راهنما 0= خالی 1 = 2=

**تابعی برای تشخیص مهره‍های تحت تاثیر**

* ورودی تابع : شماره سطر + شماره ستون.
* این تابع مقدار متغییر affectedDiscs بعنوان خروجی برمیگرداند.

**بررسی به از چپ به سمت راست:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 2 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

شماره ستون را در متغییری به نام columnIterator قرار میدهیم.

(شروع حلقه) تا مادامی که مقدار columnIterator کمتر از تعداد ستونها (0-7) است دستورات زیر اجرا شود:

با هر بار اجرای حلقه ابتدا یک عدد به مقدار columnIterator اضافه میکند تا مقدار خانه سمت راست خود را مورد بررسی قرار دهد. مانند تصویر زیر که روی خانه سبز کلیک شد و برنامه مقدار خانه قرمز را مورد ارزیابی قرار میدهد. در اینجا مقدار خانه سبز رنگ را valueAtSpot می نامیم.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | column | | | | | | | |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| row | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 2 | 1 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

اگر valueAtSpot مساوی 0 بود **یا** valueAtSpot هنگام حرکت سیاه(1) یا سفید(2) برار با عدد 1 یا 2 بود پس :

اگر هنگام حرکت سیاه(1) یا سفید(2) برار با عدد 1 یا 2 بود پس :

مقدار affectedDiscs برار شود با : مقدار affectedDiscs + مقدار couldBeAffected

خروج از حلقه.

در غیر اینصورت :

موقعیت هر مهره بصورت object به متغییر couldBeAffected بصورت زیر اضافه شود.

discLocation = {row: شماره سطر; column: columnIterator}

ادامه حلقه.

**مثال :**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | column | | | | | | | |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| row | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 |
| 2 | 0 | 0 | 2 | 2 | 2 | 0 | 0 | 0 |
| 3 | 0 | 0 | 2 | 2 | 2 | 1 | 0 | 0 |
| 4 | 0 | 2 | 0 | 2 | 2 | 1 | 1 | 0 |
| 5 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

در مثال بالا (با فرض اینکه نوبت حرکت مهره سیاه(1) است) با اجرا شدن حلقه نتیجه ی زیر حاصل میگردد :

couldBeAffected = [

{row: 3, column: 2}

{row: 3, column: 3}

{row: 3, column: 4}

]

با رسیدن حلقه به موقعیت ( row : 3 , column: 5 ) از آنجایی که حرکت با مهره سیاه (عدد 1) است و مقدار آن خانه هم برار با عدد 1 است ، پس حلقه با قرار دادن مقدار couldBeAffected در affectedDiscs به کار خود خاتمه می‍دهد.

**بررسی از راست به سمت چپ :**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 2 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

شماره ستون را در متغییری به نام columnIterator قرار میدهیم.

(شروع حلقه) : تا مادامی که مقدار columnIterator بیشتر از صفر (0) است دستورات زیر اجرا شود:

با هر بار اجرای حلقه ابتدا یک عدد به از columnIterator کم میکند تا مقدار خانه سمت چپ خود را مورد بررسی قرار دهد. مانند تصویر زیر که روی خانه سبز کلیک شد و برنامه مقدار خانه قرمز را مورد ارزیابی قرار میدهد. در اینجا مقدار خانه سبز رنگ را valueAtSpot می نامیم.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | column | | | | | | | |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| row | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 2 | 1 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

اگر valueAtSpot مساوی 0 بود **یا** valueAtSpot هنگام حرکت سیاه(1) یا سفید(2) برار با عدد 1 یا 2 بود پس :

اگر هنگام حرکت سیاه(1) یا سفید(2) برار با عدد 1 یا 2 بود پس :

مقدار affectedDiscs برار شود با : مقدار affectedDiscs + مقدار couldBeAffected

خروج از حلقه.

در غیر اینصورت :

موقعیت هر مهره بصورت object به متغییر couldBeAffected بصورت زیر اضافه شود.

discLocation = {row: شماره سطر; column: columnIterator}

ادامه حلقه.

**.مثال :**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | column | | | | | | | |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| row | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 2 | 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 2 | 0 | 1 | 0 | 0 | 0 |
| 3 | 0 | 1 | 2 | 1 | 1 | 1 | 0 | 0 |
| 4 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 5 | 0 | 2 | 2 | 1 | 0 | 1 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

در مثال بالا (با فرض اینکه نوبت حرکت مهره سفید(2) است) با اجرا شدن حلقه نتیجه ی زیر حاصل میگردد :

couldBeAffected = [

{row: 3, column: 5}

{row: 3, column: 4}

{row: 3, column: 3}

]

با رسیدن حلقه به موقعیت ( row : 3 , column: 2 ) از آنجایی که حرکت با مهره سفید (عدد 2) است و مقدار آن خانه هم برار با عدد 2 است ، پس حلقه با قرار دادن مقدار couldBeAffected در affectedDiscs به کار خود خاتمه می‍دهد.

**بررسی به از پائین به سمت بالا :**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 2 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

شماره سطر را در متغییری به نام rowIterator قرار میدهیم.

(شروع حلقه) : تا مادامی که مقدار rowIterator بیشتر از صفر (0) است دستورات زیر اجرا شود:

با هر بار اجرای حلقه ابتدا یک عدد به از rowIterator کم میکند تا مقدار خانه بالاتر از خود را خود را مورد بررسی قرار دهد. مانند تصویر زیر که روی خانه سبز کلیک شد و برنامه مقدار خانه قرمز را مورد ارزیابی قرار میدهد. در اینجا مقدار خانه سبز رنگ را valueAtSpot می نامیم.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | column | | | | | | | |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| row | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 2 | 1 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

اگر valueAtSpot مساوی 0 بود **یا** valueAtSpot هنگام حرکت سیاه(1) یا سفید(2) برار با عدد 1 یا 2 بود پس :

اگر هنگام حرکت سیاه(1) یا سفید(2) برار با عدد 1 یا 2 بود پس :

مقدار affectedDiscs برار شود با : مقدار affectedDiscs + مقدار couldBeAffected

خروج از حلقه.

در غیر اینصورت :

موقعیت هر مهره بصورت object به متغییر couldBeAffected بصورت زیر اضافه شود.

discLocation = {row: rowIterator; column: شماره ستون}

ادامه حلقه.

**.مثال :**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | column | | | | | | | |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| row | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 2 | 2 | 2 | 0 | 0 | 0 |
| 2 | 0 | 0 | 2 | 1 | 1 | 0 | 0 | 0 |
| 3 | 0 | 1 | 2 | 1 | 1 | 1 | 0 | 0 |
| 4 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 5 | 0 | 2 | 2 | 1 | 0 | 1 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

در مثال بالا (با فرض اینکه نوبت حرکت مهره سفید(2) است) با اجرا شدن حلقه نتیجه ی زیر حاصل میگردد :

couldBeAffected = [

{row: 5, column: 3}

{row: 4, column: 3}

{row: 3, column: 3}

{row: 2, column: 3}

]

با رسیدن حلقه به موقعیت ( row : 1 , column: 3 ) از آنجایی که حرکت با مهره سفید (عدد 2) است و مقدار آن خانه هم برار با عدد 2 است ، پس حلقه با قرار دادن مقدار couldBeAffected در affectedDiscs به کار خود خاتمه می‍دهد.

**بررسی به از بالا به سمت پائین :**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 2 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

شماره سطر را در متغییری به نام rowIterator قرار میدهیم.

(شروع حلقه) : تا مادامی که مقدار rowIterator کمتر از تعداد سطر (0تا7) است دستورات زیر اجرا شود:

با هر بار اجرای حلقه ابتدا یک عدد به rowIterator اضافه می‌کند تا مقدار خانه پائین‌تر از خود را خود را مورد بررسی قرار دهد. مانند تصویر زیر که روی خانه سبز کلیک شد و برنامه مقدار خانه قرمز را مورد ارزیابی قرار میدهد. در اینجا مقدار خانه سبز رنگ را valueAtSpot می نامیم.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | column | | | | | | | |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| row | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 2 | 1 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

اگر valueAtSpot مساوی 0 بود **یا** valueAtSpot هنگام حرکت سیاه(1) یا سفید(2) برار با عدد 1 یا 2 بود پس :

اگر هنگام حرکت سیاه(1) یا سفید(2) برار با عدد 1 یا 2 بود پس :

مقدار affectedDiscs برار شود با : مقدار affectedDiscs + مقدار couldBeAffected

خروج از حلقه.

در غیر اینصورت :

موقعیت هر مهره بصورت object به متغییر couldBeAffected بصورت زیر اضافه شود.

discLocation = {row: rowIterator; column: شماره ستون}

ادامه حلقه.

**.مثال :**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | column | | | | | | | |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| row | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 2 | 2 | 2 | 0 | 0 | 0 |
| 2 | 0 | 0 | 2 | 1 | 1 | 0 | 0 | 0 |
| 3 | 0 | 1 | 2 | 1 | 1 | 1 | 0 | 0 |
| 4 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 5 | 0 | 2 | 2 | 1 | 0 | 1 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

در مثال بالا (با فرض اینکه نوبت حرکت مهره سفید(2) است) با اجرا شدن حلقه نتیجه ی زیر حاصل میگردد :

couldBeAffected = [

{row: 3, column: 1}

{row: 4, column: 1}

]

با رسیدن حلقه به موقعیت ( row : 5 , column: 1 ) از آنجایی که حرکت با مهره سفید (عدد 2) است و مقدار آن خانه هم برار با عدد 2 است ، پس حلقه با قرار دادن مقدار couldBeAffected در affectedDiscs به کار خود خاتمه می‍دهد.

**بررسی اُریب از بالا چپ به سمت پائین راست :**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 2 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

شماره سطر را در متغییری به نام rowIterator قرار میدهیم.

شماره سطر را در متغییری به نام columnIterator قرار میدهیم.

(شروع حلقه) : تا مادامی که مقدار rowIterator کمتر از تعداد سطر (0تا7) و columnIterator کمتر از تعداد ستون‌ها (0تا7) است دستورات زیر اجرا شود:

با هر بار اجرای حلقه ابتدا یک عدد به rowIterator و columnIterator اضافه می‌کند تا مقدار خانه پائین‌تر در سطر و ستون بعد از خود را خود را مورد بررسی قرار دهد. مانند تصویر زیر که روی خانه سبز کلیک شد و برنامه مقدار خانه قرمز را مورد ارزیابی قرار میدهد. در اینجا مقدار خانه سبز رنگ را valueAtSpot می نامیم.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | column | | | | | | | |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| row | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

اگر valueAtSpot مساوی 0 بود **یا** valueAtSpot هنگام حرکت سیاه(1) یا سفید(2) برار با عدد 1 یا 2 بود پس :

اگر هنگام حرکت سیاه(1) یا سفید(2) برار با عدد 1 یا 2 بود پس :

مقدار affectedDiscs برار شود با : مقدار affectedDiscs + مقدار couldBeAffected

خروج از حلقه.

در غیر اینصورت :

موقعیت هر مهره بصورت object به متغییر couldBeAffected بصورت زیر اضافه شود.

discLocation = {row: rowIterator; column: شماره ستون}

ادامه حلقه.

**.مثال :**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | column | | | | | | | |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| row | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 2 | 0 | 2 | 2 | 2 | 1 | 1 | 0 | 0 |
| 3 | 0 | 2 | 2 | 2 | 2 | 1 | 1 | 2 |
| 4 | 0 | 2 | 2 | 2 | 2 | 2 | 1 | 2 |
| 5 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 2 |
| 6 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 2 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |

در مثال بالا (با فرض اینکه نوبت حرکت مهره سیاه (1) است) با اجرا شدن حلقه نتیجه ی زیر حاصل میگردد :

couldBeAffected = [

{row: 2, column: 3}

{row: 3, column: 4}

{row: 4, column: 5}

]

با رسیدن حلقه به موقعیت ( row : 5 , column: 6 ) از آنجایی که حرکت با مهره سیاه (عدد 1) است و مقدار آن خانه هم برار با عدد 1 است ، پس حلقه با قرار دادن مقدار couldBeAffected در affectedDiscs به کار خود خاتمه می‍دهد.

**بررسی اُریب از بالا راست به سمت پائین چپ :**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 2 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

شماره سطر را در متغییری به نام rowIterator قرار میدهیم.

شماره سطر را در متغییری به نام columnIterator قرار میدهیم.

(شروع حلقه) : تا مادامی که مقدار rowIterator کمتر از تعداد سطر (0تا7) و columnIterator بیشتر از صفر است دستورات زیر اجرا شود:

با هر بار اجرای حلقه ابتدا یک عدد به rowIterator اضافه و یک عدد از columnIterator کم می‌کند تا مقدار خانه پائین‌تر در سطر و ستون قبل از خود را خود را مورد بررسی قرار دهد. مانند تصویر زیر که روی خانه سبز کلیک شد و برنامه مقدار خانه قرمز را مورد ارزیابی قرار میدهد. در اینجا مقدار خانه سبز رنگ را valueAtSpot می نامیم.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | column | | | | | | | |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| row | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 2 | 2 | 2 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 2 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

اگر valueAtSpot مساوی 0 بود **یا** valueAtSpot هنگام حرکت سیاه(1) یا سفید(2) برار با عدد 1 یا 2 بود پس :

اگر هنگام حرکت سیاه(1) یا سفید(2) برار با عدد 1 یا 2 بود پس :

مقدار affectedDiscs برار شود با : مقدار affectedDiscs + مقدار couldBeAffected

خروج از حلقه.

در غیر اینصورت :

موقعیت هر مهره بصورت object به متغییر couldBeAffected بصورت زیر اضافه شود.

discLocation = {row: rowIterator; column: شماره ستون}

ادامه حلقه.

**.مثال :**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | column | | | | | | | |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| row | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 2 | 2 | 2 | 0 | 0 | 0 |
| 2 | 1 | 1 | 2 | 2 | 2 | 0 | 0 | 0 |
| 3 | 1 | 2 | 2 | 2 | 2 | 0 | 0 | 0 |
| 4 | 1 | 1 | 2 | 2 | 2 | 0 | 0 | 0 |
| 5 | 1 | 1 | 2 | 2 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 1 | 2 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

در مثال بالا (با فرض اینکه نوبت حرکت مهره سیاه (1) است) با اجرا شدن حلقه نتیجه ی زیر حاصل میگردد :

couldBeAffected = [

{row: 1, column: 3}

{row: 2, column: 2}

{row: 3, column: 1}

]

با رسیدن حلقه به موقعیت ( row : 4 , column: 0 ) از آنجایی که حرکت با مهره سیاه (عدد 1) است و مقدار آن خانه هم برار با عدد 1 است ، پس حلقه با قرار دادن مقدار couldBeAffected در affectedDiscs به کار خود خاتمه می‍دهد.

**بررسی اُریب از پائین راست به سمت بالا چپ :**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 2 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

شماره سطر را در متغییری به نام rowIterator قرار میدهیم.

شماره سطر را در متغییری به نام columnIterator قرار میدهیم.

(شروع حلقه) : تا مادامی که مقدار rowIterator بیشتر از صفر و نیز columnIterator بیشتر از صفر است دستورات زیر اجرا شود:

با هر بار اجرای حلقه ابتدا یک عدد از rowIterator و یک عدد از columnIterator کم می‌کند تا مقدار خانه بالا‌تر در سطر و ستون قبل از خود را خود را مورد بررسی قرار دهد. مانند تصویر زیر که روی خانه سبز کلیک شد و برنامه مقدار خانه قرمز را مورد ارزیابی قرار میدهد. در اینجا مقدار خانه سبز رنگ را valueAtSpot می نامیم.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | column | | | | | | | |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| row | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 2 | 2 | 2 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 2 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

اگر valueAtSpot مساوی 0 بود **یا** valueAtSpot هنگام حرکت سیاه(1) یا سفید(2) برار با عدد 1 یا 2 بود پس :

اگر هنگام حرکت سیاه(1) یا سفید(2) برار با عدد 1 یا 2 بود پس :

مقدار affectedDiscs برار شود با : مقدار affectedDiscs + مقدار couldBeAffected

خروج از حلقه.

در غیر اینصورت :

موقعیت هر مهره بصورت object به متغییر couldBeAffected بصورت زیر اضافه شود.

discLocation = {row: rowIterator; column: شماره ستون}

ادامه حلقه.

**.مثال :**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | column | | | | | | | |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| row | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | 0 |
| 2 | 0 | 2 | 2 | 2 | 2 | 0 | 0 | 0 |
| 3 | 0 | 1 | 2 | 2 | 2 | 2 | 1 | 0 |
| 4 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

در مثال بالا (با فرض اینکه نوبت حرکت مهره سیاه (1) است) با اجرا شدن حلقه نتیجه ی زیر حاصل میگردد :

couldBeAffected = [

{row: 4, column: 6}

{row: 3, column: 5}

{row: 2, column: 4}

{row: 1, column: 3}

]

با رسیدن حلقه به موقعیت ( row : 0 , column: 2 ) از آنجایی که حرکت با مهره سیاه (عدد 1) است و مقدار آن خانه هم برار با عدد 1 است ، پس حلقه با قرار دادن مقدار couldBeAffected در affectedDiscs به کار خود خاتمه می‍دهد.

**بررسی اُریب از پائین چپ به سمت بالا راست :**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 2 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

شماره سطر را در متغییری به نام rowIterator قرار میدهیم.

شماره سطر را در متغییری به نام columnIterator قرار میدهیم.

(شروع حلقه) : تا مادامی که مقدار rowIterator بیشتر از صفر و نیز columnIterator کمتر از تعداد ستون‌ها(0 تا 7) است دستورات زیر اجرا شود:

با هر بار اجرای حلقه ابتدا یک عدد از rowIterator کم و یک عدد به columnIterator اضافه می‌کند تا مقدار خانه بالا‌تر در سطر و ستون بعد از خود را خود را مورد بررسی قرار دهد. مانند تصویر زیر که روی خانه سبز کلیک شد و برنامه مقدار خانه قرمز را مورد ارزیابی قرار میدهد. در اینجا مقدار خانه سبز رنگ را valueAtSpot می نامیم.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | column | | | | | | | |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| row | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 2 | 2 | 2 | 0 | 0 | 0 |
| 3 | 0 | 0 | 2 | 2 | 1 | 1 | 0 | 0 |
| 4 | 0 | 0 | 2 | 2 | 2 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

اگر valueAtSpot مساوی 0 بود **یا** valueAtSpot هنگام حرکت سیاه(1) یا سفید(2) برار با عدد 1 یا 2 بود پس :

اگر هنگام حرکت سیاه(1) یا سفید(2) برار با عدد 1 یا 2 بود پس :

مقدار affectedDiscs برار شود با : مقدار affectedDiscs + مقدار couldBeAffected

خروج از حلقه.

در غیر اینصورت :

موقعیت هر مهره بصورت object به متغییر couldBeAffected بصورت زیر اضافه شود.

discLocation = {row: rowIterator; column: شماره ستون}

ادامه حلقه.

**.مثال :**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | column | | | | | | | |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| row | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 2 | 0 | 0 | 2 | 2 | 2 | 1 | 2 | 0 |
| 3 | 0 | 0 | 2 | 2 | 2 | 1 | 2 | 0 |
| 4 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 0 |
| 5 | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

در مثال بالا (با فرض اینکه نوبت حرکت مهره سیاه (1) است) با اجرا شدن حلقه نتیجه ی زیر حاصل میگردد :

couldBeAffected = [

{row: 5, column: 3}

{row: 4, column: 4}

]

با رسیدن حلقه به موقعیت ( row : 3 , column: 5 ) از آنجایی که حرکت با مهره سیاه (عدد 1) است و مقدار آن خانه هم برار با عدد 1 است ، پس حلقه با قرار دادن مقدار couldBeAffected در affectedDiscs به کار خود خاتمه می‍دهد.