Gry kombinatoryczne 2022
Gra Szemeredi'ego
Dokumentacja końcowa

Aleksander Malinowski
Jakub Piwko
Malwina Wojewoda

maj 2022

1 Treść gry

Dane wejściowe:
- liczba naturalna k
- liczba naturalna x

Na początku rozgrywki komputer losuje zbiór x liczb naturalnych X. Każdy gracz ma swój własny kolor. Ruch polega na wybraniu niepokolorowanej liczby ze zbioru X i pokolorowaniu jej swoim kolorem. Gracze ścigają się kto pierwszy ułoży monochromatyczny ciąg arytmetyczny o zadanej długości k. Gra człowiek kontra komputer.

2 Instrukcja obsługi

- Początkowo program poprosi o podanie liczby naturalnej z przedziału 3-1000, która jest wielkością zbioru, który zostanie wylosowany. Należy podać liczbę oraz kliknąć Enter.

- Następnie, po poproszeniu przez program, należy podać liczbę naturalną z przedziału 2-10, która określa jak długi ciągu arytmetyczny ma zostać pokolorowany.

- Następnie użytkownik ma możliwość wyboru kolejności rozgrywki. Program prosi o podanie liczby 0 lub 1. Podanie 0 oznacza, że grę rozpocznie komputer, natomiast 1 oznacza pierwszy ruch człowieka.

- Następuje pierwszy ruch. W przypadku, gdy ruch należy do gracza, wyświetlany jest zbiór możliwych liczb do wyboru. Aby wykonać ruch, wystarczy wpisać odpowiednią liczbę.

- Gra przebiega sekwencyjnie, komputer wykonuje następny ruch, następnie należy znowu wybrać liczbę z ciągu. Gra przebiega do momentu zwycięstwa jednego z graczy lub remisu.

3 Opis gry

Początkowo gracz wybiera liczność zbioru X oraz długość szukanego ciągu k. W związku z dużą złożonością obliczeniową, nawet po zastosowanej optymalizacji, musieliśmy ostrzec o ograniczenia co do wielkości pierwotnego zbioru (do 1000) oraz długości ciągu (do 10), których przekroczenie może skutkować zawieszeniem się programu. Dodatkowo uznaliśmy, że podana wartość k ma by większa niż 2, gdyż zbudowanie ciągu długości 2 następowałoby już w drugim ruchu, bez względu na wybór elementów. Ciąg X generowany jest przy pomocy metody **SetSampling.chooseRandomSet**, która tworzy x-elementowy ciąg z ze zbioru $\{1, 2, \ldots, 5x-1, 5x\}$, posortowany od najmniejszej do największej liczby. W tak wylosowanym zbiorze znajdują się wyłącznie unikatowe liczby. Dodatkowo, metoda **generateSetWithProgression** z tej samej klasy odpowiada za to, aby w wylosowanym ciągu był przynajmniej jeden ciąg arytmetyczny długości k, co jest warunkiem koniecznym rozpoczęcia gry. Następnie, korzystając z metody **SequenceOps.getSequences** generowane są wszystkie k-elementowe ciągi arytmetyczne, które występują w wylosowanym zbiorze X. Sposób, w jaki są one generowane został dokładniej opisany w dokumentacji teoretycznej. Następnie dla każdego z otrzymanych ciągów tworzymy obiekt **MySet**, w którym przechowywana jest postać ciągu, liczba już pokolorowanych liczb w tym ciągu oraz te pokolorowane liczby. Na początku przypisujemy zbiór obiektów **MySet** graczowi i komputerowi. Będą one zawierały wszytskie te ciągi, które dany gracz może jeszcze ułożyć. W kolejnych ruchach będą usuwane te, których gracz już nie będzie w stanie ułożyć. Stanie się to wtedy, gdy jakakolwiek liczba z sekwencji zostanie pokolorowana przez przeciwnika.

Główna rozgrywka przedstawiona jest w klasie **Main**, natomiast metody, które bezpośrednio obsługują grę znajdują się w klasie **Game**. Niezależnie od kolejności ruchów, w swoim pierwszym ruchu komputer Wybiera liczbę, która występuje w ciągach arytmetycznych największą liczbę razy (czyli można z nią utworzyć najwięcej ciągów). Odpowiada za to metoda **SequenceOps.selectFirstNumber**.

Gracze wykonują ruchy na zmianę. Przed ruchem komputera, poza pierwszym, sprawdzane jest, czy komputer ma blokować czy kolorować swój ciąg. Gdy nie musi blokować, koloruje liczby ze swojego najdłuższego ciągu. Najpierw przeszukuje wszystkie obiekty **MySet**, które mu odpowiadają, w celu znalezienia ciągu ciągu, w którym pokolorował najwięcej liczb poprzez funckję **Game.getColoredMaxSeq**. Następnie wybiera z tej sekwencji losową liczbę, która jeszcze nie została pokolorowana i to właśnie ją koloruje swoim kolorem. Komputer może szybko przejść w tryb defensywny. Stanie się to wtedy, gdy człowiek będzie miał dłuższy pokolorowany ciąg, ale tylko spośród tych, które ma jeszcze możliwość pokolorować. Maszyna szybko to sprawdza dzięki wspomnianej funkcji **Game.getColoredMaxSeq** i porównuje z długością swojego najdłuższego ciągu. Komputer będzie blokował także w sytuacji, gdy nie będzie mógł ułożyć więcej ciągów (Jego zbiór obiektów MySet będzie pusty). Aby zablokować przeciwnika, komputer odnajduje ciąg, w którym człowiek pokolorował najwięcej liczb i koloruje losową liczbę spośród jeszcze niepokolorowanych w tym ciągu. Oznacza to, że przeciwnik na pewno nie dokończy tej konkretnej sekwencji i będzie musiał kolorować inną.

Po każdym ruchu sprawdzana jest długość najdłuższego ciągu w zbiorze pokolo-

rowanym przez gracza, który właśnie wykonał ruch. Jeśli długość będzie równa lub większa od k, gracz wygrywa i gra się kończy. Wyświetlany jest komunikat, który gracz wygrał i jaki ciąg pokolorował.

Po każdym ruchu sprawdzana jest również możliwość remisu, który występuje gdy żaden z graczy nie ma już możliwości pokolorowania ciągu arytmetycznego o długości k. Może to nastąpić na skutek zostania zablokowanym przez przeciwnika. W naszej implementacji oznacza to, że zarówna zbiór obiektów *MySet* człowieka jak i komputera są puste. W tej sytuacji wyświetlany jest odpowiedni komunikat i gra się kończy.

4 Opis zmian w stosunku do poprzednich dokumentacji

Pierwszą zmianą w stosunku do poprzednich planów jest zrezygnowanie z interfejsu graficznego, ze względu na to, że kod okazał się bardziej wymagający niż podejrzewaliśmy. W trakcie musieliśmy zmierzyć się z kilkoma problemami i zdecydowaliśmy się na udoskonalenie strategii kosztem zrezygnowania z interfejsu graficznego. Pewne plany implementacyjne, które opisaliśmy w dokumentacji teoretycznej uległy zmianie między innymi po wnioskach z testowania. W dokumentacji teoretycznej planowaliśmy wybieranie x-elementowego zbioru spośród liczb od 1 do 10x. Testowanie pokazało jednak, że w takim przypadku trudniej o powstanie odpowiednio dużo ciągów zadanej długości k. Właśnie dlatego, aby gra była ciekawsza, ale jednocześnie nie zbyt prosta zdecydowaliśmy się na wybieranie liczb ze zbioru $\{1, 2, ..., 5x-1, 5x\}$. Uznaliśmy także, że dla ułatwienia graczowi analizowania liczb, wylosowany ciąg będzie posortowany od najmniejszych liczb do największych.

Wprowadziliśmy także niewielkie zmiany w sposobie wyznaczania wszystkich ciągów arytmetycznych występujących w wylosowanym zbiorze. Początkowo chcieliśmy mieć te ciągi o długości większej lub równej k, lecz w czasie pisania kodu uznaliśmy, że wygodniej będzie pracować, jeśli będziemy mieć wyłącznie ciągi długości równej k. Przykładowo, zakładaliśmy, że jeśli w zbiorze będą występowały liczby $2, 4, 6, 8, 10, 12$, a $k = 3$, to będziemy mieć je zapisane jako jeden ciąg. Finalnie jednak otrzymane z niego ciągi zapisujemy jako: $\{2, 4, 6\}, \{4, 6, 8\}, \{6, 8, 10\}$. Wynika z tego również zmiana w wyborze pierwszej liczby kolorowanej przez komputer - nie rozważamy już drugiego przypadku, czyli wybierania środkowej liczby z najdłuższego ciągu.

Postanowiliśmy także zmodyfikować klasę `MySet`. Zrezygnowaliśmy z przechowywania pola dotyczącego różnicy ciągu, ponieważ ta informacja okazała się zbędna dla przebiegu gry. Dodatkowo niepotrzebne okazały się informacje o tym, które liczby w danej sekwencji pokolorował komputer, a które człowiek. W naszej implementacji obiekt MySet odpowiadający konkretnemu ciągowi znajduje się w zbiorze należącym do gracza osobno tylko wtedy, gdy może go jeszcze pokolorować. Dzięki takiemu podejściu, wystarczy że klasa *MySet* będzie przechowywała elementy już pokolorowane przez danego gracza i ich liczbę, jako że gracze mają swoje, niemieszające się zbiory.

Zmieniło się również nasze podejście co do oceny, który z graczy jest bliższy wygranej. Na początku chcieliśmy sprawdzać zarówno długość już pokolorowanego ciągu, jak i liczbę ruchów potrzebną do wygranej. W finalnej wersji zostaliśmy jedynie

przy sprawdzaniu długości ciągu człowieka. Dokładniej, sprawdzamy czy długość najdłuższego ciągu człowieka jest dłuższa niż długość najdłuższego ciągu komputera, aby na tej podstawie oceniać czy trzeba wykonywać blokowanie, czy kontynuować budowanie swojego ciągu. Ze względów implementacyjnych, blokowanie będzie też wykonywane w przypadku, gdy komputer nie będzie mógł już ułożyć żadnego ciągu, o ile człowiek będzie miał jeszcze taką możliwość. Gdy obaj gracze zostaną pozbawieni możliwości ułożenia ciągu, następuje remis.

5 Opis testów

W ramach testowania postanowiliśmy kilkukrotnie przeprowadzić rozgrywkę, uwzględniając różne wielkości zbioru liczb oraz długości szukanych ciągów. Sprawdzaliśmy również czy graczowi bardziej opłaca się blokować komputer czy kolorować własny ciąg. W przypadku losowego zachowania człowieka, komputer zawsze wygra. Dlatego poniżej przedstawiamy przykładowe rozgrywki, w których gracz stara się przechytrzyć maszynę.

```
Ruch komputera.
Komputer wybrał element 32
Zbiór komputera:
[32]
Zbiór do gry:
[0, 4, 5, 9, 11, 15, 17, 20, 22, 44, 47, 53, 63, 65, 70, 81, 82, 85, 94]
Ruch gracza.
Twój zbiór:
[]
Wybierz liczbę ze zbioru, którą chcesz pokolorować swoim kolorem
11
Zbiór gracza:
[11]
Zbiór do gry:
[0, 4, 5, 9, 15, 17, 20, 22, 44, 47, 53, 63, 65, 70, 81, 82, 85, 94]
Ruch komputera.
Komputer wybrał element 44
Zbiór komputera:
[32, 44]
Zbiór do gry:
[0, 4, 5, 9, 15, 17, 20, 22, 47, 53, 63, 65, 70, 81, 82, 85, 94]
Ruch gracza.
Twój zbiór:
[11]
Wybierz liczbę ze zbioru, którą chcesz pokolorować swoim kolorem
20
Zbiór gracza:
[11, 20]
```

Rysunek 1: Początek rozgrywki dla x = 20, k = 3

Na zdjęciu 1 widać początek rozgrywki. Po pierwszym ruchu komputera nie wi-

dać jeszcze jaki ciąg może układać, dlatego człowiek wybiera liczbę 11, co może go doprowadzić do ciągu $\{5, 11, 17\}$. Po drugim ruchu komputera widać, że dąży on do pokolorowania liczby 20, aby wygrać z sekwencją $\{20, 32, 44\}$. Uprzedzamy teraz ten ruch, kolorując 20.

```
Zbiór do gry:
[0, 4, 5, 9, 15, 17, 22, 47, 53, 63, 65, 70, 81, 82, 85, 94]
Ruch komputera.
Komputer wybrał element 22
Zbiór komputera:
[32, 44, 22]
Zbiór do gry:
[0, 4, 5, 9, 15, 17, 47, 53, 63, 65, 70, 81, 82, 85, 94]
Ruch gracza.
Twój zbiór:
[11, 20]
Wybierz liczbę ze zbioru, którą chcesz pokolorować swoim kolorem
17
Zbiór gracza:
[11, 20, 17]
Zbiór do gry:
[0, 4, 5, 9, 15, 47, 53, 63, 65, 70, 81, 82, 85, 94]
Ruch komputera.
Komputer wybrał element 0
Zbiór komputera:
[22, 32, 44, 0]
KOMPUTER WYGRAŁ!
Wygrywający ciąg: [0, 22, 44]
```

Rysunek 2: Komputer wygrywa

W następnym ruchu komputer koloruje 22, co widać na rysunku 2. Człowiek nie zauważa, że mógłby zablokować sekwencję $\{0, 22, 44\}$. Wybiera za to liczbę 17, aby dokończyć budowanie swojego ciągu. Jednak komputer od razu wykorzystuje swoją okazję, nie blokuje gracza i kompletuje swój ciąg, dzięki czemu wygrywa rozgrywkę.

Gdy wielkość wejściowego zbioru liczb rośnie, coraz trudniej jest znaleźć szybko ciągi o długości większej niż 3 bez dokładniejszej analizy. W takich przypadkach komputer może mieć znaczną przewagę, jeśli człowiek nie poświęci wystarczająco dużo uwagi na dopasowanie swojego ciągu, lub nie zauważy wzorca w pokolorowanych liczbach komputera. Dlatego przeprowadziliśmy kilka testów na większych zbiorach i dłuższych ciągach, ale mając małą pomoc, w postaci wypisanych sekwencji, które może ułożyć człowiek i komputer.

```
----------------------Tura nr 3-----------------------
Zbiór do gry:
[1, 15, 16, 27, 28, 43, 60, 62, 63, 65, 66, 75, 76, 89, 92, 93, 95, 103, 117, 118, 122, 127, 142, 144]
Ruch gracza.
Twój zbiór:
[22, 51, 7]
Wybierz liczbę ze zbioru, którą chcesz pokolorować swoim kolorem
75
Zbiór gracza:
[22, 51, 7, 75]
Zbiór do gry:
[1, 15, 16, 27, 28, 43, 60, 62, 63, 65, 66, 76, 89, 92, 93, 95, 103, 117, 118, 122, 127, 142, 144]
Komputer wybrał element 117
Zbiór komputera:
[39, 71, 139, 117]
Ciagi komputera
([66, 92, 118, 144] 0)
Ciągi gracza
([66, 92, 118, 144] 0)
----------------------Tura nr 4-----------------------
Zbiór do gry:
[1, 15, 16, 27, 28, 43, 60, 62, 63, 65, 66, 76, 89, 92, 93, 95, 103, 118, 122, 127, 142, 144]
Ruch gracza.
Twój zbiór:
[22, 51, 7, 75]
Wybierz liczbę ze zbioru, którą chcesz pokolorować swoim kolorem
66
Zbiór gracza:
[22, 51, 7, 75, 66]
Zbiór do gry:
[1, 15, 16, 27, 28, 43, 60, 62, 63, 65, 76, 89, 92, 93, 95, 103, 118, 122, 127, 142, 144]
Komputer wybrał element 118
Zbiór komputera:
[39, 71, 139, 117, 118]
REMIS!
Żaden z zawodników nie ułoży już ciągu
```

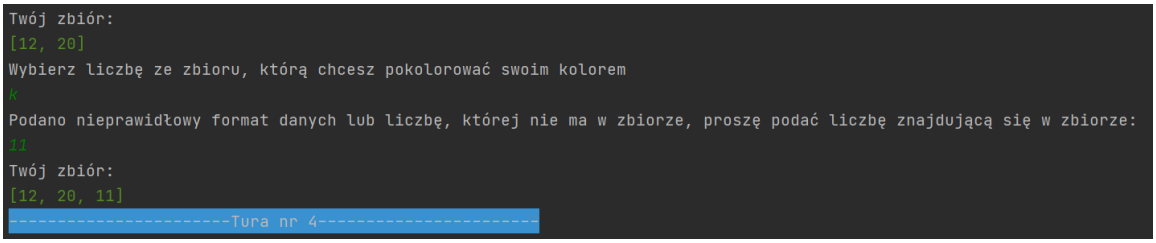
Rysunek 3: Rozgrywka dla x = 30, k = 4, remis.

Na grafice 3 przedstawiony jest końcowy fragment rozgrywki w przypadku rozpoczęcia ruchu przez gracza, oraz możliwości śledzenia sekwencji graczy, które mogą jeszcze ułożyć. Wydaje się, że przewaga jest po stronie użytkownika. Jednak już po drugim ruchu długości najdłuższych ciągów się wyrównały dzięki skutecznemu blokowaniu komputera. W kolejnym ruchu człowiek podejmuje próbę zablokowania najdłuższego ciągu maszyny. Udaje się to, ale sam gracz nie odnotowuje żadnych postępów. Kolejna próba zablokowania ruchu maszyny kończy się wyczerpaniem ruchów i remisem. Potwierdza to, że algorytm blokowania jest bardzo konsekwentny i od razu reaguje na przewagę przeciwnika. Raczej doprowadzi do remisu przez wzajemne blokowanie, ale na pewno nie pozwoli graczowi pokolorować swojego ciągu.

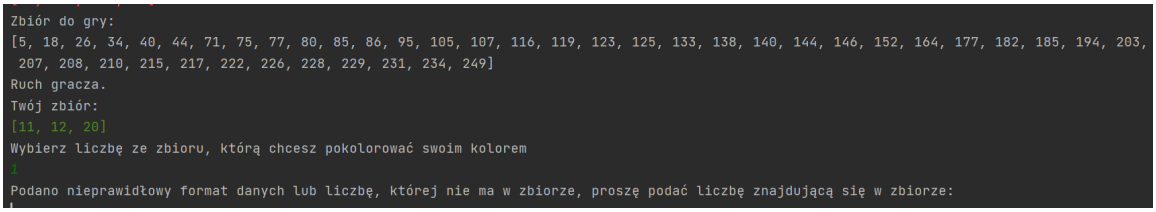
Sprawdziliśmy jeszcze jak zareaguje aplikacja na nieprawidłowe dane:

```
Gra Szemeredi'ego
Ze względu na trudność w szukaniu długich ciągów pośród wielu liczb oraz przedłużone działanie programu,najlepiej wybrać n z przedziału od 3 do
 1000 oraz k od 1 do 10.
Podaj wielkość wylosowanego zbioru:
2
Podano nieprawidłowy format danych, proszę podać liczbę od 3 do 1000.
```

Rysunek 4: Wybranie liczby spoza zakresu



Rysunek 5: Wybranie wartości, która nie jest liczbą



Rysunek 6: Wybranie liczby, której nie ma w zbiorze

Tak jak planowaliśmy, zostaliśmy ponownie poproszeni o wprowadzenie prawidłowych danych.

6 Wnioski

Gra Szmered'iego po wstępnym zapoznaniu się z nią, wydała się nam prosta oraz ciekawa. Do podobnych wniosków doszliśmy po próbie kilkukrotnego zagrania w grę na kartce papieru. Jednak, zagłębianie się w teorię opisującą ciągi arytmetyczne w zbiorze liczb naturalnych, a także próba zastosowania teorii do zbudowania strategii komputera wiązały się z wieloma nowymi zagadnieniami do rozwiązania. Twierdzenie Szemeredi'ego oraz inne formuły, do których się odwoływaliśmy w dokumentacji teoretycznej, wyjaśniały częściowo zadany problem. Dotyczyły one przeliczalnych podzbiorów liczb naturalnych, podczas gdy w praktycznym zastosowaniu korzystamy ze skończonych podzbiorów. Kolejne wyzwania czekały nas przy tworzeniu aplikacji. Sam problem szukania ciągów arytmetycznych w zbiorze liczb ma bardzo dużą złożoność obliczeniową. Jeśli algorytm ich szukania jest niepoprawnie zaimplementowany, to ma także dużą złożoność pamięciową. Dlatego nasze pierwsze próby poradzenia sobie z tym zagadnieniem były mało owocne. W procesie tworzenia aplikacji musieliśmy się skupić na przyśpieszeniu kodu pod kątem tego konkretnego algorytmu. Przy okazji optymalizacji udało nam się opracować efektywną strategię komputera przy wyborze liczb. Komputer jest w stanie skutecznie i szybko dobierać swój ciąg arytmetyczny. Dobrze też radzi sobie gdy musi blokować przeciwnika. Szczególnie dużą przewagę nad człowiekiem maszyna zdobywa, gdy wielkość zbioru początkowego jest większa. Człowiek ma bardzo małe szanse, żeby wygrać. Takie okazje pojawiają się głównie, gdy w zbiorze znajduje się dużo bardzo podobnych ciągów arytmetycznych (np. różniących się jedną liczbą) i gdy w ostatnim ruchu człowiek będzie miał możliwość pokolorowania wielu ciągów. Za każdym razem gdy człowiek będzie miał tylko jeden ciąg długości k-1, komputer go zablokuje, o ile sam nie będzie miał kończącego ruchu.