

Problem 1954: Minimum Garden Perimeter to Collect Enough Apples

Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

In a garden represented as an infinite 2D grid, there is an apple tree planted at

every

integer coordinate. The apple tree planted at an integer coordinate

(i, j)

has

$|i| + |j|$

apples growing on it.

You will buy an axis-aligned

square plot

of land that is centered at

$(0, 0)$

.

Given an integer

neededApples

, return

the

minimum perimeter

of a plot such that

at least

neededApples

apples are

inside or on

the perimeter of that plot

.

The value of

$|x|$

is defined as:

x

if

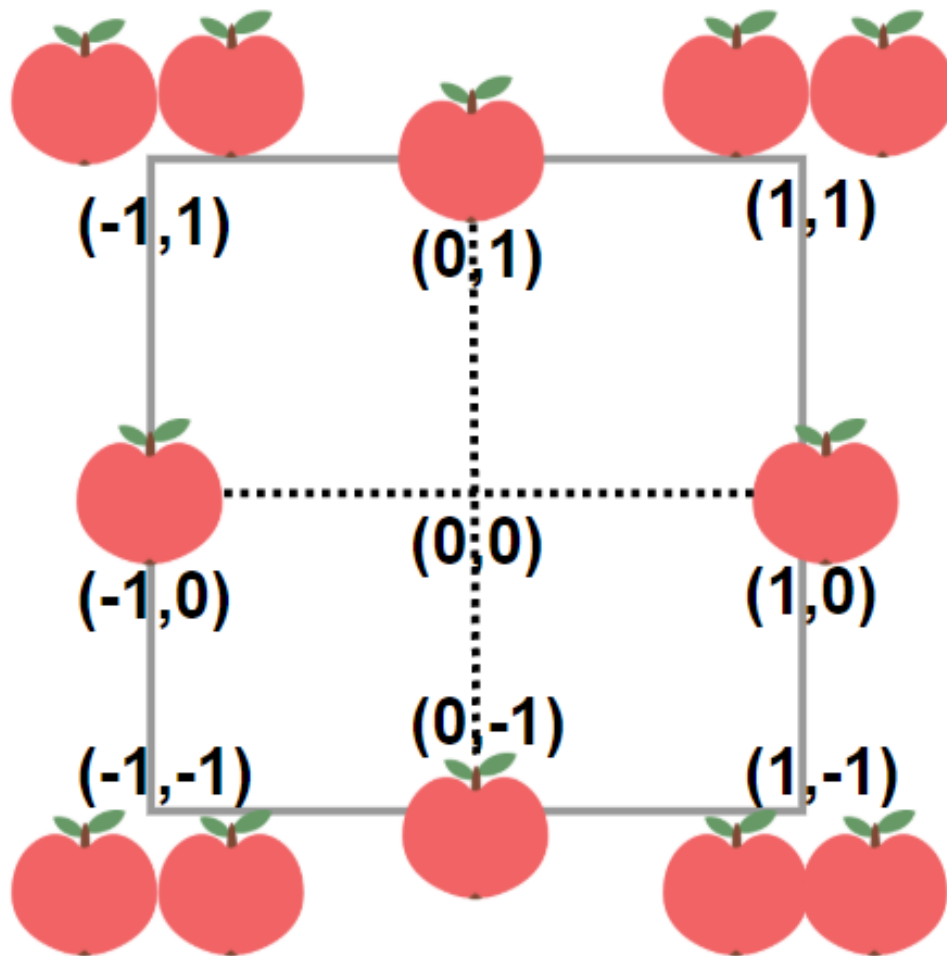
$x \geq 0$

$-x$

if

$x < 0$

Example 1:



Input:

neededApples = 1

Output:

8

Explanation:

A square plot of side length 1 does not contain any apples. However, a square plot of side length 2 has 12 apples inside (as depicted in the image above). The perimeter is $2 * 4 = 8$.

Example 2:

Input:

neededApples = 13

Output:

16

Example 3:

Input:

neededApples = 1000000000

Output:

5040

Constraints:

$1 \leq \text{neededApples} \leq 10$

15

Code Snippets

C++:

```
class Solution {
public:
    long long minimumPerimeter(long long neededApples) {

    }
};
```

Java:

```

class Solution {
public long minimumPerimeter(long neededApples) {

}

}

```

Python3:

```

class Solution:
def minimumPerimeter(self, neededApples: int) -> int:

```

Python:

```

class Solution(object):
def minimumPerimeter(self, neededApples):
"""
:type neededApples: int
:rtype: int
"""

```

JavaScript:

```

/**
 * @param {number} neededApples
 * @return {number}
 */
var minimumPerimeter = function(neededApples) {

};

```

TypeScript:

```

function minimumPerimeter(neededApples: number): number {

};

```

C#:

```

public class Solution {
public long MinimumPerimeter(long neededApples) {

}

}

```

C:

```
long long minimumPerimeter(long long neededApples) {  
  
}
```

Go:

```
func minimumPerimeter(neededApples int64) int64 {  
  
}
```

Kotlin:

```
class Solution {  
    fun minimumPerimeter(neededApples: Long): Long {  
  
    }  
}
```

Swift:

```
class Solution {  
    func minimumPerimeter(_ neededApples: Int) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn minimum_perimeter(needed_apples: i64) -> i64 {  
  
    }  
}
```

Ruby:

```
# @param {Integer} needed_apples  
# @return {Integer}  
def minimum_perimeter(needed_apples)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer $neededApples  
     * @return Integer  
     */  
    function minimumPerimeter($neededApples) {  
  
    }  
}
```

Dart:

```
class Solution {  
    int minimumPerimeter(int neededApples) {  
  
    }  
}
```

Scala:

```
object Solution {  
    def minimumPerimeter(neededApples: Long): Long = {  
  
    }  
}
```

Elixir:

```
defmodule Solution do  
    @spec minimum_perimeter(needed_apples :: integer) :: integer  
    def minimum_perimeter(needed_apples) do  
  
    end  
end
```

Erlang:

```
-spec minimum_perimeter(NeededApples :: integer()) -> integer().  
minimum_perimeter(NeededApples) ->  
.
```

Racket:

```
(define/contract (minimum-perimeter neededApples)
  (-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Minimum Garden Perimeter to Collect Enough Apples
 * Difficulty: Medium
 * Tags: tree, math, search
 *
 * Approach: DFS or BFS traversal
 * Time Complexity: O(n) where n is number of nodes
 * Space Complexity: O(h) for recursion stack where h is height
 */

class Solution {
public:
    long long minimumPerimeter(long long neededApples) {

    }
};
```

Java Solution:

```
/**
 * Problem: Minimum Garden Perimeter to Collect Enough Apples
 * Difficulty: Medium
 * Tags: tree, math, search
 *
 * Approach: DFS or BFS traversal
 * Time Complexity: O(n) where n is number of nodes
 * Space Complexity: O(h) for recursion stack where h is height
 */

class Solution {
    public long minimumPerimeter(long neededApples) {
```

```
}  
}
```

Python3 Solution:

```
"""  
Problem: Minimum Garden Perimeter to Collect Enough Apples  
Difficulty: Medium  
Tags: tree, math, search  
  
Approach: DFS or BFS traversal  
Time Complexity: O(n) where n is number of nodes  
Space Complexity: O(h) for recursion stack where h is height  
"""  
  
class Solution:  
    def minimumPerimeter(self, neededApples: int) -> int:  
        # TODO: Implement optimized solution  
        pass
```

Python Solution:

```
class Solution(object):  
    def minimumPerimeter(self, neededApples):  
        """  
        :type neededApples: int  
        :rtype: int  
        """
```

JavaScript Solution:

```
/**  
 * Problem: Minimum Garden Perimeter to Collect Enough Apples  
 * Difficulty: Medium  
 * Tags: tree, math, search  
 *  
 * Approach: DFS or BFS traversal  
 * Time Complexity: O(n) where n is number of nodes  
 * Space Complexity: O(h) for recursion stack where h is height  
 */
```

```

/**
 * @param {number} neededApples
 * @return {number}
 */
var minimumPerimeter = function(neededApples) {

};

```

TypeScript Solution:

```

/**
 * Problem: Minimum Garden Perimeter to Collect Enough Apples
 * Difficulty: Medium
 * Tags: tree, math, search
 *
 * Approach: DFS or BFS traversal
 * Time Complexity: O(n) where n is number of nodes
 * Space Complexity: O(h) for recursion stack where h is height
 */

function minimumPerimeter(neededApples: number): number {

};

```

C# Solution:

```

/*
 * Problem: Minimum Garden Perimeter to Collect Enough Apples
 * Difficulty: Medium
 * Tags: tree, math, search
 *
 * Approach: DFS or BFS traversal
 * Time Complexity: O(n) where n is number of nodes
 * Space Complexity: O(h) for recursion stack where h is height
 */

public class Solution {
    public long MinimumPerimeter(long neededApples) {

    }
}

```

```
}
```

C Solution:

```
/*
 * Problem: Minimum Garden Perimeter to Collect Enough Apples
 * Difficulty: Medium
 * Tags: tree, math, search
 *
 * Approach: DFS or BFS traversal
 * Time Complexity: O(n) where n is number of nodes
 * Space Complexity: O(h) for recursion stack where h is height
 */

long long minimumPerimeter(long long neededApples) {

}
```

Go Solution:

```
// Problem: Minimum Garden Perimeter to Collect Enough Apples
// Difficulty: Medium
// Tags: tree, math, search
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

func minimumPerimeter(neededApples int64) int64 {

}
```

Kotlin Solution:

```
class Solution {
    fun minimumPerimeter(neededApples: Long): Long {

    }
}
```

Swift Solution:

```

class Solution {
    func minimumPerimeter(_ neededApples: Int) -> Int {

    }
}

```

Rust Solution:

```

// Problem: Minimum Garden Perimeter to Collect Enough Apples
// Difficulty: Medium
// Tags: tree, math, search
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
    pub fn minimum_perimeter(needed_apples: i64) -> i64 {

    }
}

```

Ruby Solution:

```

# @param {Integer} needed_apples
# @return {Integer}
def minimum_perimeter(needed_apples)

end

```

PHP Solution:

```

class Solution {

    /**
     * @param Integer $neededApples
     * @return Integer
     */
    function minimumPerimeter($neededApples) {

    }

}

```

Dart Solution:

```
class Solution {  
  int minimumPerimeter(int neededApples) {  
  
  }  
}
```

Scala Solution:

```
object Solution {  
  def minimumPerimeter(neededApples: Long): Long = {  
  
  }  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec minimum_perimeter(needed_apples :: integer) :: integer  
  def minimum_perimeter(needed_apples) do  
  
  end  
end
```

Erlang Solution:

```
-spec minimum_perimeter(NeededApples :: integer()) -> integer().  
minimum_perimeter(NeededApples) ->  
.
```

Racket Solution:

```
(define/contract (minimum-perimeter neededApples)  
  (-> exact-integer? exact-integer?)  
)
```