# Bakalářské zkoušky (příklady otázek z informatiky)

jaro 2023

## 1 Konkatenace palindromů (3 body)

1. Uveďte formální definice všech typů gramatik v Chomského hierarchii.
2. *Palindrom* je slovo, které se čte zprava doleva stejně jako zleva doprava (tedy slovo splňující $w = w^R$). Nechť $L$ je jazyk sestávající ze všech slov nad abecedou $\{a, b\}$, která jsou konkatenací nějaké dvojice (ne nutně různých) palindromů. Zařaďte jazyk $L$ do Chomského hierarchie a sestrojte gramatiku příslušného typu, která jej generuje. (Nemusíte zdůvodňovat, proč je jazyk generovaný zkonstruovanou gramatikou, ani proč neexistuje gramatika vyššího typu.)

## 2 Třídění (3 body)

1. Popište, jak pomocí binárního vyhledávacího stromu setřídit (uspořádat vzestupně) posloupnost $x_1, \ldots, x_n$ navzájem různých prvků.
2. Co se změní, pokud se prvky mohou opakovat?
3. Co z toho plyne pro minimální možnou složitost operací s vyhledávacím stromem? Předpokládejme přitom, že klíče ve stromu je možné pouze porovnávat.
4. Jakou bude mít váš algoritmus z bodu 1 časovou složitost, pokud jako strom použijeme AVL strom a prvky posloupnosti budou řetězce délky $L$?

## 3 Databáze (3 body)

1. Uvažujte transakce $T_1$: $W(A)$ $R(B)$ $COMMIT$ a $T_2$: $W(B)$ $W(C)$ $COMMIT$. Je rozvrh $S$: $W_1(A)$ $W_2(B)$ $R_1(B)$ $COMMIT_1$ $W_2(C)$ $COMMIT_2$ zotavitelný (recoverable)? Pokud ano, proč? Pokud ne, jak by bylo potřeba jej upravit, aby zotavitelný byl? Operace $R$ reprezentuje čtení dané proměnné, operace $W$ reprezentuje její zápis.
2. Popište podmínky nutné pro to, aby relační schéma bylo ve třetí normální formě. Uveďte jednoduchý příklad relace, která v dané normální normě nebude.
3. Jak se na konceptuální úrovni liší binární vztah mezi entitami *Osoba* a $Telefon$ – pokud vůbec – jestliže je jednou reprezentován relací $OsobaTelefon(\underline{OsobaID, TelefonID})$, a podruhé relací $OsobaTelefon(\underline{OsobaID}, \underline{TelefonID})$, tedy jednou má relace jeden složený dvousloupcový klíč a jednou dva nezávislé jednosloupcové klíče. Klíč(e) relace jsou znázorněny podtržením.

## 4 Objektový návrh (3 body)

Předpokládejte, že implementujeme aplikaci pro úpravu bitmapových obrázků. Uživatel bude moci mít otevřených více obrázků najednou. Nový obrázek si uživatel otevře v uživatelském rozhraní aplikace (UI) výběrem jména souboru – aplikace dle přípony souboru rozhodne o jeho formátu. U otevřeného obrázku si uživatel může postupně vybírat některé z nabízených filtrů – v UI aplikace si uživatel vždy vybere filtr a jeho konkrétní nastavení, ten se uloží do seznamu vybraných filtrů. Nakonec bude mít uživatel možnost všechny filtry najednou aplikovat. Každý otevřený soubor si uživatel může kdykoliv uložit buď v původním nebo jiném podporovaném bitmapovém formátu. Pro jednoduchost předpokládejte, že jednomu formátu obrázku odpovídá právě jedna přípona souboru (tj. např. pro JPEG formát jen přípona `.jpg` a nikoliv už `.jpeg`), a že všechny filtry mohou libovolně měnit obrazová data, ale rozměry obrázku v pixelech vždy zůstávají zachované.

Zatím máme v C# připravenou níže uvedenou kostru hlavních tříd našeho programu, kde třída `Image` reprezentuje načtená obrazová data (další metadata jako původní jméno souboru atd. si budeme pamatovat jinde; poznámka: `[,]` v C# znamená dvojrozměrné pole), a metoda `Program.Main` představuje jednoduchý scénář využití uvedených tříd:

```
struct Color { public byte R, G, B; }

static class Filters {
        public static void ApplySharpness(Image image, float amount, float radius) { ... }
        public static void ApplyBrightness(Image image, float amount) { ... }
}

abstract class Image {
        public Color[,] bitmap = null;
        public abstract void Load(string fileName);
        public abstract void Save(string fileName);
}

class PngImage : Image {
        public override void Load(string fileName) { ... }
        public override void Save(string fileName) { ... }
}

class JpegImage : Image {
        public override void Load(string fileName) { ... }
        public override void Save(string fileName) { ... }
}

static class ImageLoader {
        public static Image Load(string fileName) {
                Image image;
                if (fileName.EndsWith(".png")) {
                        image = new PngImage();
                } else if (fileName.EndsWith(".jpg")) {
                        image = new JpegImage();
                } else {
                        throw new ArgumentException();
                }
                image.Load(fileName);
                return image;
        }
}

class Program {
        public static void Main() {
                var fileName = "a.jpg";
                var image = ImageLoader.Load(fileName);
                Filters.ApplySharpness(image, 9.0f, 1.3f);
                Filters.ApplyBrightness(image, 1.7f);
                image.Save(fileName);
        }
}
```

Přepracujte celý objektový návrh tak, aby lépe umožnoval implementovat, udržovat a dále rozšiřovat uvedenou aplikaci. Novou kostru kódu zapište v C#, C++, nebo Javě. Soustřeďte se na návrh typů a jejich vztahů (vše zapisujte v syntaxi zvoleného jazyka), deklaraci metod a pouze klíčových vlastností (properties) a datových položek. Implementaci většiny metod uvádět nemusíte – pouze upravte implementaci `Program.Main` tak, aby vhodně využívala váš OO návrh. Ve svém řešení uveďte také implementaci kódu, který bude ve vašem návrhu implementovat posloupnost kroků od jména obrazového souboru k výběru formátu souboru a načtení obrázku v tomto formátu.

Ve svém návrhu zohledněte nejen snadnou možnost implementace všech aspektů výše načrtnutého scénáře používání aplikace, ale také to, že do budoucna budeme chtít přidávat velké množství různých filtrů – implementaci nového filtru ale vždy plánujeme doplňovat do hlavních zdrojových kódů aplikace; nepředpokládáme, že bychom nové filtry dodávali ve formě pluginů. Tedy ve svém řešení zohledněte pouze možnost batchové aplikace předvybraných filtrů, ale nemusíte řešit žádnou komplikovanou infrastrukturu pro jejich obecný popis.

Formou pluginů naopak budeme chtít přidávat podporu pro nové obrazové formáty. Předpokládejte tedy, že každý nově

podporovaný obrazový formát budeme implementovat ve formě nového pluginu naší aplikace (který budeme za běhu načítat jako dynamicky linkovanou knihovnu) – chceme tedy, aby proces načítání obrázků ze souboru byl dostatečně flexibilní a umožňoval nám za běhu dodat odkaz na nový formát souborů, který bychom nalezli v nějakém pluginu (samotné načítání pluginů a podporu pro načítání dynamicky linkovaných knihoven, ani přímo hledání popisu formátu v pluginu neřešte).

# 5 Organizace paměti procesu (3 body)

V obou částech předpokládejte kontext nějakého běžného desktopového operačního systému a procesu, který vznikl spuštěním aplikace napsané v C++, C# nebo Javě.

**Část A**

1. Adresový prostor každého procesu je rozdělen na různé oblasti obsahující kód programu a data. Jaké hlavní oblasti pro data (lišící se životností dat a způsobem alokace a dealokace prostoru pro jejich uložení) bude adresový prostor procesu obsahovat?
2. Pro každou takovou oblast vysvětlete, jaká data se zde typicky nacházejí, kdy v ní dochází k alokaci a dealokaci dat a kdo je za ni zodpovědný. Ve vašem vysvětlení ilustrujte alokaci a dealokaci na fragmentu kódu v C++, C# nebo Javě.

**Část B**

Předpokládejte, že jazyk pseudokódu uvedeného níže podporuje koncept ukazatelů a operátor, který umožňuje získat adresu, na které je v paměti uložena hodnota proměnné. Pokud `T` reprezentuje typ proměnné, potom `T*` reprezentuje typ *ukazatel na hodnotu typu `T`* (v jazyce Pascal se totéž zapisuje jako `^T`). Pro získání adresy proměnné je potřeba aplikovat na proměnnou operátor `&`. Pokud tedy máme proměnnou `var` typu `T`, potom výsledkem operace `&var` je typovaný ukazatel (tj. hodnota typu `T*`), která reprezentuje adresu proměnné `var` (v jazyce Pascal ke stejnému účelu slouží operátor `@`).

Např. celočíselnou proměnnou `number` typu `int` s hodnotou `42` definujeme jako `int number = 42`. Proměnnou `number_ptr` typu *ukazatel na typ `int`*, která obsahuje adresu proměnné `number`, bychom pak definovali jako `int* number_ptr = &number`.

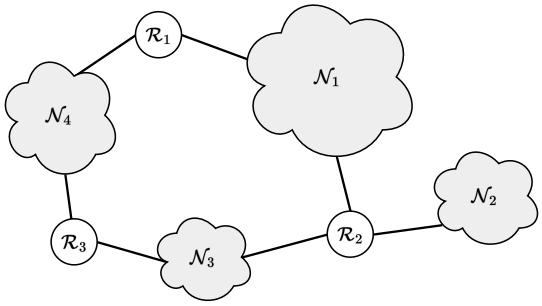
V následujícím programu tedy funkce `run()` zobrazí pro každý z bloků kódu A, B a C (označených komentáři) posloupnost různých adres. Předpokládejte, že funkce byla zavolána z hlavního vlákna, a že žádná jiná vlákna v programu neběží. Rovněž předpokládejte, že kód běží tak, jak je napsán, tj. překladač ho nijak neoptimalizoval. Velikosti datových typů uvažujte typické, tj. 4 byty pro `int` a 8 bytů pro `long` a ukazatele.

```
1  void descend(int steps) {
2      int level = steps;
3      print_pointer(&level);
4
5      if (steps > 0) {
6          descend(steps - 1);
7      }
8  }
9
10 class Record {
11     int value1;
12     long value2;
13 }
14
15 Record* records[100];
16
17 void run() {
18     // Block A
19     descend(100);
20
21     // Block B
22     for (int i = 0; i < records.length; i++) {
23        Record* record = new Record();
24        print_pointer(record);
25        records[i] = record;
26     }
27
28     // Block C
29     for (int i = 0; i < records.length; i++) {
30         print_pointer(&(records[i]));
31     }
32 }
```

Pro každý z bloků A, B a C charakterizujte hodnoty adres, které by program mohl zobrazit. Absolutní hodnoty adres nejsou podstatné — v rámci každého bloku se zaměřte na to, jakými hodnotami budou adresy dělitelné, zda bude posloupnost adres rostoucí nebo klesající, jak velké budou rozdíly mezi po sobě jdoucími adresami a jaká musí být minimální vzdálenost mezi adresami. **Charakteristiku každého bloku adres zdůvodněte!**

# 6 Směrování (3 body)

Mějme soustavu sítí $\mathcal{N}_1$ až $\mathcal{N}_4$ vzájemně propojených pomocí směrovačů $\mathcal{R}_1$ až $\mathcal{R}_3$ s topologií uvedenou na následujícím obrázku.




Směrovací tabulky jednotlivých směrovačů jsou definovány takto:

$\mathcal{R}_1$

| Destination | Netmask | Interface | Gateway | Metric |
|---|---|---|---|---|
| 210.18.76.0 | 255.255.255.0 | (A) 210.18.76.101 | on-link | 1 |
| 195.113.0.0 | 255.255.0.0 | (B) 195.113.19.20 | on-link | 1 |
| 210.18.0.0 | 255.255.0.0 | (A) 210.18.76.101 | (G) 210.18.76.202 | 10 |

$\mathcal{R}_2$

| | | | | |
|---|---|---|---|---|
| 195.113.0.0 | 255.255.0.0 | (C) 195.113.19.30 | on-link | 1 |
| 210.18.25.0 | 255.255.255.0 | (D) 210.18.25.10 | on-link | 1 |
| 210.18.35.0 | 255.255.255.0 | (E) 210.18.35.88 | on-link | 1 |

$\mathcal{R}_3$

| | | | | |
|---|---|---|---|---|
| 210.18.35.0 | 255.255.255.0 | (F) 210.18.35.99 | on-link | 1 |
| 210.18.76.0 | 255.255.255.0 | (G) 210.18.76.202 | on-link | 1 |
| default | | (F) 210.18.35.99 | (E) 210.18.35.88 | 20 |

Koncové uzly v síti $\mathcal{N}_1$ směrují veškerý nelokální provoz na směrovač $\mathcal{R}_1$, analogicky v síti $\mathcal{N}_2$ na $\mathcal{R}_2$, v síti $\mathcal{N}_3$ na $\mathcal{R}_3$ a konečně v síti $\mathcal{N}_4$ také na $\mathcal{R}_3$.

Předpokládejme postupné odeslání následujících IPv4 datagramů:

1. Datagram od (H) 210.18.35.152 pro (I) 210.18.76.18
2. Datagram od (I) 210.18.76.18 pro (J) 210.18.76.65
3. Datagram od (K) 195.113.45.17 pro (L) 210.18.25.74
4. Datagram od (L) 210.18.25.74 pro (J) 210.18.76.65
5. Datagram od (J) 210.18.76.65 pro (A) 210.18.76.101
6. Datagram od (H) 210.18.35.152 pro (M) 255.255.255.255
7. Datagram od (I) 210.18.76.18 pro (N) 210.18.95.38
8. Datagram od (H) 210.18.35.152 pro (O) 210.18.25.255

Pro zjednodušení naší práce budeme všechny použité IP adresy a jim odpovídající unicastové MAC adresy symbolicky označovat pomocí uvedených písmen (A) až (O).

Pro každou z předchozích situací určete, jakým způsobem bude daný datagram doručen od odesílatele až k zamýšlenému příjemci. Tedy přes jaké sítě, jaké uzly, jaká rozhraní a na jaké adresy. Vysvětlete a odůvodněte.

# 7 Základy teorie informace (otázka studijního zaměření – 3 body)

$X$ a $Y$ jsou dvě náhodné diskrétní veličiny a $H$ je označení entropie. Uveďte, zda níže uvedené nerovnosti pro libovolné $X, Y$ **platí**, nebo **neplatí**. Pokud **platí**, dokažte. Pokud **neplatí**, uveďte protipříklad, nebo dokažte platnost opačné nerovnosti. Nápověda: použijte vztah mezi vzájemnou informací a entropií.

1. $H(X) \geq H(X|Y)$
2. $H(X) + H(Y) \leq H(X, Y)$

# 8 Morfologická, syntaktická a sémantická analýza přirozeného jazyka (otázka studijního zaměření – 3 body)

1. Vysvětlete pojmy *morfologická analýza*, *lematizace* a *tagování*.

2. Procedury ilustrujte na příkladech a použijte anotační schéma Pražského závislostního korpusu.

# 9 Základní formalismy pro popis přirozeného jazyka (otázka studijního zaměření – 3 body)

1. Vysvětlete pojem *treebank*. Uveďte dva treebanky, které znáte.
2. Popište neprojektivitu v závislostních stromech a uveďte příklad neprojektivního závislostního stromu.

# 10 Anti-aliasing (otázka studijního zaměření – 3 body)

1. Který z parametrů rastrového zobrazovacího zařízení je anti-aliasingem vylepšen? Jak se to pozná na výsledném obrázku?
2. Jak se může anti-aliasing implementovat v klasickém rastrovém vykreslování (rasterizace, např. při kreslení vektorové grafiky)?
3. Jak se anti-aliasing implementuje v prostředí paprskového zobrazovače (např. ray-tracing)?

# 11 Homogenní souřadnice a maticové transformace (otázka studijního zaměření – 3 body)

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

# 12 Fourierova transformace (otázka studijního zaměření – 3 body)

1. Definujte (napište vzorec a popište veličiny) 2D Diskrétní Fourierovu transformaci (DFT).
2. Jak se používá DFT na odstranění šumu v obrazech?
3. U obrázků na následující straně správně přiřaďte obraz a jeho DFT.

# 13 Zjednodušený rozvrh pro zkouškové období (otázka studijního zaměření – 3 body)

Uvažujte zjednodušený rozvrh pro zkouškové období. Pro každý akademický rok a semestr existuje vlastní rozvrh. Rozvrh obsahuje vypsané termíny zkoušky a zápočtových testů. Každý termín a zápočtový test obsahuje datum, čas začátku, čas konce, místnost, omezení počtu studentů (kapacita), vyučující a přiřazené předměty. K jednomu termínu může být přiřazeno 0-n předmětů. Na předměty v termínech se přihlašují studenti. Student se může přihlásit na nejvýše jeden zkouškový termín a jeden termín zápočtového testu pro daný předmět.

Vytvořte a okomentujte schéma (jedno z XML, JSON, RDF) popisující reprezentaci zjednodušeného rozvrhu pro zkouškové období. Drobné syntaktické chyby nejsou hodnoceny. Pokud si nejste jistí syntaxí vybraného modelu, popište, co by měl vámi zvolený zápis reprezentovat.

Na příkladu jednoho vybraného termínu vysvětlete datový model RDF. Dále vysvětlete, jak lze dle 4 principů Linked Data propojit vaší RDF reprezentaci vyučujících s RDF reprezentací vyučujících ve veřejně dostupné univerzitní databázi.

# 14 Objektově relační mapování (otázka studijního zaměření – 3 body)

Stručně vysvětlete koncept „objektově relačního mapování (ORM)“. Zaměřte se přitom na základní princip, dále stručně popište, jaký problém koncept řeší. Okomentujte hlavní výhody použití tohoto způsobu konstrukce softwarových systémů oproti alternativám, a to zejména s ohledem na modularitu návrhu a testování jednotlivých komponent systému.

Vytvořte jednoduchou ukázku použití ORM ve vybraném objektově-orientovaném programovacím jazyce (C#, Java, C++, Python, PHP, JavaScript, apod.). Uvažujte aplikaci pro práci se zjednodušeným rozvrhem pro zkouškové období z předchozí otázky. Stačí jenom hrubá kostra modelu pro termín a zápis studenta a příklad použití.

# 15 Návrhový vzor Model-View-Controller (MVC) (otázka studijního zaměření – 3 body)

Popište základní princip návrhového vzoru „Model-View-Controller (MVC)“ a role hlavních komponent (elementů) aplikace navržené podle tohoto vzoru.

Nakreslete diagram interakce těchto komponent, ve kterém zachytíte směr komunikace, význam předávaných informací mezi komponentami, a také interakci uživatele s danou aplikací (tedy v diagramu nějak vyjádřete, se kterými komponentami uživatel interaguje a jak primárně). Soustřeďte se na podstatné aspekty a zanedbejte méně významné technické detaily. Vhodně doplňte jednotlivé prvky diagramu textovým komentářem. Stručně popište činnosti, které se typicky provádějí v jednotlivých komponentách aplikace navržené podle vzoru MVC – tedy popište, jakou činnost v rámci takové webové aplikace provádí komponenta „Model“ a co má na starosti, a potom to samé ještě pro komponenty „Controller“ a „View“.

Vytvořte jednoduchý příklad z oblasti webových aplikací, se zaměřením na jejich serverové části. Příklad ukáže použití pro zobrazení termínu zkoušky ze zjednodušeného rozvrh pro zkouškové období. Uživatel bude moci vytvořit nový termín a existující termíny mazat pomocí HTML stránky.

Vytvořte jednoduchý příklad rozhraní hlavních komponent ve smyslu kostry nějaké jednoduché webové aplikace. Použijte váš oblíbený programovací jazyk (C#, Java, C++, Python, PHP, JavaScript, apod.). Soustřeďte se na klíčové prvky těchto rozhraní s ohledem na interakci komponent (názvy akcí, předávaná data), a zanedbejte nepodstatné technické detaily.

Popište jakým způsobem by bylo možné váš příklad rozšířit pro přidání REST API.

# 16 Metoda zpětného učení (otázka studijního zaměření – 3 body)

Mějme přepínač (switch), který za účelem filtrování (filtering) a cíleného předávání (forwarding) používá metodu zpětného učení. Přepínač má vlastní MAC adresu (A) A4-1A-3A-E3-22-AC a čtyři porty. Přepínač zatím žádnou znalost topologie sítě nemá.

Nejprve definujte zmíněné mechanismy filtrování a cíleného předávání.

Následně postupně přijmeme následující linkové rámce:

1. Rámec od (B) 00-0A-F7-5F-88-1E pro (C) 38-BA-B0-5B-02-8A na portu (1)
2. Rámec od (D) 48-5B-39-E5-B4-78 pro (B) 00-0A-F7-5F-88-1E na portu (3)
3. Rámec od (B) 00-0A-F7-5F-88-1E pro (A) A4-1A-3A-E3-22-AC na portu (1)
4. Rámec od (C) 38-BA-B0-5B-02-8A pro (F) FF-FF-FF-FF-FF-FF na portu (3)
5. Rámec od (B) 00-0A-F7-5F-88-1E pro (C) 38-BA-B0-5B-02-8A na portu (1)
6. Rámec od (C) 38-BA-B0-5B-02-8A pro (D) 48-5B-39-E5-B4-78 na portu (3)

Pro zjednodušení naší práce budeme všechny MAC (EUI-48) adresy symbolicky označovat pomocí písmen, analogicky jednotlivé porty pomocí čísel (1) až (4).

Pro každou z předchozích situací určete, jakým způsobem bude daný rámec přepínačem zpracován. Pokud bude posílán dále, jakým směrem? Jak se postupně bude měnit znalost topologie naší sítě? Vysvětlete a odůvodněte.

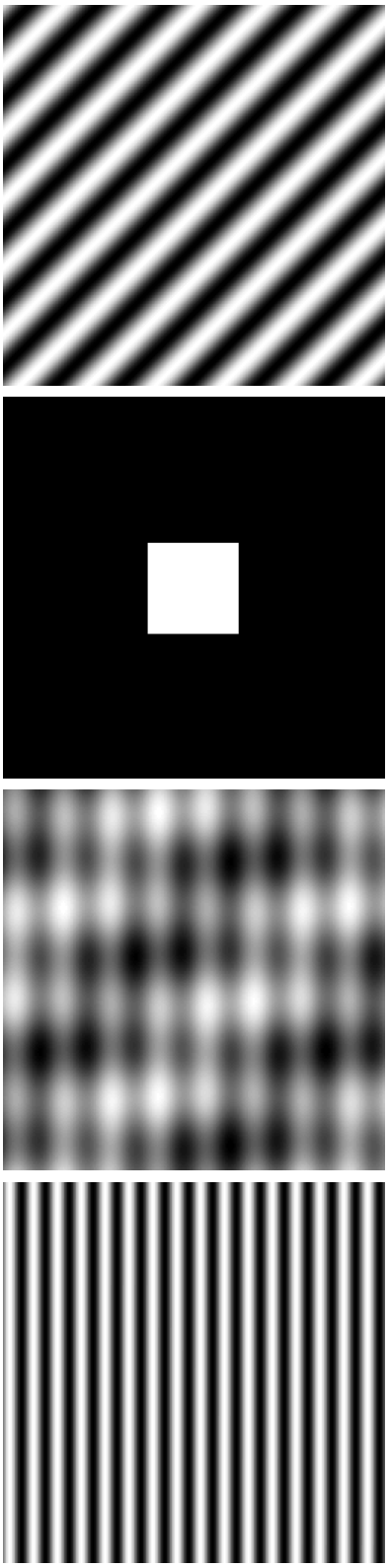
# 17 Linkové protokoly (otázka studijního zaměření – 3 body)

Navrhněte hypotetický bitově orientovaný blokový protokol pro linkovou vrstvu, který bude nespojovaný, negarantovaný (tedy Best Effort) a spolehlivý, a to v sítích s přepojováním paketů. Popište strukturu dat a význam jednotlivých položek. Stačí se věnovat jen těm, které jsou nezbytné pro zajištění funkce doručování užitečného nákladu a výše uvedených vlastností. Zajistěte transparenci dat. Popište konkrétní způsob zajištění spolehlivosti. Vše pečlivě vysvětlete a odůvodněte. Můžete aplikovat obecné principy stejně jako se inspirovat reálně existujícími protokoly.

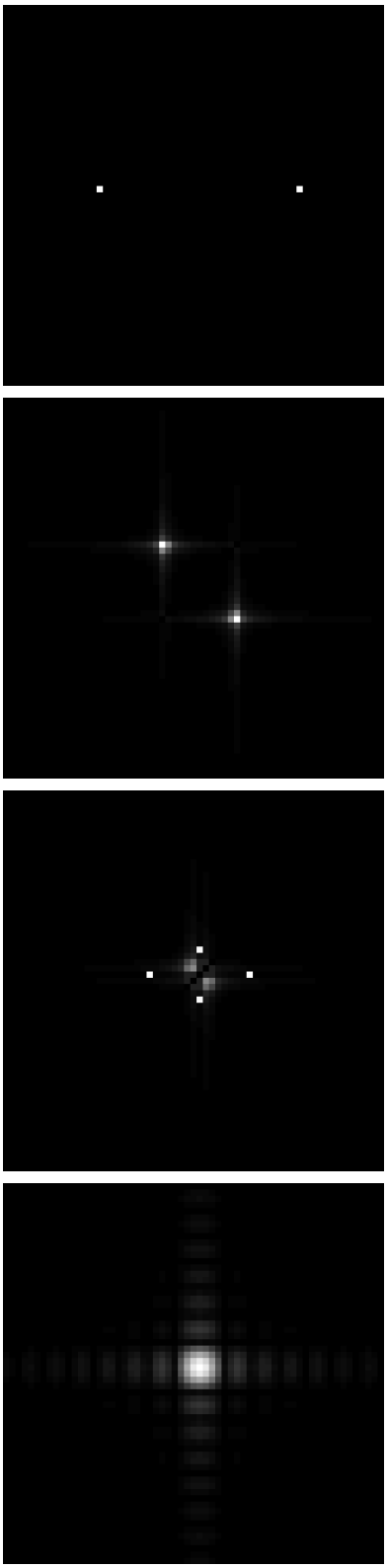
# 18 Protokol ARP (otázka studijního zaměření – 3 body)

Detailně vysvětlete fungování protokolu ARP (Address Resolution Protocol):

1. K řešení jakého problému a v jakém kontextu protokol ARP používáme?
2. Jak vypadá struktura ARP zprávy? (Nejde nám o přesnou strukturu, ale o seznam a význam jednotlivých hlavních položek.)
3. Jakým způsobem a jakým uzlům je ARP dotaz rozeslán? Jak na něj jednotlivé uzly reagují?
4. Co je ARP Cache a jak a proč se používá? Jak se liší práce se statickými a dynamickými záznamy?



Obrázek 1: Originální obraz



Obrázek 2: DFT zoom