**题目描述**

小明每周上班都会拿到自己的工作清单，工作清单内包含 n 项工作，每项工作都有对应的耗时时间（单位 h）和报酬，工作的总报酬为所有已完成工作的报酬之和，那么请你帮小明安排一下工作，保证小明在指定的工作时间内工作收入最大化。

**输入描述**

输入的第一行为两个正整数 T，n。
T 代表工作时长（单位 h， $0 < T < 1000000$），
n 代表工作数量（ $1 < n \leq 3000$ ）。
接下来是 n 行，每行包含两个整数 t，w。
t 代表该工作消耗的时长（单位 h， $t > 0$），w 代表该项工作的报酬。

**输出描述**

输出小明指定工作时长内工作可获得的最大报酬。

**用例**

| | |
|---|---|
| 输入 | 40 3<br>20 10<br>20 20<br>20 5 |
| 输出 | 30 |
| 说明 | 无 |

**题目解析**

本题是 01背包问题。可以使用动态规划求解。关于01背包问题请先看下面这个博客

算法设计 - 01背包问题_伏城之外的博客-CSDN博客

本题中

- 工作时长T相当于背包承重
- 每一项工作相当于每件物品

1. 工作消耗的时长相当于物品重量
2. 工作的报酬相当于物品的价值

**JavaScript算法源码**

```
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
let T, n;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 1) {
    [T, n] = lines[0].split(" ").map(Number);
  }

  if (n && lines.length === n + 1) {
    lines.shift();
    const tws = lines.map((line) => line.split(" ").map(Number));
    console.log(getResult(T, tws));
    lines.length = 0;
  }
});

/**
```

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
let T, n;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 1) {
    [T, n] = lines[0].split(" ").map(Number);
  }

  if (n && lines.length === n + 1) {
    lines.shift();
    const tws = lines.map((line) => line.split(" ").map(Number));
    console.log(getResult(T, tws));
    lines.length = 0;
  }
});

/**
 *
 * @param {*} T 工作时长
 * @param {*} tws 数组，元素是tw，也是数组，含义为[该工作消耗的时长，该项工作的报酬]
 */
function getResult(T, tws) {
  const maxI = tws.length + 1;
  const maxJ = T + 1;
  const dp = new Array(maxI).fill(0).map(() => new Array(maxJ).fill(0)); // 默认将dp数组元素全部初始化为0

  for (let i = 0; i < maxI; i++) {
    for (let j = 0; j < maxJ; j++) {
      if (i === 0 || j === 0) continue; // 第0行或第0列最大报酬保持0
      const [t, w] = tws[i - 1]; // 要选择的工作的[权重，价值]
      if (t > j) {
        // 如果要选择的工作的权重 > 当前背包权重，则无法放入背包，最大价值继承自上一行该列值
        dp[i][j] = dp[i - 1][j];
      } else {
        // 如果要选择的工作的权重 <= 当前背包权重
        // 则我们有两种选择
        // 1、不进行该工作，则最大价值继承自上一行该列值
        // 2、进行该工作，则纳入该工作的价值w，加上+ 剩余权重，在不进行该工作的范围内，可得的最大价值dp[i - 1][j - t]
        // 比较两种选择下的最大价值，取最大的
        dp[i][j] = Math.max(dp[i - 1][j], w + dp[i - 1][j - t]);
      }
    }
  }

  return dp.at(-1).at(-1); // 取二维数组最右下角元素作为题解
}
```

Java算法源码

```java
import java.util.*;

public class Main {
    static ArrayList<Integer[]> nodes;

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int T = sc.nextInt();
        int n = sc.nextInt();

        Integer[][] tws = new Integer[n][2];
        for (int i = 0; i < n; i++) {
            tws[i][0] = sc.nextInt();
            tws[i][1] = sc.nextInt();
        }

        System.out.println(getResult(T, tws));
    }

    /**
     *
     * @param T 工作时长
     * @param tws 数组，元素是tw，也是数组，含义为[该工作消耗的时长，该项工作的报酬]
     * @return 最大报酬
     */
    public static int getResult(int T, Integer[][] tws) {
        int maxI = tws.length + 1;
        int maxJ = T + 1;

        int[][] dp = new int[maxI][maxJ];

```

Java算法源码

```java
import java.util.*;

public class Main {
    static ArrayList<Integer[]> nodes;

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int T = sc.nextInt();
        int n = sc.nextInt();

        Integer[][] tws = new Integer[n][2];
        for (int i = 0; i < n; i++) {
            tws[i][0] = sc.nextInt();
            tws[i][1] = sc.nextInt();
        }

        System.out.println(getResult(T, tws));
    }

    /**
     *
     * @param T 工作时长
     * @param tws 数组，元素是tw，也是数组，含义为[该工作消耗的时长，该项工作的报酬]
     * @return 最大报酬
     */
    public static int getResult(int T, Integer[][] tws) {
        int maxI = tws.length + 1;
        int maxJ = T + 1;

        int[][] dp = new int[maxI][maxJ];

        for (int i = 0; i < maxI; i++) {
            for (int j = 0; j < maxJ; j++) {
                if (i == 0 || j == 0) continue; // 第0行或第0列最大报酬保持0

                int t = tws[i - 1][0];// 要选择的工作的[权重]
                int w = tws[i - 1][1];// 要选择的工作的[价值]

                if (t > j) {
                    // 如果要选择的工作的权重 > 当前背包权重，则无法放入背包，最大价值继承自上一行该列值
                    dp[i][j] = dp[i - 1][j];
                } else {
                    // 如果要选择的工作的权重 <= 当前背包权重
                    // 则我们有两种选择
                    // 1、不进行该工作，则最大价值继承自上一行该列值
                    // 2、进行该工作，则纳入该工作的价值w，加上+ 剩余权重，在不进行该工作的范围内，可得的最大价值dp[i - 1]
                    // 比较两种选择下的最大价值，取最大的
                    dp[i][j] = Math.max(dp[i - 1][j], w + dp[i - 1][j - t]);
                }
            }
        }

        return dp[maxI - 1][maxJ - 1];
    }
}
```

Python算法源码

```python
# 输入获取
t, n = map(int, input().split())
tws = [list(map(int, input().split())) for i in range(n)]


# 算法入口
def getResult(t, tws):
    """
    :param t: 工作时长
    :param tws: 列表，元素是tw，也是列表，含义为[该工作消耗的时长，该项工作的报酬]
    :return: 最大报酬
    """
    maxI = len(tws) + 1
    maxJ = t + 1

    # 默认将dp二维列表元素全部初始化为0
    dp = [[0 for j in range(maxJ)] for i in range(maxI)]

    for i in range(maxI):
        for j in range(maxJ):
            if i == 0 or j == 0:
                continue  # 第0行或第0列最大报酬保持0

            t, w = tws[i - 1]  # 要选择的工作的[权重，价值]

            if t > j:
                # 如果要选择的工作的权重 > 当前背包权重，则无法放入背包，最大价值继承自上一行该列值
                dp[i][j] = dp[i - 1][j]
            else:
                # 如果要选择的工作的权重 <= 当前背包权重
                # 则我们有两种选择
```

Python算法源码

复制

```python
# 输入获取
t, n = map(int, input().split())
tws = [list(map(int, input().split())) for i in range(n)]


# 算法入口
def getResult(t, tws):
    """
    :param t: 工作时长
    :param tws: 列表，元素是tw，也是列表，含义为[该工作消耗的时长，该项工作的报酬]
    :return: 最大报酬
    """
    maxI = len(tws) + 1
    maxJ = t + 1

    # 默认将dp二维列表元素全部初始化为0
    dp = [[0 for j in range(maxJ)] for i in range(maxI)]

    for i in range(maxI):
        for j in range(maxJ):
            if i == 0 or j == 0:
                continue  # 第0行或第0列最大报酬保持0

            t, w = tws[i - 1]  # 要选择的工作的[权重，价值]

            if t > j:
                # 如果要选择的工作的权重 > 当前背包权重，则无法放入背包，最大价值继承自上一行该列值
                dp[i][j] = dp[i - 1][j]
            else:
                # 如果要选择的工作的权重 <= 当前背包权重
                # 则我们有两种选择
                # 1、不进行该工作，则最大价值继承自上一行该列值
                # 2、进行该工作，则纳入该工作的价值w，加上 + 剩余权重，在不进行该工作的范围内，可得的最大价值dp[i - 1][j -
                # 比较两种选择下的最大价值，取最大的
                dp[i][j] = max(dp[i - 1][j], w + dp[i - 1][j - t])

    return dp[-1][-1]  # 取二维列表最右下角元素作为题解


# 算法调用
print(getResult(t, tws))
```