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: 29. 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

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, 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 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] 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.

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

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. □

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.

$$P(B > A) = \frac{P(A=1) \cdot P(B > A = 1) \cdot |\Omega| + P(A=4) \cdot P(B > A = 4) \cdot |\Omega|}{|\Omega|}$$

$$= |\Omega| \ \frac{P(A=1) \cdot P(B>1) + P(A=4) \cdot P(B>4)}{|\Omega|}$$

$$= P(A=1) \cdot P(B>1) + P(A=4) \cdot P(B>4) \tag{1.1}$$

$$= P(A=1) \cdot P(B) + P(A=4) \cdot P(B=5)$$
$$= \frac{1}{4} \cdot 1 + \frac{3}{4} \cdot \frac{1}{2}$$
$$= \frac{2}{8} + \frac{3}{8} = \frac{5}{8} = \underline{\underline{0.625}}$$

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

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:

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

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:

[2] https://en.wikipedia.org/wiki/Law_of_total_probability

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

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 $A > C$. □

1.3. Různá rozmístění čísel na kostkách

Kostky A, B a C jsou rozlišitelné, např. mají jiné barvy. Kolik existuje různých konfigurací čísel, když čísla vybíráme z množiny $[1, 6]$ s možností opakovaní.

Zpusobů jak vybrat čtyři čísla z sešti (s možností opakování) je:

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

Proto možných konfigurací rozmístění čísel na třech kostkách je:

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

Jiná úvaha (kuličky a přehrádky)

Výpočet různých konfigurací pro jednu kostku, můžeme také zapsat jako:

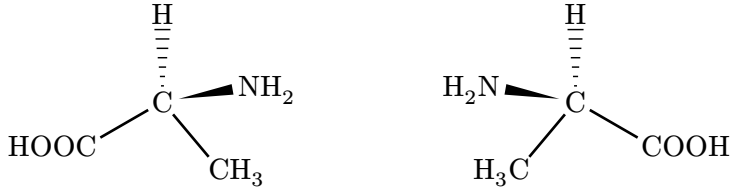
$$\underbrace{\binom{6}{4}\binom{3}{3}}_{a} + \underbrace{\binom{6}{3}\binom{3}{2}}_{b} + \underbrace{\binom{6}{2}\binom{3}{1}}_{c} + \underbrace{\binom{6}{1}\binom{3}{0}}_{d} = 126 \tag{1.3}$$

a) Vybereme 4 různá čísla z 6 možných. Umístíme 3 oddělovače mezi 4 stěnami kostky. Vzniklé 4 přehrádky naplníme těmito čtyřmi čísly.
b) Vybereme 3 čísla z 6 možných. Zvolíme dvě pozice ze tří, kam umístit 2 oddělovače mezi 4 stěnami. Vzniklé 3 přehrádky naplníme těmito třemi čísly.
c) Vybereme dvě čísla z šesti možných. Zvolíme jednu ze tří pozic, kam umístíme jeden oddělovač. Vznikle 2 přehrádky naplníme těmito dvěma čísly.
d) Vybereme 1 číslo z 6 možných. Máme jenom 1 přehrádku, kterou naplníme oným číslem.

Enantiomorfy kostek

Rovnice (1.2) a Rovnice (1.3) platí ale pouze tehdy, nebereme-li v úvahu různá rozmístění pevně zvolených čísel na kostce. Jestliže máme 4 různá čísla, tak jsme schopni tyto čísla na čtyrstěnnou kostku umístit dvěma různými způsoby. Neboť pro čtyřstěnnou kostku (se čtyřmi různými čísly) existuje její zrcadlový obraz, který s ní není identický. Tento obraz tzv. představuje chirální enantiomorf[3] kostky. Ty jsou také vůči sobě izomorfní.

Příklad z chemie:



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

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

Pokud bychom tedy chtěli být zcela přesní, tak Rovnice (1.3) (důvod proč byla zahrnuta) upravíme následovně:

$$\underbrace{2 \cdot \binom{6}{4}}_{\text{korekce}} + \binom{6}{3}\binom{3}{1} + \binom{6}{2}\binom{3}{1} + \binom{6}{1} =$$
$$= 2 \cdot 15 + 20 \cdot 3 + 15 \cdot 3 + 6 =$$
$$= 30 + 60 + 45 + 6 = \underline{141}$$

Pro tři kostky bude možných konfigurací $(141)^3 = \underline{\underline{2803221}}$.

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: https://github.com/phatt-23/Projekt-9-DIM/blob/master/program/src/main.rs

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] https://www.rust-lang.org/

```
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 pro prověřování podmínek

Implementovaný algoritmus

Celý algoritmus (naivní varianta):

```
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 však velmi neefektivní, protože opakovaně počítá pravděpodobnosti mezi týmiž kostkami. V důsledku je alogitmus velmi 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, [0] a [3], dokonce odpovídají konfiguraci ve slovním zadání, neberu-li v potaz jejich označení.

Teorie grafů

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

2.1. Důkaz existence faktoru F

Graf T je strom sudého řádu.

$$G = (V, E)$$
$$|V| \equiv 0 \pmod 2$$

Jelikož počet vrcholů je sudý, tak graf T musí mít sudý počet lichých stupní. Pokud by počet lichých stupní byl lichý, potom bychom porušili princip sudosti, jenž uvádí, že:

$$\sum_{v \in V(T)} \deg(v) = 2|E|.$$

Dokažme si, že stupňová posloupnosti s lichým počtem lichých stupní neexistuje. Mějme libovolný graf G sudého řádu.

$$G = (V, E) \quad |V| = n = 2k, k \in \mathbb{Z}$$
$$D_G = (d_1, d_2, ..., d_n)$$
$$k_e = \Big|\{d \in D : d \equiv 0 \pmod 2\}\Big|$$
$$k_o = \Big|\{d \in D : d \equiv 1 \pmod 2\}\Big|$$
$$k_o \equiv 1 \pmod 2 \Leftrightarrow k_e \equiv 1 \pmod 2$$
$$\sum_{v \in V} \deg(v) = k_e \cdot \text{sudá} + k_o \cdot \text{lichá} = \text{sudá} + \text{lichá} = \underline{\text{lichá}}$$

Součet stupní vrcholů vyšel lichý. To je dle principu sudosti nepřípustné, takový graf neexistuje. Stupňová posloupnost se tedy bude vždy skládat ze sudého počtu lichých stupní a sudého počtu sudých stupní.

Pro hledaní lichého faktoru F stromu T si pomožme tím, že si představíme jeho stupňovou posloupnost. Je nutné podotknout, že jedna posloupnost může popisovat více stromů. To nám však nevadí, jelikož pracujeme se stromy obecně, nikoliv s konkretními stromy.

Pokud se tato posloupnost skládá z lichých čísel, tak jsme jsme našli lichý faktor F grafu T. Faktorem F je totiž strom G samotný.

Pokud tato posloupnost obsahuje sudá čísla, budeme hodnoty této posloupnosti postupně po párech dekrementovat, dokud nedostaneme lichou posloupnost. Dekrementujeme po párech, jelikož jedna hrana grafu je incidentní se dvěma vrcholy grafu.

Všechna kombinace dekrementace ve stupňovové posloupnosti jsou znázorněna zde:

$$(..., \text{lichá}, ..., \text{lichá}, ...) \overset{(-1)}{\mapsto} (..., \text{sudá}, ..., \text{sudá}, ...)$$
$$(..., \text{sudá}, ..., \text{sudá}, ...) \overset{(-1)}{\mapsto} (..., \text{lichá}, ..., \text{lichá}, ...)$$
$$(..., \text{lichá}, ..., \text{sudá}, ...) \overset{(-1)}{\mapsto} (..., \text{sudá}, ..., \text{lichá}, ...)$$
$$(..., \text{sudá}, ..., \text{lichá}, ...) \overset{(-1)}{\mapsto} (..., \text{lichá}, ..., \text{sudá}, ...).$$

> ⓘ **Poznámka**
>
> Mějme na paměti, že stupně rovno jedné nesmíme snižovat. Tyto stupně odpovídají stupňům listů, jehož incidentní hrany musí být ve faktoru F. Pokud bychom tyto hrany odstranili, listy by byly stupně nula - nula není liché číslo.

Nakonec se nám vždy podaří lichou posloupnost(i) dostat. Bez ohledu na to jakým způsobem provedeme postupné snížování stupňí v posloupnosti, je zaručeno, že jedním z těchto vysledných posloupností je právě ona posloupnost faktoru F grafu G.

Toto postupné snižování sekvence a vysledné nalezení sekvence s lichými hodnotami, funguje, jelikož víme že počet vrcholů je sudý. Fakt, že existuje sudý počet lichých a sudých stupňů se po jakékoli dekrementaci, nemění (není porušen princip sudosti).

Pro strom T sudého stupně skutečně existuje lichý faktor F. □

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.

Předpokladejme, že existují dva liché faktory grafu G. Mějme faktory F_1, F_2, pro které tvrdíme, že jsou od sebe odlišné. Tedy musí platit, že mají alespoň jednu hranu, která náleží pouze jim a ne druhému.

$$\exists e \in E(F_1), e \notin E(F_2)$$

Z těchto dvou faktoru můžeme vytvořit nový graf, který obsahuje všechny vrcholy stromu T, které jsou spojeny pouze těmi hranami, které se objevují právě v jednom z faktoru, nikoli v obou.

$$(V(T), E(F_1) \oplus E(F_2)) \quad \text{nebo také} \quad (V(T), E(F_1) \Delta E(F_2))$$
$$\text{zkráceně potom}$$
$$F_1 \Delta F_2$$

Při rozhodování, zda hranu e do $F_1 \Delta F_2$ zahrneme, postupujeme takto:

hrana e je zahrnuta v F_1	hrana e je zahrnuta v F_2	hrane e je zahrnuta v $F_1 \Delta F_2$
ano	ano	ne
ne	ano	ano
ano	ne	ano
ne	ne	ne

nebo stručněji:

$e \in E(F_1)$	$e \in E(F_2)$	$e \in E(F_1)\Delta E(F_2)$
1	1	0
0	1	1
1	0	1
0	0	0

Pokud je hrana e obsažena v obou faktorech, tak se vyruší. přičemž zahrneme všechny hrany incidentní s v v F_2 ($E^v_{F_2}$). Obou jich je lichý počet, proto stupeň vrcholu v je sudý. Tedy:

$$\begin{aligned} |E^v_{F_1}| + |E^v_{F_2}| &= \deg_{F_1 \Delta F_2}(v) \\ \text{liché} + \text{liché} &= \underline{\text{sudé}}. \end{aligned}$$

Pokud se všechny hrany $E^v_{F_1}$ shodují s hranami $E^v_{F_2}$, tak nezahrneme ani jednu z nich - zahrneme 0 hran (sudý počet).

Pokud si některé hrany nachází $E^v_{F_1}$ a přitom také v $E^v_{F_2}$, tak tyto hrany do $F_1 \Delta F_2$ nezahrneme. Nýbrž zahrneme zbytek hran, kterých musí být sudý počet.

Víme totiž, že pokud je počet hran v $E^v_{F_1} \setminus E^v_{F_2}$ lichý, tak počet hran náležící oběma faktorům, $|E^v_{F_1 \cap F_2}|$, je sudý. Tedy hran v $E^v_{F_2} \setminus E^v_{F_1}$ musí být lichý počet.

$$\begin{aligned} |E^v_{F_1} \setminus E^v_{F_2}| + |E^v_{F_2} \setminus E^v_{F_1}| &= \deg_{F_1 \Delta F_2}(v) \\ \text{liché} + \text{liché} &= \underline{\text{sudé}}. \end{aligned}$$

Pokud je $|E^v_{F_1} \setminus E^v_{F_2}|$ sudé, tak je $|E^v_{F_1 \Delta F_2}|$ liché, tedy $|E^v_{F_2} \setminus E^v_{F_1}|$ je sudé.

$$\begin{aligned} |E^v_{F_1} \setminus E^v_{F_2}| + |E^v_{F_1} \setminus E^v_{F_2}| &= \deg_{F_1 \Delta F_2}(v) \\ \text{sudé} + \text{sudé} &= \underline{\text{sudé}}. \end{aligned}$$

Pro:

$$T = (V_T, E_T) \text{ je strom, kde } |V_T| \equiv 0 \pmod 2,$$
$$F_1 = \Big(V_T, E_{F_1} \subseteq E_T\Big), \text{ kde } \forall v \in V(F_1), \deg_{F_1}(v) \equiv 1 \pmod 2,$$
$$F_2 = \Big(V_T, E_{F_2} \subseteq E_T\Big), \text{ kde } \forall v \in V(F_2), \deg_{F_2}(v) \equiv 1 \pmod 2,$$

platí, že:

$$\forall v \in V(F_1 \Delta F_2), \deg_{F_1 \Delta F_2}(v) \equiv 0 \pmod 2.$$

Poznámka

Obecný vzorec pro určení sudosti a lichosti stupně libovolého vrcholu v v symetrické diferenci faktorů S_1 a S_2 stromu T je:

$$\deg_{S_1 \Delta S_2}(v) = \Big(\deg_{S_1}(v) + \deg_{S_2}(v)\Big) \bmod 2.$$

Z toho plyne, že stupně vrcholů $F_1 \Delta F_2$ jsou všechny sudé. Můžou tedy nabývat hodnot $2k, k \in \mathbb{N}_0$ - to je včetně nuly. Uvažíme-li, že vrcholy nejsou stupně 0, dostaneme vrcholy, které jsou nuceny mít stupeň alespoň 2. Pokud by měli všechny vrcholy stupeň rovno dvěma, takový graf by spadal do třídy grafů zvané jako cesty. Cesty jsou cyklické - stromy jsou jsou acyklické - je zde kontradikce. Ve stromě T jsme našli cyklus - není možné. Je zjevné, že jakýkoliv graf s vyššími stupněmi vrcholů taktéž obsahuje cyklus. Tedy jedinou přípustnou možností je, že stupně vrcholů $F_1 \Delta F_2$ jsou nulové, tedy $F_1 \Delta F_2$ je nulový graf. Z toho plyne, že fakory F_1 a F_2 popisují stejný graf.

$$F_1 \Delta F_2 = \emptyset \rightarrow F_1 = F_2$$

Došli jsme k závěru, že pro strom sudého řádu existuje právě jeden lichý faktor. □

Bibliografie