

Diskrétní matematika

Semestrální projekt – zadání 9

Příklad	Poznámky
1	
2	

Jméno: Phat Tran Dai
Osobní číslo: TRA0163

Datum: 01. 12. 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 výpočtům pravděpodobností jejich vzájemných výher. Navíc zkoumá celkový počet možných konfigurací čísel na kostkách a při daných podmínkách maximalizuje pravděpodobnosti jejich vztahů. Druhá část se zabývá důkazem existence jednoznačného faktoru ve stromech se sudým počtem vrcholů, kde všechny vrcholy faktoru mají lichý stupeň.

Obsah

<i>Úvod</i>	3
1. Kombinatorika	4
1.1. Formulace a popis úloh	4
1.2. Netranzitivita kostek	5
1.2.1. Výpočet $P(B > A)$	5
1.2.2. Výpočet pomocí zákona celkové pravděpodobnosti	6
1.2.3. Aplikace vzorce na $P(C > B)$ a $P(A > C)$	7
1.3. Různá rozmístění čísel na kostkách	8
1.4. Sestavení algoritmu	10
2. Teorie grafů	14
2.1. Důkaz existence faktoru	14
2.2. Důkaz jednoznačnosti faktoru F	20

Úvod

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

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

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

Kombinatorika

1.1. Formulace a popis úloh

Nechť máme tři čtyřstenné kostky A , B a C , jejichž 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 hodme s nimi. Když jsou kostky vrženy současně, řekneme, že kostka X je lepší než kostka Y , pokud je pravděpodobnost, že padlá hodnota na kostce X bude vyšší než hodnota na kostce Y , vyšší 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. cyklické¹. 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. Kostky jsou navíc rozlišitelné (např. barvou).

3. Maximalizace pravděpodobností

Poslední úloha spočívá v nalezení největší hodnoty parametru p při volné konfiguraci čísel na kostkách (dle úlohy 2), 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 .

¹Wikipedia, *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 výher při „soubojích“ 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 $P(X > Y)$ 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 $X \times Y$ čísel kostek X a Y . Pravděpodobnost $P(X > Y)$ se vypočítá jako podíl počtu případů, kdy padlé číslo na kostce X je větší než číslo na kostce Y , a celkového počtu možných kombinací výsledků.

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

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

Pravděpodobnostním prostorem Ω je kartezský součin čísel kostek B a A :

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

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

Z rozepsané Ω vidíme, že počet případů, kdy kostka B vyhraje nad kostkou 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 kostka A . \square

Druhá varianta řešení

Pravděpodobnostním prostorem je stále $\Omega = B \times A$. Využijeme toho, že se kostky skládají ze dvou různých čísel. Pokud padlá hodnota kostky A je 1, pak kostka B vyhraje vždy a to bez ohledu na její vrženou hodnotu. Pokud padla na kostce A hodnota 4, pak B vyhraje pouze tehdy, když bude vržena hodnota 5. To zapišeme a vypočítáme následovně:

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

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

Zbývalé pravděpodobnosti $P(C > B)$ a $P(A > C)$ bychom mohli vypočítat obdobně. Lze to ale udělat lépe? Všiměte si Rovnice (1.1). Její tvar můžeme využitím zákona celkové pravděpodobnosti² výpočet pro zbylé vztahy zobecnit.

1.2.2. Výpočet pomocí zákona celkové pravděpodobnosti

Zákon celkové pravděpodobnosti uvádí, že máme-li událost E , která závisí na známých podmínkách C_1, C_2, \dots, C_n , tak lze její pravděpodobnost vyjádřit jako:

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

kde C_i jsou disjunkttní podmínky pokrývající celý pravděpodobnostní prostor Ω , tedy:

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

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

Zkoumanými událostmi jsou v našem případě $X > Y$ (kostka X vyhraje nad kostkou Y). Disjunktivním podmínkám C_i odpovídají výsledky hodů kostky Y , která může nabývat hodnot y_0, y_1, \dots, y_n . Podmínky $Y = y_i$ jsou disjunktivní, protože platí:

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

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

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

Výpočty pravděpodobností pomocí odvozeného vzorce

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

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

Dokázali jsme, že $B > A$, $C > B$ a $A > C$. □

1.3. Různá rozmístění čísel na kostkách

Kostky A , B a C jsou rozlišitelné (například mají odlišné barvy). Zajímá nás, jaký je počet různých konfigurací čísel na kostkách, za předpokladu, že čísla vybíráme z množiny $[1, 6]$ s možností opakování.

Počet různých způsobů, jak vybrat čtyři čísla z šesti (s možností opakování), je:

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

Proto možných konfigurací čísel na třech kostkách je:

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

Jiná úvaha (kuličky a přehrádky)

Počet různých konfigurací pro jednu kostku lze také vyjádřit jako:

$$\underbrace{\binom{6}{4}\binom{3}{3}}_a + \underbrace{\binom{6}{3}\binom{3}{2}}_b + \underbrace{\binom{6}{2}\binom{3}{1}}_c + \underbrace{\binom{6}{1}\binom{3}{0}}_d = 126 \quad (1.3)$$

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 třemi čísly.

c) *Vybereme dvě čísla z šesti možných.*

Zvolíme jednu ze tří pozic, kam umístíme jeden oddělovač. Vzniklé dvě přehrádky naplníme těmito dvěma čísly.

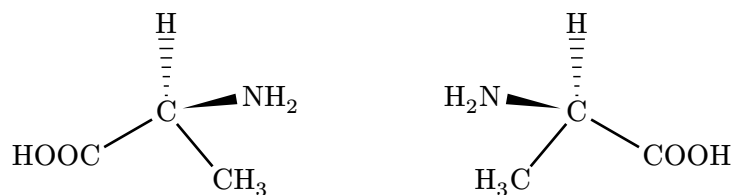
d) *Vybereme jedno číslo z šesti možných.*

Máme pouze jednu přehrádku, kterou naplníme tímto číslem.

Zrcadlové obrazy kostek

Výpočet konfigurací podle Rovnice (1.3) platí pouze za předpokladu, nebereme-li v úvahu různá rozmístění pevně zvolených čísel na kostce. Pokud máme 4 různá čísla, lze je na čtyřstěnnou kostku rozmístit dvěma různými způsoby, neboť každá konfigurace má svůj zrcadlový obraz, který s ní není totožný. Tomuto obrazu se mj. říká chirální enantiomorf³.

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 rozlišitelné tři kostky by tedy bylo možných konfigurací:

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

³Wikipedia, *Chiralita*: <https://cs.wikipedia.org/wiki/Chiralita>

1.4. Sestavení algoritmu

Cílem je maximalizovat parametr p tak, aby současně platily:

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

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

Algoritmus jsem se rozhodl implementovat v programovacím jazyce Rust⁴. Celý zdrojový kód naleznete [zde](#)⁵.

Základní idea algoritmu

1. Nejprve se vygenerují všechny číselné kombinace kostky. K vygenerování těchto kombinací jsem si napsal pomocnou funkci `fn generate_dice`, ve které volám funkci `fn combinations_with_replacement` z knihovny `itertools`⁶.

```
1 fn generate_dice(sides: usize) → Vec<Vec<usize>> {
2     (1..=6).combinations_with_replacement(sides).collect()
3 }
```

Seznam 1.1: Funkce `fn generate_dice`

2. Použitím tří vnořených cyklů se prověří, jestli stávající p splňuje $P(B > A) \geq p$, $P(C > B) \geq p$ a $P(A > C) > p$ pro některou z kombinací kostek A , B a C .

```
1 for a in &dice { // Check all combinations of A, B, C
2     for b in &dice {
3         for c in &dice {
4             let ba = pairwise_probability(b, a); // P(B > A)
5             let cb = pairwise_probability(c, b); // P(C > B)
6             let ac = pairwise_probability(a, c); // P(A > C)
7
8             if ba ≥ p && cb ≥ p && ac > p {
9                 /* ... do something */
10            }
11        }
12    }
13 }
```

Seznam 1.2: Kód na prověřování podmínek

⁴Programovací jazyk Rust: <https://www.rust-lang.org/>

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

⁶Knihovna `itertools` napsaná v jazyce Rust: <https://crates.io/crates/itertools>

3. Výpočet pravděpodobnosti $P(X > Y)$ zajišťuje následující funkce:

```

1 fn pairwise_probability(x: &[usize], y: &[usize]) → f64 {
2     let mut wins = 0; // Count the number of winning cases
3     for &xi in x.iter() { // Loop through each pair of x and y
4         for &yi in y.iter() {
5             if xi > yi { // If xi is greater than yi
6                 wins += 1; // increment the number of wins
7             }
8         }
9     }
10    let omega = (x.len() * y.len()) as f64; // Size of the probability space
11    wins as f64 / omega // Return probability
12 }

```

Seznam 1.3: Funkce `fn pairwise_probability`

Implementace algoritmu

Celý algoritmus (jeho naivní varianta) je zde:

```

1 fn find_max_p_naive(side_count: usize) → f64 {
2     println!("Finding maximal p:");
3     let mut max_p = 0.0; // Holds the maximum p
4     let dice = generate_dice(side_count); // Create all the dice combinations
5     let omega_size = side_count.pow(2);
6     let increment = 1.0 / omega_size as f64; // p grows by this step
7
8     for step in 0..omega_size { // Iterate from 0 to |Omega|
9         let p = step as f64 * increment; // Incrementing by 1/|Omega|
10        let mut valid = false; // No valid configs yet found
11        println!("Testing for p = {}: ", p);
12
13        'outer: // Tag to jump to from within the loop
14        for a in &dice { // Test out every single combination
15            for b in &dice {
16                for c in &dice {
17                    // Get the probabilities of P(B > A), P(C > B), P(A > C)
18                    let ba = pairwise_probability(b, a);
19                    let cb = pairwise_probability(c, b);
20                    let ac = pairwise_probability(a, c);
21                    if ba ≥ p && cb ≥ p && ac ≥ p { // Check them against p
22                        valid = true; // This p has a valid configuration
23                        println!("A={:?}, B={:?}, C={:?}", a, b, c);
24                        break 'outer; // Break out from the 3-nested loops
25                    }
26                }
27            }
28        }
29        // If for current p config doesnt exist, then return the last valid p
30        if !valid {
31            println!("No valid configurations!");
32            return max_p;
33        }
34
35        max_p = p; // If configuration exists assign p to max_p
36    }
37    max_p // By default return max_p
38 }

```

Seznam 1.4: Naivní varianta funkce `fn find_max_p`

Tato naivní varianta algoritmu je velmi neefektivní, neboť opakovaně počítá pravděpodobnosti mezi týmiž konfiguracemi kostek. V důsledku je algoritmus znatelně pomalý.

Je zřejmé, že by algoritmu přispělo prvně pravděpodobnosti všech kombinací dvou kostek předpočítat. Vypočtené hodnoty se uloží do matice (resp. pole polí). Konkrétně se použije dynamické pole `Vec<T>` ze standardní knihovny jazyka Rust.

Pro naplnění tohoto pole předpočtenými hodnotami pravděpodobností, jsem si napsal následující funkci, která vrací matici pravděpodobností kombinací kostek X a Y (obětuje se paměť pro rychlejší vyhledání hodnot pravděpodobností):

```

1 fn precompute_probabilities_vec(dice: &Vec<Vec<usize>>) → Vec<Vec<f64>> {
2     let size = dice.len();
3     // Matrix of X and Y
4     let mut cache: Vec<Vec<f64>> = vec![vec![0.0; size]; size];
5
6     for (i, x) in dice.iter().enumerate() {
7         for (j, y) in dice.iter().enumerate() {
8             // Insert P(X>Y) at [i,j]
9             cache[i][j] = pairwise_probability(x, y);
10        }
11    }
12
13    cache // Return the matrix of computed probabilities of X and Y
14 }

```

Seznam 1.5: Funkce `fn precompute_probabilities_vec`

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

```

1 fn find_max_p_caching_vec(side_count: usize) → f64 {
2     // ... (identical with the naive variant)
3     let cache = precompute_probabilities_vec(&dice); // ← precomputing
4
5     for step in 1..=16 {
6         let p = step as f64 * increment;
7         let mut valid = false;
8         'outer:
9         for (i, a) in dice.iter().enumerate() {
10             for (j, b) in dice.iter().enumerate() {
11                 if cache[j][i] < p { continue; } // skipping innermost loop
12                                                     // if P(A > B) doesnt hold
13                 for (k, c) in dice.iter().enumerate() {
14                     if cache[k][j] ≥ p && cache[i][k] > p {
15                         println!("A = {:?}, B = {:?}, C = {:?}", a, b, c);
16                         valid = true;
17                         break 'outer;
18                     }
19                 }
20             }
21         }
22
23         if !valid {
24             println!("No valid configurations!");
25             return max_p;
26         }
27
28         max_p = p;
29     }
30     max_p
31 }

```

Seznam 1.6: Rychlejší varianta `fn find_max_p` s předpočítanými hodnotami

Verze s předpočítanými hodnotami pravděpodobností výrazně zrychluje celý proces a to díky tomu, že místo opakovaného výpočtu pravděpodobnosti pro každý pár kostek mezi každým cyklem, máme uložené hodnoty v matici, což zrychlí nalezení pravěpodobnosti $P(X > Y)$ na konstantní čas $O(1)$.

Posledně jsem droubnou úpravou cyklů algoritmus paralelizoval:

```

1  fn find_max_p_parallel(side_count: usize) → f64 {
2      // ... (identical with the previous variant)
3
4      for step in 1..=16 {
5          let p = step as f64 * increment;
6          println!("Testing for p = {}: ", p);
7
8          // Iterations done in paralel
9          let valid = dice.par_iter().enumerate().any(|(i, _)| {
10             dice.par_iter().enumerate().any(|(j, _)| {
11                 if cache[j][i] < p { return false; }
12                 dice.par_iter().enumerate().any(|(k, _)| {
13                     cache[k][j] ≥ p && cache[i][k] > p
14                 })
15             })
16         });
17
18         if !valid {
19             println!("No valid configurations!");
20             return max_p;
21         }
22
23         println!("Config found (no printout available)!");
24         max_p = p;
25     }
26
27     max_p
28 }

```

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 maximální hodnota, kterou p může nabývat, je $\frac{9}{16}$ neboli 0.5625. Také jsem zjistil o jaké konfigurace kostek, které splňují dané podmínky, se přesně jedná. Dvě z nich dokonce odpovídají konfiguraci ve slovním zadání (index 0 a 3), 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 (lichý faktor).

2.1. Důkaz existence faktoru

Existenci faktoru stromu dokážeme indukcí – konkrétně jeho rekurzivní konstrukcí.

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

Základní případ

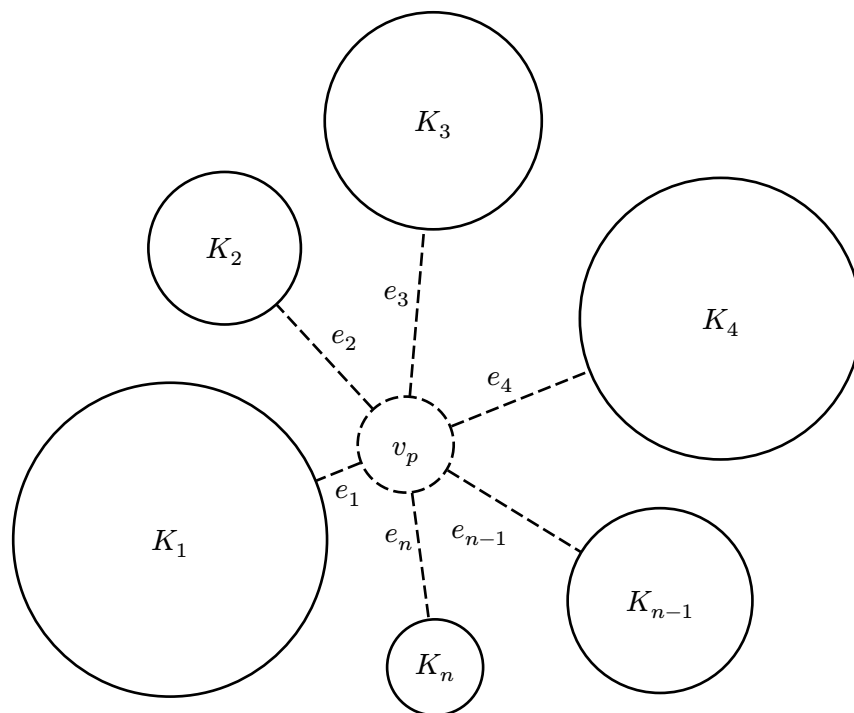
Pro strom T se dvěma vrcholy, $|V(T)| = 2$, je zřejmé, že jeho lichý faktor je právě T .

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 výtjeme ho z grafu. Výsledkem je graf $T - \{v_p\}$, jehož komponenty, které jsou rovněž stromy, označíme jako:

$$K_1, K_2, \dots, K_n \quad \text{kde } n = \deg_T(v_p)$$

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



Obrázek 2.2: Strom $T - \{v_p\}$, komponenty K_i a hrany e_i

Rozdělení komponent

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

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

Analýza počtu vrcholů

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

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

Dále platí:

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

kde znak \oplus představuje operátor symetrického rozdílu⁷ dvou množin.

Liché číslo lze sčítáním 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 počtů 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.

⁷Wikipedia, *Symmetric difference*: 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, jak získat liché číslo součtem lichých čísel (počet vrcholů komponent v L), je, když máme lichý počet (mohutnost množiny L) lichých čísel:

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

$$p = |L|$$

$$p(2k + 1) \equiv 1 \pmod{2}$$

$$2pk + p \equiv 1 \pmod{2}$$

$$p \equiv 1 \pmod{2}$$

Konstrukce faktoru

Pro komponentu $K \in L$ neexistuje faktor $F = (V(K), E_F)$, kde by $E_F \subseteq E(K)$ a každý vrchol $v \in V(F)$ 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 to by tento princip porušovalo.

Abychom zajistili sudost, přidáme do každé komponenty $K \in L$ zpět výjmutý vrchol v_p (včetně hrany e_i), čímž získáme stromy $K + \{v_p\}$ sudého řádu. Jelikož jsou sudé, můžeme na ně aplikovat indukční krok.

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

Vrchol v_p bude mít stupeň roven počtu komponent v L :

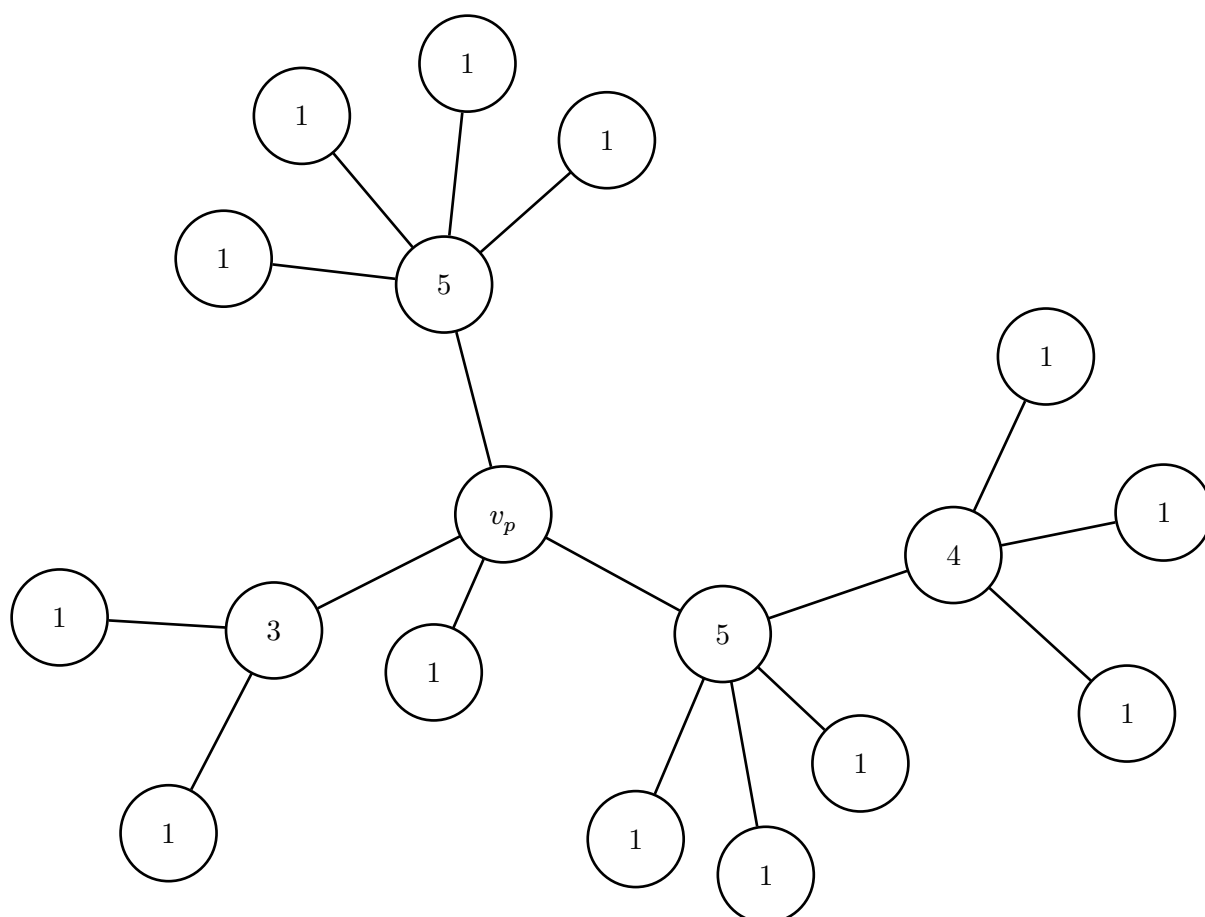
$$\deg(v_p) = |L| \equiv 1 \pmod{2}$$

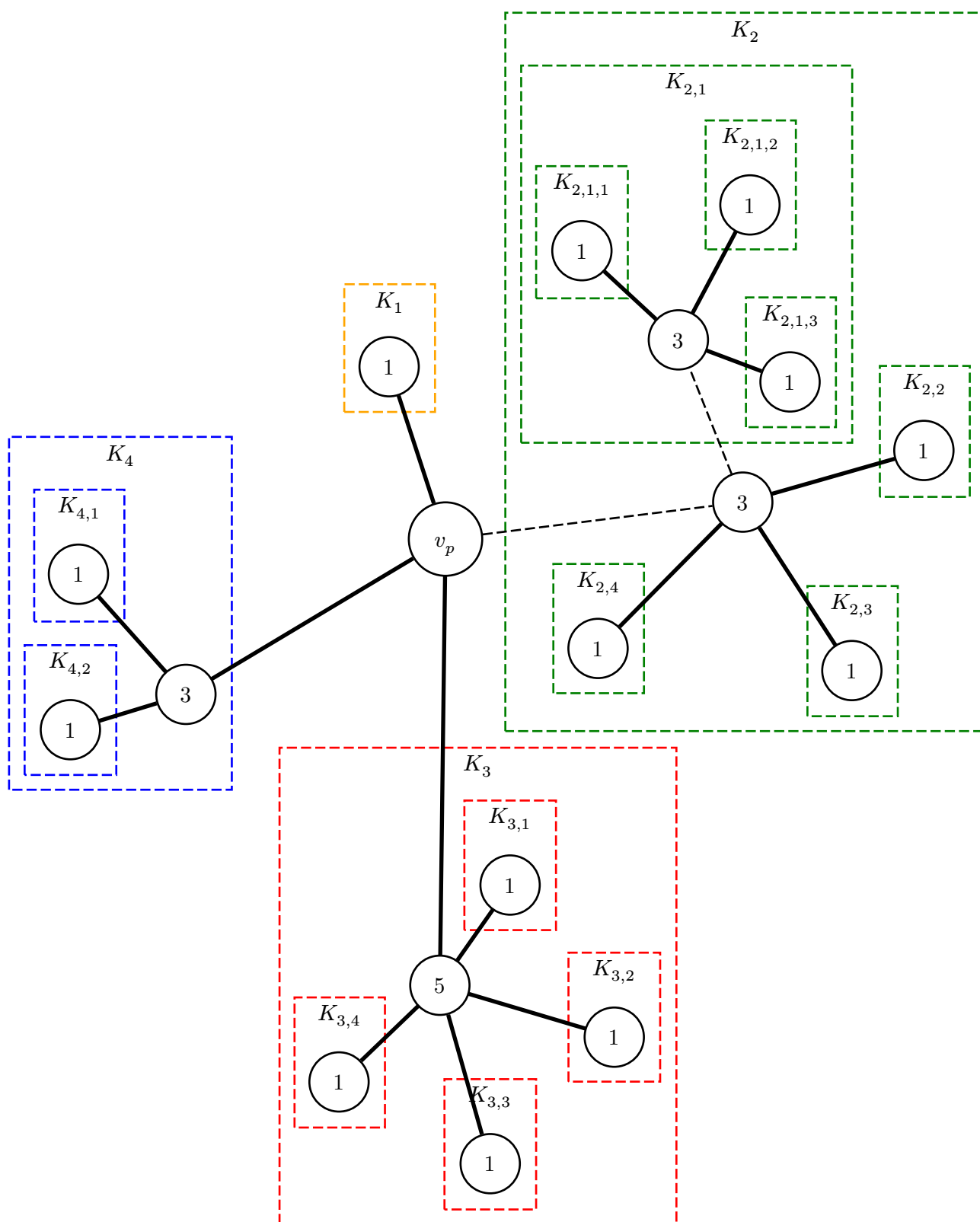
Na konci rekurze (indukce) bude každý vrchol $v \in V(T)$ lichého stupně, protože:

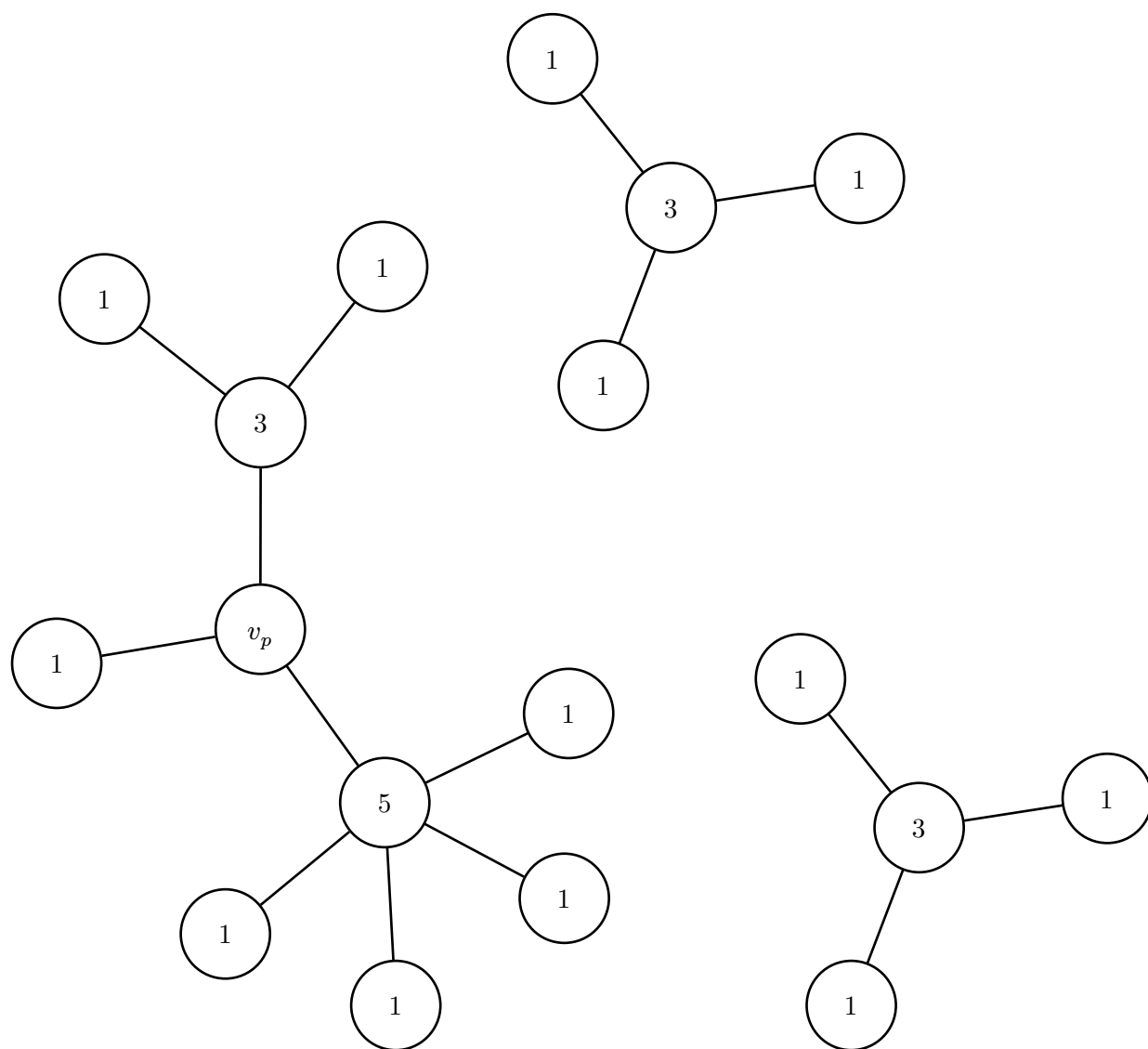
$$\deg(v) = |L|$$

přičemž mohutnost L je v každém kroku lichá. Tím je existence lichého faktoru dokázána. \square

Ukázka algoritmu (rekurze/indukce)

Obrázek 2.3: Strom T_{18} s vyznačenými stupni vrcholů a vybraným vrcholem v_p

Obrázek 2.4: Rekurzivní konstrukce lichého faktoru stromu T_{18}

Obrázek 2.5: Nalezený lichý faktor stromu T_{18}

2.2. Důkaz jednoznačnosti faktoru F

Po odůvodnění existence faktoru F nyní dokážeme, že pro strom T existuje právě jeden takový faktor F , ve kterém mají všechny jeho vrcholy lichý stupeň.

Důkaz sporem

Předpokládejme, že existují dva různé liché faktory stromu T . Mějme faktory F_A a F_B , o kterých tvrdíme, že jsou od sebe odlišné. Tedy musí platit, že jedna z nich má alespoň jednu hranu, která náleží pouze jím ale ne druhému:

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

Vytvoříme nový graf G . Ten obsahuje všechny vrcholy stromu T a hrany symetrického rozdílu hran faktorů F_A a F_B . To znamená, že G obsahuje ty hrany, které se objevují právě v jednom z faktorů, nikoliv v obou.

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

Pokud je graf G nulovým grafem (nemá žádné hrany):

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

pak platí, že $E(F_A) = E(F_B)$, což vede k závěru, že faktory F_A a F_B popisují tentýž graf.

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

Sudost a lichost stupní vrcholů v grafu G závisí na tom, kolik hran z faktorů F_A a F_B budou zahrnuty do výsledného grafu G . Množinu hran faktoru F_X , které jsou incidentní s vrcholem v zapišme jako:

$$E_v(F_X)$$

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:

$$\begin{aligned} |E_v(F_A)| &= |E_v(F_B)| \equiv 1 \pmod{2} \\ \deg_G(v) &= |E_v(F_B)| + |E_v(F_A)| \equiv 0 \pmod{2} \end{aligned}$$

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

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

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

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

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

pak jsou všechny vrcholy grafu G také sudého stupně, což nyní dokážeme. Nejdříve si však zjednodušíme 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 v průniku množin A a B sudý, pak stupeň v je sudý:

$$|A \cap B| \equiv 0 \Rightarrow |A \setminus B| = |B \setminus A| \equiv 1 \pmod{2}$$

$$\deg_G(v) = |A \setminus B| + |B \setminus A| \equiv 0 \pmod{2}$$

Pokud je počet hran v průniku množin A a B lichý, potom stupeň v je také sudý:

$$|A \cap B| \equiv 1 \Rightarrow |A \setminus B| = |B \setminus A| \equiv 0 \pmod{2}$$

$$\deg_G(v) = |A \setminus B| + |B \setminus A| \equiv 0 \pmod{2}$$

Závěr

Nyní víme, že vrcholy v grafu $G = (V(T), E(F_A) \oplus E(F_B))$ budou vždy sudého stupně:

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

Pokud $k = 0$, pak $F_A = F_B$ (popisují stejný graf). Uvažujeme-li, že $k > 0$, dostaneme vrcholy, jejichž stupeň je alespoň dva. Minimální graf, kde jsou všechny vrcholy alespoň druhého stupně, je graf cesty. Cesty jsou cyklické, zatímco stromy jsou acyklické - je zde kontradikce. Nemůžeme z acyklického stromu získat cyklický graf.

Tedy jedinou přípustnou možností je, že $k = 0$ a faktory F_A a F_B popisují jeden a ten samý graf. Jednoznačnost faktoru F je dokázána. \square