

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 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	17
Bibliografie	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. 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 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 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. 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. 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 hození 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.

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

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

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

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

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

Velikost pravděpodobnostního prostoru je $|\Omega| = 16$. Z rozepsaného 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 její hozenou hodnotu. Pokud padla na kostce A hodnota 4, tak B vyhraje pouze tehdy, když byla vržena hodnota 5. To zapíšeme a vypočítáme následně.

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

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

Zbývající pravděpodobnosti $P(C > B)$ a $P(A > C)$ bychom mohli vypočítat obdobně jako $P(B > A)$. Lze to ale udělat lépe? Všimněte si Rovnice (1.1). Její tvar můžeme využitím zákona celkové pravděpodobnosti² zobecnit.

1.2.2. Výpočet pomocí zákona celkové pravděpodobnosti

Zákon celkové pravděpodobnosti uvádí, že máme-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 disjunktí podmínky pokrývající celý pravděpodobnostní prostor Ω , tedy:

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

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

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

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

Padne-li na kostce Y např. 1 nemůže zároveň padnout 3 nebo 5 apod. Obecný vzorec pro výpočet pravděpodobnosti $P(X > Y)$ 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, například mají odlišné barvy. Jaký je počet různých konfigurací čísel, pokud vybíráme čísla z množiny $[1, 6]$ s možností opakování.

Počet 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:

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

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 kroků:

- 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 umístit jeden oddělovač. Vzniklé dvě přehrádky naplníme těmito čísly.

- d) *Vybereme jedno číslo z šesti možných.*

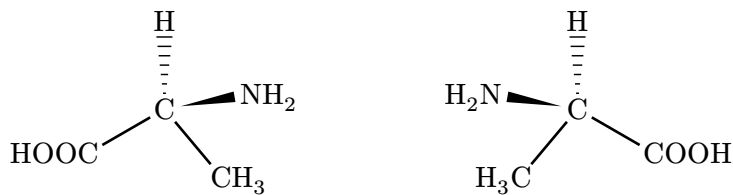
Máme pouze jednu přehrádku, kterou naplníme tímto číslem.

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 enantiomorf³.

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

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



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:

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

1.4. Sestavení algoritmu

Cílem je maximalizovat p tak, aby současně platily:

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

přičemž 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 Rust⁴. 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 vygenerování kombinací čísel kostek jsem si napsal pomocnou funkci `fn generate_dice`, ve které volám funkci `fn combinations_with_replacement` z knihovny `itertools`.

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

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

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/>

```

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

```

Seznam 1.3: Kód pro prověřování podmínek

Implementovaný algoritmus

Celý algoritmus (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 have yet been 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; // Jump out of loops (to the 'outer tag)
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 to max_p
36     }
37     max_p // By default return max_p found
38 }

```

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 algoritmus 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 pravděpodobností kombinací kostek X a Y (obětuje paměť za zaručení rychlejšího vyhledání hodnoty pravděpodobnosti):

```

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 previous)
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) doesn't 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     // ... (identical with the previous)
31 }

```

Seznam 1.6: Upravená funkce `fn find_max_p`

Posledně jsem droubnou úpravou cyklů algoritmus paralelizoval:

```

1 fn find_max_p_parallel(side_count: usize) → f64 {
2   // ... (identical with the previous)
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 parallel
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  // ... (identical with the previous)
28 }

```

Seznam 1.7: Paralelní 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, [0] a [3], dokonce odpovídají konfiguraci ve slovním zadání, neberu-li v potaz jejich označení.

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 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 $v \in V(T)$, kde $\deg_T(v) > 1$ (tj. nebereme listy), a výjme ho z grafu. Výsledkem je graf $T - \{v\}$, jehož komponenty, které jsou rovněž stromy, označíme jako:

$$K_1, K_2, \dots, K_{\deg_T(v)}$$

Hranu, která spojuje vrchol v s komponentou K_i , označíme jako 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:

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

Analýza počtu vrcholů

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

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

Dále platí:

$$\begin{aligned} T - \{v\} &= \bigoplus_{K \in S} K \oplus \bigoplus_{K \in L} K \\ V(T - \{v\}) &= \bigoplus_{K \in S} V(K) \oplus \bigoplus_{K \in L} V(K) \end{aligned}$$

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

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 platí:

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

protože suma počtů vrcholů komponent v S je vždy sudá:

$$|V(K)| = 2k \quad \text{pro } k \in \mathbb{N}$$

$$p(2k) \equiv 0 \pmod{2}$$

$$0 \equiv 0 \pmod{2}$$

$$p = \mathbb{Z}$$

Z čehož plyne:

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

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

Počet komponent v L

Součet vrcholů komponent v S je vždy sudý. Naopak součet vrcholů komponent v L je 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, je, když lichých čísel máme lichý počet.

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

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

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

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

$$p = 2t + 1 \quad \text{kde } t \in \mathbb{Z}$$

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 (včetně hrany e_i), čímž získáme strom $K + \{v\}$ sudého řádu. Na tento strom aplikujeme indukční krok. Vrchol v bude mít stupeň roven počtu komponent v L :

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

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

Na konci indukce (rekurze) bude každý vrchol $v \in V(T)$ mít lichý stupeň, protože

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

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

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

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

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	hrana 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 ($E_{F_2}^v$). Obou jich je lichý počet, proto stupeň vrcholu v je sudý. Tedy:

$$|E_{F_1}^v| + |E_{F_2}^v| = \deg_{F_1 \Delta F_2}(v)$$

$$\text{liché} + \text{liché} = \underline{\text{sudé}}.$$

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

Pokud si některé hrany nachází $E_{F_1}^v$ a přitom také v $E_{F_2}^v$, 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 $E_{F_1}^v \setminus E_{F_2}^v$ lichý, tak počet hran náležící oběma faktorům, $|E_{F_1 \cap F_2}^v|$, je sudý. Tedy hran v $E_{F_2}^v \setminus E_{F_1}^v$ musí být lichý počet.

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

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

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

Pro:

$$T = (V_T, E_T) \text{ je strom, kde } |V_T| \equiv 0 \pmod{2},$$

$$F_1 = (V_T, E_{F_1} \subseteq E_T), \text{ kde } \forall v \in V(F_1), \deg_{F_1}(v) \equiv 1 \pmod{2},$$

$$F_2 = (V_T, E_{F_2} \subseteq E_T), \text{ kde } \forall v \in V(F_2), \deg_{F_2}(v) \equiv 1 \pmod{2},$$

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:

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

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 spadl do třídy grafů zvané jako cesty. Cesty jsou cyklické - stromy 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 faktory 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. \square

Bibliografie