Michał Włodarczyk
Treść zadania, Opracowanie

Wojciech Nadara
Program

Dostępna pamięć: 256 MB. *OI, etap I, 19.10–16.11.2015*

Hydrorozgrywka

Ekipy hydrauliczne Bajtazara i Bajtoniego wygrały przetarg na doprowadzenie wody do Bajtołów Dolnych. Sieć drogowa w tym mieście składa się z n skrzyżowań połączonych m odcinkami dróg. Z każdego skrzyżowania można dojechać do każdego innego skrzyżowania za pomocą sieci dróg. Pod każdym odcinkiem drogi należy zakopać rurę wodociągową.

Aby urozmaicić sobie pracę, Bajtazar i Bajtoni postanowili zagrać w grę. Na początek ekipy obu bohaterów stają na jednym ze skrzyżowań. Przez całą grę obie ekipy będą podążać razem. Gracze wykonują ruchy na przemian, począwszy od Bajtazara. W swoim ruchu gracz wskazuje swojej ekipie odcinek drogi (pod którym jeszcze nie ma rury) wychodzący ze skrzyżowania, na którym znajdują się obie ekipy. Ekipa gracza zakopuje rurę pod tym odcinkiem drogi i następnie obie ekipy przemieszczają się do drugiego ze skrzyżowań, które łączy ten odcinek.

Gracz, który nie może wykonać ruchu, przegrywa i za karę jego ekipa musi zakopać rury pod pozostałymi odcinkami dróg. Bajtazar zastanawia się, od którego skrzyżowania może zacząć się gra, aby był w stanie wygrać niezależnie od ruchów Bajtoniego. Poprosił Cię o pomoc w ustaleniu listy takich skrzyżowań. Dodatkowo zauważył, że sieć drogowa w Bajtołach ma ciekawą własność: **wyjeżdżając ze środka dowolnego odcinka drogi, na dokładnie jeden sposób możemy zrobić „pętlę" i wrócić do punktu wyjścia, jeśli nigdy nie zawracamy i nie odwiedzamy żadnego skrzyżowania dwukrotnie**.

Wejście

Pierwszy wiersz standardowego wejścia zawiera dwie liczby całkowite n i m oddzielone pojedynczym odstępem, oznaczające liczbę skrzyżowań i liczbę odcinków dróg. Skrzyżowania numerujemy liczbami od 1 do n. Kolejne m wierszy opisuje sieć drogową: każdy z nich zawiera dwie liczby całkowite a, b ($1 \leqslant a, b \leqslant n$, $a \neq b$) oddzielone pojedynczym odstępem, oznaczające, że skrzyżowania o numerach a i b są połączone odcinkiem drogi. Możesz założyć, że żadne dwa skrzyżowania nie są połączone więcej niż jednym odcinkiem drogi.

Wyjście

Na standardowe wyjście należy wypisać dokładnie n wierszy: i-ty z nich ma zawierać liczbę 1, jeśli Bajtazar może wygrać, gdy gra zacznie się ze skrzyżowania numer i; w przeciwnym wypadku ma zawierać liczbę 2.

Przykład

Dla danych wejściowych:	*poprawnym wynikiem jest:*
`6 7`	`1`
`1 2`	`1`
`2 3`	`1`
`3 1`	`2`
`3 4`	`1`
`4 5`	`2`
`5 6`	
`6 3`	

Testy „ocen”:

`1ocen`: *$n = 9$, $m = 12$, sieć drogowa składa się z czterech „pętli”: 1–2–3–1, 1–4–5–1, 1–6–7–1 oraz 1–8–9–1.*

`2ocen`: *$n = 998$, $m = 999$, dla każdego j takiego, że $1 \leqslant j < n$, istnieje odcinek drogi pomiędzy j-tym i $(j + 1)$-wszym skrzyżowaniem; istnieją również odcinki drogi łączące skrzyżowania 1 z 499 oraz 499 z 998.*

`3ocen`: *$n = 500\,000$, $m = 500\,000$, sieć dróg tworzy jedną „pętlę”.*

Ocenianie

Zestaw testów dzieli się na następujące podzadania. Testy do każdego podzadania składają się z jednej lub większej liczby osobnych grup testów.

Podzadanie	Warunki	Liczba punktów
1	$3 \leqslant n, m \leqslant 20$	*21*
2	$3 \leqslant n, m \leqslant 1000$	*39*
3	$3 \leqslant n, m \leqslant 500\,000$	*40*

Rozwiązanie

Hydrorozgrywka to jedno z tych zadań, które trafiają na I etap Olimpiady głównie dlatego, że wymyślenie rozwiązania i ominięcie wszystkich pułapek w implementacji w ciągu zaledwie pięciu godzin wymagałoby nadludzkich umiejętności. Z drugiej strony część doświadczonych zawodników umie uporać się ze standardowymi zadaniami z I etapu w kilka dni i wielką stratą byłoby zmarnować ich chęci do spędzenia tygodnia walki z bardziej wymagającym przeciwnikiem. Autor podpisuje się pod tezą, że najlepszym sposobem treningu w takich dziedzinach jak algorytmika jest podejmowanie *nieco za trudnych* wyzwań.

Na początek zinterpretujmy grę w układanie rur w języku teorii grafów. Zaczynając od ustalonego wierzchołka v_0, gracze na przemian wybierają krawędzie tak, aby każda kolejna miała wspólny wierzchołek z poprzednią. Każda krawędź może zostać wybrana co najwyżej raz. Innymi słowy, gracze w trakcie rozgrywki budują *marszrutę*

w grafie. Krawędzie na marszrucie nie powtarzają się, jednak ten sam wierzchołek może na niej wystąpić wielokrotnie. Gracz, który nie może wykonać poprawnego ruchu, przegrywa.

Tak zdefiniowana gra jest skończona, ponieważ liczba ruchów jest ograniczona przez liczbę krawędzi w grafie. Liczba stanów tej gry jest wykładnicza ze względu na liczbę krawędzi. Stany w grze zadane są przez marszruty zaczynające się w wierzchołku v_0, natomiast każde przejście pomiędzy stanami odpowiada wydłużeniu marszruty o jedną krawędź. Łatwo zauważyć, że taka gra jest *zdeterminowana*, tzn. dla każdego stanu jeden z graczy może doprowadzić do zwycięstwa, niezależnie od tego, jakie ruchy będzie wykonywał jego rywal. Mówimy, że gracz X posiada *strategię wygrywającą*, jeśli powyższy warunek zachodzi dla X w stanie początkowym.

Najprostszy algorytm obliczający strategie wygrywające przegląda cały graf stanów gry i klasyfikuje stany na wygrywające i przegrywające. Jest to zazwyczaj mało praktyczne podejście z powodu ogromnej liczby stanów do przeanalizowania. Częściowe rozwiązanie tego problemu opiera się na obserwacji, że niektóre stany gry są równoważne. Zauważmy, że informacja o tym, które ruchy będzie można wykonać w przyszłości, jest w całości zakodowana przez zbiór dotąd wykorzystanych krawędzi oraz aktualny końcowy wierzchołek marszruty. Możemy zatem rozważać stany opisane przez podzbiór krawędzi grafu i jeden wierzchołek. Aby sprawdzić, czy pierwszy gracz ma strategię wygrywającą w grze rozpoczynającej się w wierzchołku v_0, wystarczy sprawdzić wartość obliczoną dla stanu $(\emptyset, v_0)$. Liczba stanów zostaje w ten sposób ograniczona do $O(n2^m)$ i tyle też wynosi złożoność pamięciowa naiwnego algorytmu. Rozwiązanie oparte na tym podejściu (zaimplementowane w pliku `hyds1.cpp`) działa w czasie $O(m2^m)$. Na zawodach było ono warte 21 punktów. Nie korzysta ono jednak w żaden sposób ze specjalnej struktury grafu.

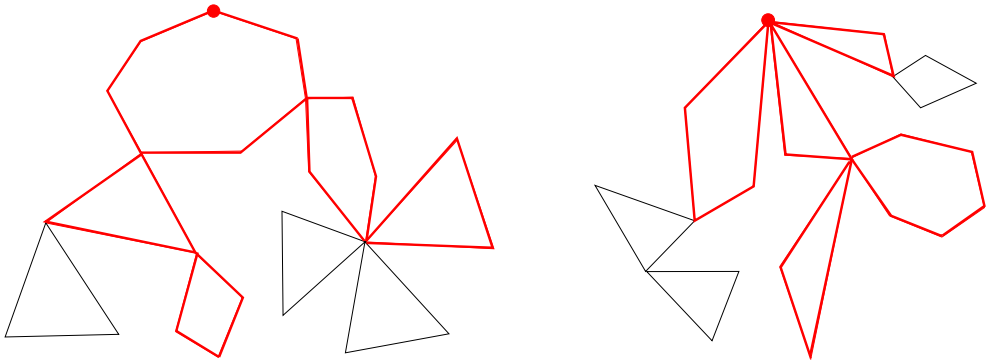
Cykle, marszruty i kaktusy

Poza gwarancją spójności grafu w treści zadania pojawia się pozornie niewiele mówiący warunek: *wyjeżdżając ze środka dowolnego odcinka drogi, na dokładnie jeden sposób możemy zrobić „pętlę" i wrócić do punktu wyjścia, jeśli nigdy nie zawracamy i nie odwiedzamy żadnego skrzyżowania dwukrotnie.* Równoważnie możemy napisać, że każda krawędź leży na dokładnie jednym *cyklu prostym*, gdzie cykl prosty to cykl, który przechodzi przez każdy wierzchołek co najwyżej jednokrotnie. Jako że często będziemy używać tego pojęcia, zamiast *cykl prosty* pisać będziemy po prostu *cykl*.

Grafy spełniające warunek z zadania będziemy nazywać *kaktusami* ze względu na podobieństwo ich graficznych reprezentacji do krzewu opuncji. Jako że rysunki potrafią czasem prowadzić do mylnych intuicji, udowodnijmy formalnie kilka własności kaktusów.

Po pierwsze, jeśli marszruta prowadzi po cyklu C, następnie opuszcza go w wierzchołku v_i, po czym wraca na C, wchodząc do wierzchołka v_j, to $v_i = v_j$.

Lemat 1. Rozważmy marszrutę $(v_i)_{i=0}^{k}$ w kaktusie. Ustalmy dowolny cykl prosty C. Jeśli $v_i \in C$, $v_{i+1} \notin C$, a v_j jest następnym po v_i wierzchołkiem należącym do C, to $v_i = v_j$.



Rys. 1: Przykłady kaktusów z wyróżnioną potencjalną marszrutą gry.

Dowód: Przypuśćmy $v_i \neq v_j$. Na podstawie cyklu C konstruujemy cykl prosty C', zastępując odcinek C pomiędzy v_i oraz v_j przez ścieżkę $v_i, v_{i+1}, \ldots, v_j$ z usuniętymi pętlami. Otrzymujemy dwa różne cykle proste C i C', które mają co najmniej jedną wspólną krawędź. Dla tej krawędzi nie jest spełniony warunek z zadania, który definiuje kaktus. ■

Lemat 2. Cykle proste kaktusa są krawędziowo rozłączne.

Dowód: Przypuśćmy, że dwa różne cykle proste C_1, C_2 posiadają wspólną krawędź. Zatem istnieje ścieżka łącząca dwa wierzchołki v_i i v_j należące do cyklu C_1, składająca się z krawędzi cyklu C_2, które nie należą do C_1. Gdyby $v_i = v_j$, to ponieważ cykl C_2 jest prosty, ścieżka ta przebiegałaby po całym cyklu C_2. Wówczas jednak C_2 nie miałby krawędzi wspólnych z C_1. Z kolei gdy $v_i \neq v_j$, otrzymujemy ścieżkę przeczącą tezie lematu 1. Zatem w obydwóch przypadkach otrzymujemy sprzeczność. ■

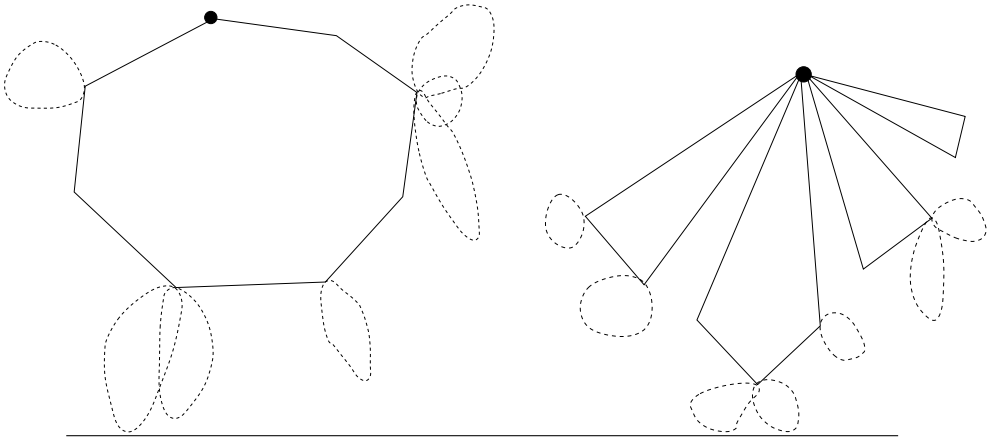
Lemat 3. Wszystkie wierzchołki kaktusa mają parzysty stopień.

Dowód: Indukcja po liczbie cykli prostych w grafie. Kaktus składający się tylko z jednego cyklu oczywiście spełnia tezę lematu. Przypuśćmy zatem, że spełniają ją kaktusy o liczbie cykli mniejszej od k, i rozważmy kaktus o k cyklach. Usuńmy z niego krawędzie dowolnego cyklu. W ten sposób stopnie niektórych wierzchołków zmniejszą się o 2, co nie wpływa na ich parzystość.

Liczba cykli przechodzących przez dowolną zachowaną krawędź na pewno nie wzrosła, a istniejący cykl nie posiadał części wspólnej z usuniętym cyklem na mocy lematu 2. Powstały graf, choć potencjalnie niespójny, nadal jest kaktusem i, z założenia indukcyjnego, wszystkie jego wierzchołki mają parzysty stopień. Zatem po przywróceniu usuniętego cyklu wszystkie wierzchołki mają nadal parzysty stopień. ■

Lemat 4. Każda rozgrywka na kaktusie kończy się w wierzchołku początkowym (patrz rys. 1).

Dowód: Oznaczmy wierzchołek początkowy przez v_0. Niech v będzie wierzchołkiem, po dojściu do którego gracz nie może wykonać ruchu. Jeśli $v \neq v_0$, to każde przejście przez wierzchołek v „zużywa” dwie krawędzie incydentne z v. Z lematu 3 wiemy, że



Rys. 2: Po lewej: kaktus prosty z zaznaczonymi przerywaną linią wewnętrznymi pękami. Po prawej: pęk składający się z czterech kaktusów prostych.

stopień v jest parzysty. Zatem tuż po wejściu do wierzchołka v mamy do wyboru nieparzystą (a więc niezerową) liczbę krawędzi incydentnych z v. ■

Lematy 1 oraz 4 charakteryzują marszruty, jakie mogą pojawić się w trakcie gry. Idąc po cyklu zawierającym v_0, gracze mogą zdecydować, czy przejść do innego cyklu. Tam rozpoczyna się wewnętrzna rozgrywka, która (z lematu 1) skończy się w tym samym punkcie. Jedyną istotną informację stanowi to, który z graczy będzie wykonywał ruch po powrocie na pierwotny cykl.

Jak się dobrać do kaktusa?

Aby móc wykorzystać specjalną strukturę grafu, musimy umieć reprezentować kaktusy w wygodny sposób; patrz rys. 2.

Definicja 1. Kaktusem prostym ukorzenionym w wierzchołku v_0 nazwiemy kaktus z wyróżnionym wierzchołkiem v_0 (korzeniem) o stopniu 2.

Definicja 2. Pękiem nazwiemy rodzinę kaktusów prostych ukorzenionych w tym samym wierzchołku v_0. Kaktusy tworzące pęk są rozłączne poza posiadaniem wspólnego korzenia. W szczególności, pęk może składać się z jednego kaktusa prostego.

Każdy kaktus prosty możemy reprezentować rekurencyjnie jako cykl przechodzący przez v_0, zawierający informacje o pękach ukorzenionych w wierzchołkach cyklu. W ten sposób otrzymujemy drzewiastą strukturę danych dla kaktusa ukorzenionego w konkretnym wierzchołku. Poniżej przedstawiamy pseudokod zmodyfikowanej procedury *DFS* (*Depth First Search*), tłumaczącej standardową reprezentację kaktusa przy pomocy list sąsiedztwa na postać rekurencyjną. Jako argumenty przekazywane są numer analizowanego i poprzednio odwiedzonego wierzchołka, zwanego *rodzicem*. W tablicy dynamicznej *stack* trzymamy stos odwiedzonych wierzchołków i kiedy trafimy do wierzchołka znajdującego się już na stosie, zapamiętujemy znaleziony cykl.

Lista *cycles*[v] zawiera pęk kaktusów ukorzenionych w v, natomiast w tablicy *belong* zapamiętujemy dla każdego wierzchołka, przez który cykl można do niego trafić, idąc od korzenia. Nie używamy tej informacji bezpośrednio w omówieniu, jednak jest ona przydatna w implementacji.

```
 1: function DFS(v, parent)
 2: begin
 3:   stack.push(v);
 4:   onStack[v] := true;
 5:   visited[v] := true;
 6:   for w ∈ neighbors[v] do begin
 7:     if onStack[w] and (w ≠ parent) then begin
 8:       C := new Cycle;
 9:       C.insert(w);
10:       i := stack.size() − 1;
11:       while stack[i] ≠ w do begin
12:         C.insert(stack[i]);
13:         belong[stack[i]] := C;
14:         i := i − 1;
15:       end
16:       cycles[w].insert(C);
17:     end
18:     if not visited[w] then
19:       DFS(w, v);
20:   end
21:   stack.pop(v);
22:   onStack[v] := false;
23: end
```

Należy zaznaczyć, że korzeń DFS-a, tzn. wierzchołek, od którego zaczęliśmy przeszukiwanie grafu, pozostanie z pustym polem w tablicy *belong* i należy rozważyć go jako przypadek szczególny. Można tego uniknąć, reprezentując graf jako kaktus prosty, jeśli rozpoczniemy przeszukiwanie grafu w wierzchołku stopnia 2 (zachęcamy Czytelnika do udowodnienia, że taki wierzchołek istnieje w każdym kaktusie) i przypiszemy go do jedynego cyklu, który z niego wyrasta.

Klasyfikacja kaktusów

W wielu teoriach matematycznych centralne miejsce zajmują twierdzenia o klasyfikacji, rozstrzygające, które obiekty możemy traktować jako równoważne, a pomiędzy którymi zachodzą istotne strukturalne różnice. Takie twierdzenia przydadzą się nam, aby uprościć opis rozwiązania wzorcowego.

Gracza rozpoczynającego rozgrywkę w interesującym nas podkaktusie nazwiemy graczem I, a jego rywala – graczem II. Pęki podzielimy na wygrywające i przegrywające, w zależności od tego, czy gracz I może zagwarantować sobie zebranie ostatniej krawędzi w grze ograniczonej do rozważanego pęku. W przypadku kaktusów prostych sytuacja jest (wbrew nazwie) bardziej skomplikowana, ponieważ niekiedy graczowi I

opłaca się zmusić rywala do wzięcia ostatniej krawędzi, aby samemu kontynuować grę w korzystnym dla siebie stanie. Dlatego wprowadzimy dwa rodzaje gier na kaktusach prostych: rodzaj A, w którym wygrywa gracz zabierający ostatnią krawędź, oraz rodzaj B, w którym taki gracz przegrywa. W zależności od istnienia strategii wygrywających w tych grach, kaktus prosty może należeć do jednego z czterech typów: **00**, **10**, **01**, **11**, gdzie kolejne bity kodują odpowiednio wynik gry A oraz B – naturalnie 1 oznacza zwycięstwo[1]. Przykładowo, typ **10** oznacza, że w grze rodzaju A gracz I ma strategię wygrywającą, a w grze rodzaju B to gracz II ma strategię wygrywającą. Zauważmy, że dla pęków rozważamy tylko grę rodzaju A i takiej gry dotyczy oryginalne pytanie z zadania.

Twierdzenie 1. *Rozważmy kaktus prosty ukorzeniony w v_0. Niech C będzie cyklem zawierającym v_0. Wówczas kaktus jest typu:*

10, *jeśli na cyklu C nie ma wygrywających pęków oraz C jest nieparzystej długości,*

01, *jeśli na cyklu C nie ma wygrywających pęków oraz C jest parzystej długości,*

11, *jeśli wychodząc z v_0 w pewną stronę, dojdziemy do wygrywającego pęku po parzystej liczbie kroków,*

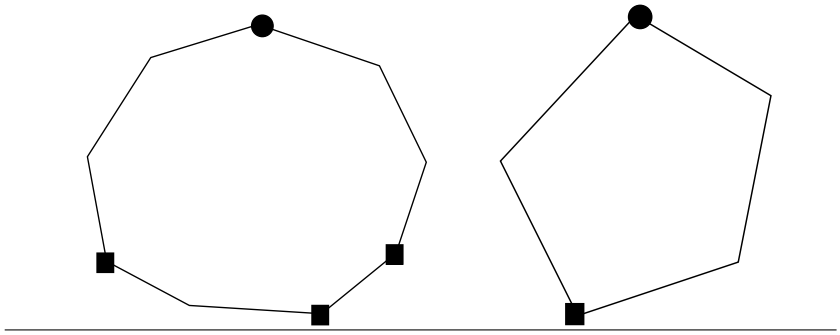
00, *jeśli wychodząc z v_0 w dowolną stronę, dojdziemy do wygrywającego pęku po nieparzystej liczbie kroków.*

Dowód: Załóżmy wpierw, że na cyklu C nie znajduje się żaden wygrywający pęk. W przypadku, gdy C jest nieparzystej długości, zaznaczmy na C krawędzie, których odległość od v_0 jest parzysta. W ten sposób zaznaczone zostaną obydwie krawędzie incydentne z v_0, a poza nimi co druga krawędź na C. Gracz I może zagwarantować sobie wzięcie wszystkich zaznaczonych krawędzi. Gracz II może potencjalnie starać się mu przeszkodzić i zamiast zebrać niezaznaczoną krawędź na C, rozpocząć wewnętrzną rozgrywkę na pęku ukorzenionym w pewnym wierzchołku v. Aby zmusić gracza I do wybrania niezaznaczonej krawędzi, gracz II musi doprowadzić do sytuacji, w której gracz I będzie miał ruch w v i wszystkie kaktusy proste wyrastające z v będą już odwiedzone. Jednak aby do tego doprowadzić, gracz II musiałby mieć strategię wygrywającą w pęku ukorzenionym w v.

Z drugiej strony, gdyby gracz I chciał zmienić kolejność ruchów i zmusić gracza II do wzięcia ostatniej krawędzi, potrzebowałby strategii wygrywającej na jednym z pęków. Zatem również gracz II jest w stanie zagwarantować, że gracz I wykona ostatni ruch na cyklu C. Analogicznie, jeżeli cykl C jest parzystej długości, gracz II potrafi zagwarantować sobie jedynie wzięcie ostatniej krawędzi, natomiast gracz I – przedostatniej. W ten sposób udało nam się sklasyfikować dwa pierwsze przypadki.

W dwóch pozostałych przypadkach w co najmniej jednym wierzchołku cyklu C ukorzeniony jest wygrywający pęk. Gracz, który jako pierwszy będzie wykonywał ruch w takim wierzchołku – oznaczmy ten wierzchołek przez v – jest w stanie zdeterminować dalszy przebieg rozgrywki na cyklu. Jako że jedna z krawędzi idących do v jest wykorzystana, o wyniku rozgrywki decyduje, który z graczy opuści wierzchołek v.

[1] W filmowym omówieniu zadania pojawiły się alternatywne definicje typów kaktusów. Kaktusy typu 01, 10 to takie, w których gracz zabierający ostatnią krawędź jest zdeterminowany, zaś typy 11, 00 odpowiadają kaktusom, w których o finale rozgrywki decyduje odpowiednio gracz I lub II. Po przemyśleniu sprawy, autor uważa, że obecna notacja jest prostsza i bardziej precyzyjna.



Rys. 3: Ilustracja do twierdzenia 1: korzeń zaznaczono kółkiem, a wierzchołki zawierające wygrywające pęki – kwadratami. Po lewej stronie kaktus typu **00**, po prawej **11**.

Jeśli ruch ten prowadzi do stanu ze strategią wygrywającą w grze A bądź B, gracz wykonujący ruch może ten ruch wykonać od razu. W przeciwnym przypadku, może rozpocząć grę na wygrywającym pęku i zagwarantować sobie wzięcie ostatniej krawędzi. Drugi gracz może próbować opuścić grę na pęku przed odwiedzeniem wszystkich wewnętrznych kaktusów, lecz w ten sposób również zbierze niechcianą krawędź.

Załóżmy, że wychodząc z v_0 w pewną stronę, można w parzystej liczbie kroków dojść do korzenia wygrywającego pęku. Oznaczmy ten korzeń przez v. Gracz I rozpocznie grę w tym właśnie kierunku i zagwarantuje sobie wzięcie krawędzi o nieparzystych indeksach, co zapewni mu ruch w wierzchołku v i zwycięstwo w obu rodzajach gry. Jeżeli zaś idąc w obu kierunkach, pierwszy wygrywający pęk znajdziemy po nieparzystej liczbie kroków, to niezależnie od początkowego ruchu, gracz II zagwarantuje sobie wzięcie krawędzi o parzystych indeksach i to on będzie miał ruch przy wygrywającym pęku. Powyższa obserwacja kończy analizę ostatnich dwóch przypadków. ■

Wynik gry na pęku zależy tylko od liczby kaktusów poszczególnych typów. Czwórkę liczb $(x_{00}, x_{10}, x_{01}, x_{11})$ opisującą, ile kaktusów typu odpowiednio **00**, **10**, **01**, **11** znajduje się w pęku, nazwiemy *konfiguracją*. Naturalnym pomysłem jest wykorzystanie programowania dynamicznego do pogrupowania konfiguracji na wygrywające i przegrywające. Aby rozstrzygnąć, jakiego typu jest dana konfiguracja, można rozważyć wszystkie ruchy gracza I, polegające na wyborze początkowego kaktusa i rodzaju wewnętrznej gry (A lub B), i przyjrzeć się wcześniej obliczonym wynikom gier dla konfiguracji z pewnym x_i pomniejszonym o 1. Strategią wygrywającą może być rozpoczęcie gry A w kaktusie gwarantującym zwycięstwo, jeśli prowadzi to do przegrywającej konfiguracji, lub podjęcie gry B, kiedy otrzymana konfiguracja zapewnia zwycięstwo. Niestety liczba konfiguracji jest rzędu $\Theta(n^4)$, co wymusza na nas znalezienie sprytniejszego sposobu klasyfikacji pęków. Takiego sposobu dostarcza poniższe kryterium.

Twierdzenie 2. *Pęk jest wygrywający wtedy i tylko wtedy, gdy zawiera kaktus prosty typu* **11** *lub liczba kaktusów typu* **10** *jest nieparzysta.*

Dowód: Indukcja po liczbie kaktusów w pęku. Konfiguracja (0,0,0,0) jest zgodnie z definicją przegrywająca, ponieważ gracz I nie może zagwarantować sobie zabrania

ostatniej krawędzi. Sytuacja jest równoważna wzięciu ostatniej krawędzi przez gracza II, ponieważ wewnętrzna gra na pęku jest zakończona i ruch należy do gracza I. Załóżmy teraz, że rozważamy konfigurację $(x_{00}, x_{10}, x_{01}, x_{11})$ i teza indukcyjna zachodzi dla wszystkich pęków składających się z mniej niż $x_{00}+x_{10}+x_{01}+x_{11}$ kaktusów.

Jeżeli pęk zawiera kaktus typu **11**, to gracz I może na nim zagrać zarówno tak, by wziąć ostatnią krawędź, jak i tak, by ostatnią krawędź wziął gracz II. W zależności od tego, czy konfiguracja $(x_{00}, x_{10}, x_{01}, x_{11}-1)$ jest wygrywająca, czy przegrywająca, wybierze on odpowiednią strategię i w ten sposób zapewni sobie zwycięstwo. Jeśli zaś $x_{11}=0$, ale x_{10} jest nieparzyste, strategią gracza I jest takie granie na kaktusie typu **10**, by wziąć ostatnią krawędź. Wówczas gracz II będzie zmuszony wykonać ruch w konfiguracji $(x_{00}, x_{10}-1, x_{01}, 0)$. Jednak zgodnie z założeniem indukcyjnym taka konfiguracja jest przegrywająca.

Pozostaje nam pokazać, że pozostałe konfiguracje są przegrywające. Przypuśćmy $x_{11}=0$ oraz $2 \mid x_{10}$. Jeśli gra rozpocznie się w kaktusie typu **10**, gracz II zadba o to, by gracz I wziął ostatnią krawędź. Prowadzi to do konfiguracji, w której x_{10} jest nieparzyste i rozpoczyna gracz II, co gwarantuje mu zwycięstwo. Jeśli zaś wybrany zostanie kaktus typu **00** lub **01**, wówczas gracz II może sprawić, że weźmie on ostatnią krawędź, przez co gracz I znajdzie się w konfiguracji $(x_{00}-1, x_{10}, x_{01}, 0)$ lub, odpowiednio, $(x_{00}, x_{10}, x_{01}-1, 0)$, która zgodnie z tezą indukcyjną jest przegrywająca. ■

Uzbrojeni w reprezentację cyklową grafu, możemy z łatwością rekurencyjnie sklasyfikować wszystkie pęki i kaktusy proste. Twierdzenia 1 i 2 pozwalają nam wykonać wszystkie obliczenia w złożoności obliczeniowej $O(m)$ i odczytać, kto posiada strategię wygrywającą w grze rozpoczynającej się w korzeniu kaktusa. To jednak nie koniec zadania – wszak w treści jesteśmy proszeni o rozstrzygnięcie wyniku gry dla wszystkich możliwych wierzchołków początkowych. Możemy obliczyć całą dekompozycję cyklową dla wszystkich możliwych wierzchołków w grafie, lecz takie rozwiązanie obarczone jest złożonością obliczeniową[2] $O(nm)$. Za zaimplementowanie tego algorytmu można było otrzymać 60 punktów.

Rozwiązanie wzorcowe

Kluczem do przyspieszenia obliczeń jest dogłębne wykorzystanie raz obliczonej dekompozycji cyklowej. Przypomnijmy, że ma ona rekurencyjną strukturę drzewiastą. Kaktus przedstawiamy jako cykl, w którego wierzchołkach zaczepione są inne kaktusy proste (być może wiele w tym samym wierzchołku). Każdy taki kaktus prosty będziemy nazywać *podkaktusem* przez analogię do poddrzewa. Za pomocą twierdzeń 1 i 2 umiemy rozstrzygnąć, jak potoczy się gra rozpoczynająca się w korzeniu v pewnego podkaktusa, o ile znamy typy podkaktusów poniżej v. Ostatecznie w ten sposób otrzymujemy wynik dla wierzchołka startowego v_0, w którym ukorzeniony jest cały kaktus.

Obliczenie wyniku gry rozpoczynającej się gdzie indziej niż w v_0 jest nieco trudniejsze, bo musimy wziąć pod uwagę kaktus, który zawiera v_0. Ustalmy wierzchołek

[2] W istocie dla każdego kaktusa zachodzi $m \leqslant \frac{3}{2}n$ (zachęcamy Czytelnika do przekonania się o tym osobiście), zatem złożoność obliczeniową tego rozwiązania można oszacować jako $O(n^2)$.

$v \neq v_0$ będący korzeniem pewnego podkaktusa. Aby rozstrzygnąć, jak potoczy się gra rozpoczynająca się w v, musimy znać typy wszystkich kaktusów ukorzenionych w v, jak również typ „nadkaktusa" v. Mówiąc formalnie, *nadkaktus* v powstaje przez usunięcie podkaktusów ukorzenionych w v (z wyjątkiem samego wierzchołka v) i następnie ukorzenienie powstałego grafu właśnie w v.

Potrzebujemy zatem wykonać dwa dodatkowe kroki: ustalić typy wszystkich nadkaktusów oraz wykorzystać te informacje do obliczenia ostatecznej odpowiedzi. Algorytm zaprezentowany został na poniższym pseudokodzie. Procedura *analyze* zostaje wywołana rekurencyjnie dla każdego podkaktusa podanego grafu. Pierwszy argument to cykl C zawierający korzeń podkaktusa (w implementacji może to być numer lub wskaźnik do obiektu). Cykl reprezentowany jest jako tablica wierzchołków, gdzie $C[0]$ to korzeń podkaktusa. Drugi argument procedury określa typ nadkaktusa (**00**, **10**, **01** lub **11**). W pierwszym wywołaniu C to cykl zaczynający się od v_0, a $overcactusType = \mathbf{00}$.

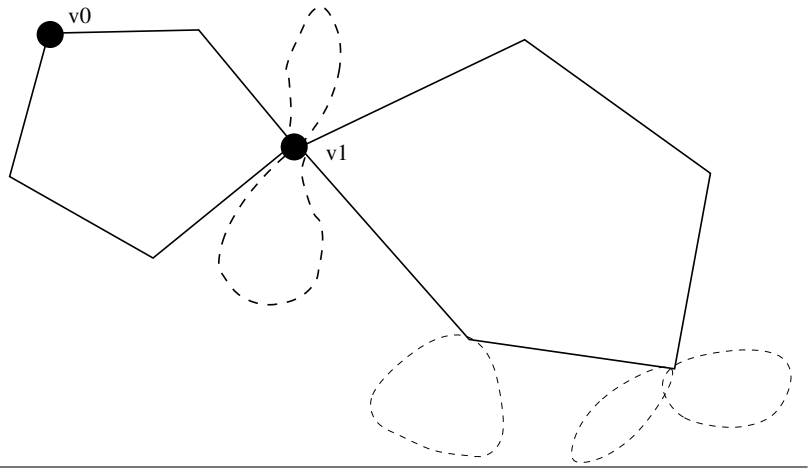
```
 1: function analyze(C, overcactusType)
 2: begin
 3:   externalGame[C[0]] := getGameResult(cycles[C[0]] \ C, overcactusType);
 4:   for i := 1 to C.size() − 1 do
 5:     externalGame[C[i]] := getGameResult(cycles[C[i]], 00);
 6:   cycleType := getCycleTypes(C, externalGame);
 7:   for i := 1 to C.size() − 1 do begin
 8:     result[C[i]] := getGameResult(cycles[C[i]], cycleType[C[i]]);
 9:     foreach C' ∈ cycles[C[i]] do
10:       analyze(C', cycleType[C[i]]);
11:   end
12: end
```

Pierwszy krok (wiersze 3-5) to wypełnienie tablicy *externalGame*, w której dla każdego wierzchołka $v \in C$ zapisujemy wynik gry rozpoczynającej się w v, przy założeniu, że z grafu usunęliśmy krawędzie cyklu C. Innymi słowy, dla każdego v rozpatrujemy jedynie kaktusy *wyrastające* z v.

Korzystamy tu z funkcji *getGameResult*, która oblicza (na podstawie twierdzenia 2), czy dany pęk jest wygrywający. Pierwszy argument funkcji to zbiór cykli stanowiących wyjściowe cykle wszystkich kaktusów w pęku poza jednym. Drugi argument to typ ostatniego kaktusa w pęku. W wierszu 5 nie potrzebujemy uwzględniać typu żadnego dodatkowego kaktusa, dlatego jako argument przekazujemy typ **00**, który nie wpływa na wynik.

W drugim kroku (wiersz 6), dla każdego $v \in C$ rozpatrujemy graf powstały przez usunięcie wszystkich kaktusów wyrastających z v. Wynik gry rozpoczynającej się w v zapisujemy w $cycleType[v]$. Za wykonanie tego kroku odpowiada funkcja *getCycleTypes* zaimplementowana na podstawie twierdzenia 1.

W ostatnim kroku łączymy wcześniej obliczone wartości, używając funkcji *getGameResult* (wiersz 8). Wyniki obliczone w tablicy *result* stanowią końcowe odpowiedzi dla wszystkich wierzchołków początkowych. Warto zaznaczyć, że wartość $result[v]$ jest obliczana podczas analizy cyklu $belong[v]$ (z wyjątkiem wyniku dla korzenia całego kaktusa, który musimy obliczyć osobno). Następnie wywołujemy pro-



Rys. 4: Cykl C widać na prawo od wierzchołka v_1. Wszystkie cykle wewnętrzne C niewyrastające z v_1 zostaną podstawione pod zmienną C' w wierszu 9. Kaktus po lewej wyrastający w stronę korzenia v_0 został narysowany ciągłą linią.

cedurę *analyze* rekurencyjnie dla wszystkich wewnętrznych cykli. Zauważmy, że typ nadkaktusa v jest równy $cycleType[v]$.

Kilku słów komentarza wymaga jeszcze funkcja *getCycleTypes*. Bezpośrednia implementacja twierdzenia 1 może działać w czasie $O(d^2)$, gdzie d równa się długości cyklu, która może być tego samego rzędu co n. Aby tego uniknąć, należy obliczyć odległość od najbliższego wygrywającego pęku w lewo oraz w prawo przy pomocy programowania dynamicznego. Przypadkiem szczególnym, na który trzeba uważać, jest cykl z tylko jednym pękiem wygrywającym, zakorzenionym w v. Patrząc z perspektywy v, na cyklu nie ma pęków wygrywających, zaś dla pozostałych wierzchołków taki kaktus będzie typu **00** lub **11**.

Ostatnia uwaga: w przedstawionych pseudokodach beztrosko przekazywaliśmy tablice jako parametry funkcji, jednak należy zadbać o to, by przekazywanymi obiektami były jedynie indeksy lub wskaźniki. W przeciwnym razie kopiowanie obiektów może prowadzić do złożoności obliczeniowej $\Omega(n^2)$. Podobnie może się skończyć przekazywanie do funkcji *getGameResult* tablicy cykli zamiast operowania na samych konfiguracjach. Poradzenie sobie z tą ostatnią pułapką pozwala na osiągnięcie złożoności liniowej i taki algorytm można znaleźć w pliku `hyd.cpp`.