PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Algoritmos em Grafos
Professor Kleber Jacques Ferreira de Souza

Arthur Andrade Gonçalves
Assuério Batista dos Santos
Lucas Braga Ferreira
Marcos Pablo Souza de Almeida
Rodrigo Gonçalves Ribeiro
Samuel Lucas Oliveira Martins

DRAGAOMG - Dragão de 100 Cabeças

Contagem
2019

SUMÁRIO

1. INTRODUÇÃO

Desenvolvidos a fim de solucionar problemas, os algoritmos em grafos apresentam soluções a diversas dificuldades enfrentadas no cotidiano. Essas quais podem ser definidas como encontrar a menor distância entre duas cidades, distribuir uma rede de computadores de forma otimizada, ou simplesmente encontrar qual a sequência correta de execução de tarefas que possuem interdependência entre si.

Sendo assim, utilizando os conceitos lecionados em classe, foi proposto o seguinte desafio:

O *DRAGAOMG*

Dragão de 100 Cabeças <https://br.spoj.com/problems/DRAGAOMG/>

2. PROBLEMA

" Em seu caminho para salvar uma princesa, um cavaleiro encontra um dragão de 100 cabeças. Com um golpe de espada ele pode cortar, por exemplo, 7, 11, ou 15 cabeças. Porém, para cada um desses golpes, novas cabeças nascem imediatamente (respectivamente 10, 16 e 11). Em outras palavras, se ele cortar 7 cabeças, nascem 10 novas. Se ele cortar 11 nascem 16, e se ele cortar 15 nascem 11. O dragão morre apenas se ficar sem cabeça, ou seja, se for aplicado um golpe que corta todas as cabeças restantes (nesse caso não nascem novas cabeças). Quantos golpes o cavaleiro precisa para matar o dragão e assim salvar a princesa?"

(fonte: <https://br.spoj.com/problems/DRAGAOMG/>)

2.1 Algumas observações

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

2.2 Entrada de Dados

Para que o algoritmo seja processado é necessário que tenha três entradas diferentes. A primeira deve ser informada um número inteiro da quantidade de golpes que o cavaleiro dará.

A segunda a quantidade de cabeças do dragão que serão cortas a cada golpe. A terceira a quantidade de cabeças que nascerão em cada um destes golpes.

2.3 Saída de dados

Para cada caso testado, o algoritmo verifica se nessa sequência de golpes o cavaleiro morreu. Caso tenha morrido é exibido na tela "cavaleiro morreu", caso seja possível chegar em um caminho que mate o dragão, é exibido o número de passos que o cavaleiro teve que passar para matar o dragão.

3. MODELAGEM DO PROBLEMA

3.1 Modelagem inicial

Ao deparar-se pela primeira vez com o desafio, o grupo encontrou dificuldades em estabelecer sua modelagem. Por conseguinte, utilizando os conceitos da Teoria dos Grafos, buscou-se modelar o problema de diversas formas, na qual a priori, a principal ideia era baseada em ter o corpo do dragão como um vértice central, e as cabeças como vértices adjacentes a ele, sendo vértices pendentes. Porém a proposta para esta modelagem estava dificultando a visualização real do problema. Com o auxílio do professor, o problema foi idealizado de forma diferente.

Nesta modelagem, nos preocupamos em ter vértices valorados com um número inteiro cada, que indica quantas cabeças há no dragão naquele instante num determinado caminho. Os vértices são gerados a cada interação de cortes do cavaleiro, com os valores de quantas cabeças o dragão está após o corte.

3.2 Codificação

Partindo para parte da codificação, a melhor maneira encontrada para solucionar foi utilizando a recursividade. Através deste método foi possível visualizar de maneira mais simples a solução. Os trechos de código a seguir, mostram base da ideia recursiva de implementação:

- Início da função recursiva:

```
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);
        }
    }
```

- Algoritmo recursivo:

```
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);
            }
        }
```

```
}
```

O algoritmo que foi implementado de maneira recursiva, calcula todos caminhos possíveis, de acordo com a entrada fornecida pelo usuário.

4. CONCLUSÃO

Utilizando os conceitos lecionados em classe, foi possível efetuar toda a modelagem e desenvolvimento do problema, alcançando uma solução com baixo nível de complexidade, que por conta disso possui baixo consumo de recursos computacionais. Sendo assim, a solução trafega por todos os caminhos existentes no grafo, eliminando os que chegam a vértices de mesmo valor com maior custo, efetuando uma Busca em profundidade do vértice inicial até o de valor zero caso seja possível.