**DIJKSTRA ALGORITHM**

**Faculty name-Dr. Anuradha Dhull**

**Project Submitted By -Ayush Singh(19csu060) &**

**Bhavya panwar (19csu068)**

**Semester:4th**

**Group: DS-A2**

**Department of Computer Science and Engineering**

**The NorthCap University, Gurugram- 122001, India**

**Session 2020-21**

**INTRODUCTION**

Dijkstra algorithm is used for finding the shortest path between nodes in a graph. It is a greedy algorithm that solves the single-source shortest path problem for a directed/undirected graph with nonnegative weights.

**Condition:** if (d(u) + c (u + v) < d(v))

d(v) = d(u)+ c (u, v)

where u= source vertex, v= destination vertex

**FEATURES**

1. Toggle buttons are used and only one operation can be performed at a time.
2. The new graph button clears the canvas.
3. Click on canvas to draw .

**TIME COMPLEXITY ANALYSIS -**When we set the initial vertices to infinity and source vertex to zero in Dijkstra’s function itself this takes upto Θ(V) time here.

Then when we move to the part where we store the vertices in heap and build the min heap ,

it by defaults takes Θ(n) time so here in this case it will take Θ(V) time where V is no. of vertices.

Then when we take out the values of vertices out of heap and perform heapify method by default it takes Θ( )time and there are V elements in it so total it will take Θ(V\*) time

And then finally the weights of the edges need to be accessed every time which takes log V time and it will happen for every edge hence it will take Θ(E\* log V) time in total

Now adding the time complexities -> Θ(V)+ Θ(V)+ Θ(V\*)+ Θ(E\* log V)

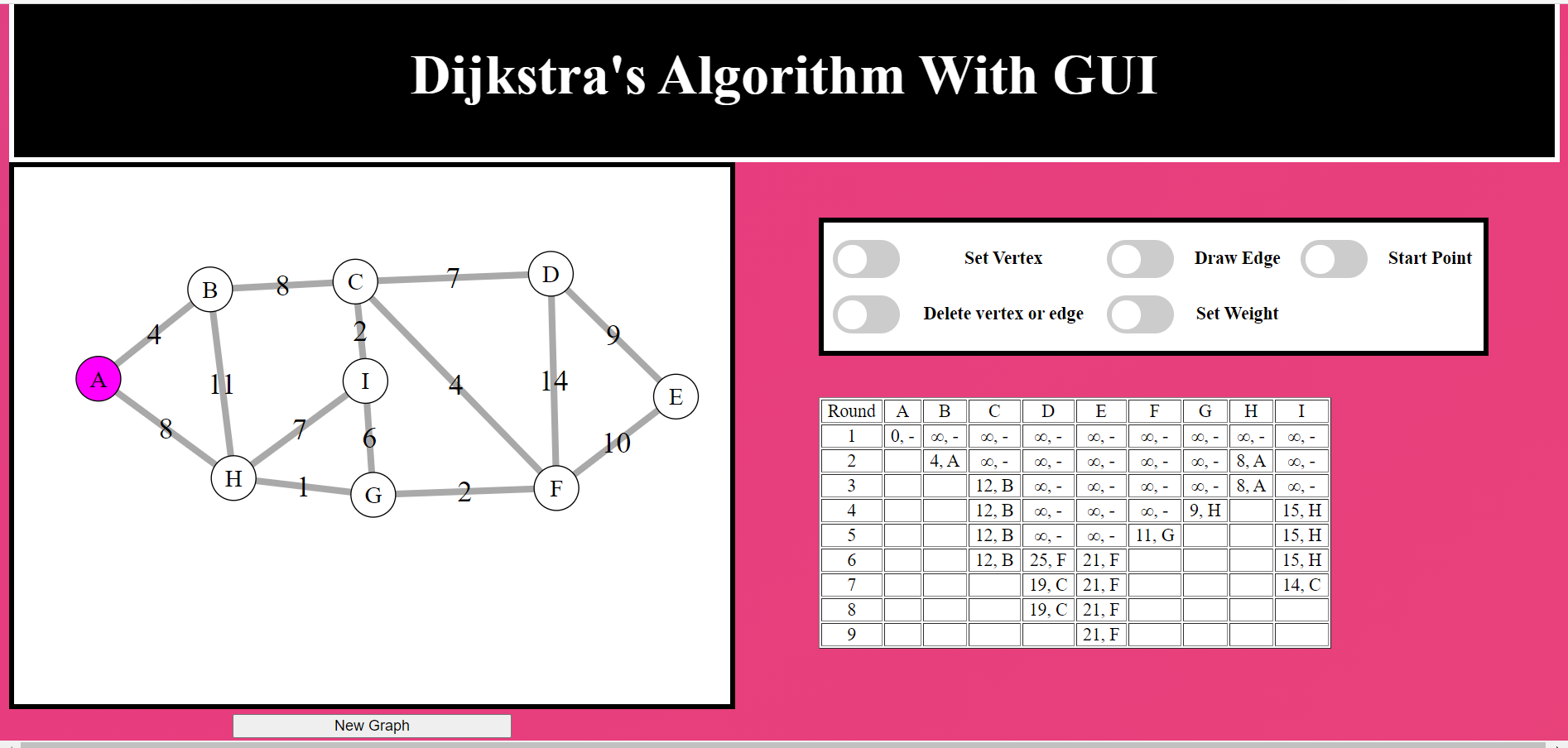
We know for most of the time there are more edges in a graph than vertices hence E>V

Therefore the time complexity comes out to be Θ(E\* log V).

**SPACE COMPLEXITY ANALYSIS—**When we implement Disjkstra's shortest path algorithm using a MinHeap , which in turn uses an array for storing the heap , and also if you use an array to store the values of the shortest distance for every node in the graph, your space complexity will be Θ(V)+ Θ(V)= Θ(2V)

And by removing constant we get Θ(V)

**OUTPUT**

****

**SOURCE CODE**

**Html**

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Shortest Path using Dijkstra's Algorithm 2</title>

<link rel="stylesheet" type="text/css" href="dijkstra.css">

</head>

<body>

    <div class="header">

        <h1>Dijkstra's Algorithm With GUI</h1>

      </div>

<div class="container">

    <div class="item">

        <svg width="640" height="480" version="1.1" id="shortestPathSvg" xmlns="http://www.w3.org/2000/svg">

            <g id="gedge"></g>

            <g id="gvertex"></g>

        </svg>

        <br>

        <button id="savesvgbutton" style="display:none">Save graph</button>

        <button type="reset" style="width: 250px; " onClick="window.location.reload();" value="newGraph"> New Graph</button>

        <br>

        <br>

        <div style=" background-color: white; border:5px solid   ;">

            <h2>How To Use</h2>

        <ul>

        <li>Switch On <b>Set vertex</b> to make a vertice, click anywhere in the canvas to create a vertex.</li>

        <li>To connect vertices switch on  <b>Draw edge</b>  then click on the vertices you want to connect.</li>

        <li>To give weight of edge switch on <b>Set Weight</b> </li>

        <li>To select the starting point switch on  <b> Start Point</b> then click on the starting vertex to get the results.</li>

        </ul>

        </div>

    </div>

    <div class="item2"   >

        <table class="table1" style="border:5px solid">

            <td>

                <label class="switch">

                <input type="checkbox" name="radWorkMode" value="drawVertex" checked="">

                <span class="slider round"></span>

            </label></td>

            <td style="word-wrap: normal;">Set Vertex</td>

            <td>

                <label class="switch">

                    <input type="checkbox" name="radWorkMode" value="drawEdge">

                    <span class="slider round"></span>

                </label></td>

            <td>Draw Edge</td>

            <td>

                <label class="switch">

                    <input type="checkbox" name="radWorkMode" value="setStart">

                    <span class="slider round"></span>

                </label>

                </td>

            <td>Start Point</td>

        </tr>

        <tr>

            <td>

                <label class="switch">

                    <input type="checkbox" name="radWorkMode" value="delVertexEdge">

                    <span class="slider round"></span>

                </label></td>

            <td>Delete vertex or edge</td>

                <td><label class="switch">

                    <input type="checkbox" name="radWorkMode" value="setCostLabel">

                    <span class="slider round"></span>

                </label></td></td>

                <td>Set Weight</td>

                <td>&nbsp;</td>

                <td>&nbsp;</td>

            </tr>

        </table>

        <br>

        <br>

        <table id="dijkstraSteps" ></table>

        <!-- The Modal -->

        <div id="divEdgeCost" class="modal">

          <!-- Modal content -->

          <div class="modal-content">

            <span class="close" onclick="closeDialogEdgeCost()">&times;</span>

            <label>Edge Cost</label>

                <input type="number" maxlength="4" required="required" autocomplete="off" min="1" max="1000" name="inputEdgeCost" id="inputEdgeCost">

                <button id="btnSetEdgeCost">Set value</button>

          </div>

        </div>

        <div id="divVertexLabel" class="modal">

          <!-- Modal content -->

          <div class="modal-content">

            <span class="close" onclick="closeDialogVertexLabel()">&times;</span>

            <label>Vertex Label</label>

                <input type="text" maxlength="1" required="required" autocomplete="off" name="inputVertexLabel" id="inputVertexLabel">

                <button id="btnSetVertexLabel">Set value</button>

          </div>

        </div>

        <script src="dijkstra.js"></script>

        <script>

            const switches = document.querySelectorAll("input[type=checkbox]");

            for(const s of switches) s.addEventListener("change", check);

            function check(e) {

              //count the number of checked switches:

              let n = 0;

              for(const s of switches) {

                if(s.checked) n++;

              }

              // if there is more than one checked (including the one you just clicked), uncheck it:

              if(n > 1) e.target.checked = false;

            }

            </script>

    </div>

</div>

<div class="setfotterdown">

    <div class="footer-dark">

        <footer>

            <div class="container">

                <br>

                <div style="margin-left: 10%; margin-top: 3.5%; " >

                    <div class="FANCYTEXT"> MADE BY  :   -  Ayush(19CSU060) & Bhavya(19CSU068)  </div>

                </div>

            </div>

        </footer>

    </div>

</div>

</body>

</html>

**Css**

svg { border: 5px solid black;

  background-color: white;

}

.modal {

    display: none; /\* Hidden by default \*/

    position: fixed; /\* Stay in place \*/

    z-index: 1; /\* Sit on top \*/

    left: 0;

    top: 0;

    width: 100%; /\* Full width \*/

    height: 100%; /\* Full height \*/

    overflow: auto; /\* Enable scroll if needed \*/

    background-color: rgb(0,0,0); /\* Fallback color \*/

    background-color: rgba(0,0,0,0.4); /\* Black w/ opacity \*/

}

.modal-content {

    background-color: #fefefe;

    margin: 15% auto; /\* 15% from the top and centered \*/

    padding: 20px;

    border: 1px solid #888;

    width: 30%; /\* Could be more or less, depending on screen size \*/

}

.close {

    color: #aaa;

    float: right;

    font-size: 28px;

    font-weight: bold;

}

.close:hover,

.close:focus {

    color: black;

    text-decoration: none;

    cursor: pointer;

}

.switch {

    position: relative;

    display: inline-block;

    width: 60px;

    height: 34px;

  }

  /\* Hide default HTML checkbox \*/

  .switch input {

    opacity: 0;

    width: 0;

    height: 0;

  }

  .slider {

    position: absolute;

    cursor: pointer;

    top: 0;

    left: 0;

    right: 0;

    bottom: 0;

    background-color: #ccc;

    -webkit-transition: .4s;

    transition: .4s;

  }

  .slider:before {

    position: absolute;

    content: "";

    height: 26px;

    width: 26px;

    left: 4px;

    bottom: 4px;

    background-color: white;

    -webkit-transition: .4s;

    transition: .4s;

  }

  input:checked + .slider {

    background-color: #2196F3;

  }

  input:focus + .slider {

    box-shadow: 0 0 1px #2196F3;

  }

  input:checked + .slider:before {

    -webkit-transform: translateX(26px);

    -ms-transform: translateX(26px);

    transform: translateX(26px);

  }

  /\* Rounded sliders \*/

  .slider.round {

    border-radius: 34px;

  }

  .slider.round:before {

    border-radius: 50%;

  }

  td

  {

      padding:0 5px;

  }

.table1{

    border-collapse: separate;

    border-spacing: 0 1em;

    width: 600px;

    font-weight: bold;

}

.container {

    display: flex;

    text-align: center;

  }

.item {

    flex-basis: 100px;

    height: 100px;

   margin-right: 5px;

  }

.item2 {

    flex-basis: 500px;

    height: 50px;

    margin-top: 50px;

    margin-left: 5%;

  }

  body {

    background: linear-gradient(-45deg, #ee7752, #e73c7e, #23a6d5, #23d5ab);

    background-size: 2000% 2000%;

    animation: gradient 15s ease infinite;

  }

  @keyframes gradient {

    0% {

      background-position: 0% 50%;

    }

    50% {

      background-position: 100% 50%;

    }

    100% {

      background-position: 0% 50%;

    }

  }

table

{

  background-color: white;

  border: 5p solid;

}

.header {

  padding: 10px;

  text-align: center;

  background: black;

  color: white;

  font-size: 25px;

  border: 5px white solid;

}

.col\_white\_amrc { color:#FFF;}

footer { width:100%; background-color:black; min-height:100px; padding:10px 0px 25px 0px ;

  border: 5px white solid;}

footer p { font-size:13px; color:#CCC; padding-bottom:0px; margin-bottom:8px;}

.setfotterdown{

margin-top: 50%;

}

.FANCYTEXT{

  position: absolute;

  left: 50%;

  transform: translate(-50%, -50%);

  font-size:30px;

  letter-spacing:0.1em;

  -webkit-text-fill-color: transparent;

  -webkit-text-stroke-width: 3px;

  -webkit-text-stroke-color: white;

  text-shadow:

            8px 8px #ff1f8f;

}

**JavaScript**

var svg = document.getElementById('shortestPathSvg'),

    gedge = document.getElementById('gedge'),

    gvertex = document.getElementById('gvertex'),

    saveButton = document.getElementById('savesvgbutton'),

    svgns = "http://www.w3.org/2000/svg",

    isDrawingEdge = false,

    edgeVertex1 = null,

    edgeVertex2 = null,

    vertices = document.getElementsByTagName("circle"),

    edges = document.getElementsByTagName("line"),

    texts = document.getElementsByTagName("text"),

    nextLabelCode= 65; //65='A' ,  97='a'

var markedVertices = [],

    table = document.getElementById("dijkstraSteps");

const   vertexRadius = 20,

        vertexLabelSize = "16pt",

        vertexColor = "white",

        vertexBorderColor = "black",

        startVertexColor = "magenta",

        markedVertexColor = "blue",

        markedEdgeColor = "cyan",

        vertexLabelColor = "black",

        edgeColor = "darkgrey",

        edgeWidth = 6,

        edgeLabelColor= "black",

        edgeLabelSize = "20pt",

        INF=100000;

svg.onmousedown = mouseClick;

saveButton.onmousedown = saveSvgFile;

//gedge.onmousedown = mouseClick;

//gvertex.onmousedown = mouseClick;

function mouseClick (e) {

    var radWorkMode=document.getElementsByName("radWorkMode"),

        workMode = null,

        clickTarget = null;

    for(var i=0; i< radWorkMode.length; i++)

        if(radWorkMode[i].checked){

            workMode = radWorkMode[i].value;

            break;

        }

    if(e.target.nodeName == "text")

        clickTarget=e.target.under;

    else

        clickTarget = e.target;

    switch (workMode) {

        case "drawVertex":

            if(clickTarget==svg){

                    drawVertex(e.clientX, e.clientY);

                }

                break;

        case "drawEdge":

            if(clickTarget.nodeName=="circle") {

                if(!isDrawingEdge){

                    edgeVertex1 = clickTarget;

                    isDrawingEdge=true;

                }

                else if(clickTarget != edgeVertex1){ //edge cannot be from a vertex to itself

                    edgeVertex2 = clickTarget;

                    drawEdge(edgeVertex1, edgeVertex2);

                    edgeVertex1 = null;

                    edgeVertex2 = null;

                    isDrawingEdge = false;

                }

            }

            break;

        case "delVertexEdge":

            if(clickTarget.nodeName == "circle")

                delVertex(clickTarget);

            if(clickTarget.nodeName=="line")

                 delEdge(clickTarget);

            break;

        case "setCostLabel":

            if(clickTarget.nodeName == "circle") {

                var vertex = clickTarget;

                showDialogVertexLabel(vertex);

            }

            if(clickTarget.nodeName == "line") {

                var edge = clickTarget;

                showDialogEdgeCost(edge);

            }

            break;

        case "setStart":

            if(clickTarget.nodeName == "circle") {

                setVertexNeighbors();

                var vertex = clickTarget;

                vertex.isSource = true;

                dijkstra(vertex);

            }

            break;

    }

}

function drawVertex(px, py){

    var vertex = document.createElementNS(svgns, "circle");

    vertex.setAttributeNS(null, "cx", px);

    vertex.setAttributeNS(null, "cy", py);

    vertex.setAttributeNS(null, "r",  vertexRadius);

    vertex.setAttributeNS(null, "fill", vertexColor);

    vertex.setAttributeNS(null, "stroke", vertexBorderColor);

    vertex.style["cursor"] = "pointer";

    gvertex.appendChild(vertex);

    vertex.label =  String.fromCharCode(nextLabelCode++);

    vertex.labelText = document.createElementNS(svgns, "text");

    vertex.labelText.setAttribute("x", vertex.cx.baseVal.value);

    vertex.labelText.setAttribute("y", vertex.cy.baseVal.value);

    vertex.labelText.setAttribute("text-anchor", "middle");

    vertex.labelText.setAttribute("alignment-baseline", "central");

    vertex.labelText.setAttribute("font-size", vertexLabelSize);

    vertex.labelText.setAttribute("fill", vertexLabelColor);

    vertex.labelText.textContent=vertex.label;

    vertex.labelText.under=vertex;

    vertex.labelText.boundTo="vertex";

    vertex.labelText.style["cursor"] = "pointer";

    gvertex.appendChild(vertex.labelText);

    vertex.edges=[];

    vertex.neighbors=[];

    vertex.isSource=false;

    vertex.cost=0;

    vertex.previous = -1;

}

function drawEdge(vertex1, vertex2) {

    var x1 = vertex1.getAttributeNS(null, "cx"),

        y1 = vertex1.getAttributeNS(null, "cy"),

        x2 = vertex2.getAttributeNS(null, "cx"),

        y2 = vertex2.getAttributeNS(null, "cy"),

        edgeExists = false;

    //check if a previous edge exists between the same vertices

    for(var i=0; i<edges.length; i++){

        if( edges[i].x1.baseVal.value == parseInt(x1) && edges[i].x2.baseVal.value == parseInt(x2) &&

            edges[i].y1.baseVal.value == parseInt(y1) && edges[i].y2.baseVal.value == parseInt(y2)) {

                edgeExists = true;

                break;

            }

    }

    if(edgeExists) {

        alert("Edge Exists!")

    }

    else {

        var edge = document.createElementNS(svgns, "line");

        edge.setAttributeNS(null, "x1", x1);

        edge.setAttributeNS(null, "x2", x2);

        edge.setAttributeNS(null, "y1", y1);

        edge.setAttributeNS(null, "y2", y2);

        edge.setAttributeNS(null, "stroke", edgeColor);

        edge.setAttributeNS(null, "stroke-width", edgeWidth);

        gedge.appendChild(edge);

        edge.fromVertex=vertex1;

        edge.toVertex=vertex2;

        edge.cost = 1;

        edge.costText = document.createElementNS(svgns, "text");

        edge.costText.setAttribute("x", 0.5\*(edge.x1.baseVal.value+edge.x2.baseVal.value));

        edge.costText.setAttribute("y", 0.5\*(edge.y1.baseVal.value+edge.y2.baseVal.value));

        edge.costText.setAttribute("text-anchor", "middle");

        edge.costText.setAttribute("alignment-baseline", "central");

        edge.costText.setAttribute("font-size", edgeLabelSize);

        edge.costText.setAttribute("fill", edgeLabelColor);

        edge.costText.textContent=edge.cost;

        edge.costText.under=edge;

        edge.costText.boundTo="edge";

        edge.style["cursor"] = "pointer";

        edge.costText.style["cursor"] = "pointer";

        gedge.appendChild(edge.costText);

        vertex1.edges.push(edge);

        vertex2.edges.push(edge);

    }

}

function setEdgeCost(edge, cost) {

    edge.cost = cost;

    edge.costText.textContent=edge.cost;

}

function setVertexLabel (vertex, label) {

    vertex.label =  label;

    vertex.labelText.textContent=vertex.label;

}

function delVertex(vertex){

    while(vertex.edges.length > 0)

        delEdge(vertex.edges[0]);

    gvertex.removeChild(vertex.labelText);

    gvertex.removeChild(vertex);

}

function delEdge(edge) {

    //delete edge.fromVertex.edges;

    for(var i=0; i< edge.fromVertex.edges.length; i++)

        if(edge.fromVertex.edges[i] == edge) {

            edge.fromVertex.edges.splice(i,1);

            break;

        }

    //delete edge.toVertex.edges

    for(var i=0; i< edge.toVertex.edges.length; i++)

        if(edge.toVertex.edges[i] == edge) {

            edge.toVertex.edges.splice(i,1);

            break;

        }

    gedge.removeChild(edge.costText);

    gedge.removeChild(edge);

}

function setVertexNeighbors(){

    for(var i =0; i< vertices.length; i++)

        for(var j=0; j< vertices[i].edges.length; j++) {

            if(vertices[i].edges[j].fromVertex == vertices[i])

                vertices[i].neighbors.push(vertices[i].edges[j].toVertex);

            else

                vertices[i].neighbors.push(vertices[i].edges[j].fromVertex);

        }

}

function redraw() {

    for(var i=0; k<edges.length; i++)

        svg.appendChild(edges[i]);

    for(var j=0; k<vertices.length; j++)

        svg.appendChild(vertices[j]);

    for(var k=0; k<texts.length; k++)

        svg.appendChild(texts[k]);

}

function clearGraph() {

    while (svg.lastChild)

        svg.removeChild(svg.lastChild);

}

function dijkstra(source) {

    //initialization

    var tr=0, round=1;

    table.innerHTML="";

    table.setAttribute("border",1);

    var row = table.insertRow(0);

    var cell = row.insertCell(0);

    cell.innerHTML = "Round";

    setVertexNeighbors();

    for(i=0; i< vertices.length; i++){

        vertices[i].cost = INF;

        vertices[i].previous = null;

        vertices[i].marked = false;

        vertices[i].markedRound = INF;

        //interface begin

        cell = row.insertCell(i+1);

        cell.innerHTML=vertices[i].label;

        if(vertices[i]==source)

            vertices[i].setAttribute("fill", startVertexColor);

        else

            vertices[i].setAttribute("fill", vertexColor);

        //interface end

    }

    markedVertices =[];

    source.cost = 0;

    //interface begin

    tr++;

    row=table.insertRow(tr);

    cell=row.insertCell(0);

    cell.innerHTML = round;

    for(i=0; i< vertices.length; i++){

        cell=row.insertCell(i+1);

        cell.innerHTML=(vertices[i].cost==INF)?'∞':vertices[i].cost;

        cell.innerHTML+=', '+ ((vertices[i].previous==null)?'-':vertices[i].previous.label);

    }

    //interface end

    do {

        //find vertex with minimum cost

        var min = INF;

        for(i=0; i< vertices.length; i++)

            if(vertices[i].cost < min && !vertices[i].marked) {

                m = i; min = vertices[m].cost;

            }

        vertices[m].marked = true; vertices[m].markedRound=round;

        markedVertices.push(vertices[m]);

        //relax edges

        for(j=0; j< vertices[m].edges.length; j++){

            neighbor= (vertices[m].edges[j].fromVertex == vertices[m])? vertices[m].edges[j].toVertex : neighbor=vertices[m].edges[j].fromVertex;

            edge=vertices[m].edges[j];

            if(neighbor.cost > vertices[m].cost + edge.cost){

                neighbor.cost = vertices[m].cost + edge.cost;

                neighbor.previous = vertices[m];

            }

        }

        //interface begin

        tr++;round++;

        if(markedVertices.length < vertices.length) {

            row=table.insertRow(tr);

            cell=row.insertCell(0);

            cell.innerHTML = round;

            for(i=0; i< vertices.length; i++){

                cell=row.insertCell(i+1);

                if(vertices[i].markedRound > round){

                    cell.innerHTML=(vertices[i].cost==INF)?'∞':vertices[i].cost;

                    cell.innerHTML+=', '+ ((vertices[i].previous==null)?'-':vertices[i].previous.label);

                }

            }

        }

        //interface end

    } while(markedVertices.length < vertices.length);

    //color tree

}

//File save from

function saveSvgFile() {

    var s = new XMLSerializer();

    content=s.serializeToString(svg);

    downloadFile("dijkstra.svg",content);

}

function downloadFile(filename, content) {

    var pom = document.createElement('a');

    pom.setAttribute('href', 'data:text/plain;charset=utf-8,' + encodeURIComponent(content));

    pom.setAttribute('download', filename);

    if (document.createEvent) {

        var event = document.createEvent('MouseEvents');

        event.initEvent('click', true, true);

        pom.dispatchEvent(event);

    }

    else {

        pom.click();

    }

}

//Modal

var dialogEdgeCost = document.getElementById('divEdgeCost');

var dialogVertexLabel = document.getElementById('divVertexLabel');

function showDialogEdgeCost (edge) {

    var inputEdgeCost = document.getElementById("inputEdgeCost");

    var btnSetEdgeCost = document.getElementById("btnSetEdgeCost");

    inputEdgeCost.value = edge.cost;

    dialogEdgeCost.style.display = "block";

    btnSetEdgeCost.onclick = function () {

        dialogEdgeCost.style.display = "none";

        var cost=parseInt(inputEdgeCost.value);

        if(cost)

            setEdgeCost(edge, cost);

    }

}

function closeDialogEdgeCost() {

    dialogEdgeCost.style.display = "none";

}

function showDialogVertexLabel (vertex) {

    var inputVertexLabel = document.getElementById("inputVertexLabel");

    var btnSetVertexLabel = document.getElementById("btnSetVertexLabel");

    inputVertexLabel.value = vertex.label;

    dialogVertexLabel.style.display = "block";

    btnSetVertexLabel.onclick = function () {

        dialogVertexLabel.style.display = "none";

        var label = inputVertexLabel.value;

        if(label)

            setVertexLabel(vertex, label);

    }

}

function closeDialogVertexLabel() {

    dialogVertexLabel.style.display = "none";

}

function newGraph()

{

    const reloadtButton = document.querySelector("#reload");

    // Reload everything:

    function reload() {

        reload = location.reload();

    }

    // Event listeners for reload

    reloadButton.addEventListener("click", reload, false);

}