

Paradigmas de Resolução de Problemas

Programação Dinâmica – Definição: Exercícios Resolvidos

Prof. Edson Alves - UnB/FGA

2020

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

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.

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 recurso com cervejas
 2. comprar uma hambúrguer por m
 3. comprar uma hambúrguer 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 $O(T)$

```
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);
19
20     if (t >= N)
```

Solução $O(T)$

```
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 st[t] = res;
33 return res;
34 }
35
36 ii solve(int N, int M, int T)
37 {
38     memset(st, -1, sizeof st);
39     return dp(T, N, M);
40 }
41
```

Solução $O(T)$

```
42 int main()
43 {
44     ios::sync_with_stdio(false);
45
46     int M, N, T;
47
48     while (cin >> M >> N >> T)
49     {
50         auto [beer, ans] = solve(M, N, T);
51
52         cout << ans;
53
54         if (beer)
55             cout << ' ' << -beer;
56
57         cout << '\n';
58     }
59
60     return 0;
61 }
```

SPOJ SQRBR – Square Brackets

Problema

You are given:

- a positive integer n ,
- an integer k , $1 \leq k \leq n$,
- an increasing sequence of k integers $0 < s_1 < s_2 < \dots < s_k \leq 2n$.

What is the number of proper bracket expressions of length $2n$ with opening brackets appearing in positions s_1, s_2, \dots, s_k ?

Illustration

Several proper bracket expressions:

```
[[]][[]][[]]  
[[]][[]][[]]
```

An improper bracket expression:

```
[[]][[]][[]]
```

There is exactly one proper expression of length 8 with opening brackets in positions 2, 5 and 7.

Task

Write a program which for each data set from a sequence of several data sets:

- reads integers n, k and an increasing sequence of k integers from input,
- computes the number of proper bracket expressions of length $2n$ with opening brackets appearing at positions s_1, s_2, \dots, s_k ,
- writes the result to output.

Input

The first line of the input file contains one integer d , $1 \leq d \leq 10$, which is the number of data sets. The data sets follow. Each data set occupies two lines of the input file. The first line contains two integers n and k separated by single space, $1 \leq n \leq 19, 1 \leq k \leq n$. The second line contains an increasing sequence of k integers from the interval $[1; 2n]$ separated by single spaces.

Output

The i -th line of output should contain one integer – the number of proper bracket expressions of length $2n$ with opening brackets appearing at positions s_1, s_2, \dots, s_k .

Exemplo de entradas e saídas

Sample Input

5

1 1

1

1 1

2

2 1

1

3 1

2

4 2

5 7

Sample Output

1

0

2

3

2

Solução $O(N^2)$

- Uma solução de força bruta listaria todas as 2^{2n} strings s_i formadas pelos caracteres '[' e ']' e, para cada i , identificaria se s_i é uma sequência válida e, em caso afirmativo, se os caracteres das posições indicadas na entrada são iguais a '['
- A verificação da validade de s_i é feita em $O(N)$, de modo que tal solução teria complexidade $O(N2^{2N})$, o que levaria a um veredito TLE
- Uma forma de reduzir esta complexidade é construir as sequências válidas iterativamente e não recalculá-las a validade de uma mesma subsequência múltiplas vezes
- Isto pode ser feito por meio de programação dinâmica

Solução $O(N^2)$

- Seja s uma string de tamanho $2n$ tal que $s_j = '['$ para toda posição j indicada na entrada
- Defina $c(i, open)$ como o número de sequências válidas que podem ser formadas a partir do sufixo $s[i, 2n]$ sem modificar os caracteres s_j pré-definidos, considerando que restaram $open$ caracteres '[' sem o ']' correspondente no prefixo $s[1, (i - 1)]$
- Uma vez que os sufixos $s[m, 2n]$ são vazios se $m > 2n$, então $c(m, open) = 0$ se $m > 2n$
- Atenção, porém, ao caso $m = 2n + 1$: ele trata o primeiro sufixo vazio, e indica que todos os caracteres anteriores foram definidos
- Assim, se $open = 0$, a sequência definida anteriormente é válida, de modo que $c(2n + 1, 0) = 1$
- Os demais permanecem iguais a zero, se $m > 2n$

Solução $O(N^2)$

- Para $1 \leq i \leq 2n$, há duas transições possíveis:
 1. adicionar mais um símbolo '[' na string
 2. adicionar um símbolo ']' na string, se $open > 0$ e se a posição não estiver pré-definida

- A primeira transição corresponde a

$$c(i, open) = c(i + 1, open + 1)$$

- Esta primeira transição sempre pode ser feita
- A segunda transição nem sempre pode ser feita, devido as restrições apresentadas
- Caso $open > 0$ e i não seja um dos índices pré-definidos com '[', então

$$c(i, open) = c(i + 1, open + 1) + c(i + 1, open - 1)$$

- Como há $O(N^2)$ estados e as transições são feitas em $O(1)$, a solução tem complexidade $O(N^2)$

Solução $O(N \log N)$

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int MAX { 25 };
6
7 int st[2*MAX][2*MAX];
8
9 int dp(int i, int open, int N, const set<int>& xs)
10 {
11     if (i == N)
12         return open ? 0 : 1;
13
14     if (st[i][open] != -1)
15         return st[i][open];
16
17     auto res = dp(i + 1, open + 1, N, xs);
18
19     if (xs.count(i) == 0 and open)
20         res += dp(i + 1, open - 1, N, xs);
21 }
```

Solução $O(N \log N)$

```
22     st[i][open] = res;
23     return res;
24 }
25
26 int solve(int N, const set<int>& xs)
27 {
28     memset(st, -1, sizeof st);
29
30     return dp(0, 0, 2*N, xs);
31 }
32
33 int main()
34 {
35     ios::sync_with_stdio(false);
36
37     int T;
38     cin >> T;
39
40     while (T--)
41     {
42         int N, K;
```

Solução $O(N \log N)$

```
43     cin >> N >> K;
44
45     set<int> xs;
46
47     for (int i = 0; i < K; ++i)
48     {
49         int x;
50         cin >> x;
51
52         xs.insert(x - 1);
53     }
54
55     auto ans = solve(N, xs);
56
57     cout << ans << '\n';
58 }
59
60 return 0;
61 }
```

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