OI, Etap I

Tetris 3D

Autorzy gry Tetris postanowili stworzyć nową, trójwymiarową wersję gry, w której prostopadłościenne klocki będą opadać na prostokątną platformę. Podobnie jak w przypadku zwykłej, dwuwymiarowej wersji gry, klocki mają opadać osobno, w pewnej ustalonej kolejności. Dany klocek opada, dopóki nie natrafi na przeszkodę w postaci platformy albo innego, już stojącego klocka, a wtedy się zatrzymuje (w pozycji, w jakiej opadał) i pozostaje na swoim miejscu do końca gry.

Autorzy nowej gry postanowili jednak zmienić charakter gry, ze zręcznościowej na grę logiczną. Znając kolejność opadania klocków na płaszczyznę i tory ich lotu, gracz będzie musiał podać wysokość najwyżej położonego punktu w układzie powstałym po opadnięciu wszystkich klocków. Wszystkie klocki opadają pionowo w dół i nie obracają się w trakcie opadania. Dla ułatwienia wprowadźmy na platformie układ współrzędnych kartezjańskich o początku w jednym z jej narożników i osiach równoległych do jej boków.

Napisz program, który zautomatyzuje sprawdzanie, czy gracz udzielił poprawnej odpowiedzi.

Zadanie

Napisz program, który:

- *wczyta ze standardowego wejścia opisy kolejno opadających klocków,*
- *wyznaczy najwyższy punkt układu klocków po zakończeniu ich opadania,*
- *wypisze wynik na standardowe wyjście.*

Wejście

W pierwszym wierszu wejścia znajdują się trzy liczby całkowite D, S i N ($1 \leqslant N \leqslant 20\,000$, $1 \leqslant D,S \leqslant 1\,000$), oddzielone pojedynczymi odstępami i oznaczające odpowiednio: długość i szerokość platformy oraz liczbę klocków, które na nią opadną. W następnych N wierszach występują opisy kolejnych klocków, po jednym w wierszu.

Każdy opis klocka składa się z pięciu liczb całkowitych: d, s, w, x oraz y ($1 \leqslant d$, $0 \leqslant x$, $d+x \leqslant D$, $1 \leqslant s$, $0 \leqslant y$, $s+y \leqslant S$, $1 \leqslant w \leqslant 100\,000$), reprezentujących klocek o długości d szerokości s i wysokości w. Klocek ten będzie opadał na platformę ścianą o wymiarach $d \times s$, przy czym długość i szerokość klocka będą równoległe odpowiednio do długości i szerokości platformy. Wierzchołki rzutu klocka na platformę będą miały współrzędne: (x,y), $(x+d,y)$, $(x,y+s)$ i $(x+d,y+s)$.

Wyjście

Pierwszy i jedyny wiersz wyjścia powinien zawierać dokładnie jedną liczbę całkowitą, oznaczającą wysokość najwyższego punktu w układzie klocków po zakończeniu ich opadania.

Przykład

Dla danych wejściowych:	*poprawnym wynikiem jest:*

`7 5 4`
`4 3 2 0 0`
`3 3 1 3 0`
`7 1 2 0 3`
`2 3 3 2 2`
	`6`

Rozwiązanie

Tetris 2D

Przy rozwiązywaniu wielu problemów trójwymiarowych dobrym pomysłem jest rozważenie odpowiadających im problemów dwuwymiarowych i następnie uogólnienie rozwiązania na trzy wymiary. Dokładnie tak postąpimy w przypadku niniejszego zadania — na początek rozważymy problem, w którym prostokąty spadają na prostą. Aby przeprowadzić symulację tego procesu, wystarczy dla każdej liczby całkowitej x (będziemy ją utożsamiać z punktem $(x,0)$ na prostej OX) pamiętać maksymalną wysokość (oznaczmy ją przez M_x), do jakiej sięga prostokąt, którego rzut na prostą OX zawiera punkt x. Tak więc dla prostokąta spadającego na odcinek $(x,x+d)$, symulację wykonujemy w dwóch krokach:

- w pierwszej kolejności obliczamy, na jakiej wysokości prostokąt zatrzyma się, wyznaczając M — maksimum z wartości M_y dla $y \in (x,x+d)$; rozważamy przedział otwarty, ponieważ nasze prostokąty nie mają brzegu,
- następnie wszystkim punktom z z przedziału $[x,x+d]$ przypisujemy wartość $M_z = M + w$ równą właśnie obliczonemu maksimum, powiększonemu o wysokość prostokąta; wartość tę przypisujemy również punktom leżącym na brzegu podstawy prostokąta, gdyż jeżeli inny prostokąt będzie zawierał je we wnętrzu podstawy, to się na nich zatrzyma.

Opisaną wyżej procedurę można wykonać bardzo prosto w czasie $O(DN)$. Uogólnienie tego rozwiązania na przypadek trójwymiarowy ma już złożoność czasową $O(DSN)$, czyli zdecydowanie za dużą jak na ograniczenia z zadania.

Spróbujmy więc trochę udoskonalić przedstawione rozwiązanie dwuwymiarowe. Aby zwiększyć jego efektywność, zastosujemy popularną strukturę danych — drzewo przedziałowe.

Drzewa przedziałowe

Struktura i własności

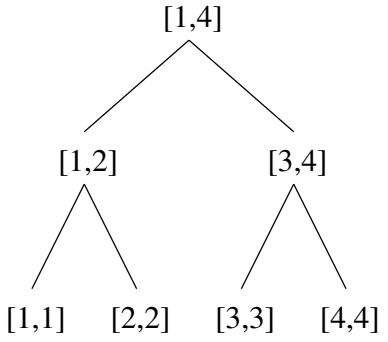
Rozważmy przedział $[1,2^k]$, gdzie k jest liczbą całkowitą nieujemną. *Drzewem przedziałowym* nazwiemy pełne drzewo binarne (czyli drzewo, w którym każdy wierzchołek, oprócz liścia, ma dokładnie dwoje dzieci), w którym:

- korzeń reprezentuje cały przedział $[1,2^k]$, zawierający 2^k elementów,

- dzieci korzenia reprezentują przedziały $[1,2^{k-1}]$ i $[2^{k-1}+1,2^k]$,
- dzieciom wierzchołka reprezentującego przedział $[a,b]$ są przypisane połówki $[a,\frac{a+b}{2}]$ i $[\frac{a+b}{2}+1,b]$ tego przedziału,
- liściom są przypisane przedziały jednoelementowe (długości zero).

Będziemy rozróżniać długość przedziału ($b-a$ dla $[a,b]$) od liczby jego elementów, czyli liczby liczb całkowitych w nim zawartych ($b-a+1$ dla przedziału $[a,b]$, gdy a i b są liczbami całkowitymi).

Jeżeli z wierzchołka v_1 można dojść do wierzchołka v_2 ścieżką prowadzącą tylko w górę drzewa, to powiemy, że v_2 jest *przodkiem* v_1 lub że v_1 jest *potomkiem* v_2. W szczególności każdy wierzchołek drzewa jest zarówno swoim przodkiem, jak i potomkiem. Ponadto będziemy mówili, że korzeń znajduje się na *głębokości* zero, jego dzieci na głębokości jeden itd. *Głębokością drzewa* nazwiemy liczbę wierzchołków na ścieżce od korzenia do liścia o maksymalnej głębokości.



Rys. 1: Przykładowe drzewo przedziałowe o głębokości 3

Przedziały przypisane wierzchołkom drzewa przedziałowego nazywamy *przedziałami bazowymi*. Zanim zastosujemy drzewa przedziałowe do rozwiązania zadania, pokażemy kilka ich własności.

Obserwacja 1 *Głębokość drzewa przedziałowego dla przedziału* $[1,2^k]$ *jest równa* $k+1$*, a całkowita liczba wierzchołków drzewa jest równa* $2\cdot 2^k-1$.

Rozmiar drzewa przedziałowego jest więc proporcjonalny do rozmiaru przedziału, a wysokość — do logarytmu z rozmiaru przedziału.

Dowód Korzeń reprezentuje przedział o 2^k elementach. Jego dzieciom przypisane są połówki tego przedziału, zawierające po 2^{k-1} elementów. Dalej, na kolejnym poziomie mamy łącznie 4 wierzchołki, każdy związany z przedziałem zawierającym 2^{k-2} elementów itd. Liczba elementów w przedziale maleje przy przejściu o jeden poziom w głąb dwukrotnie, a zatem na k-tym poziomie otrzymujemy przedziały jednoelementowe — $2^{k-k}=1$. Stąd widzimy, że drzewo przedziałowe ma głębokość $k+1$.

Oznaczmy przez S łączną liczbę wierzchołków w drzewie. Ponieważ na kolejnych poziomach drzewa liczba wierzchołków podwaja się, a startujemy od jednego korzenia i kończymy na 2^k liściach, więc S wyraża się jako suma ciągu geometrycznego i jest równe:

$$S = 1+2+2^2+\ldots+2^k$$

$$\begin{aligned} &= (1-2^{k+1})/(1-2) \\ &= 2\cdot 2^k - 1 \end{aligned}$$

Jeżeli oznaczymy przez $D = 2^k$ liczbę elementów w przedziale $[1,2^k]$ zawiązanym z korzeniem drzewa, to rozmiar drzewa jest rzędu $O(2\cdot D-1) = O(D)$, a jego wysokość — $O(k+1) = O(\log D)$. ■

Drzewa przedziałowe możemy stosować do przechowywania informacji o odcinkach lub punktach zawartych w przedziale $[1,2^k]$. W typowym schemacie działania algorytmu wykorzystującego drzewo przedziałowe z każdym punktem przedziału $[1,2^k]$ związana jest pewna wartość, a wykonywane są operacje dwojakiego typu:

- zmiana wartości wszystkich punktów zawartych w danym przedziale,
- zapytanie o wartości punktów zawartych w danym przedziale.

Liście służą wówczas do zapisu informacji o punktach, wierzchołki na poziomie wyższym — do zapisu informacji o przedziałach bazowych długości 1, na kolejnym — o przedziałach bazowych długości 2 itd. Aby zapisać w drzewie informację o przedziale innym niż bazowy, musimy najpierw rozłożyć go na sumę parami rozłącznych przedziałów bazowych. Warto zatem zauważyć, iż:

Obserwacja 2 *Każdy przedział zawarty w* $[1,2^k]$ *można rozłożyć na* $O(\log D)$ *rozłącznych przedziałów bazowych.*

Dowód Aby udowodnić powyższą obserwację, przedstawimy rekurencyjny algorytm konstrukcji odpowiedniego rozkładu. Załóżmy, że rozkładamy przedział $[a,b]$ i rozpocznijmy proces rozkładu w korzeniu drzewa, związanym z przedziałem $[1,2^k]$. W każdym kroku algorytmu:

- Na początku znajdujemy się w wierzchołku reprezentującym pewien przedział bazowy $[u,v]$.
- Jeżeli przedział $[u,v]$ jest w całości zawarty w przedziale $[a,b]$, to dołączamy $[u,v]$ do rozkładu i kończymy bieżące wywołanie rekurencyjne.
- Jeżeli lewy syn bieżącego wierzchołka (odpowiadający przedziałowi $[u,\frac{u+v}{2}]$) jest nierozłączny z $[a,b]$, to schodzimy rekurencyjnie do tego syna.
- Podobnie, jeżeli prawy syn ($[\frac{u+v}{2}+1,v]$) przecina się niepusto z przedziałem $[a,b]$, to schodzimy do niego rekurencyjnie.

Udowodnijmy poprawność powyższego algorytmu. Po pierwsze zauważmy, że dla dowolnego przedziału bazowego $[u,v]$, jeśli P jest wierzchołkiem związanym z tym przedziałem, to:

- wszystkie przedziały związane z potomkami P są zawarte w $[u,v]$ — powstają z jego podziału;
- $[u,v]$ jest zawarty we wszystkich przedziałach związanych z przodkami P — powstał z ich podziału;

- oprócz przodków i potomków P nie ma w drzewie przedziałowym innych wierzchołków, których przedziały mają niepusty przekrój z $[u,v]$.

Aby uzasadnić ostatni podpunkt, rozważmy wierzchołek Q niebędący potomkiem P i znajdujący się na niemniejszej głębokości w drzewie niż P. Niech R będzie przodkiem Q leżącym na tej samej wysokości w drzewie, co wierzchołek P — oczywiście $P \neq R$. Ponieważ łatwo widać, że przedziały związane z wierzchołkami leżącymi na tej samej głębokości są rozłączne, więc przedziały wierzchołków P i R są rozłączne, a tym samym przedziały wierzchołków Q i P także są rozłączne.

W przedstawionym algorytmie nigdy nie weźmiemy do rozkładu pary wierzchołków, z których jeden jest przodkiem drugiego. Stąd każdy element przedziału $[a,b]$ będzie zawierał się w co najwyżej jednym przedziale bazowym. Z drugiej strony, jeżeli rozważamy w algorytmie przedział bazowy $[u,v]$ nierozłączny z $[a,b]$, to żaden punkt $x \in [a,b] \cap [u,v]$ nie zostanie pominięty przez algorytm (jeżeli $[u,v]$ nie jest zawarty w $[a,b]$, to i tak jeden z synów $[u,v]$ będzie zawierał x, a zatem nastąpi wywołanie rekurencyjne dla tego syna). To kończy dowód poprawności algorytmu.

Pokażemy teraz, że powyższy algorytm ma złożoność czasową $O(\log D)$, co implikuje, że rozmiar otrzymanego rozkładu jest także $O(\log D)$. W tym celu zauważmy, że na każdym poziomie głębokości drzewa istnieją co najwyżej dwa wierzchołki, dla których zostanie wykonane wywołanie rekurencyjne procedury dla co najmniej jednego z dzieci. Przypuśćmy, że istnieją trzy takie wierzchołki: P_1, P_2 i P_3 (ich przedziały to odpowiednio $[u_1,v_1]$, $[u_2,v_2]$ i $[u_3,v_3]$). Dla ustalenia uwagi przyjmijmy, że P_1 „leży na lewo" od P_2, a P_2 „leży na lewo" od P_3, to znaczy $v_1 < u_2$ i $v_2 < u_3$. Ponieważ każdy z przedziałów $[u_1,v_1]$, $[u_2,v_2]$ i $[u_3,v_3]$ przecina się z $[a,b]$ i żaden nie jest zawarty w $[a,b]$ (tylko w takim przypadku dla odpowiedniego wierzchołka P_i wykonujemy wywołanie rekurencyjne dla jednego z dzieci), stąd istnieją punkty $x_1, x_3 \in [a,b], x_2 \notin [a,b]$, takie że $x_i \in [u_i,v_i]$ dla $i = 1,2,3$. Z przyjętego założenia o wzajemnym położeniu przedziałów wynika, że $x_1 \leqslant x_2 \leqslant x_3$, co stanowi sprzeczność z tym, że tylko $x_2 \notin [a,b]$.

Z właśnie udowodnionego spostrzeżenia wynika, że na każdym poziomie drzewa odwiedzimy co najwyżej 4 wierzchołki (gdyż zejdziemy rekurencyjnie w dół z co najwyżej dwóch wierzchołków na poziomie o jeden wyższym), a zatem liczba wszystkich odwiedzonych w trakcie działania algorytmu wierzchołków nie przekroczy $4 \cdot (k+1) = O(\log D)$. ■

Odnotujmy jeszcze jedno spostrzeżenie, które przyda się nam później. W powyższym algorytmie odwiedzamy wszystkie takie wierzchołki, które są przodkami jakichkolwiek wierzchołków z rozkładu przedziału $[a,b]$ (wynika to z faktu, że algorytm działa zgodnie ze schematem „od korzenia do liści"), czyli wszystkie takie wierzchołki, które odpowiadają przedziałom zawierającym jakiekolwiek przedziały z rozkładu.

Zastosowanie drzewa przedziałowego

Drzewo przedziałowe wykorzystamy do szybszej symulacji dwuwymiarowej gry Tetris. Będziemy w nim przechowywać informacje o przedziałach odpowiadających rzutom poziomych boków prostokątów na prostą, na którą spadają. Przy czym dla każdego dolnego boku wykorzystamy drzewo do obliczenia wysokości, na jakiej zatrzyma się podczas spadania ten odcinek (i cały prostokąt), po czym umieścimy w drzewie informację

o górnym boku, by odnotować fakt, że odpowiednia przestrzeń została zajęta przez prostokąt. W każdym węźle drzewa będziemy zapisywać dwojakiego rodzaju informacje:

- maksimum z wysokości dotychczas wstawionych przedziałów, których rozkłady zawierają przedział bazowy odpowiadający danemu wierzchołkowi drzewa — oznaczmy tę wartość przez M;
- maksimum z wartości M dla wszystkich potomków danego węzła — tę wartość oznaczymy m.

Opiszemy teraz, w jaki sposób efektywnie obliczać powyższe wartości i jak z nich korzystać w symulacji gry. Początkowo wartości M i m dla każdego wierzchołka drzewa ustawiamy na 0.

Opadanie prostokąta — przypadek bazowy Na początek spróbujemy przeprowadzić symulację spadania prostokąta P o dolnej podstawie odpowiadającej przedziałowi bazowemu $[a,b]$. Prostokąt ten zatrzyma się na górnym boku $[u,v]$ pewnego innego, wcześniej zrzuconego prostokąta — rozważmy rozkład $[u,v]$ na przedziały bazowe $[u_i,v_i]$. Zauważmy, że co najmniej jeden z tych przedziałów przecina się niepusto z $[a,b]$ (gdyż zatrzymał opadający prostokąt o podstawie $[a,b]$), ale także co najwyżej jeden taki przedział ma punkt wspólny z $[a,b]$ (gdyż żadne dwa przedziały spośród $[u_i,v_i]$ nie przecinają się i konstruując rozkład $[u,v]$ nigdy nie rozkładamy przedziału bazowego całkowicie zawartego w $[u,v]$ na mniejsze przedziały bazowe). Oznaczmy szukany jedyny przedział o powyższej własności przez $[u_k,v_k]$. Zachodzi dokładnie jedna z poniższych sytuacji:

1. $[u_k,v_k]$ jest zawarty w $[a,b]$ (węzeł drzewa odpowiadający $[u_k,v_k]$ jest potomkiem węzła odpowiadającego $[a,b]$),
2. $[u_k,v_k]$ zawiera przedział $[a,b]$ (węzły odpowiadające przedziałom mają odwrotne ułożenie niż w poprzednim punkcie: węzeł $[a,b]$ jest potomkiem $[u_k,v_k]$).

Stąd wysokość, na jaką opadnie prostokąt P, możemy wyznaczyć licząc maksimum z wysokości wszystkich przedziałów bazowych, zawartych w $[a,b]$ lub zawierających $[a,b]$. Pierwszą z tych możliwości możemy rozpatrzyć, biorąc po prostu wartość m z wierzchołka odpowiadającego przedziałowi $[a,b]$ (wynika to wprost z definicji m), drugą zaś, licząc maksimum z wartości M na ścieżce od korzenia do węzła związanego z $[a,b]$ — tam są wszystkie przedziały zawierające $[a,b]$.

Opadanie prostokąta — przypadek ogólny Rozważmy teraz przypadek, gdy $[a,b]$ nie musi być przedziałem bazowym. Można zauważyć, że wysokość, na jakiej zatrzyma się prostokąt P, wyraża się jako maksimum z wysokości, na jakie mogłyby opaść prostokąty, których podstawami są przedziały bazowe $[a_i,b_i]$, powstałe z rozkładu $[a,b]$. W takim razie, na podstawie poprzednich rozważań wynik dla $[a,b]$ możemy policzyć jako maksimum z:

1. wartości m odpowiadających wierzchołkom związanym z $[a_i,b_i]$,
2. wartości M z przodków wierzchołków odpowiadających $[a_i,b_i]$.

Przypomnijmy, że w algorytmie rozkładu przedziału na przedziały bazowe znajdujemy wszystkie wierzchołki z pierwszej grupy, a po drodze przechodzimy przez wszystkie wierzchołki z drugiej grupy. Stąd modyfikując nieznacznie algorytm możemy wyliczyć szukane maksimum równe wysokości opadnięcia prostokąta H — złożoność czasowa tej procedury wynosi $O(\log D)$.

Zapisanie prostokąta w drzewie Po opadnięciu górny brzeg prostokąta zatrzyma się zatem na wysokości $H+h$, gdzie h to wysokość samego prostokąta. Powinniśmy więc zaktualizować wartości M i m w węzłach drzewa, by zapisać informację o odcinku $[a,b]$ znajdującym się na wysokości $H+h$. Zgodnie z definicją, musimy poprawić wartości M dla wszystkich przedziałów bazowych z rozkładu przedziału $[a,b]$; wierzchołki im odpowiadające można otrzymać w wyniku wykonania algorytmu rozkładu $[a,b]$. Wskutek modyfikacji tych wartości M, zmianie mogą ulec wartości m pewnych wierzchołków — mogą to być wyłącznie wierzchołki, będące przodkami wierzchołków odpowiadających $[a_i,b_i]$. Widzimy więc, że wszystkie wierzchołki, których wartości m lub M trzeba zmienić, zostaną skonstruowane przez algorytm rozkładu przedziału na przedziały bazowe. Stąd wszystkie konieczne aktualizacje wartości M i m można wykonać w złożoności czasowej $O(\log D)$.

Drobny szczegół Pozostaje jeszcze do rozpatrzenia pewien drobny, ale istotny szczegół. Prostokąty, z którymi mamy do czynienia, nie zawierają brzegu, w związku z czym właściwie powinniśmy za podstawę prostokąta uznać przedział otwarty (a,b), zawierający tylko punkty całkowite $a+1, a+2, \ldots, b-1$. Gdybyśmy bowiem za podstawę wzięli przedział domknięty $[a,b]$, to moglibyśmy uznać, iż prostokąt zatrzyma się na innym, z którym jedynie się styka (na przykład prostokąty o podstawach $[a,b]$ i $[b,c]$). Z drugiej strony, zawężenie odcinków do najbliższych liczb całkowitych spowodowałoby inne błędy — wówczas na przykład prostokąt o podstawie $(a,a+1)$ nie blokowałby żadnego innego — do czego oczywiście również nie możemy dopuścić. Rozwiązanie tej kłopotliwej sytuacji zasygnalizowaliśmy już na początku. Otóż szukając przeszkód, które mogą zatrzymać prostokąt o podstawie (a,b) będziemy rozważać przeszkody w przedziale $[a+1,b-1]$. W ten sposób uwzględnimy fakt, że prostokąt nie może zatrzymać się na innym, z którym styka się tylko brzegiem. Z kolei wpisując do drzewa informacje, nad którymi punktami prostokąt zajmuje przestrzeń, będziemy jego podstawę traktować jak odcinek domknięty $[a,b]$, co pozwoli nam zaznaczyć, że prostokąt ten zablokuje wszystkie prostokąty, które zawierają punkty z tego przedziału jako wewnętrzne.

Powyższe modyfikacje nie zmieniają złożoności czasowej procedury symulacji spadania prostokąta, więc ostatecznie złożoność całego algorytmu możemy oszacować przez $O(N \log D)$, co daje istotną poprawę w stosunku do naiwnego algorytmu o złożoności $O(ND)$. W kolejnym rozdziale to właśnie ten algorytm będziemy starali się uogólnić na przypadek trójwymiarowy.

Tetris 3D

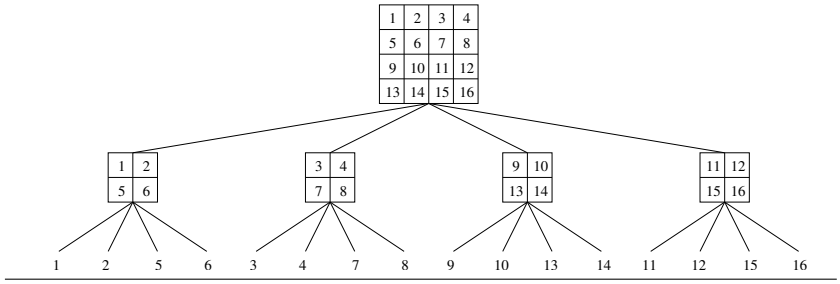
Poszukując rozwiązania dla przypadku trójwymiarowego przeanalizujemy kilka różnych sposobów uogólnienia rozwiązania dwuwymiarowego.

Wiele drzew przedziałowych

Możemy sprowadzić przypadek trójwymiarowy do dwuwymiarowego, dzieląc platformę o podstawie rozmiaru $D \times S$ na S platform o podstawie $D \times 1$ (lub D platform o podstawie $1 \times S$). Dla każdej z tych platform możemy zbudować drzewo przedziałowe i każdy spadający klocek o wymiarach $d \times s \times w$ potraktować jako s identycznych prostokątów o rozmiarach $d \times w$, spadających na odpowiednie proste. Wynikiem dla danego klocka jest maksimum z wartości uzyskanych dla prostokątów. Całkowita złożoność symulacji spadania dla jednego klocka to wówczas $O(S \log D)$ (lub symetrycznie $O(D \log S)$), co daje całkowitą złożoność czasową rozwiązania $O(NS \log D)$. Nie jest to jednak wystarczająco szybko jak na ograniczenia z zadania.

Drzewo czwórkowe

Możemy też próbować zbudować strukturę dwuwymiarową analogiczną do drzewa przedziałowego, w której będziemy przechowywać informacje o prostokątach. Takim rozwiązaniem jest *drzewo czwórkowe*, które pozwala reprezentować prostokąty zawarte w ustalonym kwadracie o wymiarach $2^k \times 2^k$. Korzeń tego drzewa odpowiada całemu kwadratowi. Z kolei dzieci korzenia — ma ich dokładnie 4 — odpowiadają kwadratom, powstałym przez podział dużego kwadratu na cztery równe ćwiartki; każde z dzieci ma również 4 dzieci, odpowiadających jego ćwiartkom itd. Przykład drzewa czwórkowego dla $k = 2$ jest pokazany na rys. 2.



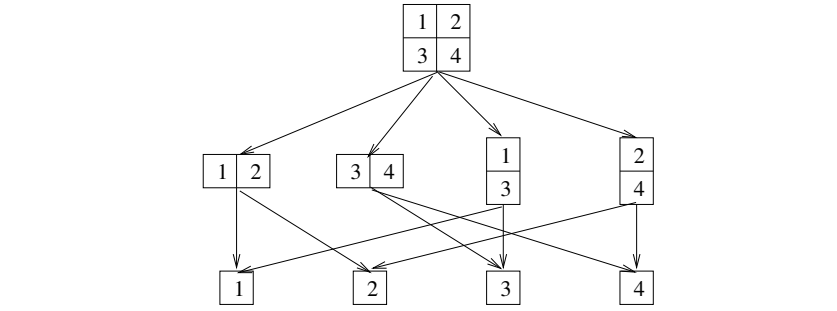
Rys. 2: Przykładowe drzewo czwórkowe

Zanim zaczniemy zapisywać w drzewie czwórkowym informacje potrzebne do symulacji gry, zastanówmy się, na ile kwadratów „bazowych" może zostać rozłożony dany prostokąt. Dla przykładu, prostokąt mający kształt paska o rozmiarze 1×2^k trzeba rozłożyć wyłącznie na kwadraty jednostkowe, do czego potrzebujemy aż 2^k kwadratów! To oznacza, że po adaptacji algorytmu dwuwymiarowego do trzech wymiarów z użyciem drzewa czwórkowego otrzymamy rozwiązanie, które w pesymistycznym przypadku będzie wymagać wykonania co najmniej ND lub NS operacji. Jest to rozwiązanie dalekie od satysfakcjonującego.

Graf prostokątowy

Właściwym uogólnieniem na przypadek trójwymiarowy jest struktura, która już nie jest drzewem, ale acyklicznym grafem skierowanym (określanym często skrótem DAG). Korzeń grafu reprezentuje kwadrat o bokach $2^k \times 2^k$. Wierzchołek ten ma czwórkę dzieci, z których dwójka reprezentuje prostokąty powstałe w wyniku podziału kwadratu symetralną pionową, a dwa pozostałe — symetralną poziomą. W kolejnych wierzchołkach grafu przeprowadzamy

analogiczny podział prostokąta związanego z wierzchołkiem. Prostokąty, które odpowiadają poszczególnym wierzchołkom grafu, nazywamy *prostokątami bazowymi*. Całą strukturę określamy mianem *grafu prostokątowego* — na rysunku 3 jest przedstawiony przykład takiego grafu dla $k = 2$.



Rys. 3: Przykładowy graf prostokątowy

Graf prostokątowy ma następujące własności, analogiczne do odpowiednich własności drzewa przedziałowego.

- Każdy prostokąt będący produktem kartezjańskim dwóch jednowymiarowych przedziałów bazowych, z których jeden jest podprzedziałem przedziału $[1,D]$, a drugi podprzedziałem $[1,S]$, jest pewnym prostokątem bazowym i odwrotnie, każdy prostokąt bazowy to produkt kartezjański dwóch przedziałów bazowych.

- Głębokość grafu prostokątowego jest równa $O(\log D \log S)$, gdzie D i S to wymiary prostokąta bazowego zawartego w korzeniu, który zresztą nie musi być nawet kwadratem. Wynika to stąd, że przechodząc od rodzica do dziecka zmniejszamy dwukrotnie jeden z wymiarów prostokąta związanego z wierzchołkiem, więc po $\log D \log S$ krokach oba wymiary muszą być już równe 1.

- Graf zawiera $O(DS)$ wierzchołków, gdyż tyle jest właśnie prostokątów bazowych utworzonych z par jednowymiarowych przedziałów bazowych.

Podobnie jak w przypadku jednowymiarowym (drzewa przedziałowego i odcinka), także dowolny prostokąt możemy rozłożyć na prostokąty bazowe. Dobry podział otrzymamy, rozbijając boki prostokąta na przedziały (odcinki) bazowe i tworząc prostokąty bazowe — iloczyny kartezjańskie par przedziałów bazowych. Metoda ta daje nam rozbicie prostokąta P o wymiarach $d \times s$ ($d \leqslant D$, $s \leqslant S$) na $O(\log D \log S)$ prostokątów bazowych. Jeżeli będziemy chcieli zastosować graf prostokątowy analogicznie, jak drzewo przedziałowe, to musimy jeszcze zastanowić się, ile jest w grafie prostokątów bazowych, zawierających w sobie jakiekolwiek prostokąty bazowe, powstałe z rozkładu prostokąta P. Ich liczbę możemy oszacować zauważając, że wszystkie muszą być produktami kartezjańskimi przedziałów bazowych, zawierających przedziały bazowe z jednowymiarowych rozkładów boków P. Skądinąd wiemy, że takich prostokątów jest w każdym wymiarze logarytmicznie wiele, stąd liczbę szukanych prostokątów bazowych także możemy oszacować przez $O(\log D \log S)$.

Algorytm wzorcowy

Co prawda powyższe spostrzeżenie ogranicza liczbę prostokątów bazowych w rozkładzie konkretnego prostokąta i nawet w sensie matematycznym pozwala je wskazać, potrzebujemy jednak algorytmu, który pozwoli nam efektywnie zidentyfikować je w grafie bez odwoływania się do drzew przedziałowych dla boków prostokąta. Przedstawimy algorytm rekurencyjny analogiczny do algorytmu rozkładu na przedziały bazowe w drzewie przedziałowym.

Załóżmy, że rozkładamy prostokąt $P = [a,b] \times [c,d]$ i rozpocznijmy proces rozkładu w korzeniu grafu, z którym związany jest prostokąt $[1,2^k] \times [1,2^k]$. W każdym kroku algorytmu:

- Na początku znajdujemy się w wierzchołku q reprezentującym pewien prostokąt bazowy $Q = [u,v] \times [x,y]$.
- Jeżeli w trakcie rozkładania P byliśmy już wcześniej w wierzchołku q, to kończymy to wywołanie rekurencyjne.
- Jeżeli prostokąt Q jest w całości zawarty w P, to dokładamy Q do rozkładu i kończymy to wywołanie rekurencyjne.
- Jeżeli nie zachodzi $[u,v] \subseteq [a,b]$, to dokonujemy podziału prostokąta Q w tym wymiarze i wykonujemy zejście rekurencyjne do tych spośród dzieci q, których prostokąty (odpowiednio $[u, \frac{u+v}{2}] \times [x,y]$ oraz $[\frac{u+v}{2}+1, v] \times [x,y]$) nie są rozłączne z P.
- Analogicznie postępujemy w drugim wymiarze: jeżeli nie zachodzi $[x,y] \subseteq [c,d]$, to wykonujemy zejście rekurencyjne do tych spośród dzieci q, których prostokąty (odpowiednio $[u,v] \times [x, \frac{x+y}{2}]$ oraz $[u,v] \times [\frac{x+y}{2}+1, y]$) nie są rozłączne z P.

Widzimy, że powyższy algorytm został tak zaprojektowany, żeby konstruował dokładnie taki rozkład prostokąta, jak produkt kartezjański rozkładów jednowymiarowych — w każdym kroku rozcinamy aktualny prostokąt bazowy wzdłuż każdej osi, jeżeli odpowiedni bok prostokąta nie jest przedziałem bazowym i zostałby podzielony w algorytmie w przypadku jednowymiarowym. Tak więc wszystkie prostokąty, które odwiedzimy po drodze, są postaci $[u',v'] \times [x',y']$, gdzie $[u',v']$ to jeden z przedziałów odwiedzanych w trakcie jednowymiarowego rozkładu odcinka $[a,b]$, a $[x',y']$ to przedział odwiedzany w trakcie jednowymiarowego rozkładu $[c,d]$. Ponieważ w algorytmie unikamy wielokrotnego zagłębiania rekurencyjnego z tego samego wierzchołka, więc jego złożoność jest proporcjonalna do liczby odwiedzonych różnych wierzchołków, czyli wynosi $O(\log D \log S)$.

Wiedząc, że potrafimy efektywnie rozłożyć prostokąt na „umiarkowaną” liczbę prostokątów bazowych w równie umiarkowanym czasie, zastanówmy się, jakie informacje powinniśmy umieścić w grafie, by móc symulować spadanie klocków. Będzie to trochę więcej danych, niż w przypadku gry dwuwymiarowej. W każdym wierzchołku grafu zapiszemy:

- maksimum z wartości dotychczas wstawionych prostokątów, których rozkłady zawierają prostokąt bazowy odpowiadający danemu wierzchołkowi grafu (oznaczenie: M),

- maksimum z wartości M dla wszystkich potomków danego węzła (oznaczenie: m); obie wartości m i M są analogiczne jak w przypadku drzewa przedziałowego,
- maksimum z wartości M wszystkich potomków danego węzła o takim samym rzucie na pierwszy wymiar (oznaczenie: m_1),
- maksimum z wartości M wszystkich potomków danego węzła o takim samym rzucie na drugi wymiar (oznaczenie: m_2).

W powyższym opisie, rzutem prostokąta $[a,b] \times [c,d]$ na pierwszy wymiar jest przedział $[a,b]$, a na drugi — przedział $[c,d]$. Wartości M, m, m_1 i m_2 na początku algorytmu ustawiamy na 0.

Opiszemy teraz, w jaki sposób efektywnie obliczać powyższe wartości i jak z nich korzystać poszukując wysokości, na jakiej zatrzyma się kolejny spadający klocek.

Przypadek bazowy Podobnie jak w przypadku drzewa przedziałowego, na początek przeprowadzimy symulację spadania klocka K o dolnej podstawie odpowiadającej prostokątowi bazowemu $P = [a,b] \times [c,d]$. Klocek ten zatrzyma się na górnej podstawie $Q = [u,v] \times [x,y]$ pewnego innego klocka; rozważmy rozkład $[u,v] \times [x,y]$ na prostokąty bazowe. Co najmniej jeden z tych prostokątów przecina się niepusto z P; oznaczmy dowolny z prostokątów bazowych o tej własności przez $Q_k = [u_k,v_k] \times [x_k,y_k]$. Zachodzi jedna z poniższych sytuacji:

1. $Q_k \subseteq P$ (węzeł odpowiadający Q_k jest potomkiem węzła odpowiadającego P) — przypadek analogiczny jak w wersji dwuwymiarowej;
2. $P \subseteq Q_k$ (węzeł odpowiadający P jest potomkiem węzła odpowiadającego Q_k) — także przypadek analogiczny jak w wersji dwuwymiarowej;
3. $[a,b] \subset [u_k,v_k]$, ale $[x_k,y_k] \subset [c,d]$ — czyli P zawiera się w pierwszym wymiarze w Q_k,a w drugim wymiarze zachodzi zależność odwrotna — wtedy węzeł odpowiadający Q_k jest potomkiem o tym samym pierwszym wymiarze pewnego węzła $[u_k,v_k] \times [x',y']$, występującego przy rozkładaniu P na prostokąty bazowe,
4. $[c,d] \subset [x_k,y_k]$, ale $[u_k,v_k] \subset [a,b]$ — czyli P zawiera się w drugim wymiarze w Q_k,a w pierwszym wymiarze zachodzi zależność odwrotna — wtedy węzeł odpowiadający Q_k jest potomkiem o tym samym drugim wymiarze pewnego węzła $[u',v'] \times [x_k,y_k]$, występującego przy rozkładaniu P na prostokąty bazowe.

Policzenie wysokości zatrzymania klocka K wymaga uwzględnienia każdej z powyższych sytuacji. Pierwsze dwa przypadki rozpatrujemy podobnie jak w zadaniu dwuwymiarowym: pierwszy odpowiada wzięciu wartości m z wierzchołka odpowiadającego prostokątowi P, drugi zaś maksimum z wartości M wszystkich przodków P w grafie. Wysokość zatrzymania wynikającą z sytuacji opisywanej w trzecim przypadku, znajdujemy wyliczając maksimum z wartości m_1 wszystkich napotkanych w procesie rozkładu P prostokątów bazowych, których pierwszy wymiar zawiera $[a,b]$, a drugi jest zawarty w $[c,d]$. Przypadek czwarty rozpatrujemy analogicznie, używając wartości m_2.

Przypadek ogólny Jesteśmy już gotowi do przeprowadzenia symulacji spadania klocka K, którego podstawa P jest dowolnym prostokątem. Podobnie jak w dwóch wymiarach, rozkładamy P na prostokąty bazowe P_i i symulujemy oddzielnie opadanie każdego z nich. Ostateczną wysokość, na jakiej osiądzie cały klocek, wyznaczamy jako maksimum wyników obliczonych dla poszczególnych prostokątów z rozkładu, czyli maksimum z:

1. wartości m odpowiadających wierzchołkom związanym z P_i,
2. wartości M z przodków wierzchołków odpowiadających P_i,
3. wartości m_1 z tych spośród przodków wierzchołków P_i, które w drugim wymiarze są zawarte w P_i,
4. wartości m_2 z tych spośród przodków wierzchołków P_i, które w pierwszym wymiarze są zawarte w P_i.

Ponieważ w procedurze rozkładu na prostokąty bazowe odwiedzimy wszystkie wierzchołki z powyższych grup 1 i 2, a grupy 3 i 4 są podzbiorami grupy 2, to widzimy, że wysokość H, na jakiej zatrzyma się klocek K, możemy policzyć w łącznej złożoności czasowej $O(\log D \log S)$.

Aktualizacja grafu Górna podstawa klocka K po opadnięciu będzie znajdować się na wysokości $H+h$, gdzie h to wysokość klocka. Wartość tę wykorzystamy do aktualizacji informacji zapisanych w grafie w tych miejscach, gdzie jest to potrzebne. Aktualizacje M i m przeprowadzamy podobnie jak w przypadku dwuwymiarowym: wartość M we wszystkich wierzchołkach odpowiadających P_i, a wartość m — we wszystkich ich przodkach. Wartość m_1 aktualizujemy w tych wierzchołkach spośród napotkanych w algorytmie rozkładu, które w pierwszym wymiarze są zawarte w P. Każdy z tych wierzchołków ma potomka o tym samym pierwszym wymiarze, który jest jednocześnie jednym z wierzchołków odpowiadających pewnemu spośród prostokątów P_i. Co więcej, każdy z przodków jakiegokolwiek z prostokątów P_i o tym samym pierwszym wymiarze zostanie odwiedzony w algorytmie rozkładu, a zatem taki sposób aktualizacji jest wystarczający, aby wszystkie wierzchołki grafu miały poprawne wartości m_1. Analogiczną procedurę przeprowadzamy dla wartości m_2 i drugiego wymiaru. Całą aktualizację wykonujemy w czasie $O(\log D \log S)$, gdyż podobnie jak przy symulacji spadania, wszystkie wierzchołki grafu, w których dokonujemy zmian, napotkamy w procedurze rozkładu P na prostokąty bazowe.

Rozwiązanie z użyciem drzewa przedziałowego ma zatem złożoność czasową $O(N \log D \log S + DS)$, gdzie $O(DS)$ to czas budowy struktury grafu prostokątowego w pamięci. Dokładny opis samej budowy grafu pozostawiamy Czytelnikowi jako ćwiczenie.

Inne rozwiązania

Zadanie Tetris 3D można próbować rozwiązać stosując zupełnie inny pomysł. Dla każdego spadającego klocka można przeglądać wszystkie, które opadły poprzednio, i sprawdzać, czy ich rzuty na platformę przecinają się z aktualnie spadającym. Wysokość, na jakiej zatrzyma się klocek, wyznaczamy wówczas jako maksimum z wysokości takich kolidujących klocków. Tego typu rozwiązanie ma pesymistycznie złożoność $O(N^2)$ i nie bierzemy w nim w ogóle pod uwagę niedużych ograniczeń na długość i szerokość platformy.

Teoretycznie jest to rozwiązanie zbyt wolne, jak na ograniczenia z zadania. Jednakże prawdziwą sztuką było ułożenie na tyle dobrych testów, żeby móc równocześnie wyeliminować bardzo efektywnie zaimplementowane (i czasami niepoprawne) wersje powyższej symulacji i nieefektywne trójwymiarowe uogólnienia drzewa przedziałowego. Oceniając po liczbie punktów uzyskanych przez zawodników, sztuka ta jurorom udała się, a za zadanie Tetris 3D było najtrudniej zdobyć maksymalną liczbę punktów w pierwszym etapie XIII Olimpiady Informatycznej.

Testy

Zadanie testowane było na 14 zestawach danych testowych, które zawierały łącznie aż 55 testów.

Nazwa	D	S	N	Opis
tet1.in	20	20	20	mały test poprawnościowy
tet2a-7a.in	1 000	1 000	20 000	maksymalne testy losowe
tet8a-10a.in	512	512	20 000	duże testy złożone tylko z prostokątów bazowych
tet11a-14a.in	1 000	1 000	20 000	losowe klocki o długości lub szerokości 30-60, a o drugim wymiarze 1 000
tet2b-10b.in	10 000	1 000	20 000	testy maksymalne z 19 000 klockami 1×1 i 1 000 dużych klocków
tet11b-14b.in	1 000	1 000	20 000	testy maksymalne z 18 000 klockami 1×1 i 2 000 dużych klocków
tet2c-10c.in	1 000	1 000	1 000	1 000 losowych klocków o dużej powierzchni
tet11c-14c.in	100	100	100	niewielkie poprawnościowe testy losowe
tet2d-7d.in	1 000	1 000	1 000	testy maksymalne z klockami o jednym boku długości lub szerokości 1, a drugim około 1 000
tet8d-10d.in	100	100	100	niewielkie testy losowe poprawnościowe
tet2e-7e.in	1 000	1 000	20 000	losowe klocki o długości lub szerokości 30-60, a o drugim wymiarze 1 000