Karol Cwalina — Treść zadania, Opracowanie
Jakub Radoszewski — Program

OI, Etap I, 22.10–19.11.2007

Robinson

Rzucony przez sztorm na bezludną wyspę Robinson zbudował łódkę, na której postanowił wypłynąć w morze i szukać ludzkich siedzib. Jest on doświadczonym żeglarzem, więc jego łódź jest zbudowana zgodnie z zasadami sztuki: ma wzdłużną oś symetrii oraz odpowiedni kształt. Na dziobie jest wąska, stopniowo rozszerza się w stronę środka, by z kolei zacząć się zwężać w kierunku rufy. W szczególności oznacza to, że w pewnym miejscu w środku łódź jest szersza niż na dziobie i na rufie.

Pech chciał, że w miejscu, w którym Robinson spuścił swoją łódź na wodę, rosną bardzo gęste trzciny. Co gorsza, są one tak sztywne, że łódź nie jest w stanie ich złamać i przepłynąć przez zajmowane przez trzciny miejsca. Być może, odpowiednio manewrując łodzią, Robinson jest w stanie wypłynąć na otwarte morze.

Ze względu na słabą manewrowość, łódź może się poruszać do przodu, do tyłu, w prawo oraz w lewo, ale nie może skręcać. Oznacza to, że może się zdarzyć sytuacja, gdy łódź porusza się rufą lub którąś burtą do przodu.

Twoim zadaniem jest ocenić, czy Robinson może wydostać się na pełne morze.

Dla uproszczenia, wyspa wraz z otoczeniem jest w zadaniu reprezentowana przez kwadratową mapę podzieloną na kwadratowe pola jednostkowe, z których każde może być zajęte przez wodę, kawałek łodzi Robinsona bądź przez przeszkodę (np. ląd lub trzcinę). Łódź jest ustawiona równolegle do jednego z czterech kierunków świata i właśnie w tym kierunku przebiega przez środek łodzi pas pól jednostkowych, idealnie przepołowionych wzdłużną osią symetrii łodzi.

Przyjmujemy, że poza mapą znajduje się otwarte morze. Robinson może na nie wypłynąć, jeżeli łódź jest w stanie w całości opuścić teren pokazany na mapie. Pojedynczy ruch to przesunięcie się łodzi o jedno pole w wybranym kierunku (północ, południe, wschód lub zachód). Aby ruch był dozwolony, przed jego wykonaniem i po nim łódź musi w całości znajdować się na wodzie.

Zadanie

Napisz programu, który:

- *wczyta ze standardowego wejścia opis mapy,*
- *obliczy minimalną liczbę ruchów łodzi potrzebnych do wypłynięcia poza obszar pokazany na mapie,*
- *wypisze obliczoną liczbę na standardowe wyjście.*

Wejście

Pierwszy wiersz zawiera jedną liczbę całkowitą $3 \leqslant n \leqslant 2\,000$, oznaczającą długość boku mapy. W każdym z następnych n wierszy znajduje się po n znaków opisujących kolejne pola mapy:

i-ty znak w wierszu numer $j+1$ określa zawartość pola o współrzędnych (i,j). W wierszu mogą się pojawić następujące znaki:

- `.` *(kropka) — oznacza pole zajęte przez wodę,*
- `X` *— oznacza przeszkodę (trzcinę lub kawałek lądu),*
- `r` *— oznacza fragment łodzi Robinsona.*

Wyjście

Twój program powinien wypisać (w pierwszym i jedynym wierszu wyjścia) jedną dodatnią liczbę całkowitą, równą najmniejszej liczbie ruchów łodzi niezbędnych do opuszczenia przez nią w całości terenu przedstawionego na mapie. Jeśli wypłynięcie na otwarte morze nie jest możliwe, należy wypisać słowo „`NIE`*”.*

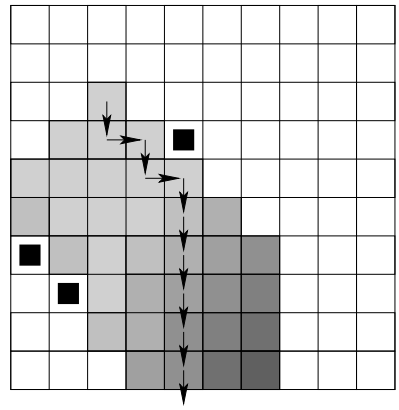
Przykład

Dla danych wejściowych:

```
10
..........
..........
..r.......
.rrrX.....
rrrrr.....
.rrr......
X.r.......
.Xr.......
..........
..........
```

poprawnym wynikiem jest:

```
10
```



Rozwiązanie

Zacznijmy od wybrania najistotniejszych informacji z treści zadania. Otóż:

- łódź może poruszać się w czterech głównych kierunkach, przy czym nie każdy ruch jest dozwolony;
- szukamy najkrótszej ścieżki, która doprowadzi do celu.

Choć bardzo uproszczony, opis ten jednoznacznie wskazuje, że mamy do czynienia z zagadnieniem znajdowania drogi w grafie. To bardzo naturalne dla doświadczonych zawodników spostrzeżenie i jednocześnie bardzo dobra wiadomość: jako jeden z klasycznych

problemów informatyki, zagadnienie to już dawno doczekało się wielu efektywnych algorytmów, na przykład BFS-a, czyli przeszukiwania wszerz[1].

Zanim przystąpimy do pracy nad rozwiązaniem zadania, zauważmy, że możliwe są dwa podstawowe przypadki — oś symetrii łodzi może mieć kierunek poziomy lub pionowy. W dalszej części opracowania założymy, że łódź ma poziomą oś symetrii. Przypadki łodzi o pionowej osi symetrii łatwo rozwiązać analogicznie lub przekształcić do przypadku o osi poziomej.

Dodatkowo, warto wyróżnić pewien punkt łodzi i utożsamiać jej położenie z położeniem tego punktu. Wyróżniony punkt możemy nazwać *punktem zakotwiczenia*, a o łodzi położonej na mapie w ten sposób, że jej punkt zakotwiczenia znajduje się na polu P mapy, powiemy, że *jest zakotwiczona* w polu P lub że *zajmuje pozycję* P. Wówczas wystarczy, byśmy potrafili określić dla każdego punktu na mapie, czy zakotwiczona w nim łódź nie koliduje z żadnymi przeszkodami — taki punkt nazwiemy *dozwolonym* — oraz rozpoznać, kiedy łódź wypłynęła poza mapę. Ponieważ graf nieskierowany, którego wierzchołki odpowiadają interesującym nas pozycjom łodzi, a krawędzie — możliwym ruchom łodzi, ma oczywiście rozmiar $O(n^2)$, to będziemy mogli go przeszukać w złożoności czasowej i pamięciowej $O(n^2)$, czyli najlepszej z możliwych.

W ten sposób sprowadziliśmy zadanie „zaledwie" do wyznaczenia podziału pól mapy na dozwolone i zabronione oraz… odszukania łodzi na mapie. W pierwszej kolejności skupimy się na wykrywaniu, czy pozycja na mapie jest dozwolona. Odnalezienie łodzi i odpowiednie jej przedstawienie w algorytmie zostawiamy na później.

Wyznaczenie pozycji zabronionych — wprawki

Zamiast zastanawiać się, czy określone położenie łodzi jest dozwolone, odwrócimy problem. Dla każdej przeszkody — pola zajętego przez ląd lub szuwary — wyznaczymy takie pola na mapie, że zakotwiczona w nich łódź koliduje z daną przeszkodą.

Rozważmy jedno pole z przeszkodą. Okazuje się, że pozycje łodzi wykluczone przez tę przeszkodę tworzą taki sam kształt jak łódź, tyle że odbity środkowosymetrycznie. Przekonają nas o tym przedstawione niżej obliczenia. Jeśli dla kogoś nie są one wystarczająco czytelne, to warto także przeanalizować rys. 1 (uwaga: przedstawia on łódź, która jest ogólniejszym przykładem, przez co nie spełnia warunków zadania).

Wprowadźmy na mapie naturalny układ współrzędnych, gdzie oś OX jest skierowana na wschód, a oś OY — na północ (czasem będziemy też używali określeń *poziomo* i *w prawo* lub *pionowo* i *do góry*). Za punkt zakotwiczenia łodzi przyjmijmy jej skrajnie lewy punkt — będziemy go nazywać także *dziobem*. Niech

$$L = \{(x_i, y_i) \ : \ i = 1, 2, \ldots\}$$

będzie zbiorem par liczb określających położenie punktów łodzi względem jej punktu zakotwiczenia. Wówczas, jeśli łódź jest na pozycji o współrzędnych (x, y), to zajmuje pola:

$$\{(x + x_i, y + y_i) \ : \ i = 1, 2, \ldots\},$$

[1] BFS i podobne zagadnienia są opisane w licznych podręcznikach algorytmiki, w tym w [20] i [17]. Występują także w wielu zadaniach z poprzednich edycji Olimpiady.

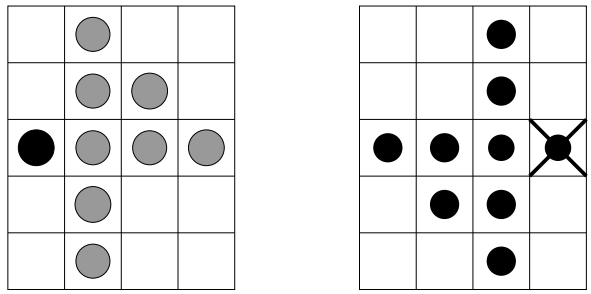
co zapiszemy krócej jako $(x,y)+L$. Jeśli (x_S,y_S) jest polem, na którym znajduje się przeszkoda, to koliduje ona z takimi pozycjami łodzi (x,y), że

$$(\exists i)\ (x,y)+(x_i,y_i)=(x_S,y_S) \Leftrightarrow (\exists i)\ (x,y)=(x_S,y_S)-(x_i,y_i),$$

co jest równoważne temu, że

$$(x,y) \in (x_S,y_S)+(-L).$$

Widzimy więc, że pozycje łodzi wykluczone przez jedną przeszkodę „mają kształt" określony przez zbiór $-L$, czyli środkowosymetryczne odbicie kształtu łódki.



(a) (b)

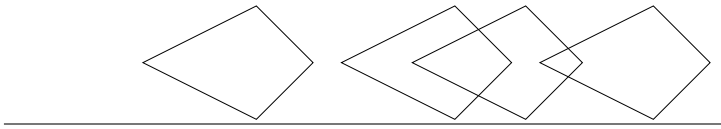
Rys. 1: Na rysunku (a) jest przedstawiona łódź (zbiór L) z wyróżnionym punktem zakotwiczenia — dziobem. Na rysunku (b) pokazano wszystkie pozycje łodzi, które kolidują z przeszkodą znajdującą się w punkcie zaznaczonym krzyżykiem. Widać, że pozycje te tworzą środkowosymetryczne odbicie kształtu łodzi. (Pokazana tu łódź nie jest osiowosymetryczna i przez to nie spełnia warunków zadania, ale pozwala za to lepiej zilustrować zależność pomiędzy zbiorem L a zabronionymi pozycjami).

Przy okazji warto zauważyć, że niektóre pozycje zabronione mogą znaleźć się poza mapą Robinsona. Nie stanowi to żadnego utrudnienia, lecz implementując rozwiązanie, trzeba będzie pamiętać o odpowiednim powiększeniu mapy.

Wyznaczenie pozycji zabronionych — zamiatamy

Wyznaczenie wszystkich pozycji zabronionych sprowadza się zatem do „naniesienia" na mapę pewnej liczby obrazów odbitych środkowosymetrycznie łódek. Prostym i narzucającym się rozwiązaniem jest bezpośrednie zaznaczenie na mapie każdego odbicia związanego z określoną przeszkodą, czyli zbioru $(x_S,y_S)-L$ dla każdego pola (x_S,y_S) zajętego przez przeszkodę. Ponieważ zarówno wielkość łódki, jak i liczba pól zajętych przez przeszkody, mogą być rzędu $O(n^2)$, to otrzymany w ten sposób algorytm ma złożoność $O(n^4)$. Jest więc zbyt wolny, choć oczywiście dla niewielkich testów poprawnościowych może być wystarczający.

Poszukując innej metody, rozważmy prostszy przypadek, gdy wszystkie pola z przeszkodami znajdują się na tej samej szerokości geograficznej, tzn. na tej samej prostej poziomej. Wówczas kształty obejmujące pozycje zabronione możemy przedstawić w uproszczeniu jak na rys. 2.



Rys. 2: Schematyczne przedstawienie zabronionych pozycji łodzi, generowanych przez przeszkody położone na jednej prostej poziomej.

Aby zaznaczyć wszystkie pola kolidujące z ułożonymi we wspomniany sposób przeszkodami, będziemy rozważać kolejne proste pionowe mapy i wykrywać pola zabronione położone na tych prostych. Tradycyjnie taki sposób przeglądania płaszczyzny nazywamy *zamiataniem*, a prostą (w tym przypadku pionową), którą zamiatamy całą płaszczyznę, przyjęło się określać mianem *miotły*.

Mapę będziemy zamiatać od lewej do prawej. W rozważanym przypadku, gdy wszystkie przeszkody i osie symetrii wszystkich odbić leżą na jednej prostej, w danym położeniu miotły jest tylko jedno odbicie, którego przekrój z miotłą trzeba rozważyć — to, dla którego ten przekrój jest największy. Takie odbicie nazwiemy *dominującym*. Zauważmy, że dla danego położenia miotły wszystkie odbicia łodzi możemy podzielić na trzy grupy:

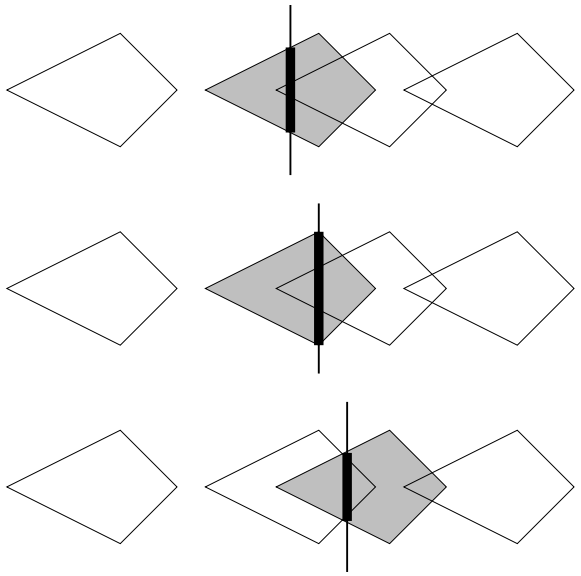
1. odbicia, które będą mogły stać się dominującymi dopiero, gdy miotła przesunie się na prawo;
2. odbicie aktualnie dominujące;
3. odbicia, które na pewno nie będą już dominowały.

Początkowo, gdy miotła znajduje się w skrajnie lewym położeniu, wszystkie odbicia należą do pierwszej grupy. W trakcie zamiatania każde odbicie przechodzi kolejno do drugiej i, dalej, do trzeciej grupy. Odbiciem, które jako kolejne opuszcza pierwszą grupę, jest zawsze to z tej grupy, które leży najbardziej na lewo. Przykłady sytuacji napotykanych w trakcie zamiatania są przedstawione na rys. 3.

Można już dostrzec, jak rozwiązać problem w rozważanym przypadku. Trzeba dysponować listą wszystkich pól z przeszkodami uporządkowaną rosnąco według pierwszej współrzędnej — początkowo wszystkie one (a właściwie związane z nimi odbicia łodzi) należą do pierwszej grupy. Następnie można zacząć rozważać kolejne pozycje miotły — począwszy od skrajnie lewej. Przy przejściu do kolejnej pozycji miotły należy sprawdzić, czy nie trzeba uaktualnić odbicia dominującego. Jest to jednak proste i przy odpowiedniej reprezentacji łodzi może być wykonane w czasie stałym — trzeba porównać aktualnie dominujące odbicie z pierwszym na liście w grupie pierwszej. Potem pozostaje tylko policzyć przekrój odbicia dominującego z miotłą.

Powróćmy do ogólnego przypadku, gdy przeszkody mogą zajmować różne szerokości geograficzne. Wówczas postępujemy bardzo podobnie, powtarzając to samo postępowanie niezależnie dla każdej prostej poziomej, na której leżą przeszkody. Trzeba jeszcze tylko zastanowić się, jak dla danej pozycji miotły szybko wyznaczyć wszystkie leżące na niej pozycje zabronione, tzn. jak połączyć wyniki otrzymane dla różnych prostych poziomych.

Gdybyśmy zaznaczali pola zabronione wynikające z przeszkód leżących na każdej prostej oddzielnie, to zaznaczenie wszystkich pól w jednym położeniu miotły mogłoby nam zająć czas $O(n^2)$ (może być bowiem około n przeszkód, każda wyznaczająca odbicie szerokości około n). Sumarycznie potrzebowalibyśmy wówczas czasu $O(n^3)$ — jest to na pewno wolniej niż byśmy sobie życzyli.



Rys. 3: Kolejne położenia miotły i wyróżnione na szaro odbicia dominujące w tych położeniach. Dla każdego odbicia dominującego, za pomocą czarnego prostokąta zaznaczone są pozycje zabronione, leżące na przecięciu tego odbicia z miotłą.

Czy zatem można lepiej? Na szczęście tak! W ustalonym położeniu miotły, zamiast nanosić na nią całe przekroje odbić dominujących, dla każdego odbicia zaznaczymy tylko jego skrajne punkty: górny i dolny. Możemy to oczywiście zrobić w czasie liniowym. Potem wystarczy raz przejrzeć wszystkie punkty miotły od dołu do góry, zliczając napotkane początki i końce odbić dominujących — jeśli początków jest więcej, to jesteśmy akurat w punkcie zabronionym. W przeciwnym przypadku, czyli gdy początków i końców jest tyle samo, jesteśmy w punkcie dozwolonym.

Postępując w opisany wyżej sposób, w każdym z $O(n)$ położeń miotły musimy:

1. dla każdej prostej poziomej uaktualnić odbicie dominujące — odbywa się to w czasie stałym;
2. dla każdego z tych odbić znaleźć początek i koniec jego przekroju z miotłą i oznaczyć je wszystkie na miotle; potrzebny na to czas zależy od przyjętej reprezentacji łodzi — musimy postarać się, by dało się to zrobić również w czasie stałym dla pojedynczego odbicia;
3. wyznaczyć wszystkie pozycje zabronione leżące na miotle — potrafimy to wykonać w czasie liniowym.

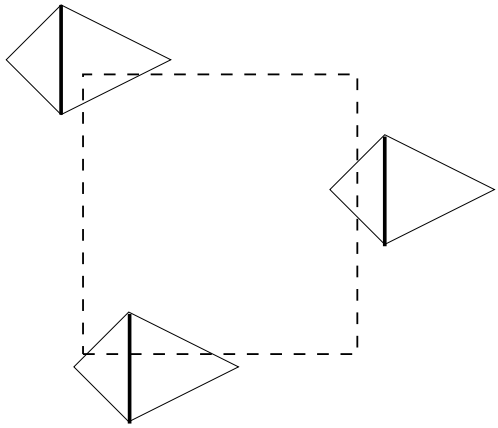
Tym samym wszystkie pola zabronione potrafimy odnaleźć w czasie rzędu $O(n^2)$. Potrzebna pamięć jest tej samej wielkości.

Reprezentacja łodzi

Ta część zadania wygląda na prostą, ale warto się nad nią zastanowić. Wiemy już, czego oczekujemy od naszej reprezentacji.

Łódź to dla nas osiowosymetryczny kształt z wyróżnionym punktem zakotwiczenia. Utożsamiamy go ze zbiorem par liczb L, który w razie potrzeby odbijamy środkowosymetrycznie, otrzymując zbiór $-L$. Umieszczając punkt zakotwiczenia odbicia we wszystkich punktach mapy, w których znajdują się przeszkody, będziemy chcieli oznaczyć wszystkie punkty zabronione na mapie. W tym celu musimy umieć wyznaczyć dowolny pionowy przekrój odbicia łodzi. Reprezentacja musi też pozwolić na szybkie rozpoznanie, czy w danym położeniu łódź wciąż znajduje się na mapie, czy już poza nią wypłynęła.

Jak widać nie mamy nadzwyczaj wygórowanych wymagań. Po wczytaniu mapy wystarczy jednokrotnie ją przejrzeć, by odnaleźć najmniejszy prostokąt zawierający łódź i stwierdzić, która z dwóch jego osi jest osią symetrii łodzi. Jeśli okaże się, że jest to oś pionowa, to odbijamy mapę, by łódź miała poziomą oś symetrii. Teraz pozostaje tylko zapamiętać ciąg liczb będących szerokościami kolejnych przekrojów poprzecznych łodzi. Przy takiej reprezentacji wyznaczanie przekroju łodzi z miotłą jest bardzo proste.



Rys. 4: Różne możliwe położenia poprzeczki łodzi względem obszaru objętego mapą: na lewo od pionowych granic mapy, pomiędzy nimi i na prawo od nich.

Rozstrzygnięcie, czy łódź wypłynęła poza planszę, wymaga rozważenia kilku przypadków. Dla ułatwienia, pionowy odcinek, prostopadły do osi wzdłużnej łodzi i łączący jej burty w najszerszym miejscu nazwijmy *poprzeczką*. Na rys. 4 widzimy kilka możliwości: poprzeczka łodzi może znajdować się na zachód od mapy, może być na długości geograficznej objętej przez mapę lub znajdować się na wschód od niej. Każda z tych możliwości wymaga nieco innego potraktowania, ale poza uważną implementacją nie jest przy tym potrzebna żadna szczególna pomysłowość:

1. gdy poprzeczka jest położona na zachód od mapy, to wystarczy sprawdzić, czy zachodni brzeg mapy jest rozłączny z łodzią, czyli czy nie przecina się z jej odpowiednim przekrojem;
2. gdy poprzeczka jest na długości geograficznej obejmowanej przez mapę, to łódź opuszcza mapę wtedy i tylko wtedy, gdy poprzeczka znajdzie się poza mapą;
3. gdy poprzeczka jest położona na wschód od mapy, to zachodzi przypadek analogiczny, jak pierwszy, tylko trzeba rozważyć wschodni brzeg mapy.

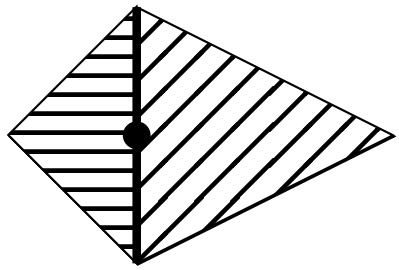
Opisane powyżej rozwiązanie zostało zaimplementowane w plikach `rob.cpp`, `rob1.c`, `rob2.pas` oraz `rob6.java`.

Jak już wspominaliśmy, przedstawione rozwiązanie to prosty przykład zastosowania techniki zamiatania[2].

Przegląd innych rozwiązań

Rozwiązanie $O(n^2)$

Okazuje się, że rozwiązanie o takiej samej złożoności jak wzorcowe można uzyskać, stosując inną metodę — *programowanie dynamiczne*. Pozwoli nam ona wyznaczyć bezpośrednio wszystkie dozwolone pozycje łodzi. Jak poprzednio, załóżmy, że oś symetrii łodzi jest położona poziomo, z dziobem skierowanym na zachód. Jak poprzednio, poprzeczką nazwijmy odcinek prostopadły do osi łodzi poprowadzony w jej najszerszym miejscu. Na potrzeby tego rozdziału część łodzi na lewo od poprzeczki (wraz poprzeczką) nazwijmy w całości *dziobem*, a część na prawo od poprzeczki (bez poprzeczki) — *rufą*. Dla wygody przyjmiemy także, że tym razem punktem zakotwiczenia łodzi będzie środkowy punkt poprzeczki (patrz rys. 5).



Rys. 5: Schemat łodzi z zaznaczonymi: poprzeczką (grubsza kreska), dziobem (zakreskowany poziomo, wraz z poprzeczką), rufą (zakreskowana ukośnie) i punktem zakotwiczenia (małe czarne kółko).

Istotą programowania dynamicznego jest wykorzystywanie wcześniejszych wyników do obliczenia kolejnych. W naszym przypadku chcielibyśmy coś powiedzieć o rozważanej pozycji łodzi w oparciu o wiedzę zgromadzoną w trakcie analizy poprzednich pozycji. Przypuśćmy, że będziemy rozważać pozycje na mapie kolejno wierszami, a w jednym wierszu od lewej do prawej. Gdy analizujemy określoną pozycję, znamy wynik dla pozycji leżącej w poprzedniej kolumnie i tym samym wierszu, co badana. Załóżmy, że była ona dozwolona. Wówczas jedyne pola, jakie trzeba sprawdzić w poszukiwaniu przeszkód dla nowej pozycji, to pola leżące bezpośrednio przy rufie łodzi. Kształt brzegu rufy może być jednak dość nieregularny.

Aby obejść tę trudność, rozważymy odrębnie dziób i rufę łodzi, wyznaczając wszystkie pozycje dozwolone dla każdej z tych części osobno. Zauważmy, że przesuwając w prawo sam dziób, mamy bardzo regularny kształt z prawej strony — odpowiada on przesuniętej

[2]Więcej na jej temat, wraz z przykładami, można znaleźć w [20] i [17]. Na Olimpiadzie również pojawiały się już zadania, w których znalazło się miejsce dla zamiatania: przede wszystkim zadanie *Lodowisko* na VI Olimpiadzie Informatycznej, ale także *Kopalnia złota*, *Wyspy* i *Gazociągi* podczas finałów VIII, XI i XIV Olimpiady.

o jedną kolumnę poprzeczce. Z kolei wyznaczając pozycje dozwolone dla rufy, możemy pola w każdym wierszu rozważać od prawej do lewej. Wtedy przesuwając rufę w lewo, także mamy regularny kształt — tym razem z lewej strony. Pozycjami dozwolonymi dla całej łodzi będą pozycje dozwolone i dla dziobu, i dla rufy.

W dalszej części skupimy się tylko na analizowaniu dziobu łodzi. Trzeba zastanowić się, jak efektywnie sprawdzić, czy dziób, który mieścił się bez przeszkód na określonej pozycji, da się bezkolizyjnie umieścić w nowej kolumnie. Niestety, przejście do nowej pozycji w sytuacji, gdy aktualna pozycja jest zabroniona, wydaje się być trudniejsze.

Aby poradzić sobie z problemem... uogólnimy go! Zamiast pytać, czy cały dziób łodzi mieści się bez przeszkód na mapie, zapytajmy, jaka jego część się mieści. *Prefiksem* dziobu długości ℓ nazwijmy fragment dziobu o długości ℓ (liczonej poziomo) począwszy od jego lewego, najwęższego krańca. Powiemy, że taki prefiks mieści się na pozycji (x,y) na mapie, jeśli można go umieścić bez przeszkód tak, że środek jego prawego boku wypada na pozycji (x,y). Dla pozycji (x,y), przez $pref_{x,y}$ oznaczmy długość najdłuższego prefiksu dziobu, który mieści się bez przeszkód na tej pozycji. Obliczając te wartości, dla każdej pozycji na mapie określimy, jak duży kawałek dziobu mieści się na niej. Dopiero na końcu wybierzemy jako dozwolone te pozycje, na których mieści się cały dziób.

Jak obliczyć wartość $pref_{x+1,y}$ na podstawie $pref_{x,y}$? Gdybyśmy wiedzieli, jaki jest najdalszy poprzeczny przekrój dziobu (licząc od lewej), który mieści się na pozycji (x,y) (oznaczmy tę liczbę jako $maxp_{x+1,y}$), to mielibyśmy gotowy wzór:

$$pref_{x+1,y} = \min(maxp_{x+1,y}, pref_{x,y}+1),$$

wskazujący na klasyczne zastosowanie techniki programowania dynamicznego!

Na szczęście obliczenie wartości $maxp_{x,y}$ również jest zagadnieniem, do którego można zastosować programowanie dynamiczne. Dla każdego pola (x,y) mapy możemy znaleźć najpierw *maksymalny promień* — maksymalną wartość r, taką że na polach $\{(x,y+i) : -r < i < r\}$ nie ma żadnych przeszkód. Potem wystarczy znaleźć najdalszy wśród najszerszych przekrój poprzeczny dziobu, który nie przekracza rozmiarów tego paska, czyli wartość $maxp_{x,y}$. Łatwo to wykonać w czasie stałym, jeśli wcześniej, w czasie $O(n)$, przygotujemy tablicę z odpowiednią informacją o łodzi.

Jak zatem znaleźć maksymalny promień dla pola mapy? Najpierw poszukajmy przeszkód ograniczających tę wartość, położonych nad danym polem. W tym celu przejrzyjmy kolumnę od góry w dół, cały czas pamiętając, gdzie po raz ostatni napotkaliśmy pole z przeszkodą — odległość do tego pola będzie ograniczeniem wartości promienia. Podobnie postępując, wyznaczymy odległość od najbliższej przeszkody na dole, ograniczającą wartość promienia. Za szukaną wartość promienia weźmiemy minimum z dwóch wyznaczonych liczb. Powyższe obliczenia dla jednej kolumny wykonujemy w czasie $O(n)$.

Pozostała część algorytmu, czyli odszukanie łodzi na mapie oraz przeszukiwanie wszerz w poszukiwaniu najkrótszej drogi, są takie same jak w opisanym wcześniej rozwiązaniu wzorcowym. Tak jak w tamtym rozwiązaniu, trzeba pamiętać, że należy rozważać położenia łodzi także poza obszarem objętym mapą.

Opisane tu rozwiązanie zostało zaimplementowane w plikach `rob3.cpp`, `rob4.pas` oraz `rob7.java`. Testy pokazują, że jest ono nieco wolniejsze (o ok. 50%) niż wzorcowe. Również jego wymagania pamięciowe są większe. Oczywiście, uważnie zaimplementowane, przechodzi pomyślnie wszystkie testy.

Rozwiązania zbyt wolne

Zadanie bez problemu można rozwiązać w czasie $O(n^4)$ — na przykład sprawdzając każde potencjalne położenie łodzi w czasie $O(n^2)$ lub implementując przeszukiwanie wszerz, w którym możliwość wykonania każdego kroku jest sprawdzana w czasie $O(n^2)$. Ze względu na dopuszczalną wielkość danych, są to rozwiązania zdecydowanie za wolne. Warto jednak wspomnieć o pewnej heurystyce, która, wykorzystana w takich prostych rozwiązaniach, może je istotnie usprawnić (nie zmieniając jednakże ich złożoności czasowej).

Dla łodzi Robinsona możemy zakreślić najmniejszy prostokąt, w którym się ona mieści. Gdyby dało się łatwo stwierdzić, że w tym prostokącie nie ma żadnych przeszkód, to moglibyśmy uniknąć pracochłonnego, „ręcznego" przeszukiwania tego fragmentu mapy. Badanie prostokątów okazuje się całkiem proste. Wystarczy dla każdego pola mapy (x,y) wyznaczyć wartość $ileszu_{x,y}$ oznaczającą liczbę pól z przeszkodami o obu współrzędnych nie większych niż, odpowiednio, x i y. Wartości $ileszu_{x,y}$ są powiązane zależnością

$$ileszu_{x,y} = ileszu_{x-1,y} + ileszu_{x,y-1} - ileszu_{x-1,y-1} + \delta_{x,y},$$

gdzie $\delta_{x,y}$ to 1 lub 0 w zależności od tego, czy na polu (x,y) jest przeszkoda, czy jej nie ma. Wszystkie wartości $ileszu_{x,y}$ można zatem wyznaczyć w czasie $O(n^2)$. Wówczas, dla określonej pozycji łodzi, liczbę przeszkód w ograniczającym ją prostokącie możemy wyrazić jako liniową kombinację czterech wartości typu $ileszu_{x,y}$[3]. Jeśli wartość ta jest niezerowa, to musimy sprawdzić poprawność pozycji ręcznie. Jeśli jednak wynosi zero, to jesteśmy pewni, że pozycja nie koliduje z żadnymi przeszkodami.

Rozwiązanie o złożoności $O(n^4)$, wykorzystujące opisaną heurystykę, zostało zaimplementowane w plikach `robs3.cpp`, `robs6.pas` oraz `robs8.c`.

Istnieje także nietrywialne rozwiązanie działające w czasie $O(n^3)$, którego pomysł warto przedstawić. Częścią decydującą o złożoności wszystkich opisanych rozwiązań jest określenie, które pozycje są dozwolone dla łodzi Robinsona. W dotychczas zaprezentowanych algorytmach było to wykonywane przed samym przeszukiwaniem wszerz. Tym razem będziemy tę informację wyliczać na bieżąco, w momencie przesuwania łodzi. Wykorzystamy przy tym własność, o której już pisaliśmy — aby przekonać się, czy można przesunąć łódź w określonym kierunku, sprawdzimy tylko, czy nie ma przeszkód wzdłuż odpowiedniego brzegu łodzi.

Niestety, ponieważ badany brzeg łodzi może mieć nieregularny kształt, sprawdzenie możliwości wykonania ruchu zajmuje czas $O(n)$. Cały algorytm, w którym analizujemy $O(n^2)$ pól, ma więc złożoność $O(n^3)$.

Powyższe rozwiązanie zostało zaimplementowane w pliku `robs5.cpp`, a połączone jeszcze z heurystyką „pustego prostokąta" — w plikach `robs4.cpp`, `robs7.pas` i `robs9.c`. Algorytm ten, dobrze zaimplementowany, otrzymywał $60-70\%$ punktów.

Rozwiązania niepoprawne

Przykrym błędem, jaki można popełnić, jest badanie brzegu łodzi (w rozwiązaniu $O(n^3)$) w niepoprawny sposób — poprzez sprawdzanie jedynie skrajnych punktów kolejnych

[3]Podobny przykład zastosowania metody programowania dynamicznego do wyznaczania pewnych sum dla prostokątów można znaleźć w opisie rozwiązania zadania *Mapa gęstości* z VIII Olimpiady Informatycznej.

przekrojów poprzecznych. Rozwiązanie to znajduje się w pliku `robb2.cpp`, będącym wariantem programu `robs4.cpp`.

Innym wartym wspomnienia błędem jest potraktowanie heurystyki „pustego prostokąta" jako jedynego kryterium tego, czy pozycja jest dozwolona. To oczywiście powoduje pominięcie niektórych dozwolonych pozycji. Podobny błąd można popełnić, rozstrzygając, czy łódź wypłynęła już na pełne morze, za pomocą przecięcia minimalnego prostokąta otaczającego łódź z mapą. Programy z opisanymi błędami zapisane są w plikach `robb1.cpp` i `robb3.cpp`.

Oprócz różnego rodzaju pomyłek implementacyjnych oraz opisanych wyżej przypadków, w rozwiązaniach zawodników nie wystąpiły żadne inne typowe błędy.

Testy

Przygotowane zostało 10 zestawów testowych, z których większość stanowią grupy kilku testów.

Testy od drugiego włącznie zostały wygenerowane automatycznie. Kilka z nich było generowanych za pomocą mechanizmu przekazywania procedurze pomocniczej położeń łodzi, które mają być na planszy dozwolone i nakazania jej wypełnienia prawie wszystkich (poza pewnymi losowymi) pozostałych pól przeszkodami. W procedurze tej do konstrukcji mapy zastosowano algorytm podobny do rozwiązania wzorcowego. Testy tego typu są nazwane *zadanymi polami*. W większości takich testów łódka ma kształt losowy, ale w miarę równomiernie rozszerzający się i zwężający.

W testach z odpowiedzią `NIE` wyjście poza mapę uniemożliwia zazwyczaj jedno lub kilka pól z sitowiem, tak aby trudno było tę sytuację wykryć.

Opis każdego testu zawiera kolejno: wartość n, wysokość i szerokość łodzi (w orientacji, w której oś symetrii przebiega poziomo), orientację osi symetrii łodzi (poziomą, pionową lub symetryczną, czyli posiadającą i pionową, i poziomą oś symetrii), wartość wyniku oraz krótki opis testu.

Nazwa	n	w	s	oś	wyn	Opis
rob1a.in	25	9	9	–	42	ręcznie generowany test poprawnościowy
rob1b.in	3	3	3	+	3	skrajnie mały test bez pól z sitowiem
rob2.in	100	5	5	–	135	ręcznie generowany test poprawnościowy zadany polami
rob3a.in	1 000	3	3	+	248 755	najmniejsza możliwa łódź, która musi przepłynąć przez całą powierzchnię mapy, aby się z niej wydostać
rob3b.in	1 000	3	3	+	*NIE*	taki jak poprzedni, ale z zablokowanym wyjściem

Nazwa	n	w	s	oś	wyn	Opis
rob4.in	1500	99	98	|	3949	bardzo zagmatwany test zadany polami z niewielką łódką
rob5.in	2000	793	601	|	2882	zadany polami test z dużą łódką, mającą dużo dłuższy dziób niż rufę
rob6a.in	999	273	300	–	1192	zadany polami test zawierający basen w kształcie przekrzywionego prostokąta
rob6b.in	999	305	300	|	*NIE*	jak poprzedni, ale bez możliwości wypłynięcia
rob7a.in	1000	401	401	–	1595	zadany polami test z łódką w kształcie strzałki
rob7b.in	1000	401	401	+	1599	zadany polami podobny test, ale z łódką w kształcie krzyża
rob7c.in	1000	401	401	+	*NIE*	jak 7b, ale bez możliwości wypłynięcia
rob8a.in	2000	1401	1472	|	2627	test z dużą łodzią, zadany polami
rob8b.in	2000	1503	1500	–	*NIE*	jak poprzedni, ale bez możliwości wypłynięcia
rob9a.in	1495	759	743	–	2100	duża łódź w prostokątnym basenie otoczonym murkiem z jedną dziurą
rob9b.in	1495	759	743	|	*NIE*	jak poprzedni, ale bez możliwości wypłynięcia
rob10a.in	2000	1981	1979	–	3001	ogromna łódź, zajmująca większą część planszy
rob10b.in	2000	1981	1979	|	*NIE*	jak poprzedni, ale bez możliwości wypłynięcia