Jakub Radoszewski
Treść zadania, Opracowanie

Jakub Radoszewski
Program

Dostępna pamięć: 512 MB.

OI, etap I, 7.10–4.11.2013

Wąż

Na planszy o rozmiarze $3 \times n$ leży wąż. Kolejne fragmenty węża są ponumerowane od 1 do $3n$. Fragmenty o kolejnych numerach (tj. 1 i 2, 2 i 3, 3 i 4...) znajdują się na polach planszy sąsiadujących bokiem. Przykładowo, na planszy rozmiaru 3×9 wąż może leżeć tak:

7	6	5	4	17	18	19	20	21
8	1	2	3	16	15	26	25	22
9	10	11	12	13	14	27	24	23

Niektóre spośród pól zajmowanych przez węża zostały zamazane. Czy potrafisz odtworzyć układ węża?

Wejście

W pierwszym wierszu standardowego wejścia znajduje się jedna liczba całkowita n ($1 \leqslant n \leqslant 1000$), oznaczająca długość planszy. Kolejne trzy wiersze zawierają opis planszy; i-ty z nich zawiera n liczb całkowitych a_{ij} ($0 \leqslant a_{ij} \leqslant 3n$ dla $1 \leqslant j \leqslant n$). Jeśli $a_{ij} > 0$, to a_{ij} oznacza numer fragmentu węża znajdującego się na j-tym polu i-tego wiersza planszy. Jeśli natomiast $a_{ij} = 0$, to numer fragmentu węża znajdującego się na rozważanym polu nie jest znany.

W testach wartych łącznie 15% punktów zachodzi warunek $n \leqslant 10$, w testach wartych łącznie 40% punktów zachodzi warunek $n \leqslant 40$, a w testach wartych łącznie 70% punktów zachodzi warunek $n \leqslant 300$.

Wyjście

Twój program powinien wypisać na standardowe wyjście trzy wiersze. W i-tym wierszu powinno znajdować się n liczb całkowitych dodatnich b_{ij} (dla $1 \leqslant j \leqslant n$). Wszystkie liczby b_{ij} łącznie powinny stanowić permutację liczb od 1 do $3n$. Układ liczb na wyjściu powinien odtwarzać możliwe położenie węża zgodne z danymi wejściowymi.

Możesz założyć, że istnieje co najmniej jeden sposób odtworzenia położenia węża na planszy. Jeśli jest więcej niż jedno rozwiązanie, Twój program może wypisać dowolne z nich.

Przykład

Dla danych wejściowych:

```
9
0 0 5 0 17 0 0 0 21
8 0 0 3 16 0 0 25 0
0 0 0 0 0 0 0 0 23
```

jednym z poprawnych wyników jest:

```
7 6 5 4 17 18 19 20 21
8 1 2 3 16 15 26 25 22
9 10 11 12 13 14 27 24 23
```

Rozwiązanie

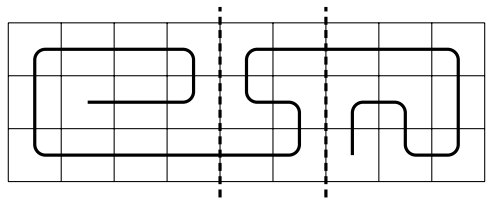
Spróbujmy na początek sformułować treść zadania w języku teorii grafów. Planszę $3 \times n$ możemy wyobrazić sobie jako graf nieskierowany, którego wierzchołkami są pola planszy. Krawędź między wierzchołkami występuje wtedy, gdy pola odpowiadające tym wierzchołkom sąsiadują bokiem. Ułożenie węża na planszy odpowiada wówczas pewnej *ścieżce Hamiltona* w tym grafie, czyli ścieżce przechodzącej przez każdy wierzchołek grafu dokładnie raz.

Niektóre pola planszy mają przypisane parami różne numery z zakresu od 1 do $3n$. Numer pola oznacza, którym z kolei wierzchołkiem na szukanej ścieżce Hamiltona jest to pole. Poszukujemy zatem jakiejkolwiek ścieżki Hamiltona *zgodnej z numeracją wierzchołków*. W treści zadania znajdujemy zapewnienie, że taka ścieżka istnieje.

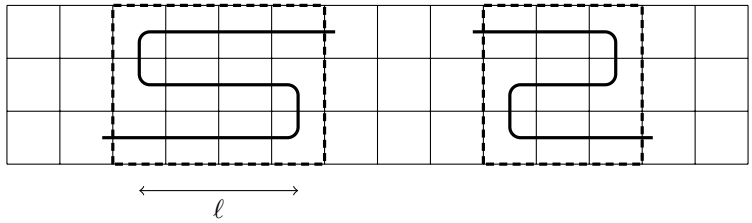
Jak może leżeć wąż?

Aby zbliżyć się do rozwiązania, warto przyjrzeć się temu, jak wyglądają różne ścieżki Hamiltona na kratce $3 \times n$, na razie nie biorąc pod uwagę numeracji wierzchołków.

Najpierw rozważmy sytuację, w której planszę da się podzielić pionowym cięciem wzdłuż linii kratki na fragmenty, między którymi ścieżka przechodzi tylko raz. Poniższy rysunek pokazuje przykład takiej ścieżki zaczerpnięty z treści zadania, z dwiema możliwymi pozycjami cięcia.

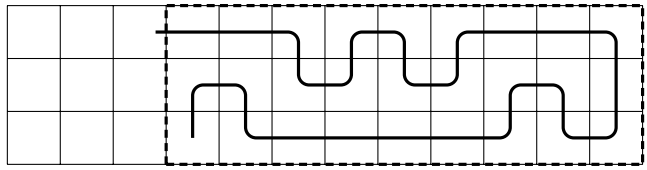


Po wykonaniu wszystkich takich cięć ścieżka Hamiltona całej planszy rozpada się na ścieżki Hamiltona krótszych fragmentów planszy (w powyższym przykładzie są to trzy ścieżki). Mamy zatem dwa skrajne fragmenty planszy oraz pewne fragmenty planszy w środku. Te drugie spełniają dodatkowy warunek, że początek ścieżki znajduje się w nich na lewym boku fragmentu, a koniec na prawym boku. Po chwili rysowania widać, że taka ścieżka Hamiltona musi mieć postać *zygzaka*:

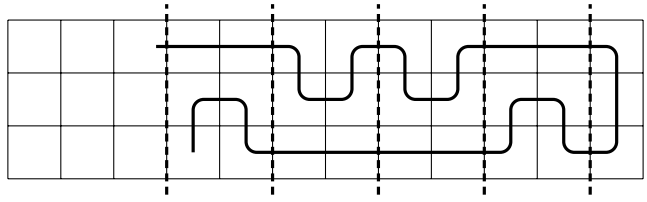


Na rysunku przedstawiono dwa jedyne możliwe ułożenia zygzaka. Drugim parametrem zygzaka jest jego długość (czyli liczba pól zajmowanych w poziomie), oznaczona na rysunku przez ℓ. W skrajnym przypadku, gdy $\ell = 1$, zygzak ograniczony do swojego fragmentu planszy jest po prostu pionowym odcinkiem.

Zostały nam jeszcze do przeanalizowania dwa skrajne fragmenty planszy. W przypadku każdego z nich o ścieżce Hamiltona wiemy tylko tyle, że jej początek znajduje się na lewym (odpowiednio prawym) boku fragmentu. W przykładzie z treści zadania tak się szczęśliwie złożyło, że w prawym skrajnym fragmencie planszy drugi koniec ścieżki znalazł się na tym samym boku co jej początek w tym fragmencie. O takiej ścieżce powiemy, że jest to ścieżka typu *tam i z powrotem*. Ma ona całkiem regularną postać, np.:

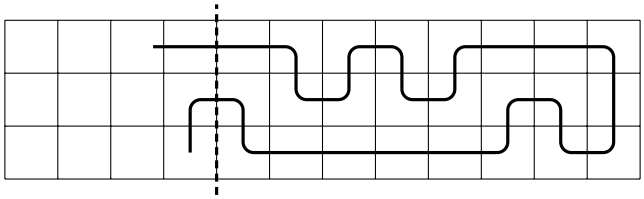


Jak opisać taką ścieżkę? Możemy podzielić fragment planszy na pary kolejnych kolumn (z wyłączeniem ostatniej kolumny, w której ścieżka zawraca). W każdej takiej parze cztery górne pola albo cztery dolne pola tworzą *garb*:

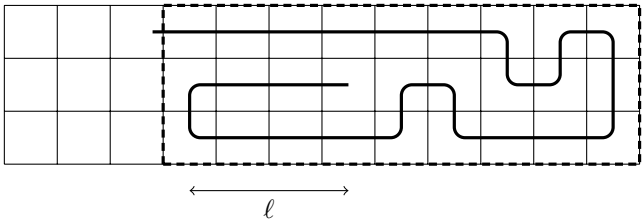


Zaznaczmy tylko, że jeśli liczba kolumn we fragmencie jest parzysta, to skrajnie lewy garb jest ucięty w połowie.

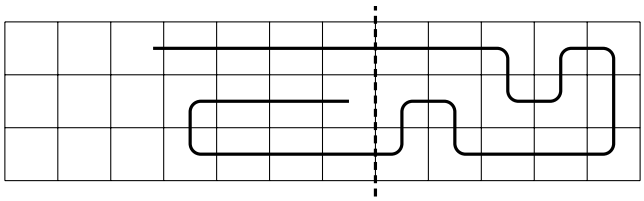
Jeszcze wygodniejszy dla nas sposób patrzenia na taką ścieżkę jest następujący. Jeśli obciąć kolumnę, w której znajdują się oba końce ścieżki, to w wyniku otrzymamy ścieżkę Hamiltona dla krótszego fragmentu planszy, również o tej własności, że oba jej końce leżą na tym samym boku fragmentu (czyli również ścieżkę *tam i z powrotem*):



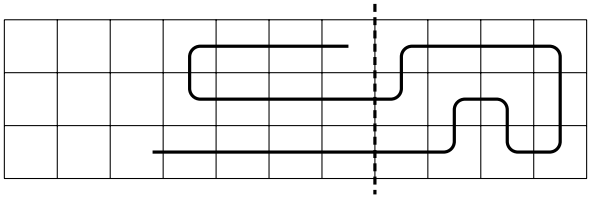
To wszystko pisaliśmy przy założeniu, że ścieżka Hamiltona w skrajnym fragmencie jest ścieżką typu *tam i z powrotem*. Jeśli ścieżka kończy się gdzieś wewnątrz skrajnego fragmentu, to żeby dojść do tego punktu, musi najpierw wrócić do tego samego boku, przy którym się zaczęła (żeby odwiedzić znajdujące się tam pola fragmentu planszy). Faktycznie, gdyby ścieżka nie musiała już wracać do początkowego boku, oznaczałoby to, że musiałaby odwiedzać te pola za pomocą *zygzaka*, co jednak przeczyłoby temu, że fragment planszy jest skrajny po podziale. Po dotarciu do początkowego boku ścieżka zawraca w kierunku swojego końca. Ponieważ mamy do dyspozycji tylko trzy wiersze, więc opisane tu zawracanie odpowiada *zawijasowi*, w którym ścieżka biegnie jednym wierszem w jedną stronę, a drugim w drugą stronę:



Na rysunku zaznaczyliśmy długość *zawijasa* ℓ. A co znajduje się za *zawijasem*? Mamy tam nic innego jak ścieżkę Hamiltona, której oba końce znajdują się na tym samym boku krótszego fragmentu – jest to dokładnie ścieżka typu *tam i z powrotem*, którą rozważaliśmy wcześniej!

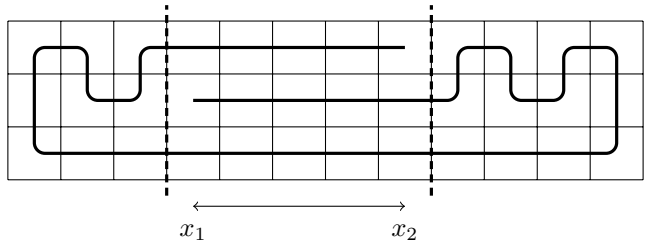


Warto jeszcze zwrócić uwagę na dwie rzeczy. Po pierwsze, *zawijas* nie musi mieć dokładnie takiego kształtu jak ten powyżej. Jedynie część „zawracająca" musi składać się z dwóch kolejnych wierszy. Oto inny, zupełnie poprawny *zawijas*:



Po drugie, może się tak zdarzyć, że zawijas będzie biegł przez całą długość skrajnego fragmentu. W takim przypadku cała ścieżka będzie miała dobrze nam znaną postać *zygzaka* rozważanego na początku tej sekcji.

Na sam koniec zostawiliśmy sobie ostatni przypadek, którego można było nawet w ogóle nie zauważyć. Jest to taka ścieżka Hamiltona, której w żadnym miejscu nie można podzielić pionowym cięciem na ścieżki Hamiltona krótszych fragmentów. Jeśli taka ścieżka zaczyna się na lewym albo na prawym brzegu planszy, to jest to po prostu pojedynczy *zygzak* lub jedna ścieżka typu *tam i z powrotem*, z *zawijasem* lub bez. Jeśli nie, to okazuje się, że jest tylko jeden rodzaj takiej ścieżki, złożony z *zawijasa drugiego typu* i dwóch ścieżek typu *tam i z powrotem*:



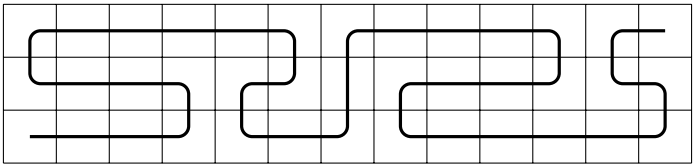
Zawijas drugiego typu ma dwa parametry, x_1 i x_2, określające pozycje jego początku i końca. Składa się z dwóch ścieżek połączonych tylko z jednym fragmentem planszy i ścieżki łączącej oba fragmenty planszy; możliwe są wszystkie rozmieszczenia tych trzech ścieżek w trzech wierszach planszy. Taką ścieżkę Hamiltona będziemy nazywać *zakręconą*. Pozwolimy sobie już pominąć dokładne uzasadnienie, że każda „niepodzielna" ścieżka jest ścieżką zakręconą.

W podsumowaniu całej sekcji możemy napisać, że każda ścieżka Hamiltona na kratce $3 \times n$ albo jest *zakręcona*, albo składa się z co najwyżej dwóch ścieżek typu *tam i z powrotem*, z których każda może mieć dodatkowy *zawijas*, i z dowolnej liczby *zygzaków* w środku.

Pierwsza przymiarka

Nadeszła pora, aby przypomnieć sobie, że nasze zadanie do pewnego stopnia precyzuje, jaką ścieżkę Hamiltona mamy znaleźć. Mamy mianowicie daną dwuwymiarową tablicę $a[1..n, 1..3]$ określającą numery poszczególnych pól na ścieżce, przy czym 0 w tablicy oznacza, że numer danego pola nie jest znany (celowo zamieniliśmy wymiary tablicy w stosunku do treści zadania, tak aby były zgodne z osiami układu współrzędnych). Chcielibyśmy wykorzystać dotychczasowe rozważania kombinatoryczne, aby stworzyć efektywny algorytm znajdowania ścieżki Hamiltona pasującej do zadanych numerów pól. Podstawą algorytmu będzie metoda programowania dynamicznego.

W tej sekcji pozwolimy sobie na pewne uproszczenie, co pomoże nam uchwycić główne pomysły zawarte w rozwiązaniu wzorcowym. Założymy mianowicie, że szukana ścieżka Hamiltona składa się tylko i wyłącznie z *zygzaków* (przykład takiej ścieżki znajduje się na rysunku na następnej stronie). W kolejnej sekcji pokażemy, jak rozszerzyć to rozwiązanie tak, aby poradzić sobie z pozostałymi typami ścieżek. Będziemy celowali w rozwiązanie o złożoności $O(n^2)$.



Spróbujemy zastosować naturalny pomysł, aby budować naszą ścieżkę Hamiltona, dokładając kolejne zygzaki od lewej do prawej. Jeśli mamy już pokryty jakiś fragment planszy, to na kolejnej pozycji będziemy próbowali przykładać różnej długości zygzaki i sprawdzać, czy pasują one do zadanej numeracji pól. W przypadku, gdy któryś z tych zygzaków będzie zgodny z numeracją pól, zapamiętamy informację, że udało nam się pokryć nowy, dłuższy fragment planszy. Na koniec sprawdzimy, czy którymś ze sposobów udało nam się pokryć całą planszę.

Na początku algorytmu musimy jeszcze podjąć kilka decyzji dotyczących początku ścieżki. Skrajnie lewy zygzak może zaczynać się w lewym dolnym albo w lewym górnym rogu planszy; początek każdego kolejnego zygzaka na ścieżce jest już wyznaczony jednoznacznie. Ponadto musimy ustalić, z której strony chcemy umieścić początek, a z której koniec ścieżki Hamiltona, innymi słowy – które pole ma mieć numer 1, a które $3n$. W tym opisie założymy, że pierwszy zygzak zaczyna się w lewym dolnym lub lewym górnym polu planszy, któremu to polu chcemy przypisać numer 1.

Napiszemy pomocniczą funkcję

$$\mathit{zygzak}(x_1, x_2, y_1)$$

sprawdzającą dla $1 \leqslant x_1 \leqslant x_2 \leqslant n$ oraz $y_1 \in \{1, 3\}$, czy fragment planszy od x_1-tej do x_2-tej kolumny włącznie można poprawnie pokryć zygzakiem, którego początek znajduje się na polu $P = (x_1, y_1)$, a koniec na polu $K = (x_2, 4 - y_1)$. Kształt takiego zygzaka jest wyznaczony jednoznacznie. Okazuje się, że – przy przyjętych przed chwilą założeniach – numeracja pól w zygzaku również jest zadana jednoznacznie! Faktycznie, numery kolejnych pól zygzaka rosną, począwszy od numeru $3(x_1 - 1) + 1$ na polu P, a skończywszy na numerze $3x_2$ na polu K.

Załóżmy, że mielibyśmy już tę funkcję. Wówczas całe rozwiązanie moglibyśmy zrealizować z użyciem jednej tablicy wartości logicznych *pokryte*$[0..n, 1..3]$, przy czym *pokryte*$[i, j]$ przechowuje informację, czy udało nam się pokryć fragment planszy złożony z pierwszych i kolumn za pomocą ścieżki kończącej się na polu (i, j). Zakładamy, że na początku cała tablica jest wypełniona wartościami **false**. Oto pseudokod:

```
1: pokryte[0,1] := pokryte[0,3] := true;
2: for i := 0 to n − 1 do
3:   for j ∈ {1,3} do
4:     if pokryte[i,j] then begin
5:       for k := i + 1 to n do
6:         if zygzak(i + 1, k, j) then
7:           pokryte[k, 4 − j] := true;
8:     end
9: return pokryte[n,1] or pokryte[n,3];
```

Jeśli będziemy umieli obliczać wartości funkcji *zygzak* w czasie stałym, to powyższy pseudokod będzie działał w pożądanym czasie $O(n^2)$.

Zajmijmy się więc funkcją *zygzak*. Oszczędzimy sobie dużo pracy, jeśli zauważymy, że każdy zygzak składa się z trzech *odcinków* położonych w poszczególnych wierszach planszy. A konkretnie, dla wywołania $zygzak(x_1, x_2, y_1)$ są to następujące odcinki:

- (x_1, y_1) – (x_2, y_1) o kolejnych numerach pól od $3x_1 - 2$ do $2x_1 + x_2 - 2$,
- $(x_2, 2)$ – $(x_1, 2)$ o kolejnych numerach pól od $2x_1 + x_2 - 1$ do $x_1 + 2x_2 - 1$,
- $(x_1, 4 - y_1)$ – $(x_2, 4 - y_1)$ o kolejnych numerach pól od $x_1 + 2x_2$ do $3x_2$.

Dla każdego z tych odcinków chcemy sprawdzić w czasie stałym, czy zadana numeracja pól planszy jest zgodna z numerami, które próbujemy przydzielić. Wykonamy to za pomocą dodatkowej tablicy *odcinek*, której wartości wyznaczymy zawczasu, znów za pomocą programowania dynamicznego. Aby zmieścić się w czasie i pamięci $O(n^2)$, musimy zaprojektować tę tablicę dosyć sprytnie.

Zrobimy to tak, aby $odcinek[x, y, k, v]$ oznaczało liczbę kolejnych pól planszy, począwszy od pola (x, y) i idąc w kierunku $k \in \{\texttt{lewo}, \texttt{prawo}\}$, których numery są zgodne z numeracją $v, v + 1, v + 2, \ldots$, przy czym pole (x, y) otrzymuje numer v. Formalnie, dla $k = \texttt{prawo}$ chcemy mieć:

$$odcinek[x, y, k, v] = \max\{l \geqslant 0 \,:\, pasuje(a[x, y], v) \wedge \ldots \wedge pasuje(a[x+l-1, y], v+l-1)\}$$

przy czym $pasuje(p, q)$ jest prawdziwe wtedy i tylko wtedy, gdy $p = 0$ lub $p = q$. Dla $k = \texttt{lewo}$ definiujemy to analogicznie.

Tak określona tablica ma $O(n^2)$ pól i możemy ją wypełnić wartościami w czasie $O(n^2)$. Należy tylko nie pogubić się w dużej liczbie jej wymiarów. Jeśli $k = \texttt{prawo}$, to elementy tablicy wyznaczamy od prawej do lewej:

```
1: for x := n downto 1 do
2:   for y := 1 to 3 do
3:     for v := 1 to 3n do
4:       if not pasuje(a[x, y], v) then
5:         odcinek[x, y, prawo, v] := 0
6:       else
7:         odcinek[x, y, prawo, v] := 1 + odcinek[x + 1, y, prawo, v + 1];
```

Przyjmujemy przy tym, że dla $x = n + 1$ mamy zawsze $odcinek[x, y, k, v] = 0$. Dla $k = \texttt{lewo}$ obliczenia wykonujemy analogicznie, tylko od lewej do prawej.

Mając wypełnioną tablicę *odcinek*, wynik funkcji *zygzak* obliczamy tak:

```
1: function zygzak(x_1, x_2, y_1)
2: begin
3:   return (odcinek[x_1, y_1, prawo, 3x_1 − 2] ⩾ x_2 − x_1 + 1) and
4:          (odcinek[x_2, 2, lewo, 2x_1 + x_2 − 1] ⩾ x_2 − x_1 + 1) and
5:          (odcinek[x_1, 4 − y_1, prawo, x_1 + 2x_2] ⩾ x_2 − x_1 + 1);
6: end
```

Rozwiązanie wzorcowe

Aby otrzymać kompletne rozwiązanie, musimy jeszcze, bagatela, rozważyć wszystkie pozostałe konfiguracje ścieżki Hamiltona na kratce $3 \times n$. Na szczęście najtrudniejsze mamy już właściwie za sobą.

Po tym, jak szczegółowo rozpisaliśmy przypadek zygzaka, Czytelnik łatwo uwierzy, że dokładnie w ten sam sposób jesteśmy w stanie w czasie stałym sprawdzić każdy *zawijas*, a także *zawijasy drugiego typu* występujące w zakręconych ścieżkach Hamiltona.

Drugim ważnym elementem są ścieżki typu *tam i z powrotem*. Występują one albo samodzielnie, albo z zawijasem, albo wreszcie jako element składowy ścieżek zakręconych. Wprowadzimy dla nich dość ogólną tablicę wartości logicznych $tizp[x, y_1, y_2, k, v]$, której elementy oznaczają: czy istnieje ścieżka typu *tam i z powrotem* o końcach na polach (x, y_1) i (x, y_2), zawierająca wszystkie pola planszy położone w kierunku k od pól końcowych, przydzielająca polu (x, y_1) numer v i dalej numery rosnąco. Mamy: $x \in \{1, \ldots, n\}$, $y_1, y_2 \in \{1, 2, 3\}$, $y_1 \neq y_2$, $k \in \{\texttt{lewo}, \texttt{prawo}\}$ oraz $v \in \{1, \ldots, 3n\}$.

Tablica ta ma rozmiar $O(n^2)$. Aby wypełnić ją w czasie $O(n^2)$, skorzystamy z poczynionej już wcześniej obserwacji, że jeśli z fragmentu pokrytego ścieżką typu *tam i z powrotem* usuniemy skrajną kolumnę (dla $k = \texttt{prawo}$ będzie to skrajnie lewa kolumna), to otrzymamy ścieżkę tego samego typu dla fragmentu o jedną kolumnę krótszego. Aby zatem wyznaczyć wartości tablicy dla $k = \texttt{prawo}$, poruszamy się od prawej do lewej ($x = n, \ldots, 1$) i każdą wartość typu $tizp[x, y_1, y_2, k, v]$ obliczamy na podstawie jednej lub dwóch wartości typu $tizp[x + 1, y_1', y_2', k, v']$. Wybór dwóch wartości mamy tylko wtedy, gdy $\{y_1, y_2\} = \{1, 3\}$, co odpowiada podjęciu decyzji, w którą stronę ma pójść *garb*. Dla $k = \texttt{lewo}$ postępujemy podobnie, tylko obliczenia wykonujemy od lewej do prawej.

W ten sposób omówiliśmy już wszystkie elementy składowe ścieżek. Na koniec pozostaje nam połączyć to wszystko w jedną całość. Przykładowo, pokażemy, w jaki sposób można rozważyć wszystkie ścieżki złożone z *zygzaków* i ścieżek typu *tam i z powrotem*:

```
 1: for x := 1 to n do
 2:   foreach y1, y2 ∈ {1,2,3}, y1 ≠ y2 do
 3:     if tizp[x, y1, y2, lewo, 1] then
 4:       pokryte[x, y2] := true;
 5: pokryte[0, 1] := pokryte[0, 3] := true;
 6: { Wypełnij resztę tablicy pokryte (programowanie dynamiczne dla zygzaków) }
 7: wynik := false;
 8: for x := 1 to n do
 9:   foreach y1, y2 ∈ {1,2,3}, y1 ≠ y2 do
10:     if tizp[x, y1, y2, prawo, 3x − 2] and pokryte[x − 1, y1] then
11:       wynik := true;
12: for y := 1 to 3 do
13:   wynik := wynik or pokryte[n, y];
14: return wynik;
```

Jeśli do powyższego pseudokodu dodać rozpatrywanie *zawijasów* w ścieżkach *tizp* (można to zrobić za pomocą analogicznej funkcji jak w przypadku *zygzaków*, wykorzystującej tablicę *odcinek*), otrzymamy rozwiązanie nieuwzględniające jeszcze tylko ścieżek *zakręconych*. Te ostatnie już tradycyjnie potraktujemy trochę po macoszemu i przemyślenie sposobu ich rozpatrzenia pozostawimy Czytelnikowi.

Na sam koniec rozwiązania pozostała nam ostatnia trudność, bardziej implementacyjna niż koncepcyjna, a mianowicie odtworzenie wyniku. Dotychczas poszukiwaliśmy jedynie odpowiedzi na pytanie, czy ścieżka istnieje (co skądinąd mamy zagwarantowane), a teraz musimy się zastanowić, jak tę ścieżkę znaleźć. Zastosujemy tu standardową metodę odtwarzania wyniku w programowaniu dynamicznym. Z każdą tablicą w rozwiązaniu przechowującą wartości logiczne związżemy dodatkową tablicę pomocniczą, tzw. tablicę ojców. Jeśli w jakimś polu tablicy logicznej wystąpi wartość **true**, to w odpowiadającym mu polu tablicy pomocniczej umieścimy informację o tym, na jakiej podstawie tę wartość wyznaczyliśmy. Przykładowo, w przypadku tablicy *pokryte* będzie to para liczb określająca indeksy pola tablicy, na podstawie którego uzyskaliśmy w danym polu wartość **true** (lub informacja, że pojawiła się ona z powodu ścieżki typu *tam i z powrotem*), natomiast w przypadku tablicy *tizp* może to być albo analogiczna piątka liczb określająca indeksy poprzedniego pola w tablicy, albo po prostu jedna liczba 0 lub 1 wskazująca, w którą stronę był skierowany ostatni *garb*. W każdym przypadku odpowiednią informację zapamiętamy przy okazji wykonywania głównego algorytmu programowania dynamicznego. Gdy teraz będziemy chcieli odtworzyć wynikową ścieżkę, wystarczy wycofać się po wartościach ojców.

Implementację rozwiązania wzorcowego można znaleźć w pliku `waz.cpp` (rekurencja ze spamiętywaniem) i `waz1.cpp` (programowanie dynamiczne). W plikach `wazs2.cpp` i `wazs7.cpp` zapisano niewiele gorsze rozwiązanie, o złożoności $O(n^3)$, za to nieco prostsze w implementacji. Pominięto w nim tablicę *odcinek*, a wszystkie *zygzaki* i *zawijasy* sprawdzano kolejno pole po polu.

Inne rozwiązania

Z racji podobieństwa zadania do problemu znajdowania ścieżki Hamiltona w grafie, w rozwiązaniu można próbować stosować techniki standardowe dla tego problemu.

Najprostsze rozwiązanie wykładnicze (`wazs1.cpp`) rekurencyjnie konstruuje ścieżkę, startując z każdego możliwego wierzchołka. Rozwiązanie to w pewnych przypadkach działa zupełnie nieźle, np. gdy plansza jest w całości wypełniona (wtedy jest to zwykłe przeszukiwanie) lub gdy jest pusta. Tego typu rozwiązania zdobywały na zawodach 20-30 punktów.

Inne podejście wykorzystuje fakt, że graf podany w zadaniu ma małą szerokość (równą 3). Jest to programowanie dynamiczne, w którym pojedynczy *stan* opisuje konfigurację na przekroju grafu, czyli w trzech wierzchołkach położonych w jednej kolumnie planszy. W takim stanie zakładamy, że ponumerowaliśmy już odpowiednio wszystko od tego stanu na lewo, i pamiętamy wszystkie informacje istotne z punktu widzenia konstruowania dalszej części ścieżki. Mogą to być np. następujące informacje:

- numer kolumny planszy (x),
- jakie trzy liczby wpisaliśmy w tę kolumnę planszy (trzyelementowa tablica t),

- jaki jest zbiór liczb A użytych dotychczas na planszy – nietrudno zauważyć, że jeśli już wpisane liczby dają się rozszerzyć do rozwiązania, to zbiór A wyraża się jako suma co najwyżej dwóch rozłącznych przedziałów,
- ilu sąsiadów na ścieżce Hamiltona brakuje każdemu z rozważanych trzech wierzchołków, po uwzględnieniu tej kolumny i wszystkiego, co było wcześniej – każdy wierzchołek ma co najwyżej dwóch sąsiadów na ścieżce Hamiltona (tablica *deg*).

Poniższej przedstawiamy fragment planszy wraz z odpowiadającym mu stanem:

17	18	19	20	21
16	13	12	1	2
15	14	11	10	9

- $x = 5$
- $t[1] = 9$, $t[2] = 2$, $t[3] = 21$
- $A = [1, 2] \cup [9, 21]$
- $deg[1] = deg[2] = deg[3] = 1$

Przygotowano kilka przykładowych implementacji tego typu podejścia (`wazs[3-6].cpp`), różniących się liczbą informacji pamiętanych w pojedynczym stanie (a co za tym idzie, liczbą wymiarów programowania dynamicznego) i sposobem przechodzenia między stanami. We wszystkich rozwiązaniach pamiętamy jedynie faktycznie dopuszczalne stany. W zależności od optymalizacji tego typu rozwiązania zdobywały na zawodach od 30 do nawet 60 punktów.

Warto jeszcze wyróżnić rozwiązanie `wazb1.cpp`, które przekraczało limit pamięciowy do tego zadania. Rozwiązanie wzorcowe zużywa mniej niż 200 MB pamięci. Jest tak jednak dzięki temu, że tablice stosowane w tym rozwiązaniu są zadeklarowane rozważnie, z użyciem typów całkowitych 1- i 2-bajtowych (tj. `char` i `short` w C++). Gdyby zamiast tego wszędzie używać rozrzutnie typu 4-bajtowego (`int`), można zużyć nawet ponad 600 MB pamięci, co właśnie ma miejsce w rozwiązaniu `wazb1.cpp`. Limit pamięciowy w zadaniu (512 MB) był dobrany tak, aby zmieszczenie się w nim wymagało jedynie elementarnej rozwagi w tym zakresie.

Testy

W tym zadaniu zaproponowano dość dużą liczbę testów, jednak niezgrupowanych, z których każdy był wart od 1 do 4 punktów. Każdy z testów można w pewnym sensie traktować jako osobną łamigłówkę.

W większości testy były generowane zgodnie z konfiguracjami z rozwiązania wzorcowego poprzez wyzerowanie pewnych losowych pól z planszy. Najtrudniejsze testy charakteryzują się niewielką liczbą pól ze znanymi numerami fragmentów węża (np. ścieżkę *zakręconą* można wymusić już za pomocą wartości trzech pól). W innych testach złośliwe ułożenie pól o znanych wartościach miało pomóc sprawdzić efektywność rozwiązań. Przykładowo, w przypadku ścieżki typu *tam i z powrotem* podanie wartości pól tylko w górnym lub tylko w dolnym wierszu determinuje całą konfigurację, jednak jeśli pójdzie się od drugiej strony, rozwiązanie wykładnicze będzie długo błądzić. Wreszcie w zestawie znalazły się testy z pustą lub prawie pustą planszą i testy z planszą całkowicie wypełnioną.

Zawody II stopnia

opracowania zadań