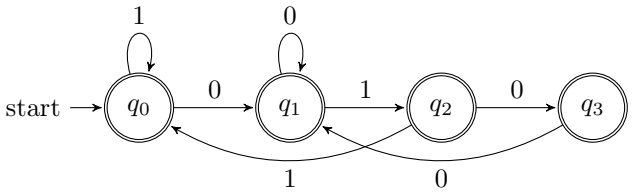
Bakalářské zkoušky (příklady otázek)

2025-02-04

1 Průnik bezkontextového a regulárního jazyka (společné okruhy)

1. Uveďte formální definici *zásobníkového automatu*.
2. Uvažme následující jazyky nad abecedou $\Sigma = \{0, 1\}$:
   - L_1 je jazyk generovaný bezkontextovou gramatikou $G = (\{S\}, \Sigma, \mathcal{P}, S)$ s množinou pravidel $\mathcal{P} = \{\, S \to SS \mid 0S1 \mid \epsilon \,\}$ (kde ϵ značí prázdné slovo),
   - L_2 je jazyk rozpoznávaný deterministickým konečným automatem $A = (\{q_0, q_1, q_2, q_3\}, \Sigma, \delta_A, q_0, \{q_0, q_1, q_2, q_3\})$, jehož přechodová funkce δ_A je dána následujícím stavovým diagramem:



   Sestrojte zásobníkový automat rozpoznávající průnik $L = L_1 \cap L_2$ jazyků L_1 a L_2.

Nástin řešení

1. Zásobníkový automat je sedmice $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, kde
   - Q je konečná, neprázdná množina stavů,
   - Σ je konečná, neprázdná množina znaků vstupní abecedy,
   - Γ je konečná množina znaků zásobníkové abecedy,
   - δ je přechodová funkce $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to P_{FIN}(Q \times \Gamma^*)$, kde $P_{FIN}(.)$ značí konečnou podmnožinu prvků,
   - $q_0 \in Q$ je počáteční stav,
   - $Z_0 \in \Gamma$ je počáteční zásobníkový symbol,
   - $F \subseteq Q$ množina přijímajících stavů.

   V případě přijímání prázdným zásobníkem lze vynechat množinu přijímajících stavů F.
2. Můžeme použít standardní konstrukci. Přijímání automatem A simulujeme pomocí stavů, generování gramatikou G pomocí zásobníku stejně jako při převodu gramatiky na jednostavový PDA. Výsledný zásobníkový automat, přijímající prázdným zásobníkem, je $P = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, S\}, \delta_P, q_0, S)$ kde δ_P je definována následovně:

$$\delta_P(q_i, a, a) = \{(\delta_A(q_i, a), \epsilon)\} \text{ pro } a \in \{0, 1\}, i \in \{0, 1, 2, 3\}$$
$$\delta_P(q_i, \epsilon, S) = \{(q_i, SS), (q_i, 0S1), (q_i, \epsilon)\} \text{ pro } i \in \{0, 1, 2, 3\}$$

   První pravidlo představuje čtení vstupního symbolu automatem A a kontrolu s vrcholem zásobníku, druhé pravidlo simuluje jeden krok derivace gramatiky G.

(Alternativně lze přímo sestrojit zásobníkový automat rozpoznávající $L_1 \cap L_2$, nebo sestrojit bezkontextovou gramatiku generující $L_1 \cap L_2$ a následně ji převést na zásobníkový automat.)

2 Knihovna pro výpočty s maticemi (společné okruhy)

Poznámky, jak odpovídat na otázku:

- Pro odpověď si vyberte jeden z jazyků Java, C++ nebo C#.
- Snažte se použít co možná nejnovější syntaxi (a idiomy, atd.) vybraného jazyka.

Představte si, že vyvíjíte **knihovnu pro výpočty s maticemi**.

1. Definujte kontrakt (bez implementace) pro reprezentaci dvourozměrné matice. Tento kontrakt musí poskytovat *prostředky* (tj. metody, funkce, vlastnosti apod. podle zvoleného jazyka) alespoň pro:
   - získání velikosti jednotlivých rozměrů,
   - získání hodnoty z matice na základě souřadnic dané hodnoty,
   - provádění aritmetických operací (sčítání s jinou maticí, odčítání, násobení, atd.).
2. Vytvořte implementaci svého kontraktu, která je vhodná pro velmi velké, ale **řídké** (tzv. *sparse*) matice (tj. matice obsahující převážně nuly). Okomentujte svou implementaci tak, aby bylo zřejmé, jak funguje. Z metod (funkcí, atd.) implementujte pouze ty, které vrací velikosti rozměrů a hodnotu z matice na základě souřadnic.

   Definujte prostorovou složitost své implementace ve stylu $O(something)$.
3. Představte si, že máte více implementací svého kontraktu pro matice (např. pro řídké matice a pro husté matice). Dále si představte, že máte metodu (nebo funkci, atd.), která dostane soubor s hodnotami matice a určí, která implementace je pro danou matici ideální.

   S využitím vhodného *návrhového vzoru* implementujte *prostředek*, který dostane soubor a vrátí instanci kontraktu matice s ideální implementací.

Nástin řešení

1. Rozhraní nebo abstraktní třída s metodami podle otázky, přetíženými operátory (pokud jsou podporovány), vlastnostmi (pokud jsou podporovány) atd.
2. Existuje mnoho možných implementací (pro matici MxN určitě není správnou implementací dvourozměrné pole MxN, které obsahuje všechny hodnoty). Správné řešení je uložit pouze nenulové hodnoty (plus nějaká „management“ data). Možná řešení: (i) seznam obsahující trojice [řádek, sloupec, hodnota] se seřazením podle řádkového indexu a pak podle sloupcového indexu, (ii) seznam seznamů obsahující dvojice [sloupec, hodnota] a seřazený, (iii) Compressed Sparse Row formát (CSR), tj. tři pole, z nichž jedno drží nenulové hodnoty, druhé drží sloupec, kde se daná hodnota nachází, a třetí drží index (v hodnotách a sloupcích), kde daný řádek začíná, (iv) atd. Implementace metody getValue závisí na zvolené reprezentaci. Pro CSR to může být například:

   ```
   public double getValue(int row, int col) {
       checkBounds(row, col);
       int start = rowPointers[row];
       int end = rowPointers[row + 1];
       for (int idx = start; idx < end; idx++) {
           if (colIndices[idx] == col) {
               return values[idx];
           }
       }
       return 0.0;
   }
   ```

   Složitost závisí na zvolené implementaci, ale měla by být podobná $O(pocetNenulovychHodnot + neco)$.
3. Odkazuje to na implementaci na základě návrhového vzoru *Factory*, například:

```
public static Matrix create(Path path) {
   return switch (getSuitableType(path)) {
       case "sparse": new SparseMatrix(path);
       case "dense": new DenseMatrix(path);
       default throw new RuntimeException("unknown␣implementation");
   }
}
```

3 Matice (společné okruhy)

1. Najděte symetrickou matici $S \in \mathbb{R}^{3\times 3}$ takovou, aby
$$\mathrm{Ker}(S) = \mathrm{span}\{(1,5,1)^T, (0,2,1)^T\}.$$
2. Definujte pojem *podobné matice* a stanovte, které z následujících veličin mají podobné matice stejné: hodnost, determinant, jádro.
3. Matice
$$A = \begin{pmatrix} 2 & 2 & 3 \\ 0 & -1 & -4 \\ 1 & 2 & 4 \end{pmatrix}$$
má vlastní čísla 3 a 1. Rozhodněte, zda existuje báze, vzhledem k níž má lineární zobrazení $f(x) = Ax$ matici v diagonálním tvaru.

Nástin řešení

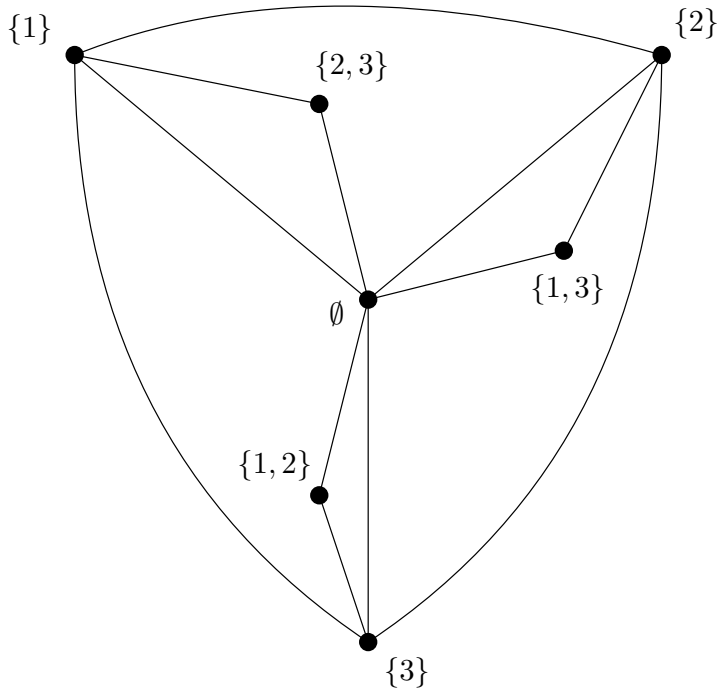
1. Řádkový prostor matice je kolmý na jádro, čili je generován vektorem $(3,-1,2)^T$. Ze symetrie pak matice musí mít tvar
$$S = \alpha(3,-1,2)^T(3,-1,2) = \alpha \begin{pmatrix} 9 & -3 & 6 \\ -3 & 1 & -2 \\ 6 & -2 & 4 \end{pmatrix},$$
kde $\alpha \neq 0$.
2. Matice $A, B \in \mathbb{R}^{n\times n}$ jsou podobné, pokud existuje regulární matice R taková, že $A = RBR^{-1}$. Podobné matice mají stejnou hodnost i determinant. Jádro ne, například pro matice $A = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$, $B = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$.
3. Ne, neexistuje. Vlastní číslo 1 je dvojnásobné, ale přísluší mu jen jeden vlastní vektor, neboť $\mathrm{rank}(A - I_3) = 2$. Tudíž matice nelze diagonalizovat.

4 Eulerovské grafy (společné okruhy)

Nechť n je přirozené číslo větší než 1. Definujeme graf G_n takto: jeho vrcholy jsou všechny množiny $A \subset \{1,\ldots,n\}$, pro něž platí $|A| < n$. Dvě (různé) z nich jsou spojeny hranou právě tehdy, když jsou disjunktní. Například pro $n = 3$ dostaneme tento graf:

1. Pro která $n > 1$ je graf G_n souvislý?
2. Nechť $A \subset \{1,\ldots,n\}$ je množina velikosti menší než n. Jaký je stupeň vrcholu, který odpovídá množině A? (Závisí nějak na velikosti množiny A, na n, nebo případně na něčem jiném)?
3. Formulujte, co znamená pojem eulerovský graf a rozhodněte pro která $n > 1$ je graf G_n eulerovský?

Odpovědi přiměřeně zdůvodněte.



Nástin řešení

1. Vrchol odpovídající $\emptyset$ je spojen se všemi ostatními vrcholy grafu, takže graf je souvislý vždy.
2. Je-li $|A| = k > 0$, je odpovídající vrchol spojen se všemi vrcholy z podmnožin $\{1, \ldots, n\} \setminus A$ a má tak stupeň 2^{n-k}. Pro $A = \emptyset$ je stupeň $2^n - 2$ (chybí hrany do $\emptyset$ a $\{1, \ldots, n\}$) a tedy všechny vrcholy mají sudé stupně.
3. Graf je eulerovský pro všechna $n > 1$ dle Eulerovy věty: G (bez izol. vrcholů) je eulerovský $\Leftrightarrow$ G souvislý a všechny vrcholy mají sudý stupeň?

5 SQL (specializace WDOP)

Uvažujte následující situaci: Uživatelé konverzují s ChatBotem. Ke konverzaci můžou přidávat dokumenty, které se v konverzaci využívají. Konverzace pak obsahuje zprávy vložené uživatelem a také zprávy vygenerované ChatBotem.

1. Pomocí jazyka SQL specifikujte příkazy pro vytvoření tabulek Uzivatel, Zprava a Dokument, v nichž jsou ukládána příslušná data. Zajistěte referenční integritu mezi tabulkami.
2. Specifikujte příkazy, které do každé tabulky vloží alespoň jeden záznam. Záznamy se musí vztahovat k jedné konkrétní konverzaci.
3. Specifikujte dotaz, který vrátí počty zpráv dlouhých konverzací. Dlouhou konverzaci definujeme jako konverzaci, která obsahuje alespoň 100 zpráv.
4. Specifikujte příkazy pro smazání tabulek.

Nástin řešení

1. Například (ale ne nutně přesně) takto:

```
create table Uzivatel (
    id_u int primary key,
    jmeno varchar(100));
create table Zprava (
    id_z int primary key,
    id_k int not null,                  // identifikátor konverzace
    cas datetime,                       // kdy byla zpáva poslána
```

```
        id_u int references Uzivatel(id_u),  // kterému uživateli zpráva patří
        chatbot bool,                        // true = poslal ChatBot, false = poslal uživatel
        text varchar(5000));                 // text zprávy
    create table Dokument (
        id_d int primary key,
        url varchar(1000),
        id_z references Zpráva(id_z));
```

2. ```
   insert into Uzivatel values (
     1, 'Johny Adamec');
   insert into Zprava values (
     101, 1, '2025-01-30 23:09', 1, false, 'Hello world !');
   insert into Dokument values (
     11, 'zadaniBctestu.txt', 101);
   ```

3. ```
   select id_k, count(*) as pocet from Zprava group by (id_k) having count(*) > 100;
   ```

4. ```
   drop table Dokument;
   drop table Zprava;
   drop table Uzivatel;
   ```

   V tomto pořadí. Nebo drop constraint pro cizí klíče a potom v libovolném pořadí.

6 Indexování dat (specializace WDOP)

Do tabulky Uzivatel z předchozího příkladu byli postupně vloženi uživatelé s id_u = [15, 9, 23, 25, 19, 40, 17] v tomto pořadí. Následně byl uživatel s id_u = 15 smazán.

1. Nakreslete, jak bude vypadat index nad sloupcem id_u, který je realizován jako B+ strom, kde maximální počet potomků m = 3, po vložení všech záznamů.
2. Nakreslete tentýž index po smazání uvedeného záznamu.
3. Specifikujte, zda se jedná o redundantní nebo neredundantní B-strom a proč.

Nástin řešení

```
1)
            [19, .]
           /    \
     [15, .]      [23, .]
    /  \          /  \
[9, 15][17, 19][23, .][25, 40]

2)
            [19, .]
           /    \
     [15, .]      [23, .]
    /  \          /  \
[9, .][17, 19][23, .][25, 40]

3) Redundantní
```

7 Návrh REST API (specializace WDOP)

Popište základní principy a úrovně návrhu REST API. Demonstrujte principy na návrhu REST API pro aplikaci 2. úrovně, ve které uživatelé konverzují s ChatBotem. API musí umožnit pracovat s následujícími zdroji: konverzace, zpráva. Popište, jak by se projevila implementace 3. úrovně.
```

Nástin řešení Úrovně:

- 0. jeden endpoint pro všechno
- 1. URL pro každý zdroji
- 2. použití HTTP akcí jako GET, POST, PUT, ...
- 3. hypermedia controls, také znám jako HATEOAS

Předpokládáme, že zpráva patří do konverzace. Máme tedy:

- ./api/v1/conversations/
  - POST - vytvoří novou konverzaci a vrátí její URL
  - GET - vrátí seznam URL všech konverzací
- ./api/v1/conversations/conversation-identifier
  - GET - vrátí informace/detail o konverzaci
  - DELETE - smaže konverzaci
  - PUT - umožní změnit detail konverzace, třeba název
- ./api/v1/conversations/conversation-identifier/messages
  - POST - vytvoří novou zprávu v konverzaci a vrátí její URL
  - GET - vrátí seznam URL všech zpráv v konverzaci
- ./api/v1/conversations/conversation-identifier/messages/message-identifier
  - GET - vrátí informace/detail o zprávě
  - DELETE - smaže zprávu

Implementace HATEOAS je možné dosáhnout pomocí vhodné volby obsahu odpovědí. Ty musí klientovi umožnit navigovat stavem aplikace, třeba pomocí odkazů.

8 Základy šifrování (specializace WDOP)

Popište úlohu a využití public key infrastructure (PKI) certifikátů a šifrování pro implementaci HTTPS. Není třeba jít do implementačních detailů, konkrétního výběru algoritmů nebo struktury certifikátu. Z odpovědi však musí být jasné, jak na sebe jednotlivé pojmy navazují a proč jsou třeba.

Nástin řešení HTTPS vkládá šifrování pod HTTP textový protokol. Šifrovat chceme symetrickou šifrou, což je rychlé a efektivní. Potřebujeme si ale vyměnit klíč, využijeme asymetrického šifrování. To umožní klientovi poslat bezpečně informaci serveru a dohodnout se na komunikaci. Potřebujeme tedy získat veřejný klíč serveru, ten získáme z certifikátu. Abychom certifikátu mohli věřit, musí být podepsaný. Certifikát je podepsaný dalším certifikátem - PKI. Končí to u root certifikační autority, jejíž certifikát dostaneme s instalací SW nebo OS.

9 Catalanova čísla, vytvořující funkce (specializace OI-G-O, OI-G-PADS, OI-G-PDM, OI-O-PADS, OI-PADS-PDM)

Podotázky 1 a 2 jsou nezávislé, neočekává se, že byste k jejich řešení použili stejnou metodu.

1. Nechť C_n je počet dobrých uzávorkování skládajících se z n otevíracích a n zavíracích závorek. Napište vzorec vyjadřující hodnotu C_n (nemusíte ho odvozovat).

   Nechť D_n je počet posloupností $a_1 a_2 \dots a_n$ kladných celých čísel takových, že $a_1 = 1$ a $a_i \leq a_{i-1} + 1$ platí pro každé $i \in \{2, \dots, n\}$. Například $D_3 = 5$, možné posloupnosti jsou 111, 112, 121, 122 a 123. Ukažte, že $D_n = C_n$.

2. Nechť s_1 a s_2 jsou řetězce písmen. Řetězec s je *vsunutí* s_1 do s_2, jestliže je roven zřetězení řetězců s_2', s_1 a s_2'', kde s_2 je zřetězení s_2' a s_2''. Například řetězec `abcdbba` je vsunutí řetězce `cd` do řetězce `abbba`.

   Nechť $\mathcal{A}$ je množina obsahující pouze řetězce z písmen `a` a `b` a $\mathcal{C}$ je množina obsahující pouze řetězce z písmen `c` a `d`. Nechť $a(x)$ je vytvořující funkce množiny $\mathcal{A}$, tj. pro každé přirozené číslo n je koeficient u x^n v $a(x)$ roven počtu řetězců délky n v $\mathcal{A}$. Obdobně, nechť $c(x)$ je vytvořující funkce množiny $\mathcal{C}$.

   Nechť $\mathcal{S}$ je množina všech vsunutí řetězce z $\mathcal{C}$ do řetězce z $\mathcal{A}$. Vyjádřete na základě funkcí $a(x)$ a $c(x)$ vytvořující funkci $s(x)$ množiny $\mathcal{S}$.

Nástin řešení

1. Catalanovo číslo C_n je rovno
$$\frac{1}{n+1}\binom{2n}{n}.$$
   Rovnost $D_n = C_n$ můžeme dokázat například nalezením bijekce: Dobrému uzávorkování n otevíracích a n zavíracích závorek můžeme přiřadit posloupnost $a_1 \dots a_n$, kde a_i je rovno úrovni zanoření i-té otevírací závorky.

2. Nechť a_n a c_n jsou počty řetězců délky n v $\mathcal{A}$ a $\mathcal{C}$, tj. $a(x) = \sum_{n\geq 0} a_n x^n$ a $c(x) = \sum_{n\geq 0} c_n x^n$. Nejprve si jako $\mathcal{B}$ označme množinu dvojic (s, p), kde $s \in \mathcal{A}$ a $p \in \{0, 1, \dots, |s|\}$; takovou dvojici můžeme interpretovat jako řetězec s s vyznačenou pozicí mezi dvěma jeho písmeny (nebo na začátku či konci řetězce). Nechť b_n je počet takových dvojic, že $|s| = n$, a $b(x) = \sum_{n\geq 0} b_n x^n$. Pak $b_n = (n+1)a_n$, funkci b tedy můžeme vyjádřit jako $(xa)'$.

   Řetězec $s \in \mathcal{S}$ délky n jednoznačně odpovídá dvojici $((s_2, p), s_1)$, kde $(s_2, p) \in \mathcal{B}$, $s_1 \in \mathcal{C}$ a $|s_1| + |s_2| = n$ (řetězec s je vsunutí s_1 do s_2 na pozici p). Z toho dostáváme $s = (xa)'b$.

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

1. Nechť P je konvexní mnohostěn. Definujte pojmy
   - tečná nadrovina mnohostěnu P,
   - stěna mnohostěnu P,
   - vrchol mnohostěnu P.

2. Nechť $G = (V, E)$ je neorientovaný graf, $m = |E|$ a nechť
$$P = \{x \in \mathbb{R}^m \mid \sum_{e\in\delta(v)} x_e = 1 \quad \forall v \in V, \quad x_e \geq 0 \quad \forall e \in E\},$$
   kde $\delta(v)$ označuje množinu hran incidentních vrcholu u. Dokažte, že každý vrchol x mnohostěnu P je poločíselný, tedy pro každou hranu $e \in E$ platí $x_e \in \{0, 1/2, 1\}$.

   Nápověda: Pro daný vrchol x mnohostěnu P se zamyslete nad strukturou množiny $\{e \in E \mid x_e > 0\}$ (může obsahovat cykly sudé délky? může obsahovat cesty jiné délky než jedna?).

3. Nechť $G = (V, E)$ je neorientovaný graf a P mnohostěn definovaný v předchozím bodě. Dokažte, s využitím předešlého bodu, následující tvrzení: Je-li x^* optimální řešení úlohy $\max_{x\in P} \sum_{e\in E} x_e$, pak v grafu G existuje párování velikosti $|V|/3$.

Nástin řešení

1. Definice: Nechť $P \subset \mathbb{R}^n$ je konvexní mnohostěn, $c \in \mathbb{R}^n$ a $t \in \mathbb{R}$. Jestliže existuje $x \in P$ t.ž. $c^T x = t$, a jestliže pro každé $x \in P$, $c^T x \leq t$, pak množinu $\{x \in \mathbb{R}^n \mid c^T x = t\}$ nazýváme tečná nadrovina mnohostěnu P, a množinu $\{x \in P \mid c^T x = t\}$ stěna mnohostěnu P. Vrchol mnohostěnu P je stěna dimenze 0.

2. Platí: Množina $F = \{e \in E \mid x_e > 0\}$ je sjednocením vrcholově disjunktních hran a lichých cyklů. Lze dokázat sporem: kdyby v množině byl cyklus sudé délky, půjde x vyjádřit jako konvexní kombinace dvou jiných bodů z P, což je spor s tím, že x je vrchol P. Podobně kdyby v množině byla cesta délky aspoň 2 (která není částí žádného cyklu), pak pro její první hranu e musí platit $x_e = 1$, dokonce pro každou lichou hranu e musí platit x_e, naopak pro každou sudou hranu e musí platit $x_e = 0$, a pro poslední hranu e také musí platit $x_e = 1$; to je spor, protože poslední hrana je lichá.

Z předešlého plyne, že pro každou hranu $e \in F$, která není částí žádného cyklu v F, musí platit $x_e = 1$. Z předešlého také plyne, že pro každou hranu $e \in F$, která patří do nějakého lichého cyklu, musí platit $x_e = 1/2$. Jiné hrany v F nejsou, čímž je tvrzení dokázáno.

3. Uvažme optimální řešení x^*. Z teorie mnohostěnů víme, že buď x^* je přímo vrchol mnohostěnu P, nebo existuje vrchol $\bar{x}$ mnohostěnu P splňující $\sum_{e \in E} \bar{x}_e = \sum_{e \in E} x^*_e$; v dalším proto bez újmy na obecnosti předpokládáme, že x^* je vrchol mnohostěnu P. Podle bodu 2. víme, že množina $F = \{e \in E \mid x_e > 0\}$ je sjednocením vrcholově disjunktních hran a cyklů liché délky. Nechť F' je podmnožina odvozená z F tak, že vezmeme všechny samostatné hrany, a z každého lichého cyklu vezmeme všechny sudé hrany (počáteční vrchol bereme libovolně). Protože hrany a cykly v F byly vrcholově disjunktní, budou disjunktní i hrany v F', tedy bude se jednat o párování. Protože z každého cyklu vezmeme alespoň jednu třetinu hran, a protože každý vrchol je v nějaké hraně nebo cyklu množiny F, dostáváme požadovaný dolní odhad na velikost párování.

11 Geometrie – Incidence bodů a přímek (specializace OI-G-O, OI-G-PADS, OI-G-PDM)

1. Definujte pojem incidence mezi množinou bodů a množinou přímek. Kolik je maximální možný počet incidencí mezi množinou n bodů a množinou m přímek v rovině? (Stačí zformulovat asymptotický horní odhad, bez důkazu.)
2. Pro libovolně velké n popište přesně konstrukci množiny n bodů a množiny n přímek v rovině, pro které je počet incidencí mezi nimi asymptoticky co největší. (Nepovinná otázka k zamyšlení: Co když navíc zakážeme rovnoběžné přímky?)

Nástin řešení

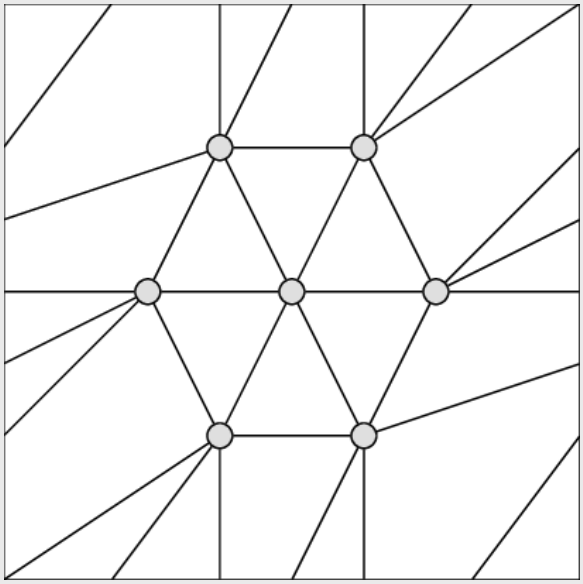
1. Jiří Matoušek, Introduction to Discrete Geometry, Theorem 4.1.1
2. Jiří Matoušek, Introduction to Discrete Geometry, Proposition 4.2.1

12 Grafy na plochách (specializace OI-G-PDM, OI-O-PDM, OI-PADS-PDM)

1. Nakreslete K_7 na torus.
2. Dokažte, že K_8 na torus nakreslit nelze.
3. Jaký je největší možný Eulerovský rod plochy, na kterou lze buňkově nakreslit K_5?

Nástin řešení

1. Například



2. Zobecněná Eulerova formule implikuje, že každý graf G nakreslený na toru splňuje $|E(G)| \leq 3|V(G)|$; graf K_8 má $28 > 3 \cdot 8$ hran.
3. Má-li graf G buňkové nakreslení na ploše Eulerovského rodu g, pak zobecněná Eulerova formule implikuje, že počet stěn tohoto nakreslení je roven $|E(G)| - |V(G)| - g + 2$. Jelikož počet stěn je alespoň 1, dostáváme z toho $g \leq |E(G)| - |V(G)| + 1$. Pro K_5 tak dostáváme $g \leq 6$.

   Naopak libovolné buňkové nakreslení, které má právě jednu stěnu, bude mít Eulerovský rod 6. Můžeme ho dostat například tak, že v nakreslení K_5 do projektivní roviny (s 6 stěnami) přidáme na jeho hrany 5 křižítek tak, aby se stěny spojily do jedné.

13 Silná souvislost (specializace OI-G-PADS, OI-O-PADS, OI-PADS-PDM)

1. Definujte silnou souvislost orientovaného grafu a její komponenty.
2. Popište algoritmus, který najde komponenty silné souvislosti zadaného grafu v čase $\mathcal{O}(n+m)$, kde n je počet vrcholů a m počet hran.
3. Navrhněte co nejefektivnější algoritmus, který pro zadaný orientovaný graf najde „šéfa“ – vrchol, z nějž jsou všechny ostatní vrcholy dosažitelné orientovanou cestou. Případně sdělí, že v grafu žádný šéf není.

Nástin řešení

1. Viz Průvodce labyrintem algoritmů, kapitola 5.9.
2. Například algoritmy z kapitol 5.9 (Kosaraju-Shamir) a 5.10 (Tarjan) v Průvodci.
3. Najdeme komponenty silné souvislosti. Každý orientovaný graf má alespoň jednu zdrojovou komponentu. Pokud je právě jedna, každý její vrchol je šéfem. Pokud jich je více, neexistuje mezi vrcholy z různých zdrojových komponent žádná cesta, takže žádný šéf neexistuje. Celý algoritmus pracuje v čase $\mathcal{O}(n+m)$. Dodejme, že existují i jiné algoritmy se stejnou složitostí založené na šikovném použití opakovaného prohledávání do hloubky/šířky.

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

V tomto příkladu na prostorech $\mathbb{R}^d$ a jejich podmnožinách vždy uvažujeme standardní euklidovskou metriku.

1. Definujme množinu $M \subseteq \mathbb{R}^2$ předpisem

$$M = \{(x,y) \in \mathbb{R}^2;\ -1 \leq x \leq 2 \ \wedge\ -3 < x+y < 2\}.$$

Načrtněte, jak množina M vypadá. Je množina M (chápaná jako podmnožina $\mathbb{R}^2$) otevřená? Je M uzavřená? Je M kompaktní?

2. Nechť M je opět množina z předchozí podotázky. Rozhodněte, zda existuje funkce $f\colon M \to \mathbb{R}$, která je v každém bodě M spojitá, zároveň je na M zdola omezená, ale nenabývá v žádném bodě svého minima. (Extrémy i spojitost funkce f uvažujeme vzhledem k množině M, mimo tuto množinu není f definovaná.)
3. Uvažujme opět stejnou množinu M a definujme na ní funkci $g\colon M \to \mathbb{R}$ předpisem $g(x,y) = x^2 - 2x + y^2 + 2y + 4$. Rozhodněte, zda tato funkce nabývá na množině M svého minima, a případně určete, v kterém bodě se minimum nabývá.

Nástin řešení

1. Množina M je kosodélník s vrcholy v bodech $(-1,-2)$, $(2,-5)$, $(1,1)$ a $(-1,3)$, z něhož jsou odstraněny horní a dolní hrana. Množina M není ani otevřená ani uzavřená, není tedy ani kompaktní.
2. Taková funkce f existuje, například $f(x,y) = y$. Pro tuto funkci máme $\inf_{(x,y)\in M} f(x,y) = -5$, ovšem v žádném bodě M se hodnota -5 nenabývá.
3. Funkce g nabývá svého minima v bodě $(1,-1)$. To lze zjistit třeba takto: ve vnitřních bodech M může g nabývat minima jen tam, kde obě její parciální derivace jsou rovny nule (případně neexistují), což dává jako jediného kandidáta bod $(1,-1)$, v němž má g hodnotu 2. Pomocí hessiánu se můžeme přesvědčit, že v tomto bodě je lokální minimum. V krajních bodech M, kde platí buď $x = -1$ nebo $x = 2$, máme $g(-1,y) = y^2 + 2y + 6 \geq 5$ resp. $g(2,y) = y^2 + 2y + 4 \geq 3$, tedy v žádném takovém bodě nenabývá g minima. Alternativně lze úlohu řešit i tak, že se g vyjádří ve tvaru $g(x,y) = (x-1)^2 + (y+1)^2 + 2$.

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

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

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

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

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

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

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

16 HDR grafika (specializace PGVVH-PG, PGVVH-VPH)

Otázka se týká HDR (High Dynamic Range) grafiky, principů a aplikací.

1. Z jakých důvodů a v jakých oblastech grafiky se používá HDR grafika? Uveďte příklady, kdy nám může pomoci nejvíce.
2. Jak se HDR obraz reprezentuje v paměti počítače a v jakém formátu ho můžeme zapsat na disk? Naznačte postup pořízení HDR obrázku s pomocí fotoaparátu a stativu.
3. Jak se HDR obraz prezentuje pozorovateli na běžném (LDR) výstupním zařízení? (není potřeba uvádět nějaké komplikované postupy, jen popsat princip)

Nástin řešení 1. HDR grafika má schopnost reprezentovat mnohem větší dynamiku obrazu, s výhodou se použije pro fotografie proti slunci, proti světelným zdrojům, atp. HDR obraz obsahuje plnou informaci o velmi kontrastní scéně, což umožňuje rekonstruovat mnohem lepší obraz i pro reprodukci na LDR zařízení (běžný LCD monitor, běžný tisk, apod.). Viz odpověď na subotázku 3.

V 3D grafice umožňuje HDR technologie dosáhnout realistických odrazů na lesklých objektech, pokud budeme mít mapu okolí (environment map) pořízenou v HDR. "Slunce bude po odrazu na lesklém povrchu stále svítit velmi silně..."

2. Hodnoty pixelů jsou uloženy v plovoucí desetinné čárce (floating point). Diskové formáty buď přímo ukládají typ "float" bez komprese nebo se používá některá kompresní metoda. Příkladem je RGBE (Radiance), kde se pro každý pixel určí společný exponent a tři barevné složky (RGB) se již pak ukládají jen jako tři různé mantisy v tomto společném exponentu. RadianceHDR formát tak potřebuje pouze 4 byty na pixel (3x8 bitů R,G,B, 8 bitů společný exponent).

Další formáty: PFM (bez komprese), OpenEXR (ILM, open-source).

Pořízení HDR obrázku: fotoaparát s režimem M (manuál) nebo bracketingem, stativ, (relativně) statická scéna. Vyfotografuje se tatáž scéna opakovaně v co nejrychlejším sledu: zafixuje se ISO a clona, mění se jen čas expozice s krokem cca 1-2 EV. Tři až deset expozic obvykle stačí k tomu, aby se pokryla škála dynamiky scény. Nakonec se vhodným programem (Picturenaut, PhotoShop, GIMP) nechají obrázky složit do jediného HDR obrazu.

3. K převodu z HDR do LDR se použije některá metoda "tone mapping". Je potřeba zkomprimovat dynamiku HDR obrázku tak, aby byla zobrazitelná běžným displejem, přitom musí zůstat zřetelný lokální kontrast ve všech částech jasového rozsahu. Příklady: logaritmická transformace, sigmoida, pro zachování barevnosti je výhodné oddělit jasovou složku ("exponent" E z RGBE) a tu podrobit kompresi, originální barevnost (RGB) se nakonec do výsledného pixelu vrátí beze změny...

17 Phongův model odrazu světla (specializace PGVVH-PG, PGVVH-VPH)

Phong navrhl slavný model odrazu světla na povrchu tělesa ve 3D scéně. Půjde nám o tento model, ne o Phongovo stínování (interpolaci na povrchu tělesa).

1. Co za vstupní data potřebujeme pro výpočet barvy, kterou vidí pozorovatel v bodě A na povrchu tělesa?
2. Které parametry (v Phongově modelu) definují materiál povrchu tělesa? Můžete klidně použít svoje vlastní značení, vždy však popište, co daný parametr znamená a jaký je rozsah přípustných hodnot.
3. Z jakých složek se Phongův model skládá? Uveďte vzorce nebo alespoň jejich kvalitativní části. Popište, co jednotlivé složky znamenají a do jaké míry jsou fyzikálně správně.

Nástin řešení 1. Potřebujeme znát: souřadnici bodu A, normálový vektor v tomto bodě n, polohu pozorovatele V, polohu a intenzitu světelných zdrojů L_i a I_i a materiálové konstanty (viz bod 2.)

2. Materiálové konstanty: k_D koeficient difusního světla (0.0 až 1.0), k_S koeficient lesklého odrazu (0.0 až 1.0), k_A koeficient okolního světla (0.0 až 1.0), (mělo by být $k_D + k_S + k_A < 1.0$), h lesklý exponent (kladné velké číslo, např. mezi 10 a 5000), C_D vlastní/difusní barva materiálu (např. trojice $[R, G, B]$).

3. Phongův model odrazu: odražené světlo se skládá z difusní složky E_D (diffuse), lesklého odrazu E_S (specular), případně se přidává nefyzikální okolní složka E_A (ambient).

Základem difusní složky je $cos(\alpha)$, kde α je úhel dopadu světla od normály. Základem lesklého odrazu je $cos(\beta)^h$, kde β je odchylka pozorovacího směru od dokonale zrcadlového odrazu a h exponent odlesku (highlight). Difusní barva je barvou tělesa C_D, barva odlesku se obvykle ztotožňuje s barvou zdroje světla. Okolní složka je jen konstantní příspěvek s barvou tělesa $(k_A \cdot C_D)$, který se přičítá bez ohledu na jakékoli směry a úhly pohledu.

Fyzikální věrnost: pouze difusní složka E_D je správně (viz Lambertův zákon dokonale matného tělesa), ostatní jsou jen hrubé odhady.

Další detaily, vzorce a obrázky - viz prezentace z přednášek NPGR003 nebo jakákoli učebnice základů 3D grafiky.

18 GPU - data a shadery (specializace PGVVH-VPH)

Budeme se zabývat daty, které aplikace posílá do GPU a tzv. shadery (shaders), které programátoři moderních GPU musí připravit.

1. Napište, které typy shaderů se v běžném zobrazování 3D grafiky používají (neuvádějte shadery pro RT a GPGPU). U každého typu uveďte, v jaké části zobrazovacího řetězce je zařazen, a jaké hlavní funkce plní. Jsou některé typy shaderů povinné a nesmějí chybět v žádné aplikaci?
2. Jakou formou se předávají data z aplikace do GPU? Uvažujte opravdu všechny typy dat, vstupy shaderů, geometrická data scény, data popisující vzhled (materiály)... Jakým způsobem se všechna tato data do GPU předávají? (Když pomineme shadery, které by se také daly považovat za data, měli byste uvést minimálně tři formy předávání dat). U každé formy odlište, k jakému použití se hodí nejvíce, například, jak často se v aplikaci mění (nastaví se jednou provždy, mění se jednou za mnoho sekund, mění se každý snímek/frame, mění se mnohokrát za snímek/frame).
3. Kdybyste měli v zobrazovacím řetězci na GPU nějaký časově náročný výpočet (například pseudo-simulaci vodní hladiny nebo nějakou složitou procedurální texturu), jak byste se ho snažili optimalizovat? Do které části zobrazovací pipeline byste se ho snažili umístit? Doplňující/návodná otázka: které shadery na GPU spotřebovávají největší část výpočetního výkonu?

Nástin řešení 1. Shadery, jak jsou za sebou v zobrazovacím řetězci:

Vertex shader - zpracovává všechna data přiřazená k vrcholu. Je povinný. Musí spočítat minimálně souřadnice vrcholu v "clip space". Typicky provádí Modelovou (model), Pohledovou (view) a Projekční (projection) transformaci a všechna ostatní streamová data vrcholu předává dál, aby je mohly později použít ostatní shadery.

Hull shader - nepovinný, přijímá indexy a připravuje data pro teselační stupně.

Tessellation shaders - nepovinné dva stupně pro teselaci kreslené geometrie, pro celkem pravidelné topologie sítě. "Tesselation Control Shader" konfiguruje celkovou topologii sítě a "Tessellation Evaluation Shader" počítá polohy vrcholů.

Geometry shader - nepovinný, může ještě mnohem obecněji modifikovat kreslená primitiva, může vytvářet zcela nové objekty (třeba z každého vrcholu může vyrobit malou trojůhelníkovou síť - např. dvacetistěn). Má přístup k datům celého vstupního grafického primitiva, na výstup posílá také celá grafická primitiva.

Fragment shader - určuje barvu výsledného fragmentu (potenciálního pixelu), je povinný. Typicky k tomu využívá vhodné vstupní streamové veličiny (viz Vertex shader), globální konstanty (uniforms), textury, apod.

2. Tzv. Buffers - speciálně Vertex buffers (VB), obsahují streamové veličiny popisující jednotlivé vrcholy geometrie. Vertex shader dostane vždy jen data jednoho vrcholu.

Index buffers (IB) - obsahují indexy popisující grafická primitiva (ze kterých vrcholů se skládají). K těmto datům má přístup stupeň "primitive assembly", případně Hull shader, pokud je použit.

Textury - speciální buffery obsahující rastrová data, ke kterým mají přístup shadery. Textura je vždy přístupná jako celek, je k dispozici několik pokročilých interpolačních technik včetně MIP-mappingu (tj. HW se nám stará o sampling a interpolaci).

Uniforms/konstanty/Constant buffers - globální proměnné pro shadery. Pro jednu zobrazovací dávku (render call) jsou tyto hodnoty konstantní a mají k nim přístup všechna GPU jádra realizující daný výpočet. Typicky tam mohou být odkazy na

textury, souřadnice světelných zdrojů, transformační matice pro Vertex shader, materiálové konstanty pro Fragment shader, apod.

Jak často se mění:

VB a IB - je snahou, aby se nahrály do paměti GPU a pak už se nemusely měnit (třeba několik minut, podle charakteru aplikace).

Textury - pokud možno by se měly také vejít do paměti GPU, ideálně se mění jen při nějaké velké změně vykreslované scény (nový level hry, vrtulník vs. chůze po zemi, vstup do podzemního komplexu...).

Uniforms - některé se mění jen jednou za čas, některé přesně jednou za snímek/frame, některé souvisí s kreslenou dávkou nebo instancí modelu a musí se měnit mnohokrát za snímek.

3. Náročný výpočet je dobré buď provést jenom jednou (např. Compute shaderem) a výsledek si uložit do sdíleného buffe-ru/textury, odkud se pak bude jen číst.

Pokud to není možné, tak je dobré co nejvíce náročného počítání přenést do Vertex shaderu, protože ten se nevyvolává tak často. Nevýhodou je však, že se spočítaná veličina už pak jenom umí lineárně interpolovat do fragmentů.

Pokud ani to není možné, musí se výpočet nechat ve Fragment shaderu. Bude se spouštět pro každý pixel. To je taky odpověď na otázku, co nejvíc výpočetně zatěžuje GPU - Fragment shadery.

19 Reprezentace dat a systém souborů ext2 (specializace PVS)

Uvažujte systém souborů ext2. Následující (zjednodušené) definice struktur popisují formát adresářové položky a inode na disku:

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

struct ext2_inode {
  __le16 i_mode;         /* File mode */
  __le16 i_uid;          /* Owner Uid */
  __le32 i_size;         /* Size in bytes */
  __le32 i_ctime;        /* Creation time */
  __le16 i_gid;          /* Group Gid */
  __le16 i_links_count;  /* Links count */
  __le32 i_blocks;       /* Blocks count */
  __le32 i_flags;        /* File flags */
  __le32 i_block [EXT2_N_BLOCKS]; /* Pointers to blocks */
};

#define EXT2_NAME_LEN 255

struct ext2_dir_entry {
  __le32 inode;          /* Inode number */
  __le16 rec_len;        /* Directory entry length */
  __le16 name_len;       /* Name length */
  char   name [];        /* File name, up to EXT2_NAME_LEN */
};
```

(Typy `__le16` a `__le32` značí integer typy délky 16 a 32 bitů bez znaménka. Zápis `char name []` v definici struktury označuje pole znaků, jehož délka není definicí typu dána (flexible array member).)

1. Jaké požadavky existují na zarovnání struktur `ext2_inode` a `ext2_dir_entry` v paměti ? Je součástí některé struktury padding ? Odpovědi zdůvodněte.
2. Vyjmenujte všechny faktory vyplývající z těchto struktur, které omezují největší možnou velikost souboru v tomto systému souborů, a tuto velikost vyčíslete (stačí vzorec, není třeba číselná hodnota).

3. Napište obsah struktur `ext2_inode` a `ext2_dir_entry`, který popisuje soubor "abcd" o délce $1024 * 1024$ bajtů. Vynechejte informace, které se netýkají jména a délky souboru a jeho uložení na disku, naopak napište, pokud jsou informace o uložení souboru na disku obsaženy ještě v jiných strukturách. Předpokládejte velikost bloku 4096 bajtů. Pokud pro řešení potřebujete nějaké další konkrétní hodnoty (například čísla bloků), rozumně si je zvolte.

Nástin řešení

1. Struktura `ext2_inode` obsahuje pouze položky o velikosti 2 B a 4 B, její zarovnání v paměti je proto na násobek 4. Položky o velikosti 2 B následované položkami o velikosti 4 B jsou vždy vedle sebe v párech, tedy nebude potřeba padding. Struktura `ext2_dir_entry` obsahuje položky o velikosti 1 B, 2 B a 4 B, její zarovnání v paměti je proto na násobek 4. Položky jsou uspořádány s klesajícími nároky na zarovnání, tedy nebude potřeba padding.
2. Velikost je uvedena v bajtech v typu `__le32`, tedy bude omezena 2^{32} bajty, dále v blocích v typu `__le32`, toto omezení bude vždy méně striktní než to předchozí, a nakonec soubor může obsahovat nanejvýš tolik bloků, kolik se vejde do seznamu `i_block`, to je $12 + bs/4 + (bs/4)^2 + (bs/4)^3$ pro velikost bloku bs. Pro blok 4096 B vyjde omezení na zhruba miliardu bloků.
3. Předpokládejme přidělení čísla inode například 100 a prostoru například od čísla bloku 1000 dále.

```
struct ext2_inode
  __le32 i_size = 1024*1024;
  __le16 i_links_count = 1;
  __le32 i_blocks = (1024 * 1024) / 4096;
  __le32 i_block [EXT2_N_BLOCKS] = [1000, 1001 ... 1012, 0, 0];

ext2_dir_entry
  __le32 inode = 100;
  __le16 name_len = 4;
  char   name [] = ['a', 'b', 'c', 'd'];

block [1012] = [1013, 1014 ... 1512, 0 ...];
```

   Soubor bude mít celkem $1024 * 1024/4096 = 512$ datových bloků, budou to bloky 1000 až 1011 a pak 1013 až 1512, blok 1012 obsahuje čísla bloků 1013 až 1512.

20 Objektový návrh (specializace PVS)

Vaším úkolem je navrhnout datovou strukturu založenou na standardních kontejnerech jazyka C++ pro implementaci třídy s následujícím rozhraním:

```
class string_storage { public:
    std::size_t insert(const std::string&);
    std::size_t insert(std::string&&);
    std::size_t find(const std::string&) const;
    const std::string& get(std::size_t) const;
};
```

Funkce `insert` ukládají řetězec do datové struktury a vracejí *index*, kterým je řetězec identifikován. Indexy jsou přidělovány v souvislé řadě od 0. Pokud již v datové struktuře existuje řetězec se shodným obsahem, je vrácen jeho index a struktura se nemění.

Funkce `find` vrací index daného řetězce; pokud v datové struktuře neexistuje, hází výjimku.

Funkce `get` vrací řetězec s daným indexem; pokud je index neplatný, hází výjimku.

Podmínky:

- Každý řetězec smí být v datové struktuře uložen pouze v jedné kopii.
- Funkce `insert` a `find` musí pracovat v logaritmickém (nebo lepším) průměrném čase vzhledem k počtu uložených řetězců.
- Funkce `get` musí pracovat v konstantním čase.

Řešení uveďte ve formě deklarací datových položek třídy `string_storage`, doplněných komentářem a/nebo obrázkem. Uveďte kompletní implementaci metody `find`; implementaci ostatních metod a pomocných objektů (funktorů apod.) neuvádějte, pouze popište slovně použité triky a vlastnosti, které nejsou zřejmé.

Zodpovězte následující otázky:

1. Proč jsou v rozhraní třídy dvě metody `insert` a čím se budou lišit jejich implementace?
2. Součástí dokumentace kontejneru `std::unordered_map` je věta *References and pointers to either key or data stored in the container are only invalidated by erasing that element.* Vysvětlete, co obecně znamená invalidace referencí, ukazatelů a iterátorů a rozhodněte, zda je ve vašem řešení uvedená záruka využita.

Drobné syntaktické vady řešení nebudou penalizovány. Pokud si nejste jisti jmény typů a metod standardních kontejnerů, zvolte vhodné jméno a napište vysvětlení, co má daný typ či metoda dělat.

Nástin řešení Invalidace referencí apod. je důsledek implementace některých kontejnerů (`vector`, `deque` a `string`), které za určitých okolností přesouvají své elementy na jiná místa v paměti. Ostatní kontejnery data nepřesouvají a mohou tedy poskytovat záruku neinvalidace referencí a ukazatelů. (Záruka neinvalidace iterátorů je složitější, hashované kontejnery ji neposkytují, přestože data nepřesouvají.)

Řešení A:

```
std::unordered_map<std::shared_ptr<std::string>,std::size_t,Hash,KeyEqual> m_;
std::vector<std::shared_ptr<std::string>> v_;

std::size_t find(const std::string& s) const {
        auto p = std::make_shared<std::string>(s);
        auto it = m_.find(p);
        if (it == m_.end()) throw std::domain_error("Not␣found.");
        return it->second;
}
```

Funktory `Hash` a `KeyEqual` jsou zde nutné vzhledem k tomu, že defaultní chování na typu *std::shared_ptr* porovnává adresy a nikoliv hodnoty. Toto řešení nepotřebuje záruku stability iterátorů, je však méně prostorově efektivní než řešení B a C. Navíc je zde nutno buďto každý vyhledávaný řetězec nejprve zkopírovat do dynamicky alokovaného `std::shared_ptr<std::string>` (anebo použít polymorfní funktory podobně jako v řešení C).

Namísto dvou `std::shared_ptr` je možno použít dvojici `std::unique_ptr<std::string>` a `const std::string*`, což může odstranit nutnost alokace před vyhledáváním.

```
std::unordered_map<const std::string*,std::size_t,Hash,KeyEqual> m_;
std::vector<std::unique_ptr<std::string>> v_;

std::size_t find(const std::string& s) const {
        auto it = m_.find(&s);
        if (it == m_.end()) throw std::domain_error("Not␣found.");
        return it->second;
}
```

Druhá metoda `insert` slouží pro vkládání přesunem a v tomto řešení se bude lišit použitím `std::move` při předávání parametru do funkce `std::make_shared` resp. `std::make_unique`.

Toto řešení záruku stability iterátorů uvedenou v doplňkové otázce nevyužívá.

Řešení B:

```
std::unordered_map<std::string, std::size_t> m_;
std::vector<const std::string*> v_;

std::size_t find(const std::string& s) const {
        auto it = m_.find(s);
        if (it == m_.end()) throw std::domain_error("Not␣found.");
        return it->second;
}
```

21 SQL (specializace PVS)

V účetním systému byly za účelem evidence vystavených faktur vytvořeny relace

Zákaznik	(**Kód**, **DIČ**, Název, Adresa)	
Produkt	(**EAN**, Název, Kategorie, Vyrobce, Cena)	
Faktura	(**ČFaktury**, **VarSymbol**, KódZ, DatVyst)	KódZ ⊆ Zákazník.Kód
Položka	(**ČFaktury, EAN**, JCena, Počet)	ČFaktury ⊆ Faktura.ČFaktury
		EAN ⊆ Produkt.EAN

Každý klíč relace je vyznačen podtržením. Pokud je klíč složený (relace Položka), jsou všechny jeho atributy podtržené souvisle. Všechny cizí klíče jsou rovněž uvedeny.

1. Ve zvoleném konceptuálním jazyce (E-R, UML) vytvořte odpovídající konceptuální model respektující kardinality vztahů. Cizí klíče nemohou obsahovat NULL hodnotu.
2. Pro aplikaci byly napsány následující dotazy. Popište slovně, co dotazy vrací, případně by měly vracet. Pokud se domníváte, že jsou napsány v rozporu s pravidly SQL, popište, v čem je problém, a uveďte, jak dotaz opravit.

   2.1
   ```
   SELECT *
   FROM Produkt P
   WHERE Kategorie='Nápoje' AND Cena = MAX(Cena);
   ```

   2.2
   ```
   SELECT F.ČFaktury, SUM(P.JCena*P.Pocet) AS CelkCena
   FROM Faktura F, Polozka P
   WHERE P.ČFaktury=F.ČFaktury AND F.ČFaktury Like '24%'
   AND SUM(P.JCena*P.Pocet)>10000;
   ```

   2.3
   ```
   SELECT F.ČFaktury, F.VarSymbol, F.DatVyst,
   (SELECT COUNT(*) FROM Polozka P WHERE P.ČFaktury=F.ČFaktury)
   FROM Faktura F
   WHERE F.DatVyst>=DATE '2025-01-01' AND F.VarSymbol<>ČFaktury;
   ```

Následně bylo zjištěno, že je potřeba evidovat rovněž úhrady za faktury.
Byla proto přidána relace

Úhrada (**ČFaktury, ČTransakce, ČÚčtu**, Banka, Datum, PlatbaKč, ÚhradaKč, VarSymbol)
ČFaktury ⊆ Faktura.ČFaktury

Doménový expert určil, že pro přidanou relaci platí funkční závislosti

ČTransakce, ČÚčtu → Datum, PlatbaKč, VarSymbol
ČÚčtu → Banka

3 Jak byste schéma databáze s výše uvedenými pěti relacemi upravili a proč?

4 Napište nad výsledným schématem v jazyce SQL dotaz, který najde faktury, které dosud nemají nic uhrazeno.

Nástin řešení

1. Relace Zákazník, Produkt a Faktura odpovídají stejnojmenným entitám/třídám konceptuálního modelu. Relace položka má identitu určenou kombinací cizích klíčů do Faktury a Produktu. Odpovídá tedy M:N vztahu mezi těmito entitami/třídami. Ostatní sloupce relace jsou atributy daného vztahu. Cizí klíč Faktura.KódZ reprezentuje vztah N Faktur : (právě) 1 Zákazník, protože nesmí být NULL.
2. (a) Nalezení nejdražších nápojů (Produktů v kategorii Nápoj). WHERE klauzule vidí vždy jen jednu řádku a nelze v ní použít MAX. Místo =Max(Cena) je potřeba poddotaz =(SELECT MAX(Cena) FROM Produkt WHERE Kategorie='Nápoje') nebo podmínku přepsat pomocí >=ALL(SELECT Cena FROM Produkt WHERE Kategorie='Nápoje')

   (b) Faktury s čísly, začínajícími na 24 včetně celkové ceny, pokud tato přesahuje 10000. Opět nelze ve WHERE (ani v SELECT) použít SUM. Je potřeba před tuto část podmínky doplnit GROUP BY F.ČFaktury a podmínku na celkovou cenu dát do HAVING

   (c) Faktury vystavené v letošním roce spolu s počtem položek. Výstup je omezen jen na ty faktury, kde variabilní symbol neodpovídá číslu fakury. Dotaz je v pořádku

1. Aby byla databáze alespoň ve 3NF (zde BCNF), a ne v původní 1NF, je potřeba přidanou relaci dekomponovat pomocí uvedených závislostí na 3 relace (na jménech relací nezáleží, ale např.
   Účet (**ČÚčtu**, Banka)
   odštěpeno dle závislosti ČÚčtu → Banka
   Platba (**ČTransakce, ČÚčtu**, Datum, PlatbaKč, VarSymbol)
   odštěpeno dle závislosti ČTransakce, ČÚčtu → Datum, PlatbaKč, VarSymbol
   Platba (**ČFaktury, ČTransakce, ČÚčtu**, ÚhradaKč)
   Zbytek po odstranění atributů na pravých stranách závislostí
2. Např. SELECT * FROM Faktura F
   WHERE NOT EXISTS(SELECT * FROM Úhrada X WHERE X.ČFaktury=F.ČFaktury)

22 Unit testy (specializace PVS)

Mějme webovou aplikaci, jejíž součástí jsou diskusní vlákna (komentáře). V každém vláknu mohou vybraní uživatelé komunikovat pomocí jednoduchých textových zpráv. V serverové části se nachází následující fragmenty kódu (těla metod pro jednoduchost neuvádíme, předpokládejte jejich rozumnou implementaci).

```
class CommentMessage
{
    public string $id;
    public User $author;
    public DateTime $createdAt;
    public string $content;
}

class CommentThread
{
```

```
    /**
     * Returns all comments in the thread posted since given date and time.
     * @param DateTime|null $since Messages added after this time are returned.
     *                             If null, all messages are returned.
     * @return CommentMessage[] array of comment messages
     */
    public function getMessages(?DateTime $since): array

    /**
     * Add a new message to the thread.
     * @param User $author who wrote the message
     * @param string $content the body of the message
     * @return string ID of the newly created message
     */
    public function addMessage(User $author, string $content): string

    /**
     * Remove (delete) message from the thread.
     * @param string $id of the message
     * @throws NotFoundException
     */
    public function removeMessage(string $id): void
}
```

1. Navrhněte smysluplné unit testy na základě znalostí, které můžete odvodit z komentářů a signatur metod. Popište stručně (slovy) alespoň 5 test cases, ve kterých dohromady zavoláte každou metodu alespoň 2×. Jeden vybraný test case popište podrobněji pomocí pseudokódu (netřeba znát detaily PHP, stačí přibližná syntax). Pokud použijete funkce, o kterých předpokládáte, že existují v PHP nebo v použitém frameworku na testy, dostatečně jasně je pojmenujte, nebo opatřete komentářem popisujícím jejich sémantiku.

2. Stručně (nejlépe jednou větou) popište termín "pokrytí kódu" (code coverage). Napište, zda vaše testy z bodu 1. mají (či nemají) 100% pokrytí (těl testovaných metod), a vaše tvrzení podložte argumenty.

3. Předpokládejme, že metoda `addMessage` obsahuje následující řádky kódu:

```
foreach ($recipients as $recipient) {
        $this->mailSender->sendNofitication($recipient->getEmail(), "New␣comment:␣$content");
}
```

Naznačte, jak tuto funkcionalitu vhodně testovat, aniž by bylo nutné zřizovat specializovaný mail server.

Nástin řešení Příklady testů:

- Zavolám `getMessage()` a ověřím, že vrátí prázdné pole.
- Zavolám `addMessage()`, následně zavolám `getMessage()` a ověřím, že obsahuje jednu zprávu a její obsah a autor odpovídá tomu, co jsem vložil.
- Zavolám `addMessage()`, následně zavolám `getMessage()` a ověřím, že obsahuje jednu zprávu, zavolám `removeMessage()` na ID, které jsem dostal při vložení, zavolám `getMessage()` a ověřím, že tam opět není nic
- Zavolám `removeMessage()` na nesmyslné ID a ověřím, že hodí výjimku
- Zavolám `addMessage()`, počkám vteřinu, uložím si aktuální čas, opět zavolám `addMessage()`, následně zavolám `getMessage()` s uloženým časem a ověřím, že vrátí pouze druhou zprávu.

Většina testovacích frameworků umožňuje před-připravit testovací data. Toto není zmíněno v zadání, pokud by ale bylo rozumně popsáno, je to zcela přípustné. Např. poslední testovací scénář by mohl být změněn na "Nechám si připravit vlákno s dvěma zprávami — jedna bude 3 dny stará, jedna bude den stará. Následně zavolám `getMessage()` s datem před 2 dny a ověřím, že vrátí pouze tu novější zprávu."

Implementace prvního scénáře:

```
public function testGetEmptyThread()
{
    $thread = $this->prepareTestingThread();
    Assert::equal([], $thread->getMessage());
}
```

Podstatné je zejména ukázání použití testovacích assertions.

2. Jeden možný popis: "Pokrytí je relativní poměr řádek testovaného kódu, které se spustí alespoň jednou během testů, a celkový počet řádek imperativního kódu testovaného programu/modulu/třídy...".

Úvaha o 100% pokrytí: Jsou dvě možné odpovědi — ano a ne. Pokud ano, je argumentem stručný kategorický rozbor všech přípustných kombinací vstupních hodnot a stavů vlákna a odkaz na pokrytí těchto případů v testech. Pokud ne (což bude vzhledem k počtu testů asi častější případ), očekává se příklad vstupu, který není pokrytý testy a kde lze předpokládat jiné chování v těle metody (tj. existuje větev, která se v testech nikdy nezavolá).

Příklady z bodu 1. spíše nebudou mít 100% pokrytí. Předpokládejme, že `removeMessage()` má na začátku test, že ID je v rozumném formátu (např. UUID). Čtvrtý test tedy skončí v této větvi (hodí výjimku) a třetí test projde do zdárného konce. Pokud by ale někdo použil např. `removeMessage()` 2x za sebou na platné ID, tak první volání zprávu smaže a druhé ji nenajde v databázi. Skončí (tj. hodí výjimku) ale v jiné větvi než test 4 a tato větev není pokrytá.

3. Nejčastější je použití techniky zvané *Mocking*, která je také často přímo podporována v testovacích frameworcích. Vyrobíme mock třídy MailSender (falešný objekt, který má stejný interface) a podstrčíme jej do členské proměnné `$mailSender` v instanci `CommentThread`. Mock objekt má navíc zpravidla interface, pomocí kterého nastavíme, jaká volání metod bude test provádět, případně jaké hodnoty má mock vracet. Testovací framework na konci ověří, že na mock objektu byly skutečně zavolány všechny metody dle očekávání, jinak test selže.

Pro zcela kompletní zodpovězení otázky ještě uveďme, že mock objekt musí být podstrčitelný do objektu komentářového vlákna. K tomu nám typicky pomůže Dependency injection koncept, který se běžně používá u komponentových systémů (členská proměnná `$mailSender` je buď public, nebo se její hodnota nastavuje na objekt, který vlákno obdrží v konstruktoru). Detaily DI jsou už ale nad rámec této otázky.

23 Systém souborů ext2 (specializace SP)

Uvažujte systém souborů ext2. Následující (zjednodušené) definice struktur popisují formát adresářové položky a inode na disku:

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

struct ext2_inode {
  __le16 i_mode;         /* File mode */
  __le16 i_uid;          /* Owner Uid */
  __le32 i_size;         /* Size in bytes */
  __le32 i_ctime;        /* Creation time */
  __le16 i_gid;          /* Group Gid */
  __le16 i_links_count;  /* Links count */
  __le32 i_blocks;       /* Blocks count */
  __le32 i_flags;        /* File flags */
  __le32 i_block [EXT2_N_BLOCKS]; /* Pointers to blocks */
};

#define EXT2_NAME_LEN 255

struct ext2_dir_entry {
  __le32 inode;          /* Inode number */
  __le16 rec_len;        /* Directory entry length */
  __le16 name_len;       /* Name length */
```

```
  char   name [];        /* File name, up to EXT2_NAME_LEN */
};
```

(Typy `__le16` a `__le32` značí integer typy délky 16 a 32 bitů bez znaménka. Zápis `char name []` v definici struktury označuje pole znaků, jehož délka není definicí typu dána (flexible array member).)

1. Vyjmenujte všechny faktory vyplývající z těchto struktur, které omezují největší možnou velikost souboru v tomto systému souborů, a tuto velikost vyčíslete (stačí vzorec, není třeba číselná hodnota).
2. Napište obsah struktur `ext2_inode` a `ext2_dir_entry`, který popisuje soubor "abcd" o délce $1024 * 1024$ bajtů. Vynechejte informace, které se netýkají jména a délky souboru a jeho uložení na disku, naopak napište, pokud jsou informace o uložení souboru na disku obsaženy ještě v jiných strukturách. Předpokládejte velikost bloku 4096 bajtů. Pokud pro řešení potřebujete nějaké další konkrétní hodnoty (například čísla bloků), rozumně si je zvolte.
3. Popište v detailu jednotlivých položek, jaké všechny struktury systému souborů na disku se změní a jak, pokud se k obsahu souboru "abcd" přidá dalších 1024 bajtů a pak se k tomuto souboru vytvoří hard link "efgh".

Nástin řešení

1. Velikost je uvedena v bajtech v typu `__le32`, tedy bude omezena 2^{32} bajty, dále v blocích v typu `__le32`, toto omezení bude vždy méně striktní než to předchozí, a nakonec soubor může obsahovat nanejvýš tolik bloků, kolik se vejde do seznamu `i_block`, to je $12 + bs/4 + (bs/4)^2 + (bs/4)^3$ pro velikost bloku bs. Pro blok 4096 B vyjde omezení na zhruba miliardu bloků.
2. Předpokládejme přidělení čísla inode například 100 a prostoru například od čísla bloku 1000 dále.

   ```
   struct ext2_inode
     __le32 i_size = 1024*1024;
     __le16 i_links_count = 1;
     __le32 i_blocks = (1024 * 1024) / 4096;
     __le32 i_block [EXT2_N_BLOCKS] = [1000, 1001 ... 1012, 0, 0];

   ext2_dir_entry
     __le32 inode = 100;
     __le16 name_len = 4;
     char   name [] = ['a', 'b', 'c', 'd'];

   block [1012] = [1013, 1014 ... 1512, 0 ...];
   ```

   Soubor bude mít celkem $1024 * 1024/4096 = 512$ datových bloků, budou to bloky 1000 až 1011 a pak 1013 až 1512, blok 1012 obsahuje čísla bloků 1013 až 1512.
3. Připojení dalších 1024 B bude vyžadovat další datový blok, tedy aktualizuje se obsah bloku 1012 a alokační bitmapa, ve které bude potřeba vyznačit obsazení dalšího datového bloku, v inode se pak aktualizuje velikost. Hard link bude potřebovat další strukturu `ext2_dir_entry`, která bude ukazovat opět na inode 100, v inode 100 se změní pole `links_count` z 1 na 2. Vytvořená struktura bude uložena v nějakém adresáři, v inode tohoto adresáře se tak změní velikost a pokud bude potřeba alokovat další blok, struktury související s tímto adresářem se změní podobně jako struktury související se souborem z předchozího kroku.

24 Koherence cache (specializace SP)

Uvažujte následující fragment kódu, funkce `threadA` a `threadB` jsou vykonávány dvěma vlákny:

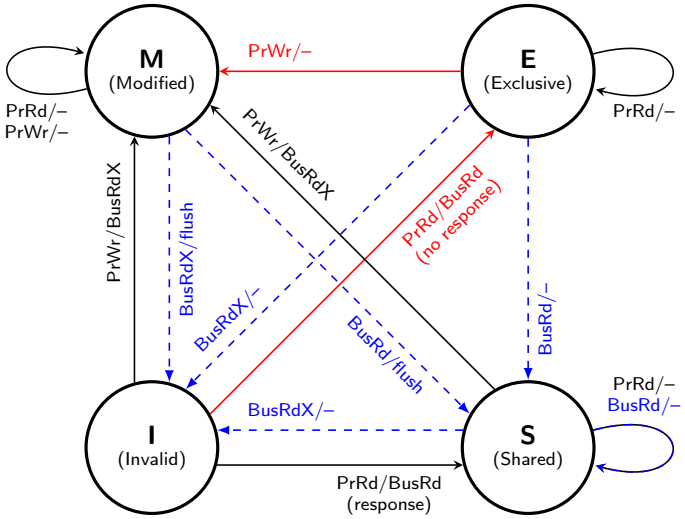
```
int taskDoneCountA = 0;
int taskDoneCountB = 0;

void *threadA (void *) {
  while (true) {
    doTaskA ();
    taskDoneCountA ++;
  }
}
```

```
void *threadB (void) * {
  while (true) {
    doTaskB ();
    taskDoneCountB ++;
  }
}
```

1. Popište, za jakých okolností dojde v uvedeném fragmentu kódu k false sharing (těla funkcí `doTaskA` a `doTaskB` neuvažujte).
2. Popište (všechny) přechody, které protokol MESI vykonává během takového false sharing (včetně operací sběrnice).
3. Navrhněte úpravu uvedeného fragmentu kódu, která odstraní false sharing, a vysvětlete její funkci.

Jako pomůcku můžete použít přechodový diagram protokolu MESI:



Nástin řešení

1. Vlákna musí běžet na různých jádrech procesoru (běh vláken na jednom jádře - alternující i hyper threading - sdílí cache a tedy nemůže způsobit false sharing), proměnné `taskDoneCountA` a `taskDoneCountB` musí být v téže cache line.
2. Předpokládejme, že operace `++` se pro účely práce s pamětí rozdělí na nezávislé čtení a zápis. Cache line s oběma proměnnými bude na tom jádře, které jí naposledy modifikovalo, ve stavu M, na druhém ve stavu I. Pokud bude dále pokračovat vlákno s cache line ve stavu M, čtení i zápis se vykonají přímo nad cache line, beze změny stavu cache line a bez operace sběrnice (černá smyčka u M při PrRd a PrWr). Pokud bude dále pokračovat vlákno s cache line ve stavu I, čtení vyvolá na druhém jádře zápis modifikované cache line do paměti (modrý přechod z M do S při BusRd), na prvním jádře se dokončí čtení a přechod do stavu S (černý přechod z I do S při PrRd). Následující zápis vyvolá na druhém jádře přechod do stavu I (modrý přechod z S do I při BusRdX), na prvním jádře se dokončí zápis a přechod do stavu M (černý přechod z S do M při PrWr). Možné je i proložení operací obou vláken, při kterém dojde ke čtení ve stavu S beze změny stavu cache line a bez operace sběrnice (černá smyčka u S při PrRd).
3. Je potřeba zajistit, že proměnné `taskDoneCountA` a `taskDoneCountB` nebudou v téže cache line. Toho lze docílit direktivou překladače vyžadující zarovnání na velikost cache line, pokud taková direktiva není k dispozici, pak umístěním jedné nebo obou proměnných do struktur, jejichž velikost vyloučí, že by se obě proměnné daly umístit v paměti dostatečně blízko sebe. Naopak nestačí prosté oddělení deklarací obou proměnných vložením dalších dat nebo kódu, protože proces překladu nemusí dodržet pořadí deklarací.

25 Překladače (specializace SP)

Napište bezkontextovou gramatiku popisující jazyk JSON minimálně v rozsahu, který je použit v následující ukázce:

```
{
    "name": "John Smith",
```

```
    "is_alive": true,
    "age": 27,
    "address": {
        "street_address": "21 2nd Street",
        "city": "New York"
    },
    "phone_numbers": [
        { "type": "home", "number": "212 555-1234" },
        { "type": "office", "number": "646 555-4567" }
    ],
    "fax_numbers": [],
    "children": [ "Catherine", "Thomas", "Trevor" ],
    "spouse": null
}
```

Sepište seznam terminálů použitých v gramatice s krátkým vysvětlením jejich významu, lexikální analýzu ale neřešte.

Gramatika musí být v klasickém formátu, rozšíření jako N_{opt} nebo regulární výrazy nejsou povolena.

Gramatika musí být jednoznačná.

Věnujte pozornost správnému umístění čárek pouze mezi elementy seznamu a prázdným seznamům.

Nástin řešení

$$
\begin{aligned}
STRING &= \text{string literal} \\
NUMBER &= \text{number literal} \\
TRUE &= \texttt{true} \\
FALSE &= \texttt{false} \\
NULL &= \texttt{null} \\
COMMA &= , \\
COLON &= : \\
LBRA &= [\\
RBRA &=] \\
LCUR &= \{ \\
RCUR &= \}
\end{aligned}
$$

$$
\begin{aligned}
Atomic &\rightarrow STRING \\
Atomic &\rightarrow NUMBER \\
Atomic &\rightarrow TRUE \\
Atomic &\rightarrow FALSE \\
Atomic &\rightarrow NULL \\
ElementList &\rightarrow ElementList\ COMMA\ Element \\
ElementList &\rightarrow Element \\
Array &\rightarrow LBRA\ ElementList\ RBRA \\
Array &\rightarrow LBRA\ RBRA \\
FieldList &\rightarrow FieldList\ COMMA\ FieldElement \\
FieldList &\rightarrow FieldElement \\
Struct &\rightarrow LCUR\ ElementList\ RCUR \\
Struct &\rightarrow LCUR\ RCUR \\
FieldElement &\rightarrow STRING\ COLON\ Element \\
Element &\rightarrow Atomic \\
Element &\rightarrow Array \\
Element &\rightarrow Struct \\
S &\rightarrow Struct
\end{aligned}
$$

26 Paralelní programování (specializace SP)

Následující úryvek kódu představuje jádro Floyd-Warshallova algoritmu pro výpočet vzdáleností v orientovaném grafu:

```
void fw(/*...*/ dist)
{
  // dist is a N*N array of distances initialized to
  // - zeros on the diagonal
  // - the non-negative length of the edge where an edge exists
  // - MAX_INT/2 otherwise
  for (k = 0; k < N; ++k)
    for (i = 0; i < N; ++i)
      for (j = 0; j < N; ++j)
      {
        if ( dist[i][j] > dist[i][k] + dist[k][j] )
          dist[i][j] = dist[i][k] + dist[k][j];
       }
  // output: dist contains minimum distances
}
```

Navrhněte a popište způsob, jak v tomto algoritmu využít schopnost procesoru vykonávat více vláken paralelně. Zvolte vhodné nástroje pro paralelizaci, která jsou součástí standardu některého z jazyků C++, C# a Java, a upravte s jejich pomocí uvedený úryvek kódu do paralelní podoby.

Předpokládejte, že nejvhodnější počet vláken pro daný hardware je určen proměnnou `T`.

Pomocné funkce, které přímo neobsahují prvky původního algoritmu ani synchronizační primitiva, nemusíte uvádět, stačí slovně popsat jejich účel.

Z hlediska přístupu do pole `dist` platí jediné omezení: Pokud různá vlákna paralelně přistupují k témuž prvku, musí se jednat o operace čtení.

Duplikace pole `dist` nebo jeho částí není žádoucí.

Je pro paralelizaci významné pozorování, že prvky pole `dist` budou vždy nezáporné? Pokud ano, kde se tato skutečnost v navržené úpravě uplatní?

Nástin řešení

```
void fw(/*...*/ dist)
{
  auto [T1, T2] = approx_sqrt(T); // approx T1*T2 == T
  auto ranges1 = divide_range(std::ranges::iota_view(0, N), T1); // divide (0, N) into T1 subranges
  auto ranges2 = divide_range(std::ranges::iota_view(0, N), T2); // divide (0, N) into T2 subranges

  for (k = 0; k < N; ++k)
  {
    std::vector<std::future<void>> fv;
    for (auto a : ranges1)
      for (auto b : ranges2)
        fv.push_back(std::async([a,b,k,&dist](){
          for (auto i : a)
            for (auto j : b)
            {
              if ( dist[i][j] > dist[i][k] + dist[k][j] )
                dist[i][j] = dist[i][k] + dist[k][j];
            }
        }));
    for (auto && f : fv)
      f.wait();
  }
}
```

Principem jakékoli paralelizace tohoto algoritmu musí být sekvenční provedení vnějšího cyklu, paralelizovat lze pouze vnitřní dva cykly.

Nezápornost `dist` zaručuje, že podmínka `( dist[i][j] > dist[i][k] + dist[k][j] )` nemůže být nikdy splněna pro `(i == k)` ani `(j == k)`, což znamená, že sloupec `k` ani řádek `k` nebudou zapisovány, tudíž mohou být čteny i v jiných vláknech.

Dělení do `T1*T2` přibližně čtvercových bloků není nutné, je možné i dělení na `T` řádek nebo `T` sloupců. Čtverce však umožňují řešit případ `(N < T)`.

27 DPLL a párování (specializace UI-SU, UI-ZPJ)

Nechť $G = (V, E)$ je neorientovaný graf s n vrcholy. Perfektní párování v grafu G je podmnožina hran $M \subseteq E$ taková, že každý vrchol je incidentní s právě jednou hranou z M.

1. Navrhněte formuli ve výrokové logice, která je splnitelná, právě když graf G má perfektní párování. Formuli popište obecně v konjunktivní normální formě (CNF) pro libovolný graf G.
2. Vysvětlete pojmy *čistý literál* (Pure symbol heuristics) a *jednotková klauzule* (Unit clause heuristics) a jejich použití v rámci algoritmu DPLL pro hledání splnitelných ohodnocení CNF formulí.
3. Napište formuli z bodu 1. pro cestu na čtyřech vrcholech a, b, c, d. Na této formuli demonstrujte použití heuristik z předchozího bodu k nalezení splňujícího ohodnocení.

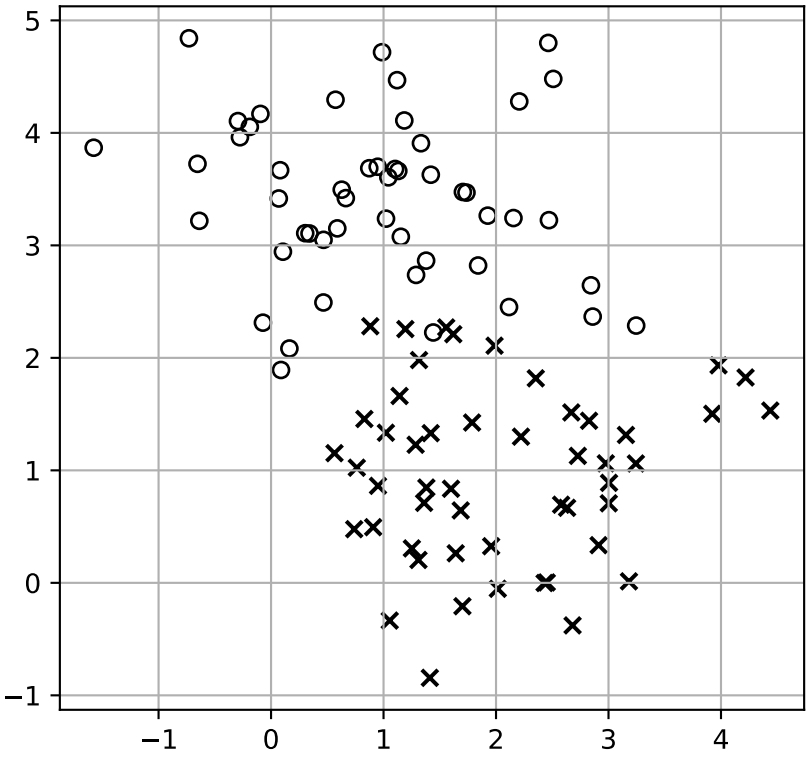
Nástin řešení

1. Formule používá výrokové proměnné m_e pro $e \in E$, které vyjadřují, že hrana e je v perfektním párování. Pro každý vrchol u a jeho incidentní hrany $e_1, \ldots, e_k$ klauzule:
   - Každý vrchol je incidentní s alespoň jednou hranou $m_{e_1} \vee \cdots \vee m_{e_k}$.
   - Pro každé dvě různé hrany e_i, e_j máme klauzuli: $\neg m_{e_i} \vee \neg m_{e_j}$.

2. Jednotková klauzule je klauzule s jednou proměnnou a tato klauzule vynucuje ohodnocení dané proměnné, které můžeme dosadit do zbytku formule. Čistá proměnná má buď všechny výskyty pozitivní nebo všechny výskyty negativní, a proto můžeme tuto proměnnou též z formule eliminovat.
3. Stačí zapsat formuli výše pro danou cestu. Z čistých klauzulí m_{ab} a m_{cd} musí být hrany ab a cd v párování a z klauzule $\neg m_{ab} \vee \neg m_{bc}$ po eliminaci m_{ab} dostáváme, že bc není v párování.

28 k-nejbližších sousedů (specializace UI-SU, UI-ZPJ)

1. Popište metodu k-nejbližších sousedů pro klasifikaci i pro regresi. Jak probíhá trénování a předpovídání?
2. Mějme data se dvěma atributy, která chceme klasifikovat do dvou tříd ($\circ$, $\times$) znázorněná na obrázku níže. Uvažujme $k = 3$ (tedy používáme tři nejbližší sousedy). Jak bude klasifikován nový vstup $(3, 2)$? Změnil by se výsledek klasifikace pokud změníme hodnotu k?
3. Popište, jakým způsobem můžeme nastavit vhodnou hodnotu k pro naše data.



Nástin řešení

1. Při tránování se pouze uloží trénovací data. Při klasifikaci/regresi se vyhledá k nejbližších sousedů. Pro klasifikaci se použije nejčastější třída těchto sousedů, pro regresi se použije průměr hodnot nejbližších sousedů.
2. Bod bude klasifikován jako $\circ$, neboť většina ze 3 nejbližších sousedů patří do této třídy. Pokud zvýšíme k, mnoho bodů s třídou $\times$ se dostane mezi nejbližší sousedy a tím se změní klasifikace (např. pro $k = 7$).
3. Vhodnou hodnotu k je možné nastavit například pomocí křížové validace.

29 Logistická regrese pro binární klasifikaci (specializace UI-SU, UI-ZPJ)

1. Popište, jakým způsobem můžeme použít logistickou regresi pro binární klasifikaci. Máme k dispozici dataset se vstupními příznaky $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ a s cílovými hodnotami $\boldsymbol{t} = (t_1, \ldots, t_N)$.

   Jak vypadá predikce modelu? Jak budeme postupovat, abychom mohli nezávisle určit očekávanou chybu modelu na neviděných datech?

2. Jak je definována ztrátová funkce? Odvoďte její derivaci a ukažte, jakým způsobem lze použít gradientní metodu (stochastic gradient descent) pro trénování.

3. Jakým způsobem zabráníte přeučení při stochastic gradient descentu?

Nástin řešení

1. Model: $p(C_1) = \sigma(\boldsymbol{x}^T \boldsymbol{w})$, predikce: if (y > 0.5) 1 else 0.

   Postup pro odhad chyby: Rozdělíme si data na trénovací, validační a testovací část. Validační data použijeme pro ladění hyperparametrů, testovací pro reportování výsledku na neviděných datech.

2. Chybová funkce: negative log-likelihood $E(\boldsymbol{w}) = \frac{1}{N} \sum_i -\log(p(C_{t_i} \mid \boldsymbol{x}_i; \boldsymbol{w}))$, dále rozepíšeme podle Bernoulliho distribuce a můžeme derivovat podle $\boldsymbol{w}$.

   Derivace vyjde $\nabla_{\boldsymbol{w}} E(\boldsymbol{w}) = (\sigma(\boldsymbol{x}^T \boldsymbol{w}) - t)\boldsymbol{x}$

   SGD: Inicializujeme váhy (buď náhodně nebo nulami), iterujeme přes dataset, v každém kroku odečteme $\alpha \cdot \nabla E(\boldsymbol{w})$.

3. Přeučení můžeme bránit: L^2 regularizací, pomocí early stopping (sledujeme validační chybu a přestaneme s trénováním, když začne růst), vhodným learning rate schedule.

30 Bayesovské učení (specializace UI-SU)

Supermarket bere bedny jablek od tří dodavatelů, polovinu beden od A, třetinu od B a šestinu od C. Bedny od sebe nelze pohledem rozeznat. Jablka jednotlivých dodavatelů se mírně liší průměrem – mají 9cm, 10cm, nebo 11cm. Vybrali jsme náhodně vzorky jablek od jednotlivých dodavatelů a spočítali jsme, jak běžné jsou různé velikosti jablek u různých dodavatelů. Výsledky tohoto měření, počty jablek různých velikostí rozdělené dle jednotlivých dodavatelů, jsou v tabulce níže.

Dodavatel	9cm	10cm	11cm
A	5	4	1
B	9	8	3
C	1	8	1

1. Uveďte maximálně věrohodné odhady pravděpodobností velikosti jablka pro jednotlivé dodavatele.

2. Definujte pojem *maximálně věrohodná hypotéza.*

3. Z téže bedny jsme náhodně vzali dvě jablka a změřili je – jedno mělo průměr 10cm a druhé průměr 11cm. Která z hypotéz: „Jablka jsou od A.“, „Jablka jsou od B.“, „Jablka jsou od C“ je maximálně věrohodná? Odpověď zdůvodněte.

4. Jaká je *bayesovsky optimální predikce* pravděpodobnosti velikosti 10cm pro další jablko ze stejné bedny (při předchozích pozorovnáních 10cm a 11cm jablek z této bedny)? Svou odpověď zdůvodněte.

Nástin řešení

1. Velikosti máme v diskrétních intervalech. Maximálně věrohodné odhady diskrétního rozložení jsou podíl četností, tj. pravděpodobnosti jsou:

$P(velikost\|h)$	9cm	10cm	11cm
A	.5	.4	.1
B	.45	.4	.15
C	.1	.8	.1

2. Maximálně věrohodná hypotéza po pozorování 10 a 11 cm je C dávající pravděpodobnost pozorování 8% oproti 4% A a 6% B, viz třetí sloupec v tabulce níže.

3. Spočteme aposteriorní pravděpodobnost jednotlivých hypotéz a tou vážíme jejich predikce, predikujeme 50% pro velikost 10 cm, výpočet viz tabulka:

h	$P(h)$	$P(data\|h)$	$P(h, data)$	$P(h\|data)$	predikce 10cm $*P(h\|data)$
A	.5	.04	.02	3/8	$.4 * 3/8$
B	1/3	.06	.02	3/8	$.4 * 3/8$
C	1/6	.08	.04/3	2/8	$.8 * 2/8$
		$P(data) = .16/3$			$\Sigma = 0.5$

31 Jazykové modely (specializace UI-ZPJ)

1. Co je jazykový model?
2. K čemu se při výpočtu jazykového modelu používají trénovací data? Napište vzorec pro predikci v rámci n-gramového jazykového modelu a vysvětlete, jak se jazykový model odhaduje metodou maximální věrohodnosti.
3. Proč je důležité vyhlazování jazykového modelu? Co je principem vyhlazování?

Nástin řešení

1. Jazykový model je pravděpodobnostní model přirozeného jazyka, který slouží pro odhad a porovnávání pravděpodobnosti různých sekvencí slov.
2. Trénovací data se používají pro odhad pravděpodobnosti pozorovaných sekvencí slov na základě jejich četností.

$$P(W) = \prod_{i=1,\ldots,d} P(w_i \mid w_{i-n+1}, w_{i-n+2}, \ldots, w_{i-1})$$

   Podmíněná pravděpodobnost výskytu daného slova se odhaduje jako podíl četností n-gramů pozorovaných v trénovacích datech.
3. Vyhlazování je formou regularizace jazykových modelů a významně zlepšuje robustnost jazykových modelů. Vyhlazováním se eliminují nulové odhady pravděpodobnosti výskytu sekvencí slov, které vznikají v důsledku řídkosti trénovacích dat.