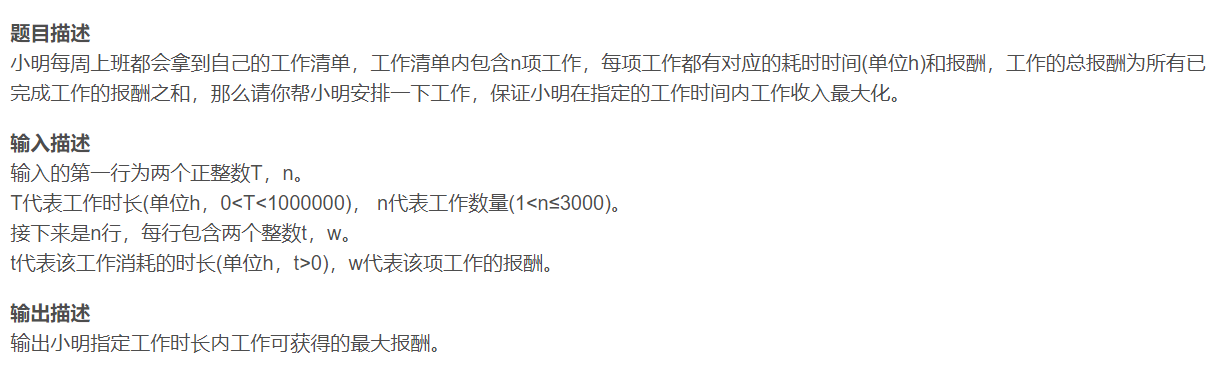
# **卷-最大报酬[100分]（ Java | Python3 | C++ | C语言 | JsNode | Go）**







import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// 读取T和n

int T = scanner.nextInt();

int n = scanner.nextInt();

// 初始化dp数组

int[] dp = new int[T + 1];

// 读取每项工作的时长和报酬

for (int i = 0; i < n; i++) {

int t = scanner.nextInt();

int w = scanner.nextInt();

// 从总时长T开始向下遍历，直到该工作的时长t,以保证当前工作不会超出总时长

for (int j = T; j >= t; j--) {

dp[j] = Math.max(dp[j], dp[j - t] + w);

}

}

// 输出指定时间内可以获得的最大报酬

System.out.println(dp[T]);

scanner.close();

}

}



# 读取输入的第一行，包含两个正整数T和n，分别表示工作的总时长和工作的数量

T, n = map(int, input().split())

# 初始化一个dp数组，长度为T+1，所有元素初始化为0

dp = [0] \* (T + 1)

# 读取每项工作的时长和报酬

for i in range(n):

t, w = map(int, input().split())

# 从总时长T开始向下遍历，直到该工作的时长t，以保证当前工作不会超出总时长

j = T

while j >= t:

# 动态规划更新dp[j]，选择当前工作和不选择当前工作，取报酬的最大值

dp[j] = max(dp[j], dp[j - t] + w)

j -= 1

# 输出指定时间内可以获得的最大报酬

print(dp[T])



#include <iostream>

#include <algorithm>

using namespace std;

int main() {

int T, n;

cin >> T >> n;

// 初始化dp数组

int dp[T + 1];

for (int i = 0; i < T + 1; i++)

dp[i] = 0;

// 读取每项工作的时长和报酬

for (int i = 0; i < n; ++i) {

int t, w;

cin >> t >> w;

// 从总时长T开始向下遍历，直到该工作的时长t，以保证当前工作不会超出总时长

for (int j = T; j >= t; j--) {

dp[j] = max(dp[j], dp[j - t] + w);

}

}

// 输出指定时间内可以获得的最大报酬

cout << dp[T] << endl;

return 0;

}



#include <stdio.h>

#include <string.h>

int main() {

int T, n;

scanf("%d %d", &T, &n);

// 初始化dp数组

int dp[T + 1];

memset(dp, 0, sizeof(dp));

// 读取每项工作的时长和报酬

for (int i = 0; i < n; ++i) {

int t, w;

scanf("%d %d", &t, &w);

// 从总时长T开始向下遍历，直到该工作的时长t，以保证当前工作不会超出总时长

for (int j = T; j >= t; --j) {

if (dp[j] < dp[j - t] + w) {

dp[j] = dp[j - t] + w;

}

}

}

// 输出指定时间内可以获得的最大报酬

printf("%d\n", dp[T]);

return 0;

}



const readline = require('readline');

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

let input = [];

rl.on('line', function(line) {

input.push(line);

});

rl.on('close', function() {

// 读取输入的第一行，包含两个正整数T和n，分别表示工作的总时长和工作的数量

let [T, n] = input[0].split(' ').map(Number);

// 初始化dp数组

let dp = Array(T + 1).fill(0);

// 读取每项工作的时长和报酬

for (let i = 1; i <= n; i++) {

let [t, w] = input[i].split(' ').map(Number);

// 从总时长T开始向下遍历，直到该工作的时长t，以保证当前工作不会超出总时长

for (let j = T; j >= t; j--) {

dp[j] = Math.max(dp[j], dp[j - t] + w);

}

}

// 输出指定时间内可以获得的最大报酬

console.log(dp[T]);

});



package main

import (

"bufio"

"fmt"

"os"

"strconv"

"strings"

)

func main() {

reader := bufio.NewReader(os.Stdin)

// 读取输入的第一行，包含两个正整数T和n，分别表示工作的总时长和工作的数量

firstLine, \_ := reader.ReadString('\n')

firstLine = strings.TrimSpace(firstLine)

tokens := strings.Split(firstLine, " ")

T, \_ := strconv.Atoi(tokens[0])

n, \_ := strconv.Atoi(tokens[1])

// 初始化dp数组

dp := make([]int, T+1)

// 读取每项工作的时长和报酬

for i := 0; i < n; i++ {

line, \_ := reader.ReadString('\n')

line = strings.TrimSpace(line)

tokens = strings.Split(line, " ")

t, \_ := strconv.Atoi(tokens[0])

w, \_ := strconv.Atoi(tokens[1])

// 从总时长T开始向下遍历，直到该工作的时长t，以保证当前工作不会超出总时长

for j := T; j >= t; j-- {

if dp[j] < dp[j-t]+w {

dp[j] = dp[j-t] + w

}

}

}

// 输出指定时间内可以获得的最大报酬

fmt.Println(dp[T])

}