Bakalářské zkoušky (příklady otázek)

2024-06-25

1 Algoritmy rozděl a panuj (společné okruhy)

1) Mějme algoritmus A, který na datech velikosti n udělá $T(n)$ elementárních operací. Algoritmus A je rekurzivní a pracuje takto (a, c, x, y jsou přirozená čísla, $a > 0$, $c > 1$):

a) Udělá $\Theta(n^x)$ elementárních operací, aby ze vstupních dat vybral a podmnožin velikosti n/c.

b) Rekurzivně pustí sám sebe na každou z vybraných podmnožin dat (pokud má velikost alespoň c, pro data menší velikosti vyřeší úlohu v konstantním čase).

c) Udělá $\Theta(n^y)$ elementárních operací, aby všechna řešení z bodu b) spojil do řešení původní úlohy na datech velikosti n.

Určete (bez důkazu) asymptoticky těsný odhad funkce $T(n)$ v závislosti na parametrech a, c, x, y.

2) Nechť X a Y jsou pole délky n, každé obsahující *setříděnou* posloupnost n přirozených čísel. Navrhněte a popište algoritmus s časovou složitostí $O(\log n)$, který najde medián (jeden z mediánů) všech $2n$ čísel obsažených v polích X a Y. Dokažte metodou z bodu 1), že složitost Vašeho algoritmu je opravdu $O(\log n)$.

Nástin řešení 1) Úloha je řešitelná pomocí Master Theoremu, tj. Kuchařky. V algoritmu A je společná režie $\Theta(n^x + n^y) = \Theta(n^{\max\{x,y\}})$ při každém rekurzivním volání. Z toho vychází rekurzivní rovnice $T(n) = a \cdot T(n/c) + \Theta(n^{\max\{x,y\}})$ pro časovou složitost řešitelná pomocí Master Theoremu. Zavedeme $d = \max\{x, y\}$. Řešení rovnice se dělí na 3 případy:

1. $T(n) = \Theta(n^d)$ pro $a/c^d < 1$;
2. $T(n) = \Theta(n^d \log n)$ pro $a/c^d = 1$;
3. $T(n) = \Theta(n^{\log_c a})$ pro $a/c^d > 1$.

2) V jednotlivých krocích algoritmu porovnáváme mediány obou polí X a Y, například prvky $x = X[\lfloor (n-1)/2 \rfloor]$ a $y = Y[\lfloor (n-1)/2 \rfloor]$ při indexování od 0.

Pokud $x < y$, můžeme odstranit levou polovinu X (prvky vlevo od x) a pravou polovinu Y (prvky vpravo od y) a na ponechané prvky voláme rekurzi. Je velmi podstatné, aby počet odstraněných prvků z X (které jsou všechny garantovaně menší než hledaný medián) byl stejný jako počet odstraněných prvků z Y (které jsou všechny garantovaně větší než hledaný medián). Tato vlastnost zajistí, že medián prvků které vstupují do rekurzivního volání je také mediánem původní množiny prvků.

Pokud $x > y$, pak odstraňujeme symetricky levou polovinu Y a pravou polovinu X a voláme rekurzi.

Pokud $x = y$, pak je x medián (jeden z mediánů).

Přístup k prvkům x a y, jejich porovnání a posunutí indexů (vymezujících aktuální části X a Y) zvládneme v konstantním čase $\Theta(1) = \Theta(n^0)$. Pro $n < 3$ dopočítáme medián v konstantním čase.

Vychází rovnice $T(n) = 1 \cdot T(n/2) + \Theta(1)$, tj. $a = 1$, $c = 2$, $d = 0$. V Kuchařce vyjde $a/c^d = 1/2^0 = 1$, tj. případ 2 s řešením $\Theta(n^0 \log n) = \Theta(\log n)$.

2 Semafor (společné okruhy)

Knihovna `System.Threading` jazyka C# obsahuje třídu `Semaphore` s metodami odpovídajícími operacím klasického semaforu, P (`WaitOne`) a V (`Release`). Objekt vytvořený pomocí `new Semaphore(0, 1)` odpovídá binárnímu semaforu inicializovanému nulou. Pokud více než jedno vlákno čeká na tentýž semafor, pořadí, ve kterém budou vlákna spouštěna po uvolnění semaforu, není specifikováno.

Program vpravo má produkovat výstup `OneTwoThreeFourFive`, přitom liché části výstupu mají být vypisovány funkcí `f1` a sudé části funkcí `f2`, přičemž tyto funkce běží v různých vláknech.

Synchronizace, implementovaná pomocí semaforu, téměř funguje, ale obsahuje časově závislé chyby (race conditions).

1. Popište scénář, který vede k jinému výstupu než `OneTwoThreeFourFive`, nebo končí uváznutím (deadlock). Scénař zapište jako posloupnost čísel řádek, přičemž každé číslo reprezentuje *ukončení* příkazu na dané řádce. Pouze vnitřky funkcí `f1` a `f2` jsou relevantní.
2. Existovaly by v tomto kódu časově závislé chyby i v případě, že by knihovna C# garantovala FIFO pořadí spouštění vláken čekajících ve `WaitOne`?
3. Napište řešení, které neobsahuje časově závislé chyby a spolehlivě vypisuje požadovaný výstup `OneTwoThreeFourFive`. Použijte přitom dva semafory a pouze jejich funkce `Release()` and `WaitOne()`. Řešení nesmí používat žádné jiné proměnné nebo objekty sdílené mezi vlákny kromě těchto dvou semaforů. Volání `Console.Write` musejí samozřejmě zůstat tam, kde jsou.

(Jazyk C# je v této otázce použitý pouze jako generický zástupce běžných programovacích jazyků, otázka ani řešení s jazykem C# jako takovým nesouvisí.)

```
 1  class Program
 2  {
 3      private static Semaphore sem;
 4
 5      private static void f1()
 6      {
 7          Console.Write("One");
 8          sem.Release();
 9          sem.WaitOne();
10          Console.Write("Three");
11          sem.Release();
12          sem.WaitOne();
13          Console.Write("Five");
14      }
15
16      private static void f2()
17      {
18          sem.WaitOne();
19          Console.Write("Two");
20          sem.Release();
21          sem.WaitOne();
22          Console.Write("Four");
23          sem.Release();
24      }
25
26      static void Main(string[] args)
27      {
28          sem = new Semaphore(0, 1);
29          Thread t1 = new Thread(f1);
30          Thread t2 = new Thread(f2);
31          t1.Start();
32          t2.Start();
33          t1.Join();
34          t2.Join();
35      }
36  }
```

Nástin řešení

1. All scenarios, in lexicographical order:
   (a) 7-8-9-10-11-12-13-deadlock(18+34): `OneThreeFive`
   (b) 7-8-9-10-11-18-19-20-12-13-deadlock(21+34): `OneThreeTwoFive`
   (c) 7-8-9-10-11-18-19-20-21-22-23-12-13: `OneThreeTwoFourFive`
   (d) 7-8-18-19-20-9-10-11-12-13-deadlock(21+34): `OneTwoThreeFive`
   (e) *7-8-18-19-20-9-10-11-21-22-23-12-13: `OneTwoThreeFourFive` - The expected program behavior*
   (f) 7-8-18-19-20-21-22-23-9-10-11-12-13: `OneTwoFourThreeFive`

   Any of the above scenarios except (e) is a sufficient answer.
2. Yes. All the scenarios above work also for FIFO-waits.
3. See code below.

```
private static void f1()
{
```

```
    Console.Write("One");
    semA.Release();
    semB.WaitOne();
    Console.Write("Three");
    semA.Release();
    semB.WaitOne();
    Console.Write("Five");
}

private static void f2()
{
    semA.WaitOne();
    Console.Write("Two");
    semB.Release();
    semA.WaitOne();
    Console.Write("Four");
    semB.Release();
}
```

3 Báze vektorových prostorů (společné okruhy)

1. Zformulujte Steinitzovu větu o výměně vhodných vektorů mezi lineárně nezávislou a generující množinou (čili nikoli nutně bázemi).
2. Mějme reálnou matici $\boldsymbol{A}$ takovou, že je podobná diagonální matici $\boldsymbol{D}$ prostřednictvím součinu $\boldsymbol{R}^{-1}\boldsymbol{A}\boldsymbol{R}$, kde

$$\boldsymbol{R} = \begin{pmatrix} -1 & -1 & -8 & -2 & -8 \\ 0 & 1 & -3 & 0 & 2 \\ -1 & -2 & 6 & -2 & 3 \\ -1 & -3 & 7 & -1 & -4 \\ 1 & 2 & 0 & 2 & 4 \end{pmatrix},$$

   přičemž je známo, že prvky na diagonále $\boldsymbol{D}$ jsou čísla $17, 17, 23, 17, 23$ v uvedeném pořadí.

   Rozhodněte, zdali některý ze sloupců $\boldsymbol{R}$ lze vyměnit za následující vektor $\boldsymbol{u_i}$, aby matici $\boldsymbol{A}$ bylo stále možné diagonalizovat i prostřednictvím výsledné matice $\boldsymbol{R}'$.

   Pokud taková výměna je možná, určete všechny sloupce matice $\boldsymbol{R}$, které mohou být vyměněny.

   (a) Řešte pro $\boldsymbol{u_1} = (4, 2, 2, -3, -2)^{\mathrm{T}}$.

   (b) Řešte pro $\boldsymbol{u_2} = (7, -6, -2, 12, -6)^{\mathrm{T}}$.

   Své odpovědi zdůvodněte.

Nástin řešení

1. Nechť B je konečná lineárně nezávislá množina ve vektorovém prostoru V a C generuje V. Pak existuje D taková, že: D generuje V, $B \subseteq D$, $|D| = |C|$ a $D \setminus B \subseteq C$.
2. Vzhledem k bázi B ze sloupců $\boldsymbol{R}$ jsou (spočteno řešením soustav):

   (a) $[\boldsymbol{u}_1]_B = (0, 2, 0, -3, 0)^{\mathrm{T}}$ tedy $\boldsymbol{u}_1$ lze vyměnit za 2. a 4. sloupce, ty odpovídají stejnému vlastnímu číslu 17, tedy se podprostor vlastních vektorů nezmění.

   (b) $[\boldsymbol{u}_2]_B = (0, 1, 1, 0, -2)^{\mathrm{T}}$ čili $\boldsymbol{u}_2$ neleží v ani jednom podprostoru vlastních vektorů, není tedy vlastním vektorem a proto výměnu provést nelze.

4 Model teorie (společné okruhy)

1. Nechť T je teorie jazyka (signatury) $L = \langle \mathcal{R}, \mathcal{F} \rangle$ v predikátové logice (kde $\mathcal{R}, \mathcal{F}$ jsou množiny relačních a funkčních symbolů s danými aritami). Uveďte definice pojmů *struktura jazyka* L a *model teorie* T.
2. Vyjádřete následující tvrzení formulemi $\varphi_1, \varphi_2, \varphi_3$ v jazyce $L = \langle Z, S, P \rangle$ predikátové logiky (s unárními predikáty pro *'složit zkoušku', 'mít štěstí', 'být připraven'*).
   (a) *Ne každý, kdo složí zkoušku, má štěstí, ale kdo má štěstí, zkoušku složí.*
   (b) *Štěstí přeje připraveným. (Kdo je připraven, má štěstí.)*
   (c) *Nějaký student byl připraven, ale zkoušku nesložil.*
3. Pro teorii $T_1 = \{\varphi_1, \varphi_2\}$, a také pro teorii $T_2 = \{\varphi_1, \varphi_2, \varphi_3\}$, buď najděte nějaký její model anebo formálně dokažte (pomocí tablo metody, rezoluce, či Hilbertovského kalkulu), že žádný model nemá.

Nástin řešení

1. Struktura jazyka L je trojice $\mathcal{A} = \langle A, \mathcal{R}^A, \mathcal{F}^A \rangle$, kde A je neprázdná množina (doména) a $\mathcal{R}^A$, $\mathcal{F}^A$ jsou soubory realizací relačních a funkčních symbolů jazyka L. Realizací n-árního relačního symbolu R je nějaká n-arní relace $R^A \subseteq A^n$, realizací n-árního funkčního symbolu f je nějaká n-ární funkce $f^A \colon A^n \to A$, přičemž jako realizaci konstantního symbolu c (jako nulárního funkčního symbolu) bereme přímo nějaký prvek $c^A \in A$.

   Pozn: Lze uznat i odpověď ve smyslu, že struktura je nějaká k-tice, kde kromě neprázdné domény jsou i příslušné realizace relačních a funkčních symbolů s vysvětlením, co ty realizace jsou.

   Model teorie T je struktura jazyka L, ve které platí všechny axiomy teorie T.
2. Například
   (a) $\varphi_1 : \neg(\forall x)(Z(x) \to S(x)) \wedge (\forall y)(S(y) \to Z(y))$
   (b) $\varphi_2 : (\forall z)(P(z) \to S(z))$
   (c) $\varphi_3 : (\exists s)(P(s) \wedge \neg Z(s))$
3. Teorie T_1 má například jednoprvkový model $\mathcal{A} = \langle \{0\}, Z^A = \{0\}, S^A = P^A = \emptyset \rangle$. Teorie T_2 model nemá, což se dokáže důkazem sporu z T_2 pomocí tablo metody či Hilbertovským kalkulem, nebo zamítnutím skolemizované teorie T_2 převedené do CNF pomocí rezoluce.

5 Datový model (specializace DW)

V informačním systému nemocnice je použit následující logický relační datový model, kde podtržením jsou vyznačeny klíče a italikou cizí klíče:

- Diagnóza(**Kód**, Název)
- Lékař(**RČ**, Jméno, Příjmení, Město, Ulice, Číslo, PSČ, Odbornost, RokyPraxe)
- Pacient(**RČ**, Jméno, Příjmení, Město, Ulice, Číslo, PSČ, *PraktickýLékařRČ*), PraktickýLékařRČ $\subseteq$ Lékař[RČ]
- LéčíSe(***RČ***, *Kód*), RČ $\subseteq$ Pacient[RČ], Kód $\subseteq$ Diagnóza[Kód]

1. Znázorněte výše uvedený logický relační datový model pomocí diagramu ve zvoleném konceptuálním jazyce (ER, UML).
2. Pokud je to potřeba, rozšiřte konceptuální model a odpovídající logický relační datový model tak, aby:
   - každý pacient se mohl léčit na libovolný počet diagnóz,
   - každý pacient měl právě jednoho praktického a mohl mít libovolný počet dalších lékařů,
   - každý pacient mohl podstoupit libovolný počet vyšetření v rámci nějaké diagnózy a naopak,
   - pacient, který je zároveň lékařem, neměl osobní údaje evidovány redundantně.

Nástin řešení

1. Tři entity (třídy): Diagnóza, Lékař, Pacient. Jeden povinný 1:N vztah mezi (praktickým) lékařem a pacienty. Jeden nepovinný 1:N vztah mezi diagnózou a pacienty.
2. – Vztahová tabulka LéčíSe už existuje, ale podporuje omezenější kardinalitu 1:N. Stačí změnit definici klíče. LéčíSe(***RČ***, ***Kód***), RČ ⊆ Pacient[RČ], Kód ⊆ Diagnóza[Kód]
   – Už má právě jednoho praktického lékaře. Nutno přidat M:N vztah a odpovídající vztahovou relaci pro M:N vztah mezi lékaři a pacienty. Léčí(***PacientRČ***, ***LékařRČ***), PacientRČ ⊆ Pacient[RČ], LékařRČ ⊆ Lékař[RČ].
   – M:N vztah Vyšetření mezi pacientem a diagnózou, a k ní odpovídající vztahová relace Vyšetření(***PacientRČ***, ***Kód***), PacientRČ ⊆ Pacient[RČ], Kód ⊆ Diagnóza[Kód].
   – Nejjednodušeji tak, že třída Lékař bude podtřídou třídy Pacient, nebo obě budou podtřídou nové třídy Osoba. V prvním případě relace Lékař přijde o atributy Jméno ... PSČ a RČ bude cizím klíčem na Pacient[RČ]. Ve druhém bude nová relace Osoba obsahovat atributy RČ ... PSČ a obě relace Pacient a Lékař přijdou o atributy Jméno ... PSČ a RČ bude cizím klíčem na Osoba[RČ].

6 Transakce (specializace DW)

Pro následující transakční rozvrh, kde R znamená operaci čtení a W operaci zápisu:

	T_1	T_2	T_3
1	W($Pacient_1$)		
2		???	
3		W($Pacient_1$)	
4		COMMIT	
5			R($Pacient_1$)
6			W($Pacient_1$)
7			COMMIT
8	R($Pacient_2$)		
9	COMMIT		

1. Jaká databázová operace čtení/zápisu v transakci T_2 v čase 2 místo ??? způsobí, že rozvrh nebude konfliktově uspořádatelný (conflict-serializable)? Své rozhodnutí zdůvodněte.
2. Jaká databázová operace čtení/zápisu v transakci T_2 v čase 2 místo ??? způsobí, že rozvrh nebude zotavitelný (recoverable)? Své rozhodnutí zdůvodněte.

Nástin řešení

1. W($Pacient_2$). Vznikne cyklus v precedenčním grafu $T_1 \rightarrow T_2$ a $T_2 \rightarrow T_1$.
2. R($Pacient_1$). Nepotvrzené čtení hodnoty zapsané transakcí T_1, ale COMMIT předchází konec transakce T_1.

7 API a skriptování (specializace DW)

1. Navrhněte a popište REST API pro nemocniční systém z dřívější otázky. API musí podporovat následující funkcionalitu:
   – získání pacientů, jejichž jméno odpovídá query patternu,
   – získání informací o konkrétním pacientovi,
   – vytvořit, aktualizovat a zrušit pacienta,
   – získat seznam diagnóz pacienta,
   – přidat a odebrat diagnózu pacienta,
   – získat seznam lékařů pacienta,
   – získat seznam pacientů, kteří mají aspoň jednu ze zadaných diagnóz (seznam diagnóz může být hodně dlouhý).

2. Mějme webovou aplikaci, která zajišťuje přístup k IS přes navržené API. Napište JavaScript fragment, který po zadání query patternu (první endpoint) a kliknutí na tlačítko vyvolá příslušný HTTP API request, získá všechny pacienty odpovídající dotazu a zobrazí je v seznamu. Když uživatel klikne na konkétního pacienta ze seznamu získaného prvním dotazem, tak se aktualizuje seznam jeho doktorů a diagnóz získáním dat z API a jejich následným zobrazením.

   Předpokládejte, že API vrací data v JSON formátu a tento krátce popište formou příkladu pro každý použitý z endpointů. Předpokládejte existenci funkcí `displayDoctors` a `displayDiagnoses` pro zajištění úpravy DOM stromu. Stejně tak můžete použít funkce `clearDoctors` a `clearDignoses`. Dále předpokládejte existenci všech potřebných HTML elementů.

   Dbejte na správné využití asynchronního zpracování požadavků. Zajistěte, aby i při nepříznivém souběhu asynchronních volání po načtení seznamu pacientů byl seznam doktorů a diagnóz v konzistentním stavu, tj. zobrazují se seznamy ke zvolenému pacientu a nezobrazují se seznamy v případě, kdy pacient vybrán nebyl. Není třeba řešit chybové stavy fetch operací ani autentikaci.

Nástin řešení

1. Příslušné endpointy by měly vypadat např. následujícím způsobem:
   (a) GET api/v1/patients?query={pattern}, případně GET api/v1/patients/{pattern}
   (b) GET/DELETE/POST api/v1/patient/{rc}
   (c) GET api/v1/patient/{rc}/diagnoses
   (d) GET/POST/DELETE api/v1/patient/{rc}/diagnoses
   (e) GET api/v1/patient/{rc}/doctors
   (f) POST api/v1/diagnoses

   Vzhledem k tomu, že seznam diagnóz v posledním API endpointu může být dlouhý, je třeba payload posílat v těle dotazu a tedy pro tento endpoint nelze použít GET request.

2. Například takto:

```
let currentQuery = '';

document.getElementById('searchButton').addEventListener('click', async () => {
  const query = document.getElementById('searchInput').value();
  currentQuery = query;

  const patientList = document.getElementById('patient-list');

  patientList.innerHTML = '';
  clearDoctors();
  clearDiagnoses();

  const patients = await (
    await fetch(`/api/v1/patients?query=${encodeURIComponent(query)}`)
  ).json();

  patients.forEach(patient => {
    const item = document.createElement('li');
    item.textContent = patient.name;
    item.dataset.rc = patient.rc;
    item.addEventListener('click', () => fetchPatientDetails(patient.rc, query));
    patientList.appendChild(item);
  });
}

async function fetchPatientDetails(rc, query) {
  const doctors = await (await fetch(`api/v1/patient/${rc}/doctors`)).json();
```

```
    const diagnoses = await (await fetch(‘api/v1/patient/${rc}/diagnoses‘)).json;
    if (query != currentQuery){
      displayDoctors(doctors);
      displayDiagnoses(diagnoses);
    }
  }
```

3. Např. API se seznamem pacientu bude mít tvar:

   `[{ "rc": <RČ pacienta>, "name": <jméno>, "surname": <příjmení>, ... }]`

4. Řešení by mělo ukazovat znalost iterace, použití async/await a práci s data atributy (list items).

5. Mělo by být zajištěno, že pokud je zavolán listener searchButtonu a zrovna se čeká na výsledky z API doktorů a diagnóz, tak tyto nepřepíšou už vyčištěné elementy.

8 Získávání informací (specializace DW)

Uvažujte situaci, kdy je třeba obohatit systém z předchozích otázek tak, aby umožňoval stanovit diagnózu na základě zadaných symptomů. Mějme kolekci dokumentů, kde každý dokument popisuje konkrétní nemoc.

1. Jak by vypadal booleovský model pro danou úlohu? Jaké termy by obsahoval spcializovaný slovník pro danou úlohu? Jak jsou reprezentovány dotazy a dokumenty v booleovském modelu a jak lze dotazování v tomto modelu efektivně implementovat?

2. Řekněme, že chceme systém modifikovat tak, aby dotaz nebyl seznam symptomů, ale přímo zpráva lékařského vyšetření. Jakou modifikaci booleovského modelu je vhodné v takovém případě použít? Jak se změní reprezentace dotazu a dokumentu? Jak bude pak vyhodnocována podobnost mezi dotazem a dokumentem a proč?

Nástin řešení

1. Booleovsý model

2. (a) Termy = symptomy, dokumenty = popisy nemocí
   (b) Specializovaný slovník symptomů, tj. není třeba získávat slovník (jako je tomu v klasickém DISu) parsováním dokumentů
   (c) Systém reprezentován binární maticí, dotazy realizovány jako bitové operace nad vektorem termů (symptomů)
   (d) Efektivní reprezentace invertovaným indexem a merge sort-like zpracováním

3. Rozšíření – např. vektorový model

4. (a) Dotaz bude reprezentován stejně jako dokument
   (b) Podobnost se vyhodnocuje typicky pomocí cosinové vzdálenosti v tf-idf prostoru mezi vektory dotazu a dokumentu

9 Samoopravné kódy (specializace OI-G-PADS, OI-G-PDM, OI-O-PADS, OI-PADS-PDM)

V této otázce budeme pracovat pouze s binárními samoopravnými kódy, tj. nad abecedou $\{0, 1\}$.

1. Definujte následující pojmy: Hammingova vzdálenost, samoopravný kód, délka, velikost a vzdálenost samoopravného kódu.

2. Nechť C je samoopravný kód velikosti $k \geq 2$ a délky n. Nechť x je slovo nad abecedou $\{0, 1\}$ délky m. Jakým způsobem by se kód C použil k zakódování slova x tak, aby výsledné slovo bylo co nejkratší? Jaká bude v závislosti na parametrech k, n a m délka výsledného slova?

3. Existuje samoopravný kód délky 5, velikosti 6 a vzdálenosti 3? Pokud ano, nějaký zkonstruujte; pokud ne, dokažte, že neexistuje.

Nástin řešení

1. *Hammingova vzdálenost* $d(x,y)$ dvou slov x a y stejné délky je počet pozic, na kterých se x a y liší. *Samoopravný kód* je libovolná množina slov stejné délky, v binárním případě tedy libovolná podmnožina $C \subseteq \{0,1\}^n$. Zde n je přirozené číslo, kterému říkáme *délka* kódu C. *Velikost* kódu C je definována jako velikost $|C|$ množiny C, tedy jako počet jeho slov. *Vzdálenost* kódu C je minimum z Hammingových vzdáleností mezi jeho slovy, tj.

$$\min_{x,y \in C, x \neq y} d(x,y).$$

2. Jednou z možností je x interpretovat jako binární zápis čísla v rozsahu od 0 do $2^m - 1$. Toto číslo si můžeme vyjádřit v soustavě o základu k jako posloupnost číslic $q_0 q_1 \dots q_{t-1}$, kde $t = \lceil \log_k 2^m \rceil$. Dále si zafixujeme bijekci $c : \{0, \dots, k-1\} \to C$ a výsledným slovem pak bude zřetězení slov $c(q_0)$, $c(q_1)$, ..., $c(q_{t-1})$. Délka tohoto slova tedy bude

$$n \cdot \lceil \log_k 2^m \rceil = n \cdot \left\lceil \frac{m}{\log_2 k} \right\rceil \approx \frac{n}{\log_2 k} \cdot m.$$

   Jinou možností by bylo rozdělit x na bloky $\lfloor \log 2k \rfloor$ bitů a obdobně kódovat každý blok zvlášť, to ale v případě, že k není mocnina 2, vede k suboptimální délce výsledného slova $\approx \frac{n}{\lfloor \log 2k \rfloor} \cdot m$.

3. Takový kód nemůže existovat, jelikož nesplňuje podmínku z Hammingova odhadu. Případně lze dokázat přímo: V kódu C vzdálenosti 3 jsou koule o poloměru 1 (v Hammingově vzdálenosti) kolem každého jeho slova disjunktní. Má-li kód C délku 5, každá z těchto koulí má velikost 6, do prostoru slov $\{0,1\}^5$ se jich tedy nemůže vejít více než

$$\left\lfloor \frac{2^5}{6} \right\rfloor = 5.$$

10 Optimalizace (specializace OI-O-PADS)

1. Je dán bipartitní graf $G = (U, V, E)$. Uveďte soubor lineárních podmínek popisující konvexní obal všech párování grafu G.

2. Napište, jak tento soubor vypadá pro graf $G = (U, V, E)$, kde $U = \{1, 2, 3\}$, $V = \{a, b\}$, $E = \{1a, 1b, 2a, 2b, 3a, 3b\}$.

   Přidejte k Vašemu souboru účelovou funkci maximalizující velikost párování a výsledný lineární program označme (P). Napište duální program (D) k primárnímu (P).

3. Formulujte přesné znění slabé a silné věty o dualitě lineárního programování.

   Najděte nějaké párování M největší velikosti v daném grafu G a s využitím slabé věty o dualitě LP a lineárního programu (D) z předešlé otázky dokažte, že M je párování největší velikosti v grafu G.

Nástin řešení

1. Nechť $\delta(u) \subseteq E$ označuje množinu hran sousedících s vrcholem $u \in U \cup V$.

$$\sum_{e \in \delta(u)} x_e \leq 1 \qquad \forall u \in U \cup V$$
$$x_e \geq 0 \qquad \forall e \in E$$

2. Pro graf $G = (U, V, E)$, kde $U = \{1, 2, 3\}$, $V = \{a, b\}$, $E = \{1a, 1b, 2a, 2b, 3a, 3b\}$, dostaneme:

$$
\begin{aligned}
(P) \qquad\qquad \max \quad x_{1a} + x_{1b} + x_{2a} + x_{2b} + x_{3a} + x_{3b} & \\
x_{1a} + x_{1b} &\le 1 \\
x_{2a} + x_{2b} &\le 1 \\
x_{3a} + x_{3b} &\le 1 \\
x_{1a} + x_{2a} + x_{3a} &\le 1 \\
x_{1b} + x_{2a} + x_{3b} &\le 1 \\
x_e &\ge 0 \quad \forall e \in E
\end{aligned}
$$

$$
\begin{aligned}
(D) \qquad\qquad \min \quad y_1 + y_2 + y_3 + y_a + y_b & \\
y_1 + y_a &\ge 1 \\
y_1 + y_b &\ge 1 \\
y_2 + y_a &\ge 1 \\
y_2 + y_b &\ge 1 \\
y_3 + y_a &\ge 1 \\
y_3 + y_b &\ge 1 \\
u &\ge 0 \quad \forall u \in U \cup V
\end{aligned}
$$

3. Nechť (P) a (D) jsou následující lineární programy:

$$
\begin{aligned}
(P) \qquad\qquad \max c^T x & \\
Ax &\le b \\
x &\ge 0
\end{aligned}
$$

$$
\begin{aligned}
(D) \qquad\qquad \min b^T y & \\
y^T A &\ge c^T \\
y &\ge 0
\end{aligned}
$$

Slabá věta o dualitě Pro každé přípustné řešení x úlohy (P) a každé přípustné řešení y úlohy (D) platí $c^T x \le b^T y$.

Silná věta o dualitě Mají-li úlohy (P) i (D) přípustné řešení, pak platí: x je optimální řešení úlohy (P) a y je optimální řešení úlohy (D) právě když $c^T x = b^T y$.

Např. uvažme párování $M = \{1a, 2b\}$ a přípustné duální řešení $y_a = y_b = 1$ a $y_1 = y_2 = y_3 = 0$. Hodnota účelové funkce pro toto y je 2, což je podle slabé věty o dualitě horní mez na hodnotu účelové funkce jakéhokoli přípustného (tedy i optimálního) řešení primární úlohy. Protože velikost M je dva, musí to být optimální řešení, tedy párování v G největší možné velikosti.

11 Hellyova věta (specializace OI-G-PADS, OI-G-PDM)

1. Zformulujte Hellyovu větu pro konečně mnoho konvexních množin v $\mathbb{R}^d$.
2. Zformulujte Hellyovu větu pro nekonečně mnoho konvexních množin v $\mathbb{R}^d$.
3. Najděte systém nekonečně mnoha konvexních množin $C_1, C_2, \ldots$ v rovině takový, že průnik každého konečného podsystému je neprázdný, ale $\bigcap_{i=1}^{\infty} C_i = \emptyset$.

Nástin řešení

1. Jiří Matoušek, Introduction to Discrete Geometry, Theorem 1.3.2

2. Jiří Matoušek, Introduction to Discrete Geometry, Theorem 1.3.3
3. Např. $C_n = \{(x, y); x \geq n\}$

12 Podgraf s předepsanými stupni (specializace OI-G-PDM, OI-PADS-PDM)

1. Definujte pojem perfektního párování v obecném grafu a formulujte nutnou a postačující podmínku pro jeho existenci (Tutteova věta).
2. Nechť G je graf a $d : V(G) \to \mathbb{N}$ je libovolná funkce. Jako H označme graf vytvořený z G tak, že každý vrchol v nahradíme úplným bipartitním grafem $K_{\deg v, \deg_G v - d(v)}$ s partitami A_v a B_v a na vrcholech $\bigcup_{v \in V(G)} A_v$ přidáme perfektní párování takové, že pro každou hranu $\{u, v\} \in E(G)$ je v H právě jedna hrana mezi A_u a A_v. Přesněji řečeno,

$$\begin{aligned} A_v &= \{(v, u) : \{v, u\} \in E(G)\} && \text{pro každé } v \in V(G) \\ B_v &= \{(v, i) : v \in V(G), i \in \{1, \ldots, \deg_G(v) - d(v)\}\} && \text{pro každé } v \in V(G) \\ V(H) &= \bigcup_{v \in V(G)} (A_v \cup B_v) \\ E(H) &= \{\{(u, v), (v, u)\} : \{u, v\} \in E(G)\} \cup \bigcup_{v \in V(G)} \{\{x, y\} : x \in A_v, y \in B_v\} \end{aligned}$$

   Ukažte, že graf H má perfektní párování právě tehdy, když G obsahuje podgraf G' takový, že $V(G') = V(G)$ a každý vrchol v má v G' stupeň přesně $d(v)$.
3. Navrhněte algoritmus s polynomiální časovou složitostí, který pro zadaný graf G a funkci $d : V(G) \to \mathbb{N}$ nalezne podgraf $G' \subseteq G$ takový, že $V(G') = V(G)$ a každý vrchol v má v G' stupeň přesně $d(v)$, nebo rozhodne, že takový podgraf neexistuje.

Nástin řešení

1. *Perfektní párování* v grafu G je podmnožina M jeho hran taková, že každý vrchol G je incidentní s právě jednou z hran M. Existují i další možné ekvivalentní definice, např. 1-regulární podgraf G s množinou vrcholů $V(G)$.

   Tutteova věta: Graf G má perfektní párování právě tehdy, když

$$\mathrm{odd}(G - S) \leq |S|$$

   pro každou podmnožinu $S \subseteq V(G)$, kde $\mathrm{odd}(F)$ je počet komponent grafu F, které mají lichý počet vrcholů.
2. Existuje-li takový podgraf G', pak $M = \{\{(u, v), (v, u)\} : \{u, v\} \in E(G')\}$ je párování v H pokrývající pro každé $v \in V(G)$ právě $d(v)$ vrcholů z A_v. Zbylých $\deg_G v - d(v)$ vrcholů z A_v lze libovolně spárovat s vrcholy B_v, čímž M rozšíříme na perfektní párování v H.

   Naopak, nechť M' je perfektní párování v H. Pro každé $v \in V(G)$ toto párování obsahuje právě $|B_v| = \deg_G v - d(v)$ hran mezi B_v a A_v, a tedy právě $d(v)$ hran s jedním koncem v A_v a druhým mimo B_v. Proto podgraf

$$G' = (V(G), \{\{u, v\} : u, v \in V(G), \{\{(u, v\}, (v, u\} \in M'\})$$

   splňuje popsané podmínky na stupně vrcholů.
3. Stačí aplikovat (např.) Edmondsův algoritmus pro nalezení perfektního párování M' v grafu H a existuje-li, použít transformaci popsanou v předchozím odstavci.

13 Komponenty silné souvislosti (specializace OI-G-PADS, OI-O-PADS, OI-PADS-PDM)

1. Definujte *komponenty silné souvislosti* orientovaného grafu. S jakou časovou složitostí je umíte nalézt? (Detaily algoritmu nemusíte popisovat.)
2. Definujte *graf komponent* (neboli kondenzaci) a popište jeho vlastnosti.

3. Navrhněte algoritmus, který dostane orientovaný graf s některými vrcholy označenými zeleně. Výstup je sled (ne nutně cesta), na němž leží co nejvíce zelených vrcholů. Algoritmus by měl pracovat v polynomiálním, nejlépe lineárním čase.

Nástin řešení

1. Viz Průvodce labyrintem algoritmů, oddíl 5.9.
2. Viz tamtéž.
3. Jakmile sled navštíví komponentu silné souvislosti, může vysbírat všechny zelené vrcholy v ní. Sestrojíme tedy graf komponent a jeho vrcholy ohodnotíme počty zelených vrcholů v komponentách. V tomto grafu chceme najít sled s největším součtem. Jelikož graf komponent je acyklický, tento sled bude cestou a můžeme ho najít indukcí podle topologického uspořádání. Vše běží v lineárním čase s počtem vrcholů a hran grafu.

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

1. Nechť $\mathbb{R}^n$ označuje n-dimenzionální euklidovský prostor a nechť $d_n(x, y)$ označuje euklidovskou vzdálenost bodů $x, y \in \mathbb{R}^n$. Pro funkci $f\colon \mathbb{R}^m \to \mathbb{R}^n$ a bod $x \in \mathbb{R}^m$ napište, co znamená, že f *je spojitá v bodě* x.
2. Uvažujme funkci $f(x, y)\colon \mathbb{R}^2 \to \mathbb{R}$ definovanou vzorcem

$$f(x, y) = \exp(x^2) \cdot \ln(2 + y^2),$$

kde $\exp(x) = e^x$ označuje exponenciální funkci a $\ln(x)$ označuje přirozený logaritmus. Spočítejte hodnoty parciálních derivací $\frac{\partial^2 f}{\partial x \partial y}$ a $\frac{\partial^2 f}{\partial x^2}$.

3. Nabývá funkce f z předchozí podotázky v některém bodě globální maximum nebo globální minimum? Pokud ano, uveďte, ve kterých bodech se tyto extrémy nabývají a o jaký typ extrému se jedná. (Tuto úlohu lze řešit více různými způsoby.)

Nástin řešení

1. Funkce $f\colon \mathbb{R}^m \to \mathbb{R}^n$ je spojitá v bodě $x \in \mathbb{R}^m$, pokud pro každé $\varepsilon > 0$ existuje $\delta > 0$ takové, že pro všechna $x' \in \mathbb{R}^m$ splňující $d_m(x, x') < \delta$ platí $d_n(f(x), f(x')) < \varepsilon$.
2. Výpočet dává

$$\frac{\partial^2 f}{\partial x \partial y} = 2x \exp(x^2) \cdot 2y \frac{1}{2 + y^2} = \frac{4xy \exp(x^2)}{2 + y^2}$$

a

$$\frac{\partial^2 f}{\partial x^2} = \frac{\partial}{\partial x} 2x \exp(x^2) \ln(2 + y^2) = (2 + 4x^2) \exp(x^2) \ln(2 + y^2).$$

3. Funkce f není shora omezená (např. $\lim_{x \to +\infty} f(x, 0) = +\infty$), a tedy nenabývá v žádném bodě globální maximum. Globální minimum nabývá v bodě $(x, y) = (0, 0)$. To lze zjistit výpočtem parciálních derivací a hessiánu, nebo třeba využitím toho, že funkce $f_1(x) = \exp(x^2)$ je kladná a má jediné minimum v $x = 0$, zatímco funkce $f_2(y) = \ln(2 + y^2)$ je kladná a má jediné minimum v $y = 0$. Z toho plyne, že funkce $f(x, y) = f_1(x) f_2(y)$ má jediné globální minimum v $(x, y) = (0, 0)$.

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

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

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

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

Pokud pracujeme s hierarchickým **stromem**, lze ještě zavést pojem "atribut" a "dědičnost atributu". Každý uzel grafu může mít přiřazeny atributy definující některé vlastnosti objektu: barvu, materiál, texturu ... nebo i negrafické atributy (metadata). Dědičnost umožňuje definovat atributy jen v některých vnitřních uzlech, do podřízených částí stromu se přenesou dědičností. DAG graf má použití atributů problematičtější, ale též se dá vymyslet smysluplná sémantika.

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

Instancing: pokud máme dovoleno použít DAG, je instancing snadný: prostě se použije tolik odkazů na daný uzel (objekt), kolik je potřeba. Každý odkaz je opatřen svou transformační maticí. Pokud bychom chtěli i modifikovat atributy, musela by

se v interní implementaci systému místo prostého ukazatele používat cesta od kořene k uzlu objektu (aby se daly vyhodnotit atributy).

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

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

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

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

17 Animace pohybu pevného tělesa (specializace PGVVH-PG, PGVVH-VPH)

V této otázce se budeme zabývat animací pevných těles ve 3D. Pevné těleso pro nás bude jakýkoliv - obvykle souvislý - 3D objekt, jehož tvar se nemění, ale je dovoleno tímto objektem ve scéně pohybovat. Představujte si Váš mobilní telefon nebo třeba hrneček, ze kterého pijete čaj.

Jednou ze základních animačních technik je použití tzv. klíčových snímků ("keyframe animation"), kdy se scéna přesně definuje jen v určitých bodech na časové ose ("keyframes") a algoritmus potom doplňuje animaci hladce tak, aby ulehčil práci člověku.

1. Jaké matematické prostředky byste použili, kdyby bylo potřeba pouze těleso přemisťovat v prostoru (říká se tomu také "translace")? Myslete na to, že by animátoři chtěli definovat polohu tělesa v klíčových okamžicích a Vaším úkolem by bylo dopočítat všechny ostatní pozice. Uvažujte dvě možnosti pro klíčové pozice: buď chceme, aby animace hladce tímto bodem proběhla (ideálně by nemělo vůbec být vidět, že tam nějaký klíčový bod byl), nebo bychom si přáli, aby se objekt v tom bodě ostře "odrazil". Navrhněte řešení pro obě varianty.
2. Druhou složkou pohybu tuhého tělesa je orientace/otočení v prostoru. Budeme chápat tuto složku zcela nezávislou na translaci, přemýšlejte o tom nyní tak, jakoby translace neexistovala. Jak byste pomohli animátorům při definici otáčení, jaké matematické prostředky byste jim nabídli? Uvažujte opět systém klíčových snímků a automatické interpolace mezi nimi. Zvolte si jeden ze známých matematických aparátů, stručně jeho použití popište a označte jeho výhody a nevýhody.
3. Nakonec se zamyslete, jak byste oba dva předchozí mechanismy spojili do jednoho animačního systému, kde budeme chtít tělesa přemisťovat v prostoru a současně jimi otáčet. Vysvětlete, jak by se to (matematicky) udělalo. Dále se zkuste zamyslet, jak vytvořit uzavřenou smyčku ("loop") animace, aby se dala přehrávat donekonečna a nebylo poznat, kde je začátek/konec?

Nástin řešení Pro **translaci** by se dal použít koncept přemisťování nějakého významného bodu tělesa (např. těžiště) vhodně zvolenou animační křivkou. Pro použití v klíčové animaci by se hodily interpolační systémy, které buď jen zadávají hodnotu (polohu) v klíčových snímcích (příkladem by mohla být Catmul-Rom křivka nebo TCB spline - viz literatura), anebo by se použila **Hermitova interpolace**, pro kterou by se dalo lépe pracovat s okrajovými směry a rychlostmi. Pro dosažení spojitého navázání dvou Hermitových křivek by se musela zavést podmínka shodnosti incidentních tečných vektorů (koncový vektor první křivky se musí rovnat počátečnímu vektoru křivky navazující). Při obtížnější parametrizaci (např. každý interval mezi sousedními keyframes má jinou délku) by se u Hermitovy interpolace dalo snadněji dosáhnout spojitosti.

Předběhneme na poslední podotázku: i tam by se dala s Hermitem snadno zajistit spojitost zacyklené animace. Nespojité animace jsou u Hermitovy interpolace snadné, prostě nadefinujeme koncovou rychlost prvního úseku a počáteční rychlost druhého úseku dle libosti.

Rotace/orientace by se dala elegantně implementovat pomocí tzv. “kvaternionů”. 4D vektory definují orientaci tělesa pomocí osy a úhlu otočení (zde by bylo asi vhodné napsat pár definic - dle jakékoli učebnice geometrie - stačí základní definice kvaternionu q, obecnou mocninu q^t pro reálné číslo t a sférickou interpolaci $SLERP(p, q, t)$). Interpolace mezi klíčovými snímky by se dala definovat pomocí sférických splinů (Bézierovy křivky se pro kvaterniony dají elegantně implementovat pomocí De Casteljau algoritmu a $SLERP()$). Tím bychom dostali možnost definovat otáčivé pohyby spojitě navazující i nespojité: Bézierovy křivky umožňují obě varianty. To samé platí i pro nekonečnou/zacyklenou animaci z poslední podotázky - použil by se spojitý případ, jestliže by bylo žádoucí pozorovat hladký přechod. Výhoda kvaternionů spočívá hlavně ve velké hladkosti přechodů mezi jednotlivými orientacemi a možnost pracovat i s nespojitostmi v případě potřeby. Implementačně se celá interpolační/animační část výpočtu odehrává v oblasti kvaternionů, ale úplně nakonec se kvaternion převede na homogenní transformační matici, kterou lze snadno zkombinovat s translací.

Kombinace translace a rotace: obě složky pohybu by vedly k homogenním maticím 4×4, které by se následně vynásobily (nejdřív rotace a pak translace). Pro jakýkoli časový okamžik by se interpolovalo mezi incidentními klíčovými snímky (translace i rotace) a výsledkem by byl součin dvou homogenních matic, kterým by se celé animované těleso transformovalo. Možnosti nekonečné/zacyklené animace již byly popsány dříve, v navržených implementacích by to bylo snadné.

18 Urychlování ray tracingu (specializace PGVVH-VPH)

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

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

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

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

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

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

1. Vstupní scéna je neuspořádaná množina trojúhelníků (triangle soup), každý trojůhelník má své identifikační (pořadové) číslo id (implicitní údaj, nemusí se ukládat) a trojici vrcholů V_1, V_2 a V_3. Každý vrchol musí mít 3D souřadnice $[x, y, z]$, ale může obsahovat i další údaje pro rendering, například normálový vektor, texturovou souřadnici či barvu, ty se ovšem nebudou při výpočtu průsečíků uvažovat. Celý trojúhelník může mít odkaz na těleso, ze kterého pochází (implicitně, pomocí svého id), materiál, textura či barva by pak mohly být uložené u tělesa. Paprsek je jednoduše reprezentován svým počátkem P_0 a směrovým vektorem $\vec{p_1}$ (pozn. předpokládáme, že paprsek již byl předem transformován do souřadného systému scény). Průsečík P bude možné reprezentovat jediným reálným číslem t ($P = P_0 + t \cdot \vec{p_1}$).

2. BVH bude binární strom, kde ve všech uzlech budou obalové AABB kvádry a v listech navíc seznamy odkazů na trojúhelníky (jen seznamy číselných *id*). Tzv Bucket tree umožňuje uložit do listů více trojúhelníků, aby se zbytečně nemusely dělit menší množiny trojúhelníků. Konstrukce stromu se bude implementovat metodou top-down, kdy se dělí počáteční celá množina na menší části. Po provedení několika dělení se může další výpočet provádět paralelně (rozděl a panuj). Oblíbená suboptimální heuristika SAH (Surface Area Heuristics) rozhoduje, jak přesně se v každém vnitřním uzlu rozdělí množina trojúhelníků na dvě menší tak, aby byl minimalizován očekávaný čas výpočtu průsečíků (pozn. zde mohou být uvedeny podmínky a náznak odvození SAH ... pravděpodobnostní vzorce). Celá konstrukce BVH stromu může být formulována tak, že povede k vyváženému stromu, což umožňuje jeho implicitní uložení (bez pointerů). Pro dobře strukturovanou scénu, která neobsahuje příliš extrémně protáhlých trojúhelníků (daly by se i uměle rozdělit...) bude dosaženo průměrné časové složitosti $O(logN)$ výpočtu průsečíku paprsku se scénou, protože hloubka stromu je logaritmická a potenciál nutnosti procházet všechny větve je malý.

3. Pro daný paprsek postupujeme od kořene BVH stromu směrem do listů. Vždy zkontrolujeme, zda paprsek protíná obalový kvádr podstromu: když ne, nemusíme do té větve vůbec vstupovat. Pokud paprsek protne obaly obou potomků daného uzlu, musíme navštívit obě větve; jako první projdeme podstrom s bližším průsečíkem. Můžeme též použít algoritmus procházející do šířky ty větve, jejichž obaly byly protnuty. Prioritou jejich zpracování je opět pořadí průsečíků s jejich obaly (blíže protnuté obaly mají větší potenciál, že v nich bude nalezen nejbližší průsečík). V každém navštíveném listu se musí otestovat paprsek proti všem uloženým trojúhelníkům. Negativní výsledek nastane, jestli jsme zpracovali všechny protnuté obaly (= jejich podstromy) a paprsek neprotnul žádný z trojúhelníků. Pozitivní výsledek můžeme ohlásit tehdy, když paprsek protnul trojúhelník A a současně byly otestovány všechny ostatní trojúhelníky v daném listu (a byly dál než průsečík s A). Dále musí platit, že ve frontě dosud nezpracovaných vnitřních uzlů není žádný obal, jehož průsečík leží blíže než průsečík s A. Paralelismus se implementuje snadno, protože urychlovací BVH strom se po své konstrukci již nemění a může se předávat jednotlivým výpočetním jednotkám jako sdílená R/O data.

19 Shadery v moderních GPU (specializace PGVVH-VPH)

Půjde nám o principy programování moderních grafických karet (GPU) pomocí tzv. shaderů. Budeme se podrobněji zabývat pouze dvěma základními typy shaderů, otázky se ptají na ta nejdůležitější fakta. Přečtěte si pečlivě celé otázky a odpovězte na všechny jejich části.

1. První základní (povinný) shader je tzv. "Vertex shader". Uveďte, ve kterém místě 3D zobrazovacího řetězce ("3D pipeline") je zařazen, které má typické vstupy a které typické výstupy? Co je obvykle jeho úkolem při 3D zobrazování?
2. Druhý povinný shader se nazývá "Fragment shader" nebo "Pixel shader". Co je jeho úkolem, kde je do 3D pipeline zařazen a které vstupy a výstupy má?
3. Popište, kterými cestami/kanály se do shaderů dostávají data z aplikace. Přemýšlejte opravdu o všech možnostech, jak data do GPU dostat a jak z aplikace ovlivňovat běh shaderů! V čem se jednotlivé přístupy liší a jaké mají výhody a nevýhody?

Nástin řešení "Vertex shader" (VS) je zařazen na začátek řetězce, kde se ještě pracuje s vrcholy samostatně a nejsou propojeny do grafických primitiv. V modernějších architekturách může být za VS zařazena dvojice "Tesselation shaders" a/nebo "Geometry shader" či "Mesh shader". Vstupem jsou obvykle data vrcholů, která se čtou z GPU bufferů a dále "uniforms" či "constants", které poskytuje aplikace hromadně pro celou dávku či instanci. Na výstupu VS je sada tzv. variabilních veličin, kterou definuje programátor shaderu. Ty pak s každým vrcholem putují dál v řetězci, stávají se předmětem interpolace v rasterizéru a nakonec je konzumuje "Fragment shader". Nejběžnějšími výstupními veličinami jsou: souřadnice vrcholu v normalizovaném souřadném systému (NDS = Normalized Device Space), původní či světové souřadnice, texturové souřadnice, barva, apod. Jen ta první zmíněná veličina je povinná a každý VS ji musí spočítat (ve starším OpenGL to bylo `gl_Position`).

Obecně řečeno: úkolem VS je spočítat (transformovat, definovat) veličiny, které patří k jednotlivým vrcholům bez ohledu na jejich příslušnost k primitivům. Protože je obvykle vrcholů řádově méně než fragmentů/pixelů, je vhodné do VS umístit i náročnější výpočty, u kterých nám nevadí, že budou jejich výsledky interpolovány při rasterizaci.

"Fragment shader" (FS) je zařazen hned po rasterizaci (anebo případně ještě za předčasný Z-test, abychom ušetřili výpočty těch fragmentů, které nebudou vidět). Vstupem jsou všechny variabilní veličiny, které spočítal VS a které byly interpolovány v rasterizéru. Úkolem FS je určit RGBA barvu fragmentu (nepovinně a hodně zřídka se mění i jeho hloubka). Hojně k tomu přispívají i textury, které jsou připojené přes tzv. texturovací jednotky a můžeme jich současně používat několik. V moderních GPU se stírá rozdíl mezi texturami a datovými buffery, takže se dá napsat, že jakékoli formy datových polí GPU jsou často

ve FS využívány. V modernějších konceptech tzv. "Multiple Render Targets" se z jednoho vyvolání FS zapisuje výsledek do více výstupních bufferů (jeden pro RGBA barvu, další například pro normálový vektor nebo jiné geometrické veličiny).

Cesty dat z aplikace do GPU k shaderům:

- **Atributy vrcholů** jsou čteny vertex shaderem a aplikace je musí dopředu nahrát do vstupních bufferů (obvykle do "Vertex buffer"). Tzv. formát vrcholu definuje typ a sémantiku jednotlivých atributů, shader musí u sebe mít kompatibilní deklaraci. Mohou to být jakákoli data, která se mění mezi jednotlivými vrcholy (pozice, normála, barva...)
- **Uniforms/Constants** (**Uniform/Constant buffers**) - zde aplikace nastavuje veličiny, které mají platit delší dobu, typicky v průběhu zpracování jedné kreslicí dávky či instance. Obvykle se setkáme s transformačními maticemi (neměnné pro celý objekt, pro celý frame nebo i déle), matriálovými konstantami, odkazy na textury a s dalšími parametry ovlivňujícími výpočet shaderů.
- **Textury** jsou vlastně také "konstanty", ale většího rozsahu. Jsou to velká (1D, 2D nebo 3D) pole barev, normálových vektorů nebo jiných údajů podle rozhodnutí programátorů. Nejčastěji se při renderingu setkáme s barevnými nebo normálovými texturami a FS využívá schopnosti texturové jednotky automaticky a rychle interpolovat, dokonce hierarchicky (viz MIP-map)
- **Buffers** jsou další obecná pole dat, která můžete v aplikaci poslat do GPU a shadery se pak na ně mohou odkazovat. Na rozdíl od textur neumějí tato pole interpolovat mezi svými prvky.

Atributy vrcholů: přístup k datům bez zdržení (přirozený streaming), nevýhoda - každá datová položka se musí alokovat pro každý vrchol, i kdybychom ji třeba nepotřebovali...

Uniforms/Constants: velmi rychlý přístup k datům, ale omezený objem dat (stovky float4 vektorů).

Textury: velký objem dat, umějí indexovat až do 3D a HW interpolovat (i hierarchicky). Rychlost střední (globální paměť).

Buffers: jako textury, jen bez interpolace.

20 Organizace paměti (specializace PVS)

Uvažujte následující program v C/C++. Předpokládejte 32-bitový procesor a velikosti datových typů char 1B, int 4B, double 8B.

```
struct Str { int a = 11; bool b = false; char c = 'C'; double d = 3.14; char e = 'E'; };

Str * sp = 0;

int f( const Str& s1) {
        *sp = s1;
        Str s2 = *sp;
        ++(*sp).a;
        // memory dump
        return s2.a;
}

int main() {
        Str sa[3];
        sp = new Str;
        ++sa[2].a;
        sa[1].a += f( sa[2]);
        delete sp;
        sp = 0;
}
```

Nakreslete a popište s přesností na bajty rozložení paměti pro všechna data programu a její obsah (tam, kde ho znáte) při zastavení programu v místě označeném `// memory dump`. Jasně odlište různé druhy paměti za běhu programu (zásobník apod.). Tam, kde je to relevantní, stručně popište možné alternativy. Schematicky znázorněte i interní data nutná při běhu programu, i když neznáte jejich přesnou strukturu ani hodnoty. Pokud budete pro zákres a popis potřebovat konkrétní adresy, vhodně si je vymyslete. Adresu na konkrétní znázorněné paměťové místo lze zakreslit šipkou. Přesnou reprezentaci hodnot typu double řešit nemusíte.

Nástin řešení Řešení by mělo počítat se zarovnáním datových položek v rámci struktury a zarovnáním struktur v poli, mělo by správně ukázat uložení globálních, lokálních a dynamicky alokovaných proměnných, předávání parametrů, aktivační záznam funkce a základní volací konvence.

21 Paralelní výpočty a synchronizace vláken (specializace PVS)

Popište tyto prostředky na synchronizaci výpočtů (akcí) několika současně běžících vláken: atomické operace, atomické proměnné. Vysvětlete jejich sémantiku především s ohledem na možné stavy programu a uveďte příklad použití v jednom z programovacích jazyků C++, C#, Java.

Popište tyto abstrakce pro vyjádření paralelních výpočtů: task, fork-join a future. Vysvětlete jejich sémantiku a uveďte příklad použití v jednom z programovacích jazyků C++, C#, Java.

Hodnotí se především správné vyjádření konceptů, drobné syntaktické nepřesnosti v příkladech nejsou podstatné.

Nástin řešení Atomické operace jsou operace (jednotlivé instrukce nebo i sekvence instrukcí), které jsou provedeny *v jednom kroku*, takže jiný proces či vlákno nemohou číst průběžný stav dotčených proměnných během vykonávání operace a modifikovat hodnoty těchto proměnných. Uvažujeme tady jenom proměnné (stav), se kterými pracuje daná operace.

Atomické proměnné jsou takové, jejichž hodnota se může změnit jen atomicky. Žádný jiný proces či vlákno nemohou vidět průběžný stav, uvídí jen stav před operací a po dokončení operace.

Task je objekt reprezentující výpočet (aktivitu, job), který se může vykonat paralelně s dalšími tasky. Je to vlastně základní jednotka paralelních výpočtů (algoritmů). Abstrakci typicky poskytuje knihovna.

Fork-join: rozdělení úlohy na dílčí tasky, které jsou vykonané paralelně, a jejichž výsledky se na konci úlohy zkombinují.

Future: handler pro výsledek nějakého výpočtu, který bude dostupný až později. Ten výpočet běží asynchronně, volající proces (klient) čeká na výsledek (blokujícím způsobem, operace get) až ve chvíli, kdy tu hodnotu potřebuje.

22 SQL (specializace PVS)

Předpokládejme následující relační schéma **databáze zoologické zahrady**:

Zamestnanec (rodneCislo, jmeno, prijmeni, oddeleni)
  Primární klíč: rodneCislo
Zvire (jmeno, druh, vek, zemePuvodu)
  Primární klíč: jmeno, druh
Osetrovani (id, jmeno, druh, zamestnanec, datumZacatku, datumKonce)
  Primární klíč: id
  Unikátní klíč: jmeno, druh, zamestnanec, datumZacatku
  Cizí klíč: Osetrovani[zamestnanec] $\subseteq$ Zamestnanec[rodneCislo]
  Cizí klíč: Osetrovani[jmeno, druh] $\subseteq$ Zvire[jmeno, druh]

Vytvořte SQL SELECT výrazy pro následující dotazy:

1. Najděte unikátní rodná čísla zaměstnanců, kteří alespoň v jednom okamžiku ošetřovali alespoň dvě zvířata najednou.
2. Najděte křestní jména a příjmení zaměstnanců z oddělení *kočkovitých šelem*, kteří nikdy neošetřovali nějaké zvíře mající zemi původu *Maroko* nebo *Tunisko*.

Nástin řešení

```
SELECT DISTINCT O1.zamestnanec
FROM Osetrovani AS O1 JOIN Osetrovani AS O2 ON
  (O1.zamestnanec = O2.zamestnanec) AND
  (O1.id < O2.id) AND
```

```
  (O1.datumZacatku <= O2.datumKonce) AND (O1.datumKonce >= O2.datumZacatku)

SELECT jmeno, prijmeni
FROM Zamestnanec
WHERE (oddeleni = 'kočkovité šelmy') AND NOT EXISTS (
  SELECT *
  FROM Zvire NATURAL JOIN Osetrovani
  WHERE (zemePuvodu = 'Maroko' OR zemePuvodu = 'Tunisko') AND (zamestnanec = rodneCislo)
)
```

23 Analýza požadavků a návrh uživatelských testů (specializace PVS)

Pracujete na nové platformě pro streamování filmů. Ze schůzky se stakeholdery jste si přinesli následující poznámku: "Naši uživatelé chtějí mít možnost jednoduše najít filmy, které je zajímají. Někteří uživatelé chtějí vyhledávat klasicky podle názvu, roku vydání, žánru, apod. Toto klasické vyhledávání tedy do platformy zavedeme. Je ale vhodné jen pro uživatele, kteří vědí, co hledají. Pro řadu uživatelů je to složité, protože přesně nevědí, na co by se chtěli podívat. Z průzkumů jsme zjistili, že by uživatelé také chtěli, aby jim naše platforma doporučovala filmy podle streamovací historie jejich přátel. Zavedeme tedy pro každého uživatele možnost si zobrazit seznam takových doporučení. K tomu musíme mít od uživatele povolení, zda si můžeme načíst seznamy jeho přátel z jeho sociálních sítí. Ze začátku chceme umět jen načítání ze sociálních sítí podporujících Facebook Graph API, ale to musí být snadno rozšiřitelné i na další druhy API."

1. Zahrnuje poznámka pouze funkční požadavky? Odpovězte ano/ne a poté vysvětelete.
2. Identifikujte alespoň 2 různé funkční požadavky a vyjádřete je jako user stories.
3. Vyberte si jeden z vašich user stories a vyjádřete jej jako strukturovaný use case (případ užití). Uveďte všechny části popisu use case, které typicky uvažujeme.
4. Pro váš use case navrhněte test case, který použijeme k otestování implementace tohoto use case.

Nástin řešení

1. Řešení by mělo popsat, že existují funkční i kvalitativní (ne-funkční) požadavky. Poslední věta v poznámce je kvalitativní (ne-funkční) požadavek na snadnou rozšiřitelnost, kvalitativní požadavek je také bezpečnost (naše platforma bude mít přístup k sociálním sítím uživatele - nutno řešit bezpečnost).
2. User story = jednoduchý popis jednoho funkčního požadavku v jedné větě striktně ve tvaru "Who, What, Why", tj. např. "Jako uživatel chci mít možnost vyhledávat filmy podle názvu, roku vydání, žánru apod., protože když vím, na jaký film se chci podívat, potřebuju mít možnost jej jednoduše a přímočaře vyhledat a začít se dívat.". Chybějící Who nebo What nebo Why je chyba.
3. Řešení může obsahovat i ten nejjednoduší user story, use case ale musí mít strukturu "název, actor, preconditions, main path/happy path jako sekvence kroků actora a systému, postconditions, alternative paths/what can go wrong".
4. Test case může popisovat základní průchod navrženým use case, ale musí být strukturován na název, alespoň jednu testing condition (co z use case testujeme), testovací data, testovací scénář jako sekvenci atomických akcí testera a snadno ověřitelných reakcí systému. Uvažujeme textové popisy pro lidského testera, automatizovaný test např. v Robot Framework není vyžadován.

24 Koherence cache (specializace SP)

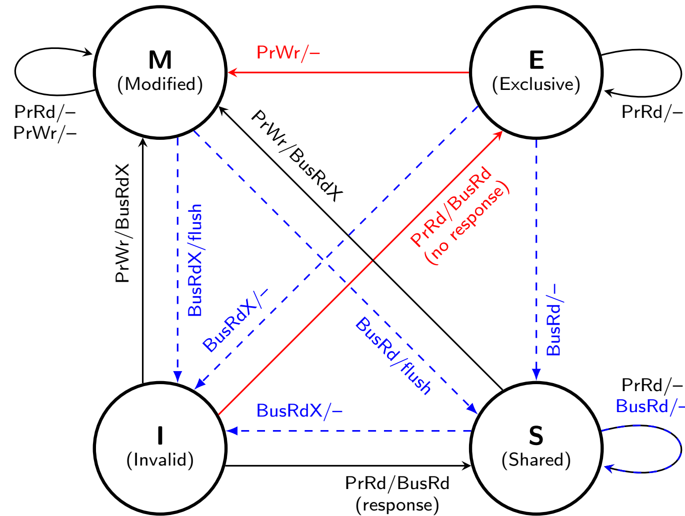
Uvažujte systém se třemi procesory, které pro dosažení koherence svých caches používají protokol MESI. Jeden ze tří procesorů má zamčený spin lock `locked`, o který se další dva ucházejí čekáním v následující smyčce:

```
while (atomic_read_and_set (&locked)) {
    // no-op
}
```

Funkce `bool atomic_read_and_set (bool *ptr)` atomicky přečte hodnotu z adresy `ptr`, zapíše na adresu `ptr` hodnotu `true`, a nakonec přečtenou hodnotu vrátí volajícímu.

1. Popište významy jednotlivých stavů cache line v protokolu MESI.
2. Popište (všechny) přechody, které protokol MESI vykonává během čekání v uvedené smyčce (včetně operací sběrnice).
3. Navrhněte úpravu uvedené smyčky, která oproti uvažovanému scénáři minimalizuje počet operací sběrnice, a vysvětlete její funkci.

Jako pomůcku můžete použít přechodový diagram protokolu MESI:



Nástin řešení

1. M – cache line obsahuje dirty data existující v jediné kopii, E – cache line obsahuje clean data existující v jediné kopii (a v paměti), S – cache line obsahuje clean data potenciálně existující ve více kopiích (a v paměti), I – cache line neobsahuje platná data.
2. Procesor držící spin lock není z pohledu otázky zajímavý – lock drží a tedy s ním již neoperuje, jeho cache line bude po první iteraci čekání ve stavu I, cache line jednoho z čekajících procesorů pak ve stavu M a druhého ve stavu I. Pokud bude dále pokračovat čekající procesor s cache line ve stavu M, operace `atomic_read_and_set` se vykoná přímo v cache line, beze změny stavu cache line a bez operace sběrnice. Pokud bude dále pokračovat procesor s cache line ve stavu I, požádá o exkluzivní čtení, které díky snooping vyvolá u druhého procesoru zápis do paměti a přechod cache line ze stavu M do stavu I, zde přejde cache line do stavu E a se zápisem do stavu M.
3. Při pravidelném střídání obou čekajících procesorů po každé iteraci čekání připadá na jednu iteraci jeden zápis do paměti a jedno čtení z paměti. Pokud se do těla smyčky přidá ještě čtecí cyklus `while (locked) {}`, budou mít po první iteraci čekání oba čekající procesory cache line ve stavu S a dále se čekání obejde bez operace sběrnice.

25 Pozičně (ne)závislý program (specializace SP)

Tento program v jazyce C byl předložen překladači gcc ve dvou konfiguracích, A a B:

```
volatile int i = 0;

int main (void) {
    return i;
}
```

Dekompilace funkce `main` ze spustitelného souboru vyrobeného překladačem gcc v konfiguraci A je:

```
08049156 <main>:
 8049156:       a1 0c c0 04 08          mov    0x804c00c,%eax
 804915b:       c3                      ret
```

Dekompilace funkce `main` ze spustitelného souboru vyrobeného překladačem gcc v konfiguraci B je:

```
0000113d <main>:
```

```
113d:       e8 0e 00 00 00          call   1150 <__x86.get_pc_thunk.ax>
1142:       05 b2 2e 00 00          add    $0x2eb2,%eax
1147:       8b 80 f8 ff ff ff       mov    -0x8(%eax),%eax
114d:       8b 00                   mov    (%eax),%eax
114f:       c3                      ret
```

1. Definujte pozičně (ne)závislý kód (position (in)dependent code).
2. Rozhodněte, která (či které, pokud nějaké) konfigurace překladače vyrobily pozičně závislý a které pozičně nezávislý program. Své rozhodnutí detailně zdůvodněte.
3. Rozhodněte, zda je pro daný program vhodnější pozičně závislý nebo pozičně nezávislý překlad. Své rozhodnutí zdůvodněte.

Nástin řešení

1. Pozičně nezávislý kód je takový, který lze v paměti umístit od libovolné adresy (bez změny operačního kódu instrukcí) bez vlivu na jeho funkci. Pozičně závislý kód tuto vlastnost nemá.
2. Varianta A je pozičně závislá, varianta B je pozičně nezávislá. Kód varianty A obsahuje adresu proměnné `i` jako absolutní konstantu `0x804c00c` uloženou v operačním kódu instrukce `mov`. Kód varianty B vyzvedává adresu proměnné `i` z globální tabulky adres (GOT), jejíž adresu vypočítává z relativní pozice programu (PC).
3. Lze argumentovat tím, že pro kód aplikace (v kontrastu s kódem knihoven) je poziční nezávislost zbytečná a přináší pouze režii, nebo naopak tím, že randomizace adresového prostoru je dnes běžné bezpečnostní opatření, pro které se poziční nezávislost kódu aplikace hodí.

26 Paralelní programování (specializace SP)

1. Vysvětlete koncept future (promise) a jeho využití v paralelním či asynchronním programování.
2. Definujte základní rozhraní objektu reprezentujícího future (promise) a vysvětlete sémantiku všech metod.
3. Upravte níže uvedený kód tak, aby konverzi obrázků prováděl paralelně s využitím futures (promises). Názvy použitých tříd a metod nemusí odpovídat třídám v existujících běhových prostředích, ale je nutné vysvětlit jejich význam.

   Řešení by se mělo vyhnout neomezenému paralelizmu a musí zachovat sekvenční čtení a zápis souborů. Po přečtení všech souborů by mělo zapisovat již transformované soubory bez ohledu na pořadí jejich načtení, tj. pokud bude např. pátý soubor převeden dříve než třetí, řešení by nemělo čekat na dokončení konverze třetího souboru před zápisem pátého.

Pro zjednodušení je kód psán formou pseudokódu, ve kterém by nemělo příliš záviset na použitém jazyce. Uvedené metody vnímejte jako statické metody nějaké třídy, která zde není podstatná. Sémantika metod a použitých typů by měla být zřejmá z názvů. Pro jednoduchost se v kódu neřeší (a v řešení není třeba řešit) výjimky.

```
1  byte[] readFile(Path path) { ... }
2
3  byte[] convertToGrayscale(byte[] bytes) { ... }
4
5  Path getOutputPath(Path path) { ... }
6
7  void saveFile(byte[] bytes, Path path) { ... }
8
9  Iterable<Path> recursivelyListFiles(String directory, String glob) { ... }
10
11 void convertFiles(Iterable<Path> imagePaths) {
12     foreach (Path inputPath in imagePaths) {
13         byte[] inputBytes = readFile(inputPath);
14         byte[] outputBytes = convertToGrayscale(inputBytes);
15         Path outputFile = getOutputPath(inputPath);
16         saveFile(outputBytes, outputPath);
17     }
```

```
18 }
19
20 void main(String[] args) {
21     Iterable<Path> imagePaths = recursivelyListFiles("/path/to/images", "*.png");
22     convertFiles(imagePaths);
23 }
```

Nástin řešení

1. Future (promise) je abstrakce pro hodnotu, která nemusí být okamžitě dostupná, ale bude dostupná někdy v budoucnosti, případně nemusí být dostupná vůbec (v případě selhání). Futures se používají k reprezentaci výsledků operací, které mohou trvat neznámou dobu a běží paralelně či asynchronně s vláknem, které na výsledcích závisí. To vláknům umožňuje vykonávat jiné operace do doby, než je výsledek reprezentovaný future potřeba.
2. Rozhraní by mělo být generické pro nějaký typ `T` a poskytovat alespoň blokující metodu pro získání hodnoty a metodu pro zjištění, jestli je future kompletní (pro případné aktivní čekání). Flexibilnější rozhraní mají řadu jiných metod, které umožňují přerušit výpočet spojení s future, nastavit funkci, která se má zavolat při dokončení, transformovat hodnotu, apod.
3. Možné řešení je uvedené níže. Místo `CompletionService` je možné použít thread-safe blokující frontu, do které by completion handler každé future vložil instanci `ImageData`, a ze které by hlavní vlákno dokončené úlohy odebíralo a ukládalo na disk. Akceptovatelné řešení je rovněž takové, které by instance `ImageData` do fronty ukládalo na konci tasku reprezentovaného future.

```
1  record ImageData (Path path, byte[] bytes) {}
2
3  ExecutorService getCpuLimitedThreadPool() { ... }
4
5  CompletionService getCompletionService(ExecutorService executor) { ... }
6
7  void convertFiles(Iterable<Path> imagePaths) {
8      ExecutorService executor = getCpuLimitedThreadPool();
9      CompletionService <ImageData> completionService = getCompletionService(executor);
10
11     int count = 0;
12     for (Path inputPath : imagePaths) {
13         byte[] inputBytes = readFile(inputPath);
14
15         completionService.submit(() -> {
16             Path outputPath = getOutputPath(inputPath);
17             byte[] outputBytes = convertToGrayscale(inputBytes);
18             return new ImageData(outputPath, outputBytes);
19         });
20
21         count++;
22     }
23
24     for (int i = 0; i < count; i++) {
25         ImageData outputImage = completionService.take().get();
26         saveFile(outputImage.bytes, outputImage.path);
27     }
28
29     executor.shutdown();
30 }
```

27 Principy objektového návrhu (specializace SP)

1. Vysvětlete, co je podstatou tzv. *substitučního principu* (Liskov Substitution Principle, LSP) v objektovém návrhu.

2. V kódu na další stránce identifikujte, kde dochází k porušení LSP, a vysvětlete proč.
3. Upravte objektový návrh v uvedeném kódu tak, aby v daném kontextu vyhovoval požadavkům LSP, a úpravy zdůvodněte. Řešení nemusí zachovat stávající třídy.

```
1  /** Bezny ucet. */
2  class Account {
3      protected double balance;
4
5      /** Vlozi castku na ucet.
6       * @param amount castka, ktera ma byt vlozena
7       * @throws IllegalArgumentException pokud je amount <= 0
8       */
9      public void deposit(double amount) {
10         if (amount <= 0) {
11             throw new IllegalArgumentException(...);
12         }
13         balance += amount;
14     }
15
16     /** Vybere castku z uctu.
17      * @param amount castka, ktera ma byt vybrana
18      * @throws IllegalArgumentException pokud je amount <= 0
19      * @throws IllegalStateException pokud na uctu neni dostatek prostredku
20      */
21     public void withdraw(double amount) {
22         if (amount <= 0) {
23             throw new IllegalArgumentException(...);
24         }
25         if (amount > balance) {
26             throw new IllegalStateException(...);
27         }
28         balance -= amount;
29     }
30
31     // ... ostatni metody (vcetne konstruktoru) ...
32 }
33
34 /** Sporici ucet. */
35 class SavingsAccount extends Account {
36     /** Limit pro vyber z uctu. */
37     private double withdrawalLimit = 1000;
38
39     /** Vybere castku z uctu.
40      * @param amount castka, ktera ma byt vybrana
41      * @throws IllegalArgumentException pokud je amount <= 0
42      * @throws IllegalOperationException pokud je amount > withdrawalLimit
43      * @throws IllegalStateException pokud na uctu neni dostatek prostredku
44      */
45     @Override
46     public void withdraw(double amount) {
47         if (amount > withdrawalLimit) {
48             throw new IllegalOperationException(...);
49         }
50         super.withdraw(amount);
51     }
52
53     // ... ostatni metody (vcetne konstruktoru) ...
54 }
```

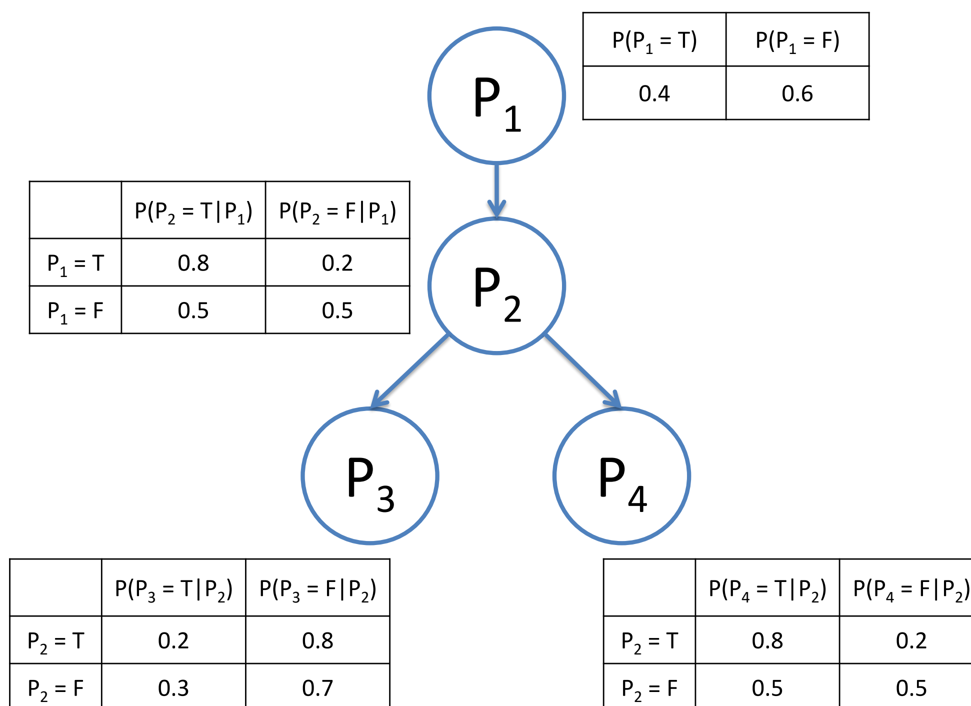
Nástin řešení

1. LSP poskytuje návod, jak vytvářet správné podtypy. Instance tříd odvozené od base class by mělo být možné použít všude, kde jsou očekávány instance base class, aniž by to ovlivnilo správnost programu. Odvozené třídy nesmí posilovat preconditions a oslabovat postconditions a musí zachovávat invarianty.
2. Třída `SavingsAcount` porušuje LSP v metodě `withdraw`, protože posiluje precondition (`amount > withdrawalLimit`) a vyhazuje výjimku `IllegalOperationException`, kterou kód používající třídu `Account` nemůže očekávat.
3. To, že některé výběry nemusí být povoleny, je potřeba zachytit v kontraktu třídy `Account`. Je nutné zavést vhodný mechanizmus, který umožní definovat, kdy je možné výběr provést, a třída `Account` toto musí explicitně kontrolovat. Např. by bylo možné zavést protected metodu `canWithdraw`, kterou by podtřídy mohly předefinovat. Alternativně by bylo možné instancím třídy `Account` předat strategii, kterou by metoda `canWithdraw` volala.

28 Bayesovské sítě (specializace UI-SU, UI-ZPJ)

Definujte Bayesovskou síť a napište, jak se takové sítě používají a jak se konstruují. Jaký je vztah mezi Bayesovskou sítí a úplnou sdruženou pravděpodobnostní distribucí?

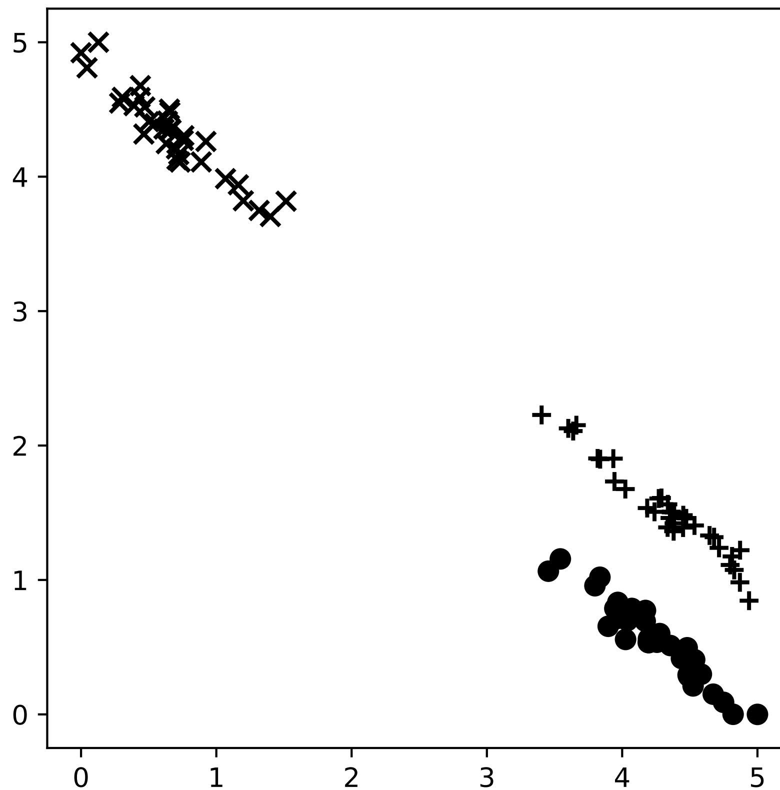
Uvažujte níže zobrazenou Bayesovskou síť se 4 náhodnými proměnnými $P_1, \ldots, P_4$ a uvedenými podmíněnými pravděpodobnostními distribucemi. Máme pozorování $P_3 = F$ a $P_4 = T$. Spočítejte, jaká je pravděpodobnost, že $P_1 = T$, tedy $P(P_1 = T | P_3 = F, P_4 = T)$.



29 PCA analýza (specializace UI-SU, UI-ZPJ)

Mějme data se dvěma atributy a ze třech tříd zobrazená na obrázku níže.

1. Vysvětlete, jak funguje PCA analýza. Jakým způsobem můžeme spočítat hlavní komponenty?

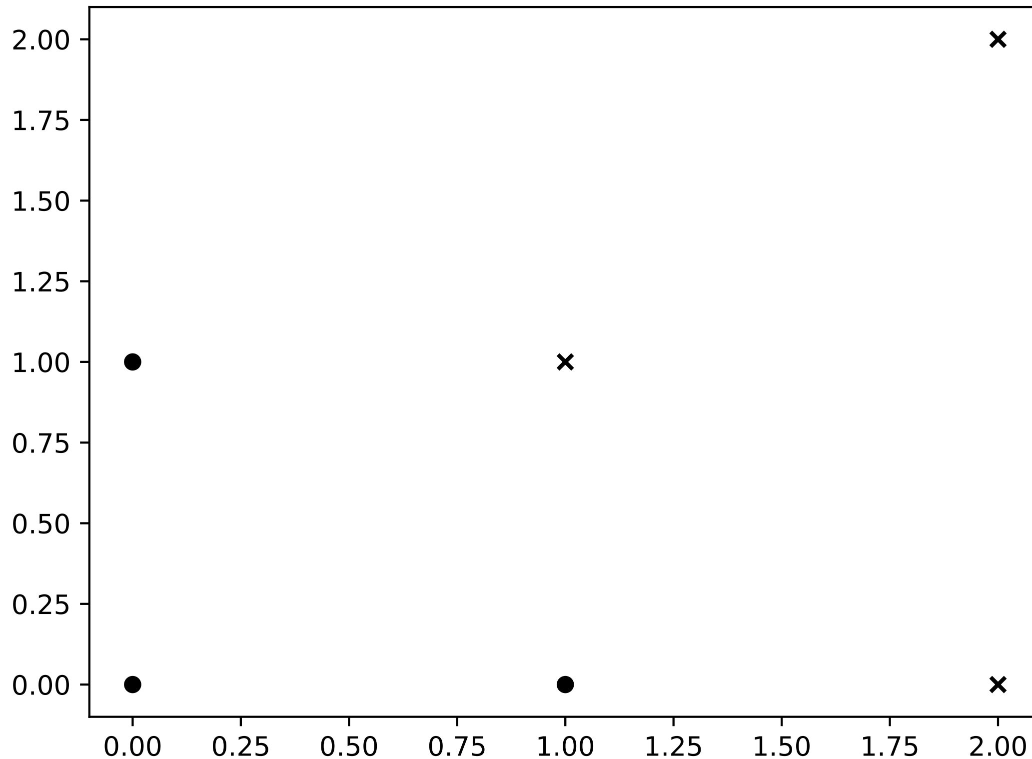


2. Bez počítání odhadněte, jaké budou směry dvou hlavních komponent pro data na obrázku výše. Zakreslete je do obrázku.
3. Uvažujme klasifikátor založený na metodě k nejbližších sousedů pro vhodné k a předpokládejme, že jsme v rámci předzpracování dat použili PCA analýzu pro snížení počtu dimenzí na 1. Jaká bude přesnost tohoto klasifikátoru v porovnání s přesnostní stejného klasifikátoru trénovaného na původních datech? Vysvětlete.

30 Metoda podpůrných vektorů (SVM) (specializace UI-SU, UI-ZPJ)

Uvažujme data se dvěmi příznaky, která chceme klasifikovat do dvou tříd. Data pro třídu A jsou na pozicích (1,1), (2,2) a (2,0), data pro třídu B jsou na pozicích (0,0), (1,0) a (0,1). Data jsou také zobrazena na obrázku níže. Třída A je označena jako křížek ($\times$), třída B jako kolečko ($\bullet$).

1. Popište, jak fungují a jak se trénují lineární support vector machines (metoda podpůrných vektorů) v případě, že předpokládáme, že data jsou lineárně separabilní.
2. Na základě dat výše zakreslete přibližnou rozhodovací hranici linárního SVM, která odděluje tyto dvě třídy. Nemusíte nic počítat.
3. Aniž byste cokoliv počítali, vysvětlete, jak přidání nového bodu (3,3) do třídy A ovlivní rozhodovací hranici lineárního SVM.
4. Vylepšilo by na datech uvedených výše použití RBF jádra klasifikaci? Zakreslete nějaký dataset, kde by RBF jádra pomohla.



31 Metoda nejmenších čtverců (specializace UI-SU)

Mějme data s jedním číselným atributem x_1 a číselnou cílovou hodnotou y zadaná v tabulce níže. Pro tato data chceme najít lineární model metodou nejmenších čtverců.

x_1	y
1	1
2	3
3	2

1. Popište lineární model a metodu nejmenších čtverců. Jakým způsobem se touto metodou odhadují koeficienty modelu?
2. Odhadněte koeficienty modelu pro data zadaná v tabulce výše.
3. Spočítejte střední kvadratickou chybu získaného modelu.

32 Evaluační metrika BLEU (specializace UI-ZPJ)

Vysvětlete, k čemu se používá evaluační metrika BLEU. Uveďte vzorec pro výpočet BLEU (nebo alespoň jeho přibližnou strukturu) a popište účel obou hlavních částí vzorce.

Nástin řešení

- BLEU is used for evaluating quality of Machine Translation systems (or, possibly, of other NLP systems in which sentences are generated) by comparing a generated output with a reference output (or more of reference outputs).
- The formula should include the precision factor, either as an exact formula or at least a sketch showing that it is some kind of average of n-gram precisions, with a textual description (for 1-grams to 4-gram present in the generated output, their average precision (presence in the reference output(s)) is computed).
- The formula should include the brevity-penalty factor, with a textual description (degenerate solutions that reach high n-gram precisions by picking only easy-to-translate parts must be penalized).