Problem 1494: Parallel Courses II

Problem Information

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

Problem Description

You are given an integer

n

, which indicates that there are

n

courses labeled from

1

to

n

. You are also given an array

relations

where

relations[i] = [prevCourse

i

, nextCourse

i

]

, representing a prerequisite relationship between course

prevCourse

i

and course

nextCourse

i

: course

prevCourse

i

has to be taken before course

nextCourse

i

. Also, you are given the integer

k

.

In one semester, you can take

at most

k

courses as long as you have taken all the prerequisites in the

previous

semesters for the courses you are taking.

Return

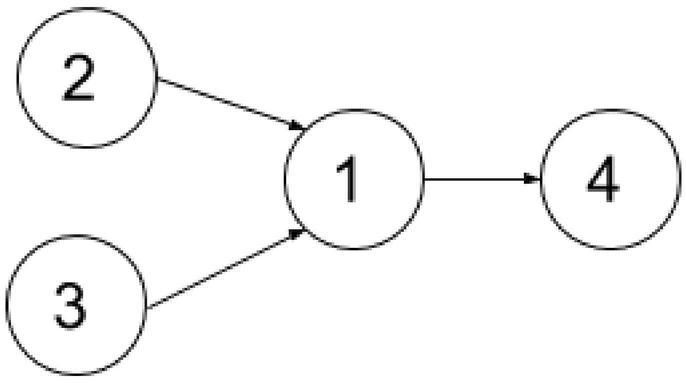
the

minimum

number of semesters needed to take all courses

. The testcases will be generated such that it is possible to take every course.

Example 1:



Input:

n = 4, relations = [[2,1],[3,1],[1,4]], k = 2

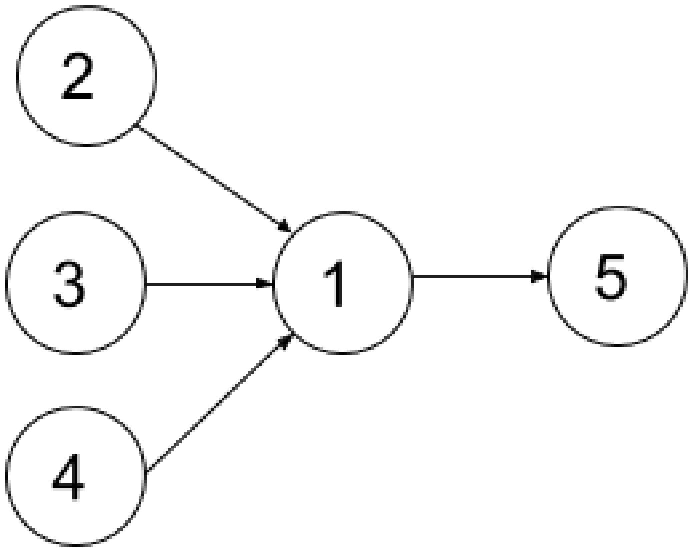
Output:

3

Explanation:

The figure above represents the given graph. In the first semester, you can take courses 2 and 3. In the second semester, you can take course 1. In the third semester, you can take course 4.

Example 2:



Input:

n = 5, relations = [[2,1],[3,1],[4,1],[1,5]], k = 2

Output:

4

Explanation:

The figure above represents the given graph. In the first semester, you can only take courses 2 and 3 since you cannot take more than two per semester. In the second semester, you can take course 4. In the third semester, you can take course 1. In the fourth semester, you can take course 5.

Constraints:

1 <= n <= 15

1 <= k <= n

0 <= relations.length <= n * (n-1) / 2

relations[i].length == 2

1 <= prevCourse

i

, nextCourse

i

<= n

prevCourse

i

!= nextCourse

i

All the pairs

[prevCourse

i

, nextCourse

i

]

are

unique

.

The given graph is a directed acyclic graph.

Code Snippets

C++:

```
class Solution {
public:
int minNumberOfSemesters(int n, vector<vector<int>>& relations, int k) {

}
};
```

Java:

```
class Solution {
public int minNumberOfSemesters(int n, int[][] relations, int k) {

}
}
```

Python3:

```
class Solution:
def minNumberOfSemesters(self, n: int, relations: List[List[int]], k: int) ->
int:
```

Python:

```
class Solution(object):
def minNumberOfSemesters(self, n, relations, k):
"""
:type n: int
:type relations: List[List[int]]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} relations
* @param {number} k
* @return {number}
*/
var minNumberOfSemesters = function(n, relations, k) {

};
```

TypeScript:

```
function minNumberOfSemesters(n: number, relations: number[][], k: number):
number {

};
```

C#:

```
public class Solution {
public int MinNumberOfSemesters(int n, int[][] relations, int k) {

}
}
```

C:

```
int minNumberOfSemesters(int n, int** relations, int relationsSize, int*
relationsColSize, int k) {

}
```

Go:

```
func minNumberOfSemesters(n int, relations [][]int, k int) int {

}
```

Kotlin:

```
class Solution {
fun minNumberOfSemesters(n: Int, relations: Array<IntArray>, k: Int): Int {
}
}
```

Swift:

```
class Solution {
func minNumberOfSemesters(_ n: Int, _ relations: [[Int]], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_number_of_semesters(n: i32, relations: Vec<Vec<i32>>, k: i32) ->
i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} relations
# @param {Integer} k
# @return {Integer}
def min_number_of_semesters(n, relations, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $relations
* @param Integer $k
* @return Integer
*/
function minNumberOfSemesters($n, $relations, $k) {
}
}
```

Dart:

```
class Solution {
int minNumberOfSemesters(int n, List<List<int>> relations, int k) {

}
}
```

Scala:

```
object Solution {
def minNumberOfSemesters(n: Int, relations: Array[Array[Int]], k: Int): Int =
{

}
}
```

Elixir:

```
defmodule Solution do
@spec min_number_of_semesters(n :: integer, relations :: [[integer]], k ::
integer) :: integer
def min_number_of_semesters(n, relations, k) do

end
end
```

Erlang:

```
-spec min_number_of_semesters(N :: integer(), Relations :: [[integer()]], K
:: integer()) -> integer().
min_number_of_semesters(N, Relations, K) ->
.
```

Racket:

```
(define/contract (min-number-of-semesters n relations k)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Parallel Courses II
* Difficulty: Hard
* Tags: array, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int minNumberOfSemesters(int n, vector<vector<int>>& relations, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Parallel Courses II
* Difficulty: Hard
* Tags: array, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int minNumberOfSemesters(int n, int[][] relations, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Parallel Courses II
Difficulty: Hard
Tags: array, graph, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def minNumberOfSemesters(self, n: int, relations: List[List[int]], k: int) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minNumberOfSemesters(self, n, relations, k):
"""
:type n: int
:type relations: List[List[int]]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Parallel Courses II
* Difficulty: Hard
* Tags: array, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
```

```
* @param {number} n
* @param {number[][]} relations
* @param {number} k
* @return {number}
*/
var minNumberOfSemesters = function(n, relations, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Parallel Courses II
* Difficulty: Hard
* Tags: array, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function minNumberOfSemesters(n: number, relations: number[][], k: number):
number {

};
```

C# Solution:

```
/*
* Problem: Parallel Courses II
* Difficulty: Hard
* Tags: array, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MinNumberOfSemesters(int n, int[][] relations, int k) {
}
}
```

C Solution:

```
/*
* Problem: Parallel Courses II
* Difficulty: Hard
* Tags: array, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int minNumberOfSemesters(int n, int** relations, int relationsSize, int*
relationsColSize, int k) {

}
```

Go Solution:

```
// Problem: Parallel Courses II
// Difficulty: Hard
// Tags: array, graph, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func minNumberOfSemesters(n int, relations [][]int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minNumberOfSemesters(n: Int, relations: Array<IntArray>, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minNumberOfSemesters(_ n: Int, _ relations: [[Int]], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Parallel Courses II
// Difficulty: Hard
// Tags: array, graph, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn min_number_of_semesters(n: i32, relations: Vec<Vec<i32>>, k: i32) ->
i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} relations
# @param {Integer} k
# @return {Integer}
def min_number_of_semesters(n, relations, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $relations
* @param Integer $k
* @return Integer
*/
function minNumberOfSemesters($n, $relations, $k) {

}
}
```

Dart Solution:

```
class Solution {
int minNumberOfSemesters(int n, List<List<int>> relations, int k) {

}
}
```

Scala Solution:

```
object Solution {
def minNumberOfSemesters(n: Int, relations: Array[Array[Int]], k: Int): Int =
{

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_number_of_semesters(n :: integer, relations :: [[integer]], k ::
integer) :: integer
def min_number_of_semesters(n, relations, k) do

end
end
```

Erlang Solution:

```
-spec min_number_of_semesters(N :: integer(), Relations :: [[integer()]], K
:: integer()) -> integer().
min_number_of_semesters(N, Relations, K) ->
.
```

Racket Solution:

```
(define/contract (min-number-of-semesters n relations k)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer?)
)
```