

Diskrétní matematika

Semestrální projekt – zadání 9

Příklad	Poznámky
1	
2	

Jméno: Phat Tran Dai
Osobní číslo: TRA0163

Datum: 26. 11. 2024

1. Combinatorics

Let there be four-sided dice A, B and C and with numbers defined as:

$$A = \{1, 4, 4, 4\}$$

$$B = \{2, 2, 5, 5\}$$

$$C = \{3, 3, 3, 6\}.$$

Given two dice X and Y , when we throw them simultaneously and observe the number, we say X is better than Y if and only if X has a higher probability of having a greater number than Y . In other words X wins over Y and we denote this as $P(X > Y)$, the probability of X winning over Y .

Prove that $P(B > A)$, $P(C > B)$ and $P(A > C)$.

1.1. Proof of $P(B > A)$

1.1.1. First solution

The sample space Ω is every combination of B 's numbers with A 's numbers.

$$\Omega_{B,A} = [\{a, b\} : b \in B, a \in A]$$

$$\begin{aligned} \Omega_{B,A} = [& \{2, 1\}, \{2, 1\}, \{5, 1\}, \{5, 1\}, \\ & \{2, 4\}, \{2, 4\}, \{5, 4\}, \{5, 4\}, \\ & \{2, 4\}, \{2, 4\}, \{5, 4\}, \{5, 4\}, \\ & \{2, 4\}, \{2, 4\}, \{5, 4\}, \{5, 4\}] \end{aligned}$$

The size of the space is $|\Omega| = 16$. From the expanded Ω we see that clearly the the number of events where B wins is higher than the number of events in which it loses. That is:

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

We see that B 's chance of winning is 62.5% which is higher than 50% meaning it's better. \square

1.1.2. Second solution

Sample space is still the same Ω . If A is 1, B wins no matter what. If A is 4, B wins if 5 is thrown.

$$A = 1: B \text{ wins in all 4 cases} \rightarrow 4 \text{ events}$$

$$A = 4: B \text{ wins, if it's 5 (2 of 4 cases)} \rightarrow 2 \text{ events}$$

The reason behind the numbers:

- $P_{\Omega}(A = 1) = \frac{1}{4}$ which then means that number of events where $A = 1$ is $\frac{1}{4}|\Omega| = 4$. B wins in all of them.
- $P_{\Omega}(\neg A = 1) = \frac{3}{4}$ meaning the number of events where $\neg(A = 1)$ is $\frac{3}{4}|\Omega| = 12$ or one could say $3 \cdot 4 = 12$ (three non-1s times zipped with elements from dice B). Out of the 12 cases, in half of them A is winning and in the other half B is winning. That is there are 6 events where B winning when A is not 1.

$$P(B > A) = \frac{4 + 6}{|\Omega|} = \frac{10}{16} = 0.625.$$

$P(B > A)$ is greater than $\frac{1}{2}$, therefore B is better than A. \square

The remaining $P(C > B)$ and $P(A > C)$ can be done analogously.

1.2. Proof of $P(C > B)$

1.2.1. Solution

Dice C and B are defined as:

$$B = \{2, 2, 5, 5\}$$

$$C = \{3, 3, 3, 6\}.$$

The sample space here is:

$$\Omega_{C,B} = [\{c, b\} : c \in C, b \in B]$$

$$\begin{aligned} \Omega_{C,B} = [& \{3, 2\}, \{3, 2\}, \{3, 5\}, \{6, 5\}, \\ & \{3, 2\}, \{3, 2\}, \{3, 5\}, \{6, 5\}, \\ & \{3, 2\}, \{3, 2\}, \{3, 5\}, \{6, 5\}, \\ & \{3, 2\}, \{3, 2\}, \{3, 5\}, \{6, 5\}]. \end{aligned}$$

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:

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

If C is not 3, it is 6. In 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:

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

□

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

1.3.1. First method

The sample space here is:

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

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

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. □

1.3.2. 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. \square

1.4. Vypocet pomoci zakonu celkove pravdepodobnosti

From the solved probabilities, we can now come up with a general form for this probability. Let there be probabilistically observable objects A and B whose events can be ordered.

1.4.1. Zakon celkove pravdepodobnosti

(zdroj) Mame-li udalost E , která závisí na podmínkách, tak její pravdepodobnost 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:

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

V našem případě jsou udalostmi $X > Y$ (kostka X vyhraje nad Y). Disjunktním podmínkám C_i odpovídají výsledky hodu kostky Y , která může nabývat hodnoty y_0, y_1, \dots, y_n . Podmínky jsou disjunktní $Y = y_i$, proto platí ze:

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

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

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

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

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

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

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

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

Dosli jsme ke stejným výsledkům.

1.5. Různé rozmístění čísel

Kostky A, B a C mají různé barvy. Kolik existuje různých rozmístění čísel z množiny $[1, 6]$ (s opakováním).

Rozmístění můžeme reprezentovat jako množinu čísel, ve které připouštíme opakování. Kostka je čtyřstěnná, to znamená, že ne každá posloupnost popisuje jiné rozmístění čísel.

Způsobu jak vybrat 4krát z šesti čísel je:

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

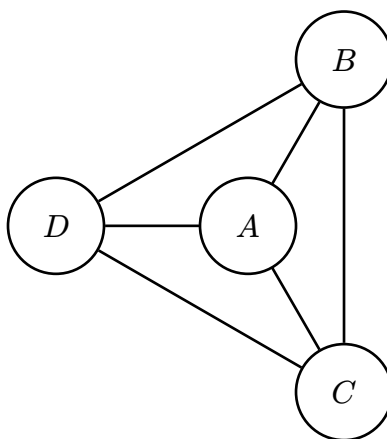
Různých prvků z šesti možností buď vybereme jeden, dva, tři nebo čtyři. Kombinaci s opakováním můžeme přepsat jako:

$$\begin{array}{ccccccc}
\binom{6}{4} & + & \binom{6}{3}\binom{3}{1} & + & \binom{6}{2}\binom{3}{1} & + & \binom{6}{1} = 126 \\
\hline
4 \text{ různá čísla,} & & 3 \text{ různá čísla,} & & 2 \text{ různá čísla,} & & 1 \text{ číslo,} \\
\text{jednina možná} & & \text{možné konfigurace jsou} & & \text{možné distribuce jsou} & & \text{jednina možná} \\
\text{distribuce je } |(1,1,1,1)|=1 & & |(2,1,1),(1,2,1),(1,1,2)|=3 & & |(3,1),(2,2),(1,3)|=3 & & \text{distribuce je } |(4)|=1
\end{array}$$

Nesmíme však zapomenout ze pro čtyřstennou kostku platí, že máme-li 4 různá čísla, tak je jsme na kostku schopni umístit dvěma různými způsoby.

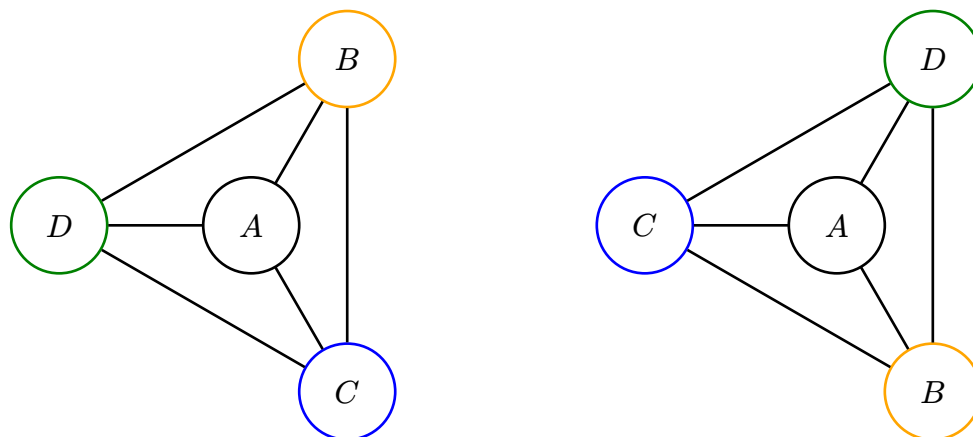
Pokud se vybere stejné číslo čtyřikrát, tak tyto čísla můžeme nanést na čtyřstennou kostku jedním jediným způsobem. Pokud se vyberou dvě různá čísla, tak je na čtyřstennou kostku můžeme nanést jedním způsobem. Rovněž, vyberou-li se tři čísla, tak tyto čísla jsou možné na kostku nanést jedním způsobem.

Mějme čtyři různá čísla A, B, C a D . Čtyřstennou kostku a ni nanesené čísla znázorníme grafem. Hrany jsou přechody mezi stěnami nižší hodnoty do vyšší. Jedna ze dvou možných konfigurací je:

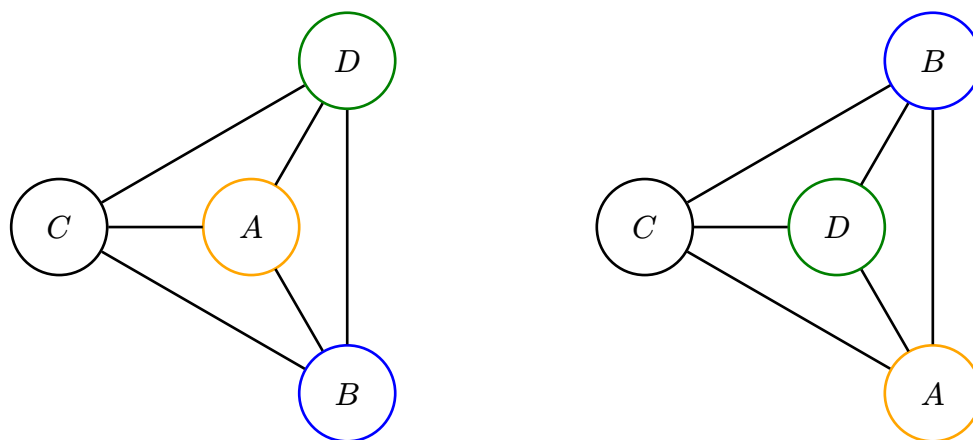


Obrázek 1: Kostka zobrazena jako neorientovaný graf

Různé rotace kostky si můžeme představit jako vzájemnou záměnu tří vrcholů.



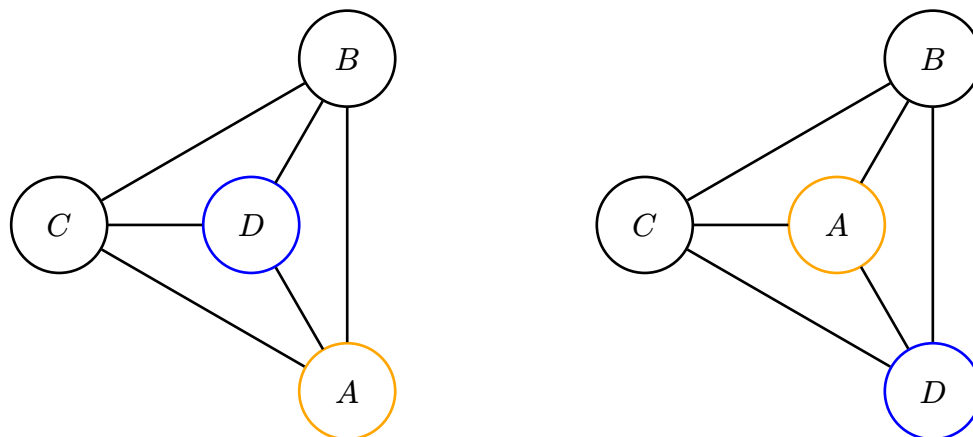
Obrázek 2: V tomto grafu se zamenili vrcholy A, C a D



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

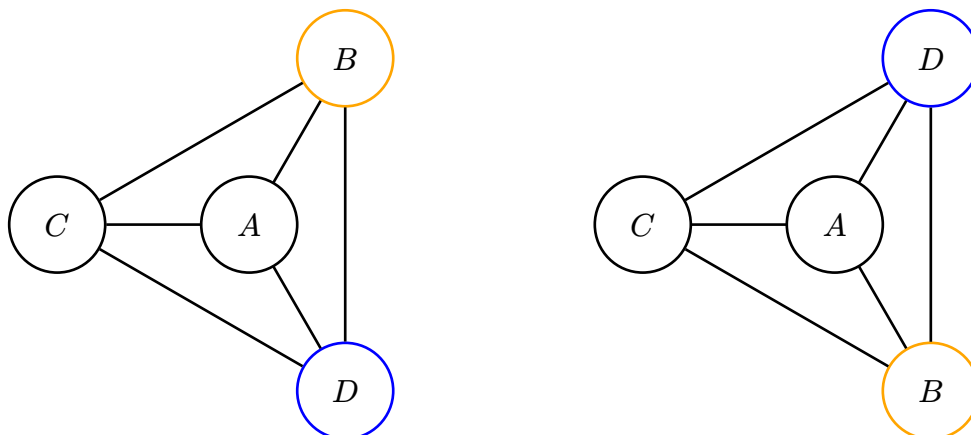
Tyto grafy reprezentují jednu a tu samou kostku.

Kdybychom však zamenili vrcholy mezi sebou pouze dva, vznikne nam kostka, která není identická s kostkou předchozí.



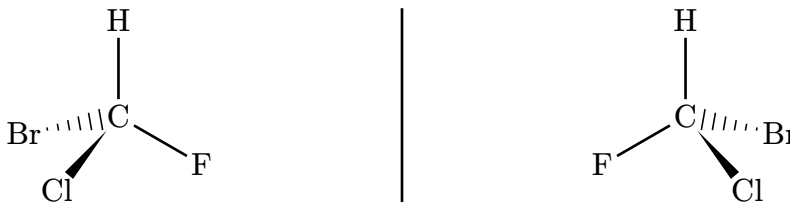
Obrázek 4: V grafu se zamenili vrcholy A a D

Opetovnou zamenou libovolných dvou vrcholu dostaneme zpet puvodni kostku.



Obrázek 5: V grafu se zamenili vrchly B a D

Principiálne funguje toto „zrcadlení“ jako jev v chemii zvaný chiralita. Dvě sloučeniny jsou vzájemně chirální, jestliže jsou k sobě zrcadlově otočeny. Obsahují stejné prvky i jejich počty, přesto ale nejsou shodné.



Obrázek 6: Bromchlorfluormethan (CHBrClF)

Proto bychom měli člen odpovídající vybraním čtyř různých čísel vynásobit dvěma.

$$\begin{aligned}
 & 2 \cdot \binom{6}{4} + \binom{6}{3} \binom{3}{1} + \binom{6}{2} \binom{3}{1} + \binom{6}{1} = \\
 & \quad \underbrace{\hspace{1.5cm}}_{\text{korekce}} \\
 & = 2 \cdot 15 + 20 \cdot 3 + 15 \cdot 3 + 6 = \\
 & = 30 + 60 + 45 + 6 = \underline{\underline{141}}
 \end{aligned}$$

2. Sestavení algoritmu

Cílem je maximalizovat p tak, aby zároveň platilo, že:

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

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

Algoritmus jsem zprvu psal v Pythonu pro jeho čitelnou syntaxi. Beh algoritmu ale trval delší dobu než se mi líbilo. Rozhodl jsem se jej přepsat v programovacím jazyce Rust.

Základní idea je jednoduchá:

1. Vygeneruje všechny číselné kombinace pro stěny kostek. K vygenerování kombinací s opakováním jsem využil funkci `combinations_with_replacement` z knihovny `itertools` ve funkci `generate_dice`.

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

Rust

2. Vypočte pravděpodobnost $P(X > Y)$ pro každou dvojici kombinací. Pro výpočet $P(X > Y)$ jsem si napsal funkci:

```
1 // Compute P(X > Y).
2 fn pairwise_probability(x: &[usize], y: &[usize]) -> f64 {
3     let wins: usize = x.iter() // Get the count of events where x > y
4         .flat_map(|&xi| y.iter().map(move |&yi| xi > yi))
5         .filter(|&b| b)
6         .count();
7     let omega = (x.len() * y.len()) as f64; // Size of the probability space
8     wins as f64 / omega // Return probability
9 }
```

Rust

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

```
1 let ba = pairwise_probability(b, a); // P(B > A)
2 let cb = pairwise_probability(c, b); // P(C > B)
3 let ac = pairwise_probability(a, c); // P(A > C)
4 if ba >= p && cb >= p && ac >= p { // Check them against p
5     ... (do something)
6 }
```

Rust

4. Současně nalezené maximální p uloží a opakuje předchozí krok, dokud stále existují konfigurace kostek, pro které platí podmínky.

Zde je algoritmus (naivní varianta):

```

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

```

Tento algoritmus je však velmi neefektivní. Hodnotu pravděpodobnosti mezi kostkami počítá neustále znovu a znovu. Počítá i ty hodnoty, co už byly vypočteny. V důsledku je tento algoritmus velmi pomalý.

Je jednoznačné, že algoritmu chybí cache systém. Vypočtené hodnoty uložíme do nějaké datové struktury (kolekce). Pokud bychom implementovali cache systém, algoritmus by mohl využít toho, že hodnota, která už byla vypočtena, nebude zbytečně počítána znovu.

Datova struktura, která se při implementaci algoritmu pro tento úkol prokazala jako nejlepší, je obyčejné dynamické pole - v Rustu je to `Vec<T>`.

Pro naplnění tohoto pole předpočtenými hodnotami pravděpodobnosti, jsem si napsal následující funkci:

```
1 // Precomputes the probabilities.
2 // Dumps computations into a vector.
3 fn precompute_probabilities_vec(dice: &Vec<Vec<usize>>) -> Vec<Vec<f64>> {
4     let size = dice.len();
5     let mut cache = vec![vec![0.0; size]; size];
6     for (i, a) in dice.iter().enumerate() {
7         for (j, b) in dice.iter().enumerate() {
8             cache[i][j] = pairwise_probability(a, b);
9         }
10    }
11    cache
12 }
```

a využil jej v upravené funkci na hledání maximální hodnoty p :

```
1 // Finds maximal p for which the conditions hold.
2 // Uses Vec data structure as cache.
3 fn find_max_p_caching_vec(side_count: usize) -> f64 {
4     let cache = precompute_probabilities_vec(&dice); // <-- precomputing
5     // ... (identical with the previous)
6     for step in 1..16 {
7         let p = step as f64 * increment;
8         let mut valid = false;
9         'outer: 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     } // ... (identical with the previous)
```

Posledně jsem drobnou úpravou cyklu algoritmus paralelizoval:

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

```

2 // Uses Vec data structure as cache.
3 // Also the iterations are done in parallel.
4 fn find_max_p_parallel(side_count: usize) -> f64 {
5     // ... (identical with the previous)
6     let cache = precompute_probabilities_vec(&dice);
7
8     for step in 1..=16 {
9         let p = step as f64 * increment;
10        println!("Testing for p = {}", p);
11
12        let valid = dice.par_iter().enumerate().any(|(i, _)| {
13            dice.par_iter().enumerate().any(|(j, _)| {
14                if cache[j][i] < p {
15                    return false;
16                }
17                dice.par_iter().enumerate().any(|(k, _)| {
18                    cache[k][j] >= p && cache[i][k] > p
19                })
20            })
21        });
22        // ... (identical with the previous)
23    }
24    max_p
25 }

```

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).

3. 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).

3.1. 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 posloupnost s lichým počtem lichých stupní neexistuje. Mějme libovolný graf G sudého řádu.

$$G = (V, E) \quad |V| = n = 2k, k \in \mathbb{Z}$$

$$D_G = (d_1, d_2, \dots, d_n)$$

$$k_e = |\{d \in D : d \equiv 0 \pmod{2}\}|$$

$$k_o = |\{d \in D : d \equiv 1 \pmod{2}\}|$$

$$k_o \equiv 1 \pmod{2} \Leftrightarrow k_e \equiv 1 \pmod{2}$$

$$\sum_{v \in V} \deg(v) = k_e \cdot \text{sudá} + k_o \cdot \text{lichá} = \text{sudá} + \text{lichá} = \underline{\text{lichá}}$$

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

Pro hledání lichého faktoru F stromu T si pomůžeme 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 konkrétními stromy.

Pokud se tato posloupnost skládá z lichých čísel, tak 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:

$$\begin{aligned}
 (... , \text{lichá}, ... , \text{lichá}, ...) &\stackrel{(-1)}{\Rightarrow} (... , \text{sudá}, ... , \text{sudá}, ...) \\
 (... , \text{sudá}, ... , \text{sudá}, ...) &\stackrel{(-1)}{\Rightarrow} (... , \text{lichá}, ... , \text{lichá}, ...) \\
 (... , \text{lichá}, ... , \text{sudá}, ...) &\stackrel{(-1)}{\Rightarrow} (... , \text{sudá}, ... , \text{lichá}, ...) \\
 (... , \text{sudá}, ... , \text{lichá}, ...) &\stackrel{(-1)}{\Rightarrow} (... , \text{lichá}, ... , \text{sudá}, ...).
 \end{aligned}$$

❶ 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 snižování stupňů v posloupnosti, je zaručeno, že jedním z těchto výsledných posloupností je právě ona posloupnost faktoru F grafu G .

Toto postupné snižování sekvence a výsledné 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 . □

3.2. 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.

Předpokládejme, 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	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. Pokud je obsažena pouze v jednom, tak se s ní počítá. Pro vrchol $v \in V(F_1 \Delta F_2)$ jeho stupeň $\deg_{F_1 \Delta F_2}(v)$ je rozhodnuta počtem všech incidentních hran e , které náležejí právě jednomu z faktorů. Jestliže se e objevuje právě v jednom faktoru, kontribuuje stupni vrcholu v jednou (+1). Pokud se objevuje v obou faktorech, tak stupni nekontribuuje. Důležité však je, že nás zajímá pouze sudost nebo lichost tohoto stupně.

O faktorech F_1 a F_2 víme, že jsou jejich vrcholové stupně jsou všechny liché.

$$\forall v \in F_1, \deg(v) \equiv 1 \pmod{2} \quad \forall v \in F_2, \deg(v) \equiv 1 \pmod{2}$$

Je zřejmé, že je-li $F_1 \Delta F_2$ nulový graf (graf bez hran), tak jsou faktory F_1 a F_2 stejné.

$$F_1 \Delta F_2 = \emptyset \Leftrightarrow F_1 = F_2$$

Jak to bude vypadat se sudostí a lichostí vrcholů symetrické difference $F_1 \Delta F_2$?

Vyberme si libovolný vrchol v z grafu T .

Pokud se žádné hrany incidentní s v v F_1 (označme $E_{F_1}^v$ jako množinu hran grafu F_1 incidentních s vrcholem v) neobjevují v F_2 , potom všechny hrany v $E_{F_1}^v$ zahrneme do $F_1 \Delta F_2$, 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:

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

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

$F_2 = (V_T, E_{F_2} \subseteq E_T)$, 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 libovolného 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 spadal 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