Marcin Kubica
Treść zadania, Opracowanie

Marek Żylak
Program

OI, Etap II, dzień pierwszy, 07.02.2007

Powódź

Bajtogród, stolica Bajtocji, to malownicze miasto położone w górskiej kotlinie. Niestety ostatnie ulewne deszcze spowodowały powódź — cały Bajtogród znalazł się pod wodą. Król Bajtocji, Bajtazar, zebrał swoich doradców, w tym i Ciebie, aby radzić, jak ratować Bajtogród. Postanowiono sprowadzić pewną liczbę pomp, rozmieścić je na zalanym terenie i za ich pomocą osuszyć Bajtogród. Bajtazar poprosił Cię o określenie minimalnej liczby pomp wystarczającej do osuszenia Bajtogrodu.

Otrzymałeś mapę miasta wraz z kotliną, w której jest położone. Mapa ma kształt prostokąta podzielonego na $m \times n$ jednostkowych kwadratów. Dla każdego kwadratu jednostkowego mapa określa jego wysokość nad poziomem morza oraz czy należy on do Bajtogrodu czy nie. Cały teren przedstawiony na mapie znajduje się pod wodą. Ponadto jest on otoczony dużo wyższymi górami, które uniemożliwiają odpływ wody. Osuszenie terenów, które nie należą do Bajtogrodu, nie jest konieczne.

Każdą z pomp można umieścić na jednym z kwadratów jednostkowych przedstawionych na mapie. Będzie ona wypompowywać wodę aż do osuszenia kwadratu, na którym się znajduje. Osuszenie dowolnego kwadratu pociąga za sobą obniżenie poziomu wody lub osuszenie kwadratów jednostkowych, z których woda jest w stanie spłynąć do kwadratu, na którym ustawiono pompę. Woda może bezpośrednio przepływać tylko między kwadratami, których rzuty na poziomą płaszczyznę mają wspólną krawędź.

Zadanie

Napisz program, który:

- *wyczyta ze standardowego wejścia opis mapy,*
- *wyznaczy minimalną liczbę pomp potrzebnych do osuszenia Bajtogrodu,*
- *wypisze wynik na standardowe wyjście.*

Wejście

W pierwszym wierszu standardowego wejścia znajdują się dwie liczby całkowite m i n, oddzielone pojedynczym odstępem, $1 \leqslant m,n \leqslant 1000$. Kolejnych m wierszy zawiera opis mapy. Wiersz $i+1$-szy opisuje i-ty pas kwadratów jednostkowych na mapie. Zawiera on n liczb całkowitych $x_{i,1}, x_{i,2}, \ldots, x_{i,n}$, pooddzielanych pojedynczymi odstępami, $-1000 \leqslant x_{i,j} \leqslant 1000$, $x_{i,j} \neq 0$. Liczba $x_{i,j}$ opisuje j-ty kwadrat w i-tym wierszu. Grunt w tym kwadracie znajduje się na wysokości $|x_{i,j}|$. Jeżeli $x_{i,j} > 0$, to kwadrat ten znajduje się na terenie Bajtogrodu, a w przeciwnym przypadku znajduje się poza miastem. Teren Bajtogrodu nie musi stanowić spójnej całości, może być podzielony na kilka oddzielonych od siebie części.

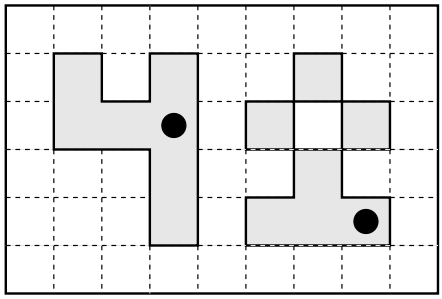
Wyjście

Twój program powinien wypisać na standardowe wyjście jedną liczbę całkowitą — minimalną liczbę pomp potrzebnych do osuszenia Bajtogrodu.

Przykład

Dla danych wejściowych:

```
6 9
-2 -2 -1 -1 -2 -2 -2 -12 -3
-2 1 -1 2 -8 -12 2 -12 -12
-5 3 1 1 -12 4 -6 2 -2
-5 -2 -2 2 -12 -3 4 -3 -1
-5 -6 -2 2 -12 5 6 2 -1
-4 -8 -8 -10 -12 -8 -6 -6 -4
```



poprawnym wynikiem jest:

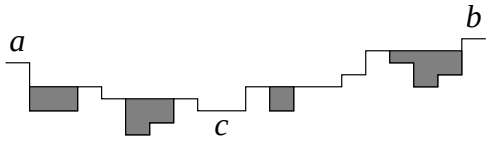
```
2
```

Na rysunku przedstawiono teren Bajtogrodu oraz przykładowe rozmieszczenie pomp.

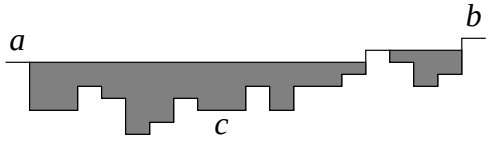
Rozwiązanie

Analiza problemu

Zaczniemy od prostej obserwacji. Niech a, b i c będą takimi kwadratami jednostkowymi, że pompa umieszczona w kwadracie c osusza zarówno kwadrat a, jak i b. Oznaczmy wysokości tych kwadratów, odpowiednio, przez h_a, h_b i h_c. Dodatkowo możemy założyć, że $h_a \leqslant h_b$ (jeśli tak nie jest, to zamieniamy oznaczenia kwadratów a i b). Wówczas pompa umieszczona w kwadracie a osusza kwadrat b.



Rys. 1: Pompa umieszczona w kwadracie c osusza kwadraty a i b.



Rys. 2: Pompa umieszczona w kwadracie a osusza kwadrat b.

Dzieje się tak, gdyż z a do c musi istnieć droga, którą woda spływa z a do c i na której wysokość terenu nie przekracza h_a. Podobnie, z b do c musi istnieć droga, którą woda spływa z b do c i na której wysokość terenu nie przekracza h_b. Tak więc istnieje droga z b do a, na której wysokość terenu nie przekracza h_b i którą woda może spłynąć z kwadratu b do pompy umieszczonej w kwadracie a.

Dzięki tej obserwacji możemy pokazać następujący fakt:

Fakt 1 *Niech a będzie najniżej położonym kwadratem Bajtogrodu — dowolnie wybranym, jeśli istnieje wiele takich kwadratów. Istnieje takie optymalne ustawienie pomp, w którym jedna z pomp znajduje się w kwadracie a.*

Dowód Rozważmy pewne optymalne ustawienie pomp i załóżmy, że w ustawieniu tym w kwadracie a nie stoi żadna pompa. Niech c będzie kwadratem, na którym znajduje się pompa osuszająca kwadrat a. (Jeśli jest kilka takich pomp, to wybieramy dowolną z nich.) Niech b będzie dowolnym kwadratem na terenie Bajtogrodu, osuszanym przez pompę znajdującą się w kwadracie c. Przenieśmy pompę z kwadratu c do kwadratu a — z poprzednio uczynionej obserwacji wynika, że przeniesiona pompa nadal osusza kwadrat b. Powyższe uzasadnienie można zastosować do dowolnie wybranego kwadratu b, z czego wynika, że pompa po przeniesieniu osusza te same kwadraty Bajtogrodu, co poprzednio. W ten sposób uzyskaliśmy optymalne ustawienie pomp, w którym w kwadracie a stoi pompa. ■

Fakt ten pozwala sformułować następujący zachłanny algorytm rozmieszczania pomp:

- umieszczamy pompę w najniżej położonym kwadracie Bajtogrodu — dowolnie wybranym, jeśli jest ich wiele,
- znajdujemy wszystkie kwadraty należące do Bajtogrodu, które osusza pompa,
- zmniejszamy problem, usuwając z terenu Bajtogrodu osuszone kwadraty,
- umieszczamy kolejną pompę itd.

Poniższe rysunki ilustrują zastosowanie tego algorytmu dla przykładowych danych z treści zadania.

2	2	1	1	2	2	2	12	3
2	1	1	2	8	12	2	12	12
5	3	1	①1	12	4	6	2	2
5	2	2	2	12	3	4	3	1
5	6	2	2	12	5	6	2	1
4	8	8	10	12	8	6	6	4

Pierwsza pompa jest ustawiana na jednym z kwadratów Bajtogrodu na wysokości 1. Osuszony przez nią teren zaznaczono na szaro.

2	2	1	1	2	2	2	12	3
2	1	1	2	8	12	2	12	12
5	3	1	①1	12	4	6	2	2
5	2	2	2	12	3	4	3	1
5	6	2	2	12	5	6	②2	1
4	8	8	10	12	8	6	6	4

Druga pompa jest ustawiana na wysokości 2. Osusza ona pozostały teren Bajtogrodu.

Rozwiązanie nieoptymalne

Prosta implementacja przedstawionego w poprzednim punkcie algorytmu może polegać na przeglądaniu kwadratów należących do Bajtogrodu w kolejności rosnących wysokości oraz symulowaniu obniżania się poziomu wody po ustawieniu każdej pompy. Do symulowania skutków ustawienia pojedynczej pompy możemy zastosować technikę podobną do algorytmu Dijkstry: kwadraty jednostkowe traktujemy jak wierzchołki grafu, krawędziami łączymy kwadraty sąsiadujące ze sobą — jednak zamiast rozpatrywać wierzchołki w kolejności rosnącej odległości od pompy, rozpatrujemy je w kolejności rosnącej wysokości, na którą opada woda. W algorytmie Dijkstry do przechowywania dostępnych w kolejnym kroku wierzchołków wykorzystywana jest kolejka priorytetowa. W naszym przypadku wysokości, według których wybieramy kwadraty, są liczbami całkowitymi od 1 do 1 000 — można więc do ich zapisania użyć tablicy zbiorów (lub jakichkolwiek kolekcji, np. list) kwadratów.

Pojedyncze przeszukiwanie ma złożoność czasową rzędu $O(m \cdot n)$. Tak więc cały algorytm działa w czasie $O(m \cdot n \cdot p)$, gdzie p jest liczbą pomp, które należy ustawić.

Rozwiązanie optymalne

Opisany w poprzednim punkcie algorytm można usprawnić, łącząc kolejne przeszukiwania w jedną całość. W tym celu zdefiniujemy dla każdej wysokości h dwa zbiory kwadratów:

- M_h — zbiór kwadratów, które leżą na terenie Bajtogrodu i są położone na wysokości h (wszystkie zbiory M_h możemy wyznaczyć na początku, przeglądając raz całą mapę) oraz
- L_h — zbiór kwadratów leżących na wysokości h, które sąsiadują z kwadratami, na których woda opadła poniżej poziomu h (początkowo wszystkie zbiory L_h są puste; zapełniamy je w trakcie obliczeń, gdy stwierdzimy, że w sąsiedztwie kwadratu leżącego na wysokości h woda opadła poniżej tego poziomu).

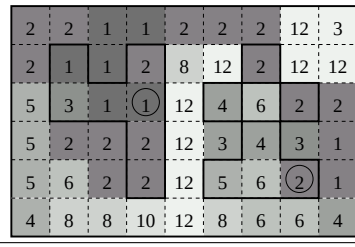
Dla każdego kwadratu pamiętamy także poziom wody w tym kwadracie (początkowo, we wszystkich kwadratach woda jest na poziomie ∞).

W algorytmie przeglądamy wysokości h w rosnącej kolejności, $h = 1, 2, \ldots, 1\,000$, ustawiając w miarę potrzeby pompy i osuszając wszystkie kwadraty na danej wysokości (i być może inne przy okazji). Kwadraty należące do L_h zostały osuszone wcześniej przez pompy postawione niżej, gdyż woda spłynęła z nich do sąsiadujących kwadratów, gdzie jej poziom jest niższy. Możemy więc wyznaczyć wszystkie kwadraty, w których woda opadnie do poziomu h (osuszając je lub nie), przeszukując mapę wszerz (BFS) lub w głąb (DFS), począwszy od kwadratów należących do L_h. Gdy w trakcie przeszukiwania napotkamy na kwadrat sąsiadujący z odwiedzanym kwadratem, ale położony na wysokości $h' > h$, to wstawiamy go do zbioru $L_{h'}$.

Po wykonaniu powyższej procedury sprawdzamy, czy w zbiorze M_h pozostał nieosuszony kwadrat — jeśli tak, to należy w nim ustawić pompę i uzupełnić przeszukiwanie, zaczynając je w tym kwadracie. Postępujemy w ten sposób tak długo, aż wszystkie kwadraty ze zbioru M_h zostaną osuszone.

Dzięki temu, że wysokości h rozpatrujemy w rosnącej kolejności, każdy kwadrat będziemy odwiedzać tylko raz, wyznaczając właściwy poziom, na jaki opadnie w nim woda.

W szczególności, kwadraty należące do Bajtogrodu zostaną zawsze całkowicie osuszone, natomiast pozostałe mogą co najmniej częściowo pozostać pod wodą. W rezultacie, złożoność czasowa i pamięciowa takiego algorytmu wynosi $O(m \cdot n)$.



Rys. 3: Kwadraty mapy są odwiedzane w kolejności zgodnej z poziomem, na jaki opada na nich woda (reprezentowanym przez odcienie szarości).

Jeżeli do przeszukiwania mapy zastosujemy przeszukiwanie w głąb, to oba zbiory L_h i M_h możemy trzymać na jednym stosie. Musimy wówczas zadbać, by elementy zbioru L_h znajdowały się na wierzchu stosu, a elementy zbioru M_h na dole stosu. W trakcie przeszukiwania musimy bowiem najpierw wyznaczyć kwadraty, z których woda spłynie za pośrednictwem kwadratów już osuszonych (L_h) i dopiero potem możemy przystąpić do rozstawiania nowych pomp na zalanych jeszcze kwadratach z M_h. Takie właśnie rozwiązanie zaimplementowano w plikach `pow.cpp` i `pow1.pas`.

Rozwiązanie alternatywne

Zadanie można również rozwiązać w odrobinę gorszej złożoności czasowej niż optymalna, używając struktury *Find-Union*[1]. Wyobraźmy sobie proces odwrotny do osuszania — przypuśćmy, że poziom wody stopniowo podnosi się, zalewając coraz większe obszary.

Niech a będzie kwadratem leżącym na terenie Bajtogrodu na wysokości h_a. Przyjrzyjmy się, jak wygląda „jezioro", które zaleje kwadrat a, gdy poziom wody przekroczy h_a (ale będzie jeszcze niższy niż $h_a + 1$). Gdyby gdziekolwiek na terenie zalanym przez to jezioro znajdowała się pompa, kwadrat a zostałby osuszony. Obserwacja ta prowadzi do następującego algorytmu:

- sortujemy kwadraty według wysokości — możemy tu zastosować sortowanie przez zliczanie, działające w czasie $O(m \cdot n)$;
- następnie przeglądamy kwadraty w kolejności niemalejących wysokości symulując podnoszenie się poziomu wody i śledzimy zasięg jezior przy aktualnym poziomie.

Przeglądając kwadrat a, rozważamy, co się z nim dzieje w momencie, gdy poziom wody przekroczy h_a. Początkowo dla kwadratu a tworzymy osobny element — jezioro jednostkowe, które łączymy z jeziorami odpowiadającymi tym sąsiadom kwadratu a, którzy są położeni nie wyżej niż on. W ten sposób otrzymujemy element reprezentujący jezioro

[1] Struktury *Find-Union* służą do przechowywania elementów pogrupowanych w rozłączne zbiory i efektywnego wykonywania operacji: połączenia zbiorów (*Union*) oraz zidentyfikowania zbioru, do którego należy element (*Find*). Więcej na ich temat można przeczytać w książkach [14] i [19].

pokrywające kwadrat a. Oczywiście, w miarę rozpatrywania coraz wyżej położonych kwadratów (czyli podnoszenia się poziomu wody), jeziora mogą łączyć się — do śledzenia tego procesu przydaje się nam struktura *Find-Union*.

Dodatkowo, dla każdego jeziora pamiętamy, czy na jego terenie znajduje się jakaś pompa. Po przejrzeniu wszystkich kwadratów znajdujących się na wysokości h, przeglądamy ponownie te z nich, które leżą na terenie Bajtogrodu. Dla każdego takiego kwadratu sprawdzamy, czy na terenie pokrywającego go jeziora znajduje się pompa — jeżeli nie, to ustawiamy pompę w tym kwadracie i odnotowujemy, że w odpowiednim jeziorze jest już pompa. Łącząc dwa jeziora, przyjmujemy, że jeżeli na terenie któregokolwiek z nich znajduje się pompa, to na terenie jeziora powstałego w wyniku połączenia też znajduje się pompa.

Łączna liczba kwadratów to $m \cdot n$, czyli *zamortyzowany*[2] czas jednej operacji na strukturze *Find-Union* wynosi $O(\log^*(m \cdot n))$. Tak więc, łączny czas działania całego algorytmu to $O(m \cdot n \cdot \log^*(m \cdot n))$. W praktyce, złożoność tego algorytmu nie odbiega znacząco od algorytmu wzorcowego. Rozwiązanie to zaimplementowano w pliku `pow2.cpp`.

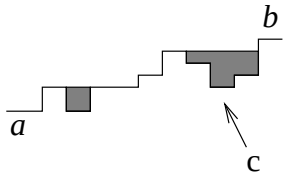
Ciekawe rozwiązanie prawie poprawne

Na koniec prezentujemy jeszcze jedno ciekawe rozwiązanie, bardzo zbliżone do opisanych już rozwiązań nieoptymalnego i optymalnego. Przeglądamy wysokości pól w rosnącej kolejności $h = 1, 2, \ldots, 1\,000$ i każdorazowo w przypadku napotkania nieosuszonego pola Bajtogrodu stawiamy w nim pompę. Różnica między tym podejściem, a wcześniej zaprezentowanymi, jest taka, że osuszanie symulujemy za pomocą algorytmu DFS, poruszając się wyłącznie „pod górę". Innymi słowy, jeżeli w pewnym momencie algorytmu znajdujemy się w polu o wysokości h, to przeszukiwanie kontynuujemy w nieosuszonych jeszcze kwadratach sąsiednich o wysokościach nie mniejszych niż h. W ten sposób mamy gwarancję, że zawsze w momencie odwiedzenia pola zostaje ono całkowicie osuszone. To oznacza, że każde pole zostanie odwiedzone w algorytmie dokładnie raz, co daje złożoność czasową opisanego algorytmu $O(m \cdot n)$. Rozwiązanie to zaimplementowano w pliku `powb3.cpp`.

Opisane podejście nie jest niestety poprawne, o czym można się przekonać, analizując chociażby rysunek 1. Faktycznie, czasem może się opłacać odwiedzić pole, nie osuszając go całkowicie, jeżeli jest to pole leżące poza Bajtogrodem. Okazuje się jednak, że w przypadku, gdy *każde pole kotliny należy do Bajtogrodu* (czyli ma on $m \cdot n$ pól), to zaprezentowany algorytm jest poprawny! Uzasadnimy to przez sprowadzenie do sprzeczności. Załóżmy, że kwadrat b jest *najniższym* kwadratem, który został osuszony przez pompę ustawioną w naszym algorytmie w jakimś kwadracie a, ale którego osuszenie zostało „przeoczone" ze względu na ograniczony sposób przeszukiwania kwadratów. Dla uproszczenia dowodu załóżmy, że wszystkie kwadraty w kotlinie mają różne wysokości. Ponieważ pompa w a osusza b, to istnieje ścieżka z a do b, na której wysokości wszystkich pól są nie większe niż wysokość kwadratu b. Rozważana ścieżka musi zawierać pewne obniżenia wysokości, gdyż w przeciwnym przypadku kwadrat b zostałby uznany przez nasz algorytm za osuszony (patrz rys. 4). Niech c oznacza pierwszy kwadrat na ścieżce, od którego biegnie ona już tylko pod górę (kolejne odwiedzane kwadraty mają wysokości nie mniejsze niż poprzednie). Ponieważ

[2] O zamortyzowanym sposobie liczenia czasu operacji można przeczytać w tych samych książkach, w których znajduje się opis struktur *Find-Union* — [14], [19].

przez b oznaczyliśmy najniższe „przeoczone” pole, to kwadrat c musiał zostać uznany za osuszony w naszym algorytmie. Skoro tak, to w momencie, gdy przeszukiwanie dotarło do kwadratu c, osuszając go, musiało ono być kontynuowane aż do dotarcia kwadratu b, co stanowi sprzeczność z założeniem o przeoczeniu b.



Rys. 4: Ścieżka z kwadratu a do b zawierająca obniżenia.

Rodzi się więc pytanie, czy opisany algorytm może się do czegokolwiek przydać w przypadku ogólnym, kiedy niekoniecznie wszystkie kwadraty należą do Bajtogrodu. W takiej sytuacji można zastosować rozwiązanie, będące hybrydą opisanego algorytmu i zaprezentowanego wcześniej rozwiązania nieoptymalnego: jeżeli w trakcie przeszukiwania (symulacji osuszania) natrafiamy na pole Bajtogrodu, to odwiedzamy je tylko wówczas, gdy spowoduje to całkowite jego osuszenie; jeżeli natomiast natrafiamy na pole spoza Bajtogrodu, to odwiedzamy je pod warunkiem, że spowoduje to obniżenie dotychczasowego poziomu wody nad tym polem. Dowód poprawności opisanego podejścia pozostawiamy Czytelnikowi, po szczegóły odsyłając równocześnie do implementacji w pliku `pows5.cpp`. Na koniec wspomnijmy, że mimo pesymistycznej złożoności czasowej $O(n \cdot m \cdot h_{max})$, opisane rozwiązanie świetnie spisuje się dla losowych danych testowych.

Testy

Rozwiązania zawodników były testowane na 57 testach pogrupowanych w 40 zestawów. Nie będziemy ich tu szczegółowo i z osobna opisywać, lecz poprzestaniemy jedynie na zbiorczej charakterystyce:

- testy 1–9 to nieduże testy poprawnościowe, przez które powinny przejść wszystkie rozwiązania poprawne, a nawet niektóre sprytne rozwiązania błędne; większość tych testów zawiera mapy złożone z zaledwie kilku kwadratów, tylko trzy z nich zawierają mapy w kształcie paska o wymiarach 1×500 lub $1 \times 1\,000$,
- testy 10–17 to mniejsze testy losowe, przez które przechodzą wszystkie poprawne rozwiązania (jakich się spodziewano); zawierają one od 25 do 93 000 kwadratów,
- testy 18–23 to duże testy losowe, w których teren zajmowany przez Bajtogród stanowi niewielką część terenu pokazanego na mapie; przez te testy przechodzą wszystkie przedstawione tu rozwiązania; testy te zawierają od 810 000 do 1 000 000 kwadratów,
- grupy testów 24–30 zawierają po dwa testy, jeden test w każdej grupie służy do odróżnienia rozwiązania optymalnego i alternatywnego od rozwiązań wolniejszych, a drugi test to mniejszy test sprawdzający poprawność rozwiązania; przez te testy przechodzi rozwiązanie wzorcowe i alternatywne; większe z testów w grupach zawierają od 324 900 do 1 000 000 kwadratów,

- grupy testów 31–39 zawierają po dwa testy, jeden test w każdej grupie służy do odróżnienia rozwiązania optymalnego i alternatywnego od rozwiązań wolniejszych, a drugi test to duży test losowy, w którym teren zajmowany przez Bajtogród stanowi niewielką część terenu pokazanego na mapie; przez testy te przechodzi rozwiązanie wzorcowe i alternatywne; zawierają one od 384 400 do 1 000 000 kwadratów,
- ostatnia grupa testów (40) to dwa testy maksymalnej wielkości: jeden specyficzny i jeden losowy, dobrane tak, żeby przechodziły przez nie jedynie rozwiązania optymalne i alternatywne.