

【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

```
1 40 3
2 20 10
3 20 20
4 20 5
```

输出

```
1 30
```

示例二

输入

```
1 100 3
2 50 10
3 20 30
4 50 20
```

输出

```
1 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

```
1 # 题目：2023A-工作安排
2 # 分值：100
3 # 作者：许老师-闭着眼睛学数理化
4 # 算法：背包DP-1维DP数组
5 # 代码看不懂的地方，请直接在群上提问
6
7 # 输入最大工作时间max_time和工作数量N
8 max_time, N = map(int, input().split())
9
10 times = list()
11 wages = list()
```

```

12
13 # 输入N份工作的时间和报酬
14 for _ in range(N):
15     t, w = map(int, input().split())
16     times.append(t)
17     wages.append(w)
18
19
20 # 初始化1维dp数组，长度为 max_time + 1
21 # dp[t] = w 表示用t时间能够获得最多的总报酬w
22 # dp[0] = 0表示在花费了0的时间只能获得0的报酬
23 dp = [0] * (max_time+1)
24
25 # 遍历每一份工作i
26 for i in range(N):
27     # 分别获得当前工作i的时间t和报酬w
28     t = times[i]
29     w = wages[i]
30     # 01背包，逆序遍历前置时间pre_t
31     for pre_t in range(max_time, -1, -1):
32         # 获取pre_t对应的最高报酬
33         pre_w = dp[pre_t]
34         # 假如从某个工作时长pre_t出发，加上当前工作时间t
35         # 所需要的总时间用cur_t表示
36         cur_t = t+pre_t
37         # 如果cur_t没有超出总限制时长
38         if cur_t <= max_time:
39             # 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
40             # 比原先的dp[cur_t]更大（默认为0），则更新dp[cur_t]
41             dp[cur_t] = max(dp[cur_t], pre_w + w)
42
43
44 # 输出dp数组中的最大值
45 print(max(dp))

```

1维DP哈希表

```

1 # 题目：2023A-工作安排
2 # 分值：100
3 # 作者：许老师-闭着眼睛学数理化
4 # 算法：背包DP-1维DP哈希表
5 # 代码看不懂的地方，请直接在群上提问
6
7
8 from collections import defaultdict

```

```

9
10
11 # 输入最大工作时间max_time和工作数量N
12 max_time, N = map(int, input().split())
13
14 times = list()
15 wages = list()
16
17 # 输入N份工作的时间和报酬
18 for _ in range(N):
19     t, w = map(int, input().split())
20     times.append(t)
21     wages.append(w)
22
23
24 # 初始化dp哈希表, key为时间t, value为报酬w
25 # dp[t] = w 表示用t时间能够获得最多的总报酬w
26 # dp[0] = 0表示在花费了0的时间只能获得0的报酬
27 # 注意此处构建 max_time * N 的dp二维数组也是可行的,
28 # 但由于max_time最大值为10^5, 用哈希表更加节省空间
29 dp = defaultdict(int)
30 dp[0] = 0
31
32
33 # 遍历每一份工作i
34 for i in range(N):
35     # 分别获得当前工作i的时间t和报酬w
36     t = times[i]
37     w = wages[i]
38     # 遍历当前dp哈希表中, 所有的时间和报酬, 用pre_t, pre_w表示
39     for pre_t, pre_w in list(dp.items()):
40         # 假如从某个工作时长pre_t出发, 加上当前工作时间t
41         # 所需要的总时间用cur_t表示
42         cur_t = t+pre_t
43         # 如果cur_t没有超出总限制时长
44         if cur_t <= max_time:
45             # 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
46             # 比原先的dp[cur_t]更大(默认为0), 则更新dp[cur_t]
47             dp[cur_t] = max(dp[cur_t], pre_w + w)
48
49
50 # 输出dp哈希表中的最大值
51 print(max(dp.values()))

```

1维DP数组

```
1 import java.util.Scanner;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6
7         // 输入最大工作时间max_time和工作数量N
8         int max_time = input.nextInt();
9         int N = input.nextInt();
10
11         int[] times = new int[N];
12         int[] wages = new int[N];
13
14         // 输入N份工作的时间和报酬
15         for (int i = 0; i < N; i++) {
16             int t = input.nextInt();
17             int w = input.nextInt();
18             times[i] = t;
19             wages[i] = w;
20         }
21
22         // 初始化1维dp数组, 长度为 max+time + 1
23         // dp[t] = w 表示用t时间能够获得最多的总报酬w
24         // dp[0] = 0表示在花费了0的时间只能获得0的报酬
25         int[] dp = new int[max_time + 1];
26
27         // 遍历每一份工作i
28         for (int i = 0; i < N; i++) {
29             // 分别获得当前工作i的时间t和报酬w
30             int t = times[i];
31             int w = wages[i];
32             // 01背包, 逆序遍历前置时间pre_t
33             for (int pre_t = max_time; pre_t >= 0; pre_t--) {
34                 // 获取pre_t对应的最高报酬
35                 int pre_w = dp[pre_t];
36                 // 假如从某个工作时长pre_t出发, 加上当前工作时间t
37                 // 所需要的总时间用cur_t表示
38                 int cur_t = t + pre_t;
39                 // 如果cur_t没有超出总限制时长
40                 if (cur_t <= max_time) {
41                     // 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
42                     // 比原先的dp[cur_t]更大 (默认为0), 则更新dp[cur_t]
43                     dp[cur_t] = Math.max(dp[cur_t], pre_w + w);
44                 }
45             }
46         }
47     }
48 }
```

```

45         }
46     }
47
48     // 输出dp数组中的最大值
49     int maxWage = 0;
50     for (int i = 0; i <= max_time; i++) {
51         maxWage = Math.max(maxWage, dp[i]);
52     }
53     System.out.println(maxWage);
54 }
55 }
56

```

1维DP哈希表

```

1  import java.util.HashMap;
2  import java.util.Map;
3  import java.util.Scanner;
4
5  public class Main {
6      public static void main(String[] args) {
7          Scanner scanner = new Scanner(System.in);
8
9          // 输入最大工作时间max_time和工作数量N
10         int max_time = scanner.nextInt();
11         int N = scanner.nextInt();
12         int[] times = new int[N];
13         int[] wages = new int[N];
14
15         // 输入N份工作的时间和报酬
16         for (int i = 0; i < N; i++) {
17             times[i] = scanner.nextInt();
18             wages[i] = scanner.nextInt();
19         }
20
21         // 初始化dp哈希表, key为时间t, value为报酬w
22         // dp[t] = w 表示用t时间能够获得最多的总报酬w
23         // dp[0] = 0表示在花费了0的时间只能获得0的报酬
24         Map<Integer, Integer> dp = new HashMap<>();
25         dp.put(0, 0);
26
27         // 遍历每一份工作i
28         for (int i = 0; i < N; i++) {
29             // 分别获得当前工作i的时间t和报酬w
30             int t = times[i];

```

```

31         int w = wages[i];
32         // 遍历当前dp哈希表中, 所有的时间和报酬, 用pre_t, pre_w表示
33         Map<Integer, Integer> currentDP = new HashMap<>(dp);
34         for (Map.Entry<Integer, Integer> entry : currentDP.entrySet()) {
35             int pre_t = entry.getKey();
36             int pre_w = entry.getValue();
37             // 假如从某个工作时长pre_t出发, 加上当前工作时间t
38             // 所需要的总时间用cur_t表示
39             int cur_t = t + pre_t;
40             // 如果cur_t没有超出总限制时长
41             if (cur_t <= max_time) {
42                 // 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
43                 // 比原先的dp[cur_t]更大 (默认为0), 则更新dp[cur_t]
44                 dp.put(cur_t, Math.max(dp.getDefault(cur_t, 0), pre_w +
w));
45             }
46         }
47     }
48
49     // 输出dp哈希表中的最大值
50     int maxWage = 0;
51     for (int wage : dp.values()) {
52         maxWage = Math.max(maxWage, wage);
53     }
54     System.out.println(maxWage);
55 }
56 }
57

```

C++

1维DP数组

```

1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main() {
6      int max_time, N;
7      cin >> max_time >> N;
8
9      vector<int> times(N);
10     vector<int> wages(N);
11
12     for (int i = 0; i < N; i++) {

```

```

13         int t, w;
14         cin >> t >> w;
15         times[i] = t;
16         wages[i] = w;
17     }
18
19     vector<int> dp(max_time + 1, 0);
20
21     for (int i = 0; i < N; i++) {
22         int t = times[i];
23         int w = wages[i];
24         for (int pre_t = max_time; pre_t >= 0; pre_t--) {
25             int pre_w = dp[pre_t];
26             int cur_t = t + pre_t;
27             if (cur_t <= max_time) {
28                 dp[cur_t] = max(dp[cur_t], pre_w + w);
29             }
30         }
31     }
32
33     int maxWage = 0;
34     for (int i = 0; i <= max_time; i++) {
35         maxWage = max(maxWage, dp[i]);
36     }
37     cout << maxWage << endl;
38
39     return 0;
40 }
41

```

1维DP哈希表

```

1  #include <iostream>
2  #include <unordered_map>
3  #include <algorithm>
4  #include <vector>
5  using namespace std;
6
7  int main() {
8      // 输入最大工作时间max_time和工作数量N
9      int max_time, N;
10     cin >> max_time >> N;
11     vector<int> times(N);
12     vector<int> wages(N);
13

```



```

14 // 输入N份工作的时间和报酬
15 for (int i = 0; i < N; i++) {
16     cin >> times[i] >> wages[i];
17 }
18
19 // 初始化dp哈希表, key为时间t, value为报酬w
20 // dp[t] = w 表示用t时间能够获得最多的总报酬w
21 // dp[0] = 0表示在花费了0的时间只能获得0的报酬
22 unordered_map<int, int> dp;
23 dp[0] = 0;
24
25 // 遍历每一份工作i
26 for (int i = 0; i < N; i++) {
27     // 分别获得当前工作i的时间t和报酬w
28     int t = times[i];
29     int w = wages[i];
30     // 遍历当前dp哈希表中, 所有的时间和报酬, 用pre_t, pre_w表示
31     unordered_map<int, int> currentDP(dp);
32     for (auto entry : currentDP) {
33         int pre_t = entry.first;
34         int pre_w = entry.second;
35         // 假如从某个工作时长pre_t出发, 加上当前工作时间t
36         // 所需要的总时间用cur_t表示
37         int cur_t = t + pre_t;
38         // 如果cur_t没有超出总限制时长
39         if (cur_t <= max_time) {
40             // 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
41             // 比原先的dp[cur_t]更大 (默认为0), 则更新dp[cur_t]
42             dp[cur_t] = max(dp[cur_t], pre_w + w);
43         }
44     }
45 }
46
47 // 输出dp哈希表中的最大值
48 int maxWage = 0;
49 for (auto entry : dp) {
50     maxWage = max(maxWage, entry.second);
51 }
52 cout << maxWage << endl;
53
54 return 0;
55 }
56

```

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 $\text{len}(dp)$ 种前置情况。

空间复杂度： $O(\max_time)$ 。dp哈希表所占空间。