Jakub Radoszewski
Treść zadania, Opracowanie

Jakub Radoszewski, Szymon Wąsik
Program

OI, Etap I

Drzewa

Bajtazar ma domek na wsi. Niedawno kupił on n drzew i zlecił swojemu ogrodnikowi posadzenie ich w jednym rzędzie. Gdy ogrodnik posadził drzewa, Bajtazarowi nie spodobała się kolejność, w jakiej zostały one posadzone. Drażni go, że drzewa niskie i wysokie zostały pomieszane ze sobą i całość nie wygląda zbyt estetycznie.

Bajtazar, chcąc wyjaśnić ogrodnikowi, jaki jest jego cel, zdefiniował **współczynnik nieporządku** *rzędu drzew jako:* $|h_1 - h_2| + |h_2 - h_3| + \cdots + |h_{n-1} - h_n|$*, gdzie* $h_1, h_2, \ldots, h_n$ *to wysokości kolejnych drzew w rzędzie. Im mniejsza wartość współczynnika nieporządku, tym ładniej wygląda rząd drzew.*

Przesadzanie drzew jest bardzo pracochłonne i kłopotliwe. Dlatego też Bajtazar zlecił ogrodnikowi przesadzenie co najwyżej dwóch drzew (tak, że zostaną one zamienione miejscami). Ogrodnik ma tak wybrać drzewa do przesadzenia, aby współczynnik nieporządku rzędu drzew stał się jak najmniejszy.

Ogrodnik nie jest pewien, czy właściwie wybrał drzewa do przesadzenia, a boi się, że w razie pomyłki straci pracę. Pomóż mu i oblicz, dla każdego drzewa, jaki najmniejszy współczynnik nieporządku może zostać uzyskany poprzez ewentualną zamianę jego pozycji z innym drzewem.

Zadanie

Napisz program, który:

- *wczyta ze standardowego wejścia wysokości drzew w rzędzie,*
- *dla każdego drzewa obliczy najmniejszy współczynnik nieporządku, jaki może być uzyskany, jeżeli rozpatrywane drzewo zostanie zamienione pozycją z pewnym innym lub żadne drzewo nie będzie przesadzane,*
- *wypisze wynik na standardowe wyjście.*

Wejście

Pierwszy wiersz wejścia zawiera jedną liczbę całkowitą n *(*$2 \leqslant n \leqslant 50\,000$*). Drugi wiersz wejścia zawiera* n *liczb całkowitych* h_i *(*$1 \leqslant h_i \leqslant 100\,000\,000$*), pooddzielanych pojedynczymi odstępami i oznaczających wysokości kolejnych drzew w rzędzie.*

Wyjście

Wyjście powinno zawierać dokładnie n *wierszy. Wiersz* i*-ty powinien zawierać dokładnie jedną liczbę całkowitą — najmniejszy współczynnik nieporządku możliwy do uzyskania przez przesadzenie* i*-tego drzewa.*

Przykład

Dla danych wejściowych:	*poprawnym wynikiem jest:*
`5`	`7`
`7 4 5 2 5`	`7`
	`8`
	`7`
	`7`
Natomiast dla danych:	*poprawnym wynikiem jest:*
`5`	`4`
`1 2 3 4 5`	`4`
	`4`
	`4`
	`4`

W pierwszym przykładzie współczynnik nieporządku równy 7 może być uzyskany poprzez przesadzenie drzew numer 1 i 4, 2 i 5 lub 4 i 5. Tak więc przesadzając każde z wymienionych w poprzednim zdaniu drzew (1, 2, 4 i 5) z odpowiednio wybranym innym drzewem, można uzyskać współczynnik nieporządku 7. Jedynie dla drzewa 3 najlepszy możliwy do uzyskania współczynnik nieporządku jest większy i wynosi 8. W drugim przykładzie przesadzenie dowolnych dwóch drzew może jedynie powiększyć współczynnik nieporządku, więc żadna zamiana nie powinna mieć miejsca i wszystkie współczynniki nieporządku równają się początkowemu współczynnikowi (4).

Rozwiązanie

Najprostsze rozwiązanie — symulacja

Najprostszy sposób rozwiązania zadania to zasymulowanie wszystkich możliwych przesadzeń i obliczenie, dla każdego z nich, współczynnika nieporządku powstałego szpaleru (czyli rzędu) drzew. Ponieważ musimy wówczas rozważyć $n(n-1)$ przesadzanych par i dla każdej z ich wyliczyć w czasie $O(n)$ współczynnik nieporządku, więc złożoność czasowa tego algorytmu wynosi $O(n^3)$. Jego implementacja znajduje się w plikach `drzs0.cpp` i `drzs1.pas`.

Rozwiązanie można łatwo usprawnić, korzystając z faktu, że przesadzenie dwóch drzew ma dość „lokalny” wpływ na współczynnik nieporządku. Jeśli wstępnie policzymy współczynnik nieporządku dla początkowego szpaleru, to po przesadzeniu wybranej pary drzew (powiedzmy zajmujących pozycje i oraz j dla $1 < i < i+1 < j < n$), wystarczy skorygować współczynnik o wartość:

$$
\begin{aligned}
& \big(|h_1-h_2|+|h_2-h_3|+\cdots+|h_{n-1}-h_n|\big)+\\
-\ & \big(|h_1-h_2|+\cdots+|h_{i-1}-h_j|+|h_j-h_{i+1}|+\cdots+\\
+\ & |h_{j-1}-h_i|+|h_i-h_{j+1}|+\cdots+|h_{n-1}-h_n|\big)=\\
=\ & \big(|h_{j-1}-h_i|+|h_i-h_{j+1}|+|h_{i-1}-h_j|+|h_j-h_{i+1}|\big)+\\
-\ & \big(|h_{j-1}-h_j|+|h_j-h_{j+1}|+|h_{i-1}-h_i|+|h_i-h_{i+1}|\big)
\end{aligned}
$$

Jak widać, możemy to zrobić w czasie stałym. Dla przypadków, gdy $i = 1$ lub $i+1 = j$, lub $j = n$ (czyli gdy przesadzamy drzewa skrajne lub sąsiadujące ze sobą), wzory są trochę inne, ale nadal bardzo proste do wyznaczenia.

Złożoność czasowa usprawnionej wersji rozwiązania to $O(n^2)$. Jej implementacja znajduje się w plikach `drzs2.cpp` oraz `drzs3.pas`.

Oba zaprezentowane rozwiązania są stosunkowo proste koncepcyjne, a ich zaprogramowanie nie stwarza zbytnich problemów. Większość zawodników nadesłała tego typu algorytmy, zdobywając za rozwiązania o złożoności $O(n^3)$ około 30 punktów na 100 możliwych, a za rozwiązania o złożoności $O(n^2)$ — od 50 do 70 punktów. Dodatkowe sposoby usprawnienia rozwiązania o złożoności $O(n^2)$ zostały przedstawione — w charakterze ciekawostki — na końcu niniejszego opracowania.

Rozwiązanie wzorcowe

Wprowadzenie

Rozwiązanie wzorcowe zadania *Drzewa* osiąga złożoność czasową zdecydowanie niższą niż rozwiązania symulacyjne — $O(n \log n)$. Jest oparte na całkowicie odmiennym podejściu, którego zrozumienie wymagać będzie od Czytelnika pewnej dozy cierpliwości. Jednak naszym zdaniem zdecydowanie warto się z nim zmierzyć, bo zaprezentowane pomysły są ciekawe i niebanalne. Implementacja tego rozwiązania także jest stosunkowo trudna i żmudna, przede wszystkim z powodu konieczności rozpatrywania wielu szczególnych przypadków. Po tym „zniechęcającym" wstępie, naprawdę szczerze zapraszamy Czytelników do dalszej lektury.

Drzewo *minTree*

W prezentowanym rozwiązaniu będziemy wykorzystywać strukturę danych, w której można przechowywać pary (*klucz*, *wartość*), gdzie *klucz*$\in [1,n]$. Na strukturze tej będziemy wykonywać operacje:

- *insert(k,w)* — wstawianie elementu o kluczu k i wartości w;
- *delete(k)* — usunięcie elementu o kluczu k,
- *min_val(l,p)* — znalezienie minimalnej wartości towarzyszącej elementom o kluczach z przedziału $[l,p] \subseteq [1,n]$.

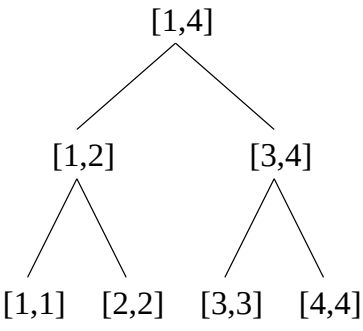
Dodatkowo wiemy, że klucze elementów zapisanych w strukturze nie powtarzają się, a dla prostoty implementacji można założyć, że n jest potęgą dwójki.

Strukturę danych *minTree* można efektywnie zrealizować za pomocą statycznego drzewa poszukiwań binarnych, tzw. *drzewa przedziałowego*. Jest to drzewo binarne, którego wierzchołki są związane z przedziałami zawartymi w $[1,n]$ w następujący sposób:

- korzeń drzewa reprezentuje przedział kluczy $[1,n]$;
- każdy węzeł wewnętrzny v drzewa, który reprezentuje niejednostkowy przedział $[l,r]$ (tzn. $r > l$), ma dwóch synów, którzy reprezentują odpowiednio przedziały kluczy: $[l, \lfloor \frac{l+r}{2} \rfloor]$ oraz $[\lfloor \frac{l+r}{2} \rfloor + 1, r]$;

- drzewo ma n liści, którym odpowiadają przedziały jednostkowe $[1,1],[2,2],\ldots,[n,n]$.

Przedziały reprezentowane przez węzły drzewa nazywamy *przedziałami bazowymi*. Przykład drzewa przedziałowego dla przedziału $[1,4]$ jest pokazany na rys. 1.



Rys. 1: Drzewo przedziałowe, skonstruowane dla przedziału kluczy $[1,4]$.

Dokładniejszy opis drzew przedziałowych można znaleźć na przykład w *Niebieskiej książeczce* z XIII Olimpiady Informatycznej ([13]) w opracowaniu zadania *Tetris 3D*. Tutaj ograniczymy się do przypomnienia ich najważniejszych z naszego punktu widzenia właściwości:

- głębokość drzewa (czyli długość najdłuższej ścieżki od korzenia do liścia) jest rzędu $O(\log n)$,
- liczba węzłów drzewa jest rzędu $O(n)$,
- każdy przedział $[l,r]$ zawarty w przedziale $[1,n]$ można rozłożyć na $O(\log n)$ parami rozłącznych przedziałów bazowych (tzn. każda liczba całkowita w nim zawarta znajdzie się w dokładnie jednym przedziale z rozkładu); przedziały i reprezentujące je wierzchołki można znaleźć w czasie $O(\log n)$.

Za pomocą drzewa przedziałowego możemy zaimplementować efektywnie *minTree*. W tym celu w każdym wierzchołku umieścimy dodatkowe pole *val* przeznaczone na wartość. Zadbamy o to, by w węźle reprezentującym przedział $[l,r]$ w polu *val* znajdowało się minimum wartości związanych z kluczami z przedziału $[l,r]$. W polu *val* liścia reprezentującego przedział $[i,i]$ zapiszemy więc wartość elementu o kluczu i. Jeśli takiego elementu nie ma w drzewie, to w liściu $[i,i]$ możemy wpisać wartość ∞. W polu *val* węzła v reprezentującego przedział $[l,r]$ zapiszemy minimum z wartości przypisanych elementom o kluczach z przedziału $[l,r]$. Zauważmy, że tę wartość możemy wyznaczyć jako minimum z pól *val* synów węzła v.

Operacje na *minTree* wykonamy w następujący sposób:

- Puste drzewo tworzymy, budując kompletną strukturę przedziałów i wypełniając wszystkie pola *val* wartościami ∞. Całą konstrukcję możemy zbudować w czasie $O(n)$. Warto także nadmienić, że drzewo, dzięki regularnej strukturze, można zaimplementować bez użycia wskaźników. Drzewo to „wkładamy" w tablicę $A[1..2n-1]$ poziomami: wówczas korzeń znajduje się w polu o indeksie 1, a synowie wierzchołka zapisanego w polu o indeksie i w polach o indeksach $2i$ oraz $2i+1$.

- Operację *insert*(k,w) wykonujemy, przechodząc ścieżką od korzenia do liścia $[k,k]$ i zmieniając w nim wartość *val* na w. Wracając od liścia $[k,k]$ do korzenia, aktualizujemy pola *val* we wszystkich węzłach napotkanych po drodze — zauważmy, że są to wszystkie węzły drzewa, reprezentujące przedziały zawierające przedział $[k,k]$. Operację wykonujemy w czasie $O(\log n)$, bo taka jest wysokość drzewa.

- Operację *delete*(k) można zrealizować, wykonując *insert*(k,∞). Koszt czasowy tej operacji to także $O(\log n)$.

- Znalezienie *min_val*(l,r) wymaga rozłożenia $[l,r]$ na przedziały bazowe, a następnie wyznaczenia minimum z wartości pól *val* dla węzłów drzewa reprezentujących te przedziały. Zarówno rozkład, jak i minimum, można znaleźć w czasie $O(\log n)$.

Zarys rozwiązania wzorcowego

Rozpoczniemy od wprowadzenia terminologii, która ułatwi nam opis rozwiązania.

W rozwiązaniu będziemy wykorzystywać drzewiaste struktury danych (*minTree*). Aby Czytelnik w każdym momencie opracowania wiedział czy słowo „drzewo" oznacza roślinę z treści zadania, czy strukturę danych, umówmy się, że wszystkie drzewa w ogrodzie Bajtazara to *cyprysy*.

Miejsca w ogrodzie Bajtazara są wstępnie ponumerowane kolejno liczbami $1,2,\ldots,n$, więc i-tym miejscem będziemy nazywać i-te miejsce w szpalerze. Wprowadzimy dodatkowe uporządkowanie. *Rangą* cyprysa nazwiemy jego pozycję w ciągu cyprysów uporządkowanych według wysokości, tzn. rangę 1 przypiszemy najniższemu cyprysowi, rangę 2 — drugiemu w kolejności itd.; najwyższy cyprys dostanie rangę n. Jednakowe cyprysy możemy uporządkować dowolnie — po przyjęciu tego porządku będziemy uważać, że wszystkie cyprysy mają różne wysokości i będziemy rozstrzygać remisy zgodnie z przyjętym porządkiem rang. Przez H_i oznaczymy rangę cyprysa rosnącego na i-tym miejscu.

Domyślnie miejsca będziemy oznaczać według numerów, a cyprysy według rang, tzn. mówiąc i-te miejsce, będziemy mieli na myśli i-te miejsce w szpalerze, a mówiąc i-ty cyprys — cyprys i-ty co do wielkości. Dodatkowo, przez D_i oznaczymy miejsce zajmowane przez i-ty cyprys. To oznacza, że najniższy cyprys stoi w szpalerze na miejscu D_1, drugi — na miejscu D_2 itd.; najwyższy stoi na miejscu D_n.

Przykład 1 Rozważmy pierwszy z przykładów w treści zadania, w którym $n = 5$, a wysokości kolejnych cyprysów w szpalerze to: $7,4,5,2,5$. Rangi H przypisane kolejnym cyprysom w szpalerze to: $5,2,4,1,3$ (przy innym rozstrzygnięciu remisu możliwe jest także przypisanie rang: $5,2,3,1,4$). Ciąg D to: $4,2,5,3,1$ (a w drugim przypadku $4,2,3,5,1$).

Cyprysy będziemy przeglądać w kolejności rosnących wysokości, czyli według rang $p = 1,2,\ldots,n$. To znaczy, że będziemy rozważać kolejno cyprysy stojące na pozycjach: $t = D_1,D_2,\ldots,D_n$. Dla każdego z nich wyznaczymy optymalną zmianę współczynnika nieporządku, jaką można uzyskać, przesadzając go z dowolnym innym (ewentualnie nic nie zmieniając w szpalerze). Przyjrzyjmy się więc teraz p-temu cyprysowi rosnącemu na t-tym miejscu i możliwościom, jakie stwarza jego przesadzenie.

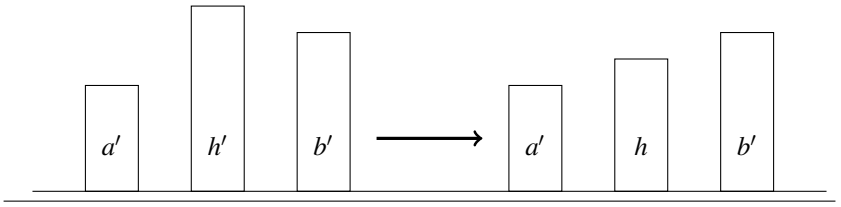
Zamianę cyprysa rosnącego na miejscu t-tym z innym, powiedzmy rosnącym na miejscu t'-tym, podzielimy na dwie operacje:

Faza 1: dokonamy zmian na miejscu t', to znaczy wyrwiemy cyprys z miejsca t' i na jego miejsce posadzimy p-ty cyprys,

Faza 2: dokonamy zmian na miejscu t, to znaczy wyrwiemy cyprys z miejsca t i w powstałej wyrwie posadzimy cyprys z miejsca t'.

Oznaczmy przez a, h oraz b wysokości cyprysów rosnących (początkowo) na miejscach $t-1$, t oraz $t+1$ (możemy bez straty ogólności założyć, że $a \leqslant b$, w przeciwnym razie zamienimy oznaczenia, przyjmując $a = h_{t+1}$ oraz $b = h_{t-1}$). Analogicznie oznaczymy przez a', h' oraz b' wysokości cyprysów z miejsc $t'-1$, t' oraz $t'+1$ (także zakładamy, że $a' \leqslant b'$).

Faza 1



Rys. 2: Efekt posadzenia cyprysa z miejsca t w miejscu t'.

Zmiany zachodzące w miejscu t' to wyrwanie cyprysa rosnącego w tym miejscu (powoduje to zmniejszenie współczynnika nieporządku o $|a'-h'|+|b'-h'|$) i posadzenie w powstałej wyrwie cyprysa o wysokości h (powoduje to wzrost współczynnika nieporządku o $|a'-h|+|b'-h|$). Zmianę współczynnika możemy zapisać jako:

1. $a'+b'-2h-|a'-h'|-|b'-h'| = -2h+(a'+b'-|a'-h'|-|b'-h'|)$, jeżeli $h \leqslant a'$,
2. $b'-a'-|a'-h'|-|b'-h'| = 0+(b'-a'-|a'-h'|-|b'-h'|)$, jeżeli $a' \leqslant h \leqslant b'$,
3. $2h-a'-b'-|a'-h'|-|b'-h'| = 2h+(-a'-b'-|a'-h'|-|b'-h'|)$, jeżeli $h \geqslant b'$.

Zauważmy, że każde z wyrażeń podzieliliśmy na dwie części: *składową cyprysa* (równą $-2h$, 0 lub $2h$) oraz *składową miejsca* (oznaczymy ją w kolejnych przypadkach $f_1(t')$, $f_2(t')$ oraz $f_3(t')$). Otrzymana postać wyrażenia skłania nas do następujących spostrzeżeń:

- z punktu widzenia cyprysa o wysokości h, pozycje t', na których możemy go posadzić, dzielą się na trzy grupy:
  - *doliny*, czyli miejsca, gdzie trafi pomiędzy dwa wyższe cyprysy, tzn. $h \leqslant a'$,
  - *zbocza*, gdzie będzie sąsiadował z wyższym i z niższym cyprysem, czyli $a' \leqslant h \leqslant b'$ oraz
  - *szczyty*, gdzie po obu stronach będzie miał niższe cyprysy, czyli $b' \leqslant h$;

  jeśli jakaś pozycja kwalifikuje się do więcej niż jednej grupy (z powodu równości), to możemy ją przydzielić dowolnie, ale tylko do jednej grupy;
- dla dolin powinniśmy policzyć wartości $f_1(t')-2h$, dla zboczy — wartości $f_2(t')$, dla szczytów — wartości $f_3(t')+2h$;

- optymalna pozycja docelowa t' dla rozważanego cyprysa o wysokości h to taka, dla której wartość z poprzedniego punktu jest minimalna (na razie nie martwimy się faktem, że trzeba będzie jeszcze posadzić cyprys wyrwany z pozycji t' — o tym później).

Pozostaje nam zrealizować zakreślony plan. Do przechowywania pozycji wraz ze składowymi miejsca wykorzystamy trzy drzewa *minTree*:

A — będziemy w nim przechowywać pary $(H_{t'}, f_1(t'))$, gdzie t' jest doliną dla rozważanego cyprysa (a $H_{t'}$, przypomnijmy, oznacza rangę cyprysa zajmującego miejsce t');

B — w tym drzewie będziemy przechowywać pary $(H_{t'}, f_2(t'))$, gdzie t' jest zboczem dla rozważanego cyprysa;

C — to drzewo przeznaczymy na pary $(H_{t'}, f_3(t'))$, gdzie t' jest szczytem dla rozważanego cyprysa.

Dzięki zastosowaniu struktur *minTree* będziemy w stanie znaleźć optymalną pozycję dla rozważanego cyprysa, wykonując operację $min\_val(1,n)$ dla każdego z drzew. Pozostaje tylko wyjaśnić, jak podzielić pozycje na doliny, zbocza i szczyty dla pierwszego rozważanego cyprysa i jak aktualizować ten podział przy przechodzeniu do kolejnych rangą cyprysów.

Teraz wyjaśni się, dlaczego wybraliśmy rozważanie cyprysów według rang. Otóż z punktu widzenia pierwszego, najniższego cyprysa *wszystkie pozycje są dolinami*, więc wszystkie powinny trafić do drzewa *A*. Następnie, załóżmy, że w trakcie rozważania p-tego cyprysa zajmującego miejsce t mamy pozycje poprawnie rozdzielone pomiędzy drzewa *A*, *B* i *C*. Wówczas prawie wszystkie pozycje są poprawnie sklasyfikowane także z punktu widzenia $(p+1)$-ego cyprysa. Jeśli bowiem na pozycjach sąsiednich względem rozważanej pozycji t' stoją cyprysy o rangach z przedziałów $[1,p-1] \cup [p+1,n]$, to relacja ich wysokości jest taka sama w stosunku do p-tego oraz $(p+1)$-szego cyprysa. Klasyfikacja może ulec zmianie jedynie dla pozycji $t' = t-1$ oraz $t' = t+1$, które sąsiadują z p-tym cyprysem. Jeśli któraś z tych pozycji była doliną dla p-tego cyprysa i p-ty cyprys był niższym z jej sąsiadów, to dla cyprysa $(p+1)$-szego pozycja ta jest zboczem (i musi zostać przeniesiona z drzewa *A* do drzewa *B*). Jeśli natomiast pozycja ta była zboczem dla p-tego cyprysa i p-ty cyprys był wyższym z sąsiadów, to dla cyprysa $(p+1)$-szego pozycja ta jest szczytem (i musi zostać przeniesiona z drzewa *B* do drzewa *C*). Oczywiście, przenosząc pozycję do nowego drzewa, umieszczamy ją tam z wartością f_i stosowaną w tym drzewie.

Złożoność. Każdą aktualizację stanu drzew: *A*, *B* i *C* potrafimy wykonać w czasie $O(\log n)$. W takim samym czasie znajdujemy również minimum z wartości w tych drzewach. Widzimy więc, że fazę pierwszą dla dowolnego cyprysa potrafimy wykonać w czasie $O(\log n)$, a dla wszystkich — w czasie $O(n \log n)$.

Przypadki szczególne. Przedstawiony opis rozwiązania dla fazy 1 jest dość przejrzysty i elegancki. Jednak zapewne nawet wyjątkowo tolerancyjny Czytelnik już niejednokrotnie odczuł niepokój związany z dość niefrasobliwym traktowaniem przypadków szczególnych — dotychczas właściwie je ignorowaliśmy. W szczególności większość z podanych wzorów nie jest poprawna, jeżeli mamy do czynienia z pozycjami skrajnymi w szpalerze, tzn.

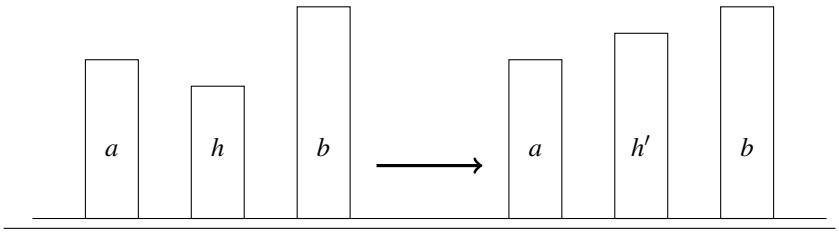
$t,t' \in \{1,n\}$, lub sąsiednimi, tzn. $|t-t'|=1$. Próba zmodyfikowania algorytmu tak, by uwzględnić wszystkie sytuacje, doprowadziłaby do nadmiernego skomplikowania całości i pogrzebania głównych idei w powodzi szczegółów. Proponujemy więc *wyeliminowanie* z powyższego rozwiązania tych przypadków i *rozważenie ich osobno* kosztem niewielkiego, dodatkowego nakładu czasu.

Eliminacja. Przede wszystkim w całym rozwiązaniu nie będziemy brać pod uwagę skrajnych pozycji 1 i n. W ten sposób będziemy mieć gwarancje, że każda rozważana pozycja jest otoczona dwiema sąsiednimi. Kwestię pozycji sąsiednich rozwiążemy, usuwając z puli pozycji (zapisanych w drzewach A, B i C) pozycje sąsiednie z zajmowaną przez aktualnie rozważany cyprys, czyli pozycje $t-1$ oraz $t+1$. Jest to o tyle łatwiejsze, że są to jedyne pozycje, które po zakończeniu rozważania p-tego cyprysa (z miejsca t) przenosimy między drzewami *minTree*. W takim razie nikt nam nie zabrania usunąć ich z drzew *minTree* tuż *przed* rozważaniem p-tego cyprysa i wstawić do odpowiednich drzew *minTree* już *po* jego rozważeniu.

Osobne rozpatrzenie. Ile jest par różnych pozycji takich, z których co najmniej jedna jest skrajna albo są to pozycje sąsiednie? Nie wdając się w dokładne rachunki, możemy śmiało powiedzieć, że jest ich co najwyżej $3n = O(n)$. Ponieważ wpływ zamiany cyprysów z dowolnej takiej pary pozycji na współczynnik nieporządku można wyznaczyć w stałej złożoności czasowej (patrz opis rozwiązania o złożoności czasowej $O(n^2)$), to osobne rozpatrzenie wszystkich przypadków szczególnych może zostać wykonane w dodatkowym czasie liniowym względem n. Taki narzut nie ma więc żadnego wpływu na całkowitą złożoność obliczeniową niniejszego rozwiązania.

Podsumowanie fazy 1. Na zakończenie opisu rozwiązania fazy 1 zauważmy, że zastosowanie drzew *minTree* wydaje się być przesadą — wszak do wykonania wszystkich potrzebnych operacji wystarczyłaby zwykła kolejka priorytetowa! Co więcej, nie widać jeszcze żadnego powodu, by jako klucze elementów odpowiadających pozycjom wybierać wartości H_t, a nie na przykład t. Chcielibyśmy zapewnić Czytelnika, że to wszystko jest celowe i ma ułatwić integrowanie realizacji faz w dalszej części, o czym można się przekonać, kontynuując lekturę.

Faza 2



Rys. 3: Efekt wstawienia cyprysa z miejsca t' w miejscu t.

Przypomnijmy, że nadal rozważamy kwestię znalezienia optymalnego miejsca dla p-tego (według rangi) cyprysa, rosnącego na miejscu t. W fazie 1 rozważyliśmy już koszt wynikający ze zmian przeprowadzonych na jego nowym miejscu. Teraz uwzględnimy zmiany zachodzące na miejscu t. Współczynnik nieporządku zmniejszamy

więc o $|a-h|+|h-b|$ (co odpowiada wyrwaniu cyprysa z miejsca t), a następnie powiększamy o $|a-h'|+|h'-b|$ (co odpowiada posadzeniu cyprysa z miejsca t' w powstałej wyrwie). Jak poprzednio, zmianę współczynnika nieporządku możemy podzielić na składowe: miejsca (tym razem t) oraz cyprysa (tym razem wziętego z miejsca t'):

1. $a+b-2h'-|a-h|-|h-b| = -2h'+(a+b-|a-h|-|b-h|)$, jeżeli $h' \leqslant a$,
2. $b-a-|a-h|-|h-b| = 0+(b-a-|a-h|-|b-h|)$, jeżeli $a \leqslant h' \leqslant b$,
3. $2h'-a-b-|a-h|-|h-b| = 2h'+(-a-b-|a-h|-|b-h|)$, jeżeli $h' \geqslant b$.

Oznaczmy składową zależną od miejsca, jak poprzednio, $f_1(t)$, $f_2(t)$ lub $f_3(t)$, a składową zależną od cyprysa odpowiednio $g_1(t')$, $g_2(t')$ i $g_3(t')$. Zauważmy także, że tym razem wybór formuły, którą mamy zastosować, zależy od tego, czy dla sadzonego w miejscu t cyprysa miejsce to jest doliną, zboczem czy szczytem. Podział cyprysów ze względu na to kryterium jest bardzo prosty. Oznaczmy przez p_a rangę cyprysa o wysokości a (z miejsca $t-1$ lub $t+1$), a przez p_b rangę cyprysa o wysokości b (z drugiego miejsca sąsiadującego z miejscem t). Miejsce t jest:

- doliną dla cyprysów o rangach z przedziału $[1, p_a]$,
- zboczem dla cyprysów o rangach z przedziału $[p_a, p_b]$ oraz
- szczytem dla cyprysów o rangach z przedziału $[p_b, n]$.

Aby sprawnie zrealizować opisane wyżej wyznaczanie zmiany współczynnika nieporządku, tym razem, podobnie jak w fazie 1, także zastosujemy trzy *minTree*, które dla odmiany oznaczymy T_1, T_2 oraz T_3. W drzewie T_i będziemy przechowywać *wszystkie* pozycje t' w postaci par $(H_{t'}, g_i(t'))$, gdzie, przypomnijmy, $H_{t'}$ jest rangą cyprysa z miejsca t'. W ten sposób wyboru optymalnego cyprysa zastępującego p-ty cyprys na miejscu t możemy dokonać, wyznaczając minimalną spośród wartości: $f_1(a,b,h)+min\_val_{T1}(1,p_a)$, $f_2(a,b,h)+min\_val_{T2}(p_a,p_b)$ oraz $f_3(a,b,h)+min\_val_{T3}(p_b,n)$. Co równie istotne, drzewa T_1, T_2 i T_3 są takie same dla kolejnych rozważanych cyprysów i nie trzeba ich przebudowywać przy przejściu od p-tego cyprysa do $(p+1)$-szego.

Przypadki szczególne. Jak poprzednio, przy realizacji fazy 1, chcielibyśmy usunąć przypadki szczególne z zasadniczego algorytmu i rozważyć je oddzielnie. Bez problemu możemy pominąć rozważanie cyprysów pochodzących ze skrajnych pozycji $t' \in \{1,n\}$. Większym problemem jest uniknięcie rozważania cyprysów z pozycji sąsiednich względem t, czyli $t' \in \{t-1, t+1\}$. Aby to zrealizować, przed przystąpieniem do rozważania cyprysa z pozycji t usuniemy ze struktur T_1, T_2 i T_3 cyprysy zajmujące newralgiczne pozycje. Wstawimy je z powrotem po wykonaniu odpowiednich zapytań *min_val*. Jak widać, nie skomplikuje to nam zbytnio całości obliczeń. Oczywiście, po wyeliminowaniu przypadków szczególnych, rozważymy je odrębnie i uwzględnimy otrzymane wyniki przy poszukiwaniu optymalnego cyprysa, zastępującego cyprys z miejsca t.

Złożoność. Wstępna budowa wszystkich drzew *minTree* wymaga czasu $O(n)$. Znalezienie wartości *min_val* dla każdego rozważanego cyprysa możemy wykonać w czasie $O(\log n)$. Eliminacja przypadków szczególnych w jednej rundzie wymaga czasu $O(\log n)$, a ich

uwzględnienie — łącznego czasu $O(n)$. Całkowity czas realizacji fazy 2 wynosi więc $O(n \log n)$.

Podsumowanie fazy 2. Podobnie jak poprzednio, tym razem także nie wykorzystaliśmy wszystkich właściwości drzew przedziałowych, a potrzeba użycia drzewa T_2 — zdegenerowanej struktury, w które przechowujemy zera dla wszystkich wartości — jest całkiem wątpliwa. Pozostaje mieć nadzieję, że wszystko znajdzie swoje zastosowanie w chwili połączenia fazy pierwszej i drugiej, bo nietrudno zauważyć, że fazy te, traktowane oddzielnie, nie dają nam sensownego rozwiązania problemu.

Połączenie faz 1 i 2

Przyszła wreszcie pora na połączenie rozwiązań dla faz 1 i 2. Aby je przeprowadzić, nakreślmy plan działania całego algorytmu.

Przeglądamy cyprysy w kolejności rosnących rang $p = 1, 2, \ldots, n$ (p-ty cyprys rośnie na miejscu $t = D_p$).

- Dla p-tego cyprysa rozważamy wszystkie możliwości zamiany go z innym cyprysem, rosnącym na miejscu t'. Wpływ takiej zamiany jest zależny od:
  - sytuacji, w jakiej znajdzie się cyprys p-ty w miejscu t' — czy będzie to dla niego dolina, zbocze czy szczyt; oraz
  - sytuacji, w jakiej znajdzie się cyprys z miejsca t' przesadzony w miejsce t — czy będzie to dla niego dolina, zbocze czy szczyt.

  Kombinacja powyższych warunków tworzy nam dziewięć sytuacji — w każdej z nich obowiązuje inna formuła wyznaczania zmiany współczynnika nieporządku.

  Dzielimy więc wszystkie potencjalne pozycje t' dla p-tego cyprysa na dziewięć kategorii, w każdej z nich utrzymujemy właściwą wartość zmiany współczynnika nieporządku.
- Poszukując optymalnej pozycji do zamiany dla p-tego cyprysa, wybieramy optymalną pozycję z dziewięciu optymalnych pozycji znalezionych odrębnie w każdej kategorii.
- Dodatkowo dla cyprysa p-tego rozważamy przypadki szczególne i sprawdzamy, czy nie są lepsze od znalezionych dotychczas.

Najbardziej skomplikowany moment w realizacji przedstawionego planu to podział potencjalnych pozycji t' na dziewięć kategorii (z właściwą wartością funkcji zmiany współczynnika nieporządku) i utrzymywanie poprawności tego podziału przy rozważaniu kolejnych cyprysów według rangi p. Aby go zrealizować, posłużymy się *dziewięcioma* (!) drzewami *minTree*: X_i, gdzie $X \in \{A, B, C\}$ oraz $i \in \{1, 2, 3\}$. Drzewo X_i będzie zawierało dane dotyczące pozycji t' w postaci pary (r, w), gdzie:

- r jest rangą drzewa rosnącego na pozycji t',
- w jest sumą wartości zapisanych dla pozycji t' w drzewach X oraz T_i, czyli wartością zmiany współczynnika nieporządku odpowiednią dla sytuacji, gdy pozycja t' jest dla

p-tego cyprysa odpowiednio doliną ($X = A$), zboczem ($X = B$) lub szczytem ($X = C$), oraz gdy cyprys sadzony w miejsce cyprysa p-tego trafia do doliny ($i = 1$), na zbocze ($i = 2$) lub na szczyt ($i = 3$).

Aby sobie to lepiej wyobrazić, przyjrzyjmy się dokładniej kilku przykładowym drzewom *minTree*. W chwili rozważania cyprysa p-tego z pozycji $t = D_p$ o wysokości $h = h_t$:

- A_1 zawiera pozycje t', dla których zachodzi $h \leqslant \min(h_{t'-1}, h_{t'+1})$, a wartość przechowywana w nim jest równa sumie wartości z *minTree* A oraz T_1, czyli $f_1(t') + g_1(t')$; z tego drzewa wybieramy pozycje z przedziału $[1, p_a]$, gdzie $p_a = \min(H_{t-1}, H_{t+1})$, by ostatecznie otrzymać wszystkie pozycje, które są *dolinami* dla cyprysa z pozycji t, zajętymi przez cyprysy, które przesadzone w miejsce t także trafią *w dolinę*;

- A_2 zawiera dokładnie te same pozycje t', co A_1, czyli doliny dla p-tego cyprysa; wartości przechowywane dla pozycji są postaci $f_1(t') + g_2(t')$, co odpowiada sumie wartości z A oraz T_2; rozważając w tym drzewie tylko przedział $[p_a, p_b]$, gdzie $p_a = \min(H_{t-1}, H_{t+1})$ i $p_b = \max(H_{t-1}, H_{t+1})$, mamy wszystkie pozycje, które są *dolinami* dla cyprysa z pozycji t, zajętymi przez cyprysy, które przesadzone w miejsce t trafią *na zbocze*.

- B_3 zawiera pozycje, które są *zboczami* dla p-tego cyprysa, czyli $\min(h_{t'-1}, h_{t'+1}) \leqslant h \leqslant \max(h_{t'-1}, h_{t'+1})$, a wartości w nich zapisane są równe $f_2(t') + g_3(t')$; rozważając w tym drzewie tylko przedział $[p_b, n]$, gdzie $p_b = \max(H_{t-1}, H_{t+1})$, mamy wszystkie pozycje, które są *zboczami* dla cyprysa z pozycji t, zajętymi przez cyprysy, które przesadzone w miejsce t trafią *na szczyt*.

Posiadając opisane wyżej drzewa (nad sposobem ich konstrukcji i aktualizacji zastanowimy się za chwilę), możemy łatwo wyznaczyć optymalną zmianę współczynnika nieporządku wynikającą z zamiany p-tego cyprysa z dowolnym innym. Musimy tylko wyznaczyć pozycję t', dla której wypada minimum z wartości:

- $f_1(t) - 2h + min\_val_{A1}(1, p_a)$,
- $f_2(t) - 2h + min\_val_{A2}(p_a, p_b)$,
- $f_3(t) - 2h + min\_val_{A3}(p_b, n)$,
- $f_1(t) + 0 + min\_val_{B1}(1, p_a)$,
- $f_2(t) + 0 + min\_val_{B2}, (p_a, p_b)$,
- $f_3(t) + 0 + min\_val_{B3}(p_b, n)$,
- $f_1(t) + 2h + min\_val_{C1}(1, p_a)$,
- $f_2(t) + 2h + min\_val_{C2}(p_a, p_b)$,
- $f_3(t) + 2h + min\_val_{C3}(p_b, n)$.

Mimo stosunkowo dużej liczby otrzymanych przypadków (a zatem także niemałej stałej multiplikatywnej) mamy więc logarytmiczną względem n złożoność czasową pojedynczego zapytania.

Powróćmy do problemu stworzenia i aktualizacji wszystkich *minTree*. Na samym początku wszystkie pozycje t' przechowujemy w trzech kopiach, po jednej w A_1, A_2 oraz A_3. Jest to poprawne dzięki obranej przez nas kolejności rozpatrywania cyprysów według rang. Następnie dla kolejnych cyprysów o rangach $p = 1, 2, \ldots, n$ (zajmujących odpowiednio pozycję $t = D_p$):

- usuwamy pozycje $t-1$ oraz $t+1$ z tych drzew *minTree*, w których się one aktualnie znajdują,
- wyznaczamy dziewięć wartości *min_val*, z których wyznaczamy optymalną,
- wstawiamy pozycje $t-1$ oraz $t+1$ do odpowiednich drzew *minTree* (zgodnie z zasadami opisanymi przy fazie 1).

Realizacja jednego kroku aktualizacji wymaga stałej liczby operacji na drzewach *minTree*, co pokazuje, że łączny koszt czasowy utrzymywania wszystkich *minTree* w trakcie działania algorytmu to $O(n \log n)$.

Zauważmy wreszcie, że na tyle dokładnie zajęliśmy się przypadkami szczególnymi w poszczególnych fazach, że tym razem nie będą one stanowić dla nas większego problemu. Przede wszystkim możemy z wszystkich rozważań usunąć skrajne pozycje: pierwszą oraz n-tą. Problem zamian na sąsiednich pozycjach rozwiązujemy, usuwając z *minTree* pozycje sąsiadujące z rozważaną przed wyznaczeniem minimów i umieszczając je z powrotem tuż przed zakończeniem danego kroku. Wreszcie na samym końcu możemy przypadki brzegowe rozważyć osobno, co wobec małej ich liczby nie powoduje wzrostu złożoności czasowej całego algorytmu.

Podsumowanie i analiza złożoności

Ponieważ opis całego algorytmu był dosyć długi i zawiły, na koniec przypominamy poszczególne jego etapy, przy okazji analizując ich złożoności czasowe:

1. Na początku sortujemy wszystkie cyprysy według wysokości i na podstawie posortowanego ciągu wyznaczamy ciągi D_i oraz H_i (dokładniejszy opis tej procedury pozostawiamy Czytelnikowi jako ćwiczenie). Złożoność czasowa tego kroku to — w przypadku zastosowania efektywnego algorytmu sortowania, na przykład przez scalanie — $O(n \log n)$.
2. Budujemy dziewięć pustych drzew *minTree*. Operacja ta zajmuje nam czas $O(n)$.
3. Wstawiamy po jednej kopii każdej pozycji $t' \in [2, n-1]$ do każdego z drzew A_1, A_2 oraz A_3. Krok ten ma łączną złożoność czasową $O(n \log n)$, ponieważ wstawienie jednego elementu wymaga czasu $O(\log n)$. Realizując go procedurą analogiczną do inicjalizacji drzewa przedziałowego, możemy tę złożoność zredukować do $O(n)$, choć niestety nie zmniejszy to złożoności całego algorytmu, gdyż dominujący jest kolejny krok.
4. Rozważamy kolejno cyprysy według rang (pomijając cyprysy zajmujące pierwszą i ostatnią pozycję w szpalerze). Dla każdego z nich:

(a) dwie sąsiadujące z nim pozycje $t-1$ oraz $t+1$ zostają usunięte ze wszystkich *minTree* (czas $O(\log n)$),

(b) wyliczamy minimalną wartość z dziewięciu wartości zwróconych przez funkcję *min_val* (czas $O(\log n)$),

(c) dwie sąsiadujące z cyprysem pozycje ($t-1$ i $t+1$) wstawiamy do odpowiednich *minTree* (czas, jak poprzednio, to $O(\log n)$).

Widzimy, że łączny koszt czasowy tych wszystkich operacji jest rzędu $O(n \log n)$.

5. Rozważenie wszystkich przypadków szczególnych ma łącznie dodatkową złożoność czasową $O(n)$.

Otrzymujemy więc algorytm o całkowitej złożoności czasowej $O(n \log n)$. Jego złożoność pamięciowa jest liniowa względem n, ponieważ wielkość każdego drzewa *minTree* można oszacować przez $O(n)$, a są to największe struktury używane w rozwiązaniu.

Implementacje rozwiązania wzorcowego można znaleźć w plikach `drz.cpp` i `drz0.pas`. Gorąco zachęcamy do przyjrzenia się im, gdyż może to istotnie ułatwić dogłębne zrozumienie istoty całości rozwiązania.

Usprawnienia rozwiązania $O(n^2)$

Na koniec powrócimy do rozwiązania o złożoności $O(n^2)$ i omówimy sposoby jego usprawnienia. Chociaż żaden z nich nie powodował istotnego spadku złożoności rozwiązania (pozostawała rzędu n^2), to jednak pozwalały one zdobyć do 70 punktów na 100. Wszystkie usprawnienia polegały na eliminowaniu sprawdzeń pewnych par przesadzanych drzew.

Pierwszy pomysł. Zauważmy, że zamiana miejscami dwóch drzew rosnących na miejscach, które nazwaliśmy wcześniej zboczami (drzewo rosnące na zboczu sąsiaduje z drzewem od siebie niższym i wyższym), nie powoduje zmniejszenia współczynnika nieporządku. Trochę trudniejszy do pokazania jest fakt, że zamiana miejscami dwóch drzew rosnących w dolinach (pomiędzy dwoma wyższymi drzewami) albo dwóch drzew rosnących na szczytach (pomiędzy dwoma mniejszymi drzewami) nie może poprawić tego współczynnika. Pominięcie takich przypadków może istotnie zredukować liczbę sprawdzeń wykonywanych w algorytmie, zwłaszcza dla testów specyficznej postaci, w których dominuje jeden typ pozycji (doliny albo zbocza, albo szczyty).

Drugi pomysł. Rozwiązanie kwadratowe możemy także usprawnić, dzieląc proces zamiany miejscami drzew z pozycji t i t' na zmiany zachodzące na pozycji t oraz zmiany zachodzące na pozycji t', podobnie jak w rozwiązaniu wzorcowym. Oznaczmy wysokość drzewa z pozycji t przez h, a drzewa z pozycji t' — przez h'. Dodatkowo przez a i b oznaczmy wysokości drzew zajmujących pozycje sąsiadujące z t (jak w rozwiązaniu wzorcowym). Zmiany zachodzące w miejscu t powodują zmniejszenie współczynnika nieporządku o $|a-h|+|h-b|$ i zwiększenie go o $|a-h'|+|h'-b|$. Z nierówności $|a-h'|+|h'-b| \geqslant |a-b|$, którą łatwo pokazać, dostajemy dolne oszacowanie

na zmianę współczynnika nieporządku, którą można uzyskać, wstawiając jakiekolwiek drzewo w miejsce t:

$$o_t = |a-b| - |a-h| - |h-b|.$$

Jak można to oszacowanie wykorzystać? Jeżeli znaleźliśmy już zamianę drzewa z pozycji t z innym, która zmniejsza współczynnik nieporządku o x, a dla wszystkich nierozważonych jeszcze pozycji t' zachodzi nierówność $o_t + o_{t'} \geqslant x$, to możemy zaprzestać dalszych poszukiwań kandydata do przesadzenia na pozycję t. Zwróćmy uwagę, że powyższe oszacowania są poprawne jedynie dla zamian drzew rosnących na pozycjach niesąsiednich i nieskrajnych, czyli przypadki szczególne wymagają odrębnego rozpatrzenia.

Zastosowanie powyższego pomysłu powodowało w średnim przypadku kilkukrotne przyspieszenie rozwiązania.

Testy

Rozwiązania zawodników były sprawdzane na zestawie 10 testów. Testy poprawnościowe miały specyficzne struktury, natomiast w pozostałych testach występowała mniej więcej zrównoważona liczba pozycji typu dolina, zbocze i szczyt. Efekt zrównoważenia uzyskano dzięki zamieszczeniu w teście naprzemiennych krótkich ciągów drzew o rosnących wysokościach, poprzeplatanych również krótkimi ciągami drzew o malejących wysokościach. Wysokości drzew w tych ciągach były generowane losowo.

Nazwa	n	Opis
drz1.in	10	test poprawnościowy
drz2.in	20	test poprawnościowy
drz3.in	100	test poprawnościowy
drz4.in	1 000	test eliminujący rozwiązania o złożonościach $O(n^3)$
drz5.in	9 600	test eliminujący rozwiązania $O(n^3)$ i gorsze rozwiązania $O(n^2)$
drz6.in	20 000	test eliminujący większość rozwiązań $O(n^2)$
drz7.in	35 000	duży test wydajnościowy
drz8.in	48 000	duży test wydajnościowy
drz9.in	50 000	duży test wydajnościowy
drz10.in	50 000	duży test wydajnościowy