

# Kombinatoryczna Teoria Liczb

Teoretyczny opis problemu Anty-Van der Waerden'a

Sylwia Nowak

Radosław Kutkowski

Roman Siry

Anty-Van der Waerden

Rozmiar zbioru liczb (n)

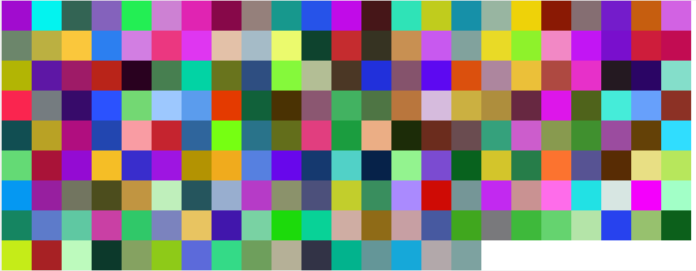
300

Długość tęczy wygrywająca (k)

30

Liczba dostępnych kolorów (c)

200



Poziom trudności gracza

0

1

2

3

4

Poziom zaawansowania strategii komputera

0

1

2

3

4

START

# 1 Cel projektu

Celem projektu jest implementacja gry Anty-Van der Waerden'a, w której gra komputer kontra człowiek.

**Dane wejściowe:** liczba naturalna  $n$ , liczba naturalna  $k$ , liczba dostępnych kolorów  $c$ . Gracze na zmianę wybierają liczbę ze zbioru  $[n]$  i kolorują ją na wybrany kolor. **Zwycięstwo:**

- komputer wygrywa, jeśli w otrzymanym pokolorowaniu zbioru  $[n]$  będzie tęczowy podciąg arytmetyczny o długości  $k$  (każdy wyraz tego ciągu musi mieć inny kolor)  
 $(\exists S \subset [n])(\exists d \in N)(\forall s_1, s_2 \in S, s_1 \neq s_2)c(s_1) \neq c(s_2)$
- człowiek wygrywa, jeśli nie będzie takiego podciągu

# 2 Liczba Anty-Van der Waerden'a

**Definicja 2.1.** Kolorowanie  
 $r$ -kolorowanie zbioru  $S$  to funkcja  $c : S \rightarrow C$  taka, że  $|C| = r$ .

**Definicja 2.2.** Tęczowe kolorowanie  
Zbiór  $S$  jest tęczowy jeżeli  $c(s_1) \neq c(s_2)$ , dla każdej różnej pary  $s_1, s_2 \in S$ .

**Definicja 2.3.** Liczba Anty-Van der Waerden'a  
Liczba anty-van der Waerden'a nazwiemy najmniejszą dodatnią liczbę całkowitą  $r = aw(n, k)$  taką, że dla dowolnych liczb całkowitych  $n, k$  takich, że  $k \leq n$ , dowolne  $r$ -kolorowanie  $n$  liczb zawiera tęczowy podciąg arytmetyczny długości  $k$ .

# 3 Sprawdzanie warunku zakończenia gry

## 3.1 Sprawdzanie sensu rozgrywania gry

- Jeżeli liczba kolorów  $c$  jest mniejsza od  $k$ , nie ma możliwości aby zwycięzcą rozgrywki był komputer.
- Jeżeli  $aw(n, k) \leq 2 * \lceil \frac{n}{2} \rceil$  i  $aw(n, k) \leq c$  to komputer zawsze zwycięża grę o ile użyje przynajmniej  $aw(n, k)$  różnych kolorów

## 3.2 Regularna rozgrywka

Sprawdzenie czy gra nie została zakończona poprzez zwycięstwo komputera jest wykonywane w dwóch przypadkach:

- po ruchu komputera
- po wykonaniu ruchu gracza

W każdym ruchu pamiętamy największy możliwy do tej pory ciąg arytmetyczny tęczowy i na podstawie tego możemy dokonać optymalizacji czasu przeszukiwania planszy.

### 3.3 Problem znajdowania najdłuższego ciągu arytmetycznego tęczowego

Wykorzystywana struktura do przechowywania informacji o ciągu arytmetycznym długości  $k$ , będącym tęczą to *serie* zawierająca w sobie 2 pola

- `weight` - waga ciągu, będąca liczbą jeszcze niepokolorowanych pól w ciągu
- `fields` - lista pól zawartych w ciągu, każde pole posiada informację o numerze i kolorze (*number*, *color*)

Podczas trwania rozgrywki przechowywana jest informacja o wszystkich ciągach arytmetycznych znajdujących się na planszy, takich, że mogą one zostać ciągami tęczowymi. Dane te przechowywane są w kolejce priorytetowej oznaczanej *PQ*. W danej kolekcji priorytetem jest waga ciągu, elementy z mniejszą wagą znajdują się przed elementami z większą. Przed wykonaniem pierwszego ruchu kolekcja *PQ* jest pusta. Wypełniana jest ona wszystkimi możliwymi ciągami arytmetycznymi długości  $k$ . Do zapewniania kolejki stosowany jest algorytm *Algorytm tworzenia kolejki PQ*.

---

#### Algorithm 1 Algorytm tworzenia kolejki *PQ*

---

```

PQ-kolejka priorytetowa (ciąg, priorytet)
for  $i=1 \dots n-k+1$  do
     $r=1$ 
    while  $r \leq n/(k-1) \wedge i+r*(k-1) \leq n$  do
        PQ.Add( $i+r*0, \dots, i+r*(k-1)$ ,  $k$ );
    end while
end for

```

---

Po wykonaniu każdego ruchu z graczy kolejka jest aktualizowana. Ciągi, które nie mogą już być tęczowe są usuwane z kolekcji. Jeżeli dany ciąg jest edytowany to jego priorytet w kolejce aktualizuje się.

## 4 Strategie gry

W przygotowanej aplikacji istnieje możliwość wybrania poziomu trudności gry. Istnieje 5 stopniów zaawansowania strategii gry komputera oraz 5 stopni zaawansowania odpowiedzi dostępnych w każdym ruchu dla gracza. Przygotowane strategie gry w dużej mierze wykorzystują algorytm znajdowania najdłuższego ciągu arytmetycznego istniejącego na planszy i będącego tęczą tzw. *tęczowy ciąg*. Przedstawione strategie zostały podzielone w taki sposób, że zaawansowanie strategii rośnie wraz z jej przyporządkowanym numerem.

## 4.1 Strategie gry komputera

Po każdym ruchu człowieka zgodnie z wybraną strategią komputera następuje wybór pola i koloru, które zostaną wykorzystane w ruchu komputera.

1. Losowy wybór koloru oraz pola.  
kolor - losowo wybrany kolor z dostępnej puli  
pole - losowo wybrana liczba  $i$  z przedziału  $[1, n]$ , dla której pole  $p_i$  jest jeszcze niepokolorowane  
Podczas losowania danych wartości nie jest wykorzystywana kolejka priorytetowa  $PQ$
2. Losowy wybór dowolnego ciągu znajdującego się w kolejce priorytetowej  $PQ$  oznaczonego jako  $a_i$ . Wybranie dowolnego niepokolorowanego jeszcze pola w ciągu  $a_i$  oraz dowolnego koloru z dostępnej puli kolorów jeszcze nie użytych w wylosowanym ciągu  $a_i$ .
3. Losowy wybór dowolnego ciągu znajdującego się w kolejce priorytetowej  $PQ$  - oznaczonego jako  $a_i$ . Prawdopodobieństwo wylosowania ciągu  $a_i$  wynosi  $1 - (suma_{x_i} / \sum_{p=1}^{size(PQ)} suma_{x_i})$ , gdzie  $suma_{x_i}$  to ilość pól niepokolorowanych w ciągu  $a_i$ . Wybranie dowolnego niepokolorowanego jeszcze pola w wylosowanym ciągu  $a_i$  oraz dowolnego koloru z dostępnej puli kolorów jeszcze nie użytych w ciągu  $a_i$ .
4. Z kolejki priorytetowej  $PQ$  wybierany jest element z maksymalnym priorytetem. Z wybranego ciągu  $a_i$  losowane jest jeszcze niepokolorowane pole oraz kolor nie występujący do tej pory w ciągu  $a_i$ .
5. Z kolejki priorytetowej  $PQ$  wybierany jest zbiór elementów oznaczony  $A$  o największym priorytecie. Tworzona jest lista par (kolor, waga koloru) oznaczona jako *ColorList*, zawierająca wszystkie kolory  $k$ , które można by było użyć w dowolnym ciągu  $a_i \in A$ , nie psując przy tym warunku bycia tęczą ciągu  $a_i$ . Dla każdego koloru waga koloru wyliczana jest na podstawie występowalności we wszystkich ciągach  $a_i \in PQ$ . Jeżeli ciąg  $a_i$  wystąpił 5 razy w całej kolejce priorytetowej  $PQ$  to  $wagaKoloru_i = 5$ . Elementy w liście *ColorList* posortowane są od najmniejszej wartości wagi koloru do największej. Kolory występujące w liście *ColorList* są unikalne. Dodatkowo tworzona jest lista par (numer pola, waga pola) oznaczona jako *FieldList*, zawierająca wszystkie numery pól niepokolorowanych występujących w ciągach  $a_i \in A$ . Dla każdego pola waga pola wyliczana jest na podstawie występowalności we wszystkich ciągach  $a_i \in PQ$ . Jeżeli pole o numerze  $p$  występuje 10 razy w kolejce priorytetowej  $PQ$  to  $wagaPola_p = 10$ . Elementy w liście *FieldList* posortowane są od największej wartości wagi pola do najmniejszej. Następuje iteracja po każdym ciągu  $a_i \in A$ . Dla każdego ciągu  $a_i$  ustawiana jest tzw. tymczasowa waga  $tmp\_weight_i$ , wyliczana na podstawie  $tmp\_weight_i = \text{minus waga pierwszego koloru z listy } ColorList \text{ nie wykorzystanego w ciągu } a_i \text{ (oznaczonego jako } w_{c_j}) + \text{waga pierwszego pola z listy } FieldList \text{ (oznaczonego jako } w_{c_j})$

$w_{f_l}$ ) znajdującego się w ciągu  $a_i$  a nie będącego jeszcze pokolorowanym. Wybierany jest taki ciąg  $a_i$ , dla którego waga  $tmp\_weight_i$  jest największa. Wówczas w swoim ruchu komputer wybiera pole o numerze  $l$  oraz  $j$ -ty kolor.

---

**Algorithm 2** Algorytm generowania listy ColorList

---

$c$  - liczba wszystkich kolorów dostępnych przy kolorowaniu, oznaczamy  $c_i$  jako  $i$ -ty kolor,  $i \in [1, c]$   
ColorDictionary to słownik zawierający jako klucz numer koloru a jako wartość jego wagę  
ColorList = {}  
ColorDictionary = {}  
**for all**  $a \in PQ$  **do**  
    **for**  $f \in a.fields$  **do**  
        **if**  $not(ColorDictionary.containsKey(f.color))$  **then**  
            ColorDictionary.put( $f.color, 0$ )  
        **end if**  
        ColorDictionary.put( $f.color, ColorDictionary.get(f.color) + 1$ )  
    **end for**  
**end for**  
ColorList = ColorDictionary.values  
ColorList.sortAscending() **return** ColorList

---

Złożoność algorytmu *Algorytm generowania listy ColorList*  $|size(PQ) * k|$

---

**Algorithm 3** Algorytm generowania listy FieldList

---

FieldDictionary to słownik zawierający jako klucz numer pola a jako wartość jego wagę  
FiledList = {}  
FiledDictionary = {}  
**for all**  $a \in PQ$  **do**  
    **for**  $f \in a.fields$  **do**  
        **if**  $not(FieldDictionary.containsKey(f.number))$  **then**  
            FiledDictionary.put( $f.number, 0$ )  
        **end if**  
        FiledDictionary.put( $f.number, FiledDictionary.get(f.number) + 1$ )  
    **end for**  
**end for**  
FiledList = FieldDictionary.values  
FiledList.sortDescending() **return** FiledList

---

Złożoność algorytmu *Algorytm generowania listy FieldList*  $|size(PQ) * k|$

---

**Algorithm 4** Algorytm wyznaczania pola i koloru na podstawie zbioru  $A$ 

---

SeriesWeightsList zawierający w sobie jako pary ciągów  $a \in A$  i odpowiadających im wag  $tmp\_weight$ .

MovementDictionary zawierający w sobie jako klucze ciągi  $a \in A$  oraz jako wartości odpowiadające im pary (numer pola, numer koloru) będące potencjalnym ruchem komputera.

```
for all  $a \in A$  do
    selectedFieldPair = null
    selectedColorPair = null
    tmpWeight = -INF;
    for  $f \in a.fields$  do
        if  $f.color == null$  then
            for  $pair \in FiledList$  do
                if  $pair.first == f.number$  then
                    if  $pair.second > selectedFieldPair.second$  then
                        selectedFieldPair = pair
                    end if
                break
            end if
        end for
    end if
    for  $pair \in ColorList$  do
        if  $pair.first == f.color$  then
            if  $pair.second < selectedColorPair.second$  then
                selectedColorPair = pair
            end if
        break
    end if
    end for
    tmpWeight = -selectedFieldPair.second
    tmpWeight = tmpWeight + selectedColorPair.second
    MovementDictionary.put( $a$ , pair(selectedField, selectedColor))
    SeriesWeightsDictionary.put( $a$ , tmpWeight)
end for
tmpa = null
for all  $a \in SeriesWeightsDictionary.keys$  do
    if  $tmpa == null$  then
        tmpa =  $a$ 
        continue
    end if
    if  $SeriesWeightsDictionary.get(a) > SeriesWeightsDictionary.get(tmpa)$ 
then
        tmpa =  $a$ 
    end if
end for
return MovementDictionary.get( $a$ )
```

---

Złożoność algorytmu *Algorytm wyznaczania pola i koloru na podstawie zbioru A* to  $|size(A) * k * (size(FiledList) + size(ColorList))|$

## 4.2 Strategie gry dla gracza - Wskazówki

Po każdym ruchu komputera zgodnie z wybranym stopniem zaawansowania wskazówek istnieje możliwość podejrzenia podpowiadanego koloru i pola, które sugerowane są graczowi jako najlepsze w danym ruchu. Wskazówkę można podejrzyć wiele razy, bez ograniczenia na liczbę wyświetleń. Za każdym razem możliwa jest sytuacja, że wyświetli się inna podpowiedź, pomimo braku zmiany stanu gry. Taka sytuacja wystąpi w przypadku wybrania strategii korzystającej z pseudo losowego gneratora.

1. Brak wskazówek dla gracza.
2. Losowy wybór dowolnego ciągu znajdującego się w kolejce priorytetowej  $PQ$  oznaczonego jako  $a_i$ . Wybranie dowolnego niepokolorowanego jeszcze pola w ciągu  $a_i$  oraz dowolnego koloru z dostępnej puli kolorów już użytych w wylosowanym ciągu  $a_i$ . Jeżeli takiego koloru nie ma, losowany jest dowolny kolor ze wszystkich kolorów dostępnych podczas gry.
3. Losowy wybór dowolnego ciągu znajdującego się w kolejce priorytetowej  $PQ$  - oznaczonego jako  $a_i$ . Prawdopodobieństwo wylosowania ciągu  $a_i$  wynosi  $suma_{x_i} / \sum_{p=1}^{size(PQ)} suma_{x_i}$ , gdzie  $suma_{x_i}$  to liczba pól niepokolorowanych w ciągu  $a_i$ . Wybranie dowolnego niepokolorowanego jeszcze pola w wylosowanym ciągu  $a_i$  oraz dowolnego koloru z dostępnej puli kolorów już użytych w ciągu  $a_i$ . Jeżeli takiego koloru nie ma, losowany jest dowolny kolor ze wszystkich kolorów dostępnych podczas gry.
4. Z kolejki priorytetowej  $PQ$  wybierany jest element z maksymalnym priorytetem. Z wybranego ciągu  $a_i$  losowane jest jeszcze niepokolorowane pole oraz kolor występujący już w ciągu  $a_i$ . Jeżeli takiego koloru nie ma, losowany jest dowolny kolor ze wszystkich kolorów dostępnych podczas gry.
5. Z kolejki priorytetowej  $PQ$  wybierany jest zbiór elementów oznaczony  $A$  o największym priorytecie. Tworzona jest lista par (kolor, waga koloru) oznaczona jako  $ColorList$ , zawierająca wszystkie kolory  $k$ , które można by było użyć w dowolnym ciągu  $a_i \in A$ , psując przy tym warunek bycia tęczą ciągu  $a_i$ . Dla każdego koloru waga koloru wyliczana jest na podstawie występowalności we wszystkich ciągach  $a_i \in PQ$ . Jeżeli ciąg  $a_i$  wystąpił 5 razy w całej kolejce priorytetowej  $PQ$  to  $wagakoloru_i = 5$ . Elementy w liście  $ColorList$  posortowane są od największej wartości wagi koloru do najmniejszej. Kolory występujące w liście  $ColorList$  są unikalne. Dodatkowo tworzona jest lista par (numer pola, waga pola) oznaczona jako  $FieldList$ , zawierająca wszystkie numery pól niepokolorowanych występujących w ciągach  $a_i \in A$ . Dla każdego pola waga pola wyliczana jest na podstawie występowalności we wszystkich ciągach  $a_i \in PQ$ . Jeżeli pole o numerze  $p$

występuje 10 razy w kolejce priorytetowej  $PQ$  to  $wagaPola_p = 10$ . Elementy w liście *FiledList* posortowane są od największej wartości wagi pola do najmniejszej. Następuje iteracja po każdym ciągu  $a_i \in A$ . Dla każdego ciągu  $a_i$  ustawiana jest tzw. tymczasowa waga  $tmp\_weight_i$ , wyliczana na podstawie  $tmp\_weight_i = \text{waga pierwszego koloru z listy } ColorList \text{ nie wykorzystanego w ciągu } a_i \text{ (oznaczonego jako } w_{cj}) \text{ plus waga pierwszego pola z listy } FieldList \text{ (oznaczonego jako } w_{fl}) \text{ znajdującego się w ciągu } a_i \text{ a nie będącego jeszcze pokolorowanym}$ . Wybierany jest taki ciąg  $a_i$ , dla którego waga  $tmp\_weight_i$  jest największa. Wówczas w swoim ruchu komputer wybiera pole o numerze  $l$  oraz  $j$ -ty kolor.

Algorytmy wykorzystane są analogiczne do przedstawionych dla strategii komputera.

## 5 Bibliografia

1. <http://orion.math.iastate.edu/dstolee/presentations/Young-2014-02-Boca.pdf>
2. <https://computationalcombinatorics.wordpress.com/2014/04/30/rainbow-arithmetic-progressions-i-search-algorithm/#more-572>
3. [http://ikrech.up.krakow.pl/h\\_wykl/mat-dod/Naiwna%20metoda%20probabilistyczna%20-%20Liczby%20van%20der%20Waerdena1.pdf](http://ikrech.up.krakow.pl/h_wykl/mat-dod/Naiwna%20metoda%20probabilistyczna%20-%20Liczby%20van%20der%20Waerdena1.pdf)