Problem 2227: Encrypt and Decrypt Strings

Problem Information

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

Problem Description

You are given a character array

keys

containing

unique

characters and a string array

values

containing strings of length 2. You are also given another string array

dictionary

that contains all permitted original strings after decryption. You should implement a data structure that can encrypt or decrypt a

0-indexed

string.

A string is

encrypted

with the following process:

For each character

c

in the string, we find the index

i

satisfying

keys[i] == c

in

keys

.

Replace

c

with

values[i]

in the string.

Note that in case a character of the string is

not present

in

keys

, the encryption process cannot be carried out, and an empty string

""

is returned.

A string is

decrypted

with the following process:

For each substring

s

of length 2 occurring at an even index in the string, we find an

i

such that

values[i] == s

. If there are multiple valid

i

, we choose

any

one of them. This means a string could have multiple possible strings it can decrypt to.

Replace

s

with

keys[i]

in the string.

Implement the

Encrypter

class:

Encrypter(char[] keys, String[] values, String[] dictionary)

Initializes the

Encrypter

class with

keys, values

, and

dictionary

.

String encrypt(String word1)

Encrypts

word1

with the encryption process described above and returns the encrypted string.

int decrypt(String word2)

Returns the number of possible strings

word2

could decrypt to that also appear in

dictionary

.

Example 1:

Input

["Encrypter", "encrypt", "decrypt"] [[['a', 'b', 'c', 'd'], ["ei", "zf", "ei", "am"], ["abcd", "acbd", "adbc", "badc", "dacb", "cadb", "cbda", "abad"]], ["abcd"], ["eizfeiam"]]

Output

[null, "eizfeiam", 2]

Explanation

Encrypter encrypter = new Encrypter([['a', 'b', 'c', 'd'], ["ei", "zf", "ei", "am"], ["abcd", "acbd", "adbc", "badc", "dacb", "cadb", "cbda", "abad"]); encrypter.encrypt("abcd"); // return "eizfeiam". // 'a' maps to "ei", 'b' maps to "zf", 'c' maps to "ei", and 'd' maps to "am". encrypter.decrypt("eizfeiam"); // return 2. // "ei" can map to 'a' or 'c', "zf" maps to 'b', and "am" maps to 'd'. // Thus, the possible strings after decryption are "abad", "cbad", "abcd", and "cbcd". // 2 of those strings, "abad" and "abcd", appear in dictionary, so the answer is 2.

Constraints:

1 <= keys.length == values.length <= 26

values[i].length == 2

1 <= dictionary.length <= 100

1 <= dictionary[i].length <= 100

All

keys[i]

and

dictionary[i]

are

unique

.

1 <= word1.length <= 2000

2 <= word2.length <= 200

All

word1[i]

appear in

keys

.

word2.length

is even.

keys

,

values[i]

,

dictionary[i]

,

word1

, and

word2

only contain lowercase English letters.

At most

200

calls will be made to

encrypt

and

decrypt

in total

.

Code Snippets

C++:

```
class Encrypter {
public:
Encrypter(vector<char>& keys, vector<string>& values, vector<string>&
dictionary) {

}
```

```
string encrypt(string word1) {

}

int decrypt(string word2) {

}
};

/**
* Your Encrypter object will be instantiated and called as such:
* Encrypter* obj = new Encrypter(keys, values, dictionary);
* string param_1 = obj->encrypt(word1);
* int param_2 = obj->decrypt(word2);
*/
```

Java:

```
class Encrypter {

public Encrypter(char[] keys, String[] values, String[] dictionary) {

}

public String encrypt(String word1) {

}

public int decrypt(String word2) {

}
}

/**
* Your Encrypter object will be instantiated and called as such:
* Encrypter obj = new Encrypter(keys, values, dictionary);
* String param_1 = obj.encrypt(word1);
* int param_2 = obj.decrypt(word2);
*/
```

Python3:

```
class Encrypter:

def __init__(self, keys: List[str], values: List[str], dictionary:
List[str]):


def encrypt(self, word1: str) -> str:


def decrypt(self, word2: str) -> int:



# Your Encrypter object will be instantiated and called as such:
# obj = Encrypter(keys, values, dictionary)
# param_1 = obj.encrypt(word1)
# param_2 = obj.decrypt(word2)
```

Python:

```
class Encrypter(object):

def __init__(self, keys, values, dictionary):
"""
:type keys: List[str]
:type values: List[str]
:type dictionary: List[str]
"""


def encrypt(self, word1):
"""
:type word1: str
:rtype: str
"""


def decrypt(self, word2):
"""
:type word2: str
:rtype: int
"""
```

```
# Your Encrypter object will be instantiated and called as such:
# obj = Encrypter(keys, values, dictionary)
# param_1 = obj.encrypt(word1)
# param_2 = obj.decrypt(word2)
```

JavaScript:

```
/**
* @param {character[]} keys
* @param {string[]} values
* @param {string[]} dictionary
*/
var Encrypter = function(keys, values, dictionary) {

};

/**
* @param {string} word1
* @return {string}
*/
Encrypter.prototype.encrypt = function(word1) {

};

/**
* @param {string} word2
* @return {number}
*/
Encrypter.prototype.decrypt = function(word2) {

};

/**
* Your Encrypter object will be instantiated and called as such:
* var obj = new Encrypter(keys, values, dictionary)
* var param_1 = obj.encrypt(word1)
* var param_2 = obj.decrypt(word2)
*/
```

TypeScript:

```
class Encrypter {
constructor(keys: string[], values: string[], dictionary: string[]) {

}

encrypt(word1: string): string {

}

decrypt(word2: string): number {

}
}

/**
* Your Encrypter object will be instantiated and called as such:
* var obj = new Encrypter(keys, values, dictionary)
* var param_1 = obj.encrypt(word1)
* var param_2 = obj.decrypt(word2)
*/
```

C#:

```
public class Encrypter {

public Encrypter(char[] keys, string[] values, string[] dictionary) {

}

public string Encrypt(string word1) {

}

public int Decrypt(string word2) {

}
}

/**
* Your Encrypter object will be instantiated and called as such:
* Encrypter obj = new Encrypter(keys, values, dictionary);
```

```
* string param_1 = obj.Encrypt(word1);
* int param_2 = obj.Decrypt(word2);
*/
```

C:

```
typedef struct {

} Encrypter;


Encrypter* encrypterCreate(char* keys, int keysSize, char** values, int
valuesSize, char** dictionary, int dictionarySize) {

}

char* encrypterEncrypt(Encrypter* obj, char* word1) {

}

int encrypterDecrypt(Encrypter* obj, char* word2) {

}

void encrypterFree(Encrypter* obj) {

}

/**
* Your Encrypter struct will be instantiated and called as such:
* Encrypter* obj = encrypterCreate(keys, keysSize, values, valuesSize,
dictionary, dictionarySize);
* char* param_1 = encrypterEncrypt(obj, word1);

* int param_2 = encrypterDecrypt(obj, word2);

* encrypterFree(obj);
*/
```

Go:

```
type Encrypter struct {

}


func Constructor(keys []byte, values []string, dictionary []string) Encrypter
{

}


func (this *Encrypter) Encrypt(word1 string) string {

}


func (this *Encrypter) Decrypt(word2 string) int {

}


/**
* Your Encrypter object will be instantiated and called as such:
* obj := Constructor(keys, values, dictionary);
* param_1 := obj.Encrypt(word1);
* param_2 := obj.Decrypt(word2);
*/
```

Kotlin:

```
class Encrypter(keys: CharArray, values: Array<String>, dictionary:
Array<String>) {

fun encrypt(word1: String): String {

}

fun decrypt(word2: String): Int {

}
```

```
}

/**
* Your Encrypter object will be instantiated and called as such:
* var obj = Encrypter(keys, values, dictionary)
* var param_1 = obj.encrypt(word1)
* var param_2 = obj.decrypt(word2)
*/
```

Swift:

```
class Encrypter {

init(_ keys: [Character], _ values: [String], _ dictionary: [String]) {

}

func encrypt(_ word1: String) -> String {

}

func decrypt(_ word2: String) -> Int {

}
}

/**
* Your Encrypter object will be instantiated and called as such:
* let obj = Encrypter(keys, values, dictionary)
* let ret_1: String = obj.encrypt(word1)
* let ret_2: Int = obj.decrypt(word2)
*/
```

Rust:

```
struct Encrypter {

}
```

```
/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl Encrypter {

fn new(keys: Vec<char>, values: Vec<String>, dictionary: Vec<String>) -> Self
{

}

fn encrypt(&self, word1: String) -> String {

}

fn decrypt(&self, word2: String) -> i32 {

}
}

/**
* Your Encrypter object will be instantiated and called as such:
* let obj = Encrypter::new(keys, values, dictionary);
* let ret_1: String = obj.encrypt(word1);
* let ret_2: i32 = obj.decrypt(word2);
*/
```

Ruby:

```
class Encrypter

=begin
:type keys: Character[]
:type values: String[]
:type dictionary: String[]
=end
def initialize(keys, values, dictionary)

end


=begin
```

```
:type word1: String
:rtype: String
=end
def encrypt(word1)

end


=begin
:type word2: String
:rtype: Integer
=end
def decrypt(word2)

end



end

# Your Encrypter object will be instantiated and called as such:
# obj = Encrypter.new(keys, values, dictionary)
# param_1 = obj.encrypt(word1)
# param_2 = obj.decrypt(word2)
```

PHP:

```
class Encrypter {
/**
* @param String[] $keys
* @param String[] $values
* @param String[] $dictionary
*/
function __construct($keys, $values, $dictionary) {

}

/**
* @param String $word1
* @return String
*/
function encrypt($word1) {
```

```
}

/**
* @param String $word2
* @return Integer
*/
function decrypt($word2) {

}
}

/**
* Your Encrypter object will be instantiated and called as such:
* $obj = Encrypter($keys, $values, $dictionary);
* $ret_1 = $obj->encrypt($word1);
* $ret_2 = $obj->decrypt($word2);
*/
```

Dart:

```
class Encrypter {

Encrypter(List<String> keys, List<String> values, List<String> dictionary) {

}

String encrypt(String word1) {

}

int decrypt(String word2) {

}
}

/**
* Your Encrypter object will be instantiated and called as such:
* Encrypter obj = Encrypter(keys, values, dictionary);
* String param1 = obj.encrypt(word1);
* int param2 = obj.decrypt(word2);
*/
```

Scala:

```
class Encrypter(_keys: Array[Char], _values: Array[String], _dictionary:
Array[String]) {

def encrypt(word1: String): String = {

}

def decrypt(word2: String): Int = {

}

}

/**
* Your Encrypter object will be instantiated and called as such:
* val obj = new Encrypter(keys, values, dictionary)
* val param_1 = obj.encrypt(word1)
* val param_2 = obj.decrypt(word2)
*/
```

Elixir:

```
defmodule Encrypter do
@spec init_(keys :: [char], values :: [String.t], dictionary :: [String.t])
:: any
def init_(keys, values, dictionary) do

end

@spec encrypt(word1 :: String.t) :: String.t
def encrypt(word1) do

end

@spec decrypt(word2 :: String.t) :: integer
def decrypt(word2) do

end
end
```

```
# Your functions will be called as such:
# Encrypter.init_(keys, values, dictionary)
# param_1 = Encrypter.encrypt(word1)
# param_2 = Encrypter.decrypt(word2)

# Encrypter.init_ will be called before every test case, in which you can do some necessary initializations.
```

Erlang:

```
-spec encrypter_init_(Keys :: [char()], Values :: [unicode:unicode_binary()], Dictionary :: [unicode:unicode_binary()]) -> any().
encrypter_init_(Keys, Values, Dictionary) ->
.

-spec encrypter_encrypt(Word1 :: unicode:unicode_binary()) -> unicode:unicode_binary().
encrypter_encrypt(Word1) ->
.

-spec encrypter_decrypt(Word2 :: unicode:unicode_binary()) -> integer().
encrypter_decrypt(Word2) ->
.


%% Your functions will be called as such:
%% encrypter_init_(Keys, Values, Dictionary),
%% Param_1 = encrypter_encrypt(Word1),
%% Param_2 = encrypter_decrypt(Word2),

%% encrypter_init_ will be called before every test case, in which you can do some necessary initializations.
```

Racket:

```
(define encrypter%
(class object%
(super-new)

; keys : (listof char?)
; values : (listof string?)
; dictionary : (listof string?)
```

```
(init-field
keys
values
dictionary)

; encrypt : string? -> string?
(define/public (encrypt word1)
)
; decrypt : string? -> exact-integer?
(define/public (decrypt word2)
)))

;; Your encrypter% object will be instantiated and called as such:
;; (define obj (new encrypter% [keys keys] [values values] [dictionary
dictionary]))
;; (define param_1 (send obj encrypt word1))
;; (define param_2 (send obj decrypt word2))
```

Solutions

C++ Solution:

```
/*
* Problem: Encrypt and Decrypt Strings
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Encrypter {
public:
Encrypter(vector<char>& keys, vector<string>& values, vector<string>&
dictionary) {

}

string encrypt(string word1) {
```

```
}

int decrypt(string word2) {

}
};

/**
* Your Encrypter object will be instantiated and called as such:
* Encrypter* obj = new Encrypter(keys, values, dictionary);
* string param_1 = obj->encrypt(word1);
* int param_2 = obj->decrypt(word2);
*/
```

Java Solution:

```
/**
* Problem: Encrypt and Decrypt Strings
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Encrypter {

public Encrypter(char[] keys, String[] values, String[] dictionary) {

}

public String encrypt(String word1) {

}

public int decrypt(String word2) {

}
}
```

```
/**
* Your Encrypter object will be instantiated and called as such:
* Encrypter obj = new Encrypter(keys, values, dictionary);
* String param_1 = obj.encrypt(word1);
* int param_2 = obj.decrypt(word2);
*/
```

Python3 Solution:

```
"""
Problem: Encrypt and Decrypt Strings
Difficulty: Hard
Tags: array, string, tree, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Encrypter:

def __init__(self, keys: List[str], values: List[str], dictionary:
List[str]):



def encrypt(self, word1: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Encrypter(object):

def __init__(self, keys, values, dictionary):
"""
:type keys: List[str]
:type values: List[str]
:type dictionary: List[str]
"""
```

```
def encrypt(self, word1):
"""
:type word1: str
:rtype: str
"""


def decrypt(self, word2):
"""
:type word2: str
:rtype: int
"""



# Your Encrypter object will be instantiated and called as such:
# obj = Encrypter(keys, values, dictionary)
# param_1 = obj.encrypt(word1)
# param_2 = obj.decrypt(word2)
```

JavaScript Solution:

```
/**
* Problem: Encrypt and Decrypt Strings
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {character[]} keys
* @param {string[]} values
* @param {string[]} dictionary
*/
var Encrypter = function(keys, values, dictionary) {

};
```

```
/**
* @param {string} word1
* @return {string}
*/
Encrypter.prototype.encrypt = function(word1) {

};

/**
* @param {string} word2
* @return {number}
*/
Encrypter.prototype.decrypt = function(word2) {

};

/**
* Your Encrypter object will be instantiated and called as such:
* var obj = new Encrypter(keys, values, dictionary)
* var param_1 = obj.encrypt(word1)
* var param_2 = obj.decrypt(word2)
*/
```

TypeScript Solution:

```
/**
* Problem: Encrypt and Decrypt Strings
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Encrypter {
constructor(keys: string[], values: string[], dictionary: string[]) {

}
```

```
encrypt(word1: string): string {

}

decrypt(word2: string): number {

}
}

/**
* Your Encrypter object will be instantiated and called as such:
* var obj = new Encrypter(keys, values, dictionary)
* var param_1 = obj.encrypt(word1)
* var param_2 = obj.decrypt(word2)
*/
```

C# Solution:

```
/*
* Problem: Encrypt and Decrypt Strings
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Encrypter {

public Encrypter(char[] keys, string[] values, string[] dictionary) {

}

public string Encrypt(string word1) {

}

public int Decrypt(string word2) {

}
```

```
}

/**
* Your Encrypter object will be instantiated and called as such:
* Encrypter obj = new Encrypter(keys, values, dictionary);
* string param_1 = obj.Encrypt(word1);
* int param_2 = obj.Decrypt(word2);
*/
```

C Solution:

```
/*
* Problem: Encrypt and Decrypt Strings
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/




typedef struct {

} Encrypter;


Encrypter* encrypterCreate(char* keys, int keysSize, char** values, int
valuesSize, char** dictionary, int dictionarySize) {

}

char* encrypterEncrypt(Encrypter* obj, char* word1) {

}

int encrypterDecrypt(Encrypter* obj, char* word2) {

}
```

```
void encrypterFree(Encrypter* obj) {

}

/**
* Your Encrypter struct will be instantiated and called as such:
* Encrypter* obj = encrypterCreate(keys, keysSize, values, valuesSize,
dictionary, dictionarySize);
* char* param_1 = encrypterEncrypt(obj, word1);

* int param_2 = encrypterDecrypt(obj, word2);

* encrypterFree(obj);
*/
```

Go Solution:

```
// Problem: Encrypt and Decrypt Strings
// Difficulty: Hard
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

type Encrypter struct {

}


func Constructor(keys []byte, values []string, dictionary []string) Encrypter
{

}


func (this *Encrypter) Encrypt(word1 string) string {

}
```

```
func (this *Encrypter) Decrypt(word2 string) int {

}


/**
* Your Encrypter object will be instantiated and called as such:
* obj := Constructor(keys, values, dictionary);
* param_1 := obj.Encrypt(word1);
* param_2 := obj.Decrypt(word2);
*/
```

Kotlin Solution:

```
class Encrypter(keys: CharArray, values: Array<String>, dictionary:
Array<String>) {

fun encrypt(word1: String): String {

}

fun decrypt(word2: String): Int {

}

}

/**
* Your Encrypter object will be instantiated and called as such:
* var obj = Encrypter(keys, values, dictionary)
* var param_1 = obj.encrypt(word1)
* var param_2 = obj.decrypt(word2)
*/
```

Swift Solution:

```
class Encrypter {

init(_ keys: [Character], _ values: [String], _ dictionary: [String]) {
```

```
}

func encrypt(_ word1: String) -> String {

}

func decrypt(_ word2: String) -> Int {

}
}

/**
* Your Encrypter object will be instantiated and called as such:
* let obj = Encrypter(keys, values, dictionary)
* let ret_1: String = obj.encrypt(word1)
* let ret_2: Int = obj.decrypt(word2)
*/
```

Rust Solution:

```
// Problem: Encrypt and Decrypt Strings
// Difficulty: Hard
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

struct Encrypter {

}



/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl Encrypter {

fn new(keys: Vec<char>, values: Vec<String>, dictionary: Vec<String>) -> Self
```

```
{

}

fn encrypt(&self, word1: String) -> String {

}

fn decrypt(&self, word2: String) -> i32 {

}
}

/**
* Your Encrypter object will be instantiated and called as such:
* let obj = Encrypter::new(keys, values, dictionary);
* let ret_1: String = obj.encrypt(word1);
* let ret_2: i32 = obj.decrypt(word2);
*/
```

Ruby Solution:

```
class Encrypter

=begin
:type keys: Character[]
:type values: String[]
:type dictionary: String[]
=end
def initialize(keys, values, dictionary)

end


=begin
:type word1: String
:rtype: String
=end
def encrypt(word1)

end
```

```
=begin
:type word2: String
:rtype: Integer
=end
def decrypt(word2)

end



end

# Your Encrypter object will be instantiated and called as such:
# obj = Encrypter.new(keys, values, dictionary)
# param_1 = obj.encrypt(word1)
# param_2 = obj.decrypt(word2)
```

PHP Solution:

```
class Encrypter {
/**
* @param String[] $keys
* @param String[] $values
* @param String[] $dictionary
*/
function __construct($keys, $values, $dictionary) {

}

/**
* @param String $word1
* @return String
*/
function encrypt($word1) {

}

/**
* @param String $word2
* @return Integer
*/
```

```
function decrypt($word2) {

}
}

/**
* Your Encrypter object will be instantiated and called as such:
* $obj = Encrypter($keys, $values, $dictionary);
* $ret_1 = $obj->encrypt($word1);
* $ret_2 = $obj->decrypt($word2);
*/
```

Dart Solution:

```
class Encrypter {

Encrypter(List<String> keys, List<String> values, List<String> dictionary) {

}

String encrypt(String word1) {

}

int decrypt(String word2) {

}
}

/**
* Your Encrypter object will be instantiated and called as such:
* Encrypter obj = Encrypter(keys, values, dictionary);
* String param1 = obj.encrypt(word1);
* int param2 = obj.decrypt(word2);
*/
```

Scala Solution:

```
class Encrypter(_keys: Array[Char], _values: Array[String], _dictionary:
Array[String]) {
```

```
def encrypt(word1: String): String = {

}

def decrypt(word2: String): Int = {

}

}

/**
* Your Encrypter object will be instantiated and called as such:
* val obj = new Encrypter(keys, values, dictionary)
* val param_1 = obj.encrypt(word1)
* val param_2 = obj.decrypt(word2)
*/
```

Elixir Solution:

```
defmodule Encrypter do
@spec init_(keys :: [char], values :: [String.t], dictionary :: [String.t])
:: any
def init_(keys, values, dictionary) do

end

@spec encrypt(word1 :: String.t) :: String.t
def encrypt(word1) do

end

@spec decrypt(word2 :: String.t) :: integer
def decrypt(word2) do

end
end

# Your functions will be called as such:
# Encrypter.init_(keys, values, dictionary)
# param_1 = Encrypter.encrypt(word1)
# param_2 = Encrypter.decrypt(word2)
```

```
# Encrypter.init_ will be called before every test case, in which you can do
some necessary initializations.
```

Erlang Solution:

```
-spec encrypter_init_(Keys :: [char()], Values :: [unicode:unicode_binary()],
Dictionary :: [unicode:unicode_binary()]) -> any().
encrypter_init_(Keys, Values, Dictionary) ->
.

-spec encrypter_encrypt(Word1 :: unicode:unicode_binary()) ->
unicode:unicode_binary().
encrypter_encrypt(Word1) ->
.

-spec encrypter_decrypt(Word2 :: unicode:unicode_binary()) -> integer().
encrypter_decrypt(Word2) ->
.


%% Your functions will be called as such:
%% encrypter_init_(Keys, Values, Dictionary),
%% Param_1 = encrypter_encrypt(Word1),
%% Param_2 = encrypter_decrypt(Word2),

%% encrypter_init_ will be called before every test case, in which you can do
some necessary initializations.
```

Racket Solution:

```
(define encrypter%
(class object%
(super-new)

; keys : (listof char?)
; values : (listof string?)
; dictionary : (listof string?)
(init-field
keys
values
```

```
dictionary)

; encrypt : string? -> string?
(define/public (encrypt word1)
)
; decrypt : string? -> exact-integer?
(define/public (decrypt word2)
)))

;; Your encrypter% object will be instantiated and called as such:
;; (define obj (new encrypter% [keys keys] [values values] [dictionary
dictionary]))
;; (define param_1 (send obj encrypt word1))
;; (define param_2 (send obj decrypt word2))
```