【DP】-工作安排

题目描述与示例

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长(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 < 100000`)，`n` 代表工作数量(`1 < n < 3000`)

接下来是 `n` 行，每行包含两个整数 `t` ， `w` 。 `t` 代表该项工作消耗的时长(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。

示例一

输入

```
40 3
20 10
20 20
20 5
```

输出

```
30
```

示例二

输入

```
100 3
50 10
20 30
50 20
```

输出

```
50
```

解题思路

本题属于**路径无关，求和最大值的01背包问题**。

代码

Python

1维DP数组

```python
# 题目：2023A-工作安排
# 分值：100
# 作者：许老师-闭着眼睛学数理化
# 算法：背包DP-1维DP数组
# 代码看不懂的地方，请直接在群上提问

# 输入最大工作时间max_time和工作数量N
max_time, N = map(int, input().split())

times = list()
wages = list()
```

```python

# 输入N份工作的时间和报酬
for _ in range(N):
    t, w = map(int, input().split())
    times.append(t)
    wages.append(w)


# 初始化1维dp数组，长度为 max+time + 1
# dp[t] = w 表示用t时间能够获得最多的总报酬w
# dp[0] = 0表示在花费了0的时间只能获得0的报酬
dp = [0] * (max_time+1)

# 遍历每一份工作i
for i in range(N):
    # 分别获得当前工作i的时间t和报酬w
    t = times[i]
    w = wages[i]
    # 01背包，逆序遍历前置时间pre_t
    for pre_t in range(max_time, -1, -1):
        # 获取pre_t对应的最高报酬
        pre_w = dp[pre_t]
        # 假如从某个工作时长pre_t出发，加上当前工作时间t
        # 所需要的总时间用cur_t表示
        cur_t = t+pre_t
        # 如果cur_t没有超出总限制时长
        if cur_t <= max_time:
            # 如果先前用pre_t时间获得的报酬pre_w加上当前工作获得的报酬w
            # 比原先的dp[cur_t]更大（默认为0），则更新dp[cur_t]
            dp[cur_t] = max(dp[cur_t], pre_w + w)


# 输出dp数组中的最大值
print(max(dp))
```

1维DP哈希表

```python
# 题目：2023A-工作安排
# 分值：100
# 作者：许老师-闭着眼睛学数理化
# 算法：背包DP-1维DP哈希表
# 代码看不懂的地方，请直接在群上提问


from collections import defaultdict
```

```python

# 输入最大工作时间max_time和工作数量N
max_time, N = map(int, input().split())

times = list()
wages = list()

# 输入N份工作的时间和报酬
for _ in range(N):
    t, w = map(int, input().split())
    times.append(t)
    wages.append(w)


# 初始化dp哈希表，key为时间t，value为报酬w
# dp[t] = w 表示用t时间能够获得最多的总报酬w
# dp[0] = 0表示在花费了0的时间只能获得0的报酬
# 注意此处构建 max_time * N 的dp二维数组也是可行的，
# 但由于max_time最大值为10^5，用哈希表更加节省空间
dp = defaultdict(int)
dp[0] = 0


# 遍历每一份工作i
for i in range(N):
    # 分别获得当前工作i的时间t和报酬w
    t = times[i]
    w = wages[i]
    # 遍历当前dp哈希表中，所有的时间和报酬，用pre_t, pre_w表示
    for pre_t, pre_w in list(dp.items()):
        # 假如从某个工作时长pre_t出发，加上当前工作时间t
        # 所需要的总时间用cur_t表示
        cur_t = t+pre_t
        # 如果cur_t没有超出总限制时长
        if cur_t <= max_time:
            # 如果先前用pre_t时间获得的报酬pre_w加上当前工作获得的报酬w
            # 比原先的dp[cur_t]更大（默认为0），则更新dp[cur_t]
            dp[cur_t] = max(dp[cur_t], pre_w + w)


# 输出dp哈希表中的最大值
print(max(dp.values()))
```

Java

1维DP数组

```java
import java.util.Scanner;

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

        // 输入最大工作时间max_time和工作数量N
        int max_time = input.nextInt();
        int N = input.nextInt();

        int[] times = new int[N];
        int[] wages = new int[N];

        // 输入N份工作的时间和报酬
        for (int i = 0; i < N; i++) {
            int t = input.nextInt();
            int w = input.nextInt();
            times[i] = t;
            wages[i] = w;
        }

        // 初始化1维dp数组，长度为 max+time + 1
        // dp[t] = w 表示用t时间能够获得最多的总报酬w
        // dp[0] = 0表示在花费了0的时间只能获得0的报酬
        int[] dp = new int[max_time + 1];

        // 遍历每一份工作i
        for (int i = 0; i < N; i++) {
            // 分别获得当前工作i的时间t和报酬w
            int t = times[i];
            int w = wages[i];
            // 01背包，逆序遍历前置时间pre_t
            for (int pre_t = max_time; pre_t >= 0; pre_t--) {
                // 获取pre_t对应的最高报酬
                int pre_w = dp[pre_t];
                // 假如从某个工作时长pre_t出发，加上当前工作时间t
                // 所需要的总时间用cur_t表示
                int cur_t = t + pre_t;
                // 如果cur_t没有超出总限制时长
                if (cur_t <= max_time) {
                    // 如果先前用pre_t时间获得的报酬pre_w加上当前工作获得的报酬w
                    // 比原先的dp[cur_t]更大（默认为0），则更新dp[cur_t]
                    dp[cur_t] = Math.max(dp[cur_t], pre_w + w);
                }
```

```
            }
        }

        // 输出dp数组中的最大值
        int maxWage = 0;
        for (int i = 0; i <= max_time; i++) {
            maxWage = Math.max(maxWage, dp[i]);
        }
        System.out.println(maxWage);
    }
}

```

1维DP哈希表

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

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

        // 输入最大工作时间max_time和工作数量N
        int max_time = scanner.nextInt();
        int N = scanner.nextInt();
        int[] times = new int[N];
        int[] wages = new int[N];

        // 输入N份工作的时间和报酬
        for (int i = 0; i < N; i++) {
            times[i] = scanner.nextInt();
            wages[i] = scanner.nextInt();
        }

        // 初始化dp哈希表，key为时间t，value为报酬w
        // dp[t] = w 表示用t时间能够获得最多的总报酬w
        // dp[0] = 0表示在花费了0的时间只能获得0的报酬
        Map<Integer, Integer> dp = new HashMap<>();
        dp.put(0, 0);

        // 遍历每一份工作i
        for (int i = 0; i < N; i++) {
            // 分别获得当前工作i的时间t和报酬w
            int t = times[i];
```

```
            int w = wages[i];
            // 遍历当前dp哈希表中，所有的时间和报酬，用pre_t，pre_w表示
            Map<Integer, Integer> currentDP = new HashMap<>(dp);
            for (Map.Entry<Integer, Integer> entry : currentDP.entrySet()) {
                int pre_t = entry.getKey();
                int pre_w = entry.getValue();
                // 假如从某个工作时长pre_t出发，加上当前工作时间t
                // 所需要的总时间用cur_t表示
                int cur_t = t + pre_t;
                // 如果cur_t没有超出总限制时长
                if (cur_t <= max_time) {
                    // 如果先前用pre_t时间获得的报酬pre_w加上当前工作获得的报酬w
                    // 比原先的dp[cur_t]更大（默认为0），则更新dp[cur_t]
                    dp.put(cur_t, Math.max(dp.getOrDefault(cur_t, 0), pre_w + w));
                }
            }
        }

        // 输出dp哈希表中的最大值
        int maxWage = 0;
        for (int wage : dp.values()) {
            maxWage = Math.max(maxWage, wage);
        }
        System.out.println(maxWage);
    }
}

```

C++

1维DP数组

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int max_time, N;
    cin >> max_time >> N;

    vector<int> times(N);
    vector<int> wages(N);

    for (int i = 0; i < N; i++) {
```

```
        int t, w;
        cin >> t >> w;
        times[i] = t;
        wages[i] = w;
    }

    vector<int> dp(max_time + 1, 0);

    for (int i = 0; i < N; i++) {
        int t = times[i];
        int w = wages[i];
        for (int pre_t = max_time; pre_t >= 0; pre_t--) {
            int pre_w = dp[pre_t];
            int cur_t = t + pre_t;
            if (cur_t <= max_time) {
                dp[cur_t] = max(dp[cur_t], pre_w + w);
            }
        }
    }

    int maxWage = 0;
    for (int i = 0; i <= max_time; i++) {
        maxWage = max(maxWage, dp[i]);
    }
    cout << maxWage << endl;

    return 0;
}

```

1维DP哈希表

```
#include <iostream>
#include <unordered_map>
#include <algorithm>
#include <vector>
using namespace std;

int main() {
    // 输入最大工作时间max_time和工作数量N
    int max_time, N;
    cin >> max_time >> N;
    vector<int> times(N);
    vector<int> wages(N);

```

```cpp
    // 输入N份工作的时间和报酬
    for (int i = 0; i < N; i++) {
        cin >> times[i] >> wages[i];
    }

    // 初始化dp哈希表，key为时间t，value为报酬w
    // dp[t] = w 表示用t时间能够获得最多的总报酬w
    // dp[0] = 0表示在花费了0的时间只能获得0的报酬
    unordered_map<int, int> dp;
    dp[0] = 0;

    // 遍历每一份工作i
    for (int i = 0; i < N; i++) {
        // 分别获得当前工作i的时间t和报酬w
        int t = times[i];
        int w = wages[i];
        // 遍历当前dp哈希表中，所有的时间和报酬，用pre_t，pre_w表示
        unordered_map<int, int> currentDP(dp);
        for (auto entry : currentDP) {
            int pre_t = entry.first;
            int pre_w = entry.second;
            // 假如从某个工作时长pre_t出发，加上当前工作时间t
            // 所需要的总时间用cur_t表示
            int cur_t = t + pre_t;
            // 如果cur_t没有超出总限制时长
            if (cur_t <= max_time) {
                // 如果先前用pre_t时间获得的报酬pre_w加上当前工作获得的报酬w
                // 比原先的dp[cur_t]更大（默认为0），则更新dp[cur_t]
                dp[cur_t] = max(dp[cur_t], pre_w + w);
            }
        }
    }

    // 输出dp哈希表中的最大值
    int maxWage = 0;
    for (auto entry : dp) {
        maxWage = max(maxWage, entry.second);
    }
    cout << maxWage << endl;

    return 0;
}

```

时空复杂度

时间复杂度：`O(max_time*N)`。需要遍历`N`份工作，每一份工作都要考虑`len(dp)`种前置情况。

空间复杂度：`O(max_time)`。dp哈希表所占空间。