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: 28. 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

1. 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.

2. Kombinatorika

2.1. 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) a nezáleží na pořadí čísel, např. $\{1, 4, 4, 4\}$ a $\{4, 4, 4, 1\}$ považujeme za stejné.

3. **Maximalizace pravděpodobností**

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

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

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

2.2. Netrazitivita 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:

$$\begin{aligned} P(B > A) &> 0.5, \\ P(C > B) &> 0.5 \\ a\ P(A > C) &> 0.5. \end{aligned}$$

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:

$$P(X > Y) = \frac{\text{počet případů, kdy číslo na } X \text{ je větší než číslo na } Y}{\text{celkový počet možných kombinací výsledků}},$$

stručně:

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

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

2.2.1. Výpočet $P(B > A)$

První varianta řešení

Pravděpodobnostní prostor $\Omega_{B,A}$ je kartezský součin čísel na kostkách B a A:

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

$$\begin{aligned} \Omega_{B,A} = \Big\{ & (2{,}1), (2{,}1), (5{,}1), (5{,}1), \\ & (2, 4), (2, 4), (5{,}4), (5{,}4), \\ & (2, 4), (2, 4), (5{,}4), (5{,}4), \\ & (2, 4), (2, 4), (5{,}4), (5{,}4)\Big\}. \end{aligned}$$

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

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

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

2.2.2. Druhá varianta řešení

Pravděpodobnostní prostorem je stále $\Omega_{B,A}$. Využijeme toho, že se kostky skládají z 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.

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

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

Zbývalé pravděpodobnosti $P(C > B)$ a $P(A > C)$ lze udělat obdobně.

2.2.3. Výpočet $P(C > B)$

$$C = \{3, 3, 3, 6\}, \quad B = \{2, 2, 5, 5\}, \quad \Omega_{C,B} = \Big\{(c, b) : c \in C, b \in B\Big\}$$

$$\begin{aligned}
\Omega_{C,B} = \Big\{&(3, 2), (3, 2), (3, 5), (6, 5), \\
&(3, 2), (3, 2), (3, 5), (6, 5), \\
&(3, 2), (3, 2), (3, 5), (6, 5), \\
&(3, 2), (3, 2), (3, 5), (6, 5)\Big\}.
\end{aligned}$$

If $C = 3$, then B will win $\frac{1}{2}$ of a time, meaning C will win $\frac{1}{2}$ of the time.

$$P(B > C = 3) = \frac{1}{2} \;\mapsto\; P(C = 3 > B) = \frac{1}{2}$$

The number of outcomes where $C = 3$ is:

$$\left|\Big[\{c, b\} \in \Omega_{C,B} : c = 3\Big]\right| = 3 \cdot 4 = 12.$$

If C is not 3, it is 6. It that case it will win every time.

$$P(B > C = 6) = 0 \;\mapsto\; P(C = 6 > B) = 1$$

$$\left|\left[\{c,b\} \in \Omega_{C,B} : \neg(b=3)\right]\right| = 1 \cdot 4 = 4$$

That means C will win over B with the probability of:

$$P(C > B) = \frac{\frac{1}{2}12 + 4}{\Omega_{C,B}} = \frac{6+4}{16} = \frac{10}{16} = \frac{5}{8} = 0.625$$

□

2.3. Proof of $P(A > C)$

2.3.1. First method

The sample space here is:

$$\Omega_{A,C} = \left[\{a,c\} : a \in A, c \in C\right].$$

$$\begin{aligned}\Omega_{A,C} = \big[&\{1,3\},\{1,3\},\{1,3\},\{1,6\},\\ &\{4,3\},\{4,3\},\{4,3\},\{4,6\},\\ &\{4,3\},\{4,3\},\{4,3\},\{4,6\},\\ &\{4,3\},\{4,3\},\{4,3\},\{4,6\}\big].\end{aligned}$$

From sheer observation of $\Omega_{A,C}$ we conclude the number of outcomes where A is victorious.

$$P(A > C) = \frac{9}{|\Omega_{A,C}|} = \frac{9}{16} = 0.5625$$

The probability of A winning over C is greater than half, meaning A, although not by a huge margin, is overall better then C. □

2.3.2. Second method

If $A = 1$ than it lose no matter what.

$$P(A = 1 > C) = 0 \ \mapsto\ P(C > A = 1) = 1$$

If A is not 1, it is 4 and because the distribution of the numbers on C is 3 on $\frac{3}{4}$ of the sides and 6 on $\frac{1}{4}$ of the sides, A will win $\frac{3}{4}$ of the time.

$$P(A = 4 > C) = \frac{3}{4} \ \mapsto\ P(C > A = 4) = \frac{1}{4}$$

The number of events in $\Omega_{A,C}$ where $A = 1$ is 4. It will lose in all of them.

$$P(A = 1 > C) \cdot |\Omega_{A,C}^{A=1}| = 0 \cdot 4 = 0$$

The number of events in $\Omega_{A,C}$ where $\neg(A = 1) \mapsto A = 4$ is 12. In $\frac{3}{4}$ of these 12 outcomes it will be the winning dice. That means it will win in $\frac{3}{4}12 = 9$ of the possible outcomes.

$$P(A = 4 > C) \cdot |\Omega_{A,C}^{A=4}| = \frac{3}{4} \cdot 12 = 9$$

Then the probability of A winning over C is:

$$P(A > C) = \frac{9}{\Omega_{A,C}} = \frac{9}{16} = 0.5625.$$

A is better than C. □

2.4. Vypočet pomoci zákonu celkové pravděpodobnosti

Zákon celkové pravděpodobnosti (zdroj) říká, 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 a pokrývají celý pravdepodobnostní prostor Ω, tedy

$$\bigcup_{i=0}^{n} C_i = \Omega.$$

Událostmi jsou v našem případě $X > Y$ (kostka X vyhraje nad 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, proto platí

$$\bigcup_{i=0}^{n} (Y = y_i) = \Omega_{XY}.$$

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

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

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

Pro připomenutí:

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

Výpočty pravděpodobností pomocí odvozeného vzorce:

$$
\begin{aligned}
P(B>A) &= \sum_{a\in A} P(A=a)\cdot P(B>a) = \\
&= P(A=1)\cdot P(B>1) + P(A=4)\cdot P(B>4) = \\
&= \frac{1}{4}\cdot\frac{4}{4} + \frac{3}{4}\cdot\frac{1}{2} = \\
&= \frac{2}{8} + \frac{3}{8} = \frac{5}{8} = \underline{\underline{0.625}}
\end{aligned}
$$

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

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

Došli jsme ke stejným vysledkům.

2.5. Různé 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

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

Jiná úvaha

Různých čísel vybereme buď jeden, dva, tři nebo čtyři. Původní kombinací s opakovanim můžeme přepsat na

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

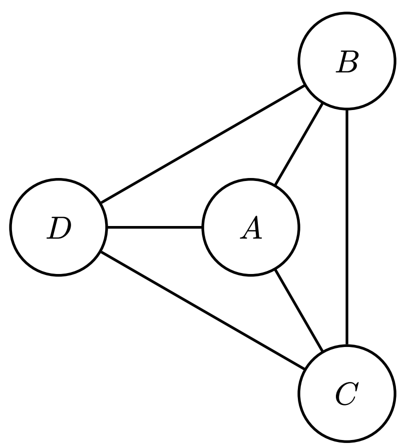
a) Vybereme různá 4 čísla z 6 a vybereme 3 pozice ze 3, kam umístit 3 z nich, poslední pozice je dána jednoznačně.

b) Vybereme 3 čísla a vybereme 2 pozice ze 3, kam umístit 2 z nich (různá čísla), poslední pozice naplníme čísly, které se objevují dvákrát.
c) Vybereme 2 čísla z 6 a vybereme 1 pozice ze 3, kam je umístit jednu z nich, poslední pozice jsou dány jednoznačně.
d) Vybereme 1 číslo z 6, pozice jsou dáný jednoznačně.

Nesmime vsak zapomenout ze pro ctyrstennou kostku plati, ze mame-li 4 ruzna cisla, tak je jsme na kostku schopni umistit dvema ruznymi zpusoby.

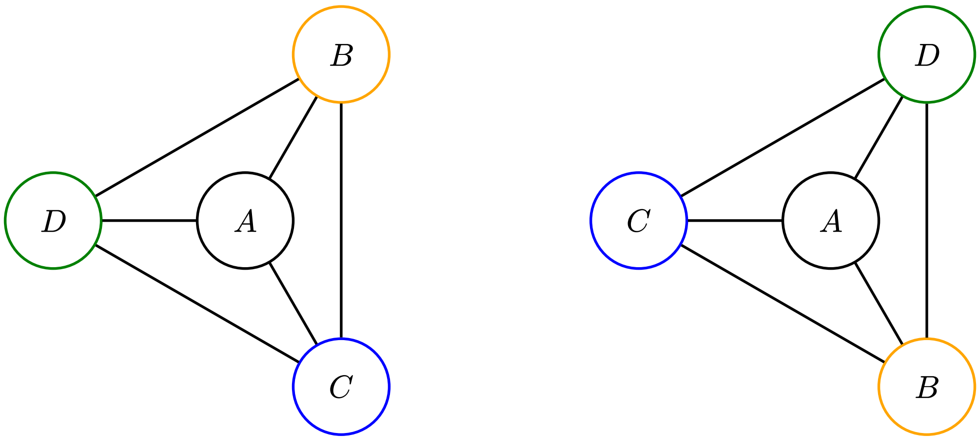
Pokud se vybere stejne cislo ctyrikrat, tak tyto cisla muzeme nanest na ctyrstennou kostku jednim jedinym zpusobem. Pokud se vyberou dve ruzna cisla, tak je na ctyrstennou kostku muzeme nanest jednim zpusobem. Rovnez, vyberou-li se tri cisla, tak tyto cisla jsou mozne na kostku nanest jedinym zpusobem.

Mejme ctyri ruzna cisla A, B, C a D. Ctyrstennou kostku a ni nanesene cisla znazornime grafem. Hrany jsou prechody mezi stenami nizsi hodnoty do vyssi. Jedna ze dvou moznych konfiguraci je:

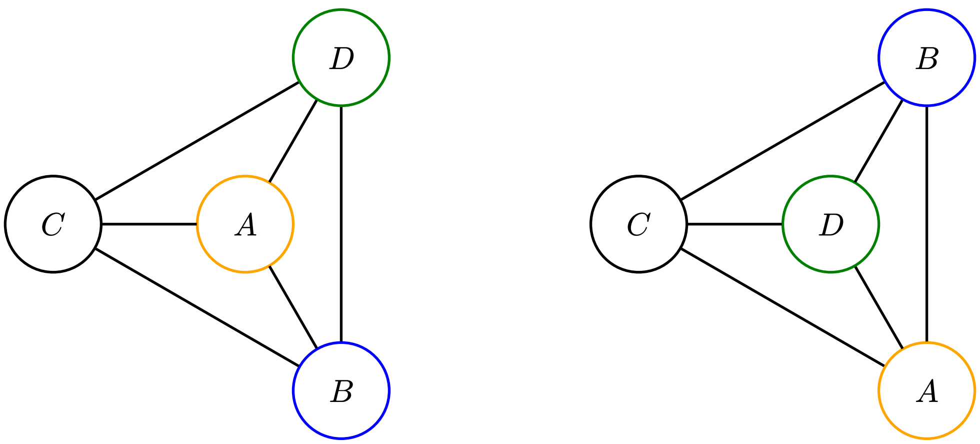


Obrázek 1: Kostka zobrazena jako neorientovany graf

Ruzne rotace kostky si muzeme predstavit jako vzajemnou zamenu tri vrcholu.



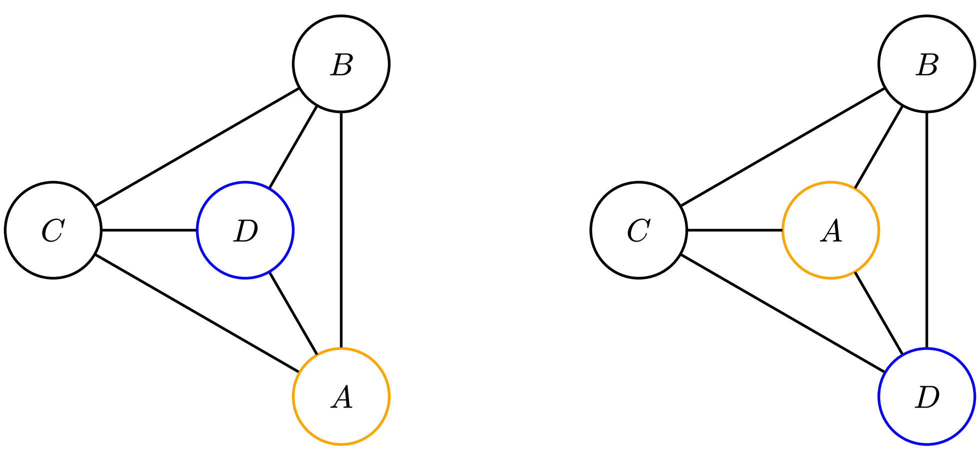
Obrázek 2: V tomto grafu se zamenili vrcholy A, C a D



Obrázek 3: V tomto grafu se zamenili vrcholy A, B a D

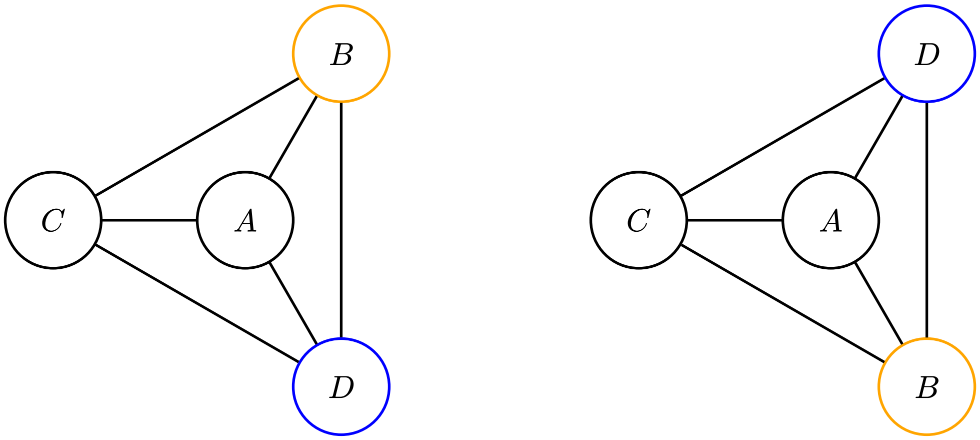
Tyto grafy reprezentuji jednu a tu samou kostku.

Kdybychom vsak zamenili vrcholy mezi sebou pouze dva, vznikne nam kostka, ktera neni identicka s kostkou predchozi.



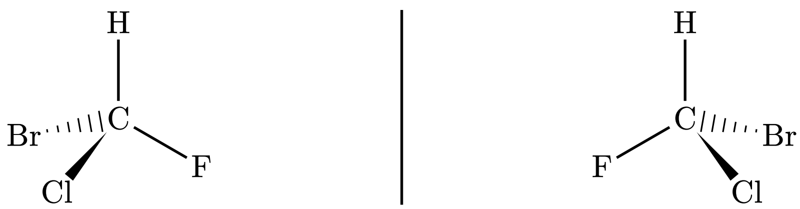
Obrázek 4: V grafu se zamenili vrchly A a D

Opetovnou zamenou libovolnych dvou vrcholu dostaneme zpet puvodni kostku.



Obrázek 5: V grafu se zamenili vrchly B a D

Pricipialne funguje toto „zrcadleni“ jako jev v chemii zvany chiralita. Dve slouceniny jsou vzajemne chiralni, jestlize jsou k sobe zrcadlove otocene. Obsahuji stejne prvky i jejich pocty, presto ale nejsou shodne.



Obrázek 6: Bromchlorfluormethan (CHBrClF)

Proto bychom meli clen odpovidajici vybranim ctyr ruznych cisel vynasobit dvema.

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

3. Sestaveni algoritmu

Cilem je maximalizovat p tak, aby zaroven platilo, ze:

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

pricemz hodnoty pravdepodobnosti $P(X > Y)$ manipulujeme vybiranim cisel sten kostek z mnoziny $[1, 6]$ (s moznosti opakovani).

Algoritmus jsem zprvu psal v Pythonu pro jeho citelnou syntaxi. Beh algoritmu ale trval delsi dobu nez se mi libilo. Rozhodl jsem se jej prepsat v programovacim jazyce Rust.

Zakladni idea je jednoducha:

1. Vygeneruje vsechny ciselne kombinace pro steny kostek. K vygenerovani kombinaci s opakovanim jsem vyuzil funkci `combinations_with_replacement` z knihovny `itertools` ve funkci `generate_dice`.

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

2. Vypocte probabilitu $P(X > Y)$ pro kazdou dvojici kombinaci. Pro vypocet $P(X > Y)$ jsem si napsal funkci:

```rust
// Compute P(X > Y).
fn pairwise_probability(x: &[usize], y: &[usize]) -> f64 {
    let wins: usize = x.iter()              // Get the count of events where x > y
                       .flat_map(|&xi| y.iter().map(move |&yi| xi > yi))
                       .filter(|&b| b)
                       .count();
    let omega = (x.len() * y.len()) as f64; // Size of the probability space
    wins as f64 / omega                     // Return probability
}
```

3. V cyklu proveri jestli pro p plati ze $P(B > A) \geq p, P(C > B) \geq p$ a $P(A > C) > p$.

```rust
let ba = pairwise_probability(b, a);      // P(B > A)
let cb = pairwise_probability(c, b);      // P(C > B)
let ac = pairwise_probability(a, c);      // P(A > C)
if ba >= p && cb >= p && ac >= p {        // Check them against p
  ... (do something)
}
```

4. Soucasne nalezene maximalni p ulozi a opakuje predchozi krok, dokud stale existuji konfigurace kostek, pro ktere plati podminky.

Zde je algoritmus (naivni varianta):

```rust
// Finds maximal p for which the conditions hold.
// Doesnt use any caching mechanism.
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 increment = 1.0 / (side_count * side_count) as f64; // Probability can grow
                                                            // only by this margin
    for step in 1..=16 {                      // Iterate from 1 to 16,
        let p = step as f64 * increment;      // incrementing by 1/|Omega|.
        let mut valid = false;                // No configurations for
                                              // current p 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 {
                    let ba = pairwise_probability(b, a);  // Get the probabilities
                    let cb = pairwise_probability(c, b);  // of P(B > A), P(C > B) ...
                    let ac = pairwise_probability(a, c);
                    if ba >= p && cb >= p && ac >= p {    // Check them against p
                        println!("A={:?}, B={:?}, C={:?}", a, b, c);
                        valid = true; // This p has a valid configuration.
                        break 'outer; // Jump to the 'outer tag.
                    }
                }
            }
        }
        if !valid {                                   // If for current p configuration
            println!("No valid configurations!"); // doesnt exist, then return the
            return max_p;                             // last valid p.
        }
        max_p = p;                    // If configuration exists assign to max_p.
    }
    max_p                             // By default return max_p found.
}
```

Tento algoritmus je vsak velmi neefektvni. Hodnotu probabilit mezi kostkami pocita neustale znova a znova. Pocita i ty hodnoty co uz byly vypocteny. V dusledku je tento alogitmus velmi pomaly.

Je jednoznacne, ze algoritmu chybi cache system. Vypoctene hodnoty ulozime do nejake datove struktury (kolekce). Pokud bychom implementovali cache system, algoritmus by mohl vyuzit toho, ze hodnota, ktera uz byla vypoctena, nebude zbytecne pocitana znova.

Datova struktura, ktera se pri implementaci algoritmu pro tento ukol prokazala jako nejlepsi, je obycejne dynamicke pole - v Rustu je to `Vec<T>`.

Pro naplneni tohoto pole predpoctenymi hodnotami pravdepodobnosti, jsem si napsal nasledujici funkci:

```rust
// Precomputes the probabilities.
// Dumps computations into a vector.
fn precompute_probabilities_vec(dice: &Vec<Vec<usize>>) -> Vec<Vec<f64>> {
    let size = dice.len();
    let mut cache = vec![vec![0.0; size]; size];
    for (i, a) in dice.iter().enumerate() {
        for (j, b) in dice.iter().enumerate() {
            cache[i][j] = pairwise_probability(a, b);
        }
    }
    cache
}
```

a vyuzil jej v upravene funkci na hledani maximalni hodnoty p:

```rust
// Finds maximal p for which the conditions hold.
// Uses Vec data structure as cache.
fn find_max_p_caching_vec(side_count: usize) -> f64 {
    let cache = precompute_probabilities_vec(&dice);  // <-- precomputing
    // ... (identical with the previous)
    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;
                    }
                }
            }
        }
        // ... (identical with the previous)
```

Posledne jsem droubnou upravou cyklu algoritmus paralelizoval:

```rust
// Finds maximal p for which the conditions hold.
```

```
// Uses Vec data structure as cache.
// Also the iterations are done in parallel.
fn find_max_p_parallel(side_count: usize) -> f64 {
    // ... (identical with the previous)
    let cache = precompute_probabilities_vec(&dice);

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

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

Cely zdrojovy kod nalezenete zde: `https://github.com/phatt-23/Projekt-9-DIM`

Vystup po zpusteni programu je zde:

```
Maximal p = 0.5625                                              stdout
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]
```

Tim jsem tedy zjistil, ze maximalni hodnota p je 0.5625 neboli $\frac{9}{16}$. Take jsem zjistil o jake konfigurace kostek se presne jedna. Dve z nich ([0] a [3]), odpovidaji konfiguraci v slovnim zadani (neberu v potaz jejich oznaceni).

4. 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).

4.1. 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.

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

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 stupňové posloupnosti jsou znázorněna zde:

$$\begin{aligned}(\ldots, \text{lichá}, \ldots, \text{lichá}, \ldots) &\overset{(-1)}{\Rrightarrow} (\ldots, \text{sudá}, \ldots, \text{sudá}, \ldots)\\ (\ldots, \text{sudá}, \ldots, \text{sudá}, \ldots) &\overset{(-1)}{\Rrightarrow} (\ldots, \text{lichá}, \ldots, \text{lichá}, \ldots)\\ (\ldots, \text{lichá}, \ldots, \text{sudá}, \ldots) &\overset{(-1)}{\Rrightarrow} (\ldots, \text{sudá}, \ldots, \text{lichá}, \ldots)\\ (\ldots, \text{sudá}, \ldots, \text{lichá}, \ldots) &\overset{(-1)}{\Rrightarrow} (\ldots, \text{lichá}, \ldots, \text{sudá}, \ldots).\end{aligned}$$

> ⓘ **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. □

4.2. 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.

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

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.

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

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.

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

Pro:

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

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:
>
> $$\deg_{S_1 \Delta S_2}(v) = \left(\deg_{S_1}(v) + \deg_{S_2}(v)\right) \bmod 2.$$

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. $\square$

Bibliografie

[1] „Intransitivity". [Online]. Dostupné z: https://en.wikipedia.org/wiki/Intransitivity?useskin=vector