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.

- Teraz rozpoczyna się właściwa rozgrywka. Jako pierwszy ruch wykonuje komputer, a potem następuje przejście do ruchu gracza. Należy w tym momencie wybrać jedną z liczb w wyświetlającym się ciągu.

- 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 długość ciągu n oraz długość szukanego ciągu k. W związku z dużą złożonością obliczeniową, nawet po zastosowanej optymalizacji, musieliśmy zastosować ograniczenia co do wielkości pierwotnego zbioru (do 1000) oraz długości ciągu (do 10). Dodatkowo uznaliśmy, że podana wartość k ma by większa niż 2, gdyż zbudowanie ciągu długości 2 następowałoby już w drugi ruchu, bez względu na wybór elementów. Ciąg X generowany jest przy pomocy metody **SetSampling.chooseRandomSet**, która tworzy n-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.

Główna rozgrywka przedstawiona jest w klasie **Main**, natomiast metody, które bezpośrednio obsługują grę znajdują się w klasie **Game**. Grę zawsze zaczyna ruch komputera. Wybiera on liczbę, która występuje w ciągach arytmetycznych największą liczbę razy (czyli można z nią utworzyć najwięcej ciągów). Odpowiada za to metoda **SequenceOps.selectFirstNumber**.

Następnie gracze wykonują ruchy na zmianę, przy czym przed ruchem komputera sprawdzane jest, czy komputer ma blokować czy kolorować swój ciąg. W przypadku kiedy użytkownikowi brakuje tylko jednej liczby do skompletowania ciągu, koloruje taką liczbę, aby zablokować mu możliwość pokolorowania tego ciągu. W przeciwnym razie komputer sprawdza jaki ciąg w danej chwili jest najlepszy do kolorowania. Jest to najdłuższy spośród tych, które może ułożyć z połączenia niepokolorowanych, oraz tych liczb, które już pokolorował. Wtedy koloruje losową, jeszcze nie pokolorowaną liczbę z tej sekwencji. Gdy graczowi uda się pokolorować liczby, w których znajdzie się textit(k-1)-elementowy ciąg, komputer przechodzi w tryb defensywny. Polega on na sprawdzeniu jaki ciąg gracz może potencjalnie ułożyć i pokolorować brakującą mu liczbę, przez co ostatni ruch zostanie zablokowany. Zanim jednak maszyna zdecyduje się na to posunięcie, sprawdza czy ruch jest opłacalny. Gdy w zbiorze komputera też będzie textit(k-1)-elementowy ciąg, priorytetem będzie skompletowanie swojego ciągu i pokolorowanie ostatniej liczby.

Po każdym ruchu sprawdzana jest długość najdłuższego ciągu w zbiorze pokolorowanym 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. Wówczas gra się zakończy i zostanie wyświetlony odpowiedni komunikat.

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 n-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 $\{1, 2, ..., 5x-1, 5x\}$.

W ostatecznej wersji aplikacji zrezygnowaliśmy z losowania kolejności wykonywania ruchów. Grę zawsze będzie rozpoczynał komputer. 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 zrezygnować z implementacji klasy `MySet`, a wszystkie jej funkcjonalności, w tym przede wszystkim strategie blokowania ruchów użytkownika, przenieść do klasy `Game`, która odpowiada za przebieg całej gry.

Zmieniło się również nasze podejście co do oceny, który z graczy jest bliższy wygranej. Na początku chcieliśmy sprawdzać zarówno długość już pokolorowanego ciąg, jak i ilość ruchów potrzebną do wygranej. W finalnej wersji zostaliśmy jedynie przy sprawdzaniu długości ciągu człowieka, aby na jej podstawie oceniać czy trzeba wykonywać blokowanie, czy kontynuować budowanie swojego ciągu

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 maszyne.

```
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 widać 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 wcześniej wyliczając wszystkie możliwe do pokolorowania ciągi zaimplementowaną funkcją **SequenceOps.getSequences**, tak jak zaprezentowano na zdjęciu 3

```
Gra Szemeredi'ego
Ze względu na trudność w szukaniu długich ciągów pośród wielu liczb oraz przedłużone działanie programu,najle
Podaj wielkość wylosowanego zbioru:
50
Podaj długość poszukiwanego ciągu arytmetycznego od 2 do 25:
4
[[51, 54, 57, 60], [54, 106, 158, 210], [48, 51, 54, 57], [2, 54, 106, 158], [120, 136, 152, 168], [14, 60, 1
Zbiór do gry:
[2, 6, 13, 14, 20, 22, 24, 25, 32, 35, 42, 47, 48, 51, 54, 57, 60, 75, 79, 81, 82, 94, 100, 105, 106, 109, 12
Ruch komputera.
Komputer wybrał element 54
Zbiór komputera:
[54]
```

Rysunek 3: Początek rozgrywki z zapisanymi ciągami dla X = 50, k = 4

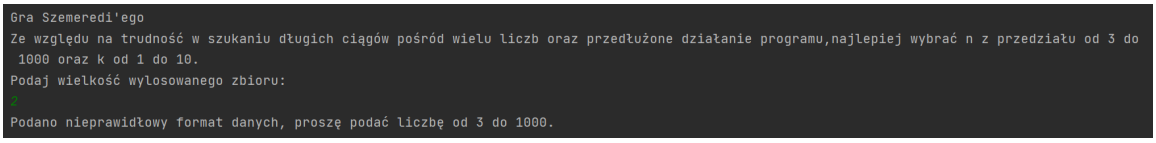
Mając je przed sobą, gracz może pokonać komputer. Wystarczy, że będzie budował jeden z obranych na początku ciągów, ale przy okazji będzie blokował te które buduje komputer. Jeśli znajdziemy się w sytuacji, gdy komputer ma już w swoim zbiorze ciąg *(k-1)*-elementowy, ale rozpoznamy go i pokolorujemy liczbę, aby zablokować ostatni ruch, komputer będzie w uśpionym trybie defensywnym, czyli nie będzie blokował ruchów człowieka.

```
Wybierz liczbę ze zbioru, którą chcesz pokolorować swoim kolorem
152
Zbiór gracza:
[48, 106, 120, 136, 152]
Zbiór do gry:
[2, 6, 13, 14, 20, 22, 24, 25, 32, 42, 47, 60, 75, 79, 81, 82, 94, 100, 105, 109, 122, 130, 133, 151, 1
Ruch komputera.
Komputer wybrał element 2
Zbiór komputera:
[35, 51, 54, 57, 210, 2]
Zbiór do gry:
[6, 13, 14, 20, 22, 24, 25, 32, 42, 47, 60, 75, 79, 81, 82, 94, 100, 105, 109, 122, 130, 133, 151, 156,
Ruch gracza.
Twój zbiór:
[48, 106, 120, 136, 152]
Wybierz liczbę ze zbioru, którą chcesz pokolorować swoim kolorem
168
Zbiór gracza:
[48, 106, 120, 136, 152, 168]
GRACZ WYGRAŁ!
Wygrywający ciąg: [120, 136, 152, 168]
```

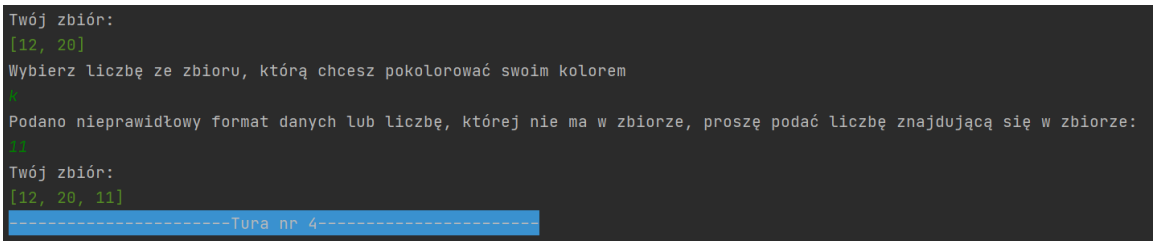
Rysunek 4: Wygrana gracza, gdy komputer nie wykona blokowania

Wynika to z tego, że dla komputera cały czas będzie widoczna informacja, że ma w swoim zbiorze *(k-1)*-elementowy ciąg i będzie w błędnym przekonaniu, że jego następny ruch będzie ostatnim. Jako, że ostatnia liczba tego ciągu jest już pokolorowana przez człowieka, komputer zacznie szukać innego dobrego ciągu do kolorowania. Maszyna wybiera kolejne liczby losowo z najbardziej optymalnego ciągu w tej chwili. W większości przypadków jednak, żeby zbudować ten ciąg, potrzebuje więcej ruchów, a w tym czasie gracz może pokolorować swój, nie będąc blokowanym i wygrać, tak jak stało się w rozgrywce na zdjęciu 4. Ze względu na długą rozgrywkę, zamieszczamy jedynie końcowy fragment.

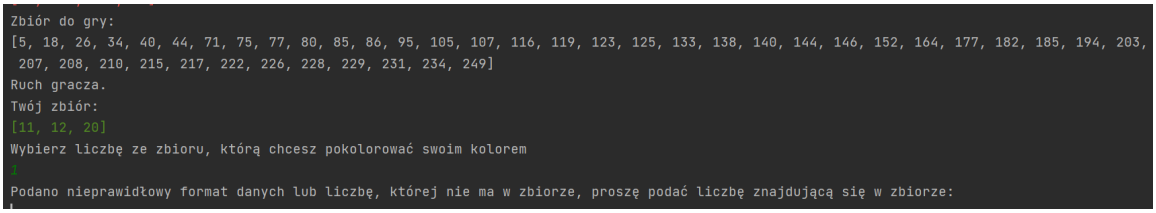
Sprawdziliśmy jeszcze jak zareaguje aplikacja na nieprawidłowe dane:



Rysunek 5: Wybranie liczby spoza zakresu



Rysunek 6: Wybranie wartości, która nie jest liczbą



Rysunek 7: 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 optymalizacji kodu pod kątem tego konkretnego algorytmu. Przy okazji przyśpiesznia, 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. Jednak gdy przeciwnik zna algorytm komputera, a także bezbłędnie zauważa wszystkie możliwe ciągi istniejące w wejściowym zbiorze, to komputera maszyny nie jest dużym problemem.