

# OJ 10465

*Homer Simpson*

---

Prof. Edson Alves – UnB/FGA

Homer Simpson, a very smart guy, likes eating Krusty-burgers. It takes Homer  $m$  minutes to eat a Krusty-burger. However, there's a new type of burger in Apu's Kwik-e-Mart. Homer likes those too. It takes him  $n$  minutes to eat one of these burgers. Given  $t$  minutes, you have to find out the maximum number of burgers Homer can eat without wasting any time. If he must waste time, he can have beer.

## Input

Input consists of several test cases. Each test case consists of three integers  $m, n, t$  ( $0 < m, n, t < 10000$ ). Input is terminated by EOF.

## Output

For each test case, print in a single line the maximum number of burgers Homer can eat without having beer. If homer must have beer, then also print the time he gets for drinking, separated by a single space. It is preferable that Homer drinks as little beer as possible.

## Exemplo de entradas e saídas

### Sample Input

3 5 54

3 5 55

### Sample Output

18

17

## Solução $O(T)$

- Observe que o problema consiste em minimizar o consumo de cerveja e, em segundo lugar, maximizar o número de hambúrguer
- O problema pode ser caracterizado por um único parâmetro: o recurso disponível  $t$
- Para cada subproblema  $p(t)$  a solução é caracterizada por um par de valores  $(b, h)$ : o número mínimo de cervejas e o máximo de hambúrgueres a serem consumidos
- Para manter a ordenação das soluções ótimas e usar a função `max()` do C++, o valor de  $b$  será representado pelo seu simétrico

## Solução $O(T)$

- O caso base ocorre quando  $t = 0$ : neste caso,  $p(0) = (0, 0)$
- Há 3 transições possíveis para o estado  $p(t)$ :
  1. gastar todo o tempo restante com cervejas
  2. comer um hambúrguer durante  $m$  minutos
  3. comer um hambúrguer durante  $n$  minutos
- Como há  $O(T)$  estados distintos, e as transições tem custo  $O(1)$ , uma solução baseada em programação dinâmica tem complexidade  $O(T)$
- A complexidade em memória também é  $O(T)$

## Solução $O(T)$

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4 using ii = pair<int, int>;
5
6 const int MAX { 10010 };
7
8 ii st[MAX];
9
10 ii dp(int t, int N, int M)
11 {
12     if (t == 0)
13         return { 0, 0 };
14
15     if (st[t] != ii(-1, -1))
16         return st[t];
17
18     auto res = ii(-t, 0);
```

## Solução $O(T)$

```
20  if (t >= N)
21  {
22      auto [beer, sol] = dp(t - N, N, M);
23      res = max(res, ii(beer, sol + 1));
24  }
25
26  if (t >= M)
27  {
28      auto [beer, sol] = dp(t - M, N, M);
29      res = max(res, ii(beer, sol + 1));
30  }
31
32  return (st[t] = res);
33 }
34
35 ii solve(int N, int M, int T)
36 {
37     memset(st, -1, sizeof st);
38     return dp(T, N, M);
39 }
```



## Solução $O(T)$

```
41 int main()
42 {
43     ios::sync_with_stdio(false);
44
45     int M, N, T;
46
47     while (cin >> M >> N >> T)
48     {
49         auto [beer, ans] = solve(M, N, T);
50
51         cout << ans;
52
53         if (beer)
54             cout << ' ' << -beer;
55
56         cout << '\n';
57     }
58
59     return 0;
60 }
```