Paradigmas de Resolução de Problemas

Programação Dinâmica – Definição: Exercícios Resolvidos

Prof. Edson Alves - UnB/FGA

2020

Sumário

1. OJ 10465 – Homer Simpson

2. SPOJ MAXMATCH – Maximum Self-Matching

OJ 10465 – Homer Simpson

Problema

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.

Entrada e saída

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 hambúrger
- 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 hambúrgeres 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 recurso com cervejas
  2. comprar uma cerveja por m
  3. comprar uma cerveja por n
- 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 $O(T)$

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

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

const int MAX { 10010 };

ii st[MAX];

ii dp(int t, int N, int M)
{
    if (t == 0)
        return { 0, 0 };

    if (st[t] != ii(-1, -1))
        return st[t];

    auto res = ii(-t, 0);

    if (t >= N)
```

Solução $O(T)$

```cpp
    {
        auto [beer, sol] = dp(t - N, N, M);
        res = max(res, ii(beer, sol + 1));
    }

    if (t >= M)
    {
        auto [beer, sol] = dp(t - M, N, M);
        res = max(res, ii(beer, sol + 1));
    }

    st[t] = res;
    return res;
}

ii solve(int N, int M, int T)
{
    memset(st, -1, sizeof st);
    return dp(T, N, M);
}
```

Solução $O(T)$

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

    int M, N, T;

    while (cin >> M >> N >> T)
    {
        auto [beer, ans] = solve(M, N, T);

        cout << ans;

        if (beer)
            cout << ' ' << -beer;

        cout << '\n';
    }

    return 0;
}
```

SPOJ SQRBR – Square Brackets

Problema

You're given a string s consisting of letters 'a', 'b' and 'c'.

The matching function $m_s(i)$ is defined as the number of matching characters of s and its i-shift. In other words, $m_s(i)$ is the number of characters that are matched when you align the 0-th character of s with the i-th character of its copy.

You are asked to compute the maximum of $m_s(i)$ for all i $(1 \leq i \leq |s|)$. To make it a bit harder, you should also output all the optimal i's in increasing order.

Entrada e saída

Input

The first and only line of input contains the string s $(2 \leq |s| \leq 10^5)$.

Output

The first line of output contains the maximal $m_s(i)$ over all i.

The second line of output contains all the i's for which $m_s(i)$ reaches maximum.

Exemplo de entradas e saídas

Sample Input	Sample Output
caccacaa	4
	3

Solução $O(N^2)$

- A função $m_s(i)$ corresponde à distância de Hamming entre a string s e a substring $b_i = s[i..(N-1)]$, com $i = 1, 2, \ldots, N$
- Esta distância de Hamming entre as strings s e t é dada por

$$D(s,t) = \sum_{i=0}^{m} (1 - \delta_{s[i]}(t[i])),$$

  onde $m = \min(|s|, |t|)$ e $\delta_j(j) = 1$ e $\delta_j(k) = 0$, se $j \neq k$
- Assim, D tem complexidade $O(N)$
- Uma solução que computa $D(s, b_i)$ para todos os valores de i tem complexidade $O(N^2)$, e leva ao veredito TLE

Solução $O(N \log N)$

- Considere as strings t_k tais que $t_k[i] = \delta_k(s[i])$
- Na string dada no exemplo, $t_a =$ "01001011", $t_b =$ "00000000" e $t_c =$ "10110100"
- A ideia é computar $m_s(i)$ como a soma das funções $m_i^k(s)$, com $k =$ 'a', 'b' e 'c', onde $m_i^k(s)$ é calculada a partir da string t_k
- Usando esta representação binária das strings, o cálculo da distância de Hamming corresponde ao produto escalar entre a string t_k e a substring b_i
- Estes produtos escalares surgem na multiplicação dos polinômios correspondentes a strings t_k e a reversa da string b_i

Solução $O(N \log N)$

- Assim, os valores de $m_i^k(s)$ para cada i podem ser computados todos de uma só vez, por meio da multiplicação de polinômios, em $O(N \log N)$
- Atente que, devido à multiplicação polinomial, o valor de $m_i^k(s)$ será o coeficiente do monômio de grau $i + N$, onde N é o tamando da string t_k
- Embora $m_i^k(N) = 0$, é preciso considerá-lo na composição final da resposta, uma vez que 0 pode ser o valor máximo obtido, e neste caso o índice N também deve ser listado
- Repetido o processo para cada valor de k, o problema pode ser resolvido em $O(N \log N)$

Solução $O(N \log N)$

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

using namespace std;

const int MAX { 25 };

int st[2*MAX][2*MAX];

int dp(int i, int open, int N, const set<int>& xs)
{
    if (i == N)
        return open ? 0 : 1;

    if (st[i][open] != -1)
        return st[i][open];

    auto res = dp(i + 1, open + 1, N, xs);

    if (xs.count(i) == 0 and open)
        res += dp(i + 1, open - 1, N, xs);

```

Solução $O(N \log N)$

```
    st[i][open] = res;
    return res;
}

int solve(int N, const set<int>& xs)
{
    memset(st, -1, sizeof st);

    return dp(0, 0, 2*N, xs);
}

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

    int T;
    cin >> T;

    while (T--)
    {
        int N, K;
```

Solução $O(N \log N)$

```
        cin >> N >> K;

        set<int> xs;

        for (int i = 0; i < K; ++i)
        {
            int x;
            cin >> x;

            xs.insert(x - 1);
        }

        auto ans = solve(N, xs);

        cout << ans << '\n';
    }

    return 0;
}
```

Referências

1. OJ 10465 – Homer Simpson
2. SPOJ SQRBR - Square Brackets