Problem 2162: Minimum Cost to Set Cooking Time

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

A generic microwave supports cooking times for:

at least

1

second.

at most

99

minutes and

99

seconds.

To set the cooking time, you push

at most four digits

. The microwave normalizes what you push as four digits by

prepending zeroes

. It interprets the

first

two digits as the minutes and the

last

two digits as the seconds. It then

adds

them up as the cooking time. For example,

You push

9

5

4

(three digits). It is normalized as

0954

and interpreted as

9

minutes and

54

seconds.

You push

0

0

0

8

(four digits). It is interpreted as

0

minutes and

8

seconds.

You push

8

0

9

0

. It is interpreted as

80

minutes and

90

seconds.

You push

8

1

3

0

. It is interpreted as

81

minutes and

30

seconds.

You are given integers

startAt

,

moveCost

,

pushCost

, and

targetSeconds

.

Initially

, your finger is on the digit

startAt

. Moving the finger above

any specific digit

costs

moveCost

units of fatigue. Pushing the digit below the finger

once

costs

pushCost

units of fatigue.

There can be multiple ways to set the microwave to cook for

targetSeconds

seconds but you are interested in the way with the minimum cost.

Return

the

minimum cost

to set

targetSeconds

seconds of cooking time

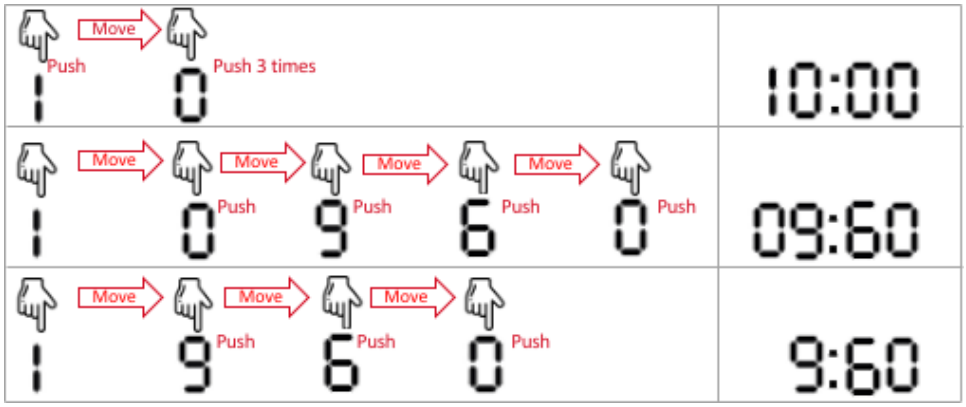
.

Remember that one minute consists of

60

seconds.

Example 1:



Input:

startAt = 1, moveCost = 2, pushCost = 1, targetSeconds = 600

Output:

6

Explanation:

The following are the possible ways to set the cooking time. - 1 0 0 0, interpreted as 10 minutes and 0 seconds. The finger is already on digit 1, pushes 1 (with cost 1), moves to 0 (with cost 2), pushes 0 (with cost 1), pushes 0 (with cost 1), and pushes 0 (with cost 1). The cost is: 1 + 2 + 1 + 1 + 1 = 6. This is the minimum cost. - 0 9 6 0, interpreted as 9 minutes and 60 seconds. That is also 600 seconds. The finger moves to 0 (with cost 2), pushes 0 (with cost 1), moves to 9 (with cost 2), pushes 9 (with cost 1), moves to 6 (with cost 2), pushes 6 (with cost 1), moves to 0 (with cost 2), and pushes 0 (with cost 1). The cost is: 2 + 1 + 2 + 1 + 2 + 1 + 2 + 1 = 12. - 9 6 0, normalized as 0960 and interpreted as 9 minutes and 60 seconds. The finger moves to 9 (with cost 2), pushes 9 (with cost 1), moves to 6 (with cost 2), pushes 6 (with cost 1), moves to 0 (with cost 2), and pushes 0 (with cost 1). The cost is: 2 + 1 + 2 + 1 +

2 + 1 = 9.

Example 2:

Input:

startAt = 0, moveCost = 1, pushCost = 2, targetSeconds = 76

Output:

6

Explanation:

The optimal way is to push two digits: 7 6, interpreted as 76 seconds. The finger moves to 7 (with cost 1), pushes 7 (with cost 2), moves to 6 (with cost 1), and pushes 6 (with cost 2). The total cost is: 1 + 2 + 1 + 2 = 6 Note other possible ways are 0076, 076, 0116, and 116, but none of them produces the minimum cost.

Constraints:

0 <= startAt <= 9

1 <= moveCost, pushCost <= 10

5

1 <= targetSeconds <= 6039

Code Snippets

C++:

```
class Solution {
public:
```

```
int minCostSetTime(int startAt, int moveCost, int pushCost, int
targetSeconds) {

}
};
```

Java:

```
class Solution {
public int minCostSetTime(int startAt, int moveCost, int pushCost, int
targetSeconds) {

}
}
```

Python3:

```
class Solution:
def minCostSetTime(self, startAt: int, moveCost: int, pushCost: int,
targetSeconds: int) -> int:
```

Python:

```
class Solution(object):
def minCostSetTime(self, startAt, moveCost, pushCost, targetSeconds):
"""
:type startAt: int
:type moveCost: int
:type pushCost: int
:type targetSeconds: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} startAt
* @param {number} moveCost
* @param {number} pushCost
* @param {number} targetSeconds
* @return {number}
*/
```

```
var minCostSetTime = function(startAt, moveCost, pushCost, targetSeconds) {

};
```

TypeScript:

```
function minCostSetTime(startAt: number, moveCost: number, pushCost: number,
targetSeconds: number): number {

};
```

C#:

```
public class Solution {
public int MinCostSetTime(int startAt, int moveCost, int pushCost, int
targetSeconds) {

}
}
```

C:

```
int minCostSetTime(int startAt, int moveCost, int pushCost, int
targetSeconds) {

}
```

Go:

```
func minCostSetTime(startAt int, moveCost int, pushCost int, targetSeconds
int) int {

}
```

Kotlin:

```
class Solution {
fun minCostSetTime(startAt: Int, moveCost: Int, pushCost: Int, targetSeconds:
Int): Int {

}
}
```

Swift:

```
class Solution {
func minCostSetTime(_ startAt: Int, _ moveCost: Int, _ pushCost: Int, _
targetSeconds: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_cost_set_time(start_at: i32, move_cost: i32, push_cost: i32,
target_seconds: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} start_at
# @param {Integer} move_cost
# @param {Integer} push_cost
# @param {Integer} target_seconds
# @return {Integer}
def min_cost_set_time(start_at, move_cost, push_cost, target_seconds)

end
```

PHP:

```
class Solution {

/**
* @param Integer $startAt
* @param Integer $moveCost
* @param Integer $pushCost
* @param Integer $targetSeconds
* @return Integer
*/
function minCostSetTime($startAt, $moveCost, $pushCost, $targetSeconds) {

}
```

```
}
```

Dart:

```
class Solution {
int minCostSetTime(int startAt, int moveCost, int pushCost, int
targetSeconds) {

}
}
```

Scala:

```
object Solution {
def minCostSetTime(startAt: Int, moveCost: Int, pushCost: Int, targetSeconds:
Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_cost_set_time(start_at :: integer, move_cost :: integer, push_cost
:: integer, target_seconds :: integer) :: integer
def min_cost_set_time(start_at, move_cost, push_cost, target_seconds) do

end
end
```

Erlang:

```
-spec min_cost_set_time(StartAt :: integer(), MoveCost :: integer(), PushCost
:: integer(), TargetSeconds :: integer()) -> integer().
min_cost_set_time(StartAt, MoveCost, PushCost, TargetSeconds) ->
.
```

Racket:

```
(define/contract (min-cost-set-time startAt moveCost pushCost targetSeconds)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer?)
```

```
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Cost to Set Cooking Time
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int minCostSetTime(int startAt, int moveCost, int pushCost, int
targetSeconds) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Cost to Set Cooking Time
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int minCostSetTime(int startAt, int moveCost, int pushCost, int
targetSeconds) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Minimum Cost to Set Cooking Time
Difficulty: Medium
Tags: math

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def minCostSetTime(self, startAt: int, moveCost: int, pushCost: int,
targetSeconds: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minCostSetTime(self, startAt, moveCost, pushCost, targetSeconds):
"""
:type startAt: int
:type moveCost: int
:type pushCost: int
:type targetSeconds: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Cost to Set Cooking Time
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
```

```
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} startAt
* @param {number} moveCost
* @param {number} pushCost
* @param {number} targetSeconds
* @return {number}
*/
var minCostSetTime = function(startAt, moveCost, pushCost, targetSeconds) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Cost to Set Cooking Time
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function minCostSetTime(startAt: number, moveCost: number, pushCost: number,
targetSeconds: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Cost to Set Cooking Time
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
```

```
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int MinCostSetTime(int startAt, int moveCost, int pushCost, int
targetSeconds) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Cost to Set Cooking Time
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

int minCostSetTime(int startAt, int moveCost, int pushCost, int
targetSeconds) {

}
```

Go Solution:

```
// Problem: Minimum Cost to Set Cooking Time
// Difficulty: Medium
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func minCostSetTime(startAt int, moveCost int, pushCost int, targetSeconds
int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minCostSetTime(startAt: Int, moveCost: Int, pushCost: Int, targetSeconds:
Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minCostSetTime(_ startAt: Int, _ moveCost: Int, _ pushCost: Int, _
targetSeconds: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Cost to Set Cooking Time
// Difficulty: Medium
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn min_cost_set_time(start_at: i32, move_cost: i32, push_cost: i32,
target_seconds: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} start_at
# @param {Integer} move_cost
# @param {Integer} push_cost
# @param {Integer} target_seconds
# @return {Integer}
```

```
def min_cost_set_time(start_at, move_cost, push_cost, target_seconds)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $startAt
* @param Integer $moveCost
* @param Integer $pushCost
* @param Integer $targetSeconds
* @return Integer
*/
function minCostSetTime($startAt, $moveCost, $pushCost, $targetSeconds) {

}
}
```

Dart Solution:

```
class Solution {
int minCostSetTime(int startAt, int moveCost, int pushCost, int
targetSeconds) {

}
}
```

Scala Solution:

```
object Solution {
def minCostSetTime(startAt: Int, moveCost: Int, pushCost: Int, targetSeconds:
Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_cost_set_time(start_at :: integer, move_cost :: integer, push_cost
:: integer, target_seconds :: integer) :: integer
def min_cost_set_time(start_at, move_cost, push_cost, target_seconds) do

end
end
```

Erlang Solution:

```
-spec min_cost_set_time(StartAt :: integer(), MoveCost :: integer(), PushCost
:: integer(), TargetSeconds :: integer()) -> integer().
min_cost_set_time(StartAt, MoveCost, PushCost, TargetSeconds) ->
.
```

Racket Solution:

```
(define/contract (min-cost-set-time startAt moveCost pushCost targetSeconds)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer?)
)
```