Vysoká škola báňská - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

Diskrétní matematika

Semestrální projekt – zadání 9

Příklad	Poznámky
1	
2	

Jméno: Phat Tran Dai
Osobní číslo: TRA0163

Datum: 30. 11. 2024

Abstrakt

Tato práce se zaměřuje na dvě matematické úlohy z oblasti kombinatoriky a teorie grafů. První část se věnuje analýze netranzitivních vlastností čtyřstěnných kostek a výpočtu pravděpodobností jejich vzájemných vítězství. Navíc zkoumá celkový počet možných konfigurací čísel na kostkách a při daných pravidlech maximalizuje pravděpodobnotí jejich vztahů. Druhá část se zabývá důkazem existence jednoznačného faktoru ve stromech se sudým počtem vrcholů, kde všechny vrcholy faktoru mají lichý stupeň.

Obsah

Úvod

V této práci se zaměřuji na dvě konkrétní úlohy. První úloha pojednává o na kombinatorických vlastnostech netranzitivních kostek, které vykazují překvapivé neintuitivní pravděpodobnostní vztahy. Druhá úloha pochází z oblasti teorie grafů. Zabývá se faktory stromů se sudým počtem vrcholů a hledáním jednoznačného faktoru s lichými stupni vrcholů.

Kombinatorická část práce se zabývá výpočty pravděpodobností mezi kostkami, rozmístěními čísel na kostkách a maximalizací pravděpodobnostní hranice při splnění daných podmínek.

V části teorie grafů se pak zabývám důkazem existence faktoru s lichými stupněmi ve stromech se sudým počtem vrchlů. Dokážu také jednoznačnost tohoto faktoru.

Kombinatorika

1.1. Formulace a popis úloh

Nechť máme tři čtyřstenné kostky A, B a C, jejíchž čísla na stěnnách jsou definovaná množinami takto:

$$A = \{1, 4, 4, 4\}$$
$$B = \{2, 2, 5, 5\}$$
$$C = \{3, 3, 3, 6\}$$

Zvolme si dvě kostky X, Y a hoďme s nimi. Když jsou kostky vrženy současně, řekneme, že kostka X je lepší než kostka Y, pokud pravděpodobnost, že hodnota na kostce X bude vyšší než hodnota na kostce Y, je větší než 50%. Tuto skutečnost zapíšeme jako $X > Y$. Pravděpodobnost výhry kostky X nad kostkou Y označíme potom jako $P(X > Y)$.

Úlohy jsou následující:

1. **Prokázání netranzitivity**

   První úkolem je ukázat, že vztahy mezi kostkami nejsou tranzitivní, to znamená, že vztahy mezi kostkami jsou tzv. cyklické[1] Tvrdíme totiž, že platí $B > A$, $C > B$ a současně $A > C$. To znamená, že žádná kostka není „nejlepší" ve všech případech.

   Pro každou dvojici kostek vypočítáme pravděpodobnost vítězství jedné kostky nad druhou, konkrétně $P(B > A)$, $P(C > B)$ a $P(A > C)$, a ověříme, že všechny tyto pravděpodobnosti jsou větší než $\frac{1}{2}$.

2. **Kombinatorická analýza možných konfigurací**

   V druhém úkolu máme stanovit celkový počet možných konfigurací čísel tří kostek. Čísla na stěny kostek vybíráme z množiny $[1, 6]$, přičemž se čísla mohou opakovat. Navíc jsou kostky jsou rozlišitelné (např. barvou).

3. **Maximalizace pravděpodobností**

   Poslední úloha spočívá v nalezení největší hodnoty parametru p při volné konfiguraci čísel na kostkách (dle druhé úlohy), přičemž parametr p musí splňovat:

$$P(B > A) \geq p$$
$$P(C > B) \geq p$$
$$P(A > C) > p$$

   To vyžaduje navržení algoritmu, který systematicky prověří všechny možné konfigurace čísel na kostkách, vypočítá odpovídající pravděpodobnosti a maximalizuje p.

[1] Wikipedia, *Intransitivity*: https://en.wikipedia.org/wiki/Intransitivity

1.2. Netranzitivita kostek

Cílem této části je analyzovat vlastnosti tří čtyřstěnných kostek A, B a C a ukázat, že vykazují netranzitivní chování. To znamená, že pravděpodobnosti výhry při „souboji" mezi jednotlivými kostkami splňují vztahy:

$$P(B > A) > 0.5$$
$$P(C > B) > 0.5$$
$$P(A > C) > 0.5$$

Pro každý pár kostek X a Y definujeme pravděpodobnost $P(X > Y)$ jako pravděpodobnost, že při hodu kostkami X a Y padne na kostce X vyšší číslo než na kostce Y.

Každá kostka má čtyři stěny, takže celkový počet možných kombinací výsledků při „souboji" dvou kostek je $4 \cdot 4 = 16$, což odpovídá mohutnosti množiny kartezského součinu $X \times Y$ kostek X a Y.

Tato pravděpodobnost se vypočítá jako podíl počtu případů, kdy číslo na kostce X je větší než číslo na kostce Y a celkového počtu možných kombinací výsledků.

$$P(X > Y) = \frac{\Big|\{(x, y) \in X \times Y : x > y\}\Big|}{|X \times Y|}$$

V této části postupně určíme pravděpodobnosti $P(B > A)$, $P(C > B)$ a $P(A > C)$.

1.2.1. Výpočet $P(B > A)$

První varianta řešení
Pravděpodobnostní prostor Ω je kartezský součin čísel na kostkách B a A:

$$B = \{2, 2, 5, 5\} \quad A = \{1, 4, 4, 4\} \quad \Omega = \Big\{(b, a) : b \in B, a \in A\Big\} \Leftrightarrow B \times A$$

$$\begin{aligned}\Omega = \Big\{&(2,1),(2,1),(5,1),(5,1),\\ &(2,4),(2,4),(5,4),(5,4),\\ &(2,4),(2,4),(5,4),(5,4),\\ &(2,4),(2,4),(5,4),(5,4)\Big\}\end{aligned}$$

Velikost pravděpodobnostního prostoru je $|\Omega| = 16$. Z rozepsané Ω vidíme, že počet případů, kdy kostka B vyhraje nad A je vyšší (10) než počet, kdy prohraje (6). Pravděpodobnost vypočteme jako:

$$P(B > A) = \frac{2 + 4 \cdot 2}{|\Omega|} = \frac{10}{16} = \underline{\underline{0.625}}$$

Vidíme, že pravděpodobnost výhry kostky B nad kostkou A je vyšší než 50%. To znamená, že kostka B je lepší než A. $\square$

Druhá varianta řešení
Pravděpodobnostní prostorem je stále $\Omega = B \times A$. Využijeme toho, že se kostky skládají ze dvou různých čísel. Pokud hodnota kostky A je 1, tak kostka B vyhraje vždy a to bez ohledu na na její hozenou hodnotu. Pokud padla na kostce A hodnota 4, tak B vyhraje pouze tehdy, když byla vržena hodnota 5. To zapišeme a vypočítáme následně.

$$
\begin{aligned}
P(B > A) &= \frac{P(A=1) \cdot P(B > A = 1) \cdot |\Omega| + P(A=4) \cdot P(B > A = 4) \cdot |\Omega|}{|\Omega|} \\
&= |\Omega| \; \frac{P(A=1) \cdot P(B > 1) + P(A=4) \cdot P(B > 4)}{|\Omega|} \\
&= P(A=1) \cdot P(B > 1) + P(A=4) \cdot P(B > 4) \qquad (1.1) \\
&= P(A=1) \cdot P(B) + P(A=4) \cdot P(B=5) \\
&= \frac{1}{4} \cdot 1 + \frac{3}{4} \cdot \frac{1}{2} \\
&= \frac{2}{8} + \frac{3}{8} = \frac{5}{8} = \underline{\underline{0.625}}
\end{aligned}
$$

$P(B > A) = \frac{5}{8} > \frac{1}{2}$, proto kostka B je lepší než A. $\square$

Zbývalé pravděpodobnosti $P(C > B)$ a $P(A > C)$ bychom mohli vypočítat obdobně jako $P(B > A)$. Lze to ale udělat lépe? Všiměte si Rovnice (1.1). Její tvar můžeme využitím zákonu celkové pravděpodobnosti[2] zobecnit.

1.2.2. Výpočet pomocí zákonu celkové pravděpodobnosti

Zákon celkové pravděpodobnosti uvádí, že mame-li událost E, která závisí na známých podmínkách, tak její pravděpodobnost lze vyjádřit jako:

$$P(E) = \sum_{i=0}^{n} P(C_i) \cdot P(E|C_i)$$

kde C_i jsou disjunktní podmínky pokrývající celý pravdepodobnostní prostor Ω, tedy:

$$\bigcup_{i=0}^{n} C_i = \Omega \quad \text{a} \quad C_i \cap C_j \quad \text{pro} \quad i \neq j$$

[2]Wikipedia, *Law of total probability*: https://en.wikipedia.org/wiki/Law_of_total_probability

Událostmi jsou v našem případě $X > Y$ (kostka X vyhraje nad kostkou Y). Disjunktní podmínky C_i odpovídají výsledkům hodů kostky Y, která může nabývat hodnot $y_0, y_1, ..., y_n$. Podmínky $Y = y_i$ jsou disjunktní, protože platí:

$$\bigcup_{i=0}^{n}(Y = y_i) = \Omega_{XY} \quad \text{a} \quad (Y = y_i) \cap \left(Y = y_j\right) = \emptyset \;\; \text{pro} \;\; i \neq j$$

Padne-li na kostce Y např. 1 nemůže zároveň padnout 3 nebo 5 apod. Obecný vzorec pro výpočet pravděpodobnosti $P(X > Y)$ je:

$$P(X > Y) = \sum_{y \in Y} P(Y = y) \cdot P(X > y)$$

1.2.3. Aplikace vzorce na $P(C > B)$ a $P(A > C)$

Pro připomenutí, množiny A, B a C jsou:

$$A = \{1, 4, 4, 4\} \quad B = \{2, 2, 5, 5\} \quad C = \{3, 3, 3, 6\}$$

Výpočty pravděpodobností pomocí odvozeného vzorce:

$$\begin{aligned}
P(C > B) &= \sum_{b \in B} P(B = b) \cdot P(C > b) \\
&= P(B = 2) \cdot P(C > 2) + P(C = 5) \cdot P(C > 5) \\
&= \frac{1}{2} \cdot \frac{4}{4} + \frac{1}{2} \cdot \frac{1}{4} \\
&= \frac{4}{8} + \frac{1}{8} = \frac{5}{8} = \underline{\underline{0.625}}
\end{aligned}$$

$$\begin{aligned}
P(A > C) &= \sum_{c \in C} P(C = c) \cdot P(A > c) \\
&= P(C = 3) \cdot P(A > 3) + P(C = 6) \cdot P(A > 6) \\
&= \frac{3}{4} \cdot \frac{3}{4} + \frac{1}{4} \cdot \frac{0}{4} = \frac{9}{16} = \underline{\underline{0.5625}}
\end{aligned}$$

Dokázali jsme, že $B > A$, $C > B$ a $A > C$. $\square$

1.3. Různá rozmístění čísel na kostkách

Kostky A, B a C jsou rozlišitelné, například mají odlišné barvy. Jaký je počet různých konfigurací čísel, pokud vybíráme čísla z množiny z množiny $[1, 6]$ s možností opakovaní.

Počet zpusobů, jak vybrat čtyři čísla z sešti (s možností opakování), je:

$$C^*(6,4) = \binom{9}{4} = 126$$

Proto možných konfigurací čísel na třech kostkách je:

$$\Big[C^*(6,4)\Big]^3 = 126^3 = \underline{\underline{2000376}} \tag{1.2}$$

Jiná úvaha (kuličky a přehrádky)
Počet různých konfigurací pro jednu kostku lze také vyjádřit jako:

$$\underbrace{\binom{6}{4}\binom{3}{3}}_{a} + \underbrace{\binom{6}{3}\binom{3}{2}}_{b} + \underbrace{\binom{6}{2}\binom{3}{1}}_{c} + \underbrace{\binom{6}{1}\binom{3}{0}}_{d} = 126 \tag{1.3}$$

Vysvětlení členů:

a) *Vybereme čtyři různá čísla z šesti možných.*
Umístíme tři oddělovače mezi čtyřmi stěnami kostky (kuličky). Vzniklé čtyři přehrádky naplníme těmito vybranými čísly.

b) *Vybereme tři čísla z šesti možných.*
Zvolíme dvě pozice ze tří, kam umístit dva oddělovače mezi čtyřmi stěnami. Vzniklé tři přehrádky naplníme těmito čísly.

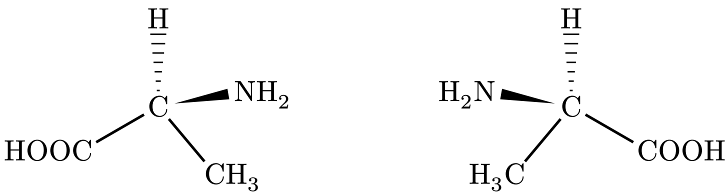
c) *Vybereme dvě čísla z šesti možných.*
Zvolíme jednu ze tří pozic, kam umístíme jeden oddělovač. Vzniklé dvě přehrádky naplníme těmito čísly.

d) *Vybereme jedno číslo z šesti možných.*
Máme pouze jednu přehrádku, kterou naplníme tímto číslem.

Enantiomorfy kostek

Výpočet konfigurací podle Rovnice (1.3) platí pouze za předpokladu, že nebereme v úvahu různá rozmístění pevně zvolených čísel na kostce. Pokud máme například 4 různá čísla, lze je na čtyřstěnnou kostku rozmístit dvěma různými způsoby, neboť každá konfigurace má svůj zrcadlový obraz, který s ní není totožný. Tento obraz se nazývá chirální enantiomorf[3].

Podobný jev lze pozorovat v chemii u enantiomérů, což představuje analogii k našim kostkám.



Obrázek 1.1: L-alanin a D-alanin (enantioméry)

Pokud bychom chtěli být zcela přesní, je třeba výpočet podle Rovnice (1.3) upravit následujícím způsobem:

$$\underbrace{2 \cdot \binom{6}{4}}_{\text{korekce}} + \binom{6}{3}\binom{3}{1} + \binom{6}{2}\binom{3}{1} + \binom{6}{1}$$

Tento výpočet se dále rozepíše jako:

$$\begin{aligned} &2 \cdot 15 + 20 \cdot 3 + 15 \cdot 3 + 6 \\ &= 30 + 60 + 45 + 6 = \underline{141} \end{aligned}$$

Pro tři kostky tedy bude možných konfigurací:

$$(141)^3 = \underline{\underline{2803221}}$$

[3]Wikipedia, *Chiralita*: https://cs.wikipedia.org/wiki/Chiralita

1.4. Sestavení algoritmu

Cílem je maximalizovat p tak, aby současně platily:

$$\begin{cases} P(B > A) & \geq & p \\ P(C > B) & \geq & p \\ P(A > C) & > & p \end{cases}$$

pričemž hodnoty pravděpodobnosti $P(X > Y)$ manipulujeme volným výběrem čísel stěn kostek z množiny čísel $[1, 6]$ (s možností opakování).

Algoritmus jsem se rozhodl implementovat v programovacím jazyce Rust[4]. Celý zdrojový kód naleznete zde[5].

Základní idea algoritmu

1. Nejprve se vygenerují všechny číselné kombinace čísel kostek. K vygenerovaní kombinací čísel kostek jsem si napsal pomocnou funkci `fn generate_dice`, ve které volám funkci `fn combinations_with_replacement` z knihovny `itertools`.

```rust
fn generate_dice(sides: usize) -> Vec<Vec<usize>> {
    (1..=6).combinations_with_replacement(sides).collect()
}
```

Seznam 1.1: Funkce `fn generate_dice`

2. Vypočítání probability $P(X > Y)$ zajišťuje následující funkce:

```rust
fn pairwise_probability(x: &[usize], y: &[usize]) -> f64 {
    let mut wins = 0;              // Count the number of pairs where x > y
    for &xi in x.iter() {          // Loop through each element in x and y
        for &yi in y.iter() {
            if xi > yi {
                wins += 1;         // Increment wins if xi is greater than yi
            }
        }
    }
    let omega = (x.len() * y.len()) as f64; // Size of the probability space
    wins as f64 / omega                     // Return computed probability
}
```

Seznam 1.2: Funkce `fn pairwise_probability`

3. Použitím tří vnořených cyklů se prověří, jestli pro stávající p platí, že $P(B > A) \geq p$, $P(C > B) \geq p$ a $P(A > C) > p$ pro všechny možné kombinace kostek A, B a C.

[4]Programovací jazyk Rust: https://www.rust-lang.org/

[5]GitHub repozitář se zdrojovým kódem: https://github.com/phatt-23/Projekt-9-DIM/blob/master/program/src/main.rs

```rust
for a in &dice {          // Check all combinations of A, B, C
    for b in &dice {
        for c in &dice {
            if pairwise_probability(b, a) >= p    // P(B > A) >= p
              && pairwise_probability(c, b) >= p  // P(C > B) >= p
              && pairwise_probability(a, c) > p   // P(A > C) >  p
            { /* ... do something */ }
        }
    }
}
```

Seznam 1.3: Kód na prověřování podmínek

Implementace algoritmu

Celý algoritmus (naivní varianta) je zde:

```rust
fn find_max_p_naive(side_count: usize) -> f64 {
    println!("Finding maximal p:");
    let mut max_p = 0.0;                        // Holds the maximum p
    let dice = generate_dice(side_count); // Create all the dice combinations
    let omega_size = side_count.pow(2);
    let increment = 1.0 / omega_size as f64; // p grows by this step

    for step in 0..=omega_size {                // Iterate from 0 to |Omega|
        let p = step as f64 * increment;  // Incrementing by 1/|Omega|
        let mut valid = false;                  // No valid configs have yet been found
        println!("Testing for p = {}:", p);

        'outer:                                 // Tag to jump to from within the loop
        for a in &dice {                        // Test out every single combination
            for b in &dice {
                for c in &dice {
                    // Get the probabilities of P(B > A), P(C > B), P(A > C)
                    let ba = pairwise_probability(b, a);
                    let cb = pairwise_probability(c, b);
                    let ac = pairwise_probability(a, c);
                    if ba >= p && cb >= p && ac >= p { // Check them against p
                        valid = true; // This p has a valid configuration
                        println!("A={:?}, B={:?}, C={:?}", a, b, c);
                        break 'outer; // Jump out of loops (to the 'outer tag)
                    }
                }
            }
        }
        // If for current p config doesnt exist, then return the last valid p
        if !valid {
            println!("No valid configurations!");
            return max_p;
        }

        max_p = p; // If configuration exists assign to max_p
    }
    max_p           // By default return max_p found
}
```

Seznam 1.4: Naivní varianta funkce `fn find_max_p`

Tento algoritmus je velmi neefektivní, protože opakovaně počítá pravděpodobnosti mezi týmiž kostkami. V důsledku je alogitmus značně pomalý.

Je zřejmé, že by algoritmu přispělo předpočítat pravděpodobnosti všech kombinací dvou kostek. Vypočtené hodnoty uložíme pole. Konkrétně použijeme dynamické pole v Rustu zvaný jako `Vec<T>`.

Pro naplnění tohoto pole předpočtenými hodnotami pravděpodobností, jsem si napsal následující funkci, která vrací matici pravděpobností kombinací kostek X a Y (obětuje paměť za zaručení rychlejšího vyhledání hodnoty pravděpodobnosti):

```rust
fn precompute_probabilities_vec(dice: &Vec<Vec<usize>>) -> Vec<Vec<f64>> {
    let size = dice.len();
    // Matrix of X and Y
    let mut cache: Vec<Vec<f64>> = vec![vec![0.0; size]; size];

    for (i, x) in dice.iter().enumerate() {
        for (j, y) in dice.iter().enumerate() {
            // Insert P(X>Y) at [i,j]
            cache[i][j] = pairwise_probability(x, y);
        }
    }

    cache // Return the matrix of computed probabilities of X and Y
}
```

Seznam 1.5: Funkce `fn precompute_probabilities_vec`

a využil ji v upravené funkci pro hledaní maximální hodnoty p:

```rust
fn find_max_p_caching_vec(side_count: usize) -> f64 {
    // ... (identical with the previous)
    let cache = precompute_probabilities_vec(&dice);  // <-- precomputing

    for step in 1..=16 {
        let p = step as f64 * increment;
        let mut valid = false;
        'outer:
        for (i, a) in dice.iter().enumerate() {
            for (j, b) in dice.iter().enumerate() {
                if cache[j][i] < p { continue; }  // skipping innermost loop
                                                  // if P(A > B) doesnt hold
                for (k, c) in dice.iter().enumerate() {
                    if cache[k][j] >= p && cache[i][k] > p {
                        println!("A = {:?}, B = {:?}, C = {:?}", a, b, c);
                        valid = true;
                        break 'outer;
                    }
                }
            }
        }

        if !valid {
          println!("No valid configurations!");
          return max_p;
        }

        max_p = p;
    }
    // ... (identical with the previous)
}
```

Seznam 1.6: Upravená funkce `fn find_max_p`

Posledně jsem droubnou úpravou cyklů algoritmus paralelizoval:

```rust
fn find_max_p_parallel(side_count: usize) -> f64 {
    // ... (identical with the previous)

    for step in 1..=16 {
        let p = step as f64 * increment;
        println!("Testing for p = {}:", p);

        // Iterations done in paralel
        let valid = dice.par_iter().enumerate().any(|(i, _)| {
            dice.par_iter().enumerate().any(|(j, _)| {
                if cache[j][i] < p { return false; }
                dice.par_iter().enumerate().any(|(k, _)| {
                    cache[k][j] >= p && cache[i][k] > p
                })
            })
        });

        if !valid {
            println!("No valid configurations!");
            return max_p;
        }

        println!("Config found (no printout available)!");
        max_p = p;
    }

    // ... (identical with the previous)
}
```

Seznam 1.7: Paralení varianta funkce `fn find_max_p`

Vzhledem k tomu, že se zde zabýváme čtyřstěnnými kostkami, paralelizace přinesla pouze mírné časové zlepšení.

Výpis po zpuštění programu

```
Maximal p = 0.5625
Valid configurations for p = 0.5625 are:
[0]   A = [2, 2, 5, 5]  B = [3, 3, 3, 6]  C = [1, 4, 4, 4]
[1]   A = [2, 2, 5, 6]  B = [3, 3, 3, 6]  C = [1, 4, 4, 4]
[2]   A = [3, 3, 3, 6]  B = [1, 4, 4, 4]  C = [1, 2, 5, 5]
[3]   A = [3, 3, 3, 6]  B = [1, 4, 4, 4]  C = [2, 2, 5, 5]
```

Seznam 1.8: Standardní výstup v konzoli

Z výpisu algoritmu jsem zpozoroval, že maximalní hodnota, kterou p může nabývat, je $\frac{9}{16}$ neboli 0.5625. Také jsem zjístil o jaké konfigurace kostek, které splňuji dané podmínky, se přesně jedná. Dvě z nich (s indexy 0 a 3) dokonce odpovídají konfiguraci ve slovním zadání, neberu-li v potaz označení kostek.

Teorie grafů

Mějme strom T se sudým počtem vrcholů (je sudého řádu). Cílem je ukázat, že pro T existuje faktor F, kde všechny vrcholy grafu F jsou lichého stupně (budeme nazývat jako lichý faktor).

2.1. Důkaz existence faktoru

Existenci faktoru stromu dokážeme indukci – konkrétně jeho rekurzivní konstrukcí.

Nechť $T = (V, E)$ je strom, kde: $|V(T)| \equiv 0 \pmod 2$ a $|V(T)| \geq 2$.

Základní případ

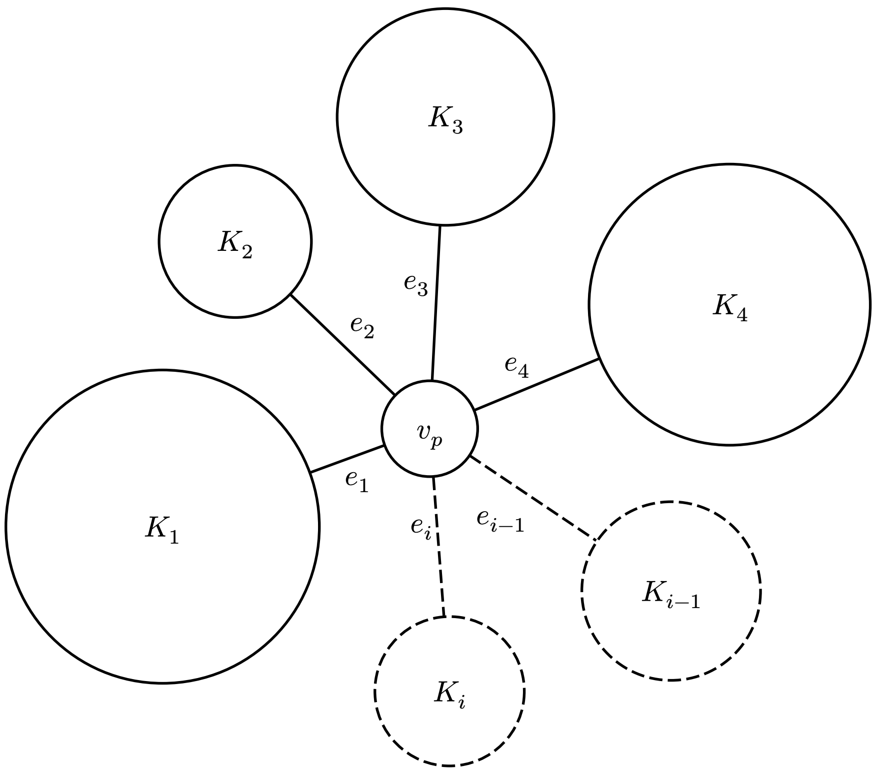
Pro strom T se dvěma vrcholy, $|V(T)| = 2$, je zřejmé, že jeho lichý faktor je právě T.

Indukční krok

Uvažujme pro $|V(T)| > 2$. Vybereme libovolný vrchol $v_p \in V(T)$, kde $\deg_T(v_p) > 1$ (tj. nebereme listy), a výjmeme ho z grafu. Výsledkem je graf $T - \{v_p\}$, jehož komponenty, které jsou rovněž stromy, označíme jako:

$$K_1, K_2, ..., K_{\deg_T(v_p)}$$

Hranu, která spojuje vrchol v_p s komponentou K_i, označíme jako e_i.



Obrázek 2.2: Strom T s vyznačeným vrcholem v_p

Rozdělení komponent

Komponenty si rozdělíme do dvou množin S a L tak, aby S obsahovala komponenty sudého řádu, zatímco L obsahovala komponenty lichého řádu:

$$S = \left\{K_i : |V(K_i)| \equiv 0\right\} \quad L = \left\{K_i : |V(K_i)| \equiv 1\right\} \pmod 2$$

Analýza počtu vrcholů

Počet vrcholů grafu $T - \{v_p\}$ je lichý:

$$\left|V\left(T - \{v_p\}\right)\right| \equiv 1 \pmod 2$$

Dále platí:

$$V\left(T - \{v_p\}\right) = \left(\bigoplus_{K \in S} V(K)\right) \oplus \left(\bigoplus_{K \in L} V(K)\right)$$

$$|V\left(T - \{v_p\}\right)| = \sum_{K \in S} |V(K)| + \sum_{K \in L} |V(K)| \equiv 1 \pmod 2$$

kde znak $\oplus$ představuje operátor symetrického rozdílu[6] dvou množin.

Liché číslo lze získat pouze součtem sudého a lichého čísla:

$$\underbrace{2t}_{\text{sudé}} + \underbrace{(2k+1)}_{\text{liché}} \equiv 1 \pmod 2, \quad \text{pro } k, t \in \mathbb{Z}$$

Jedna z množin tedy musí mít lichý součet počtu vrcholů svých komponent, přičemž pro množinu S platí:

$$\sum_{K \in S} |V(K)| \equiv 0 \pmod 2$$

protože suma počtů vrcholů komponent v S je vždy sudá. Z čehož plyne:

$$\sum_{K \in L} |V(K)| \equiv 1 \pmod 2$$

tedy, že suma vrcholů všech komponent v L je lichá.

[6] Wikipedia, *Symetrický rozdíl*: https://en.wikipedia.org/wiki/Symmetric_difference

Počet komponent v L

Součet vrcholů komponent v S je vždy sudý. Naopak součet vrcholů komponent v L je vždy lichý. To znamená, že množina L obsahuje lichý počet komponent:

$$|L| \equiv 1 \pmod 2$$

neboť jediný způsob, kterým získáme liché číslo součtem lichých čísel (počet vrcholů komponenty v L), je, když máme lichý počet (mohutnost množiny L) lichých čísel.

$$\forall K \in L: \quad |V(K)| = 2k + 1 \quad \text{pro } k \in \mathbb{N}$$
$$p = |L|$$

$$\begin{aligned} p(2k+1) &\equiv 1 \pmod 2 \\ 2pk + p &\equiv 1 \pmod 2 \\ p &\equiv 1 \pmod 2 \end{aligned}$$

Konstrukce faktoru

Pro komponentu $K \in L$ neexistuje faktor $F = (V(K), E_F)$, kde $E_F \subseteq E(K)$ a každý vrchol $v \in V(F)$ by byl lichého stupně. To vyplývá z principu sudosti:

$$\sum_{v \in K} \deg_K(v) \equiv 0 \pmod 2$$

Nemůže tedy existovat lichý faktor stromu lichého řádu, protože by to porušovalo tento princip.

Abychom zajistili sudost, přidáme do každého $K \in L$ zpět výjmutý vrchol v_p (včetně hrany e_i), čímž získáme stromy $K + \{v_p\}$ sudého řádu. Na tento strom aplikujeme indukční krok. Vrchol v_p bude mít stupeň roven počtu komponent v L:

$$\deg(v_p) = |L| \equiv 1 \pmod 2$$

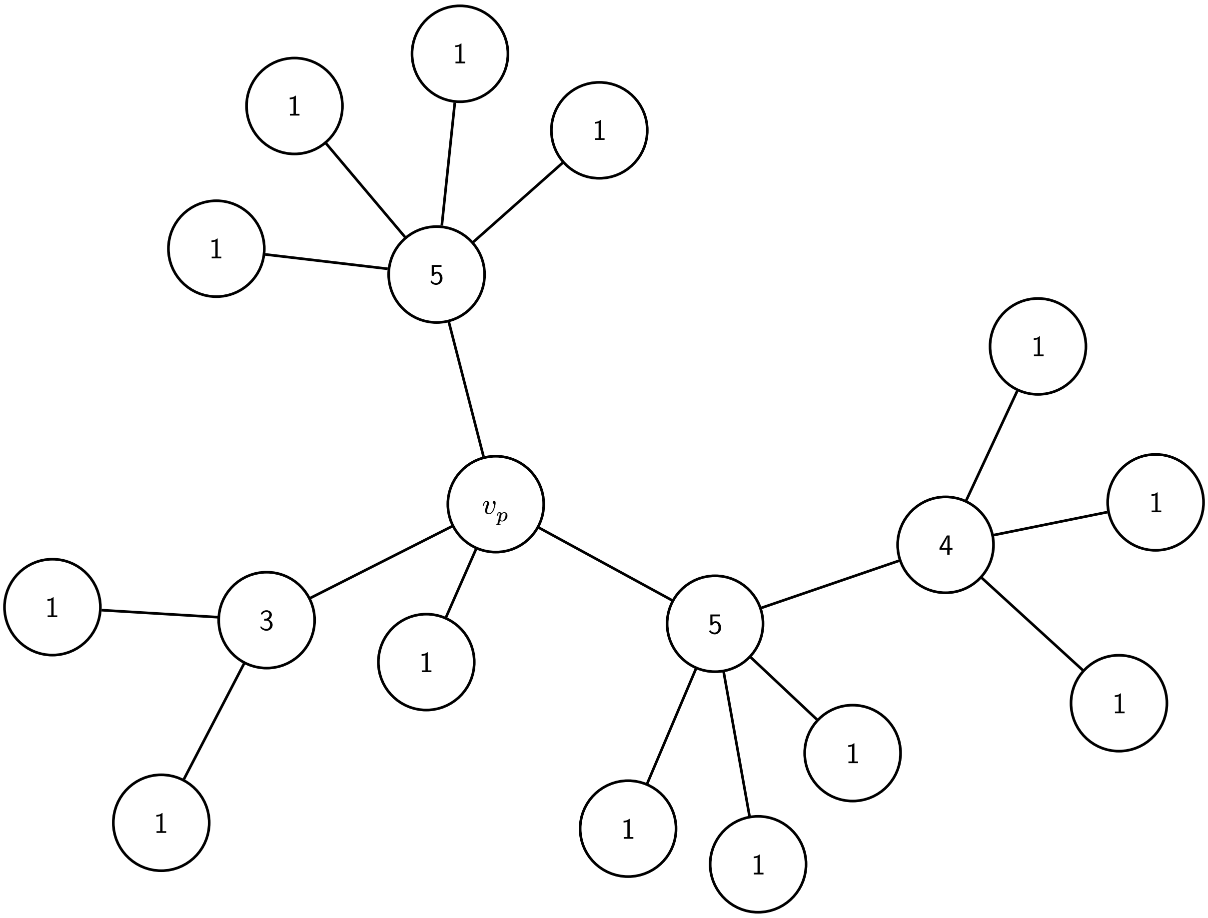
Komponentám v S nepřidáváme vrchol v_p, neboť už jsou sudého řádu. I na ně aplikujeme indukční krok.

Na konci indukce (rekurze) bude každý vrchol $v \in V(T)$ lichého stupně, protože:

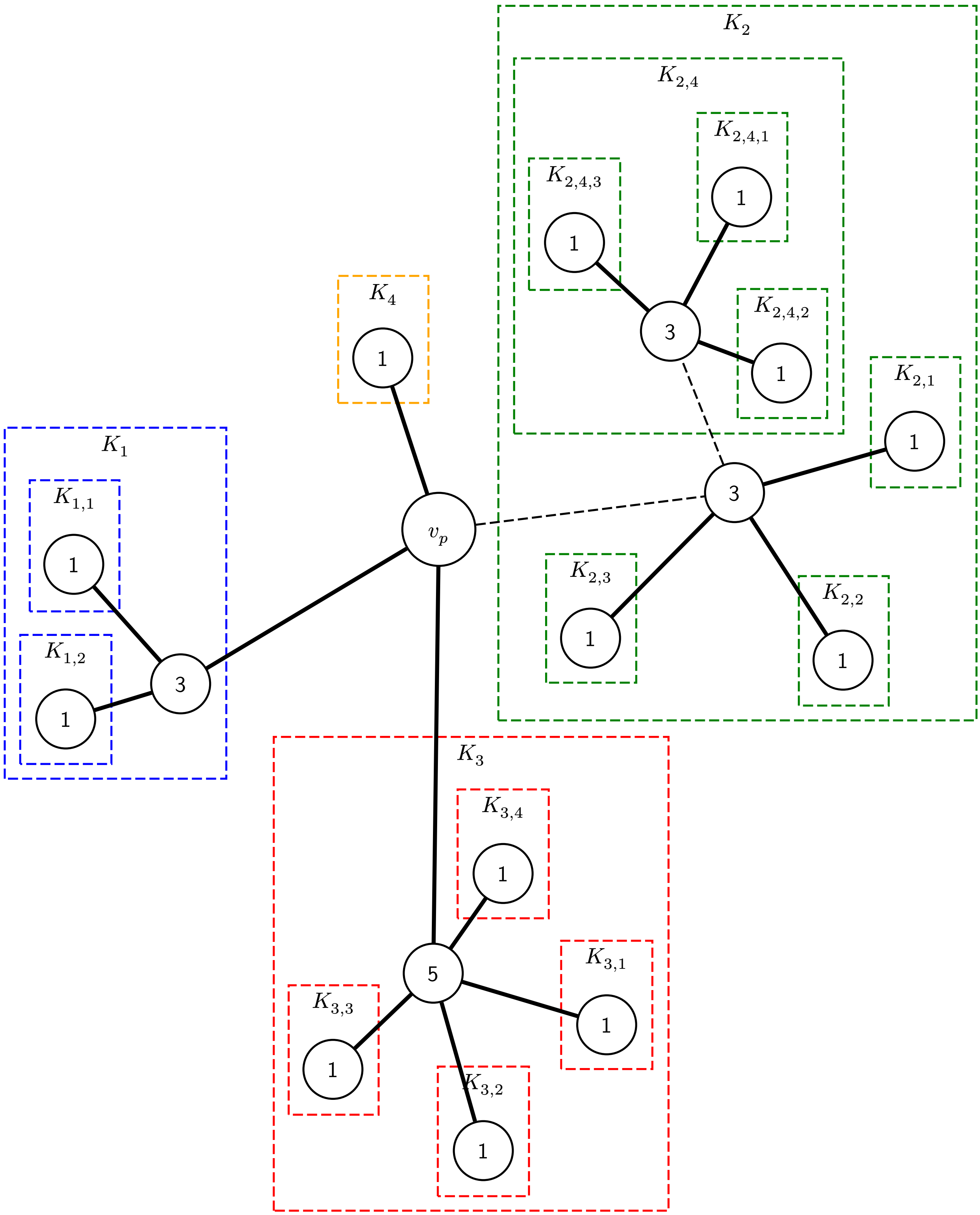
$$\deg(v) = |L|$$

přičemž mohutnost L je v každém kroku lichá. Tím je existence faktoru prokázána. $\square$

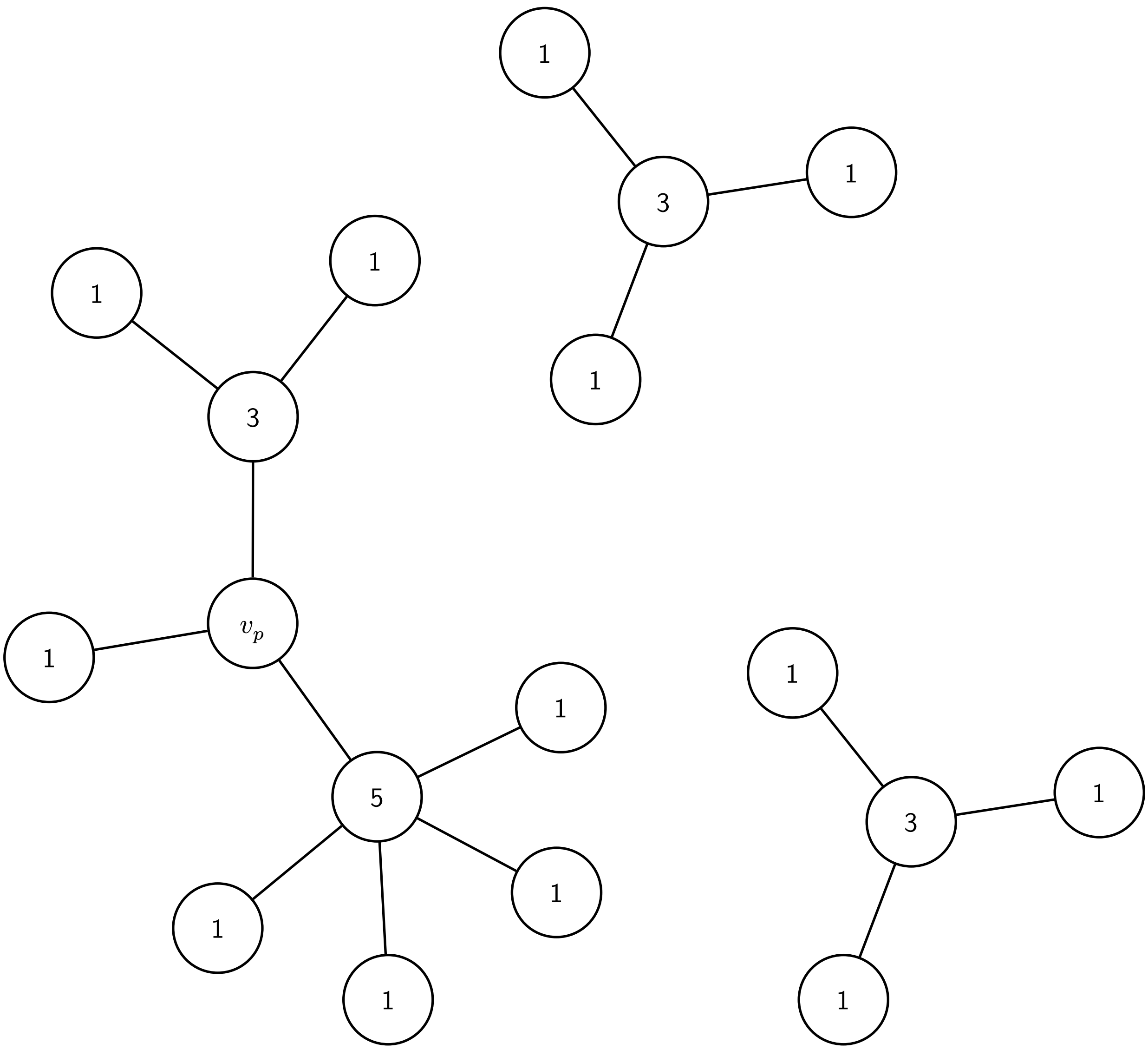
Ukázka algoritmu



Obrázek 2.3: Strom T s vyznačenými stupněmi vrcholů a vybraným vrcholem v_p



Obrázek 2.4: Rekurzivní konstrukce lichého faktoru stromu T



Obrázek 2.5: Nalezený lichý faktor stromu T

2.2. Důkaz jednoznačnosti faktoru F

Existenci faktoru F jsme si odůvodnili. Nyní si dokážme, že takových faktorů F grafu G, kde jsou všechny vrcholy lichého stupně, je právě jeden jediný.

Důkaz sporem

Předpokladejme, že existují dva liché faktory grafu G. Mějme faktory F_A, F_B, u kterých tvrdíme, že jsou od sebe odlišné. Tedy musí platit, že mají alespoň jednu hranu, která náleží pouze jim a ne druhému.

$$\exists e \in E(F_A), e \notin E(F_B)$$

Vytvoříme nový graf G. Ten obsahuje všechny vrcholy stromu T a obsahuje symetrický rozdíl hran faktorů F_A a F_B. Tedy obsahuje ty hrany, které se objevují právě v jednom z faktorů, nikoliv v obou.

$$G = \Big(V(T), E(F_A) \oplus E(F_B)\Big)$$

Pokud je graf G nulovým grafem:

$$G = \Big(V(T), \emptyset\Big)$$

tak $E(F_A) = E(F_B)$, z čehož plyne, že $F_A = F_B$ (popisují tentýž graf).

Sudost a lichost stupní vrcholů $v \in V(G)$

Sudost a lichost stupně vrcholů grafu G závisí na tom, jaké hrany z faktorů F_A a F_B budou zahrnuty do vysledného grafu G. Množinu hran faktoru F_X, které jsou incidentní s vrcholem v zapíšeme:

$$E_v(F_X)$$

Při symetrickém rozdílu hran faktorů F_A a F_B, mohou nastat tyto případy:

a) Faktory F_A a F_B nesdílejí jedinou hranu:

$$E(F_A) \cap E(F_B) = \emptyset$$

Každý vrchol v grafu G je sudého stupně, neboť:

$$|E_v(F_A)| = |E_v(F_B)| \equiv 1 \pmod 2$$
$$\deg_G(v) = |E_v(F_B)| + |E_v(F_A)| \equiv 0 \pmod 2$$

b) Pokud se všechny hrany faktorů F_A a F_B shodují:

$$E(F_A) = E(F_B) \quad \Leftrightarrow \quad E(F_A) \oplus E(F_B) = \emptyset$$

tak graf G je nulový a pro $\forall v \in V(G)$ platí $\deg_G(v) = 0$, jsou všechny sudého stupně.

c) Pokud se některé hrany náchazejí v obou faktorech F_A a F_B:

$$\exists e \in E(F_A), e \in E(F_B) \quad \Leftrightarrow \quad E(F_A) \cap E(F_B) \neq \emptyset$$

tak jsou všechny vrcholy grafu G také sudého stupně. což lze dokázat. Nejdříve si však zjednodušme syntaxi touto substitucí:

$$A = E_v(F_A)$$
$$B = E_v(F_B)$$

Víme, že:

$$|A| = |B| \equiv 1 \pmod 2$$

Pokud je počet hran ve sjednocení množin A a B sudý, tak stupeň v je sudý:

$$|A \cap B| \equiv 0 \quad \Rightarrow \quad |A \setminus B| = |B \setminus A| \equiv 1 \pmod 2$$
$$\deg_G(v) = |A \setminus B| + |B \setminus A| \equiv 0 \pmod 2$$

Pokud je počet hran ve sjednocení množin A a B lichý, tak stupeň v je taktéž sudý:

$$|A \cap B| \equiv 1 \quad \Rightarrow \quad |A \setminus B| = |B \setminus A| \equiv 0 \pmod 2$$
$$\deg_G(v) = |A \setminus B| + |B \setminus A| \equiv 0 \pmod 2$$

Závěr

Nyní víme, že vrcholy v grafu G budou vždy sudého stupně:

$$\forall v \in V(G), \deg_G(v) = 2k \quad \text{pro } k \in \mathbb{N}_0$$

Pokud $k = 0$ tak $F_A = F_B$. Uvažujeme-li, že $k > 0$, dostaneme vrcholy stupně alespoň 2. Minimální graf, kde jsou všechny vrcholy alespoň druhého stupně, je graf cesty. Cesty jsou cyklické - stromy jsou jsou acyklické - je zde kontradikce. Nemůžeme z acyklického stromu získat cyklický graf.

Tedy jedinou přípustnou možností je, že fakory F_A a F_B popisují jeden a ten samý graf. Jednoznačnost faktoru F je dokázána. □