Problem 3007: Maximum Number That Sum of the Prices Is Less Than or Equal to K

Problem Information

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

Problem Description

You are given an integer

k

and an integer

x

. The price of a number

num

is calculated by the count of

set bits

at positions

x

,

2x

,

3x

, etc., in its binary representation, starting from the least significant bit. The following table contains examples of how price is calculated.

x

num

Binary Representation

Price

1

13

0

0

0

0

0

1

1

0

1

3

2

13

0

0

0

0

0

1

1

0

1

1

2

233

0

1

1

1

0

1

0

0

1

3

3

13

0

00

0

01

1

01

1

3

362

1

01

1

01

0

10

2

The

accumulated price

of

num

is the

total

price of numbers from

1

to

num

.

num

is considered

cheap

if its accumulated price is less than or equal to

k

.

Return the

greatest

cheap number.

Example 1:

Input:

k = 9, x = 1

Output:

6

Explanation:

As shown in the table below,

6

is the greatest cheap number.

x

num

Binary Representation

Price

Accumulated Price

1

1

0

0

1

1

1

1

2

0

1

0

1

2

1

3

0

1

1

2

4

1

4

1

0

0

1

5

1

5

1

0

1

2

7

1

6

1

1

0

2

9

1

7

1

1

1

3

12

Example 2:

Input:

k = 7, x = 2

Output:

9

Explanation:

As shown in the table below,

9

is the greatest cheap number.

x

num

Binary Representation

Price

Accumulated Price

2

1

0

0

0

1

0

0

2

2

0

0

1

0

1

1

2

3

0

0

1

1

1

2

2

4

0

1

0

0

0

2

2

5

0

1

0

1

0

2

2

6

0

1

1

0

1

3

2

7

0

1

1

1

1

4

2

8

1

0

0

0

1

5

2

9

1

0

0

1

1

6

2

10

1

0

1

0

2

8

Constraints:

1 <= k <= 10

15

1 <= x <= 8

Code Snippets

C++:

```
class Solution {
public:
long long findMaximumNumber(long long k, int x) {

}
};
```

Java:

```
class Solution {
public long findMaximumNumber(long k, int x) {

}
}
```

Python3:

```
class Solution:
def findMaximumNumber(self, k: int, x: int) -> int:
```

Python:

```
class Solution(object):
def findMaximumNumber(self, k, x):
"""
:type k: int
:type x: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} k
* @param {number} x
* @return {number}
*/
var findMaximumNumber = function(k, x) {

};
```

TypeScript:

```
function findMaximumNumber(k: number, x: number): number {

};
```

C#:

```
public class Solution {
public long FindMaximumNumber(long k, int x) {

}
}
```

C:

```
long long findMaximumNumber(long long k, int x) {

}
```

Go:

```
func findMaximumNumber(k int64, x int) int64 {

}
```

Kotlin:

```
class Solution {
fun findMaximumNumber(k: Long, x: Int): Long {

}
}
```

Swift:

```
class Solution {
func findMaximumNumber(_ k: Int, _ x: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_maximum_number(k: i64, x: i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer} k
# @param {Integer} x
# @return {Integer}
def find_maximum_number(k, x)

end
```

PHP:

```
class Solution {

/**
* @param Integer $k
* @param Integer $x
* @return Integer
*/
function findMaximumNumber($k, $x) {

}
}
```

Dart:

```
class Solution {
int findMaximumNumber(int k, int x) {

}
}
```

Scala:

```
object Solution {
def findMaximumNumber(k: Long, x: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_maximum_number(k :: integer, x :: integer) :: integer
def find_maximum_number(k, x) do

end
end
```

Erlang:

```
-spec find_maximum_number(K :: integer(), X :: integer()) -> integer().
find_maximum_number(K, X) ->
.
```

Racket:

```
(define/contract (find-maximum-number k x)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number That Sum of the Prices Is Less Than or Equal to K
* Difficulty: Medium
* Tags: dp, math, search, heap
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
long long findMaximumNumber(long long k, int x) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number That Sum of the Prices Is Less Than or Equal to K
* Difficulty: Medium
* Tags: dp, math, search, heap
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public long findMaximumNumber(long k, int x) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number That Sum of the Prices Is Less Than or Equal to K
Difficulty: Medium
Tags: dp, math, search, heap

Approach: Dynamic programming with memoization or tabulation
Time Complexity: O(n * m) where n and m are problem dimensions
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def findMaximumNumber(self, k: int, x: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findMaximumNumber(self, k, x):
"""
:type k: int
:type x: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Number That Sum of the Prices Is Less Than or Equal to K
* Difficulty: Medium
* Tags: dp, math, search, heap
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

```
/**
* @param {number} k
* @param {number} x
* @return {number}
*/
var findMaximumNumber = function(k, x) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number That Sum of the Prices Is Less Than or Equal to K
* Difficulty: Medium
* Tags: dp, math, search, heap
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

function findMaximumNumber(k: number, x: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Number That Sum of the Prices Is Less Than or Equal to K
* Difficulty: Medium
* Tags: dp, math, search, heap
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public long FindMaximumNumber(long k, int x) {

}

}
```

C Solution:

```
/*
* Problem: Maximum Number That Sum of the Prices Is Less Than or Equal to K
* Difficulty: Medium
* Tags: dp, math, search, heap
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

long long findMaximumNumber(long long k, int x) {

}
```

Go Solution:

```
// Problem: Maximum Number That Sum of the Prices Is Less Than or Equal to K
// Difficulty: Medium
// Tags: dp, math, search, heap
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

func findMaximumNumber(k int64, x int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun findMaximumNumber(k: Long, x: Int): Long {

}
}
```

Swift Solution:

```
class Solution {
func findMaximumNumber(_ k: Int, _ x: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Number That Sum of the Prices Is Less Than or Equal to K
// Difficulty: Medium
// Tags: dp, math, search, heap
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn find_maximum_number(k: i64, x: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer} k
# @param {Integer} x
# @return {Integer}
def find_maximum_number(k, x)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $k
* @param Integer $x
* @return Integer
*/
function findMaximumNumber($k, $x) {

}
}
```

Dart Solution:

```
class Solution {
int findMaximumNumber(int k, int x) {

}
}
```

Scala Solution:

```
object Solution {
def findMaximumNumber(k: Long, x: Int): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_maximum_number(k :: integer, x :: integer) :: integer
def find_maximum_number(k, x) do

end
end
```

Erlang Solution:

```
-spec find_maximum_number(K :: integer(), X :: integer()) -> integer().
find_maximum_number(K, X) ->
.
```

Racket Solution:

```
(define/contract (find-maximum-number k x)
(-> exact-integer? exact-integer? exact-integer?)
)
```