

# Explicação Exercícios Programação Dinâmica 2

Exercícios A e D

# A - Deque

---

- É dada uma sequência com  $n$  valores.
- Existem dois jogadores que sempre vão jogar de maneira a maximizar seus pontos
- Um jogador ganha pontos quando retira uma das pontas do conjunto dado.
- O objetivo é fazer o primeiro jogador ter o máximo de pontos possíveis

# A - Deque

---

Exemplo:

**4 5 1 3**

Player 1: 0

Player 2: 0

# A - Deque

---

Exemplo:

**4** **5** **1** **3**

Player 1: 0 -> 4

Player 2: 0

# A - Deque

---

Exemplo:

**5** **1** **3**

Player 1: 4

Player 2: 0 -> 5

# A - Deque

---

Exemplo:

**1** **3**

Player 1: 4 -> 7

Player 2: 5

# A - Deque

---

Exemplo:

**1**

Player 1: 7

Player 2: 5 -> 6

# A - Deque

---

Exemplo: Melhor Solução

**4 5 1 3**

Player 1: 3

Player 2: 0



# A - Deque

---

Exemplo: Melhor Solução

**4** **5** **1**

Player 1: 3

Player 2: 4

# A - Deque

---

Exemplo: Melhor Solução

**5** **1**

Player 1: 8

Player 2: 4

# A - Deque

---

Exemplo: Melhor Solução

**1**

Player 1: 8

Player 2: 5

# A - Deque

---

- Temos duas escolhas:

Tirar o elemento da ponta esquerda

Tirar o elemento da ponta direita

- Recursão

Quando o player 1 joga eu tenho o máximo daquela jogada.

Quando o player 2 joga eu fico com o mínimo dessa jogada.

# A - Deque

---

Jogada do Player 1:

```
max(  
    item[início] + Recursiva(inicio+1, fim, !flag),  
    item[fim] + Recursiva(inicio, fim-1, !flag)  
)
```

# A - Deque

---

Jogada do Player 2:

```
min(  
    Recursiva(inicio+1, fim, !flag),  
    Recursiva(inicio, fim-1, !flag)  
)
```

# A - Deque

— — —

```
int brute(int ini, int fim, bool flag){
    if(ini > fim){
        return 0;
    }
    if(dp[ini][fim][flag] != -1) return dp[ini][fim][flag];
    int chamada1 = brute(ini + 1, fim, !flag);
    int chamada2 = brute(ini, fim - 1, !flag);

    if(flag) return dp[ini][fim][flag] = max(v[ini] + chamada1, v[fim] + chamada2);
    return dp[ini][fim][flag] = min(chamada1, chamada2);
}
```

# A - Deque

---

No final só precisa imprimir a soma dos itens menos o resultado da recursão.

**Resultado final = Soma\_Total - Resultado\_Recursão**



# A - Deque

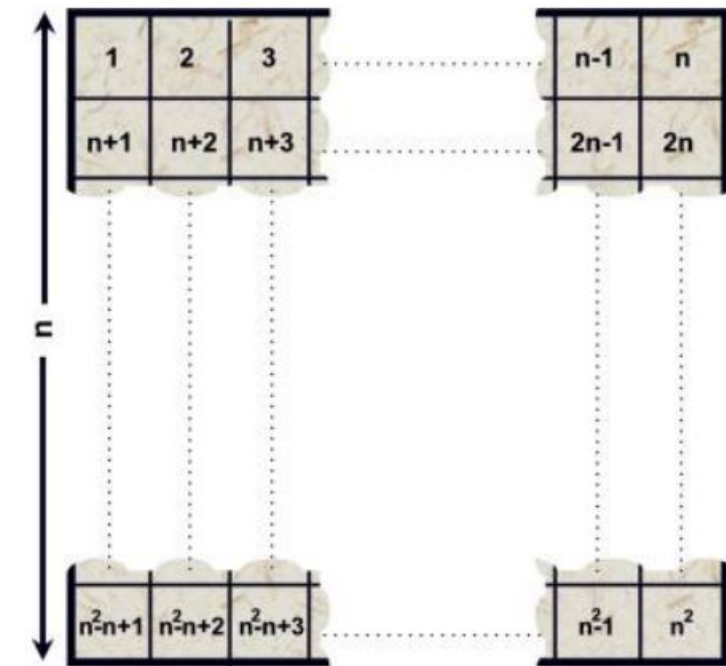
— — —

Sugestões:

[Removal Game - CSES](#)

[Cards - Online Judge \(UVA\)](#)

# D - Prince and Princess



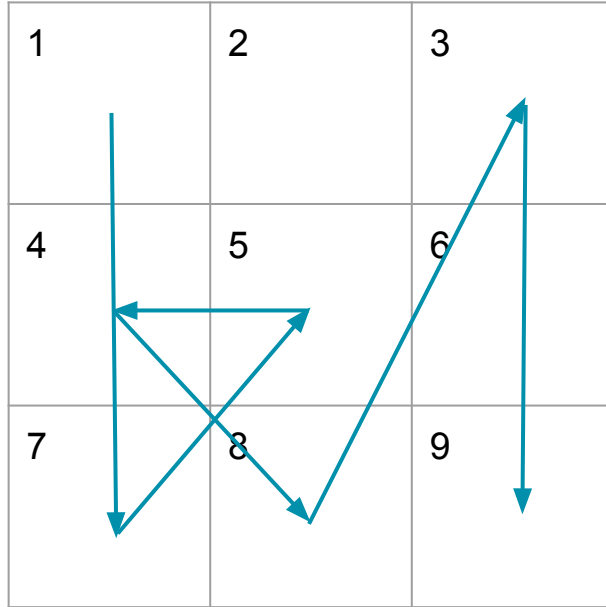
Em um tabuleiro  $n \times n$ , o número de cada casa é representado como na imagem.

Utilizando desse tabuleiro, o príncipe e a princesa decidem jogar um jogo:

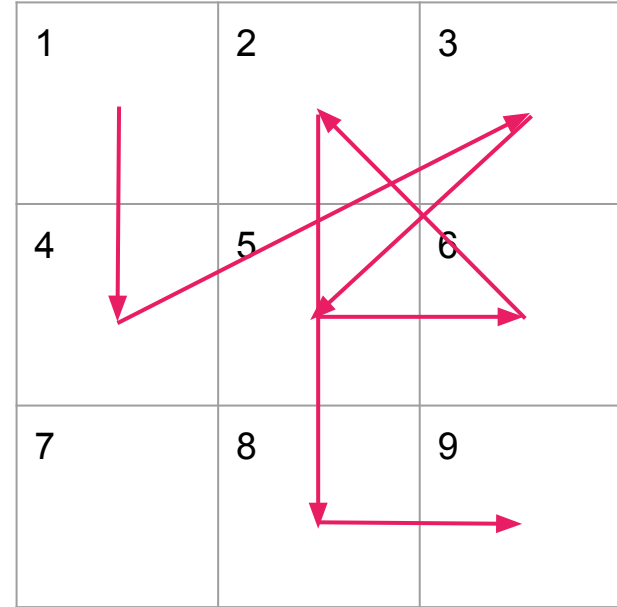
- O príncipe faz  $p$  pulos até chegar na casa  $n \times n$ , sendo todas as  $x_p$  casas diferentes.
- A princesa faz  $q$  pulos até chegar na casa  $n \times n$ , sendo todas as  $x_q$  casas diferentes.

# D - Prince and Princess

---



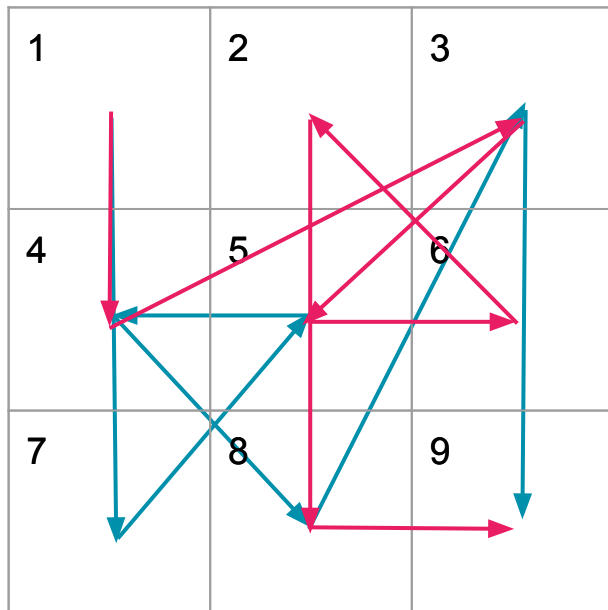
1 -> 7 -> 5 -> 4 -> 8 -> 3 -> 9



1 -> 4 -> 3 -> 5 -> 6 -> 2 -> 8 -> 9

# D - Prince and Princess

— — —



King: “Why move separately?”

1 -> 7 -> 5 -> 4 -> 8 -> 3 -> 9

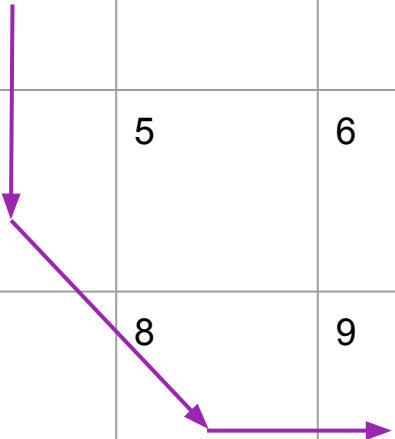
1 -> 4 -> 3 -> 5 -> 6 -> 2 -> 8 -> 9

Qual a maior rota que ambos podem  
fazer juntos?

# D - Prince and Princess

— — —

1	2	3
4	5	6
7	8	9



1 -> 7 -> 5 -> 4 -> 8 -> 3 -> 9

1 -> 4 -> 3 -> 5 -> 6 -> 2 -> 8 -> 9

**Maior subsequência em comum:**

1 -> 4 -> 8 -> 9

# D - Prince and Princess

---

## Algoritmo LCS?

Para conseguirmos utilizar o Algoritmo de LCS, precisamos ter disponível para o problema uma complexidade que permita realizar  $O(\text{tam\_sequencia1} * \text{tam\_sequencia2})$ .

Pela entrada do problema conseguimos perceber que a para cada sequência podemos ter no pior caso:

- $\text{tam\_sequencia1} = \text{tam\_sequencia2} = 250 * 250 = 62500$

Caso utilizemos LCS, teremos complexidade:

$$O(62500 * 62500) = 3.906.250.000 = 3 * 10^9$$

# D - Prince and Princess

---

## Algoritmo LCS?

Para conseguirmos utilizar o Algoritmo de LCS, precisamos ter disponível para o problema uma complexidade que permita realizar  $O(\text{tam\_sequencial} * \text{tam\_sequencia2})$ .

Pela entrada do problema conseguimos perceber que a para cada sequência podemos ter no pior caso:

- $\text{tam\_sequencial} = \text{tam\_sequencia2} = 250 * 250 = 62500$

**Time Limit Exceeded**

Caso utilizemos LCS, teremos complexidade:

$$O(62500 * 62500) = 3.906.250.000 = 3 * 10^9$$

**Vjudge:**



~heh

# D - Prince and Princess

— — —

No enunciado é informado que nas rotas de ambos, nenhuma posição se repete, sendo assim, podemos ordenar uma das sequências e fazer uma busca binária em relação às posições que ambas são iguais:

**Seq\_1 = 1 -> 7 -> 5 -> 4 -> 8 -> 3 -> 9**, ordenando e salvando suas posições relativas:

**Seq\_1 = (1,1) -> (3,6) -> (4,4) -> (5,3) -> (7,2) -> (8,5) -> (9,7)**



# D - Prince and Princess

— — —

**Seq\_1** = (1,1) -> (3,6) -> (4,4) -> (5,3) -> (7,2) -> (8,5) -> (9,7)

**Seq\_2** = 1 -> 4 -> 3 -> 5 -> 6 -> 2 -> 8 -> 9

**Seq\_nova** = 1, 4, 6, 3, 5, 7

Utilizando a ordem inicial da **Seq\_2**, realiza uma **busca binária** da sequência ordenada e salva a posição em relação à posição inicial da **Seq\_1**

Podemos então perceber que através da nova sequência gerada, podemos calcular a maior rota que ambos podem fazer utilizando o algoritmos de **LIS** (Longest Increasing Subsequence)

# D - Prince and Princess

---

```
vector<ll>seq_1(p), seq_2(q), seq_3;
map<int, int>posicao_inicial;
cin >> seq_1 >> seq_2;
posicao_inicial[seq_1[i]] = i;
//Salva em posicao_inicial as posicoes de seq_1 antes da ordenação
sort(seq_1)
for(i=0; i<seq_2.size(), i++){
    if(binary_search(seq_1.begin(), seq_1.end(), seq_2[i]))
        seq_3.push_back(posicao_inicial[seq_2[i]]);
}

cout << LIS(seq_3) << "\n";
```