OJ 990

Diving for Gold

Prof. Edson Alves – UnB/FGA

Problema

John is a diver and a treasure hunter. He has just found the location of a pirate ship full of treasures. The sofisticated sonar system on board his ship allows him to identify the location, depth and quantity of gold in each suken treasure. Unfortunatelly, John forgot to bring a GPS device and the chances of ever finding this location again are very slim so he has to grab the gold now. To make the situation worse, John has only one compressed air bottle.

John wants to dive with the compressed air bottle to recover as much gold as possible from the wreck. Write a program John can use to select which treasures he should pick to maximize the quantity of gold recovered.

Problema

The problem has the following restrictions:

- There are n treasures $\{(d_1, v_1), (d_2, v_2), \ldots, (d_n, v_n)\}$ represented by the pair (depth, quantity of gold). There are at most 30 treasures.
- The air bottle only allows for t seconds under water. t is at most 1000 seconds.
- In each dive, John can bring the maximum of one treasure at a time.
- The descent time is $td_i = w \times d_i$, where w is an integer constant.
- The ascent time is $ta_i = 2w \times d_i$, where w is an integer constant.
- Due to instrument limitations, all parameters are integer.

Entrada e saída

Input

The input to this program consists of a sequence of integer values. Input contains several test cases. The first line of each dataset should contain the values t and w. The second line contains the number of treasures. Each of the following lines contains the depth d_i and quantity of gold v_i of a different treasure.

A blank line separates each test case.

Note:

In this sample input, the bottle of compressed air has a capacity of 200 seconds, the constant w has the value 4 and there are 3 treasures, the first one at a depth of 10 meters and worth 5 coins of gold, the second one at a depth of 10 meters and worth 1 coin of gold, and the third one at 7 meters and worth 2 coins of gold.

Entrada e saída

Output

The first line of the output for each dataset should contain the maximum amount of gold that John can recover from the wreck. The second line should contain the number of recovered treasures. Each of the following lines should contain the depth and amount of gold of each recovered treasure. Treasures should be presented in the same order as the input file.

Print a blank line between outputs for different datasets.

Exemplo de entradas e saídas

Sample Input

```
210 4
3
10 5
10 1
7 2
```

Sample Output

```
7
2
10 5
7 2
```

Solução $O(TN)$

- Este é um problema de mochila binária cuja solução demanda não apenas o valor ótima, mas também a lista dos elementos selecionados
- O tempo máximo de mergulho T é a capacidade da mochila
- O valor de cada elemento é v_i
- Como cada mergulho só permite reaver um único elemento, o "peso" do elemento é a soma do tempo de mergulho com o tempo de retorno à superfície
- Assim,
$$w_i = w \times d_i + 2w \times d_i = 3wd_i$$
- Para evitar o veredito WA, não se esqueça de incluir uma linha em branco entre as saídas de dois casos de teste consecutivos

Solução $O(TN)$

```cpp
#include <bits/stdc++.h>

using namespace std;
using ii = pair<int, int>;

pair<int, vector<int>> solve(int T, int W, int N, vector<ii>& cs)
{
    vector<vector<int>> st(N + 1, vector<int>(T + 1, 0));
    vector<vector<int>> ps(N + 1, vector<int>(T + 1, -1));

    for (int i = 1; i <= N; ++i)
        cs[i].first *= 3*W;

    for (int i = 1; i <= N; ++i)
    {
        for (int t = 1; t <= T; ++t)
        {
            st[i][t] = st[i - 1][t];
            ps[i][t] = 0;
            auto [d, v] = cs[i];
```

Solução $O(TN)$

```
                if (d <= t and st[i - 1][t - d] + v > st[i][t])
                {
                    st[i][t] = st[i - 1][t - d] + v;
                    ps[i][t] = 1;
                }
            }
        }

        int t = T;
        vector<int> is;

        for (int i = N; i >= 1; --i)
        {
            if (ps[i][t]) {
                is.push_back(i);
                t -= cs[i].first;
            }
        }

        reverse(is.begin(), is.end());
```

Solução $O(TN)$

```
    return { st[N][T], is };
}

int main()
{
    ios::sync_with_stdio(false);

    int T, W, first = 1;

    while (cin >> T >> W)
    {
        int N;
        cin >> N;

        vector<ii> cs(N + 1);

        for (int i = 1; i <= N; ++i)
            cin >> cs[i].first >> cs[i].second;

        auto [ans, is] = solve(T, W, N, cs);
```

Solução $O(TN)$

```
        if (not first)
            cout << '\n';

        cout << ans << '\n';
        cout << is.size() << '\n';

        for (auto i : is)
            cout << cs[i].first / (3*W) << ' ' << cs[i].second << '\n';

        first = 0;
    }

    return 0;
}
```