Wojciech Nadara — Treść zadania, Opracowanie
Adam Polak — Program

Dostępna pamięć: 128 MB. *OI, etap III, dzień drugi, 3.04.2014*

Lampy słoneczne

Bajtazar ma bardzo duży i piękny ogród, lecz gdy zapadnie zmrok, nie może podziwiać jego uroków. W związku z tym postanowił zaopatrzyć się w lampy, których światło pozwoli na rozkoszowanie się widokiem ogrodu również w nocy.

Zakupione przez Bajtazara lampy nie oświetlają wszystkiego wokół, a jedynie obszar zawarty w pewnym kącie. Dla wszystkich lamp kąt ten jest taki sam, co więcej, lampy muszą być zamontowane tak, aby wszystkie świeciły w tym samym kierunku. Ponadto są to lampy słoneczne. Co prawda w nocy słońca nie ma, ale wystarczy, że na lampę będzie świecić odpowiednia liczba innych lamp, a ona także zacznie świecić. Oczywiście, lampy zapalają się również po podłączeniu ich do prądu.

Bajtazar zamontował lampy w swoim ogrodzie i ustalił, w jakiej kolejności będzie podłączał do nich prąd. Dla uproszczenia ponumerujmy lampy liczbami od 1 do n w tej właśnie kolejności, tzn. Bajtazar w momencie i podłączy prąd do lampy o numerze i. Bajtazar zastanawia się, kiedy zaczną świecić poszczególne lampy. Pomóż mu i napisz program, który odpowie na to pytanie.

Wejście

Pierwszy wiersz standardowego wejścia zawiera jedną liczbę całkowitą n ($1 \leqslant n \leqslant 200\,000$) oznaczającą liczbę lamp, które zakupił Bajtazar. W drugim wierszu wejścia znajdują się cztery liczby całkowite X_1, Y_1, X_2, Y_2 ($-10^9 \leqslant X_i, Y_i \leqslant 10^9$, $(X_i, Y_i) \neq (0, 0)$) pooddzielane pojedynczymi odstępami, wyznaczające obszar oświetlany przez każdą z lamp. Jeśli pewna lampa stoi w punkcie (x, y), to oświetla obszar (wraz z brzegiem), który znajduje się w mniejszym z kątów wyznaczonych przez dwie półproste o początkach w (x, y), z których i-ta (dla $i = 1, 2$) przechodzi także przez punkt $(x + X_i, y + Y_i)$. Kąt ten jest zawsze różny od 180 stopni.

Kolejne n wierszy wejścia opisuje rozstawienie lamp: i-ty z tych wierszy zawiera dwie liczby całkowite x_i, y_i ($-10^9 \leqslant x_i, y_i \leqslant 10^9$) oddzielone pojedynczym odstępem, oznaczające, że lampa o numerze i jest umieszczona w punkcie (x_i, y_i). Możesz założyć, że żadne dwie lampy nie stoją w tym samym punkcie.

Ostatni wiersz wejścia zawiera n liczb całkowitych $k_1, k_2, \ldots, k_n$ ($1 \leqslant k_i \leqslant n$) pooddzielanych pojedynczymi odstępami, oznaczających, że jeżeli lampa o numerze i znajdzie się w obszarze oświetlanym przez co najmniej k_i innych lamp, to ona także zacznie świecić.

W testach wartych łącznie 30% punktów zachodzi dodatkowy warunek $n \leqslant 1000$.

Wyjście

Twój program powinien wypisać na standardowe wyjście jeden wiersz zawierający n liczb całkowitych $t_1, \ldots, t_n$ pooddzielanych pojedynczymi odstępami. Liczba t_i ma oznaczać moment zaświecenia się lampy o numerze i.

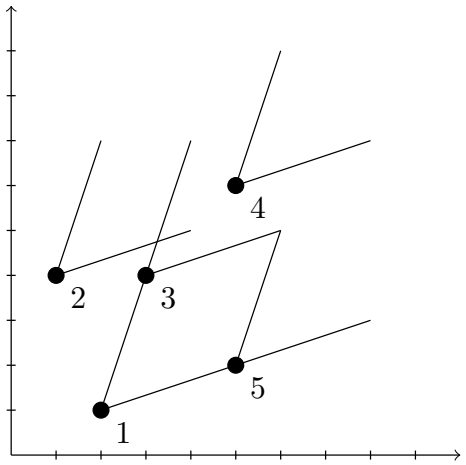
Przykład

Dla danych wejściowych:

```
5
3 1 1 3
2 1
1 4
3 4
5 6
5 2
1 2 1 3 2
```

poprawnym wynikiem jest:

```
1 2 1 2 5
```



Wyjaśnienie do przykładu: *W chwili 1 Bajtazar podłącza prąd do lampy 1, co powoduje zaświecenie się również lampy 3. Po podłączeniu prądu do lampy 2, zaczyna świecić również lampa 4 (oświetlona przez lampy 1, 2 i 3).*

Testy „ocen":

`1ocen`: *$n = 7$, mały test losowy;*

`2ocen`: *$n = 6$, lampy oświetlają kąt zero stopni (Bajtazar zainwestował w lasery);*

`3ocen`: *$n = 160\,000$, lampy ustawione w kratę o wymiarach 400×400, oświetlają kąt 90 stopni o ramionach równoległych do osi układu współrzędnych.*

Rozwiązanie

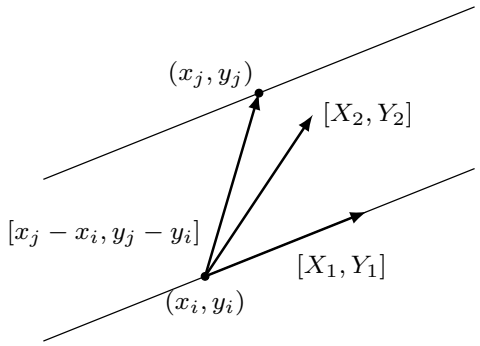
Na początek zauważmy, że przedstawione zadanie pod względem ideologicznym nie różni się wiele od zadania, w którym wszystkie lampy oświetlałyby obszar zadany przez wektory $[1, 0]$ oraz $[0, 1]$ (na razie w naszych rozważaniach załóżmy, że kąt, który oświetlają lampy, jest niezerowy). I rzeczywiście okazuje się, że dla każdego zestawu danych wejściowych można skonstruować zestaw n lamp, z których każda będzie odpowiadała pewnej lampie z oryginalnego zestawu i oświetlać będzie obszar zadany przez wektory $[1, 0]$ oraz $[0, 1]$, a ponadto i-ta lampa z nowego zestawu będzie oświetlać j-tą lampę z nowego zestawu wtedy i tylko wtedy, gdy i-ta lampa z oryginalnego zestawu oświetlała j-tą lampę z oryginalnego zestawu. Skupimy się teraz na wykonaniu tej konstrukcji.

Niektórzy mogą zauważyć, że można zastosować tutaj tzw. przekształcenie afiniczne płaszczyzny, w którym nowe pozycje lamp wyrażają się pewnymi funkcjami liniowymi od ich oryginalnych pozycji. My jednak zastosujemy tutaj inne podejście; będzie ono miało różne zalety, o których przekonamy się później.

Rozważmy rodzinę prostych równoległych do wektora $[X_1, Y_1]$ z zadania. Zauważmy, że każda taka prosta ma dokładnie jeden punkt wspólny z prostą przechodzącą przez punkty $(0, 0)$ oraz (X_2, Y_2) (założyliśmy na razie, że każda lampa

oświetla niezerowy kąt), zatem przecina tę prostą w punkcie (tX_2, tY_2) dla pewnej liczby rzeczywistej t. Każdej takiej prostej przypiszmy odpowiadającą jej liczbę t. Intuicyjnie rzecz ujmując, możemy sobie wyobrazić prostą równoległą do wektora $[X_1, Y_1]$, przemieszczającą się ze stałą prędkością w kierunku zgodnym ze zwrotem wektora $[X_2, Y_2]$, tak że w chwili 0 przechodzi przez punkt $(0, 0)$, a w chwili 1 przez punkt (X_2, Y_2). Wówczas w chwili t (t może być ujemne) będzie się ona pokrywać z tą prostą, której przyporządkowaliśmy liczbę t. Możemy też rozważyć analogiczną rodzinę prostych równoległych do $[X_2, Y_2]$ i przyporządkować im liczby u, względem prostej przechodzącej przez punkty $(0, 0)$ i (X_1, Y_1). Teraz, każda lampa leży na dokładnie jednej prostej należącej do pierwszej rodziny i na dokładnie jednej prostej należącej do drugiej rodziny. Lampie przyporządkujemy parę liczb (t_i, u_i), gdzie t_i to liczba t przyporządkowana prostej należącej do pierwszej rodziny, analogicznie u_i. Łatwo teraz zauważyć, że lampa o numerze i oświetla lampę o numerze j wtedy i tylko wtedy, gdy $t_i \leqslant t_j$ oraz $u_i \leqslant u_j$.

Przekształcenie $(x_i, y_i) \to (t_i, u_i)$ jest zatem przekształceniem, które wspomnieliśmy wcześniej. W szczególności, można je zapisać konkretnymi wzorami. Zauważmy jednak, że nie interesują nas wcale dokładne wartości t_i dla lamp, a jedynie ich porządek. Możemy zatem posortować lampy według tych wartości i każdej lampie przydzielić jako t_i pozycję, na której stoi po takim posortowaniu. Podobnie możemy zrobić z wartościami u_i. Aby posortować lampy po liczbach t_i, wcale nie trzeba tych liczb *explicite* wyznaczać. Wystarczy umieć stwierdzać, dla danych indeksów i, j, która z liczb t_i oraz t_j jest większa. W tym celu sprawdzamy, czy wektor $[x_j - x_i, y_j - y_i]$ leży po tej samej stronie wektora $[X_1, Y_1]$, co wektor $[X_2, Y_2]$ – jeśli po tej samej stronie, to $t_i < t_j$, a w przeciwnym razie $t_i > t_j$ (rys. 1). Do tego celu służy nam iloczyn wektorowy: znak iloczynu wektorowego $[a, b] \times [c, d] = ad - bc$ jest dodatni, jeśli wektor $[c, d]$ leży na lewo od wektora $[a, b]$, a ujemny, jeśli leży on na prawo[1].



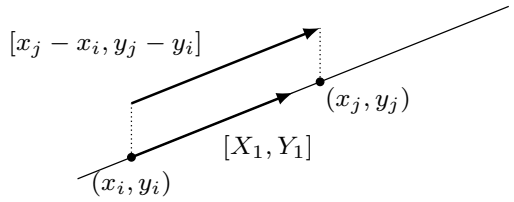
Rys. 1: Mamy $t_i < t_j$, gdyż oba wektory $[x_j - x_i, y_j - y_i]$, $[X_2, Y_2]$ leżą po tej samej stronie wektora $[X_1, Y_1]$ (w tym przypadku oba na lewo). Innymi słowy, prosta równoległa do wektora $[X_1, Y_1]$, przemieszczająca się zgodnie ze zwrotem wektora $[X_2, Y_2]$, odwiedzi najpierw lampę (x_i, y_i), a potem lampę (x_j, y_j).

W ten sposób każdej lampie możemy przyporządkować parę (a_i, b_i) liczb całkowitych z przedziału $[1, n]$, taką że a_i jest pozycją liczby t_i w posortowanym ciągu t, a b_i jest pozycją u_i w posortowanym ciągu u. Każda z liczb z przedziału $[1, n]$ zostanie

[1] Więcej o iloczynie wektorowym można przeczytać np. w książce [25].

na każdej współrzędnej wykorzystana dokładnie raz oraz to przyporządkowanie ma tę własność, że lampa o numerze i oświetla lampę o numerze j wtedy i tylko wtedy, gdy $a_i \leqslant a_j$ oraz $b_i \leqslant b_j$.

Do tej pory jednak przemilczeliśmy pewną kwestię: mianowicie co powinniśmy zrobić, jeżeli dwóm lampom odpowiadałaby ta sama liczba t? Jeżeli chcemy otrzymać przyporządkowanie liczb takie, jak przed chwilą, to nie możemy ustalić między nimi dowolnego porządku. Dla przykładu, gdybyśmy mieli do czynienia z sytuacją, w której $[X_1, Y_1] = [1, 0]$ i $[X_2, Y_2] = [0, 1]$, oraz dwiema lampami o współrzędnych odpowiednio $(0, 0)$ i $(1, 0)$, to ich liczby t wynosiłyby 0, jednak pierwsza lampa oświetla drugą, zatem musielibyśmy mieć $a_1 = 1$ oraz $a_2 = 2$, a nie na odwrót. W przypadku równości liczb t (które wykrywamy na podstawie tego, że opisany wcześniej iloczyn wektorowy się zeruje) musimy ten porządek ustalić zgodnie ze zwrotem wektora $[X_1, Y_1]$. Do tego celu idealnie nadaje się iloczyn skalarny (rys. 2).



Rys. 2: Mamy $t_i = t_j$. Iloczyn skalarny $[X_1, Y_1] \cdot [x_j - x_i, y_j - y_i]$ jest dodatni, więc i-ta lampa otrzyma mniejszy numer niż j-ta.

Podsumujmy: porównując lampy i-tą oraz j-tą, należy sprawdzić, czy iloczyn wektorowy $[X_1, Y_1] \times [x_i - x_j, y_i - y_j]$ jest dodatni, czy ujemny, i na tej podstawie (oraz na podstawie znaku iloczynu $[X_1, Y_1] \times [X_2, Y_2]$) ustalić porządek między liczbami t_i oraz t_j. W przypadku, gdy $t_i = t_j$, czyli gdy dany iloczyn wektorowy wynosi 0, należy porządek między tymi lampami ustalić zgodnie ze znakiem iloczynu skalarnego tych dwóch wektorów (jeżeli iloczyn wektorowy dwóch niezerowych wektorów jest równy 0, to ich iloczyn skalarny nie może wynosić 0).

Otrzymaliśmy zatem ostateczną wersję wstępnego przetwarzania współrzędnych. Lampy należy posortować za pomocą opisanej funkcji porównującej, w której używamy wektora $[X_1, Y_1]$, przypisać im ich pozycje w tym porządku, a potem zrobić to samo dla analogicznej funkcji porównującej, w której zamiast wektora $[X_1, Y_1]$ będziemy używać wektora $[X_2, Y_2]$. Cała konstrukcja działa w czasie $O(n \log n)$.

Jakie są zalety tego rozwiązania? Po pierwsze, operuje ono jedynie na liczbach całkowitych. Po drugie, wynikowe nowe „współrzędne” są od razu przenumerowane, tzn. są to liczby całkowite z przedziału $[1, n]$, co będzie pomocne, gdy będziemy nimi potem indeksować pewne struktury danych. Po trzecie, jest ono dość łatwe w implementacji. Poza już wymienionymi ma jeszcze jedną najistotniejszą zaletę – zawiera ono w sobie także poprawną analizę przypadku kątów zerowych! Zdejmuje to z nas obowiązek specjalnego traktowania tego, mogłoby się wydawać, bardzo niewygodnego przypadku. Sprawdzenie, że także i w takich przypadkach zaproponowany algorytm ma żądane własności, jest łatwym ćwiczeniem.

Rozwiązanie wzorcowe

Odtąd zakładamy, że wszystkie pierwsze oraz wszystkie drugie współrzędne lamp są różne i należą do przedziału $[1, n]$.

Pokażemy najpierw, że jeżeli ustalimy konkretną chwilę s, to jesteśmy w stanie w rozsądnej złożoności czasowej stwierdzić, które z lamp będą w tym momencie zapalone. Będziemy przetwarzać lampy w kolejności rosnących pierwszych współrzędnych. Dzięki temu, jeżeli ustalimy pewną lampę, to wszystkie lampy, które mogą potencjalnie na nią świecić, będą przetworzone przed przetworzeniem jej samej. Lampa będzie świecić w momencie s, jeżeli jej numer i jest nie większy niż s lub jeśli znajduje się ona w obszarze oświetlanym przez co najmniej k_i innych lamp (równoważnie: znajduje się w obszarze oświetlanym przez co najmniej k_i z lamp przetworzonych przed nią). Aby efektywnie sprawdzać drugi z tych warunków, będziemy utrzymywali drzewo przedziałowe. W istocie, ze wszystkich przetworzonych do tej pory lamp interesuje nas wyłącznie liczba zapalonych lamp spośród tych, które mają drugą współrzędną w przedziale $[1, u_i - 1]$, gdzie i jest numerem aktualnie rozpatrywanej lampy. Wystarczy więc drzewo przedziałowe indeksować drugimi współrzędnymi lamp i cały algorytm działa w czasie $O(n \log n)$.

Jeżeli rozpatrywalibyśmy łatwiejszą wersję naszego zadania, mianowicie, wyznaczenie momentu zapalenia konkretnej lampy, a nie wyznaczenie tego dla wszystkich lamp, to używając opisanego przed chwilą algorytmu oraz wyszukiwania binarnego, rozwiązalibyśmy ten problem efektywnie. Jednak nasz problem w istocie jest trudniejszy. Aby go rozwiązać, sprawdźmy najpierw, które lampy będą zapalone w momencie $\lfloor \frac{n}{2} \rfloor$. W ten sposób lampy podzielą nam się na dwa zbiory. Nazwijmy je: A – lampy, które świeciły w momencie $\lfloor \frac{n}{2} \rfloor$, oraz B – wszystkie pozostałe. Zauważmy, że dla lamp ze zbioru A możemy uzyskać odpowiedzi za pomocą wywołania rekurencyjnego – lampy ze zbioru B nie wpływają na momenty zapalenia się lamp ze zbioru A, zatem rzeczywiście możemy tu po prostu ograniczyć przedział poszukiwania z $[1, n]$ do $[1, \lfloor \frac{n}{2} \rfloor]$ i zapomnieć o lampach ze zbioru B. Ze zbiorem B jest podobnie. Tutaj przedział poszukiwania zawęża się do $[\lfloor \frac{n}{2} \rfloor + 1, n]$, ale nie możemy tak po prostu zapomnieć o lampach ze zbioru A, ponieważ wpływają one na momenty zapalenia się lamp z B. Przed wywołaniem rekurencyjnym dla lamp ze zbioru B i zapomnieniem w tym wywołaniu o istnieniu lamp ze zbioru A musimy dla każdej lampy ze zbioru B stwierdzić, ile lamp ze zbioru A na nią świeci, i odjąć tę liczbę od zapotrzebowania tej lampy na liczbę lamp, które muszą na nią świecić, aby ona również się zaświeciła. Jednak tę informację już mamy – wyznaczaliśmy ją przecież w sposób jawny w trakcie stwierdzania, które lampy są zapalone w momencie $\lfloor \frac{n}{2} \rfloor$! Ten pomysł wystarcza, aby rozwiązać nasze zadanie. Kolejne wywołania rekurencyjne będą przebiegały na tej samej zasadzie – dla ustalonego przedziału odpowiedzi oraz zbioru lamp, które w jego trakcie zapalą się, po prostu sprawdzamy, które lampy zapalą się do połowy tego przedziału, a które zapalą się później, aktualizujemy zapotrzebowanie lamp, które zapalą się później, i wywołujemy się rekurencyjnie dla obu części. Rekurencję przerywamy oczywiście wtedy, kiedy przedział odpowiedzi będzie jednopunktowy. Opisane rozwiązanie jest rozwiązaniem wzorcowym.

Zastanówmy się teraz nad złożonością takiego rozwiązania. Na pierwszy rzut oka nie wygląda to jak klasyczny algorytm „dziel i zwyciężaj”, ponieważ lampy mogą się

w każdym momencie wywołania rekurencyjnego podzielić bardzo nierówno, a zazwyczaj wymagane jest, aby podzieliły się one po połowie! Zauważmy jednak, że to, co dzieli się na pół, to przedział czasu – dzięki temu nasze drzewo rekurencji będzie miało wysokość co najwyżej $O(\log n)$. Ponadto zauważmy, że na każdym poziomie rekurencji każda lampa pojawia się w co najwyżej jednym wywołaniu, a czas, którego potrzebujemy na stwierdzenie dla danego zbioru lamp o rozmiarze r, które z nich będą świeciły w ustalonym momencie czasu, to $O(r \log n)$ (musimy tylko pamiętać inteligentnie zerować drzewo przedziałowe wykorzystywane w każdym wywołaniu). Tak więc skoro na danym poziomie rekurencji sumaryczna liczność zbiorów lamp nie przekracza n, to na każdym poziomie rekurencji wykonamy pracę, która zajmie nam co najwyżej $O(n \log n)$ czasu. Jako że poziomów rekurencji jest co najwyżej $O(\log n)$, to oznacza to, że przedstawione rozwiązanie działa w złożoności czasowej $O(n \log^2 n)$ oraz pamięciowej $O(n)$. Podobny algorytm (przypominający równoczesne wyszukiwanie binarne dla wielu elementów naraz) wystąpił w rozwiązaniu zadania *Meteory* z XVIII Olimpiady Informatycznej [18].

Rozwiązanie wzorcowe jest zaimplementowane w plikach `lam.cpp`, `lam2.pas` i `lam3.cpp`.

Rozwiązanie alternatywne

Rozwiązanie alternatywne opiera się na dwuwymiarowym drzewie przedziałowym. Jest trudniejsze implementacyjnie od rozwiązania wzorcowego, ale dla zawodnika posiadającego odpowiedni warsztat może być znacząco prostsze koncepcyjnie.

Struktura danych, której będziemy używać, to statyczne drzewo przedziałowe, którego liście odpowiadają kolejnym jednostkom czasu. Węzeł reprezentujący przedział $[l, r]$ zawiera zrównoważone drzewo BST (w dostarczonej implementacji jest to tzw. drzepiec, z ang. *treap*), w którym będziemy przechowywać drugie współrzędne lamp, które zaświecą się w chwili $s \in [l, r]$.

Na początku wykonajmy przekształcenie współrzędnych do postaci (t_i, u_i) oraz posortujmy lampy według pierwszej współrzędnej. Będziemy obliczać wyniki dla lamp w tej właśnie kolejności. Dzięki temu, gdy przetwarzamy daną lampę, wszystkie lampy, które ją oświetlają, mają już obliczone wyniki i znajdują się w naszej strukturze danych. Skoro w strukturze danych znajdują się tylko lampy, które mają mniejszą pierwszą współrzędną niż aktualna lampa, to jest ona oświetlana przez dokładnie te lampy ze struktury, które mają mniejszą drugą współrzędną. Zatem obliczenie wyniku dla lampy i sprowadza się do zejścia w dół drzewa przedziałowego w poszukiwaniu najwcześniejszej chwili s takiej, że do niej włącznie zaświeciło się przynajmniej k_i spośród już przetworzonych lamp, które mają drugą współrzędną mniejszą niż u_i. Minimum z tej liczby oraz i to wynik dla i-tej lampy. Kiedy już go obliczymy, możemy dodać tę lampę do drzewa i przejść do kolejnej.

Złożoność czasowa tego rozwiązania to $O(n \log^2 n)$, a pamięciowa $O(n \log n)$. Takie rozwiązanie dostawało około 90 punktów. Implementacja znajduje się w pliku `lams3.cpp`.

Rozwiązanie wolne

Rozwiązanie wolne to prosta symulacja działająca w czasie $O(n^2)$, która dla każdej kolejno zapalanej lampy przegląda wszystkie pozostałe, żeby stwierdzić, które z nich są przez nią oświetlane. Implementacja tego rozwiązania znajduje się w plikach `lams1.cpp` i `lams2.pas`. Na zawodach otrzymywała ok. 30 punktów.

Rozwiązania błędne

Zostały przygotowane cztery błędne rozwiązania będące niepoprawnymi wersjami rozwiązania wzorcowego:

`lamb1.cpp` – przy obliczaniu iloczynów wektorów nie używa typów całkowitych 64-bitowych – dostaje 60 punktów;

`lamb2.cpp` – nie obsługuje przypadku kąta równego zero stopni – dostaje 80 punktów;

`lamb3.cpp` – ignoruje lampy leżące na brzegu oświetlanego obszaru – dostaje 30 punktów;

`lamb4.cpp` – działa tak, jakby lampa zaczynała oświetlać inne lampy dopiero po podłączeniu jej do prądu, nawet jeśli zaświeciła się wcześniej – dostaje 0 punktów (takie zadanie można by rozwiązać znacząco prościej).

Testy

Zestaw testów zawierał dziesięć grup, po dwa testy w każdej grupie. W każdej grupie pierwszy test jest losowy, natomiast drugi został wygenerowany według jednej z metod opisanych poniżej:

1b, 6b, 8b, 10b – po przekształceniu współrzędnych pozycje lamp tworzą regularną kratkę,

2b, 7b – lampy ustawione są w linii, każda oświetla każdą następną, wszystkie zapalają się w tym samym momencie,

3b, 9b – kąt ma zero stopni,

4b, 5b – lampy ustawione są w dwóch rzędach – każda lampa z pierwszego oświetla każdą lampę z drugiego.