



DRAGÃO E O GUERREIRO

- Arthur Gonçalves
- Assuerio Santos
- Lucas Braga
- Marcos Pablo
- Rodrigo Gonçalves
- Samuel Oliveira

1. INTRODUÇÃO

- Dado um determinado desafio, foi solicitada a solução utilizando a Teoria dos Grafos



2. O PROBLEMA SELECIONADO

O problema DRAGAOMG - Dragão de 100 Cabeças que pode ser encontrado no link <https://br.spoj.com/problems/DRAGAOMG/> foi o problema escolhido para o desenvolvimento da solução.



3. O PROBLEMA

- Com um golpe de espada o cavaleiro pode cortar um determinado número de cabeças.



3. O PROBLEMA

- Porém, para cada um desses golpes, novas cabeças nascem imediatamente.



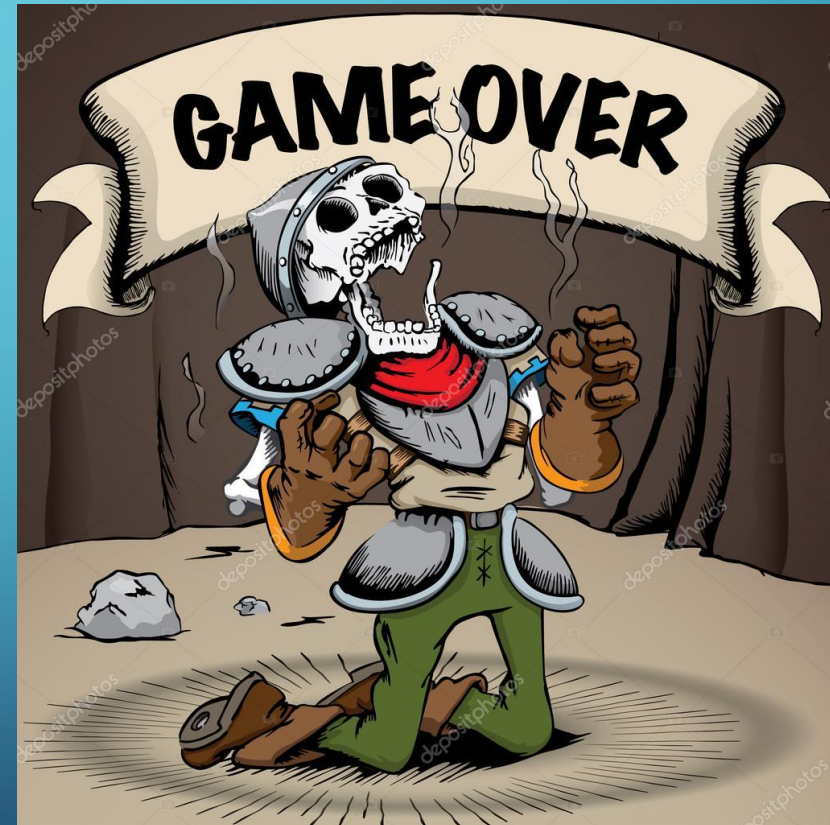
3. O PROBLEMA

- Quantos golpes o cavaleiro precisa para matar o dragão e assim salvar a princesa?



3.1. ALGUMAS OBSERVAÇÕES

- Se durante a sequência de golpes o dragão ficar com 1.000 cabeças ou mais, neste caminho o cavaleiro morre.



3.1. ALGUMAS OBSERVAÇÕES

- Caso o número de cabeças do dragão chegue a 0, neste caminho o cavaleiro mata o dragão. Não é possível dar um ataque que corte que corte mais cabeças do que o dragão possui.



3.2. ENTRADA DE DADOS

- Primeiramente, o algoritmo recebe a quantidade de total de rodadas por turno



3.2. ENTRADA DE DADOS

- Após isso, para cada rodada é informada a quantidade de cabeças a serem cortadas



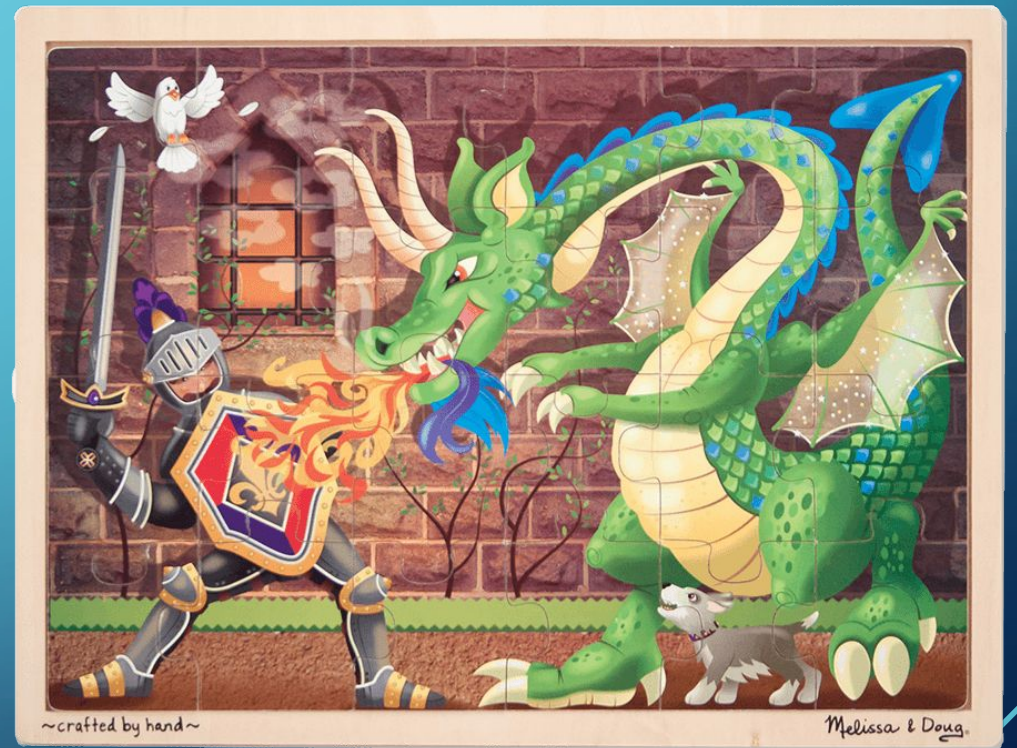
3.2. ENTRADA DE DADOS

- E para cada corte efetuado, é informado também a quantidade de nascimentos de cabeças



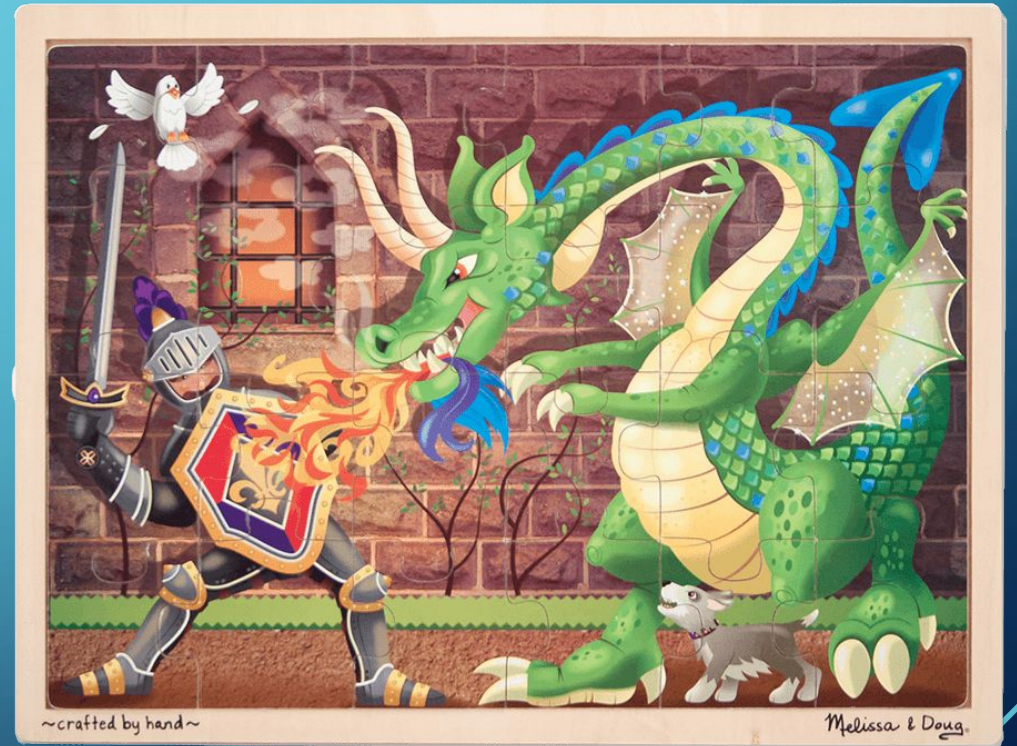
3.3. SAÍDA DE DADOS

- Portanto, com base nas entradas recebidas, o algoritmo deve informar com quantas rodadas o cavaleiro derrotou o dragão



3.3. SAÍDA DE DADOS

- Entretanto, caso o dragão atinja 1000 cabeças, o algoritmo deve informar que o cavaleiro morreu



The background is a blue gradient. In the corners, there are decorative white line art elements resembling circuit boards or neural networks, with lines and small circles.

MODELAGEM DO PROBLEMA

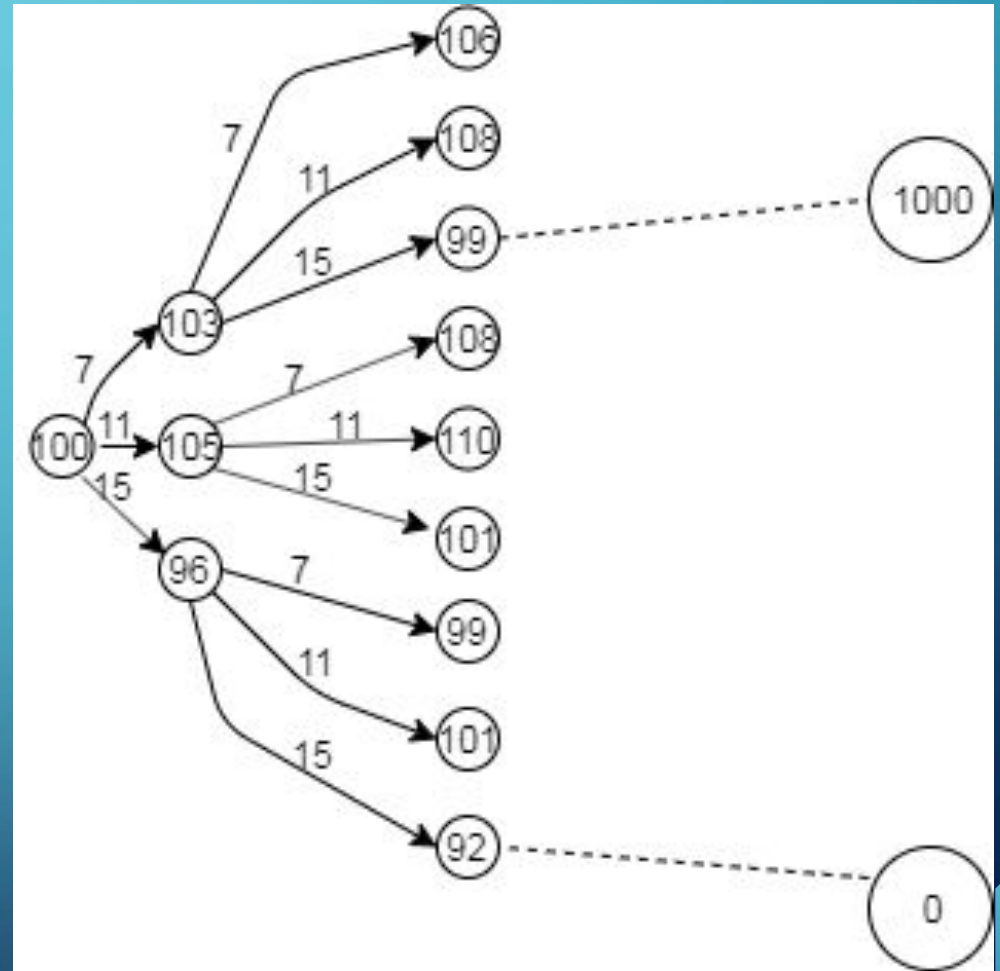
4.1. MODELAGEM INICIAL

- Como modelar um grafo que represente o problema apresentado?



4.1. MODELAGEM INICIAL

- Definimos como seriam representados os vértices e arestas



4.2. CODIFICAÇÃO

- Usamos um algoritmo recursivo que analisa todas as possibilidades da entrada

```
class MathHelper
{
public static string BigNumbersSum(string n1, string n2)
{
    string[] numbers = new string[2];
    int shortNumLength, longNumLength;

    InitBigNumbersSum(n1, n2, numbers, out shortNumLength, out longNumLength);
    return ComputeBigNumbersSum(numbers, shortNumLength, longNumLength);
}

private static void InitBigNumbersSum(string n1, string n2, string[] numbers)
{
    if (n1.Length > n2.Length)
    {
        numbers[0] = n1;
        numbers[1] = n2;
    }
    else
    {
        InitBigNumbersSum(n1, n2, numbers, out shortNumLength, out longNumLength);
        numbers[0] = n2;
        numbers[1] = n1;
    }
    longNumLength = numbers[0].Length;
    shortNumLength = numbers[1].Length;
}

private static string ComputeBigNumbersSum(string[] numbers, int shortNumLength, int longNumLength)
{
    StringBuilder result = new StringBuilder(longNumLength + 1);
    bool overTen = false;
    for (int i = 0; i < longNumLength; i++)
    {
        if (shortNumLength <= i)
        {
            int sum = (int)char.GetNumericValue(numbers[0][i]) + (int)char.GetNumericValue(numbers[1][i]);
            if (sum > 9)
            {
                overTen = true;
                sum -= 10;
            }
            result.Append(sum.ToString());
        }
        else
        {
            result.Append("0");
        }
    }
    if (overTen)
    {
        result.Append("1");
    }
    return result.ToString().Reverse().ToString();
}
}
```


4.2. CODIFICAÇÃO

- Tratamos dos casos específicos citados pelo problema, assim como evitamos a criação de loops e de um uso excessivo da memória

```
class MathHelper
{
public static string BigNumbersSum(string n1, string n2)
{
    string[] numbers = new string[2];
    int shortNumLength, longNumLength;

    InitBigNumbersSum(n1, n2, numbers, out shortNumLength, out longNumLength);
    return ComputeBigNumbersSum(numbers, shortNumLength, longNumLength);
}

private static void InitBigNumbersSum(string n1, string n2, string[] numbers)
{
    if (n1.Length > n2.Length)
    {
        numbers[0] = n1;
        numbers[1] = n2;
    }
    else
    {
        InitBigNumbersSum(n2, n1, numbers, out shortNumLength, out longNumLength);
    }
    return ComputeBigNumbersSum(numbers, shortNumLength, longNumLength);
}

private static string ComputeBigNumbersSum(string[] numbers, int shortNumLength, int longNumLength)
{
    StringBuilder result = new StringBuilder(longNumLength + 1);
    bool overTen = false;
    for (int i = 0; i < longNumLength; i++)
    {
        int sum = (int)char.GetNumericValue(numbers[0][i]) + (int)char.GetNumericValue(numbers[1][i]);
        if (sum > 9)
        {
            overTen = true;
            sum -= 10;
        }
        result.Append(sum.ToString());
    }
    if (overTen)
    {
        result.Append(1);
    }
    return result.ToString().Reverse().ToString();
}

private static void InitBigNumbersSum(string n1, string n2, string[] numbers, out int shortNumLength, out int longNumLength)
{
    if (n1.Length > n2.Length)
    {
        numbers[0] = n1;
        numbers[1] = n2;
        shortNumLength = n2.Length;
        longNumLength = n1.Length;
    }
    else
    {
        numbers[0] = n2;
        numbers[1] = n1;
        shortNumLength = n1.Length;
        longNumLength = n2.Length;
    }
}
```

4.3 VALIDAÇÃO DA SOLUÇÃO

ID	DATA	Usuário:	PROBLEM	RESULT	TIME	MEM	LING
24549506	2019-10-07 06:05:32	marcos_pablo	Dragão de 100 Cabeças	accepted edit run	0.35	17M	CSHARP

- Por fim, a solução desenvolvida foi submetida e aprovada em todos os casos de teste realizados pela plataforma

```
static void Start(int qtdCabeças, int passos)
{
    for(int i = 0; i < cortes.Length; i++)
    {
        if (cortes[i].qtdCabeçasCortadas <= qtdCabeças)
            Mutilar(qtdCabeças, i, passos + 1);
    }
}

static void Mutilar(int qtdCabeças, int corte, int passos)
{
    qtdCabeças -= cortes[corte].qtdCabeçasCortadas;
    if(qtdCabeças == 0)
    {
        if (passos < menorPasso)
            menorPasso = passos;
    }
    else
    {
        qtdCabeças += cortes[corte].qtdCabeçasNascidas;

        if (vet[qtdCabeças] != 0 && vet[qtdCabeças] <= passos)
            return;
        else
            vet[qtdCabeças] = passos;

        if (qtdCabeças >= 1000)
            return;

        for(int i = 0; i < cortes.Length; i++)
        {
            if (cortes[i].qtdCabeçasCortadas <= qtdCabeças)
                Mutilar(qtdCabeças, i, passos + 1);
        }
    }
}
```