2. Algoritmos de Busca

|  |
| --- |
| **Questão 2.1** |
| Analise os problemas a seguir e indique se o mesmo pode ser resolvido por busca em largura, em profundidade ou pelos dois tipos de busca. |
| **Problema 1:** O matemático húngaro Paul Erdos (1913-1996), um dos mais brilhantes do século XX, é considerado o mais prolífico matemático da história. Erdos publicou mais de 1500 artigos, em colaboração com cerca de outros 450 matemáticos. Em homenagem a este gênio húngaro, os matemáticos criaram um número denominado "número de Erdos". Toda pessoa que escreveu um artigo com Erdos tem o número 1. Todos que não possuem número 1, mas escreveram algum artigo juntamente com alguém que possui número 1, possuem número 2. E assim por diante. Quando nenhuma ligação pode ser estabelecida entre Erdos e uma pessoa, diz-se que esta possui número de Erdos infinito. Por exemplo, o número de Erdos de Albert Einstein é 2. O número de Erdos de Bill Gates é 4. |
| **(  X  )** busca em largura **(     )** busca em profundidade**(     )** ambos tipos de busca |
| **Problema 2:** O corpo de bombeiros de uma cidade está testando uma nova tecnologia para auxiliar no salvamento de pessoas. O sistema consiste de um drone que fica parado em uma estrutura à uma distância x da costa. De tempos em tempos esse drone saí e vai verificar pontos em direção à costa. Querendo que as regiões mais fundas sejam atingidas antes das regiões mais rasas, o drone foi implementado com um algoritmo que garante esse comportamento. |
| **(    )** busca em largura **(  X  )** busca em profundidade**(     )** ambos tipos de busca |
| **Problema 3:** A rede elétrica de uma cidade foi construída de forma que todos os bairros tenham energia elétrica. Os bairros são interligados por grupos, ou seja, existe um grupo de bairros que estão ligados na mesma rede elétrica. O problema desse tipo de ligação é que, se a rede cair em qualquer um dos pontos do grupo, todos os bairros ligados àquela rede com problema terão a energia cortada. A companhia de eletricidade possui um sistema que verifica de tempo em tempos o status da rede. No sistema é possível identificar quais os grupos de bairros associados e se neste grupo a rede está em perfeitas condições. |
| **(  X  )** busca em largura **(     )** busca em profundidade**(     )** ambos tipos de busca |
| **Problema 4:** Deseja-se criar um sistema que dado uma foto qualquer, procure em uma rede social um determinado usuário a partir da foto passada. A proposta é começar a busca a partir da pessoa que está conectada na rede social e comparar a imagem com todos os seus amigos. Em seguida, para cada amigo, comparar todos os amigos deste amigo e assim por diante. |
| **(  X  )** busca em largura **(     )** busca em profundidade**(     )** ambos tipos de busca |

|  |
| --- |
| **Questão 2.2** |
| A busca em largura e a busca em profundidade são dois dos principais conceitos do estudo de Teoria dos Grafos. Os pseudocódigos a seguir representam as duas buscas que servem de base para uma série de algoritmos dentro da teoria dos grafos. Analise os pseudocódigos e reescreva-os para que atendam o que se pede em cada item. Para cada item, basta mostrar a resposta para um algoritmo. Faz parte da atividade que o aluno interprete o que se pede e decida qual dos dois algoritmos é mais adequado.  A screenshot of a social media post  Description automatically generated |
| **Item 2.1:** Modifique um dos algoritmos para que permita verificar se um grafo é conexo ou não. Se não for conexo, o algoritmo deve retornar a quantidade de componentes conexos do grafo. |
| Implementação inicial:  static void BuscaLargura<T>(Vertex<T> verticeInicial) {  var fila = new Queue<Vertex<T>>();  verticeInicial.Visited = true;  fila.Enqueue(verticeInicial);    do  {  var vertice = fila.Dequeue();   foreach (var vertex in vertice.Neighbors.Where(v => !v.Visited))  {  vertex.Visited = true;  fila.Enqueue(vertex);  }  } while (fila.Any()); }  Modificado:  static int BuscaLargura<T>(UndirectedGraph<T> grafo, Vertex<T> verticeInicial = null) {  verticeInicial ??= grafo.Vertices.First();  var conexosCount = 1;  var fila = new Queue<Vertex<T>>();  verticeInicial.Visited = true;  fila.Enqueue(verticeInicial);    do  {  var vertice = fila.Dequeue();   foreach (var vertex in vertice.Neighbors.Where(v => !v.Visited))  {  vertex.Visited = true;  fila.Enqueue(vertex);  conexosCount++;  }  } while (fila.Any());    return grafo.Vertices.Count != conexosCount ? conexosCount : 0; } |
| **Item 2.2:** Modifique um dos algoritmos de tal forma que o mesmo permita ordenar os vértices de forma decrescente pela distância deste vértice para origem. Ao final, o algoritmo deve mostrar os vértices ordenados na ordem correta. |
| static int BuscaLargura<T>(UndirectedGraph<T> grafo, Vertex<T> verticeInicial = null) {  verticeInicial ??= grafo.Vertices.First();  var conexosCount = 1;  var nivelCount = 0;  var fila = new Queue<Vertex<T>>();  verticeInicial.Visited = true;  fila.Enqueue(verticeInicial);  var ordenado = new Dictionary<int, ICollection<Vertex<T>>>();    do  {  var vertice = fila.Dequeue();    ordenado.Add(nivelCount++, new List<Vertex<T>>());  ordenado[nivelCount].Add(vertice);   foreach (var vertex in vertice.Neighbors.Where(v => !v.Visited))  {  vertex.Visited = true;  fila.Enqueue(vertex);  conexosCount++;  }  } while (fila.Any());   for (int i = 0; i < nivelCount; i++)  {  Console.WriteLine($"{i}: ");  foreach (var vertex in ordenado[i])  {  Console.Write($"{vertex.Value.ToString()}\t");  }  Console.WriteLine();  }    return grafo.Vertices.Count != conexosCount ? conexosCount : 0; } |