W poszukiwaniu wyzwań / Wybór zadań z konkursów programistycznych Uniwersytetu Warszawskiego

PWN

W poszukiwaniu wyzwań

W poszukiwaniu wyzwań / Wybór zadań z konkursów programistycznych Uniwersytetu Warszawskiego

PWN

UNIWERSYTET WARSZAWSKI

Pierwsze wydanie książki sfinansowano z pomocą
Ministerstwa Gospodarki,
PKO Banku Polskiego
oraz Fundacji PKO Banku Polskiego

redakcja Krzysztof Diks, Tomasz Idziaszek, Jakub Łącki, Jakub Radoszewski
projekt graficzny i skład Emilka Bojańczyk / Podpunkt
ilustracje Emilka Bojańczyk, Diana Gawronkiewicz / Podpunkt
wydawcy Łukasz Łopuszański, Edyta Kawala
korekta Ewa Mika
koordynator produkcji Anna Bączkowska

Copyright © by Wydział Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego
Warszawa 2012, 2018
Copyright © by Wydawnictwo Naukowe PWN SA
Warszawa 2018

ISBN 978-83-01-19946-3

Wydanie II poprawione (wyd. I w WN PWN)
Warszawa 2018

Wydawnictwo Naukowe PWN SA
02-460 Warszawa, ul. Gottlieba Daimlera 2
tel. 22 69 54 321, faks 22 69 54 288
infolinia 801 33 33 88
e-mail: pwn@pwn.com.pl, www.pwn.pl

książka została złożona krojami pisma Foral Pro oraz FdSymbol

druk Totem.com.pl

Przedmowa do wydania drugiego

W 2012 roku w Warszawie odbyły się finały 36. edycji Akademickich Mistrzostw Świata w Programowaniu Zespołowym. W zawodach wzięło udział 112 trzyosobowych drużyn złożonych ze studentów informatyki, wyselekcjonowanych spośród ponad 8000 zespołów z 2219 uczelni w 85 krajach. Zwyciężyła drużyna z Uniwersytetu ITMO w Petersburgu, a drugie miejsce zajął zespół z Uniwersytetu Warszawskiego w składzie: Tomasz Kulczyński, Jakub Pachocki i Wojciech Śmietanka. Cała trójka to uczestnicy, ale też współorganizatorzy konkursów algorytmiczno-programistycznych przeprowadzanych przez pracowników, doktorantów i studentów Wydziału Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego. Konkursy te pozwoliły tysiącom młodych adeptów informatyki z Polski i ze świata doskonalić umiejętności w zakresie algorytmiki i programowania. Dla wielu uczestników stały się też podstawą dalszych sukcesów w najbardziej prestiżowych konkursach programistycznych świata, a także późniejszych dokonań naukowych i zawodowych.

Z okazji tamtych zawodów powstało pierwsze wydanie książki *W poszukiwaniu wyzwań. Wybór zadań z konkursów programistycznych Uniwersytetu Warszawskiego*. W tym roku oddajemy do rąk czytelników jej drugie wydanie. Treść oryginału nie uległa żadnym merytorycznym zmianom, jednak redaktorzy pierwszego wydania, Tomasz Idziaszek, Jakub Łącki i Jakub Radoszewski, dokonali korekty usterek, które sami dostrzegli lub które zostały wykryte przez uważnych czytelników.

Liczne sukcesy autorów opracowań zawartych w książce są potwierdzeniem, że konkursy pełnią trudną do przecenienia rolę w odkrywaniu i kształceniu wyjątkowo uzdolnionych uczniów i studentów oraz pozwalają im realizować się w późniejszym życiu zawodowym. Przez ostatnie sześć lat w karierach prawie wszystkich autorów wydarzyło się bardzo wiele, dlatego krótkie biogramy zawarte w pierwszym wydaniu uzupełniliśmy o najważniejsze osiągnięcia autorów w latach 2012–2018.

Szymon Acedański: nauczyciel akademicki na Wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego, założyciel i CTO w firmie InviNets zajmującej się technologiami Internetu Rzeczy.

Marcin Andrychowicz: w 2016 roku uzyskał na Wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego stopień doktora nauk matematycznych w zakresie informatyki (kryptografia), w 2013 roku został srebrnym medalistą Akademickich Mistrzostw Świata w Programowaniu Zespołowym, a w 2016 roku był finalistą konkursu Google Code Jam. Obecnie pracuje w OpenAI w Kalifornii.

Piotr Chrząstowski: nauczyciel akademicki na Wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego, aktywny członek Komitetu Głównego Olimpiady Informatycznej.

Marek Cygan: pracownik naukowo-dydaktyczny na Wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego, w 2018 roku uzyskał stopień doktora habilitowanego w dziedzinie nauk matematycznych w zakresie informatyki, laureat Starting Grant Europejskiej Rady ds. Badań (ERC), współzałożyciel firmy NoMagic.AI zajmującej się programowaniem robotów.

Tomasz Czajka: od 2014 roku pracuje w SpaceX w Kalifornii, gdzie zajmuje się oprogramowaniem dla komputerów pokładowych rakiet nośnych Falcon i statków kosmicznych Dragon. W 2014 roku zajął drugie miejsce w konkursie Facebook Hacker Cup.

Krzysztof Diks: profesor informatyki na Wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego, od 2016 roku Przewodniczący Polskiej Komisji Akredytacyjnej, 20-krotny uczestnik Finałów Akademickich Mistrzostw Świata w Programowaniu Zespołowym jako współopiekun

reprezentacji Uniwersytetu Warszawskiego. W latach 2012–2017 drużyny z Uniwersytetu Warszawskiego zdobyły 2 medale złote, 3 medale srebrne i 1 medal brązowy, w tym dwukrotnie wicemistrzostwo świata (2012 i 2017).

Andrzej Gąsienica-Samek: założyciel i prezes zarządu polskiej firmy informatycznej Atinea, pomysłodawca i projektant portali edukacyjnych wspierających naukę języków obcych – Insta.Ling oraz programowania – instaKod.

Tomasz Idziaszek: w 2014 roku uzyskał na Wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego stopień doktora nauk matematycznych w zakresie informatyki (teoria języków drzew nieskończonych), współpracuje z firmą Codility w zakresie przygotowywania zadań algorytmicznych i rozwijania oprogramowania do sprawdzania kompetencji programistów, od 2013 roku sekretarz naukowy Olimpiady Informatycznej.

Grzegorz Jakacki: założyciel i CEO firmy Codility budującej narzędzia do automatycznej weryfikacji kompetencji programistów. Z usług Codility przy rekrutacji programistów korzystają m.in. Amazon, Intel, Microsoft, Nokia, PayPal, Samsung, Slack, Tesla.

Tomasz Kociumaka: doktorant informatyki na Wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego, w 2013 roku laureat diamentowego grantu MNiSW, współautor ponad 50 prac opublikowanych przez czołowe światowe wydawnictwa w zakresie algorytmiki, w latach 2013 i 2014 srebrny medalista Akademickich Mistrzostw Świata w Programowaniu Zespołowym, w 2015 roku finalista konkursu Google Code Jam oraz Facebook Hacker Cup.

Eryk Kopczyński: adiunkt na Wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego, wielokrotny laureat konkursów informatycznych, w 2017 roku wystąpił w finałach Facebook Hacker Cup i Yandex.Algorithm.

Marcin Kubica: nauczyciel akademicki na Wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego, od 2016 roku lider zespołu inżynierskiego w firmie Codility.

Tomasz Kulczyński: współzałożyciel firmy CodiLime, od 2012 roku inżynier oprogramowania w Google w Warszawie, w latach 2013 i 2016 finalista konkursu Facebook Hacker Cup.

roku uzyskał stopień doktora nauk matematycznych w zakresie informatyki (algorytmy tekstowe), od 2016 roku wiceprzewodniczący Komitetu Głównego Olimpiady Informatycznej.

Wojciech Rytter: profesor informatyki na Wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego, aktywny twórca ciekawych i inspirujących zadań na konkursy algorytmiczno-programistyczne.

Krzysztof Stencel: profesor informatyki na Wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego.

Wojciech Śmietanka: w 2012 roku ukończył studia na Wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego, jest współzałożycielem świetnie rozwijającej się polskiej firmy informatycznej CodiLime, był inżynierem oprogramowania w Google w Zurychu, obecnie pracuje w Londynie w funduszu inwestycyjnym Quadrature Capital.

Tomasz Waleń: pracownik naukowo-dydaktyczny na Wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego, współzałożyciel i ekspert firmy Codility.

Jakub Wojtaszczyk: pracownik Google w Warszawie, pomysłodawca i współtwórca oprogramowania na potrzeby konkursów z zakresu algorytmów rozproszonych, juror podczas finałów Akademickich Mistrzostw Świata w Programowaniu Zespołowym.

Filip Wolski: od 2012 roku pracował dla czołowych amerykańskich firm zajmujących się *high-frequency trading*, m.in. dla Teza Technologies, od sierpnia 2016 roku prowadzi badania naukowe związane ze sztuczną inteligencją w firmie OpenAI.

Oddając w ręce czytelników drugie wydanie *W poszukiwaniu wyzwań,* mamy nadzieję, że zawarte w nim zadania będą inspiracją do zgłębiania tajników algorytmiki i programowania, a zdobyte w ten sposób umiejętności pozwolą czytelnikom stawić czoła wyzwaniom współczesnej informatyki z takimi samymi sukcesami, jakie są udziałem autorów opracowań zawartych w tej książce.

Krzysztof Diks
Warszawa, 2018

/ Wstęp

Ta książka przedstawia skromną część niemal dwudziestoletniego dorobku polskich konkursów algorytmiczno-programistycznych, organizowanych lub współorganizowanych przez pracowników i studentów Wydziału Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego. Historia konkursów w Polsce to przede wszystkim ludzie – organizatorzy i uczestnicy. O jakości przeprowadzonych konkursów świadczą między innymi wybór Polski jako organizatora Międzynarodowej Olimpiady Informatycznej w 2005 roku oraz powierzenie Uniwersytetowi Warszawskiemu roli gospodarza światowych finałów konkursu ACM International Collegiate Programming Contest w 2012 roku.

Wizytówką polskich konkursów są również międzynarodowe osiągnięcia ich uczestników. Najważniejsze z nich to zwycięstwa Filipa Wolskiego i Tomasza Kulczyńskiego w Międzynarodowej Olimpiadzie Informatycznej, odpowiednio w latach 2006 i 2007, oraz dwukrotne zwycięstwo zespołów Uniwersytetu Warszawskiego w finałach konkursu ACM International Collegiate Programming Contest, odpowiednio w latach 2003 (drużyna w składzie: Tomasz Czajka, Andrzej Gąsienica-Samek i Krzysztof Onak) i 2007 (drużyna w składzie: Marek Cygan, Marcin Pilipczuk i Filip Wolski). Należy też wspomnieć o znakomitych występach Polaków, w tym zwycięstwach, w konkursach TopCoder Open, Google Code Jam, Facebook Hacker Cup i Microsoft Imagine Cup. Te sukcesy nie byłyby możliwe, gdyby nie odpowiednio wysoki poziom konkursów krajowych, które pozwoliły wyłowić algorytmiczne diamenty, a następnie je oszlifować.

O konkursach

Konkursy pełnią rolę nie do przecenienia w odkrywaniu i kształceniu wyjątkowo uzdolnionych uczniów i studentów. Wymagają wiedzy i umiejętności wybiegających dalece poza to, czego uczy się w szkole i na studiach. Dobry konkurs dotyka zawsze podstaw dziedziny, której dotyczy, a wiedza i umiejętności w nim zdobyte dają niezbędne podwaliny dalszego rozwoju w tej dziedzinie. Jeszcze ważniejszym jest, żeby konkurs kształcił w młodych ludziach umiejętności, które są potrzebne w ich późniejszej aktywności zawodowej: pracowitość, systematyczność, samodyscyplinę, dociekliwość, samodoskonalenie, pracę w zespole, uczciwość, ambicję, chęć konkurowania, dążenie do sukcesu. Startowanie w dobrym konkursie powinno być dla młodego człowieka wyzwaniem intelektualnym, a osiągnięty w nim sukces – nobilitować.

O jakości konkursu decydują przede wszystkim zadania. Powinny być one oryginalne, ciekawe i o różnym stopniu trudności. Ich rozwiązanie powinno sprawiać satysfakcję, a kłopoty z rozwiązaniem zachęcać do pogłębiania swojej wiedzy i umiejętności. Ta książka zawiera subiektywny wybór „najlepszych" zadań z konkursów algorytmiczno-programistycznych, współorganizowanych przez Uniwersytet Warszawski, wraz z opisem rozwiązań wzorcowych. Wyboru zadań dokonały osoby, które odegrały znaczące role w historii polskich konkursów algorytmiczno-programistycznych jako ich uczestnicy lub organizatorzy. Autorzy wszystkich opracowań w tej książce mają ścisłe związki z Wydziałem Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego jako jego absolwenci, pracownicy lub studenci.

Zadania przedstawione i omówione w tej książce pochodzą z następujących konkursów informatycznych: Olimpiada Informatyczna, Olimpiada Informatyczna Gimnazjalistów, Obóz Naukowo-Treningowy im. Antoniego Kreczmara, Olimpiada Informatyczna Krajów Europy Środkowej, Akademickie Mistrzostwa Polski w Programowaniu Zespołowym i Potyczki Algorytmiczne.

Olimpiada Informatyczna została powołana w 1993 roku przez Instytut Informatyki Uniwersytetu Wrocławskiego, kierowany wówczas przez profesora Macieja Sysłę. Wśród członków założycieli Olimpiady było czterech pracowników Instytutu Informatyki Uniwersytetu Warszawskiego: Piotr Chrząstowski-Wachtel, Jan Madey (wówczas dyrektor Instytutu), Wojciech Rytter i Stanisław Waligórski, który został pierwszym przewodniczącym Komitetu

Głównego Olimpiady Informatycznej. Od samego początku zapleczem technicznym i naukowym Olimpiady był Instytut Informatyki Uniwersytetu Warszawskiego. Wszyscy autorzy rozdziałów w tej książce byli lub są nadal związani z Olimpiadą.

Olimpiada Informatyczna jest konkursem indywidualnym przeznaczonym dla uczniów szkół średnich. Zwycięzcy (pierwsza czwórka) krajowych odpowiedników Olimpiady reprezentują swoje kraje w zawodach Międzynarodowej Olimpiady Informatycznej.

Olimpiada Informatyczna składa się z trzech etapów. Pierwszy z nich jest etapem szkolnym, rozgrywanym na przełomie października i listopada, i gromadzi około tysiąca uczestników. Mają oni do rozwiązania zazwyczaj pięć zadań i pracują nad nimi w domu. Wyniki swojej pracy przesyłają organizatorom do oceny przez Internet. Do drugiego etapu awansuje około 400 najlepszych zawodników z pierwszego etapu. Jest on organizowany w kilku ośrodkach regionalnych ściśle współpracujących z najlepszymi uczelniami informatycznymi w Polsce i trwa trzy dni. Pierwszy dzień jest poświęcony na zapoznanie się z warunkami rozgrywania zawodów. W każdym z następnych dwóch dni uczestnicy mają do samodzielnego rozwiązania 2–3 zadania w trakcie pięciogodzinnej, kontrolowanej sesji. Rozwiązania z całej Polski są zbierane centralnie i oceniane w takim samym środowisku i na tych samych testach. Około 80 najlepszych uczestników drugiego etapu awansuje do finału Olimpiady. Finał jest rozgrywany w jednym miejscu i trwa pięć dni. Trzy dni są przeznaczone na same zawody, a dwa na rekreację i turystykę. Sposób rozgrywania finału jest podobny do tego z drugiego etapu. Czterej najlepsi zawodnicy reprezentują Polskę na międzynarodowych zawodach informatycznych, w tym Międzynarodowej Olimpiadzie Informatycznej.

Łącznie w 19 edycjach Olimpiady Informatycznej wystartowało około 15 000 uczniów, którzy rozwiązywali łącznie prawie 300 zadań.

Międzynarodowa Olimpiada Informatyczna jest rozgrywana latem każdego roku i gromadzi najlepszych na świecie młodych informatyków – uczniów szkół średnich. Pierwsza Międzynarodowa Olimpiada Informatyczna miała miejsce w 1989 roku. Od tego czasu Polacy zdobyli 31 medali złotych, 28 medali srebrnych i 22 medale brązowe.

Pracownicy i studenci Uniwersytetu Warszawskiego brali czynny udział w organizacji zawodów międzynarodowych blisko związanych z Olimpiadą Informatyczną: Międzynarodowej Olimpiady Informatycznej (IOI 2005), trzykrotnie **Olimpiady Informatycznej Krajów Europy Środkowej** (CEOI 1997, 2004, 2011) oraz dwukrotnie **Bałtyckiej Olimpiady Informatycznej** (BOI 2001, 2008).

Co roku w wakacje finaliści Olimpiady Informatycznej mają okazję uczestniczyć w **Obozie Naukowo-Treningowym im. Antoniego Kreczmara** (ONTAK), rozwiązując i omawiając zadania w większości przygotowane przez ich starszych kolegów z Uniwersytetu Warszawskiego.

Od sześciu lat dla uczniów gimnazjów jest organizowana „mała olimpiada" – **Olimpiada Informatyczna Gimnazjalistów**. Pomysł powołania do życia tej Olimpiady pochodzi od wybitnego nauczyciela Ryszarda Szubartowskiego, a Instytut Informatyki Uniwersytetu Warszawskiego aktywnie uczestniczył w jej powstaniu i rozwoju.

Wielu finalistów Olimpiady Informatycznej wybiera jako miejsce swoich studiów informatycznych Uniwersytet Warszawski. Tutaj mogą kontynuować swoją pasję konkursową, walcząc o miejsca w reprezentacji Uniwersytetu Warszawskiego na zawody w programowaniu zespołowym, w tym o możliwość startu w konkursie **ACM International Collegiate Programming Contest**. Konkurs ACM ICPC jest najstarszym i najbardziej prestiżowym konkursem informatycznym na świecie i jest uznawany za mistrzostwa świata w programowaniu zespołowym. Każda drużyna biorąca w nim udział składa się z trzech studentów reprezentujących tę samą uczelnię. Pierwszym etapem konkursu są eliminacje regionalne. Jest ich kilkadziesiąt i rozgrywane są na wszystkich zamieszkałych kontynentach. Najlepsze drużyny z eliminacji (zwycięzcy oraz kilka następnych drużyn, zależnie od siły regionu i liczby startujących drużyn) awansują do finałów.

Zarówno zawody eliminacyjne, jak i finałowe są rozgrywane w ten sam sposób. Każda trzyosobowa drużyna ma do dyspozycji jeden komputer, pięć godzin i od 8 do 12 zadań do rozwiązania. Zgłaszane przez uczestników rozwiązania poszczególnych zadań są sprawdzane na bieżąco, a drużyny są informowane o wyniku sprawdzenia zadania, który przyjmuje postać krótkiego komunikatu: zaakceptowane, niepoprawne wyniki, przekroczenie limitu czasu na rozwiązanie, błąd wykonania, niepoprawna prezentacja wyników. Poszczególne

zadania są oceniane binarnie – albo zadanie jest zaliczane, albo rozwiązanie jest odrzucane. Zawody wygrywa drużyna, która zaliczy najwięcej zadań. W przypadku takiej samej liczby zaliczonych zadań przez więcej niż jedną drużynę, o ich kolejności w rankingu decyduje krótszy, łączny czas przeznaczony na rozwiązanie wszystkich zaliczonych zadań, przy czym za każde odrzucone zgłoszenie zaliczonego zadania nalicza się 20 minut kary.

Historia konkursu ACM ICPC sięga 1977 roku i jest nierozerwalnie związana z osobą profesora Williama B. Pouchera, pomysłodawcy i animatora konkursu. Na grunt polski konkurs ten przeszczepił profesor Jan Madey, który w 1994 roku po raz pierwszy sformował drużynę, złożoną ze studentów informatyki Uniwersytetu Warszawskiego, i razem z nią pojechał na eliminacje do Amsterdamu, które to zespół Uniwersytetu Warszawskiego wygrał. Jako zwycięzcy eliminacji regionalnych warszawiacy pojechali na finały do Nashville, gdzie zajęli 11. miejsce. Od tego czasu każdego roku zespół Uniwersytetu Warszawskiego kwalifikował się do finału konkursu ACM ICPC, wygrywając go dwukrotnie w latach 2003 i 2007. W 2012 roku Uniwersytet Warszawski jest gospodarzem finałów konkursu.

Warto wspomnieć, że Uniwersytet Warszawski ma już pewne doświadczenie w organizacji tego typu zawodów. W latach 2001–2003 był organizatorem **Akademickich Mistrzostw Europy Środkowej w Programowaniu Zespołowym** – eliminacji konkursu ACM ICPC, a w latach 1998–2000 oraz w 2011 roku – **Akademickich Mistrzostw Polski w Programowaniu Zespołowym**.

W 2001 roku, przy okazji Bałtyckiej Olimpiady Informatycznej, wystartował konkurs programistyczny Pogromcy Algorytmów, który w 2005 roku przyjął nazwę **Potyczki Algorytmiczne**. Jest to konkurs indywidualny, w którym udział może wziąć każdy chętny, bez względu na wiek i status zawodowy. Składa się on z dwóch etapów. Etap pierwszy rozgrywany jest przez Internet i trwa około tygodnia. Jest on podzielony na 5–6 rund. W każdej rundzie uczestnicy mają do rozwiązania od jednego do czterech zadań. Skala trudności rośnie z rundy na rundę. Najlepsza dwudziestka po pierwszym etapie jest zapraszana na stacjonarne zawody finałowe, które są rozgrywane w stylu konkursu ACM ICPC (z tą różnicą, że uczestnicy rozwiązują zadania indywidualnie). Zwycięzca finałów otrzymuje tytuł Algorytmistrza Roku. Konkurs Potyczki Algorytmiczne cieszy

się bardzo dużym zainteresowaniem. Co roku startuje w nim kilka tysięcy osób, a sukcesy w konkursie jego uczestnicy odnotowują w swoich CV.

W ramach organizowanych przez nas konkursów ułożono i opracowano setki oryginalnych zadań. Żeby nie zaprzepaścić tego dorobku, staramy się umieszczać je w portalu dydaktycznym **MAIN** (dostępnym pod adresem http://main.edu.pl*), który powstał z inicjatywy organizatorów Olimpiady Informatycznej i jest prowadzony przez pracowników i studentów Uniwersytetu Warszawskiego. Zasoby portalu MAIN są zintegrowane z systemem komputerowym Olimpiady Informatycznej zwanym **SIO**, który umożliwia sprawdzanie w czasie rzeczywistym, 24 godziny na dobę, rozwiązań zadań umieszczonych w portalu. W szczególności każdy Czytelnik tej książki może samodzielnie przygotować rozwiązania zadań w niej zamieszczonych i sprawdzić wyniki swojej pracy, korzystając z portalu MAIN. Zapraszamy!

O autorach

Autorami opracowań w tym wydawnictwie są osoby, które mają ścisłe związki z Uniwersytetem Warszawskim i odcisnęły znaczący ślad w historii konkursów informatycznych organizowanych lub współorganizowanych przez Uniwersytet. Ze względu na charakter tej książki ograniczyliśmy się tylko do osób, które mają doświadczenie w przygotowywaniu i rozwiązywaniu zadań konkursowych. Wśród autorów są wykładowcy akademiccy i studenci, zawodnicy o międzynarodowych sukcesach w konkursach programistycznych, edukatorzy i popularyzatorzy informatyki.

Każdy autor został poproszony o wybranie i opisanie rozwiązań dwóch zadań z konkursów współorganizowanych przez Uniwersytet Warszawski, które uznał za najciekawsze, przy czym na wybór zadania mogły mieć wpływ zarówno jego trudność, metody rozwiązania i walory edukacyjne, jak i osobiste wspomnienia z nim związane. Każdy autorski opis rozwiązań jest poprzedzony krótkim biogramem, przedstawiającym związki autora z konkursami programistycznymi oraz jego zainteresowania naukowe i pozanaukowe.

* Aktualnie portal MAIN nie jest wspierany, a archiwum zadań zostało przeniesione do portalu http://szkopul.edu.pl.

Podziękowania

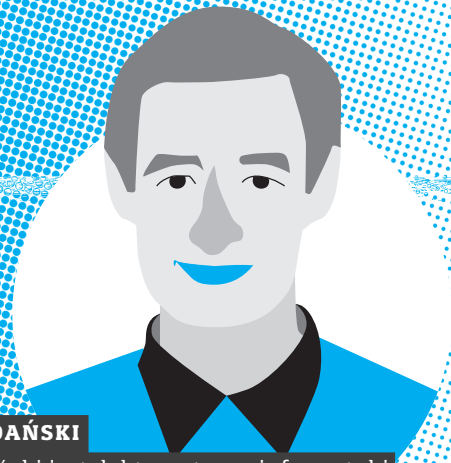
W tej książce mogliśmy tylko uhonorować z imienia i nazwiska około 30 osób, które przyczyniły się do rozwoju konkursów informatycznych w Polsce, a tym samym do popularyzacji informatyki i pośrednio do wykształcenia rzeszy bardzo dobrych informatyków. Wyrazy uznania należą się jednak wszystkim, którzy w tym dziele uczestniczą. Pragnę gorąco podziękować autorom zamieszczonych tu opracowań, którzy zgodzili się przygotować swoje teksty w ekstremalnie krótkim czasie. Na szczególne wyróżnienie zasługują ci spośród nich, którzy jednocześnie pełnili rolę redaktorów merytorycznych tego opracowania: Tomasz Idziaszek, Jakub Łącki i Jakub Radoszewski. Bez ich zaangażowania i ciężkiej redakcyjnej pracy ta publikacja nie miałaby szans powstania. Przygotowując angielskie tłumaczenie tej książki, korzystaliśmy z pomocy mojej córki Justyny Diks i mojego przyjaciela Jerzego Jaromczyka. Natomiast za redakcję językową wersji angielskiej szczególne podziękowania należą się Richardowi Hallasowi, wspieranemu w tej pracy przez Grzegorza Jakackiego i Marcina Kubicę. Jestem niezmiernie wdzięczny studiu graficznemu Podpunkt za projekt szaty graficznej książki i za cierpliwość w pracy z redaktorami przy jej składaniu. Przy powstawaniu tego dzieła wspierał nas i dopingował Rafał Sikorski, prawnik kochający informatykę.

Wydanie tej książki w takiej postaci, w językach polskim i angielskim, w nakładach po 1000 egzemplarzy w każdym języku, nie byłoby możliwe bez wsparcia finansowego Ministerstwa Gospodarki, PKO Banku Polskiego i Fundacji PKO Banku Polskiego. Kierownictwom tych instytucji, doceniającym olbrzymią rolę działalności edukacyjnej, składam gorące podziękowania.

Wszystkim Czytelnikom życzę przyjemnej lektury i satysfakcji z poznawania tajników algorytmiki!

Krzysztof Diks

Zadania >



SZYMON ACEDAŃSKI

Szymon Acedański jest doktorantem z informatyki na Wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego, gdzie zajmuje się dydaktyką oraz pracą naukową nad przetwarzaniem języka naturalnego. Współpracuje z Zespołem Inżynierii Lingwistycznej Polskiej Akademii Nauk, realizując zadania w projektach z tej dziedziny.

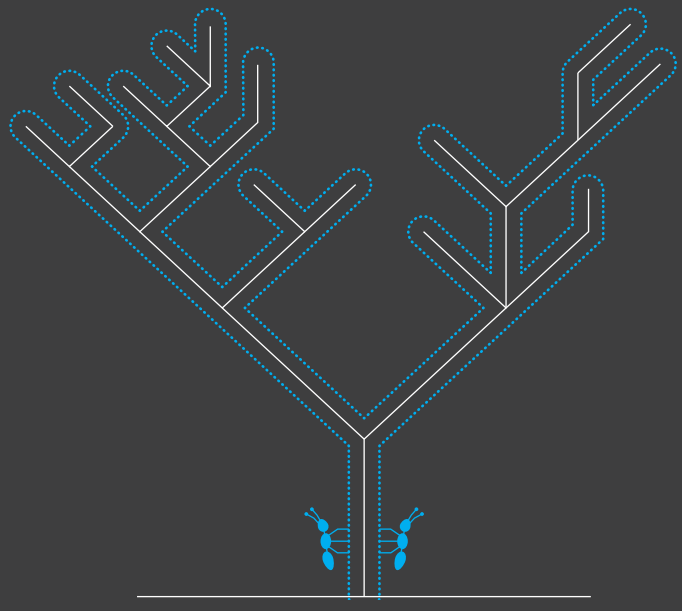
Jest zdobywcą dwóch złotych medali na Bałtyckiej Olimpiadzie Informatycznej. Zwyciężył również w Mistrzostwach Europy Środkowej w Programowaniu Zespołowym w 2004 roku i reprezentował wówczas Uniwersytet Warszawski w światowych finałach konkursu ACM ICPC.

Aktualnie kieruje zespołem technicznym Olimpiady Informatycznej, a także wspiera Krajowy Fundusz na rzecz Dzieci, co roku prowadząc warsztaty dla jego podopiecznych.

W wolnym czasie fotografuje, a w cieplejsze pory roku lubi żeglować oraz jeździć po świecie.

/ Drzewo *i mrówki*

Konkurs: Akademickie Mistrzostwa Polski w Programowaniu Zespołowym 2011
Autor zadania: Szymon Acedański
Pamięć: 6 MB
https://oi.edu.pl/pl/archive/amppz/2011/drz



Informatycy lubią drzewa. Mrówki także lubią drzewa. Niech więc będzie dane drzewo, po którym chodzą dwie mrówki – Lewa Mrówka i Prawa Mrówka – w sposób pokazany na powyższym rysunku (po ścieżce oznaczonej kropkowaną linią). Rozpoczynają one swoją wędrówkę na początku pnia, po przeciwnych jego stronach. Idąc od korzenia (w górę), Lewa Mrówka pokonuje każdą krawędź drzewa w dwie sekundy, zaś idąc w kierunku korzenia (w dół) – w sekundę. Prawa Mrówka jest od niej dwa razy szybsza. W momencie, gdy mrówki spotykają się, obydwie zawracają. Jeśli któraś z nich zejdzie z drzewa na ziemię, natychmiast zaczyna wchodzić na drugą stronę pnia. Poza tym mrówki są tak malutkie, że nawet mikroskop nie wystarczyłby do ich zobaczenia (na rysunku celowo zostały powiększone). Twoim zadaniem jest napisanie programu, który obliczy, po jakim czasie mrówki zawrócą po raz drugi.

Wejście

W pierwszym wierszu wejścia znajduje się jedna liczba całkowita t ($1 \leqslant t \leqslant 1000$) oznaczająca liczbę zestawów testowych opisanych w dalszej części wejścia.

Opis każdego zestawu składa się z dwóch wierszy. W pierwszym z nich znajduje się parzysta liczba n ($2 \leqslant n \leqslant 100\,000\,000$) oznaczająca liczbę krawędzi drzewa. W drugim wierszu znajduje się opis drzewa. Jest to napis długości $\frac{n}{2}$ reprezentujący liczbę binarną $2n$-bitową zapisaną w systemie szesnastkowym (przy użyciu cyfr oraz małych liter od a do f). Liczba ta opisuje, jak wygląda obejście całego drzewa przez Lewą Mrówkę przy założeniu, że Prawa Mrówka stoi w miejscu. Kolejne bity tej liczby (od lewej) oznaczają, czy idąc po kolejnych krawędziach na swojej ścieżce, Lewa Mrówka oddala się od korzenia (bit 1), czy też zbliża się do korzenia (bit 0). Drzewo posiada pień, tzn. dokładnie jedna krawędź wychodzi z korzenia drzewa.

Rozmiar żadnego pliku wejściowego nie przekroczy 50 MB. Uwaga: to jest istotnie więcej niż rozmiar dostępnej pamięci.

Wyjście

Twój program powinien wypisać t wierszy z odpowiedziami dla kolejnych zestawów testowych. Odpowiedzią dla zestawu jest czas w sekundach, po którym mrówki zawrócą po raz drugi, wypisany jako nieskracalny ułamek p/q (bez spacji wokół /), gdzie p i q to liczby całkowite dodatnie. Jeśli odpowiedź jest liczbą całkowitą, to oczywiście $q = 1$.

Przykład

Dla danych wejściowych:

1
28
fb1da30d1b7230

poprawną odpowiedzią jest:

282/5

Wyjaśnienie do przykładu: Przykład odpowiada rysunkowi. Ciąg bitów wygląda następująco:

1111 1011 0001 1101 1010 0011 0000 1101 0001 1011 0111 0010 0011 0000

/ Rozwiązanie

Drzewo i mrówki to jedyne moje zadanie, które zostało wymyślone metodą „to teraz ułożę zadanie". Do tej pory próby natychmiastowego ułożenia zadania kończyły się niepowodzeniem. Ale tym razem, pewnego słonecznego popołudnia, gdzieś za morzem, w inspirującej obecności Jakuba Radoszewskiego oznajmiającego, że potrzeba zadań na Akademickie Mistrzostwa Polski w Programowaniu Zespołowym, w mojej głowie powstał taki oto nieciekawy monolog:

— Hmm, zadanie... No dobrze, to musi mieć jakiegoś bohatera.
— Bajtazar? Ee, nie, to nie Olimpiada Informatyczna... To może jakieś zwierzątko. Wszyscy lubimy zwierzęta.
— Wybierzmy coś z informatycznego zoo: biedronki, mrówki, pchły?
— O, mrówki! Było kiedyś takie fajne zadanie, które usłyszałem od Pawła Parysa:

Zadanie. Po poziomej gałęzi, wychodzącej z jednej strony z pnia drzewa, spacerują mrówki. Każda z taką samą prędkością. Gdy któreś dwie się spotykają, obie zawracają. Gdy któraś dojdzie do pnia, też zawraca, a gdy dojdzie do drugiego końca gałęzi, to spada na ziemię. Znając początkowy układ mrówek, oblicz, kiedy ostatnia mrówka spadnie z gałęzi*.

— A co by było, gdyby mrówki nie spadały z gałęzi, tylko trzeba by liczyć ich pozycje?
— E tam, nic ciekawego, łaziłyby w kółko. A gdyby do tego niektóre mrówki były szybsze?
— E tam, za trudne, takie zadanie umiem rozwiązać jedynie metodą brutalną. A gdyby jeszcze było mniej mrówek, na przykład dwie?
— E tam, za proste. A gdyby teraz zastąpić tę gałąź czymś bardziej skomplikowanym?
— O, do tego idealnie pasuje drzewo. I to takie „normalne": korzeniem w dół.
— E tam, to samo w sobie jedynie trochę kamufluje fakt, że mrówki po prostu chodzą po okręgu.
— O, to niech chociaż odcinki „w górę" będą trudniejsze do przejścia.
— No dobrze, tylko na czym teraz ma polegać zadanie? Może kazać obliczyć miejsce spotkania się mrówek?

* Czytelniku, gorąco polecam Ci spróbować zmierzyć się z tym zadaniem. Ma przepiękne rozwiązanie.

— O, umiem to zrobić bez wczytywania całego drzewa: wystarczy symulować drogę jednej z mrówek. Wiedząc, ile odcinków „w dół" i ile „w górę" zostało, mogę obliczyć w każdym momencie czas potrzebny drugiej mrówce na pokonanie reszty drzewa... tak, to by było fajne.

Kusiło mnie, żeby tu skończyć i ogłosić sukces. Ale jednak:

— A gdyby kazać obliczyć miejsce ponownego spotkania się mrówek?

Muszę przyznać, że zaciekawiło mnie to pytanie. Nie spodziewałem się, że tak opisana sytuacja ma zadziwiające własności. W szczególności zupełnie nie oczekiwałem, że prawdziwa będzie taka obserwacja:

Spostrzeżenie. Po pierwszym spotkaniu Lewa Mrówka zawsze wróci do korzenia szybciej niż Prawa Mrówka.

Tym bardziej, że to Prawa Mrówka miała być szybsza...

Rozwiązanie

Drogi Czytelniku, powyższa historia całkiem prawdopodobnie nie przybliżyła Cię zbytnio do pełnego rozwiązania. Ba, nawet przytoczone spostrzeżenie wydaje się co najmniej podejrzane. Dlatego też teraz spróbuję opowiedzieć Ci rozwiązanie krok po kroku. Ograniczenie na rozmiar wejścia zostało dobrane tak, żeby uniemożliwić przechowanie całego drzewa w pamięci. Rozwiązanie wzorcowe jednokrotnie wczytuje dane z wejścia, bez ich zapamiętywania. Szkic rozwiązania wygląda następująco:

1. Wczytujemy rozmiar drzewa, dzięki czemu znamy liczbę odcinków „w górę" i „w dół".
2. Wczytując opis drzewa, symulujemy marsz Lewej Mrówki do momentu pierwszego spotkania. Dla każdej krawędzi sprawdzamy, czy doszło na niej do spotkania, bo wiedząc, jaką odległość w górę i w dół pokonała Lewa Mrówka, znamy dystans, jaki musiała pokonać Prawa.
3. Wczytujemy wejście dalej, tym razem symulując marsz Prawej Mrówki do momentu, aż Lewa Mrówka wróci do korzenia.
4. Dalej symulujemy marsz Prawej Mrówki aż do momentu drugiego spotkania.

Żeby do końca przekonać się o tym, że to zadanie rzeczywiście da się tak rozwiązać, przyjrzyjmy się formalnemu opisowi wzorcowego rozwiązania przygotowanemu przez Tomasza Idziaszka, który miał przyjemność weryfikować opracowanie zadania przed ostatecznym przekazaniem go na zawody.

Punkt na drzewie możemy opisać przez parę liczb rzeczywistych (a, k), gdzie a oznacza, ile krawędzi musiała przejść Lewa Mrówka od korzenia, żeby dojść do tego punktu, zaś k oznacza wysokość punktu nad poziomem gruntu.

Jeżeli Lewa Mrówka krawędź do góry pokonuje w czasie t_g, a krawędź w dół w czasie t_d, to do punktu (a,k) dojdzie w czasie

$$[a,k,t_g,t_d] := \frac{a+k}{2}t_g + \frac{a-k}{2}t_d.$$

Zatem punkt pierwszego spotkania (a_1,k_1) zostanie osiągnięty, gdy

$$[a_1,k_1,2,1] = [2n-a_1,k_1,1,\tfrac{1}{2}]. \tag{1}$$

Aby znaleźć taki punkt, możemy, wczytując drzewo, symulować ruch Lewej Mrówki. W momencie, gdy stoimy w punkcie (a,k), który jest początkiem pewnej krawędzi, potrafimy sprawdzić, czy spotkanie nastąpi na tej właśnie krawędzi. Niech $b=1$, jeśli jest to krawędź w górę, i $b=-1$, jeśli jest to krawędź w dół. Gdy do spotkania miałoby dojść po tym, jak przejdziemy kawałek krawędzi równy ε, to równość (1) będzie spełniona dla

$$a_1 = a+\varepsilon, \quad k_1 = k+b\varepsilon,$$

zatem

$$\varepsilon = \frac{6n-9a-k}{9+b}.$$

Jeśli więc $0 \leqslant \varepsilon < 1$, to znaleźliśmy punkt pierwszego spotkania. Czas potrzebny na dojście do tego punktu to

$$t_1 = [a_1,k_1,2,1] = \frac{3a_1+k_1}{2}.$$

Jeśli pozwolilibyśmy teraz wrócić mrówkom do korzenia, to Lewa Mrówka zrobiłaby to w czasie $[a_1,-k_1,2,1] = t_1 - k_1$, natomiast Prawa - w czasie

$$[2n-a_1,-k_1,1,\tfrac{1}{2}] = t_1 - \frac{k_1}{2}.$$

Widać zatem, że Lewa Mrówka zawsze zdąży wrócić do korzenia przed Prawą.

Możemy teraz szukać punktu drugiego spotkania. Będziemy to robić, symulując ruch Prawej Mrówki. Jeśli przejdzie ona fragment długości ε' krawędzi wychodzącej z punktu (a', k'), to dojdzie do punktu spotkania (a_2, k_2), o ile będzie spełnione

$$a_2 = a' + \varepsilon', \quad k_2 = k' + b'\varepsilon',$$

$$[a_1, -k_1, 2, 1] + [2n - a_2, k_2, 2, 1] = [a_2 - a_1, k_2 - k_1, 1, \tfrac{1}{2}],$$

tak więc

$$\varepsilon' = \frac{12n - 9(a' - a_1) + (k' - k_1)}{9 - b'}.$$

Jeśli zatem $0 \leqslant \varepsilon' < 1$, to znaleźliśmy drugi punkt spotkania. Czas potrzebny na dojście tam to

$$t_2 = [a_2 - a_1, k_2 - k_1, 1, \tfrac{1}{2}] = \frac{3(a_2 - a_1) + (k_2 - k_1)}{4}.$$

Zatem sumaryczny czas, który upłynie do momentu ponownego spotkania obu mrówek, jest równy

$$t_1 + t_2 = \frac{3(a_1 + a_2) + (k_1 + k_2)}{4}.$$

Zauważmy, że ε to wielokrotność $\frac{1}{9+b}$, natomiast proste obliczenia pokazują, że ε' to wielokrotność $\frac{9-b}{(9+b)(9-b')}$. Wynika z tego, że wartości a_1, a_2, k_1, k_2 są wielokrotnościami ułamka $\frac{1}{800}$. Można zatem wszystkie obliczenia wykonywać w arytmetyce stałopozycyjnej, używając 64-bitowych zmiennych całkowitoliczbowych, w których przechowujemy wartości przemnożone przez 800.

W efekcie otrzymujemy algorytm, który w każdym kroku wczytuje nową krawędź i wykonuje stałą liczbę rachunków. Działa zatem w czasie $O(n)$, w pamięci stałej.

MARCIN ANDRYCHOWICZ

Marcin Andrychowicz jest studentem na Wydziale Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego. W liceum trzykrotnie zdobył złoty medal na Międzynarodowej Olimpiadzie Informatycznej, zajął trzecie miejsce w finale konkursu TopCoder High School oraz wygrał Potyczki Algorytmiczne 2008. Na studiach był kapitanem drużyny, która wygrała Mistrzostwa Europy Środkowej w Programowaniu Zespołowym oraz zdobyła brązowy medal w światowych finałach konkursu ACM ICPC w 2009 roku.

Obecnie interesuje się systemami kryptograficznymi, dla których można udowodnić odporność na ataki fizyczne, oraz automatyczną spekulacją na rynkach finansowych. W czasie wolnym chodzi po górach, jeździ na snowboardzie i podróżuje.

/ Hiperzegar

Konkurs: Obóz Naukowo-Treningowy im. A. Kreczmara 2007
Autor zadania: Jakub Pawlewicz
Pamięć: 32 MB
https://oi.edu.pl/pl/archive/ontak/2007/hip

Smerf Ważniak stwierdził, że dni są dla niego zbyt krótkie i nie zdąży przez to napisać wszystkich tomów *Złotych myśli Smerfa Ważniaka*. Postanowił więc udać się do Ojca Czasu i nakłonić go, aby ten wydłużył dobę do 48 godzin. Ojciec Czas początkowo nie chciał przystać na tę propozycję, ale namolny smerf nie dawał za wygraną i w końcu staruszek się zgodził. Postawił jednak jeden warunek: Ważniak ma rozwiązać zagadkę Hiperzegara.

Hiperzegar składa się z N zegarów. Każdy z nich ma jedną wskazówkę i kilka liczb na tarczy. Na tarczy i-tego zegara (dla $1 \leqslant i \leqslant N$) znajdują się liczby od 1 do k_i. Początkowo na wszystkich zegarach wskazówki pokazują liczbę 1. Ojciec Czas tak objaśniał zagadkę Hiperzegara smerfowi:

Masz wykonać serię ruchów. Każdy ruch polega na przesunięciu wskazówki dowolnie wybranego zegara o jedną liczbę w lewo lub w prawo. Po ostatnim ruchu wskazówki Hiperzegara muszą wrócić do pierwotnego położenia. Ponadto w wyniku przesuwania wskazówek, każda możliwa konfiguracja Hiperzegara musi pojawić się dokładnie raz.

Ważniak już godzinę majstruje coś przy Hiperzegarze i jest coraz bardziej przekonany, że pomimo całej swej mądrości nie będzie umiał rozwiązać jego zagadki. Pomóż biednemu smerfowi!

Wejście

Pierwszy wiersz wejścia zawiera jedną liczbę całkowitą dodatnią N. W kolejnych N wierszach znajduje się po jednej liczbie całkowitej większej niż 1: w $(i+1)$-szym wierszu znajduje się liczba k_i. Liczba wszystkich możliwych konfiguracji zegara (czyli $k_1 k_2 \cdots k_N$) jest nie większa niż 10^6.

Wyjście

Jeśli zagadka Hiperzegara nie ma rozwiązania, należy wypisać jedno słowo NIE. W przeciwnym wypadku należy wypisać $k_1 k_2 \cdots k_N$ wierszy – opisy kolejnych ruchów. Każdy z nich powinien zawierać dwie liczby całkowite. Pierwsza liczba i $(1 \leqslant i \leqslant N)$ oznacza numer zegara, którego wskazówkę się przesuwa. Druga

liczba oznacza, w którą stronę przesuwamy wskazówkę: 1 (jeśli w prawo) lub −1 (jeśli w lewo). Możliwe są różne poprawne odpowiedzi, Twój program ma wypisać dowolną z nich.

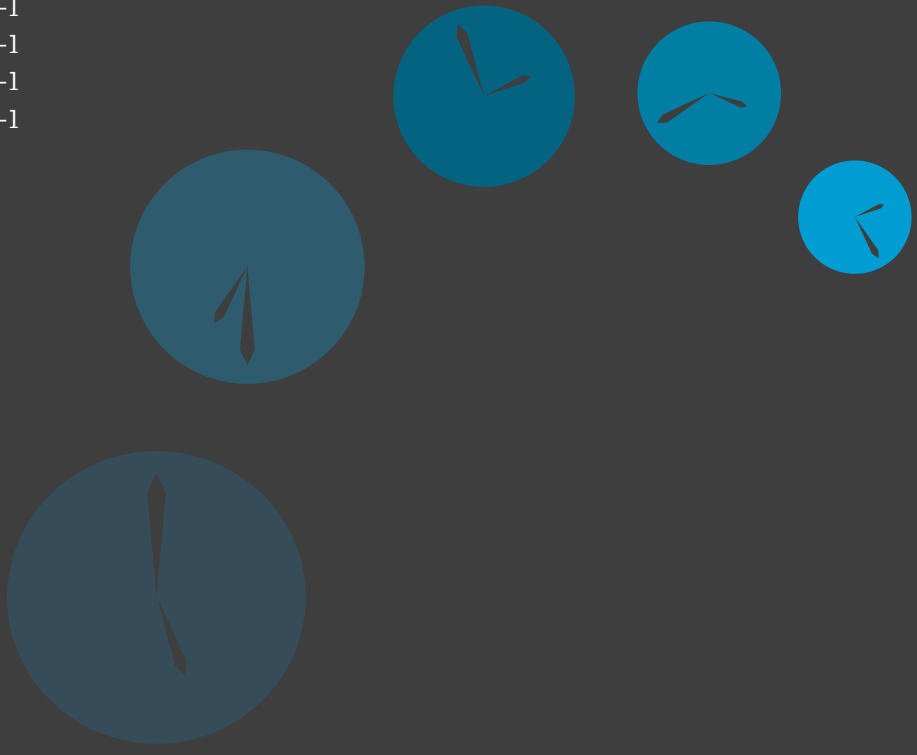
Przykład

Dla danych wejściowych:

3
2
2
3

poprawną odpowiedzią jest:

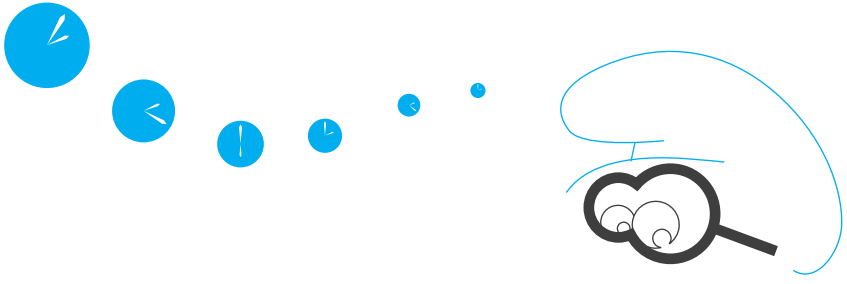
1 1
2 1
1 −1
3 1
1 1
2 −1
3 1
2 1
1 −1
2 −1
3 −1
3 −1



/ Rozwiązanie

Zacznijmy od matematycznego sformułowania problemu postawionego w zadaniu. Konfigurację Hiperzegara możemy opisać jako ciąg $x_1, x_2, \ldots, x_N$ oznaczający liczby wskazywane przez poszczególne zegary, przy czym $x_i \in \{1, 2, \ldots, k_i\}$. Ruch polega na przesunięciu jednego z zegarów w prawo lub lewo; możemy zatem w jednym kroku przejść ze stanu $x_1, x_2, \ldots, x_N$ do $y_1, y_2, \ldots, y_N$, jeśli istnieje taki indeks j, że $y_i = x_i$ dla $i \neq j$ oraz $y_j = x_j \pm 1$. Ponadto liczby 1 i k_j sąsiadują ze sobą na tarczy j-tego zegara, więc dopuszczamy ruchy, w których $x_j = 1$ i $y_j = k_j$ lub $x_j = k_j$ i $y_j = 1$.

Naturalna jest interpretacja Hiperzegara jako grafu, w którym wierzchołki odpowiadają możliwym stanom Hiperzegara, zaś krawędzie – możliwym przejściom między tymi stanami. Jak można zinterpretować zagadkę Ojca Czasu w tym grafie? Szukamy cyklu przechodzącego przez każdy wierzchołek dokładnie raz, czyli tzw. *cyklu Hamiltona*. Problem znalezienia cyklu Hamiltona w dowolnym grafie należy do klasy problemów NP-trudnych. Oznacza to, że najprawdopodobniej nie istnieje wielomianowe rozwiązanie dla tego problemu. Nasz graf z zadania ma jednak specyficzną strukturę*, co pozwala rozwiązać zadanie w czasie liniowym.



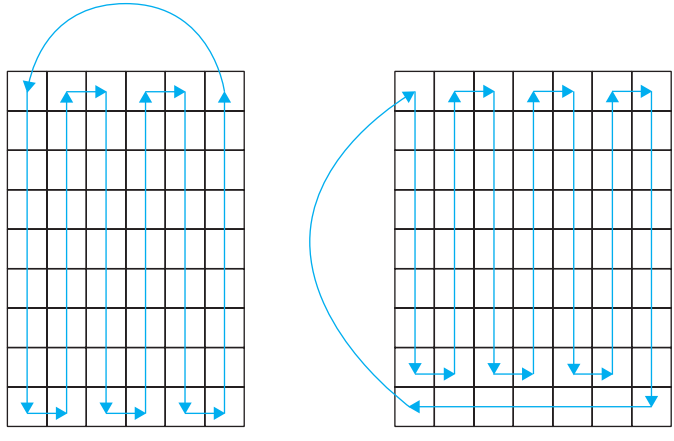
* Formalnie graf ten jest iloczynem kartezjańskim cykli prostych.

Dwa zegary

Spróbujmy rozwiązać najpierw zagadkę, w której Hiperzegar składa się z dokładnie dwóch zegarów. Sytuację łatwo wyobrazić sobie wówczas jako prostokątną kratkę rozmiaru k_1 na k_2. Możemy przejść z jednego pola do drugiego, jeśli mają one wspólny bok. Ponadto możemy „przenikać przez boki planszy", tzn. przejść z ostatniego pola każdego wiersza lub kolumny do pierwszego i na odwrót.

Na rys. 1 przedstawiono sytuację dla $k_1 = 6$ i $k_2 = 9$ wraz z rozwiązaniem. Ten pomysł łatwo uogólnić na sytuację, kiedy przynajmniej jeden bok ma długość parzystą. Poniższy pseudokod odpowiada rysunkowi, zakładając, że zaczynamy z lewego górnego rogu:

```
Algorithm ParzystyBok(k1,k2)
    {zakładamy, że 2 | k1}
    for i := 1 to k1/2 do
        for j := 1 to k2 − 1 do
            print 2 1 {w dół}
        print 1 1 {w prawo}
        for j := 1 to k2 − 1 do
            print 2 −1 {w górę}
        print 1 1 {w prawo}
```



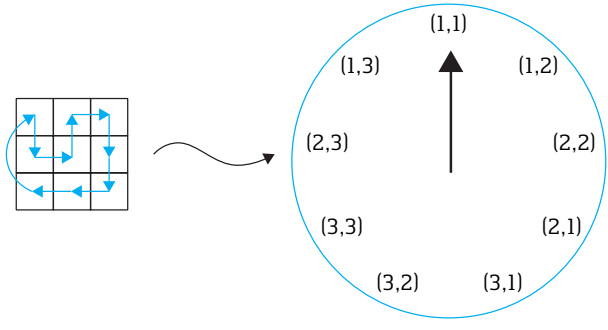
Rysunek 1: *Rozwiązanie dla $k_1 = 6$ i $k_2 = 9$ (po lewej) oraz dla $k_1 = 7$ i $k_2 = 9$ (po prawej).*

Sytuacja, kiedy oba boki mają nieparzyste długości, wymaga jedynie niewielkiej modyfikacji powyższego pomysłu. Na rys. 1 znajduje się przykładowe rozwiązanie dla $k_1 = 7$ i $k_2 = 9$. Znów zakładamy, że zaczynamy z lewego górnego rogu:

```
Algorithm NieparzystyBok(k1,k2)
    {zakładamy, że 2 ∤ k1}
    for i := 1 to (k1 − 1)/2 do
        for j := 1 to k2 − 2 do
            print 2 1 {w dół}
        print 1 1 {w prawo}
        for j := 1 to k2 − 2 do
            print 2 −1 {w górę}
        print 1 1 {w prawo}
    for i := 1 to k2 − 1 do
        print 2 1 {w dół}
    for i := 1 to k1 − 1 do
        print 1 −1 {w lewo}
    print 2 1 {w dół}
```

Więcej zegarów

Wiemy już, jak rozwiązać zagadkę dla dwóch zegarów. Okazuje się, że to wystarcza do rozwiązania ogólnej wersji zagadki. Po znalezieniu cyklu Hamiltona dla pierwszych dwóch zegarów możemy zacząć traktować je jako jeden zegar o k_1k_2 stanach - wystarczy znaleziony cykl Hamiltona nałożyć na tarczę nowego zegara!



Rysunek 2: *Operacja łączenia dwóch zegarów długości* 3 *w jeden.*

Wskazówka nowego zegara wskazuje parę liczb oznaczającą, w jakim stanie znajdują się oba zegary, a przesunięcie jej w prawo lub lewo oznacza przesunięcie pierwszego lub drugiego zegara w odpowiednią stronę. Pomiędzy każdymi dwoma sąsiednimi stanami możemy przejść w jednym ruchu, gdyż stany te tworzyły cykl (patrz rys. 2).

Tak utworzony zegar możemy następnie połączyć z trzecim zegarem. Kontynuując to postępowanie, dochodzimy do sytuacji, kiedy mamy tylko jeden zegar prezentujący stan całego Hiperzegara. Wystarczy wówczas przekręcić jego wskazówki o 360° i zagadka Ojca Czasu jest rozwiązana. Oznacza to w szczególności, że rozwiązanie zawsze istnieje.

Reprezentacja zegarów

Aby zaimplementować takie rozwiązanie, musimy zdecydować, w jaki sposób będziemy reprezentować tworzone zegary. Zauważmy, że nie musimy znać wartości na tarczy zegara – wystarczy nam informacja, jaki ruch odpowiada przekręceniu tego zegara w prawo. Dla powyższego przykładu musimy wiedzieć, że przy przejściu z pierwszej pozycji (ze stanu $(1,1)$) do drugiej (do stanu $(1,2)$) konieczne jest przekręcenie drugiego zegara w prawo (tzn. wykonanie ruchu 2 1) itd. Zegar o k stanach będziemy więc reprezentować jako tablicę długości k, gdzie pod i-tym indeksem znajduje się para (a,b) ($a \in \{1,2,\ldots,N\}$ i $b \in \{-1,1\}$) oznaczająca ruch, który trzeba wykonać, aby przejść z i-tego stanu do $(i+1)$-szego (bądź ewentualnie z ostatniego do pierwszego).

Ta informacja wystarcza do przekręcania zegara w obie strony, gdyż jeśli para (a,b) odpowiada przejściu z i-tego stanu do $(i+1)$-szego, to przejściu z $(i+1)$-szego stanu do i-tego odpowiada para $(a,-b)$.

Używając rozwiązania dla dwóch zegarów, możemy scalić dwa zegary w czasie liniowym względem liczby stanów w wynikowym zegarze. Oszacujmy łączny czas potrzebny do dokonania wszystkich $N-1$ scaleń. Niech $M = k_1 k_2 \cdots k_N$. Całe rozwiązanie działa w czasie proporcjonalnym do

$$k_1k_2 + k_1k_2k_3 + \cdots + M = M\left(\frac{1}{k_3k_4\cdots k_N} + \frac{1}{k_4k_5\cdots k_N} + \cdots + 1\right) \leqslant$$
$$\leqslant M\left(\frac{1}{2^{N-2}} + \frac{1}{2^{N-3}} + \cdots + 1\right) < 2M,$$

czyli liniowym względem rozmiaru wyjścia.

/ Ryby

Konkurs: Potyczki Algorytmiczne 2009
Autor zadania: Eryk Kopczyński
Pamięć: 256 MB
https://oi.edu.pl/pl/archive/pa/2009/ryb

W wodach pewnego archipelagu żyje rzadki gatunek drapieżnych ryb. Prowadzą one bardzo regularny tryb życia. Każda ryba budzi się codziennie o tej samej porze i wypływa na polowanie. Wieczorem, po zakończonych łowach, wraca w miejsce, z którego wyruszyła. Tam kładzie się spać (każdego dnia o tej samej godzinie), jednak może obudzić się następnego dnia w innym miejscu, gdyż prądy morskie mogą ją nieco znieść.

W ciągu całego dnia ryba zachowuje następującą zasadę: w każdym momencie musi widzieć punkt, w którym była w tym samym momencie poprzedniego dnia, czyli dokładnie 24 godziny wcześniej. Oczywiście, ryba nie może widzieć punktu, który jest po przeciwnej stronie jakiejś wyspy.

Ichtiolodzy przez dłuższy czas prowadzili obserwację ryb w archipelagu i co kilka dni notowali jedną trasę, którą w ciągu tego dnia przepłynęła pewna ryba. Niestety po zgromadzeniu dużej ilości danych nastąpił wypadek i dokumentacja rozsypała się przy przenoszeniu. Część informacji całkowicie zaginęła, a reszta jest pomieszana. Ichtiolodzy nie wiedzą nawet, która trasa opisuje ruch której z ryb. Proszą Cię zatem o pomoc – przekażą Ci pomieszany zbiór opisów tras, a Ty im powiesz, jaka mogła być minimalna liczba różnych ryb, które obserwowali podczas badań.

Wejście

Wejście składa się z dwóch części: z opisu archipelagu i opisu tras ryb. Pierwszy wiersz opisu archipelagu zawiera dwie liczby całkowite w oraz h ($3 \leqslant w \leqslant 1000$, $3 \leqslant h \leqslant 1000$), oddzielone pojedynczym odstępem. W każdym z kolejnych h wierszy znajduje się opis fragmentu archipelagu w postaci ciągu w znaków. Znak `.` oznacza ocean, a `#` – ląd. We wszystkich polach na brzegu mapy jest woda (znak `.`).

Z pewnego punktu w oceanie ryba widzi inny, jeśli odcinek, który je łączy, nie ma punktów wspólnych z wnętrzem ani brzegiem jakiegokolwiek fragmentu lądu. Kierunek płynięcia ryby nie ma tu znaczenia.

W następnym wierszu znajduje się liczba całkowita n ($2 \leqslant n \leqslant 1000$), oznaczająca liczbę zanotowanych tras ryb. W kolejnych $2n$ wierszach następują opisy poszczególnych tras. W pierwszym wierszu opisu trasy znajdują się trzy liczby całkowite x, y oraz d ($1 \leqslant x \leqslant w$, $1 \leqslant y \leqslant h$, $2 \leqslant d \leqslant 10\,000$), pooddzielane pojedynczymi odstępami. Liczby x i y to współrzędne miejsca, w którym ryba się obudziła (numer kolumny i wiersza mapy), zaś d to długość trasy. Drugi wiersz składa się z d znaków N, W, S lub E określających kierunek płynięcia ryby. Oznaczają one odpowiednio kierunki: w górę, w lewo, w dół i w prawo. Opis trasy jest tak skonstruowany, że przebiega ona jedynie po polach z wodą, nie wybiega poza fragment archipelagu opisany w wejściu, a jej koniec wypada w tym samym miejscu co początek.

Ryba porusza się jedynie w pionie lub poziomie i podczas polowania płynie po łamanej łączącej środki pól na ścieżce; nie znamy jednak jej prędkości. Prędkość ta nie musiała być stała na całej trasie, ryba mogła przyspieszać i zwalniać, byleby tylko widzieć punkt, w którym była dokładnie dobę wcześniej.

Prądy morskie mogą znieść śpiącą rybę co najwyżej o jedno pole w górę, dół, lewo lub prawo od miejsca, w którym położyła się ona spać. Możesz założyć, że dla dowolnych dwóch pól oznaczających wodę, istnieje (hipotetyczna) trasa ryby, która przez nie przepływa.

Wyjście

W pierwszym wierszu wyjścia należy wypisać liczbę całkowitą k – najmniejszą możliwą liczbę różnych ryb, jaka pasuje do danych zebranych przez ichtiologów. W każdym z kolejnych k wierszy powinny znaleźć się listy numerów tras, które mogła pokonać dana ryba – trasy *nie musiały* jednak zostać przez nią przebyte *w kolejnych dniach*, a potencjalnie w dowolnych momentach jej życia.

Dwie trasy mogą być pokonane przez daną rybę *w dwóch kolejnych dniach*, jeżeli zaczynają się w tym samym bądź w sąsiednich (mających wspólną krawędź) polach oraz jeżeli ryba mogła przepłynąć je tak, żeby w każdym momencie dnia, płynąc po drugiej trasie, widziała punkt z pierwszej trasy, w którym była dokładnie dobę wcześniej.

Trasy są ponumerowane od 1 do n, zgodnie z kolejnością w wejściu. Numery tras w każdym wierszu wyjścia należy wypisać w kolejności rosnącej, natomiast wiersze muszą być wypisane w takiej kolejności, by pierwsze numery w każdym z nich tworzyły ciąg rosnący.

Przykład

Dla danych wejściowych:

```
10   8
..........
........#..
........#.
..........
...#......
.......###.
.......###.
..........
4
2   7   12
NNNEEESSSWWW
2   8   24
WNNNNNNNEEEEESSSSSSSWWWW
1   8   46
NNNNNNNEEEEESSSSSSSEEEENNNNNNNSSSSSSSWWWWWWWWW
1   8   32
NNNNNNNEEEEEEEESSSSSSSWWWWWWWWW
```

poprawną odpowiedzią jest:

```
2
1   2   3
4
```

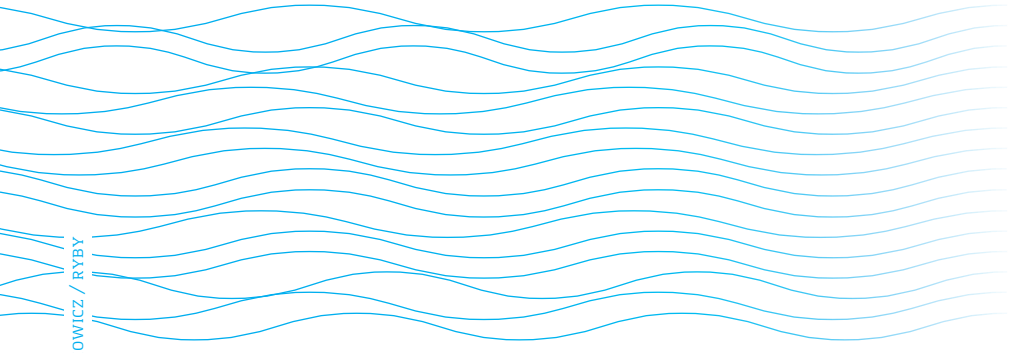
Wyjaśnienie do przykładu: Pierwsze dwie trasy mogły być przebyte przez tę samą rybę (nawet w dwóch kolejnych dniach). Po kilku kolejnych dniach ryba mogła popłynąć trzecią trasą. Ostatnia trasa musiała być jednak przebyta przez inną rybę. W przeciwieństwie do pierwszych trzech, okrąża ona dużą wyspę.

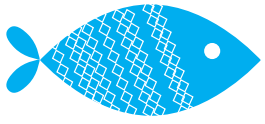
/ Rozwiązanie

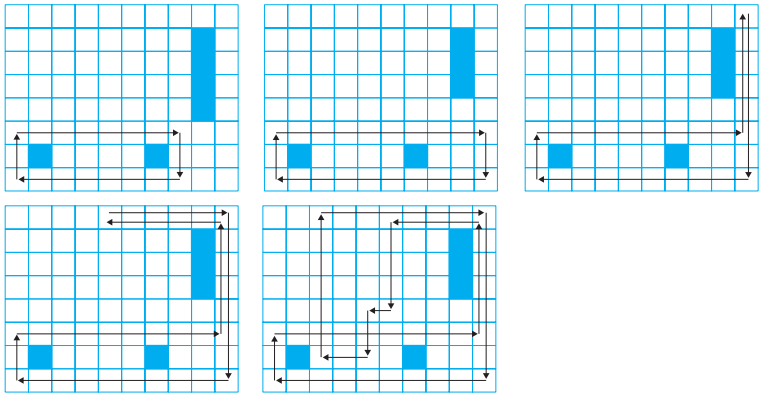
Zadanie *Ryby* z pewnością nie jest standardowe. Jest to jedyne znane mi zadanie mające związek z topologią. Opis zachowania ryb został w treści podany w taki sposób, że po jego lekturze niełatwo jest o intuicję, kiedy dwie trasy mogły być przepłynięte przez jedną rybę (niekoniecznie w kolejnych dniach). Okazuje się jednak, że można ten warunek wyrazić dużo prościej i że odpowiada on pewnemu pojęciu z topologii. Trasy takie będziemy w dalszej części rozwiązania nazywać *równoważnymi*. Zadanie polega na wyznaczeniu klas abstrakcji relacji równoważności tras.

Zauważmy najpierw, że jeśli mamy dwie trasy o identycznym kształcie i kierunku płynięcia, różniące się jedynie miejscem, w którym budzi się ryba, to są one równoważne. Jest tak, ponieważ ryba mogła płynąć codziennie tą samą trasą, ze stałą szybkością, zaczynając każdego dnia z kolejnego pola trasy (na które była zniesiona poprzedniej nocy). Wybór punktu, w którym budzi się ryba, nie ma więc żadnego znaczenia i wystarczy rozważać kształty utworzonych tras, które są pętlami skierowanymi.

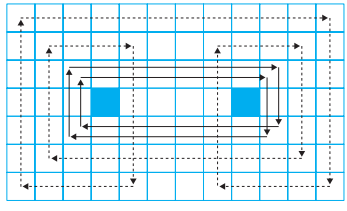
Patrząc na zamieszczony na rys. 1 przykład, można domyślić się, że dwie trasy są równoważne, kiedy jedna z tras może być uzyskana z drugiej przez ciągłe „rozciąganie" lub „ściąganie" bez rozrywania. W topologii takie pętle nazywają się *homotopijnymi*. Szkic dowodu tego faktu znajduje się na końcu rozwiązania. Jego znajomość nie jest w żadnym razie konieczna do zrozumienia reszty rozwiązania.







Rysunek 1: *Przykładowe trasy, które mogły być przepłynięte przez jedną rybę w kolejnych dniach. Miejsce, w którym ryba się budzi, nie zostało zaznaczone, ponieważ, jak pokazaliśmy, nie ma ono znaczenia.*



Rysunek 2: *Obie trasy okrążają każdą z wysp dwa razy. Pomimo to, nie są one równoważne.*

Pierwszy pomysł

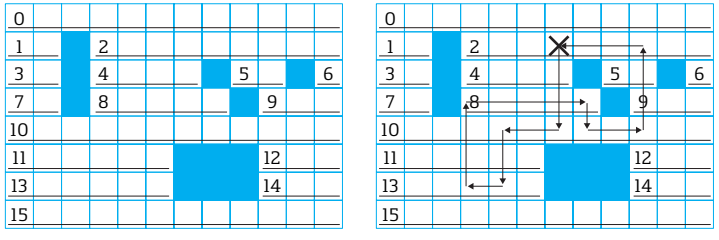
Zastanówmy się, kiedy dwie trasy są równoważne. Z rys. 1 widać, że dwie takie trasy mogą mieć zupełnie różne kształty. Na archipelagu, na którym nie ma żadnych wysp, wszystkie trasy są równoważne. Jeśli jednak mamy wyspy, to trasa, która opływa wyspę, nie jest równoważna z taką, która tej wyspy nie opływa. Ponadto trasa opływająca wyspę raz nie jest równoważna z taką, która opływa tę wyspę dwa razy. Ważny jest również kierunek obiegu wyspy. Powyższy przykład prowadzi do pomysłu, że liczenie, ile razy każda trasa obiega każdą wyspę, pozwoli odróżnić nierównoważne trasy. Okazuje się jednak, że jest to kryterium niewystarczające (patrz rys. 2).

Reprezentacja tras

Kluczem do rozwiązania zadania jest sprowadzenie każdej trasy do pewnej postaci kanonicznej. Chcielibyśmy, aby trasy były równoważne wtedy i tylko wtedy, kiedy ich kanoniczne reprezentacje są równe. Wyobraźmy sobie, że trasy ryb zamieniamy na gumki recepturki. Kurczą się one i zatrzymują na brzegach wysp. Takie kurczenie jest na pewno ciągłym przekształceniem, więc jeśli z dwóch tras otrzymaliśmy taki sam kształt gumki (i takie samo jej skierowanie), to obie trasy są równoważne. Implikacja w drugą stronę również jest prawdziwa, tzn. trasy są równoważne wtedy i tylko wtedy, gdy utworzone z nich gumki kurczą się do identycznego kształtu (uwzględniając skierowanie). Czytelnik może przekonać się o tym, rysując dowolne pętle i wyznaczając dla każdej z nich kształt gumki po ściągnięciu. Uznajemy przy tym, że trasy ryb, które nie opływają żadnej wyspy, kurczą się do mikroskopijnych rozmiarów i znikają. Są więc one wszystkie równoważne.

Powstaje pytanie, jak można zasymulować taki proces ściągania tras. Zacznijmy od wyobrażenia sobie, że na każdym spójnym poziomym fragmencie oceanu umieściliśmy laserową barierę, która potrafi wykryć fakt przepłynięcia ryby przez barierę oraz kierunek jej ruchu (patrz rys. 3).

Zakładamy, że ryba przecina barierę, jeśli wypływa z pola, na którym znajduje się bariera, na południe, bądź wpływa na pole z barierą od południa. Dla każdej trasy możemy wyznaczyć, jakie bariery i w którą stronę ona przecina. Zauważmy, że jeśli ryba przecina jakąś barierę i zaraz potem zawraca, to możemy oba takie przecięcia zignorować, otrzymując trasę równoważną. Może to spowodować, że sąsiednie staną się dwa inne przecięcia, które będzie można usunąć itd.



Rysunek 3: *Po lewej: Przykładowy archipelag z zaznaczonymi barierami. Bariery zostały dodatkowo ponumerowane liczbami naturalnymi. Po prawej: Trasa ryby, dla której wyznaczymy ciąg przecięć. Pole, w którym ryba się budzi, oznaczone jest krzyżykiem.*

Dla przykładu z rys. 3 otrzymalibyśmy następujący ciąg przecięć:

2S, 4S, 8S, 10S, 11S, 11N, 10N, 8N, 8S, 9N, 5N, 2N,

gdzie N oznacza, że ryba płynęła na północ, a S – że na południe. Usuwając iteracyjnie sąsiadujące ze sobą przecięcia dotyczące tej samej bariery, dostajemy następujący ciąg przecięć (pogrubiona została para usuwana w danym kroku):

1. 2S, 4S, 8S, 10S, **11S**, **11N**, 10N, 8N, 8S, 9N, 5N, 2N,
2. 2S, 4S, 8S, **10S**, **10N**, 8N, 8S, 9N, 5N, 2N,
3. 2S, 4S, **8S**, **8N**, 8S, 9N, 5N, 2N,
4. **2S**, 4S, 8S, 9N, 5N, **2N** (trasa jest w rzeczywistości cykliczna, więc uznajemy, że ostatni element sąsiaduje z pierwszym),
5. 4S, 8S, 9N, 5N.

Przecięcia do zignorowania można wyznaczyć w czasie liniowym podczas wyznaczania barier, które dana trasa przecina. W tym celu można napotkane na danej trasie przecięcia pamiętać na stosie. Po napotkaniu nowego przecięcia wystarczy sprawdzić, czy nie dotyczy ono tej samej bariery co poprzednie i jeśli tak jest, to oba przecięcia należy usunąć. Na koniec wystarczy usuwać pierwsze i ostatnie przecięcie tak długo, jak długo dotyczą one tej samej bariery. Podczas usuwania przecięć można nie uwzględniać w ogóle kierunku przecięć, gdyż ryba nie może przekroczyć dwa razy z rzędu tej samej bariery z tej samej strony. Tak utworzony ciąg będziemy nazywać *reprezentacją* dla danej trasy. Kierunek przecięć ma znaczenie jedynie przy porównywaniu reprezentacji dla różnych tras, gdyż pozwala odróżnić rybę opływającą wyspę zgodnie z kierunkiem ruchu wskazówek zegara od ryby opływającej tę wyspę w kierunku przeciwnym.

Równoważność cykliczna

Okazuje się, że dwie powstałe trasy są równoważne wtedy i tylko wtedy, kiedy ich reprezentacje są równoważne cyklicznie*. Fakt ten pozostawimy bez dowodu, ale Czytelnik może przekonać się o tym, analizując przykładowe trasy ryb. Opisy mogą być przesunięte cyklicznie, ponieważ cykliczne przesunięcie trasy zależy od miejsca, w którym ryba rozpoczęła swoją wędrówkę, co, jak wiemy, nie wpływa na równoważność tras.

* Dwa ciągi nazywamy *równoważnymi cyklicznie*, jeśli pewne cykliczne przesunięcie jednego ciągu jest równe drugiemu ciągowi. *Cykliczne przesunięcie* słowa $a_1a_2 \ldots a_n$ to każde słowo postaci $a_ia_{i+1} \ldots a_na_1a_2 \ldots a_{i-1}$.

Zadanie polega na znalezieniu klas abstrakcji relacji równoważności tras, co odpowiada klasom abstrakcji relacji równoważności cyklicznej ich reprezentacji. Najprostszy algorytm sprawdzający, czy dwa słowa są cyklicznie równoważne, działa w czasie $O(d^2)$, gdzie d to maksymalna długość obu słów. Możemy wyznaczyć podział na klasy abstrakcji, sprawdzając, czy każda para tras jest cyklicznie równoważna. Daje to rozwiązanie działające w czasie $O(wh + n^2d^2)$.

Poprawienie złożoności

Szybsze rozwiązania można uzyskać, stosując efektywniejsze sposoby sprawdzania równoważności cyklicznej. Nietrudno jest udowodnić, że słowa *v* i *w* są równoważne cyklicznie wtedy i tylko wtedy, gdy są równej długości oraz *v* jest podsłowem *ww*. Używając dowolnego liniowego algorytmu wyszukiwania wzorca w tekście (np. algorytmu Knutha-Morrisa-Pratta), możemy sprawdzić, czy dwa słowa są równoważne cyklicznie, w czasie liniowym. Prowadzi to do rozwiązania o złożoności $O(wh + n^2d)$.

Można jednak zastosować inne podejście. Zamiast sprawdzać, czy każda para reprezentacji jest równoważna cyklicznie, możemy sprowadzić każdą reprezentację do pewnej szczególnej postaci, którą nazwiemy *reprezentacją kanoniczną*. Będzie nią maksymalny leksykograficznie równoważnik cykliczny (MLRC) reprezentacji, czyli takie słowo równoważne cyklicznie reprezentacji trasy, które jest największe w porządku leksykograficznym (słownikowym). We wspomnianym przykładzie reprezentacja kanoniczna dla 4S, 8S, 9N, 5N to 9N, 5N, 4S, 8S. Warto zauważyć, że problem wyznaczenia MLRC można łatwo zredukować do wyznaczania maksymalnego leksykograficznie sufiksu słowa. Nietrudno udowodnić, że maksymalny sufiks słowa *vv#*, gdzie # jest nowym symbolem mniejszym od wszystkich występujących w *v*, zaczyna się na pozycji wskazującej początek równoważnika cyklicznego *v*, który jest maksymalny. Do wyznaczania maksymalnego leksykograficznie sufiksu możemy zastosować algorytm Duvala, który działa w czasie liniowym.

Dwie trasy są wówczas równoważne, jeśli mają identyczne reprezentacje kanoniczne. Możemy więc wyznaczyć dla każdej trasy jej reprezentację kanoniczną, a następnie posortować wszystkie reprezentacje kanoniczne. Proste przejście po posortowanej tablicy pozwala wyznaczyć podział na klasy abstrakcji. Rozwiązanie takie działa w czasie $O(wh + nd \log n)$, jeśli zastosujemy sortowanie liniowo-logarytmiczne. Złożoność można poprawić do $O(wh + nd)$, czyli liniowej względem wielkości wejścia, stosując sortowanie kubełkowe lub sortując hasze zamiast całych reprezentacji (w tym drugim przypadku do złożoności dochodzi składnik $n \log n$).

Szkic dowodu twierdzenia o homotopii tras

Rozdział ten będzie poświęcony szkicowi dowodu poniższego faktu:

Twierdzenie. Dwie trasy są równoważne wtedy i tylko wtedy, gdy są homotopijne.

Niech $W \subseteq \mathbb{R}^2$ oznacza obszar pokryty przez wodę. Zacznijmy od przypomnienia kilku pojęć z topologii.

Definicja 1. *Pętlą* (skierowaną) w W nazywamy każdą funkcję ciągłą $p : [0,1] \to W$, taką że $p(0) = p(1)$. Intuicyjnie $p(t)$ oznacza miejsce, w którym znajdowała się ryba w chwili t.

Definicja 2. Niech p i q będą pętlami w W. *Homotopią* łączącą p i q nazywamy dowolne ciągłe odwzorowanie $F : [0,1]^2 \to W$, takie że dla dowolnych $0 \leqslant r, t \leqslant 1$ zachodzi

$$F(0,t) = p(t), \quad F(1,t) = q(t), \quad F(r,0) = F(r,1).$$

Pętle nazywamy *homotopijnymi*, jeśli istnieje homotopia je łącząca.

Lemat. Trasy przepłynięte przez rybę w dwóch kolejnych dniach są homotopijne.

Dowód. Niech p i q będą takimi trasami i niech $H : [0,1]^2 \to W$ będzie dana wzorem $H(r,t) = (1-r)p(t) + rq(t)$. Obraz H jest zawarty w W, gdyż ryba w każdym momencie widzi punkt, w którym była dokładnie dobę wcześniej, co oznacza, że odcinek łączący $p(t)$ i $q(t)$ jest zawarty w W dla dowolnego t. Łatwo sprawdzić, że H jest wówczas homotopią łączącą p i q. □

Relacja homotopii jest relacją równoważności, więc jeśli trasy przepłynięte przez jedną rybę w dwóch kolejnych dniach są homotopijne, to homotopijne są również dwie trasy przepłynięte przez nią w dowolnych dniach. Kończy to dowód implikacji „w prawo" w naszym twierdzeniu.

Zauważmy, że warunek nałożony na trasy ryb w dwóch kolejnych dniach oznacza dokładnie, że istnieje homotopia liniowa względem r je łącząca. Trasy są więc równoważne wtedy i tylko wtedy, gdy istnieje homotopia kawałkami liniowa je łącząca (dodatkowo wymagamy, aby $H(r)$ było poprawną trasą dla r będących punktami sklejenia liniowych fragmentów). Nietrudno uwierzyć, że istnienie dowolnej homotopii w W łączącej dwie trasy implikuje istnieje homotopii wymaganej postaci, czego jednak nie zamierzam dowodzić.

PIOTR CHRZĄSTOWSKI

Piotr Chrząstowski jest docentem w Instytucie Informatyki Uniwersytetu Warszawskiego. Od lat prowadzi na I roku informatyki wykład *Wstęp do programowania*. Dostrzega problemy algorytmiczne tam, gdzie inni w ogóle się ich nie spodziewają. Potem można je ujrzeć w postaci zadań na kolokwiach i egzaminach, a z najlepszymi z nich mierzą się uczestnicy Olimpiady Informatycznej. Jest członkiem Komitetu Głównego Olimpiady Informatycznej, z niewielką przerwą od początku jej powstania. Naukowo zajmuje się teorią współbieżności, a w szczególności sieciami Petriego, które wykorzystuje jako model opisu procesów biznesowych. Jest autorem wielu projektów programistycznych i programów, z bogatą praktyką w łączeniu informatyki z biznesem. Ostatnio zajmuje się też interfejsami użytkownika.
Jest przewodniczącym Jury Konkursu Młodych Naukowców Unii Europejskiej. Uwielbia grać w szachy, brydża i go. Pasjami słucha muzyki klasycznej.