Problema "Dona Minhoca" da Fase 3 da OBI2021 – Nível Sênior

Gabriel A. G. Sales[1], Pedro E. Santana[1], Guthyerri A. Barbosa[1], Davi S. Freitas[1], Wladimir A. Tavares[1]

[1]Universidade Federal do Ceara – Campus Quixadá (UFC)
Av. José de Freitas Queiroz, 5003 – Cedro, Quixadá – CE, 63902-580

{gabriel.alsamir, pedroems, galexandrino, davidossantosfreitas}@alu.ufc.br,
wladimirufc@gmail.com

Abstract. *This paper analyzes and describes a solution to the "Dona Minhoca" problem present in Phase 3 of the Senior Level of the 2021 Olímpiada Brasileira de Informática (OBI2021). The aim is to achieve an efficient solution that meets the requirements of consumed memory and maximum execution time required by the question. For this, an adapted use of a Depth-first Search Algorithm is made. Finally, the performance of the solution is presented.*

Resumo. *Este artigo analisa e descreve uma solução para o problema "Dona Minhoca" presente na Fase 3 do Nível Sênior da Olimpíada Brasileira de Informática de 2021 (OBI2021). O intuito é alcançar uma solução eficiente que atenda às exigências de memória consumida e tempo de execução máximo exigidas pela questão. Para isso, faz-se uso adaptado de um Algoritmo de Busca em Profundidade. Por fim, é apresentada a performance da solução.*

1. Introdução

A Olimpíada Brasileira de Informática (OBI) é uma competição de programação que, seguindo os moldes das demais olimpíadas científicas de conhecimento, visa despertar o interesse dos alunos pelo estudo da lógica matemática, ciência da computação e programação [SBC 2022]. Realizada anualmente pela Sociedade Brasileira de Computação (SBC) e organizada pelo Instituto de Computação da Unicamp desde 1999, a Olimpíada é dividida em Modalidade Iniciação e Modalidade Programação (a qual o problema aqui descrito pertence). Durante a competição, os competidores devem colocar os problemas em ordem de dificuldade, deduzir requisitos dos problemas, construir casos de testes e propor uma solução algorítmica que resolva os problemas conforme os limites de tempo e memória propostos pelos elaboradores dos problemas. Dessa forma, cada problema exige conhecimento prático e teórico dos alunos, além de criatividade e capacidade de adaptar algoritmos conhecidos para os desafios em questão.

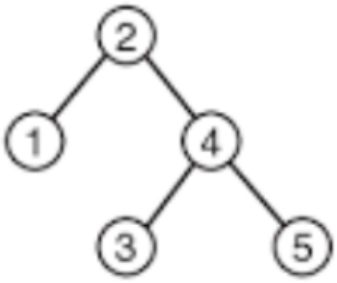
Nesse sentido, o presente trabalho tem por objetivo trazer um estudo tanto teórico quanto prático do problema "Dona Minhoca", presente na Fase 3 da XXIII OBI, em 2021, no Nível Sênior. Para resolver o problema, precisamos contar quantos caminhos de tamanho máximo existem em uma árvore.

2. Fundamentação Teórica

Um *grafo simples* G é um par ordenado $G = (V(G), E(G))$, composto por um conjunto finito $V(G) = \{1, 2, \ldots, n\}$, cujos elementos são chamados de *vértices*, e por

um conjunto de pares não ordenados $E(G) \subseteq \{\{u, v\} : u, v \in V, u \neq v\}$, cujos elementos são chamados de *arestas*. A partir de agora, vamos representar o conjunto $\{u, v\}$ simplesmente por uv ou vu. Definimos por $n = |V(G)|$ e $m = |E(G)|$. Os vértices $u, v \in V(G)$ são chamados *adjacentes (ou vizinhos)* se $uv \in E(G)$ e *não-adjacentes* se $uv \notin E(G)$. O conjunto dos *vizinhos* de um vértice v é denotado por $N(v)$. O *grau* de um vértice v em um grafo G, denotado por $d_G(v)$, corresponde ao número de vizinhos de v no grafo G, isto é $|N(V)|$.

Dado um grafo G, um caminho de u até w em G é uma sequência de vértices e arestas partindo de u até w. Um grafo é dito conexo se existe um caminho entre qualquer par de vértice. Um caminho de v até v é ciclo se todas as arestas são distintas e somente o vértice v aparece duas vezes. Um grafo é dito acíclico quando não contém nenhum ciclo. Um grafo conexo e acíclico é chamado de árvore. Um vértice é chamado de folha quando o seu grau é igual a 1.



Na Figura 3, temos o exemplo de um grafo acíclico e conexo com 5 vértices e 4 arestas. A excentricidade ϵ de um vértice v, denotado por $\epsilon(v)$, é a maior distância entre v e qualquer outro vértice. O diâmetro de um grafo é a excentricidade máxima de qualquer vértice do grafo. Ou seja, ele é a maior distância entre qualquer par de vértices. O raio de um grafo, denotado por $rad(G)$, é a excentricidade mínima do grafo de qualquer vértice do grafo em um gráfico.

A excentricidade de cada vértice é:

v	1	2	3	4	5
$\epsilon(v)$	3	2	3	2	3

O centro do grafo é um conjunto de vértices com excentricidade igual ao raio do grafo:

$\mathcal{C}(G) = \{v | \epsilon(v) = rad(G)\}$

Teorema 1. *Seja L o conjunto de folhas de G. Se $|V| \leq 2$ então L é o centro de G, caso contrário o centro do grafo G permanece o mesmo após a remoção de L*

$\mathcal{C}(G) = \mathcal{C}(G \setminus L)$

Este teorema nos leva ao seguinte algoritmo: remova as folhas, nível por nível, até que haja ≤ 2 nós. Esses nós serão o centro do grafo.

Na Figura 3, o conjunto de nós folhas $L = \{1, 3, 5\}$. Após a remoção destes vértices, sobram apenas 2 vértices. Estes vértices são o centro do grafo. Logo, $\mathcal{C}(G) = \{2, 4\}$

O diâmetro do grafo pode ser calculado da seguinte maneira:

Algorithm 1 Encontrando o diâmetro de G

function DIAMETRO(G)

Escolha um vértice arbitrário p como o nó raiz.
Execute uma busca em profundidade a partir de p e seja a o nó mais distante de p
Execute uma busca em profundidade a partir de a e seja b o nó mais distante de a.
Devolva a distância entre a e b é o diâmetro do grafo.

Durante a busca em profundidade, construímos um grafo direcionado chamado grafo dos predecessores $G_p(V, E_p)$ onde $E_p = \{(pred[v], v) | v \in V \text{ e } pred[v] \neq NIL\}$ A busca em profundidade pode ser modificada para devolver o vértice mais distante e a distância para ele.

O Algoritmo 2 recebe como entrada um grafo G e um vértice $raiz$ que será a raiz da árvore de busca em profundidade. Inicialmente, o predecessor de todos os nós começam com NIL. O Algoritmo 4 devolve a distância do vértice mais distante de u e o vértice mais distante.

Algorithm 2 Algoritmo de Busca em Profundidade

```
function DFS(G, raiz)
    for v ∈ V(G) do
        pred[v] ← NIL
    DFS_VISIT(raiz)
```

Algorithm 3 Algoritmo DFS_VISIT

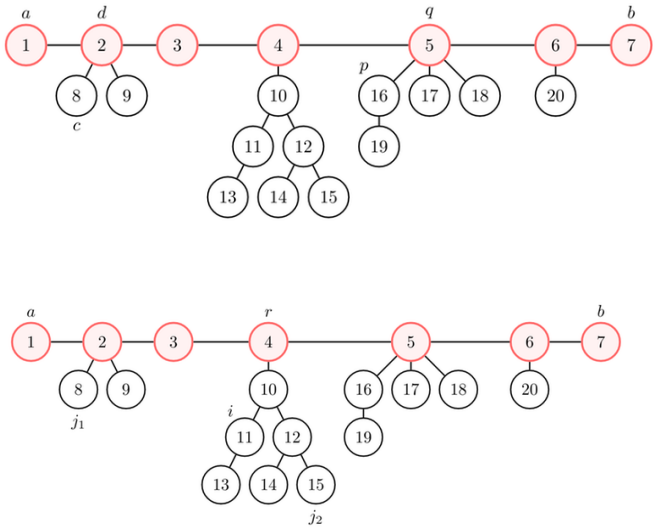
```
function DFS_VISIT(G, u)
    res ← (1, u)
    for v ∈ Adj[u] do
        if v ≠ pred[v] then
            pred[v] ← u
            (d, x) ← DFS_VISIT(G, v)
            res ← max(res, (d + 1, x))
    return res
```

A prova apresentada neste artigo pode ser encontrada neste link [1]. Antes de provar a corretude do algoritmo 1, vamos analisar a estrutura da árvore. Podemos representar o diâmetro em uma linha. Cada vértice nó no diâmetro será a raiz de uma árvore. É fácil mostrar que a altura de cada componente com a raiz é no máximo a distância da raiz do componente até a extremidade mais próxima do diâmetro. Por exemplo, se $d(d, c) \geq d(a, d)$ então o diâmetro da árvore seria $d(c, b)$.

Teorema 2. *Para cada nó i, seja o nó j tal que $dist(i, j)$ seja máximo, então $j = a$ ou $j = b$*

- Se $j = j1$ (suponha que $j1$ não está no mesmo componente de i; suponha sem perda de generalidade que $j1$ está mais próximo de a do que de b), $dist(i, j1) = dist(i, r) + dist(r, j1) \leq dist(i, r) + dist(r, a) = dist(i, a)$. Então, $j = a$.

[1] `https://codeforces.com/blog/entry/101271`



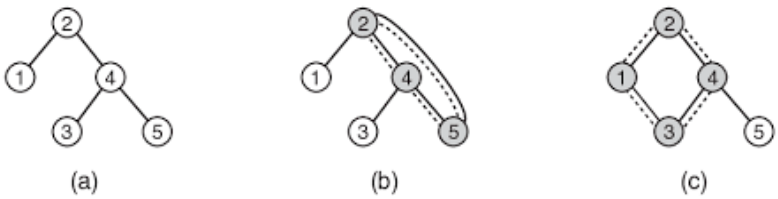
- Se $j = j2$ ($j2$ está no mesmo componente de i), $dist(i, j1) \leq dist(i, r) + dist(r, j1) \leq dist(i, r) + dist(r, a) = dist(i, a)$. Então, $j = a$.

Teorema 3. *dist(a,b) é o diâmetro do grafo G.*

Proof. Suponha que dist(u,v) seja um diâmetro. Pelo teorema anterior, a e b são os nós mais distantes de qualquer nó, ou seja, $dist(u, a) \geq dist(u, v)$ ou $dist(u, b) \geq dist(u, v)$. Vamos supor sem perda de generalidade que $dist(u, b) \geq dist(u, v)$. Obtemos $dist(a, b) \geq dist(u, b) \geq dist(u, v)$, então dist(a,b) é um diâmetro de G. □

3. Descrição do Problema

Dona Minhoca construiu uma bela casa, composta de N salas conectadas por N-1 túneis. Cada túnel conecta exatamente duas salas distintas, e pode ser percorrido em qualquer direção. A casa de dona Minhoca foi construída de modo que, percorrendo os túneis, é possível partir de qualquer sala e chegar a qualquer outra sala da casa. Dona Minhoca quer se exercitar, e para isso planeja construir um túnel adicional, de modo a criar um ”ciclo” de salas e túneis. Vamos chamar de comprimento do ciclo o número de salas do ciclo. A figura (a) abaixo mostra um exemplo de casa. É possível obter um ciclo de comprimento três construindo um túnel entre as salas 2 e 5, ou um ciclo de comprimento quatro construindo um túnel entre as salas 1 e 3 [OBI 2022].



Note que o grafo de entrada do problema trata-se de um grafo acíclico e conexo, ou seja, uma árvore. O problema consiste em contar o número de caminho de tamanho igual ao diâmetro do grafo. No exemplo da Figura 4, o diâmetro do grafo é 4. Temos os seguintes caminhos de comprimento igual a 4: 1-2-4-3, 1-2-4-5.

4. Solução do Problema

Propomos as duas soluções a seguir:

1. O Teorema 1 traz a ideia de um algoritmo para encontrar os vértices centrais do grafo G e ao encontrar o vértice central, achamos também o raio da árvore.
   - Suponha que G possui dois vértices centrais u e v. Seja d o diâmetro da árvore dado por $2 * raio$. Seja $cont1$ a quantidade de caminhos partindo de u com distância $\frac{d}{2} - 1$ e $cont2$ a quantidade de caminhos partindo de v com distância $\frac{d}{2} - 1$. Logo, a quantidade total de caminhos com distância igual a d é $cont1 * cont2$.
   - Suponha que G possui apenas um vértice central chamado de u. Para cada vértice i em $N(u)$. Seja d o diâmetro da árvore dado por $2 * raio + 1$. Seja $cont[i]$ a quantidade de caminhos partindo de u passando por v com distância igual a $\frac{d}{2}$. Logo, a quantidade total de caminhos com distância igual a d é:

     $$\sum_{i=1}^{|N(u)|} (\sum_{j=1}^{i-1} cont[j]) \cdot cont[i]$$

     Por exemplo, se u possui três vértices vizinhos com 2, 3 e 2 caminhos com distância igual a $\frac{d}{2} - 1$. Seja $\{c1, c2\}$,$\{c3, c4, c5\}$ e $\{c6, c7\}$ os caminhos encontrados pelos vértices vizinhos de u. É fácil ver que juntando um caminho do grupo 1 com um caminho do grupo 2 criamos um caminho de distância igual a d, ou seja, $\{c1c3, c1c4, c1c5, c2c3, c2c4, c2c5\}$. É fácil ver que juntando qualquer caminho do grupo1 e grupo2 com um caminho do grupo3, encontramos um caminho de distância igual a d.
2. A execução do algoritmo 1 devolve o diâmetro d da árvore. Uma pequena alteração do algoritmo 3 permite obter o ciclo de maior comprimento e a quantidade de maneiras possíveis de construir um ciclo com esse comprimento a partir da execução do algoritmo 2.

Algorithm 4 Algoritmo DFS_Dona_Minhoca

```
function DFS_DONA_MINHOCA(G, u, diam)
    res ← (1, 1)
    if res[0] = diam then
        total ← total + res[1]
    for v ∈ Adj[u] do
        if v ≠ pred[v] then
            pred[v] ← u
            (d, x) ← DFS_DONA_MINHOCA(G, v)
            if d + res[0] = diam then
                total ← total + x * res[1]
            d ← d + 1
            if d > res[0] then
                res = (d, x)
            else if d = res[0] then
                res[1] ← res[1] + x
    return res
```

5. Conclusão e Resultados

Os dois algoritmos foram executados em uma máquina com o processador Intel Core i5-8265U, 1.60 GHz, 4 núcleos e 8 GB de memória, ambos implementados na linguagem C++.

	CPU time	
	Algorithm 2	Algorithm 1
$n = 1000$	0.464768_{ms}	0.193522_{ms}
$n = 4000$	1.56873_{ms}	0.622468_{ms}
$n = 9471$	2.73141_{ms}	1.96355_{ms}
$n = 9861$	3.57227_{ms}	1.9604_{ms}
$n = 20001$	4.38392_{ms}	3.14109_{ms}
$n = 27742$	11.9688_{ms}	6.49107_{ms}
$n = 41372$	13.2284_{ms}	11.6983_{ms}
$n = 41919$	13.5924_{ms}	16.7391_{ms}
$n = 41948$	12.1335_{ms}	11.5757_{ms}

O primeiro algoritmo executa 3 algoritmos de Busca em Profundidade no objetivo de percorrer o grafo e alcançar os vértices centrais do grafo G. Sabendo que E = O(n), sendo E o valor referente ao número de vértices do grafo. Dessa forma, sabendo que a complexidade de cada DFS executada é de O(E), a complexidade temporal final, com as constantes em consideração; será de $3 * O(E)$. Em relação ao segundo algoritmo, como há uma procura atingir a distância do vértice mais ditante de u e o próprio vértice, a complexidade geral assemelha-se ao algoritmo anterior, contudo com a diferença de que o valor da constante intercala entre 2 e 3 a depender da entrada fornecida, dessa forma variando entre $2 * O(E)$ e $3 * O(E)$.

De forma geral é possível afirmar que ambos os algoritmos contêm complexidade linear, a diferenciar somente pelos valores de suas constantes para cada chamada de uma Busca em Profundidade.

6. Agradecimentos

Agradecemos ao nosso orientador, Wladimir, e ao Paulo Miranda, finalista da Maratona de Programação pela UFC Quixadá, pela ajuda e motivação.

References

OBI (2022). Pratique - dona minhoca. `https://olimpiada.ic.unicamp.br/pratique/ps/2021/f3/minhoca/`. Acesso em: 01 out. 2022.

SBC (2022). Olimpiada brasileira de informática. `https://www.sbc.org.br/eventos/competicoes-cientificas/7-obi-evento`. Acesso em: 25 set. 2022.