GRAF Test-Eksamen (OPGAVE)

**Opgave 1**Metoden addVertex er ikke helt optimal, fordi den tillader at flere Vertex med samme navn kan indsættes. Du skal ændre koden således at man kun må bruge et navn én gang.

**Opgave 2**

|  |  |  |
| --- | --- | --- |
| Du skal indsætte følgenden metode i Graph klassen:   public void randomTraversal().  Metoden skal starte et tilfældigt sted og flytte sit tilfældigt i grafen. Lad metoden køre igennem 10 Vertex.  Her til højre kan du se 2 af mine køresler. |  |  |

**Opgave 3**

|  |  |  |
| --- | --- | --- |
| Du skal indsætte følgenden metode i Graph klassen:   **public void countConnectionToVertex()** Den skal udskrive antal forbindelser som en node har IND til sig (ikke ud fra sig).  Her til højre kan du se min udskrift. Den siger at A ikke har nogle forbindelser ind til sig. Vi ved at A har flere forbindelser ud fra sig, men det er kun IND vi har interesse i her. | Sammenlign denne udskrift med grafen til højre, så ser du at det hænger sammen. A har ingen pile ind til sige. H har 3 pile ind til sig, D har kun en, osv. |  |

|  |
| --- |
| **UDLEVERET-KODE** |
|  |
| using System;  using System.Collections.Generic;  namespace Graph\_3  {  class Program  {  static void Main(string[] args)  {  Graph graf = new Graph();  graf.addVertex("A");  graf.addVertex("B");  graf.addVertex("C");  graf.addVertex("D");  graf.addVertex("E");  graf.addVertex("F");  graf.addVertex("G");  graf.addVertex("H");  graf.addVertex("I");  graf.addEdge("A", "B", 2); //var 2  graf.addEdge("A", "D", 5);  graf.addEdge("A", "E", 4);  graf.addEdge("B", "E", 1);  graf.addEdge("D", "G", 2);  graf.addEdge("E", "F", 3);  graf.addEdge("E", "H", 6);  graf.addEdge("F", "C", 4);  graf.addEdge("F", "H", 3);  graf.addEdge("C", "B", 3);  graf.addEdge("H", "I", 1);  graf.addEdge("I", "F", 1); // var 1,  graf.addEdge("G", "H", 1);  Console.WriteLine("------------Breath-First Traversal Graph 1.----------");  graf.BreathFirstTraversal("A");  graf.setAllToNotVisited();  Console.WriteLine("------------Depth-First Traversal Graph 1.----------");  graf.DepthFirstTraversal("A");  graf.setAllToNotVisited();  Console.WriteLine("------------Dijkstra.----------");  graf.setAllToNotVisited();  graf.dijkstra("A", "H");  }  }  class Graph  {  List<Vertex> grafList = new List<Vertex>();  public void dijkstra(String navnPaaStartVertex, String navnPaaStopVertex)  {  Console.WriteLine();  Console.WriteLine("--------------------- From " + navnPaaStartVertex + " to " + navnPaaStopVertex + " ----------------");  Console.WriteLine();  List<Vertex> priorityQueue = new List<Vertex>();  Vertex startVertex = getVertex(navnPaaStartVertex);  startVertex.visited = true;  startVertex.lengthFromStartVertex = 0;  priorityQueue.Add(startVertex);  Vertex frontVertex = startVertex;  Boolean search = true;  while (priorityQueue.Count > 0 && search)  {  //---- Prioritizing from the Queue-----  int value = 1000000000;  int teller = 0;  foreach (Vertex v in priorityQueue)  {  if (v.lengthFromStartVertex < value)  {  frontVertex = priorityQueue[teller];  value = v.lengthFromStartVertex;  }  teller++;  }  priorityQueue.Remove(frontVertex);  frontVertex.visited = true;  Console.WriteLine(" <- Dequeue: " + frontVertex.navn + " with weight: " + frontVertex.lengthFromStartVertex + " Queue size:" + priorityQueue.Count);  //----Prioritizing STOP-------------------------------------------  //-----We have found the Vertex-------------------------------------  if (frontVertex.navn.Equals(navnPaaStopVertex))  {  Console.WriteLine(" ---------- Found: Name: " + frontVertex.navn + " Weight: " + frontVertex.lengthFromStartVertex);  Vertex holder = frontVertex;  while (!holder.navn.Equals(navnPaaStartVertex))  {  Console.WriteLine("backtracking: " + holder.navn);  holder = holder.previousVertex;  }  Console.WriteLine("backtracking: " + navnPaaStartVertex);  search = false;  break;  }  //--------------------------------------  foreach (Edge e in frontVertex.edgeList)  {  // lidt indviklet: Først må vi tjekke for at vertexen ikke allerede er  // besøgt. Det er den første if sætning. Siden må vi tjekke om vægten  // således at vi ikke overskriver en Vertex som allerede er i køen, og  // som har en mindre vægt. Har den mindre vægt, så skal den nye ikke med  // i køen. Men om den ny nu har mindre vægt end en som allerede er i køen,  // så overskriver vi denne, i stedet for at lave en ny reference, det er if else  // sætningen.  int w = (int)e.weight;  if (e.vertex.visited == false) //Den er ikke besøgt før, så skal den med.  {  Boolean laengdeOK = true;  Boolean mindreOK = false;  foreach (Vertex v in priorityQueue)// Hvis længden er større på den ny, skal  { // den heller ikke med her.  if (v.navn.Equals(e.vertex.navn))  {  if (frontVertex.lengthFromStartVertex + w >= v.lengthFromStartVertex)  {  laengdeOK = false;  }  else  {  mindreOK = true;  }  }  }  if (laengdeOK == true && mindreOK == false)  {  e.vertex.lengthFromStartVertex = frontVertex.lengthFromStartVertex + w;  e.vertex.previousVertex = frontVertex;  priorityQueue.Add(e.vertex);  Console.WriteLine(" -> Enqueing: " + e.vertex.navn + " Weight: " + e.vertex.lengthFromStartVertex + " Previous Vertex: " + e.vertex.previousVertex.navn + " Queue size:" + priorityQueue.Count);  }  else if (laengdeOK == true && mindreOK == true)// Så er Vertex´en allered i Køen, Så overskriver vi bare,  { // i stedet for at lægge endnu en reference af samme objekt i køen.  Vertex v = getVertex(e.vertex.navn);  v.lengthFromStartVertex = frontVertex.lengthFromStartVertex + w;  v.previousVertex = frontVertex;  Console.WriteLine(" \* Overskriver Vertex " + e.vertex.navn + " Weight: " + e.vertex.lengthFromStartVertex);  }  else if (laengdeOK == false && mindreOK == false)  {  Console.WriteLine(" \* Findes allerede og har mindre eller samme vægt: " + e.vertex.navn);  }  }  else  {  Console.WriteLine(" \* Besøgt Vertex før: " + e.vertex.navn);  }  }  }  }  public void findShortesPathUnweightedGraph(String navnPaaStartVertex)  {  Queue<Vertex> vertexQueue = new Queue<Vertex>();  Queue<Vertex> traversalOrder = new Queue<Vertex>();  Vertex startVertex = getVertex(navnPaaStartVertex);  startVertex.visited = true;  startVertex.lengthFromStartVertex = 0;  vertexQueue.Enqueue(startVertex);  traversalOrder.Enqueue(startVertex);  Vertex frontVertex;  while (vertexQueue.Count > 0)  {  frontVertex = vertexQueue.Dequeue();  foreach (Edge e in frontVertex.edgeList)  {  if (e.vertex.visited == false)  {  e.vertex.visited = true;  e.vertex.lengthFromStartVertex = frontVertex.lengthFromStartVertex + 1;  e.vertex.previousVertex = frontVertex;  vertexQueue.Enqueue(e.vertex);  traversalOrder.Enqueue(e.vertex);  }  }  }  Console.WriteLine("-------- Shortes Path in an Unweighted Graph-------------");  Console.WriteLine("-------- From vertex: " + navnPaaStartVertex + " to all other rechable Vertex-------");  foreach (Vertex v in traversalOrder)  {  Console.Write("Name: " + v.navn + " Distance: " + v.lengthFromStartVertex + " - ");  }  Console.WriteLine();  }  public Boolean isThereAConnection(String a, String b)  {  Boolean fundet = false;  Queue<Vertex> vertexQueue = new Queue<Vertex>();  Queue<Vertex> traversalOrder = new Queue<Vertex>();  Vertex startVertex = getVertex(a);  startVertex.visited = true;  vertexQueue.Enqueue(startVertex);  traversalOrder.Enqueue(startVertex);  Vertex frontVertex;  while (vertexQueue.Count > 0)  {  frontVertex = vertexQueue.Dequeue();  foreach (Edge e in frontVertex.edgeList)  {  if (e.vertex.visited == false)  {  vertexQueue.Enqueue(e.vertex);  traversalOrder.Enqueue(e.vertex);  e.vertex.visited = true;  if (e.vertex.navn.Equals(b))  {  fundet = true;  }  }  }  }  return fundet;  }  public void addVertex(string navn)  {  Vertex v = new Vertex(navn);  grafList.Add(v);  }  public void addEdge(String VertexBegin, String VertexEnd, double edgeWeight)  {  Edge edge = new Edge();  edge.vertex = getVertex(VertexEnd);  edge.weight = edgeWeight;  getVertex(VertexBegin).edgeList.Add(edge);  }  public Vertex getVertex(String navn)  {  Vertex gemmer = null;  foreach (Vertex vertex in grafList)  {  if (vertex.navn.Equals(navn))  {  gemmer = vertex;  }  }  return gemmer;  }  public void setAllToNotVisited()  {  foreach (Vertex v in grafList)  {  v.visited = false;  }  }  public void printGraph()  {  foreach (Vertex v in grafList)  {  foreach (Edge e in v.edgeList)  {  Console.WriteLine("From: " + v.navn + " To: " + e.vertex.navn + " weight: " + e.weight);  }  }  }  public void BreathFirstTraversal(String navnPaaStartVertex)  {  Queue<Vertex> vertexQueue = new Queue<Vertex>();  Queue<Vertex> traversalOrder = new Queue<Vertex>();  Vertex startVertex = getVertex(navnPaaStartVertex);  startVertex.visited = true;  vertexQueue.Enqueue(startVertex);  traversalOrder.Enqueue(startVertex);  Vertex frontVertex;  while (vertexQueue.Count > 0)  {  Console.WriteLine("Items in Queue: " + vertexQueue.Count);  frontVertex = vertexQueue.Dequeue();  Console.WriteLine(" <- Dequeue: " + frontVertex.navn + " frontVertex");  foreach (Edge e in frontVertex.edgeList)  {  if (e.vertex.visited == false)  {  Console.WriteLine(" -> Enqueue: " + e.vertex.navn);  vertexQueue.Enqueue(e.vertex);  traversalOrder.Enqueue(e.vertex);  e.vertex.visited = true;  }  }  }  Console.Write(" Traversal order: ");  foreach (Vertex v in traversalOrder)  {  Console.Write(v.navn + " ");  }  Console.WriteLine();  }  public void DepthFirstTraversal(String navnPaaStartVertex)  {  Stack<Vertex> vertexStack = new Stack<Vertex>();  Queue<Vertex> traversalOrder = new Queue<Vertex>();  Vertex startVertex = getVertex(navnPaaStartVertex);  startVertex.visited = true;  Vertex topVertex;  vertexStack.Push(startVertex);  traversalOrder.Enqueue(startVertex);  while (vertexStack.Count > 0)  {  Console.WriteLine("Items in Stack: " + vertexStack.Count);  topVertex = vertexStack.Pop();  Console.WriteLine(" <- Pop from stack: " + topVertex.navn + " topVertex");  foreach (Edge e in topVertex.edgeList)  {  if (e.vertex.visited == false)  {  Console.WriteLine(" -> Push to stack: " + e.vertex.navn);  vertexStack.Push(e.vertex);  traversalOrder.Enqueue(e.vertex);  e.vertex.visited = true;  }  }  }  Console.Write(" Traversal order: ");  foreach (Vertex v in traversalOrder)  {  Console.Write(v.navn + " ");  }  Console.WriteLine();  }  }  class Vertex  {  public String navn;  public List<Edge> edgeList = new List<Edge>();  public Boolean visited = false;  public int lengthFromStartVertex = 0;  public Vertex previousVertex = null;  public Vertex(String n)  {  navn = n;  }  }  class Edge  {  public Vertex vertex;  public double weight;  }  } |