n}$ s prvky $a_{i,j} = \min(i, j)$) je pozitivně definitní. Formulujte příslušné kritérium a ověřte ho, případně ukažte protipříklad.

Nástin řešení

1. (a) Ano, matice je pozitivně definitní, má tedy kladná vlastní čísla i determinant.
   (b) Ano, B^2 je symetrická s kladnými vlastními čísly, tedy pozitivně definitní.
2. Cauchyho–Schwarzova nerovnost: Pro každé $x, y \in V$ platí
$$|\langle x, y\rangle| \leq \|x\| \cdot \|y\|.$$
V našem případě má nerovnost tvar
$$|x^T A y| \leq \sqrt{x^T A x}\sqrt{y^T A y}.$$
3. Ano, matice A je pozitivně definitní. Například Choleského rozklad má tvar $A = LL^T$, kde
$$L = \begin{pmatrix} 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & 0 \\ 1 & \dots & \dots & 1 \end{pmatrix}$$
Snadno lze ověřit i Gaussovou eliminací nebo Sylvestrovým kritériem.

4 Centrální limitní věta (společné okruhy)

1. Napište znění Centrální limitní věty.
2. Předpokládejme, že program jeden vstup zpracovává náhodnou dobu se střední hodnotou 100 ms a směrodatnou odchylkou 20 ms. Použijte Centrální limitní větu pro odhad pravděpodobnosti, že zpracování sto vstupů bude trvat více než 10.3 s. Napište přesnou formuli pomocí Φ a také vyčíslete pomocí tabulky níže.
3. Vysvětlete, jaké jsme udělali přepoklady (záleží na tom, jaké je rozdělení doby běhu programu?) a jak jsou realistické.

x	-2.5	-2.0	-1.5	-1.0	-0.5	0.0	0.5	1.0	1.5	2.0	2.5
$\Phi(x)$	0.01	0.02	0.07	0.16	0.31	0.5	0.69	0.84	0.93	0.98	0.99

Nástin řešení 1. Znění věty: Nechť $X_1, X_2, \ldots$ jsou stejně rozdělené n.n.v. se střední hodnotou μ a rozptylem σ^2. Označme $Y_n = \frac{(X_1+\cdots+X_n)-n\mu}{\sqrt{n}\cdot\sigma}$.

Pak

$$Y_n \xrightarrow{d} N(0,1).$$

Neboli, pokud F_n je distribuční funkce Y_n, tak

$$\lim_{n\to\infty} F_n(x) = \Phi(x) \quad \text{pro každé } x \in \mathbb{R}.$$

Říkáme, že posloupnost Y_n konverguje k $N(0,1)$ *v distribuci*.

2. Označme X_i dobu běhu pro i-tý vstup. To, že $X_1 + \cdots + X_{100} > 10.3$ je ekvivalentní tomu, že $Y_{100} > \frac{10.3-100\cdot 0.1}{\sqrt{100}\cdot 0.02} = 1.5$. Podle CLV je $P(Y_{100} < 1.5) \approx \Phi(1.5)$. Proto hledaná pravděpodobnost je přibližně $1 - \Phi(1.5) \approx 1 - 0.93 = 0.07$, neboli zhruba 7 %.

3. CLV nepředpokládá nic speciálního o rozdělení jednotlivých X_i (pokud jsou veličiny stejně rozdělené s konečnou střední hodnotou a rozptylem). Předpokládáme ale nezávislost – to může být problém; pokud je například počítač přetížen, bude na všech vstupech pomalejší. Také nahrazujeme limitou hodnotu pro $n = 100$, to ale typicky při aplikaci CLV není problém.

5 SQL schéma pro datové katalogy (specializace WDOP)

Navrhněte SQL schéma pro reprezentaci datových katalogů na základě DCAT slovníku. Schéma musí obsahovat reprezentaci pro následující entity a jejich vlastnosti:

- Entita datový katalog (`dcat:Catalog`) s vlastnostmi:
  - IRI
  - název (`dct:title`)
  - datové sady (`dcat:dataset`)
- Entita datová sada (`dcat:Dataset`) s vlastnostmi:
  - IRI
  - název (`dct:title`)
  - klíčová slova (`dct:keyword`)
  - distribuce (`dcat:distribution`)
- Entita distribuce datové sady (`dcat:Distribution`) s vlastnostmi:
  - IRI
  - URL pro stažení (`dcat:downloadURL`)
  - formát (`dcat:format`)
  - poslední úprava (`dct:modified`)

Při návrhu předpokládejte následující:

- název je pouze v jednom jazyce,
- datová sada může mít více klíčových slov,
- datová sada náleží do právě jednoho katalogu,
- distribuce náleží do právě jedné datové sady.

Vysvětlete 3. normální formu a zajistěte, aby jí navržené schéma splňovalo.

Napište SQL, které vytvoří uvedené tabulky a vhodná integritní omezení.

Nástin řešení Je potřeba 4 tabulek:

- Catalog
- Dataset
- Keyword
- Distribution

Splnění 3. NF je přímočaré.

SQL by mělo obsahovat:

- vytvoření tabulek (`CREATE TABLE`),
- použití vhodného typu pro `dct:modified`,
- definici umělých primárních klíčů (`PRIMARY KEY`), nebo využití IRI,
- (`CONSTRAINT`) cizích klíčů (`FOREIGN KEY`), včetně cascade delete (`ON DELETE CASCADE`).

6 JSON-schéma (specializace WDOP)

Navrhněte JSON schéma pro publikaci dat o datové sadě z databáze v předchozí otázce. Vlastnost „název“ (`dct:title`) bude povinná jak pro datovou sadu, tak pro katalog.

Nástin řešení Řešení je opět poměrně přímočaré, tedy pouze základ pro ilustraci, bez distribucí.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "http://example.com/schema/data-catalog",
  "type": "object",
  "required": ["title"],
  "properties": {
      "title": { "type" : "string" },
      "keywords": { "type" : "array", "items": {"type": "string"}}
  }
}
```

7 JSON-LD (specializace WDOP)

Navrhněte JSON-LD kontext a popište úpravu JSON schématu tak, aby bylo možné na data pohlížet jako na RDF reprezentaci entity datové sady (`dcat:Dataset`) obsahující distribuce.

Nástin řešení JSON schéma rozšíříme o:

- Klíč `@id` pro IRI jednotlivých entit, je možné aliasovat.
- Klíč `@type` pro jednotlivé entity s odpovídající hodnotou, je možné aliasovat.
- Klíč `@context` pro kořen JSON dokumentu odkazující na soubor kontextu.

Je třeba také změnit reprezentaci `title` pro podporu indikace jazyka ve formátu JSON-LD language map, tj.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "http://example.com/schema/data-catalog",
  "type": "object",
  "required": ["@context", "@id", "@type", "title"],
  "properties": {
```

```
        "@id": { "type" : "string" },
        "@type": { "const" : "Catalog" },
        "title": {
           "type" : "object",
           "required" : [ "en" ],
           "properties" : {
              "en": { "type" : "string" }
           }
        },
        "keywords": { "type" : "array", "items": {"type": "string"}}
    }
  }
```

Alternativně lze vložit kontext i to dokumentu přímo, což by ale vyžadovalo delší popis jeho vnitřku.

Vybraná část JSON-LD pro ilustraci - konkrétní IRI nemusí být správně, je ale třeba vědět, že tam nějaké odpovídající použitým slovníkům je:

```
{
  "@context" : {
    "@version": 1.1,
    "dcat": "http://www.w3.org/ns/dcat#",
    "dcterms": "http://purl.org/dc/terms/",
    "Catalog": "dcat:Catalog",
    "title": {
  "@id": "dcterms:title",
  "@container": "@language"
  },
    "keywords": "dcat:keyword"
  },
}
```

8 PHP (specializace WDOP)

Navrhněte kostru implementace PHP skriptu pro obsluhu HTTP GET dotazu pro získání JSON reprezentace datové sady. HTTP GET dotaz obsahuje v URL QUERY části název `"title"`, který je použitý pro získání a vrácení první datové sady s daným názvem. Pokud je datová sada nalezena, PHP skript jí pošle jako odpoveď na požadavek ve formátu JSON. Schéma JSON dokumentu není specifikováno, můžete využít schéma z předchozích otázek.

Obslužte vhodně chybové stavy pomocí HTTP stavových kódů.

Cílem není poskytnout úplnou implementaci, ale popsat její hlavní body.

Nástin řešení Základní záchytné body:

- Kontrola vstupního argumentu, třeba pomocí funkce `filter_input`.
- Připojení k SQL pomocí mysqli.
- Vytvoření a sanitizace query.
- Uzavření spojení do databáze.
- Ošetření pomocí 404 a 500.
- Zaslání `content-type` hlavičky pro JSON na klienta.
- Zaslání JSON reprezentace, klidně pomocí `json_encode`.

9 Erdős-Ko-Radoova věta (specializace OI-G-O, OI-G-PADS, OI-G-PDM, OI-O-PADS, OI-PADS-PDM)

1. Zformulujte Erdős-Ko-Radoovu větu (o systémech navzájem se protínajících r-prvkových podmnožin n-prvkové množiny).
2. Uveďte příklad takového systému největší velikosti pro všechna přirozená čísla $r \leq n$ (včetně kombinací těchto parametrů vyloučených předpoklady Erdős-Ko-Radoovy věty).
3. Nechť n je liché přirozené číslo. Určete největší možnou velikost L systému $\mathcal{S}$ podmnožin množiny $\{1, \ldots, n\}$ takového, že každé dva prvky systému $\mathcal{S}$ mají neprázdný průnik. Dále určete největší přirozené číslo m takové, že nějaký takový systém velikosti L obsahuje pouze prvky velikosti alespoň m. Svá tvrzení dokažte.

Nástin řešení

1. Nechť n a r jsou přirozená čísla taková, že $n \geq 2r$. Nechť $\mathcal{S}$ je systém r-prvkových podmnožin množiny $\{1, \ldots, n\}$ takový, že každé dva prvky systému $\mathcal{S}$ mají neprázdný průnik. Pak
$$|\mathcal{S}| \leq \binom{n-1}{r-1}.$$
2. Jestliže $n \geq 2r$, pak můžeme vzít například systém všech r-prvkových podmnožin, které obsahují prvek 1. Jestliže $n < 2r$, pak se každé dvě r-prvkové podmnožiny protínají, a (jediným) příkladem tedy je systém všech r-prvkových podmnožin.
3. Systém $\mathcal{S}$ může z každé dvojice disjunktních podmnožin X a $\{1, \ldots, n\} \setminus X$ obsahovat nejvýše jednu, nutně tedy platí $|\mathcal{S}| \leq 2^{n-1}$. Velikost $L = 2^{n-1}$ je možná, příkladem je systém všech podmnožin velikosti větší než $n/2$. Každý takový systém $\mathcal{S}$ velikosti L nutně obsahuje právě jednu množinu z každé dvojice X a $\{1, \ldots, n\} \setminus X$; pro $X = \{1, \ldots, \lceil n/2 \rceil\}$ z toho plyne, že každý takový systém musí obsahovat prvek velikosti nejvýše $\lceil n/2 \rceil$. Dohromady s výše uvedeným příkladem tedy dostáváme $m = \lceil n/2 \rceil$.

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

1. Definujte pojem totálně unimodulární matice.
2. Uveďte přesné znění Minkowski-Weylovy věty o mnohostěnech.
3. Nechť $G = (V, E)$ je neorientovaný graf, s, t dva jeho různé vrcholy, a P množina všech nejkratších cest mezi s a t (každou cestu zde chápeme jako množinu hran). Popište algoritmus, který v polynomiálním čase (vzhledem k velikosti grafu G) najde množinu S minimální velikosti mající neprázdný průnik s každou množinou z P (tzv. min. hitting set množiny P).

Nástin řešení

1. **Definice:** Matice M je *totálně unimodulární*, pokud pro každou její čtvercovou podmatici A platí $\det(A) \in \{1, -1, 0\}$.
2. **Věta (Minkowski-Weyl):** Nechť $X \subseteq \mathbb{R}^n$. Pak X je omezený konvexní mnohostěn právě tehdy, když existuje konečná množina $V \subseteq \mathbb{R}^n$ taková, že X je konvexní obal množiny V.
3. Nechť $d(u, v)$ označuje délku (tj. počet hran) nejratší cesty mezi vrcholy u a v v grafu G. Označme $d = d(s, t)$ a položme
$$F = \{\{u, v\} \in E \mid d(s, u) + d(v, t) = d - 1\}\ ;$$
množinu F lze zkonstruovat v polynomiálním čase pomocí Dijkstrova algoritmu pro hledání nejkratších cest: pro každý vrchol $v \in V$ nejprve určíme $d(s, u)$ a $d(u, t)$, a pak z množiny E všech hran grafu G vybereme ty, které splňují výše uvednou podmínku. Nechť $H = (V, F)$.

   Všimneme si, že platí následující dvě tvrzení:

   a) Cesta p je nejkratší cesta mezi s a t v grafu G právě tehdy, když p je cesta mezi s a t v grafu H.

   b) Množina $S \subseteq F$ je tzv. hitting set množiny P právě tehdy, když $S \subseteq F$ je $s - t$-řez v grafu H.

Z výše uvedeného plyne, že stačí najít $s-t$-řez v grafu H nejmenší velikosti, což lze udělat jakýmkoli standardním algoritmem na maximální tok.

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

Observatoř extraterestriálních neutrin v antarktickém ledovci sestává z jednotlivých detektorů, identických koulí o poloměru r, rozmístěných v krychlové mřížce s rozestupy 1 m, zabírající celkem krychli K o hraně 2024 m (takže v každém ze tří souřadnicových směrů je 2025 koulí v řadě). Odhadněte, jak velké minimálně musí být r, aby platilo, že každé neutrino, které přiletí z Galaxie v Andromedě, projde středem K a odletí zpět do vesmíru, muselo projít aspoň dvěma detektory. Zformulujte větu, kterou použijete.

Nástin řešení Použijeme Minkowského větu, obdobně jako v případě pravidelného lesíku [Jiří Matoušek, Introduction to Discrete Geometry, Theorem 4.1.1, Example 2.1.2]. Stačí, aby válec o poloměru r a výšce 2024 m měl objem aspoň 8 m^3, tedy přibližně $r \geq 35.5$ mm.

12 Minory (specializace OI-G-PDM, OI-O-PDM, OI-PADS-PDM)

1. Napište definici minoru grafu a zformulujte charakterizaci rovinných grafů používající zakázané minory.
2. Graf je *vnějškově rovinný*, jestliže má rovinné nakreslení takové, že všechny jeho vrcholy jsou obsaženy v hranici vnější stěny. Ukažte, že graf je vnějškově rovinný právě tehdy, když neobsahuje K_4 ani $K_{2,3}$ jako minor.
3. Ukažte, že každý graf je minorem nějakého 3-regulárního grafu.

Nástin řešení

1. Graf H je minorem grafu G, jestliže graf izomorfní grafu H lze získat z podgrafu G kontrakcemi hran. Graf je rovinný právě tehdy, když neobsahuje K_5 ani $K_{3,3}$ jako minor.
2. Snadno nahlédneme, že podgraf vnějškově rovinného grafu je vnějškově rovinný, a že kontrakce hran zachovávají vnějškovou rovinnost. Každý minor vnějškově rovinného grafu je tedy vnějškově rovinný. Jelikož grafy K_4 a $K_{2,3}$ nejsou vnějškově rovinné, nemohou tedy být ani minory žádného vnějškově rovinného grafu.

   Naopak předpokládejme, že G je graf neobsahující K_4 ani $K_{2,3}$ jako minor. Uvažme graf G' vzniklý z G přidáním vrcholu u sousedícího se všemi vrcholy grafu G. Pak G' nemůže obsahovat K_5 ani $K_{3,3}$ jako minor a je tedy rovinný. Libovolné rovinné nakreslení grafu G' můžeme modifikovat tak, aby vrchol u byl incidentní s vnější stěnou (např. pomocí kruhové inverze podle bodu v libovolné vnitřní stěně incidentní s u). Odebráním u tedy dostáváme rovinné nakreslení G, kde všechny vrcholy jsou incidentní s vnější stěnou.
3. Bez újmy na obecnosti můžeme předpokládat, že G je graf minimálního stupně alespoň 3, jelikož každý graf je zjevně podgrafem (a tedy i minorem) nějakého takového grafu. Uvažme libovolný vrchol u grafu G stupně $k \geq 4$ a jeho sousedy označme $v_1, \ldots, v_k$. Nechť G' je graf vzniklý z $G-u$ přidáním cyklu $u_1u_2\ldots u_k$ a hran $u_1v_1, \ldots, u_kv_k$. Pak G vznikne z G' kontrakcí hran tohoto cyklu a G je tedy minorem G'. Navíc G' má menší počet vrcholů stupně většího než tři. Opakováním této operace tedy vznikne 3-regulární graf obsahující G jako minor.

13 Vyhledávání v textu (specializace OI-G-PADS, OI-O-PADS, OI-PADS-PDM)

Uvažujme algoritmus Aho-Corasicková pro hledání *slov* $\alpha_1, \ldots, \alpha_k$ v *textu* σ.

1. Definujte vyhledávací automat, jeho dopředné a zpětné hrany.
2. Nakreslete vyhledávací automat pro množinu slov {`ODPOR`, `OPORA`, `PODPORA`, `POROD`, `RODOPY`}.
3. Jakou časovou složitost má konstrukce vyhledávacího automatu a jakou jeho použití na nalezení všech výskytů slov v textu? Složitost vyjádřete vzhledem k součtu délek slov S, délce textu T a počtu výskytů V.

4. Navrhněte efektivní algoritmus, který dostane slova $\alpha_1, \ldots, \alpha_k$ a text σ a rozhodne, zda lze text získat jako *rotaci* některého ze slov. Rotace řetězce vznikne tak, že odebereme i znaků ze začátku řetězce a přesuneme je na konec. Například rotace řetězec `KOZA` jsou `OZAK`, `ZAKO`, `AKOZ`, `KOZA`. Algoritmus popište a analyzujte jeho časovou složitost. Pokud jako podprogram použijete některý ze standardních algoritmů na vyhledávání v textu, nemusíte ho detailně popisovat.

Nástin řešení

1. Stavy vyhledávacího automatu jsou prefixy hledaných slov. Dopředné hrany odpovídají prodloužení prefixu o 1 znak. Zpětná hrana ze stavu β vede do nejdelšího suffixu řetězce β, který je stavem.
2. Postupujeme přímo podle defince automatu.
3. Konstruce automatu má složitost $\mathcal{O}(S)$, hledání výskytů má složitost $\mathcal{O}(T+V)$.
4. Vytvoříme vyhledávací automat pro ta slova, která jsou stejně dlouhá jako text. Hledáme jejich výskyty ve *zdvojení* textu, tedy řetězci $\sigma\sigma$. Výskyt existuje právě tehdy, je-li text rotací daného slova.

14 Extrémy funkce více proměnných (specializace OI-G-O, OI-G-PADS, OI-G-PDM, OI-O-PADS, OI-PADS-PDM)

1. Definujme množiny $A, B, C \subseteq \mathbb{R}^2$ takto

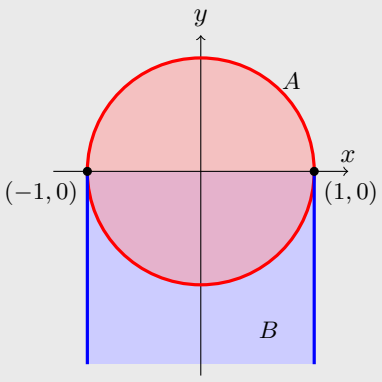
$$A = \{(x,y) \in \mathbb{R}^2;\ x^2 + y^2 \leq 1\}$$
$$B = \{(x,y) \in \mathbb{R}^2;\ -1 \leq x \leq 1 \wedge y < 0\}$$
$$C = A \cup B.$$

   Načrtněte, jak vypadá množina C. Je množina C otevřená? Je množina C uzavřená? Je množina C kompaktní?

2. Na množině C z předchozí podotázky definujeme funkci $f\colon C \to \mathbb{R}$ předpisem $f(x,y) = 4y - 3x$. Nabývá tato funkce na množině C svého minima? Nabývá tato funkce na množině C svého maxima? Pokud ano, v kterých bodech? (Extrémy funkce f uvažujeme vždy jen vůči množině C, v bodech mimo C není funkce definována.)

Nástin řešení

1. Náčrt množiny $C = A \cup B$ (A je červeně, B modře):



   Množina C je uzavřená (neboť obsahuje všechny body na své hranici), není otevřená a není kompaktní (neboť není omezená).

2. Funkce $f(x,y) = 4y - 3x$ není na C zdola omezená, což lze vidět například pokud se omezíme na hodnoty na polopřímce $x = 0 \wedge y \leq 0$. Nenabývá tedy v žádném bodě svého minima.

   Maxima nabývá f v bodě $b = (-\frac{3}{5}, \frac{4}{5})$, kde má hodnotu 5. To lze zjistit například touto úvahou: jelikož má f nenulové parciální derivace, nenabývá extrému v žádném vnitřním bodě C, extrémy tedy mohou být jen na hranici C. Hranici C tvoří sjednocení polopřímek $p_1 = \{x = 1 \wedge y \leq 0\}$ a $p_2 = \{x = -1 \wedge y \leq 0\}$ a půlkružnice $k = \{x^2 + y^2 = 1 \wedge y > 0\}$. Snadno nahlédneme, že na $p_1 \cup p_2$ nabývá f maxima v bodě $(-1, 0)$, kde má hodnotu 3.

Pro nalezení maxima na půlkružnici k můžeme buď použít přepis $y = \sqrt{1-x^2}$ a hledat maximum funkce $f(x, \sqrt{1-x^2}) = 4\sqrt{1-x^2} - 3x$, nebo pomocí Lagrangeových multiplikátorů určit, že potenciální extrém může existovat jen v takovém bodě k, kde je gradient (vektor parc. derivací) funkce f násobkem gradientu funkce $g(x, y) = x^2 + y^2 - 1$. Jelikož má funkce f konstantní gradient $(-3, 4)$ a g má gradient $(2x, 2y)$, připadá v úvahu pouze výše uvedený bod b. Tento bod je vskutku maximum funkce f na C, neboť celá množina C leží v polorovině určené nerovnicí $4y - 3x \leq 5$.

15 Anti-aliasing (specializace PGVVH-PG, PGVVH-VPH, PGVVH-PV)

Anti-aliasing je technika, která může významně vylepšit vzhled rastrových grafických výstupů.

1. Který z parametrů rastrového zobrazovacího zařízení je anti-aliasingem vylepšen (obrázek se nám jeví jako na „kvalitnějším“ zařízení – ale v jakém smyslu)? Jak se to pozná na výsledném obrázku? Jak byste se u výstupu do okna ve Windows přesvědčili, zda byl anti-aliasing použit?
2. Jak se může anti-aliasing implementovat v klasickém rastrovém vykreslování (rasterizace – např. při kreslení vektorové grafiky)?
3. Jak se anti-aliasing implementuje v prostředí paprskového zobrazovače (např. Ray-tracing)?

Nástin řešení

1. Anti-aliasing (vyhlazení) je vylepšení vzhledu nakresleného objektu do rastrového výstupního zařízení. Okraje vypadají více hladké, textury se v dálce „nezrní“ (potlačení interference, Moiré efektu). Je to vlastně imitace vyššího rozlišení displeje za pomoci barevných přechodových odstínů v okrajových pixelech. Virtuálně se zvyšuje rozlišení displeje.

   Matematický model pixelu: čtvereček (plocha!)

   Barva pixelu: integrální průměr barvy na ploše toho čtverečku. Tj. např. když se kreslí vybarvený kruh, tak na jeho okraji jsou pixely ne zcela pokryté kruhem, ty je potřeba vybarvit barvou tak sytou, jak je podíl zakryté plochy.

   Přesněji: je-li α podíl zakrytí pixelu objektem, je výsledná barva: $\alpha \cdot barvaObjektu + (1 - \alpha) \cdot barvaPozadi$.

   Jak se o A-A přesvědčit ve Windows: pořídit si screenshot daného okna a potom si tento rastrový obrázek prohlédnout se zvětšním (je lepší mít prohlížeč obrázků nastavený tak, aby sám neprováděl žádná vylepšení, žádné interpolace) - pokud budou na hranách vidět přechodové barevné odstíny, byl použit anti-aliasing.
2. Při rastrovém vykreslování (např. v interpretu SVG) se dá představit cílový pixel jako čtvereček $M \times M$ subpixelů (např. 4×4), kreslit vše původními algoritmy do obrázku s 4×4 vyšším rozlišením a potom výsledek zprůměrovat (filtrovat) do původního požadovaného rozlišení. Umí to dělat (nebo je to snadno implementovatelné na) GPU.
3. V Ray-tracingu se do jednoho pixelu posílá více paprsků místo jednoho. Ve 2D modelu pixelu (jednotkový čtvereček na ploše virtuálního senzoru) se musí použít některá z vzorkovacích metod (sampling), např. Jittering, Hammersley nebo “N-rooks”. V případě potřeby (vyšší efektivita) lze implementovat i adaptivní převzorkování, kde se více vzorků (paprsků) posílá jen tam, kde je to potřeba: kde je obrazová funkce “zajímavá”, to se zjistí primárním hrubším vzorkováním.

   Jittering: rozdělím čtvereček pixelu na $M \times M$ podčtverečků a do každého umístím jeden vzorek jako nezávislý náhodný pokus. Je to nestranný odhad požadovaného integrálu a potlačuje interference i vytváření shluků vzorků (šum).

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

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).

Nástin řešení 1. Objekty jsou obvykle definovány svými vrcholy nebo jinými významnými body. Všechny tyto body se podrobují maticovým transformacím. Je žádoucí, aby transformační systém uměl všechny běžné operace (translace, otočení, škálování...) a aby se daly složitější transformace sestavovat z těch jednodušších. To všechno splňují maticové transformace maticemi 3×3 (homomorfismus = bez translace) nebo 4×4 (homogenní matice transformace, která umí i translaci).

2. Umožňují translaci (posunutí) a dále projektivní (ne-afinní) transformace. Všechny tyto věci umí mnoho let i grafický HW (GPU karty). Bylo by vhodné ukázat normální i homogenní matici (viz každá učebnice).

3. Elementární transformační matice jsou v každé učebnici geometrie.

Příklad praktické složené transformace ve 3D: potřebujeme otočit těleso kolem jeho středu C maticí rotace R (ta může být třeba jen 3×3). Řešení – použijeme dvě translace a danou rotaci, výsledná matice bude mít rozměr 4×4. Celková matice bude $T = Tr(C) \cdot R \cdot Tr(-C)$

17 Rekurzivní sledování paprsku (specializace PGVVH-PG, PGVVH-VPH, PGVVH-PV)

Budeme se zabývat algoritmem zobrazování 3D scény, který se nazývá „Ray tracing“ (RT, česky by to bylo „Rekurzivní sledování paprsku“). Vaše slovní odpovědi můžete doprovodit obrázky a schématy, ale nezapomeňte je opatřit vysvětlivkami.

1. Popište princip algoritmu rekurzivního sledování paprsku
2. Z jakých složek se počítá světlo v rekurzivní funkci `shade()`? U jednotlivých složek můžete uvést, do jaké míry jsou fyzikálně správné (škála může být „přesně takto to v přírodě funguje“ – „ok, ale vzorec je trochu zjednodušen“ – „neodpovídá to fyzikální realitě, je tam jen kvalitativní shoda“).
3. Jaké hlavní nedostatky vykazuje základní algoritmus RT popsaný v prvním bodě, pokud bychom ho chtěli použít jako fotorealistickou zobrazovací metodu? Zkuste uvést aspoň rámcově, jak se dají jednotlivé nedostatky potlačit či omezit.

Nástin řešení Každým pixelem virtuálního rastrového obrázku (obrazovky) se pošle paprsek proti směru šíření světla do 3D scény (to navrhl již 1984 Whitted). Paprsek se protne se scénou a pokud na něco narazí, přenese se do pixelu barva toho průsečíku na povrchu tělesa. Pokud není zasaženo žádné těleso, vezme se barva pozadí. Sondovací paprsek se nejčastěji implementuje jako rekurzivní funkce

```
RGBcolor shade(Point3D origin, Vector3D direction, int recursionDepth)
```

Uvnitř funkce `shade()` se zkombinují tyto složky:

1. Přímé osvětlení povrchu tělesa ze všech definovaných světelných zdrojů. U každého zdroje světla se nejprve zkontroluje, zda má přímou viditelnost na průsečík (pokud ne, zdroj se ignoruje). Pak se aplikuje některý z lokálních modelů odrazu světla, například Phong. Přesnost: závisí na přesnosti modelu odrazu světla, ten může být hodně věrný. Ostré stíny se přepočítávající přesně.
2. Pokud inkrementovaná hloubka rekurze dosáhla limitu, žádně další složky se nepočítají a dosavadní hodnota se rovnou vrátí. Přesnost: zde se dopouštíme chyby, protože redukujeme množství světla. Obrázek bude tmavší než by měl být.
3. U potenciálně lesklého povrchu se spočítá zrcadlově odražený vektor a funkce `shade()` se zavolá rekurzivně. Výsledek se přičte přenásobený vhodnou konstantou (“lesklost” povrchu). Přesnost: nejnovější chybou je předpoklad zrcadlového směru odrazu paprsku. Reálné materály takhle nefungují.
4. U průhledného materiálu se podle indexů lomu určí zalomený směr paprsku a také se funkce `shade()` zavolá rekurzivně. Výsledek se přičte přenásobený vhodnou konstantou (“průhlednost” povrchu). Přesnost: chybou je opět předpoklad dokonale rovného povrchu, na kterém se světlo láme.

Akumulovaná barva se z funkce shade() vrátí jako výsledek. Má mít sémantiku “co by viděl pozorovatel z budou ’origin’, kdyby se díval směrem ’direction’”.

Hlavní nedostatky:

1. Rekurze je omezována, obrázek je tedy tmavší.

2. Zrcadlový směr odrazu/lomu. Mělo by se použít větší množství odražených/zalomených paprsků.
3. Nepřímé osvětlení – RT umí jen přímé osvětlení od viditelných zdrojů světla, to hrubě neodpovídá realitě. Měly by se započítat i delší cesty světla od zdrojů k průsečíku, to ale základní R-T neumí.
4. Bodové nebo směrové světelné zdroje (u kterých lze jednoduše spočítat viditelnost z průsečíku) jsou nerealistické. V přírodě existují jen plošné zdroje světla, ty by se měly zohlednit tím, že by se zavedla částečná viditelnost zdroje.
5. Jeden paprsek na pixel je málo, je třeba použít anti-aliasing (supersampling) a vzorkovat barvu pixelu mnoha primárními paprsky.

18 Vzorkování (specializace PGVVH-PG)

Algoritmy vzorkování (sampling) se používají v mnoha částech realistického renderingu (anti-aliasing, distribuovaný Ray-tracing, Monte-Carlo rendering).

1. Vysvětlete, co je to vzorkování (sampling) a které vlastnosti by mělo mít. Které vlastnosti byste vy osobně považovali za nejdůležitější (chceme váš názor, není jen jedna správná odpověď)? Pokud vám to pomůže, omezte se ve vašem uvažování pouze na 2D případy.
2. Uveďte alespoň dva příklady algoritmů pro vzorkování ve 2D, které splňují výše uvedené podmínky.
3. Praktická úloha: navrhněte vzorkovací algoritmus, který rovnoměrně pokrývá kruh umístěný v 3D prostoru (můžete si představit, že je vaším úkolem vzorkovat rovnoměrně svítící plošný zdroj nebo vzorkujete přední čočku fotografického objektivu). Zadání: střed kruhu $C = [x_C, y_C, z_C]$, normálový vektor $n = [x_n, y_n, z_n]$ a poloměr kruhu R. Uveďte alespoň náznak důkazu, že je váš algoritmus korektní.

Nástin řešení 1. Vzorkování je algoritmus generující vzorky (body) z daného cílového oboru, např. čtverce $S = [0.0, 1.0] \times [0.0, 1.0]$. Více formální definice: vzorkování je zobrazení množiny přirozených čísel $\{0, 1, 2 \dots N-1\}$ do dané cílové množiny S.

Žádoucí vlastnosti vzorkování:

- Rovnoměrné pokrytí cílové množiny (odpovídající konstantní hustotě pravděpodobnosti)
- Nahodilý vzhled
- Jednoduchý a efektivní výpočet
- Možnost generovat neomezenou třídu pseudo-náhodných vzorkování (číselný vstupní parametr), které jsou už sama o sobě deterministická

Nežádoucí vlastnosti vzorkování:

- Pravidelnost, zejména vizuálně zřejmá
- Některé body (nebo celé oblasti) cílové množiny nemohou být nikdy zvoleny

2. Dva příklady vzorkování čtverce 1×1:

- Náhodné vzorkování, kdy pokaždé vybereme nezávislým náhodným generátorem jeden bod z celého čtverce. $S_{rnd}(i) = [Rnd(0,1), Rnd(0,1)]$. Vznikají přirozené shluky, ale je korektně pokryta celá plocha čtverce a algoritmus se hodí pro paralelní výpočty (není třeba synchronizovat výpočetní jednotky, jen jim přiřadit různé generátory náhody). Velmi snadno se přidávají nové vzorky
- Jittering je postup, kdy nejprve rozdělíme čtverec na stejně veliké menší množiny (čtverce nebo obdélníky) a v nich potom aplikujeme náhodný výběr jako u předchozího vzorkování. Metoda je stále matematicky zcela korektní, přitom zůstává snadno implementovatelná a má méně shluků, což vede k výpočtům s menším šumem. Obtížněji lze přidávat nové vzorky. Formálně by se dal výběr popsat jako $S_{jitter}(i) = [Rnd(lx_i, hx_i), Rnd(ly_i, hy_i)]$, kde lx_i, hx_i, ly_i a hy_i vyjadřují meze sub-intervalů.

3. Vzorkování kruhu o poloměru R si převedeme do 2D, budeme vzorkovat čtverec $S = [-R, R] \times [-R, R]$ a odmítat pokusy, které padnou mimo požadovaný kruh o poloměru R. Jedná se o princip "rejection sampling". Lze použít například nejjednodušší Náhodné vzorkování (i Jittering by se dal použít, jen by dával různý počet vzorků). Vygenerujeme tolik

korektních vzorků, kolik je třeba, nakonec je přeneseme zpět do 3D (vhodným otočením kolem počátku souřadnic a následnou translací o vektor C).

Korektnost: Random sampling je korektní pro čtverec, odmítání neplatných vzorků tuto vlastnost neporuší.

19 Shadery generující geometrii (specializace PGVVH-VPH)

Moderní GPU umožňují psát shadery, které přímo na GPU generují další/novou geometrii.

1. Vyjmenujte alespoň dva typy takových shaderů. Uveďte, jak jsou do GPU pipeline zapojeny a s jakými daty pracují.
2. Předpokládejte, že na CPU (nebo výpočetním shaderem) počítáte nějakou náročnou částicovou simulaci ve 3D, pro visualizaci byste chtěli, aby se částice kreslily jako malé kuličky, každá má mít individuální barvu a poloměr. Navrhněte, který typ shaderů by se pro takovou vizualizaci nejlépe hodil, když budeme chtít ušetřit co nejvíce dat na vstupu visualizace. Stručně svůj návrh popište, doplňte pár deklarací (syntax není kritická, jen princip).
3. Váš 3D renderer by měl kreslit oblé předměty, které budou stínovány a i na obrysech mají vypadat pěkně (oblé). Vaše scéna je hodně velká a vy nechcete posílat do GPU tolik vrcholů, kolik byste potřebovali. Navrhněte řešení pomocí výše uvedených shaderů. Návrh dostatečně podrobně popište, alespoň hrubě naznačte, co by shader dělal (či jednotlivé shadery dělaly).

Nástin řešení 1. typy shaderů relevantních ke geometrii:

- Tesselation shaders se skládá ze tří komponent, dvě z nich jsou programovatelné/konfigurovatelné uživatelem, jedna je pevná. Tento systém je zařazen hned za Vertex shader, před Geometry shaderem. Úkolem teselace je zjemnit síť vrcholů, které tvoří primitivum (trojúhelníkovou síť, sadu lomených čar...). Je pevně dáno, jaká bude topologie na výstupu, programátor může jen měnit kvantitativní parametry (kolik nových vrcholů vznikne) a umisťovat nové vrcholy. Dvě programovatelné komponenty tedy jsou
  - Tesselation Control shader – konfigurace kvantitativních parametrů, podle kterých se budou nové vrcholy generovat (v 'pevně zadrátovaném' Tesselation Primitive Generatoru)
  - Tesselation Evaluation Shader – výpočet vlastních atributů nových vrcholů (poloha, normála, texturové souřadnice...)
- Geometry shader je úplně na konci zpracování vrcholů, před sestavováním primitiv. Konzumuje celá primitiva (tj. má přístup k jejích datům) a generuje místo nich nová. Je pomalejší, ale zato může být hodně obecný.

2. Nejúspornější reprezentace částice bude: střed, poloměr, barva. Elegantní možnost vykreslovat hodně vrcholů, ale nemuset je posílat z CPU/compute shaderu by byla implementovat Geometry shader, který bude konzumovat jednotlivé částice jako body (GL_POINTS) a vytvářet z nich celé kuličky. Vhodný kompromis by byl například generovat pravidelný dvacetistěn (icosahedron) s 12 vrcholy a 20 stěnami tvaru rovnostranného trojúhelníka.

Rozsáhlejší trojúhelníkové sítě nahrazující kouli už bych asi implementovat nedoporučil, to by se muselo přistoupit k použití Tesselation shaderů a složitějších vstupních dat (čtyřstěn).

3. Jedno z možných řešení spočívá v použití tzv. „PN Triangles“ (Point Normal Triangles). Model bude organizován jako velká trojúhelníková síť, přitom na GPU se každý originální trojúhelník rozdělí na devět manších, rozdělením každé původní hrany na tři části (plus jeden nový vrchol uprostřed, tím vznikne celkem sedm nových vrcholů). Tesselation shadery nejsou složité, k výpočtu nových vrcholů se použije trojúhelníkový Bézierův plát, takže se tím zachová spojitost celkového tělesa.

20 Segmentace obrazu (specializace PGVVH-PV)

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:

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

   (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)

21 Volací konvence, předávání parametrů (specializace PVS)

Uvažujte funkci, která akceptuje dva parametry typů „celé číslo velikosti 64 bitů“ a „pointer velikosti 64 bitů na celé číslo velikosti 64 bitů“ a vrací typ „logická hodnota“, a která má dále lokální proměnné stejných typů, jako jsou jejich argumenty a návratová hodnota, tedy například:

```
bool fn (int64_t a, int64_t *pb) {
    int64_t la = a + 1;
    int64_t *lpb = pb;
    bool lr = false;
    ...
}
```

1. Načrtněte standardní aktivační záznam (standard stack frame) funkce `fn` za předpokladu, že všechny parametry a všechny lokální proměnné jsou uložené na zásobníku, a že aktivační záznamy jsou běžným způsobem řetězeny. Vysvětlete roli jednotlivých položek (s výjimkou argumentů a lokálních proměnných, kde je význam zřejmý).
2. Do stávajícího náčrtku doplňte další aktivační záznam, který by vznikl, kdyby funkce `fn` uvnitř svého těla provedla jedno své vnořené volání, konkrétně kdyby v těle funkce (na výpisu výše nahrazeným třemi tečkami) byl řádek `lr = fn (la, lpb)`.
3. Vysvětlete (již nemusíte kreslit), jak by se uvedené aktivační záznamy změnily, kdyby volací konvence podporovala předávání parametrů v registrech. Věnujte se i otázce, jak by fungovalo použití registrů pro předání parametrů v případě vnořeného volání.

Pokud zadání neobsahuje některé informace, které potřebujete pro řešení úlohy, vhodně je zvolte a napište. Pokud jsou v náčrtcích položky, které obsahují adresy, nezapomeňte vyznačit, odkud se vezmou, případně kam ukazují. V řešení nemusíte uvažovat žádné potenciální optimalizace, kterými by překladač mohl řešení zásadně měnit.

Nástin řešení Řešení potřebuje dodefinovat některé drobnosti volací konvence – o návratové adrese se dá předpokládat, že bude mít velikost 64 bitů, podobně jako pointery v zadání, pořadí ukládání parametrů na zásobník je typicky zprava doleva (ale v zadání není nic, co by vylučovalo opačné pořadí), také je vhodné zvolit zarovnání, minimálně na velikost adres. Návratová hodnota se může předávat v registrech a tedy v řešení nebude figurovat.

1. Například:

   ```
   top + 41: pb (8 bytes) (hodnotu uloží volající kód)
   top + 33: a (8 bytes) (hodnotu uloží volající kód)
   top + 25: návratová adresa (8 bytes) (hodnota ukazuje za instrukci volající tuto funkci)
   top + 17: ukazatel na předchozí frame (8 bytes) (hodnota z frame pointer registru)
   top + 9:  la (8 bytes)
   top + 1:  plb (8 bytes) (podle kódu se vezme hodnota z pb)
   top:      lr (1 byte)
   ```

2. Další aktivační záznam bude podobný předchozímu, navíc bude ale padding vložený mezi lokální proměnné volající funkce a parametry volané funkce tak, aby byl aktivační záznam příslušně zarovnaný:

```
...
top + 49: zarovnání (7 bytes) (hodnota není důležitá)
top + 41: pb vnořeného volání (8 bytes) (hodnotu uloží volající kód)
top + 33: a vnořeného volání (8 bytes) (hodnotu uloží volající kód)
top + 25: návratová adresa (8 bytes) (hodnota ukazuje za instrukci volající tuto funkci)
top + 17: ukazatel na předchozí frame (8 bytes) (zde například původní top + 17, nyní top + 73)
top + 9:  la vnořeného volání (8 bytes)
top + 1:  plb vnořeného volání (8 bytes) (podle kódu se vezme hodnota z pb)
top:      lr vnořeného volání (1 byte)
```

Hodnota `top` v tomto bodě je o velikost aktivačního záznamu a zarovnání (dohromady 56 bajtů) menší než u předchozího bodu. Pokud by funkce `fn` ukládala na zásobník ještě další hodnoty (například hodnoty registrů, které musí být zachovány při volání funkce), pak by zmíněné aktivační záznamy mohly být vzdálenější.

3. Funkce `fn` má pouze dva parametry, oba velikosti 64 bitů, na běžných architekturách s registry stejné velikosti by se pravděpodobně oba předaly v registrech a nebyly by součástí aktivačního záznamu. Pokud předpokládáme, že volací konvence nevyžaduje zachování registrů obsahujících parametry, pak ani při vnořeném volání nevzniká problém, jinak by překladač musel použít zásobník pro uschování obsahu těchto registrů.

22 Template Method (specializace PVS)

1. Stručně popište základní myšlenky a důležité součásti návrhového vzoru *Template Method*.
2. Na vhodném příkladě ukažte dvě verze implementace: v Javě nebo C# ukažte implementaci pomocí dědičnosti, v C++ ukažte implementaci pomocí šablon. Zvolte takový příklad, na kterém demonstrujete všechny podstatné vlastnosti návrhového vzoru. Na zdrojovém kódu ukažte jak deklaraci příslušných tříd, tak jejich použití v klientském kódu. Srovnejte výhody a nevýhody těchto dvou způsobů implementace.

Zdrojový kód pište čitelně tiskacím písmem. V případě většího než zanedbatelného počtu škrtání, vsuvek, přepisování apod. přepište prosím výslednou verzi na nový papír. Drobné syntaktické nepřesnosti nebudou považovány za chybu.

Nástin řešení Dědičnost:

```
class Base {
protected:
    virtual preHook() {}
    virtual postHook() {}
    virtual doFirst() = 0;
    virtual doSecond() = 0;
    someFnc() {..}
public:
    execute() {
        preHook();
        doFirst();
        someFnc();
        doSecond();
        postHook();
    }
};
```

Šablony:

```
template< class Derived>
class Base {
public:
    execute() {
        Derived::doFirst();
        Derived::doSecond();
    }
```

```
};

class Alpha {
  doFirst();
  doSecond();
};

Base<Alpha> b;
b.execute();
```

execute = neměnná kostra algoritmu. doXxx = metody, které musí být konkrétní třídou implementovány. Hook = vhodné místo na rozšíření, může být implementováno.

Implementace pomocí šablon nevyužívá dědičnost a virtuální funkce a typicky je rychlejší. Použití konkrétního typu by ale mělo být známo v době překladu, jinak je technicky náročnější (type erasure). Implementace pomocí dědičnosti umožňuje snazší runtime polymorfismus, lze ji použít i v jazycích se slabou (nebo žádnou) podporou pro generické programování.

23 Relační databáze (specializace PVS)

Uvažujte následující popis cílové domény:

Vytváříte informační systém pro cestovní kancelář. Cestovní kancelář organizuje zájezdy. Každý zájezd má unikátní ID, název zájezdu a textový popis. Pro každý zájezd může existovat vícero termínů, ve kterých se koná. Termín zájezdu obsahuje datum odjezdu a příjezdu, cenu pro daný termín a celkovou kapacitu. Pro zájezd nemůže existovat vícero termínů se stejným odjezdem a příjezdem. Nicméně, může se stát (typicky během vytváření), že pro daný zájezd neexistuje žádný termín. Kancelář dále eviduje objednávky zájezdů (tedy přesněji konkrétních termínů) od klientů. Každý klient má přiřazené unikátní ID, jméno, příjmení a rodné číslo. Každá objednávka je vytvořena právě jedním klientem na právě jeden termín zájezdu. Dále u ní potřebujeme evidovat počet objednaných osob a datum, kdy byla vytvořena.

1. Vytvořte konceptuální model (UML nebo E-R diagram) pro daný popis domény. Pokud popis domény obsahuje nejasnosti, které by mohly ovlivnit modelování, slovně je popište a zmiňte, jakou interpretaci jste v modelu použili.[1]
2. Převeďte vytvořený konceptuální model do relačního schématu. Dodržujte formát `Název_relace(Atribut1, Atribut2, Atribut3, Atribut4, ...)`, kde každý klíč relace je vyznačen podtržením. V případě složeného klíče (`Atribut1, Atribut2`) jsou všechny položky podtrženy souvisle. Cizí klíče uvádějte ve formátu `Cizi_klic` $\subseteq$ `Relace.Atribut`.
3. Pro výsledné relační schéma napište SQL dotazy, které
   - Vrátí všechny objednávky klienta "Jan Novák".
   - Vrátí termíny zájezdů které jsou plně obsazené (suma počtu objednaných osob odpovídá celkové kapacitě termínu).

   Při řešení je možné použít SQL dialekt běžně známých databází (Oracle, MySQL, PostgreSQL, MSSQL, ...).

Nástin řešení Konceptuální diagram

Entity **Zájezd**, **Termín**, **Klient** a **Objednávka**. U objednávky je přípustné i modelování jako oboustranně nepovinná M:N relace mezi klientem a termínem s atributy datum a počet osob. Vztah mezi zájezdem a termínem je 0:N. Termín obsahuje složený klíč odjezd+příjezd+relace k zájezdu.

V popisu není explicitně definováno, zda jeden klient může vytvořit vícero objednávek ke stejnému zájezdu. To odpovídá modelování objednávky buď jako M:N relace, nebo samostatné entity. Z popisu není zcela zřejmé, zda mohou být vytvořeni klienti bez objednávky – po vysvětlení jsou přípustné jak 0:N tak 1:N kardinality.

Relační schéma

- Zájezd(**ID**, název, popis)
- Termín(**ID**, **ID_zájezd, od, do**, cena, kapacita); ID_zájezd $\subseteq$ Zájezd.ID

[1] Popis cílové domény je záměrně zjednodušený. Nevytvářejte jiné koncepty (třeba účastníky zájezdu) než ty, které jsou přímo zmíněny v zadání.

- Klient(**ID**, jméno, příjmení, RČ)
- Objednávka(**ID**, ID_termín, ID_klient, počet_osob); ID_termín ⊆ Termín.ID, ID_klient ⊆ Klient.ID

SQL dotazy

První dotaz např.

```
SELECT o.* FROM Objednávka o JOIN Klient k ON o.ID_klient = k.ID WHERE k.jméno = 'Jan' AND k.příjmení
= 'Novák';
```

Druhý dotaz např.

```
SELECT t.* FROM Termín t WHERE t.kapacita = (select sum(počet_osob) from Objednávka o WHERE
o.ID_termín = t.ID);
```

```
SELECT t.* FROM Termin t JOIN ( SELECT ID_termin, SUM(pocet_osob) AS celkem_osob FROM Objednavka
GROUP BY ID_termin ) o_sum ON t.ID = o_sum.ID_termin WHERE o_sum.celkem_osob = t.kapacita;
```

24 Validace požadavků a implementace softwarového systému (specializace PVS)

Středně velký tým vývojářů začíná pracovat na novém modulu rozsáhlého systému v bance nebo pojišťovně. Můžete si představit například modul pro internetové bankovnictví nebo klientský portál, který bude mít tři hlavní části:

- Front-end ve formě webové aplikace.
- Serverovou aplikaci (back-end), která bude mít na starosti vykonávání transakcí (jako převody mezi účty, atd.).
- Komponentu pro ukládání (logování) všech transakcí do vhodné databáze.

Požadavky na funkčnost jsou zapsané ve formě plain-text dokumentu, který obsahuje také popis jednotlivých scénářů ve strukturované podobě.

Definice scénářů vypadají podobně jako tato:

- Scénář 25: zadání nové platby
  - Obrazovka „detail běžného účtu“, kde na levé straně je zobrazen seznam možných akcí (tlačítek, odkazů).
  - Spouštěcí akce: uživatel klikne na tlačítko „Zadat platbu“.
  - Výsledek: zobrazení formuláře, který obsahuje nezbytné položky (číslo účtu, částka, variabilní symbol, zpráva, ...), tlačítko „odeslat“ a tlačítko „zrušit“.
- Scénář 26: odeslání platby
  - Obrazovka „formulář pro zadání nové platby“, kde jsou všechna pole, která musí vyplnit uživatel.
  - Spouštěcí akce: uživatel po vyplnění formuláře klikne na tlačítko „Odeslat platbu“.
  - Výsledek: kontrola vyplnění potřebných údajů na straně klienta (webová aplikace) a následné odeslání všech informací na server, kde se přidají do fronty požadavků ke zpracování; uživateli se pak zobrazí stav operace (úspěšné provedení nebo chyba).

Taková specifikace požadavků může být doplněna ještě o UML diagramy, které zachycují důležité entity systémů a jejich vztahy.

1. Uveďte, jaké postupy (metodologie), z pohledu softwarového inženýrství a běžných procesů vývoje software, byste použili pro validaci samotných požadavků, a jakým způsobem byste zajistili jejich přenesení do implementace. Popište způsob, kterým zajistíte dodržování a kontrolu splnění požadavků během vývoje systému, průběžně a také na konci před dodáním zákazníkovi. Vysvětlete potřebné důležité koncepty (jejich základní myšlenky).
2. Dále si představte situaci, kdy tester (zaměstnanec oddělení Q&A) nebo zákazník objeví v systému chybu, kterou oznámí spolu s potřebnými informacemi jako identifikace obrazovky, vstupní data (údaje zadané do formuláře), co přesně s aplikací dělal, platformu (verzi OS), atd.

   Stručně popište obecný postup, jak budete tuto chybu řešit (na úrovni kroků procesu vývoje software) a jaké akce byste udělali. Jakým způsobem se pokusíte zajistit, aby se stejná chyba už neopakovala, respektive byla včas zachycena

vývojáři nebo testery před vydáním další verze software? Zdůvodněte navržený postup a opět vysvětlete relevantní koncepty a jejich základní myšlenky.

Nástin řešení Měly by být zmíněny aspoň test-driven development, continuous integration, reprodukce chyb a regression testing. Detaily postupů viz materiály předmětů NSWI041 Úvod do softwarového inženýrství a NSWI154 Nástroje pro vývoj software.

25 DHCP (specializace SP)

Stručně popište fungování protokolu DHCP. Níže uvedená tabulka obsahuje popis (základního) paketu: od ethernetového rámce až po DHCP zprávu. Při řešení zahrňte do vašeho popisu činnosti všech účastníků (udržované seznamy adres atp.), nezapomeňte na detailní popis adresování jednotlivých paketů (mj. jak je možné pakety doručit správnému příjemci, když nemá IP adresu přidělenou).

V ukázkové zprávě `DHCP Offer` vyznačte části důležité pro adresování a zjistěte, jaká je *pravděpodobná budoucí* IP adresa klienta. Tabulka níže pak obsahuje čísla nejobvyklejších zpráv protokolu a pořadí všech DHCP zpráv, jak jsme je zachytili na síti (jde o textové shrnutí informací tak, jak je zobrazil Wireshark). Iniciální posun hexadecimálního výpisu o dva bajty je kvůli snazšímu zarovnání popisu IP paketu.

<table>
<tr><td colspan="32">0 3 4 7 8 15 16 18 19 23 24 31</td></tr>
<tr><td colspan="16">(ignorováno)</td><td colspan="16">Destination MAC (1)</td></tr>
<tr><td colspan="32">Destination MAC (2)</td></tr>
<tr><td colspan="32">Source MAC (1)</td></tr>
<tr><td colspan="16">Source MAC (2)</td><td colspan="16">EtherType</td></tr>
<tr><td colspan="4">Ver</td><td colspan="4">IHL</td><td colspan="8">TOS</td><td colspan="16">Total Length</td></tr>
<tr><td colspan="16">Identification</td><td colspan="3">Flags</td><td colspan="13">Fragment Offset</td></tr>
<tr><td colspan="8">Time to Live</td><td colspan="8">Protocol</td><td colspan="16">Header Checksum</td></tr>
<tr><td colspan="32">Source IP Address</td></tr>
<tr><td colspan="32">Destination IP Address</td></tr>
<tr><td colspan="16">Source Port</td><td colspan="16">Destination Port</td></tr>
<tr><td colspan="16">Length</td><td colspan="16">Checksum</td></tr>
<tr><td colspan="8">Op</td><td colspan="8">Htype</td><td colspan="8">Hlen</td><td colspan="8">Hops</td></tr>
<tr><td colspan="32">Transaction ID (xid)</td></tr>
<tr><td colspan="16">Secs</td><td colspan="16">Flags</td></tr>
<tr><td colspan="32">Client IP (ciaddr)</td></tr>
<tr><td colspan="32">Your IP (yiaddr)</td></tr>
<tr><td colspan="32">Server IP (siaddr)</td></tr>
<tr><td colspan="32">Relay agent IP (giaddr)</td></tr>
<tr><td colspan="32">Client MAC Address (chaddr) (total 16 bytes)</td></tr>
<tr><td colspan="32">(zbytek není potřeba)</td></tr>
</table>

Op	Message type
1	`DHCPDISCOVER`
2	`DHCPOFFER`
3	`DHCPREQUEST`
4	`DHCPDECLINE`
5	`DHCPACK`

Time	Length	Info
0.000000	314	`DHCP Discover: Transaction ID 0x3d1d`
0.000295	342	`DHCP Offer: Transaction ID 0x3d1d`
0.070031	314	`DHCP Request: Transaction ID 0x3d1e`
0.070345	342	`DHCP ACK: Transaction ID 0x3d1e`

DHCP Offer (kompletní paket):

```
0000  -- -- 00 0b 82 01 fc 42  00 08 74 ad f1 9b 08 00
0010  45 00 01 48 04 45 00 00  80 11 00 00 c0 a8 00 01
0020  c0 a8 00 0a 00 43 00 44  01 34 22 33 02 01 06 00
0030  00 00 3d 1d 00 00 00 00  00 00 00 00 c0 a8 00 0a
0040  c0 a8 00 01 00 00 00 00  00 0b 82 01 fc 42 00 00
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

Nástin řešení Protokol má 4 základní kroky, klient nejprve vyhledává volnou adresu a o tu si pak explicitně řekne. Rozšířením je pak lease na určitou dobu nebo možnost mít více DHCP serverů.

Adresování využívá broadcast a znalost MAC adres.

DHCP server si musí udržovat seznam volných a použitých adres (a ještě mezistav mezi offer-request a ack).

26 Synchronizace – semafor (specializace SP)

Popište základní rozhraní semaforu (tj. signatury funkcí a jejich význam) a pak jej naimplementujte pomocí mutexu. Plnohodnotné řešení by nemělo obsahovat aktivní čekání.

Můžete předpokládat, že máte k dispozici následující funkce. Jejich činnost by měla být zřejmá z jejich názvu; pokud očekáváte další předpoklady, nezapomeňte je zmínit (při speciálních předpokladech mějte, prosím, na paměti, že cílem otázky je prověřit vaše znalosti synchronizačních primitiv).

Funkce `thread_suspend_with_mutex` uspí aktuální vlákno, které ale tou dobou *musí* držet zamčený mutex. Při uspání je mutex odemčen a funkce garantuje, že po návratu (tj. po probuzení) je mutex opět uzamčen (jinými slovy, funkce předpokládá, že je volána v kritické sekci chráněné daným mutexem, ale atomicky mutex odemyká/zamyká se změnou stavu aktuálního vlákna).

Funkce `thread_wakeup` probudí vlákno dříve uspané pomocí `thread_suspend_with_mutex`. Na probuzení není potřeba držet mutex (spjatý s uspávací funkcí) zamčený a funkce neposkytuje garance, kdy přesně k probuzení dojde (tj. probouzené vlákno se fakticky může rozběhnout ještě před návratem z této funkce, ale i kdykoliv později).

Pokud potřebujete použít nějakou datovou strukturu (např. seznam), můžete předpokládat, že máte rozumnou implementaci k dispozici (a operace jako zařazení konkrétního vlákna do seznamu stačí reprezentovat vhodným voláním externí funkce); u datových struktur nepředpokládejte atomicitu operací bez explicitní synchronizace „zvenčí“.

```
typedef struct { ... } mutex_t;
typedef ... thread_id_t;

void mutex_init(mutex_t *mtx);
void mutex_lock(mutex_t *mtx);
void mutex_unlock(mutex_t *mtx);

thread_id_t thread_get_current(void);
void thread_suspend_with_mutex(mutex_t *mtx);
void thread_wakeup(thread_id_t thr);
```

Nástin řešení Signatury funkcí stačí typické `post`/`wait`, semafor si udržuje hodnotu vnitřního počitadla (množství účastníků v kritické sekci). `post` zvyšuje hodnotu a případně probouzí vlákna, `wait` snižuje hodnotu (a uspí vlákno, pokud by byla záporná).

Implementace potřebuje seznam, do kterého přidává a odebírá vlákna.

Vlastní implementace pak využije mutex pro ochranu struktury mutexu se seznamem vláken. Pokud by došlo ke snížení pod nulu, tak je potřeba přidat vlákno do fronty a uspat; při `post` se naopak probouzí vlákno ve frontě.

27 Thread pool (specializace SP)

Stručně popište význam a typické využití *thread poolu*.

Naimplementujte jednoduchý thread pool (s pevně daným množstvím worker vláken), který bude přijímat nové tasky pomocí metody `submit`, která vrátí výsledek jako tzv. *future*.

Jako součást řešení navrhněte triviální implementaci třídy `Future`, která bude mít (blokující) metodu `get` pro získání výsledku.

Při implementaci je možné použít aktivní čekání. Zaměřte se především na implementaci metody `submit` a odpovídajícího kódu ve *worker* vláknu, které řeší vlastní běh tasku.

Při implementaci můžete využít vhodná synchronizační primitiva (např. mutex) a základní datové struktury (pole, fronta apod.). Cílem otázky není funkčnost triviálně delegovat na jinou existující implementaci (tj. použití tříd jako `std::future` nebo `std::async` je zakázáno)!

Pro psaní kódu si zvolte jeden z jazyků C++, Java nebo C#; při návrhu rozhraní využijte vhodným způsobem generika, implementace funkcí může být řešena v pseudokódu.

Nástin řešení Thread pool je způsob implementace výkonné jednotky pro zpracování nezávislých tasků.

Metoda `submit` zařadí task do fronty a ihned vrací `Future`. Worker vlákna pak z této fronty vybírají, po dokončení výpočtu volají `complete`, která uloží výsledek a pokud je někdo blokovaný v `get`, tak jej probouzí (aby mohl vrátit spočtený výsledek).

28 Obsluha přerušení (specializace SP)

Předpokládejme RISC procesor a pro jednoduchost uvažujeme běh jen v privilegovaném režimu, kde de-facto neprobíhá překlad adres (tj. virtuální adresa je totožná s fyzickou).

Datové registry jsou `R0` až `R31`, kde `R31` je *return address*, `R30` je *stack pointer*.

Mezi řídící registry pak patří mj. `PC` a také registr pro pomocné výpočty `TMP` (který není nikdy použit v kódu generovaném překladačem a procesor jej k žádným účelům nepoužívá). Předpokládáme, že k přístupu ke speciálním (řídícím) registrům je potřeba zvláštní instrukce, která kopíruje hodnotu mezi řídícím registrem a normálním registrem (např. `MFC $TMP, R1`, která přesune z řídícího registru `TMP` do normálního `R1` a podobně `MTC` pro opačný směr).

Popište, které činnosti musí provést procesor, pokud dojde k externímu (asynchronnímu) přerušení (např. z časovače). Zřetelně popište, kdy má/může/musí mít zodpovědnost operační systém a kdy naopak je nutná (nebo vhodná) podpora ze strany hardwaru (procesoru).

Podle potřeby si doplňte další potřebné registry (např. jakým způsobem bude procesor sdělovat detaily o zdroji přerušení) nebo speciální instrukce a v hrubých obrysech naznačte, jak bude vypadat začátek a konec kódu obsluhy přerušení a kde bude umístěn.

Nástin řešení Potřebné registry jsou typicky `EPC` (původní pozice `PC`), `CAUSE` (pro uložení důvodu přerušení) a `STATUS` s režimem procesoru a bity, která přerušení jsou povolena.

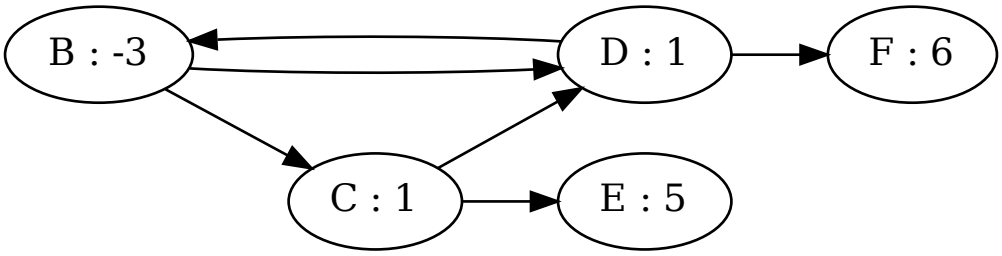
Při výjimce: procesor musí uložit `PC` do `EPC` a do `PC` nastavit adresu obsluhy přerušení (může být v samostatném registru nebo hard-coded dle ISA). Do `CAUSE` ještě uloží důvod (např. rozlišení časovač vs periferie). Pokud `STATUS` přerušení nedovoluje, může se tam poznamenat nějaký pending (a zkontrolovat při další instrukci). Na tohle už může navázat OS, který uloží kontext, k tomu může využít speciální registr `TEMPORARY`.

Procesor by mohl i uložit obsah registrů třeba na zásobník (podle `R30`, ale je jednodušší to nechat na software). Pro návrat je typicky speciální instrukce `ERET`, která převede `EPC` do `PC` a tím restartuje původní instrukci (plus může obnovit stav procesoru, tj. třeba návrat do uživatelského režimu).

29 Bellmanova rovnice užitku (specializace UI-SU, UI-ZPJ, UI-ROB)

Robot se pohybuje v konečném stavovém prostoru S. Pro každý stav $s \in S$ známe množinu možných akcí $A(s)$ a pravděpodobnost $P(s'|s,a)$, že se robot ze stavu s akcí $a \in A(s)$ přesune do stavu $s' \in S$. Při průchodu stavem $s \in S$ dostane robot odměnu (reward) $R(s)$. Celkový užitek ze stavu s' je diskontován faktorem $\gamma = 0.9$.

(1) Napište rovnici pro výpočet maximálního užitku (utility) $U(s)$ ve stavu s.



Obrázek udává stavový prostor, odměnu v jednotlivých stavech a dvě možné akce z každého stavu kromě koncových stavů E a F. Pravděpodobnost, že se robot přesune do stavu, který je daný orientovanou hranou zvolené akce, je 0.8 a ve zbylé pravděpodobnosti 0.2 se přesune do stavu, do kterého směřuje druhá možná akce.

(2) Napište soustavu rovnic, jejichž řešení dá užitky jednotlivých stavů, jestliže zadáme robotu akce (strategii/policy) $B \to C$, $C \to D$ a $D \to F$.

(3) Jakým algoritmem bychom tuto soustavu vyřešili?

(4) Jak ze spočtených užitků zjistíme, zda zadaná volba akcí dává maximální možné užitky pro všechny stavy?

(5) Jak bychom pokračovali, abychom našli akce pro všechny stavy maximalizující užitek?

30 Lineární regrese (specializace UI-SU, UI-ZPJ)

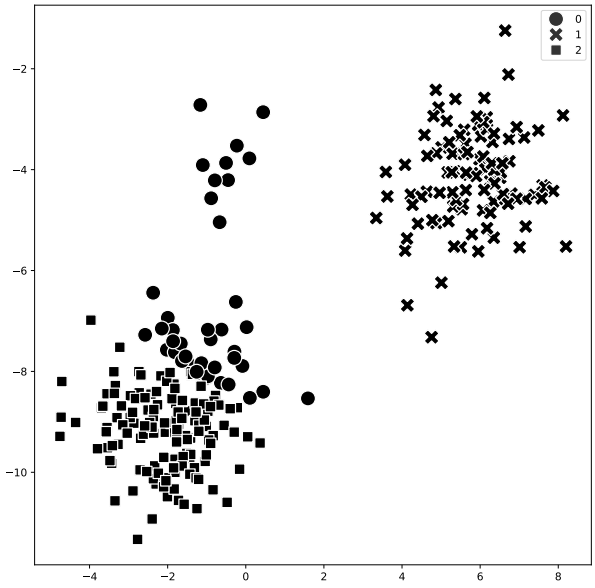
1. Pro trénovací dataset X_{train} a cílové hodnoty t_{train} zformulujte model lineární regrese a chybovou funkci, kterou v lineární regresi minimalizujeme.
2. Ukažte, jakým způsobem lze explicitně získat optimální řešení na trénovacích datech.
3. Zformulujte optimalizaci pomocí stochastic gradient descent.
4. Okomentujte výhody a nevýhody obou přístupů, především z hlediska možnosti kontroly chyby na případných validačních datech.

Nástin řešení Definice modelu a podrobnější odvození jsou ve slidech k prvním dvěma přednáškám z NPFL129.

Je třeba spočítat derivaci chyby, tu pak položit rovnou nule a vyřešit buď s využitím inverzní matice nebo s pomocí SVD dekompozice. Nevýhoda je, že invertování matice je pomalé a matice je velká. Optimální řešení na trénovací množině nemusí být to, co chceme, protože může dojít k přeučení. SGD bude méně náročné na paměť, navíc můžeme kontrolovat loss na validační množině a udělat early stopping a vyhnout se přeučení.

31 Shlukování (specializace UI-SU, UI-ZPJ)

1. Popište stručně algoritmus k-means. Jakým způsobem se volí počáteční polohy středů shluků (centroidů)? Jak se jejich polohy v průběhu algoritmu upravují? Kdy algoritmus končí?
2. Jaké kritérium algoritmus k-means optimalizuje?
3. Algorimus k-means pro $k = 3$ jsme spustili na data se třemi shluky z obrázku níže. Jednotlivé značky ($\times$, $\bullet$, $\blacksquare$) odlišují nalezené shluky. Proč algoritmus na těchto datech nenašel optimální shluky podle kritéria v předchozím bodě? Jak se dá zvýšit šance, že je algoritmus najde?



Nástin řešení

1. Počáteční centroidy se mohou volit náhodně z dat. V průběhu algoritmu se každý bod z dat přiřadí k nejbližšímu centroidu, ten se potom přepočítá jako střed bodů k němu přiřazených. Tyto kroky se opakují dokud se mění přiřazení bodů ke shlukům, nebo po daný počet iterací.
2. Minimalizuje se
$$\sum_{i=1}^{N}\sum_{k=1}^{K} z_{i,k}||x_i - \mu_k||^2,$$
kde N je počet bodů, K je počet středů a $z_{i,k}$ vyjadřuje, že bod x_i patří do shluku se středem μ_k ($z_{i,k} = 1$ pokud tam patří, jinak 0).
3. Prostřední shluk, který byl přiřazen k části levého, je mnohem menší než ostatní dva shluky. Šance na lepší výsledek se dá zlepšit opakováním náhodné inicializace počátečních centroidů a výběrem nejlepšího výsledku podle funkce v předchozím bodě.

32 EM algoritmus (specializace UI-SU)

1. Popište EM algoritmus.
2. Uvažujme pravděpodobnostní model definovaný bayesovskou sítí s veličinou *Cluster* s hodnotami $\{1, 2\}$ jakožto rodičem veličin *Color* a *Holes*, které nabývají hodnot $\{blue, green\}$ resp. $\{yes, no\}$. Veličiny *Color* a *Holes* nejsou spojeny hranou.

   Určete parametry modelu a inicializujte je hodnotami z množiny $\{0.4, 0.6\}$, aby tvořily korektní model a podmíněné pravděpodobnosti pro podmínku $Cluster = 1$ nebyly identické s těmi pro $Cluster = 2$.
3. Zvolte jeden z parametrů z předchozího bodu a vypočtěte pro něj jednu aktualizaci dle EM algoritmu. Uvažujte následující data, kde „?“ značí chybějící hodnotu.

Color	Holes	Cluster
blue	no	?
blue	yes	?
blue	no	?
green	yes	?
green	yes	?

Při výpočtu můžete použít následující aproximace: $0.6^0 * 0.4^3 \approx 0.064$, $0.6^1 * 0.4^2 \approx 0.096$, $0.6^2 * 0.4^1 \approx 0.144$, $0.6^3 * 0.4^0 \approx 0.216$.

Nástin řešení

1. EM algoritmus umožňuje odhadnout parametry modelu s nepozorovanými proměnnými. Inicializuje model a iteruje kroky E a M, dokud parametry modelu nezkonvergují.

   E Spočte očekávané hodnoty chybějících dat.

   M Spočte maximálně věrohodné odhady parametrů na datech s doplněnými chybějícími hodnotami.

2. Parametry modelu (s jejich inicializací pro následující bod):

parametr	inicializace	značení
$P(Cluster = 1)$	0.6	θ
$P(Color = blue \mid Cluster = 1)$	0.6	θ_{B1}
$P(Color = blue \mid Cluster = 2)$	0.4	θ_{B2}
$P(Holes = yes \mid Cluster = 1)$	0.6	θ_{H1}
$P(Holes = yes \mid Cluster = 2)$	0.4	θ_{H2}

3. Nejjednodušší aktualizace je pro parametr θ. Přes datové řádky $[color_j, holes_j]$ spočteme $\sum_j P(Cluster = 1|color_j, holes_j)$ a vydělíme počtem příkladů.

Color	Holes	$P(Cluster = 1\|color_j, holes_j)$
blue	no	$\frac{\theta \cdot \theta_{B1} \cdot (1-\theta_{H1})}{\theta \cdot \theta_{B1} \cdot (1-\theta_{H1}) + (1-\theta) \cdot \theta_{B2} \cdot (1-\theta_{C2})} = \frac{0.6*0.6*0.4}{0.6*0.6*0.4+0.4*0.4*0.6} = \frac{0.6*0.6*0.4}{0.4*0.6} = 0.6$
blue	yes	$\frac{\theta \cdot \theta_{B1} \cdot \theta_{H1}}{\theta \cdot \theta_{B1} \cdot \theta_{H1} + (1-\theta) \cdot \theta_{B2} \cdot \theta_{H2}} = \frac{0.6*0.6*0.6}{0.6*0.6*0.6+0.4*0.4*0.4} = \frac{0.216}{0.216+0.064} = 0.77$
blue	no	jako výše, 0.6
green	yes	$\frac{\theta \cdot (1-\theta_{B1}) \cdot \theta_{H1}}{\theta \cdot (1-\theta_{B1}) \cdot \theta_{H1} + (1-\theta) \cdot (1-\theta_{B2}) \cdot \theta_{H2}} = \frac{0.6*0.4*0.6}{0.6*0.4*0.6+0.4*0.6*0.4} = 0.6$
green	yes	jako výše, 0.6

Pro námi zvolené počáteční parametry je nová hodnota θ rovna $\frac{4 \cdot 0.6 + 0.77}{5} = \frac{3.17}{5} = 0.634$.

33 Šumový kanál a strojový překlad (specializace UI-ZPJ)

1. Uveďte, jak lze úlohu strojového překladu převést na model šumového kanálu (noisy channel model, v kontextu klasického statistického strojového překladu).
2. Popište, jak lze v tomto přístupu natrénovat parametry obou hlavních komponent (modelů).

Nástin řešení

– Klasický statistický strojový překlad je často formulován jako problém dekódování v modelu šumového kanálu, kde se hledá nejpravděpodobnější věta v cílovém jazyce e, která mohla generovat větu ve zdrojovém jazyce f:

$$e^* = \arg\max_e P(e|f)$$

Pomocí Bayesova pravidla lze tento výraz přepsat jako:

$$e^* = \arg\max_e P(f|e) \cdot P(e)$$

Kde:

– $P(f|e)$ je **překladový model** – modeluje, jak by se věta e přeložila do f,

  - $P(e)$ je **jazykový model** – určuje pravděpodobnost a plynulost věty v cílovém jazyce.

- Jazykový model se obvykle trénuje na rozsáhlých monolingválních korpusech v cílovém jazyce. Často se používají n-gramové modely, které aproximují pravděpodobnost věty jako součin podmíněných pravděpodobností:

$$P(e) \approx \prod_{i=1}^{n} P(e_i | e_{i-1}, \ldots, e_{i-n+1})$$

  Odhad pravděpodobností se provádí pomocí relativních četností slovních posloupností, typicky doplněných o vyhlazování.

- Překladový model se učí z **paralelních korpusů**, které obsahují odpovídající věty ve zdrojovém a cílovém jazyce. Trénovací postup zahrnuje:

  1. Zarovnání slov, např. pomocí modelů IBM.
  2. Extrakci frází podle zarovnání.
  3. Výpočet překladových pravděpodobností, např.:

$$P(f|e) = \frac{\text{počet výskytů dvojice } (f, e)}{\text{počet výskytů } e}$$