

## Zadanie A - Potrójne drzewa

W naszym rozwiązaniu potrzebne nam będą dwie tablice, *WYNIK* i *DRZEWO*. W tablicy *WYNIK* zapiszemy wartości, które odczytamy z wejścia oraz skorzystamy z niej przy odczytywaniu wyniku. W tablicy *DRZEWO* wpisujemy natomiast w komórkę odpowiadającą odczytanej wartości  $i$  liczbę odpowiadającą numerowi porządkowemu zapytania postawionego przez Jasia. Tak więc dla przykładowego wejścia

9 1 2 3 4 5 6 7 8

wartości w naszych tablicach będą następujące:

WYNIK -> 9 1 2 3 4 5 6 7 8

DRZEWO -> 2 3 4 5 6 7 8 9 1

Następnie zbudujemy strukturę drzewa trzy-regularnego na tablicy *DRZEWO*. Dla każdego z trzech potomnych wierzchołków będziemy przekazywać do rodzica maksymalną dostępną wartość. Tak postępujemy do czasu, kiedy pozostanie jeden wierzchołek.

DRZEWO ->               9

DRZEWO ->   4       7       9

DRZEWO -> 2 3 4 5 6 7 8 9 1

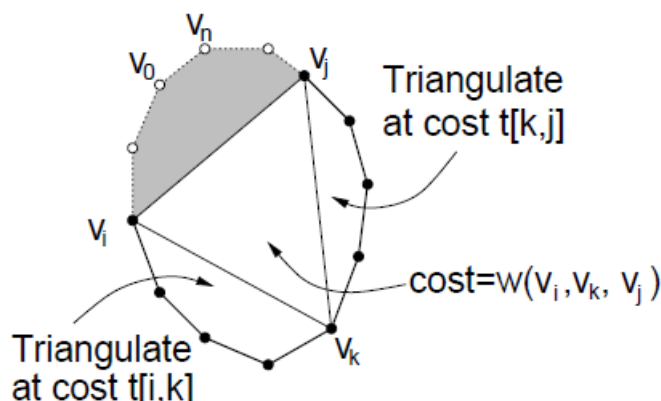
Następnie możemy wybrać wartość na samej górze drzewa (1 lub 0). Wymusza to wartości dla potomków tego wierzchołka (w przykładzie oznacza to, że wartość dla 4 jest różna od wartości dla 7). Postępując w ten sam sposób na kolejnych poziomach naszego drzewa otrzymamy wartościowanie wszystkich jego liści. Na koniec korzystając z tablicy *WYNIK* możemy wypisać wartości w zadanej kolejności.

## Zadanie B - Triangulacja

Rozwiązanie jest zastosowaniem metody programowania dynamicznego.

Częściowe wyniki przechowywać będziemy w macierzy  $n \times n$ , gdzie  $n$  jest liczbą wierzchołków zadanego wielokąta wypukłego. Oznaczmy tę macierz przez  $t$ . Element  $t[i, j]$  będzie zawierał najmniejszą możliwą sumę obwodów trójkątów wyznaczonych przez triangulację wielokąta rozpiętego na wierzchołkach  $\langle v_i, v_{i+1}, \dots, v_j \rangle$ . Na początku definiujemy wagę „dwukąta” jako zero, tzn.  $t[i, i+1] = 0$ . W dalszej kolejności rozważać będziemy minimalne triangulacje kolejnych trójkątów, czworokątów, pięciokątów,... itd. złożonych z kolejnych wierzchołków wyjściowego wielokąta.

Wyznaczając wartość  $t[i, j]$ , rozważamy wielokąt rozpięty na wierzchołkach  $\langle v_i, v_{i+1}, \dots, v_j \rangle$ , gdzie  $j > i + 1$ . Odcinek  $\overline{v_i v_j}$  jest jednym z boków tego wielokąta. Zauważmy, że możemy podzielić nasz wielokąt trójkątem rozpiętym na tym odcinku i dowolnym wierzchołku  $v_k$ , gdzie  $i < k < j$ . Trójkąt ten dzieli nasz wielokąt na wielokąty  $\langle v_i, v_{i+1}, \dots, v_k \rangle$  i  $\langle v_k, v_{k+1}, \dots, v_j \rangle$ .



Rysunek 1: Podział  $\langle v_i, v_{i+1}, \dots, v_j \rangle$  na dwa mniejsze wielokąty (Źródło obrazka).

Do wyznaczenia minimalnej długości triangulacji możemy teraz wykorzystać wartości  $t[i, k]$  oraz  $t[k, j]$  obliczone we wcześniejszych iteracjach. Doliczając do wyniku obwód trójkąta  $\Delta v_i v_k v_j$  (oznaczony przez  $w(v_i v_k v_j)$ ), otrzymujemy zależność rekurencyjną:

$$t[i, j] = \begin{cases} 0 & \text{jeżeli } j = i + 1 \\ \min_{i < k < j} (t[i, k] + t[k, j] + w(v_i v_k v_j)) & \text{jeżeli } j > i + 1. \end{cases}$$

Ostateczny wynik możemy odczytać odejmując od wartości komórki  $t[0, n]$  długość obwodu wielokąta i dzieląc otrzymany wynik przez 2. Złożoność obliczeniowa rozwiązania wynosi  $O(n^3)$ .

Dokładniejszą analizę tego problemu można znaleźć w starszych wersjach Cormena lub np. na stronie <http://www.cs.sunysb.edu/~jgao/CSE548-fall107/David-mount-DP.pdf>.

## Zadanie C - Strażak

Popatrzmy na opis powiązań pomiędzy budynkami naszego miasta jak na graf nieskierowany. Aby rozwiązać zadanie, musimy po pierwsze zauważyć, że dla ukorzonego grafu  $(G, r)$  o maksymalnym stopniu  $\Delta(G) \leq 3$  oraz  $\deg(r) \leq 2$ , istnieje optymalna strategia ochrony budynków, w której każdy wybierany do ochrony wierzchołek jest sąsiadem co najmniej jednego palącego się budynku.

Poprzez analizę treści zadania, możemy wyróżnić następujące przypadki:

- Wierzchołek, w którym wybuchł pożar, ma stopień równy 3  $\Rightarrow$  odpowiedź jest równa  $|G| - 1$ .
- Wierzchołek, w którym wybuchł pożar, ma stopień 1  $\Rightarrow$  odpowiedź jest równa  $|G| - 1$ .
- Jedyny przypadek, który musimy przeanalizować dokładniej, to sytuacja, w której  $\deg(r) = 2$ . Rozważmy dwa podzbiory zbioru domostw:  $V_2$  - domostwa o stopniu równym co najwyżej dwa, oraz  $V_C$  - domostwa o stopniu równym 3 należące do cykli grafu reprezentującego miasto. Nasze zadanie sprowadza się do minimalizacji funkcji  $f$  zdefiniowanej następująco (dla każdego wierzchołka  $u$ ):

### Definicja

$$f(u) = \begin{cases} C(u) - 2 & \text{dla } u = r \text{ (o ile } C(u) \text{ jest określone)} \\ \text{dist}(u, r) + 1 & \text{dla } u \in V_2 \setminus \{r\} \\ \text{dist}(u, r) + C(u) - 1 & \text{dla } u \in V_C \end{cases},$$

gdzie  $C(u)$  to długość najkrótszego cyklu, w którym zawiera się wierzchołek  $u$ . Można udowodnić, że nie da się ochronić przed spalaniem więcej niż  $|G| - \min\{f(u) | x \in V_2 \cup V_C \cup \{r\}\}$  wierzchołków. Algorytm obliczania wartości funkcji  $f$  jest następujący. Najpierw używamy algorytmu BFS, by dla każdego wierzchołka obliczyć jego odległość od źródła pożaru  $r$ . W kroku drugim, dla każdego wierzchołka stopnia 3 szukamy (znów za pomocą BFS) długości minimalnego cyklu, w którym się on znajduje. Jako odpowiedź podajemy wartość  $|G| - \min\{f(u) | x \in V_2 \cup V_C \cup \{r\}\}$ .

## Zadanie D - Słoneczna wyspa

W języku teoriografowym zadanie polega na wyznaczeniu liczby multichromatycznej dowolnego grafu zewnętrzplanarnego, a następnie multikolorowania takiego grafu. Graf zewnętrzplanarny to graf planarny, dla którego istnieje takie zanurzenie w  $\mathbb{R}^2$ , że każdy wierzchołek znajduje się na zewnętrznej ścianie. Graf wejściowy spełnia to założenie, gdyż wszystkie wierzchołki leżą na brzegu pewnego cyklu lub na wychodzących z niego prostych ścieżkach, które nie tworzą dodatkowej ściany. Dodatkowo krawędzie w środku cyklu (mosty) z założenia się nie przecinają, więc nie psują planarności. Multikolorowanie to funkcja, która wierzchołkom przypisuje pewne zbiory kolorów w taki sposób, aby dwa wierzchołki połączone krawędzią posiadały rozłączne zbiory kolorów. Opisane w treści zadania założenia prowadzą wprost do takiej funkcji.

Aby rozwiązać postawiony problem, trzeba w rzeczywistości rozwiązać kilka podproblemów.

- wyznaczyć minimalne cykle
- uszeregować cykle w odpowiedniej kolejności
- multikolorować cykle
- multikolorować promienie (ścieżki)

Wyznaczenie minimalnych cykli jest problemem dość prostym, w którym można wykorzystać np. przeszukiwanie grafu wszerz.

Uszeregowanie cykli powinno odbyć się w taki sposób, aby rozpoczynać od dowolnego cyklu, a następnie kolorować te cykle, które mają dokładnie dwa wierzchołki już pokolorowane. Można tego dokonać poprzez stworzenie dodatkowego grafu, w którym wierzchołki reprezentują minimalne cykle, a krawędzie łączą dwa cykle wtedy i tylko wtedy, kiedy posiadają one dokładnie dwa wspólne wierzchołki. Przeszukanie takiego grafu dowolną metodą wyznaczy nam kolejność multikolorowania cykli.

Rozwiązanie problemu multikolorowania cykli (podobnie zresztą jak całego zadanego tu problemu) jest szczegółowo opisane w pracy Narayanan, Shende, *Static Frequency Assignment in Cellular Networks*, Algorithmica (2001) 29: 396–409 w rozdziale 3 *Optimal Multicoloring of Cycles* (str. 399-400).

Mutikolorowanie ścieżek jest problemem dość prostym. Wystarczy wyznaczyć liczbę multichromatyczną jako największą sumę wag wierzchołków połączonych krawędzią. Następnie należy wziąć przedział liczb od 1 do tej wyznaczonej wartości i na przemian kolorować wierzchołki kolorami z początku i końca tego przedziału. Metoda jest analogiczna do metody opisanej we wspomnianej pracy w przypadku cykli parzystych.

Odpowiedzią na wyjściu powinna być maksymalna wartość liczby multichromatycznej wyznaczona dla cykli oraz promieni. Multikolorowanie powinno być wyznaczone poprzez odpowiednie przekolorowywanie multikolorowań cykli według wyznaczonej kolejności, a następnie przekolorowanie promieni, zawsze zaczynając od wierzchołka sąsiadującego z cyklem.

## Zadanie E - Dziadkowie

Chociaż można ryzykować, zakładając, że niewiadome w zadaniu są dobrane zdroworozsądkowo, to wyznaczenie ich komputerowo nie powinno zajmować wiele czasu (komputera na obliczenia oraz człowieka na implementację).

Na wstępie warto zauważyć, iż  $X$  nie może być tak duże, żeby  $X^2$  oraz  $X^3$  zawierały łącznie więcej niż 10 cyfr. Stąd można je ograniczyć z góry np. przez 100, bo  $100^2$  i  $100^3$  mają łącznie 12 cyfr. Prawidłowa wartość  $X$  wynosi 69. Ponieważ  $9!$  wynosi tylko 362880, nic nie stoi na przeszkodzie, by znaleźć  $G$  rozpatrując wszelkie możliwe permutacje. Dla porządku nadmienimy, iż  $G = 2^{32}$ .

Ostatecznie kod rozwiązujący problem jest następujący:

```
int main()
{
    const int X = 69;
    int iTests;
    scanf( "%d", &iTests );
    long long N;
    for( int iTestCnt = 1; iTestCnt <= iTests; iTestCnt++ )
    {
        scanf( "%lld", &N );
        if( iTestCnt % X == 0 )
            printf( "AMPPZ 2010\n");
        else
            printf( "%lld\n", (3*N+3)/7 );
    }
    return 0;
}
```

Należy zwrócić uwagę, by zmienna  $N$  była typu 64-bitowego. Gdyby zmienna  $N$  była 32-bitowa, to w drugim wywołaniu funkcji printf nastąpiłoby przekroczenie zakresu.

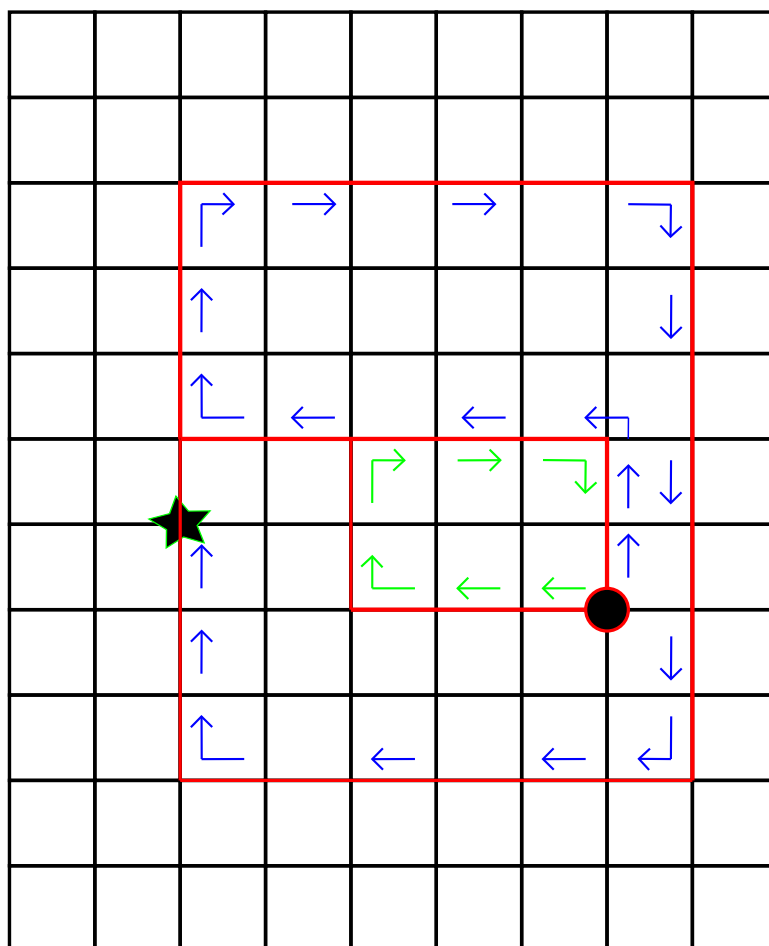
## Zadanie F - Ciasteczkowy potwór

Zadanie polega na znalezieniu dwóch sąsiednich cyfr usuniętych z silni pewnej liczby. Rozwiązanie wzorcowe generuje silnie dla  $n = 5, \dots, 11$ , natomiast dla  $n > 11$  bada podzielność zadanej silni przez 7, 9 i 11. Liczby te wybrane są z tego powodu, że podzielność przez 9 i 11 możemy wyznaczyć w prosty sposób, korzystając tylko z dodawania i odejmowania oraz jednej operacji modulo. Niestety, jak można zauważyć, podzielność przez liczby 11 i 9 nie rozróżnia przypadków, gdy dwie usunięte cyfry to 0 0 lub 9 9. Stąd też musimy wykorzystać dodatkowo informację na temat reszty z dzielenia naszej silni np. przez 7.

## Zadanie G - Spacer

W rozwiązaniu tego zadania zapamiętujemy na każdym skrzyżowaniu, które odwiedził Jaś, kierunek, z którego przybył on na to skrzyżowanie i kierunek, w którym udał się opuszczając je. Następnie, zaczynając od punktu, w którym Jaś ukończył swoją wędrówkę, próbujemy cofnąć się jego śladami przy zachowaniu zasady, że na każdym skrzyżowaniu skręcamy możliwie najbardziej w prawo (to znaczy, jeżeli jest to możliwe, najpierw wybieramy skręt w prawo, następnie pójście prosto, zaś w późniejszej kolejności skręt w lewo czy też wycofanie się). Jeżeli w którymś momencie trafimy do punktu początkowego, z którego Jaś rozpoczął swoją wędrówkę, oznacza to, że jest możliwy jego powrót do domu unikający przecięcia w jakimkolwiek punkcie szlaku jego pierwotnego spaceru. Jeżeli natomiast na naszej drodze ponownie trafimy do punktu, w którym Jaś ukończył swój spacer, oznacza to, że taki powrót nie jest możliwy.

Jeżeli z punktu, w którym Jaś ukończył swój spacer, możemy pójść jego śladami w więcej niż jednym kierunku, to uruchamiamy nasz algorytm niezależnie dla każdej z dróg (tzn. w każdym z możliwych kierunków). Obrazuje to poniższy rysunek :



Rysunek 2: Działanie algorytmu. Gwiazdka - punkt startowy Jasia, kółko - punkt końcowy spaceru Jasia. Na czerwono oznaczono drogi, którymi Jaś przechodził w trakcie spaceru. Zielone strzałki oznaczają drogę algorytmu przy pierwszym uruchomieniu, niebieskie przy drugim. Odpowiedź jest w tym przypadku równa „TAK”.

## Zadanie H - Suma kontrolna

Po sprawdzeniu, że generator  $G$  jest liczbą pierwszą, wystarczy dokonać odpowiedniego xoro-  
wania wiadomości  $M'$  z  $G$ , by obliczyć sumę kontrolną (zakładam, że `acMessage` zawiera bity  
 $M$  oraz wystarczającą liczbę miejsc, by dopisać  $|G| - 1$  zer; `acGen` zawiera bity generatora  $G$ ):

```
typedef long long ll;

ll solve()
{
    for( int iCnt = 0; iCnt < iGenLen - 1; iCnt++ )
        acMessage[ iMessLen + iCnt ] = 0;
    iMessLen += ( iGenLen - 1 );

    for( int iPos = 0; iPos <= iMessLen - iGenLen; iPos++ )
    {
        if( acMessage[ iPos ] == 0 )
            continue;
        for( int iPosGen = 0; iPosGen < iGenLen; iPosGen++ )
            acMessage[ iPos + iPosGen ] ^= acGen[ iPosGen ];
    }

    ll llRes = 0, llFactor = 1;
    for( int iCnt = 0; iCnt < iGenLen - 1; iCnt++ )
    {
        llRes += ( acMessage[ iMessLen - 1 - iCnt ] * llFactor );
        llFactor *= 2;
    }

    return llRes;
}
```

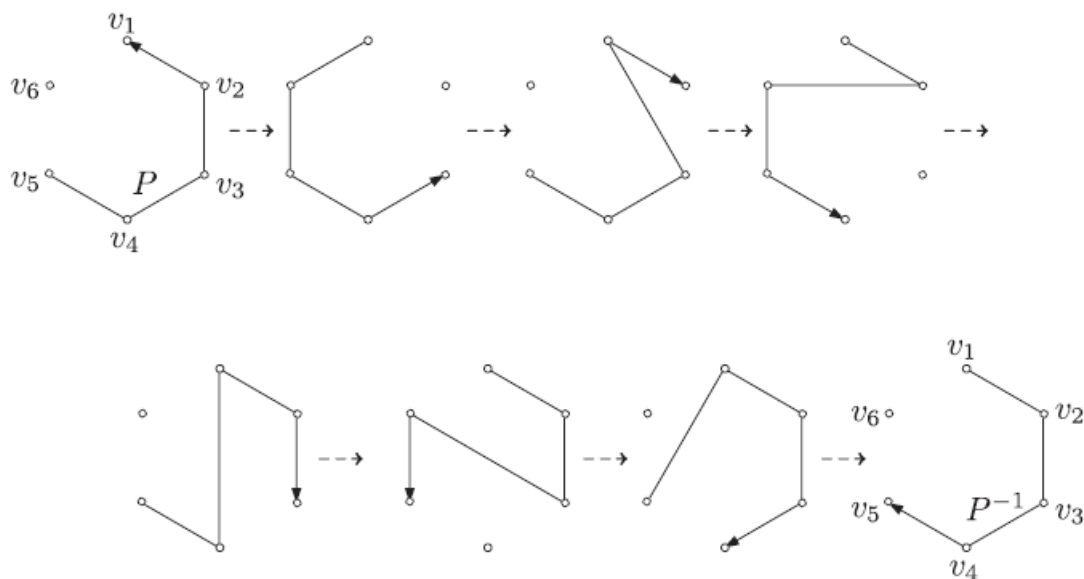
Pozostaje odpowiedź na pytanie, jak szybko sprawdzić, czy liczba 48-bitowa jest pierwsza. Można to zrobić przez swoistą „derandomizację” algorytmu (testu) Millera-Rabina. Faktem jest bowiem, że dla świadków złożoności 2, 3, 5, 7, 11, 13, 17 test Millera-Rabina nie myli się, jeśli badamy pierwszość liczby mniejszej od 341550071728312. Więcej informacji o teście Millera-Rabina można znaleźć np. we „Wprowadzeniu do algorytmów” (wyd. PWN) lub w „Algorytmice praktycznej” (wyd. PWN).



## Zadanie I - Wąż

Najpierw należy zauważyć, że dla  $n \geq 4$  zawsze możemy odwrócić węża, jeśli jego długość jest nie większa niż  $n - 1$ . Jeśli wąż jest dłuższy lub  $n = 3$ , to węża nie daje się odwrócić.

Mając dane ułożenie początkowe węża  $v_1, \dots, v_n$  przechodzimy nim po grafie w taki sposób, aby wierzchołek  $v_1$  znalazł się poza zbiorem wierzchołków zajmowanych przez węża (lub też był ogonem węża). Następnie przechodzimy głowę węża do wierzchołka  $v_1$ , a potem do wierzchołka  $v_2$ . Operacja ta pozwala nam odwrócić orientację jednej krawędzi węża. W dalszych krokach powtarzamy tę samą metodę powiększając za każdym razem fragment odwróconej części węża o 1. Całość obrazuje rysunek:



Rysunek 3: Kolejne kroki algorytmu.

Rozwiązanie jest realizowane przez kod:

```
printf("%d\n", n*(m-1));
for(int y=0; y<m-1; y++){
    for(int i=0; i<n-2-y; i++) printf("%d ", wi[i]);
    for(int i=n-1; i>=n-2-y; i--) printf("%d ", wi[i]);
}
```

Tablica „wi” przechowuje tu wierzchołki grafu posortowane w taki sposób, by najpierw występowało w niej  $n - m$  wierzchołków, których wąż nie zajmuje w początkowym ustawieniu, a następnie  $m$  wierzchołków, które zajmuje on na początku, podanych w kolejności od ogona do głowy.

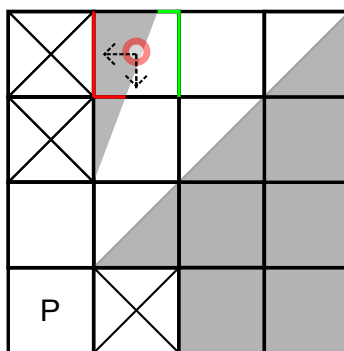
Zadanie było zainspirowane wynikami Ryuzo Toriego.

## **Zadanie J - Chciwcy**

Celem zadania było wyliczenie wartości uogólnionej funkcji Sprague'a-Grundy'ego na grafie. Definicję funkcji i sam algorytm jej wyznaczania można odnaleźć na stronie 15 publikacji Aviezri S. Fraenkela.

## Zadanie K - ADOM

W naszych rozważaniach skupimy się tylko na blokach leżących ponad i na prawo od punktu obserwacji, gdyż dla każdej innej ćwiartki układu współrzędnych można oddzielnie zastosować podaną metodę. Zaczynając od punktu obserwatora „P” przechodzimy po planszy rozpatrując pola w kolejności niemalejących odległości od „P” w metryce taksówkowej. Dla każdego pola badamy, w jakim zakresie jest ono widoczne, spoglądając na jego bezpośrednich sąsiadów znajdujących się na lewo i pod nim.



Rysunek 4: Na czerwono oznaczono aktualnie przetwarzane pole. Patrząc na sąsiadów badamy widoczność tego pola i zapamiętujemy widoczność z niego do jego górnego oraz prawego sąsiada.

Osobno trzeba określić, czy fragment ściany, teoretycznie widoczny z punktu „P”, nie znajduje się poza promieniem widzialności gracza, gdyż w takim wypadku blok ten nie jest widoczny. Przykład takiego specyficznego testu jest pokazany poniżej:

```

1
10 19 4
...XXXX.X....XX.X.
X.XXX.X.X.X.X..XX..
.....XX...X....XX.X
.XX.....XXX...XX..
...X.....X...X
..X..X...P....X...
X.XX.X.....X....
X.X.....XX....XX.X
.X.XX.X.X...X..XX.X
X...X..X.XXXX...X..

.....
.....X.....
.....X.....
.....XXX.....
.....
.....X...P.....
.....X.....
.....XX.....

```

```

.....X.X...X.....
.....*.....

```

Powyżej przez „\*” oznaczono ścianę, której widoczna część znajduje się poza promieniem widzialności. Prawidłową odpowiedzią jest w tym przypadku:

```

.....
.....X.....
.....X.....
.....XXX.....
.....
.....X...P.....
.....X.....
.....XX.....
.....X.X...X.....
.....

```