Polygon Space
Manual Técnico

indice

Tecnologías:

Al ser esta una pagina web exclusivamente front-end las tecnologías utilizadas fueron:

- Html
- Javascript
- Css

Bibliotecas:

vis.js

Esta completa biblioteca te permite crear grafos dinámicos entre otras muchas cosas. Disponible en: http://visjs.org/

Módulos

La aplicación esta desarrollada bajo el modelo vista controlador por lo que se subdivide en tres módulos:

- ui.js : Encargado de la interfaz gráfica del usuario.
- search.js: Contiene los algoritmos de búsqueda y el modelado del grafo.
- sim.js: Es el que lleva a cabo la simulación del recorrido.

ui.js (obtiene el gráfico) → search.js(Procesa y devuelve el recorrido) → sim.js(ejecuta el ciclo del recorrido) → ui.js (re-dibuja el gráfico).

Arbol de archivos:

/tree-search

- index.html

/js

- a.js
- search.js
- simulation.js
- draw.js
- convexhull.js

/css

- style.css

Codigo:

index.html

```
<!DOCTYPE HTML>
<!--
Author: David Ruiz Garcia
github-repository: https://github.com/
git-page: https://david195.github.io/
-->

<html>
<head>
  <title>*</title>
  <meta name="author" content="David Ruiz">
  <meta charset="utf-8">
  <link rel="stylesheet" type="text/css" href="css/style.css">
  <script src="js/draw.js"></script>
  <script src="js/search.js"></script>
  <script src="js/a.js"></script>
  <script src="js/simulation.js"></script>
  <script src="js/vis.min.js"></script>

  <script>
    var canvas;
    function init(){
      canvas = new draw_space(document.getElementById('canvas'),600,1000);
      document.getElementById('defaultOpen').click();
    }
    function search(){
      var algoritmo = document.getElementById('opt').value;
       canvas.search(canvas.inode,canvas.gnode,algoritmo,document.getElementById('data'),document.ge
tElementById('search'));
      openCity(event, 'solution');
    }
    function crear_grafo(){
      canvas.vgraph(document.getElementById('graph'));
    }
    function sim(){
      var div = document.getElementById('simulation');
      route = [{x:0,y:0},{x:20,y:20},{x:200,y:100}];
      var s = new boot(5,route,1000,div);
      s.start();
    }
  </script>

</head>

<body onload="init()">
  <div class="tab">
        <button  class="tablinks"  id="defaultOpen"  onclick="openCity(event,  'draw')">Areá  de
edicion</button>
    <button class="tablinks" onclick="openCity(event, 'solution')">Árbol solución</button>
    <button class="tablinks" onclick="openCity(event, 'simulation')">Simulación</button>
  </div>
  <div id="layout">
    <div id='draw' class="tabcontent">
      <div id="canvas_buttons">
        <button onclick="canvas.draw()">Nuevo Poligono</button>
        <button onclick="canvas.set_inode()">Nodo inicial</button>
        <button onclick="canvas.set_gnode()">Nodo Meta</button>
        <button onclick="canvas.draw_visibles()">Vertices visibles</button>
        <button onclick="crear_grafo()">Crear Grafo</button>
        Algoritmo:&nbsp&nbsp&nbsp<select id="opt">
          <option value="a*">Busqueda A*</option>
          <option value="avida">Busqueda Avida</option>
        </select>
        <button onclick="search()">Busqueda</button>
      </div><br><br>
      <div id="canvas"></div>
      <div id="graph"></div>
      <div id="graphS"></div>
    </div>
    <div id='solution' class="tabcontent">
      <div id="search"></div>
      <div id="data"></div>
    </div>
    <div id="simulation" class="tabcontent" >
      <button onclick="sim()">start</button>
    </div>
  </div>
</body>
</html>
```

```
/*convexhull.js*/
(function () {
    'use strict';

    function convexHull(points) {
        points.sort(function (a, b) {
            return a.x != b.x ? a.x - b.x : a.y - b.y;
        });

        var n = points.length;
        var hull = [];

        for (var i = 0; i < 2 * n; i++) {
            var j = i < n ? i : 2 * n - 1 - i;
             while (hull.length >= 2 && removeMiddle(hull[hull.length - 2], hull[hull.length - 1],
points[j]))
                hull.pop();
            hull.push(points[j]);
        }

        hull.pop();
        return hull;
    }

    function removeMiddle(a, b, c) {
        var cross = (a.x - b.x) * (c.y - b.y) - (a.y - b.y) * (c.x - b.x);
        var dot = (a.x - b.x) * (c.x - b.x) + (a.y - b.y) * (c.y - b.y);
        return cross < 0 || cross == 0 && dot <= 0;
    }

    // export as AMD module / Node module / browser or worker variable
    if (typeof define === 'function' && define.amd) define(function () { return convexHull; });
    else if (typeof module !== 'undefined') module.exports = convexHull;
    else if (typeof self !== 'undefined') self.convexHull = convexHull;
    else window.convexHull = convexHull;
})();
```

```
/*a.js*/
var type="a*";
var h;

function avida(ei,ef,data,callback){
  type = "avida";
  a(ei,ef,data,callback);
}

function a(ei,ef,data,callback){
  h = heuristic(ef,data);
  var html_list = '<hr>Lista de nodos<br><br>';
  var cont = 0;
  var nodes = [];
  var edges = [];
  var list = [];
  var np=0;
  var n = {id:ei.toString()+"-"+h[ei]+"-"+ei,node:ei,val:h[ei],label:ei,level:0};
  while(n.node!=ef){
    var neighbors = get_neighbors(n,data.edges._data);
    if(neighbors.length !=0){
      for (var i=0;i<neighbors.length;i++){
        var nb = neighbors[i].node;
        if(!is_in(nb,nodes)){
          var v = h[n.node];
          if(type=='a*')
            v += neighbors[i].val-h[n.node];
          var exist = false;
          for (var ind in list) {
            if(nb.toString()+"-"+v+"-"+n.node == list[ind].id){
                exist = true;
                break;
            }
          }
          if(!exist){
            list.push({id:nb.toString()+"-"+v+"-"+n.node,node:nb,val:v,label:nb,level:n.level+1});
            var e = {from:n.id.toString()};
            e.to = nb.toString()+"-"+v+"-"+n.node;
            if(type=='a*')
              e.label = neighbors[i].val-n.val;
            else {
              e.label = neighbors[i].val;
            }
            edges.push(e);
          }
        }
      }
    }
    nodes.push(n);
    list.sort(compare);
    cont++;
    html_list += "interaction: "+cont+"<br>";
    for(var i=list.length-1;i>=0;i--)
      html_list+=list[i].id.replace("-","<sub>")+"</sub>"+" -> ";
    html_list+='<br>';
    np++;
    n = list.shift();
    if(n==null)
      break;
  }
  nodes.push(n);
  var route=[];
  var cost = 0;
  var nn = nodes[nodes.length-1].id;
  while(nn!=nodes[0].id){
    for(var i=0;i<edges.length;i++){
      if(edges[i].to == nn){
        route.push(nn);
        cost+= parseInt(edges[i].label);
        nn=edges[i].from;
      }
    }
  }
  route.push(nn);
                 var         sol         =         {route:route,cost:cost,list:html_list,np:np,tree:
{nodes:nodes,edges:edges},algorithm:type};
  callback(sol);
  type = "a*";
}

/***functions***/


function heuristic(ef,data){//Returns the heuristic table
  var ht = [];
  var nodes = data.nodes._data;
  for (var i in nodes){
                  var     hn     =     distance({x:data.nodes._data[ef].x,y:data.nodes._data[ef].y},
{x:nodes[i].x,y:nodes[i].y});
    ht.push(hn);
  }
  return ht;
}

function get_neighbors(n,edges){
  var nb = [];
  for (var i in edges){
    var g = 0;
    if(type == 'a*')
      g=n.val;
    var nn = {val:g+parseInt(edges[i].label)};
    if(edges[i].from == n.node){
      nn.label = edges[i].to;
      nn.node = edges[i].to;
      nn.id = edges[i].to.toString()+"-"+nn.val.toString()+"-"+n.node;
      nb.push(nn);
    }
    if(edges[i].to == n.node){
      nn.label = edges[i].from;
      nn.node = edges[i].from;
      nn.id = edges[i].from.toString()+"-"+nn.val.toString()+"-"+n.node;
      nb.push(nn);
    }
  }
  return nb;
}

function compare(a,b) {
  if (a.val < b.val)
    return -1;
  if (a.val > b.val)
    return 1;
  return 0;
}

function distance(p1,p2){
  return Math.round(Math.sqrt(Math.pow(p2.x-p1.x,2)+Math.pow(p2.y-p1.y,2)));
}

function is_in(x,l){
  for(var i=0;i<l.length;i++){
    if(l[i].node==x)
      return true;
  }
  return false;
}
```

```
/*draw.js*/

function openCity(evt, cityName) {
    // Declare all variables
    var i, tabcontent, tablinks;

    // Get all elements with class="tabcontent" and hide them
    tabcontent = document.getElementsByClassName("tabcontent");
    for (i = 0; i < tabcontent.length; i++) {
        tabcontent[i].style.display = "none";
    }

    // Get all elements with class="tablinks" and remove the class "active"
    tablinks = document.getElementsByClassName("tablinks");
    for (i = 0; i < tablinks.length; i++) {
        tablinks[i].className = tablinks[i].className.replace(" active", "");
    }

    // Show the current tab, and add an "active" class to the button that opened the tab
    document.getElementById(cityName).style.display = "block";
    evt.currentTarget.className += " active";
}

function importScript(nombre) {
    var s = document.createElement("script");
    s.src = nombre;
    document.querySelector("head").appendChild(s);
}

importScript('js/convexhull.js');

function draw_space(div,height,width){

  this.inode = -1;
  this.gnode = -1;
  this.solution = {};
  this.edges = []
  this.nodes = []
  this.drawing = div;
  this.graph = null;
  this.tree = null;
  var scale = 5;
  this.canvas = document.createElement('canvas');
  this.canvas.width = width;
  this.canvas.height = height;
  this.ctx = this.canvas.getContext("2d");
  this.drawing.appendChild(document.createElement('div').appendChild(this.canvas));
  this.polygons = [];
  var polygon = [];

  this.canvas.addEventListener('mousedown', function(evt) {
    var rect = this.getBoundingClientRect();
    var x = evt.clientX - rect.left;
    var y =  evt.clientY - rect.top;
    var ctx = this.getContext("2d");
    polygon.push({x:x,y:y});
    ctx.beginPath();
    ctx.arc(x,y,scale/2,0,(Math.PI/180)*360,true);
    ctx.fillStyle = "rgba(0, 0, 200, 0.5)";
    ctx.fill();
  },true);

  this.canvas.addEventListener('mousemove', function(evt) {
    var mousePos = getMousePos(this, evt);
    var message = 'x: '+ mousePos.x + ' y:' + mousePos.y;
    writeMessage(this, message);
  }, false);

  this.draw = function(){
    var hullPoints = convexHull(polygon);
    var from = hullPoints[0];
    from.id=this.nodes.length;
    var n_init = this.nodes.length;
    var poly = {};
    poly.nodes = [];
    poly.edges = [];
    this.ctx.beginPath();
    this.ctx.moveTo(from.x,from.y);
    var nnode = this.nodes.length+1;
    for(var i=1; i<hullPoints.length;i++){
      poly.nodes.push(from);
      this.nodes.push(from);
      this.ctx.lineTo(hullPoints[i].x,hullPoints[i].y);
      var edge ={};
      edge.value = distance(from,hullPoints[i]);
      edge.from = from;
      edge.to = hullPoints[i];
      poly.edges.push(edge);
      this.edges.push(edge);
      from = hullPoints[i];
      from.id = nnode;
      nnode++;
    }
    var edge ={};
    edge.value = distance(from,hullPoints[0]);
    edge.from = from;
    edge.to = {id:n_init,x:hullPoints[0].x,y:hullPoints[0].y};
    poly.edges.push(edge);
    poly.nodes.push(from);
    this.nodes.push(from);
    this.edges.push(edge);
    this.ctx.closePath();
    var color = color_rand();
    this.ctx.fillStyle = color;
    poly.color = color;
    this.ctx.fill();
    this.polygons.push(poly);
    polygon = [];
  }

  this.set_inode = function(){
    if(polygon.length!=1 || this.inode!=-1)
      return;
    polygon[0].id = this.nodes.length;
    this.inode = polygon[0].id;
    this.nodes.push(polygon[0]);
    draw_node(10,polygon[0],'red',this.ctx);
    polygon=[];
  }

  this.set_gnode = function(){
    if(polygon.length!=1 || this.gnode!=-1)
      return;
    var n = 0;
    polygon[0].id = this.nodes.length;
    this.gnode = polygon[0].id;
    this.nodes.push(polygon[0]);
    draw_node(10,polygon[0],'blue',this.ctx);
    polygon=[];
  }

  this.draw_visibles = function(){
    for(var i=0;i<this.nodes.length;i++){
      this.visibles(this.nodes[i].id,this.nodes[i].x,this.nodes[i].y);
    }
  }

  this.vgraph = function(div){
    /***/
    if(this.inode == -1 || this.gnode==-1)
      return;
    var vnodes = [];
    var vedges = [];
    for(var i=0;i<this.nodes.length;i++){
      var  n = {};
      n.id = i;
      n.label = i;
      n.x = this.nodes[i].x;
      n.y = this.nodes[i].y;
      n.color = this.nodes[i].color;
      vnodes.push(n);
    }
    for(var i=0;i<this.edges.length;i++){
      var e = {};
      e.from = this.edges[i].from.id;
      e.to = this.edges[i].to.id;
      e.label = this.edges[i].value;
      vedges.push(e);
    }
    div.style.zIndex = 3;
    this.graph = network(vnodes,vedges,div);
  }

  this.visibles = function(id,x,y){
    for(var i=0; i<this.nodes.length;i++){
      n = this.nodes[i];
      if(x==n.x && y == n.y || same_polygon(id,n.id,this.polygons))
        break;
      var edge = {from:{id:id,x:x,y:y},to:{id:n.id,x:n.x,y:n.y}};
      ni = 0;
      for(var j=0; j<this.polygons.length;j++)
        ni+= instersect(edge,this.polygons[j].edges);
      if(ni==0){
        edge.value = distance(edge.from,edge.to);
        this.ctx.moveTo(x,y);
        this.ctx.lineTo(n.x,n.y);
        this.ctx.lineWidth = 0;
        this.ctx.strokeStyle = "#000";
        this.edges.push(edge);
      }
    }
    this.ctx.stroke();
  }

  this.getGraf = function(){
    return this.graph;
  }
```

```
  this.search = function(ei,ef,type,data_div,div){
    if(ei==ef)
      return;
    for (i in this.nodes)
      this.nodes[i].color = null;
    var aux_nodes = this.nodes;
    if(type == 'a*'){
    var aux = null;
    if(type == 'a*')
      a(ei,ef,{nodes:this.graph.body.data.nodes,edges:this.graph.body.data.edges},function(sol){
        aux = sol;
        draw_sol(sol,data_div,div);
        for(i in sol.route){
          var ind = parseInt(sol.route[i].split("-")[0]);
          aux_nodes[ind].color = 'red'
        }
      });
    }
    else
       avida(ei,ef,{nodes:this.graph.body.data.nodes,edges:this.graph.body.data.edges},function(sol)
{
        aux = sol;
        draw_sol(sol,data_div,div);
        for(i in sol.route){
          var ind = parseInt(sol.route[i].split("-")[0]);
          aux_nodes[ind].color = 'red'
        }
      });
    this.canvas.style.opacity = 0.5;
    this.vgraph(this.graph.body.container);
    this.graph.redraw();
  }

  this.draw_polygons = function(){
    for(var i=0;i<this.polygons.length;i++){
      var n = this.polygons[i].nodes[0];
      var n0=n;
      var j;
      this.ctx.beginPath();
      this.ctx.moveTo(n.x,n.y);
      for(j=1;j<this.polygons[i].nodes.length;j++){
        this.ctx.lineTo(this.polygons[i].nodes[j].x,this.polygons[i].nodes[j].y);
      }
      this.ctx.closePath();
      var color = color_rand();
      this.ctx.fillStyle = color;
      this.ctx.fill();
    }
  }

}

function draw_sol(sol,data_div,div){
  var html = "<p>Busqueda"+sol.algorithm+"<br><br>Ruta solución: <br>";
  for(var i=0;i<sol.route.length;i++){
    for(var j=0;j<sol.tree.nodes.length;j++){
      if(sol.tree.nodes[j].id==sol.route[i]){
        html+=sol.route[i].replace("-","<sub>")+"</sub>"+" -> ";
        sol.tree.nodes[j].color = 'red';
        break;
      }
    }
  }
  html+="<br>Costo de ruta: "+sol.cost+"<br>Pasos realizados: "+sol.np+"<br>"+sol.list+"</p>";
  data_div.innerHTML=html;
  var options = {
      layout:{
        hierarchical: {
          enabled : true
        }
      },
  };
  var n = network(sol.tree.nodes,sol.tree.edges,div,options);
}

function draw_node(r,node,color,ctx){
  ctx.beginPath();
  ctx.arc(node.x,node.y,r,0,(Math.PI/180)*360,true);
  ctx.strokeStyle = color;
  ctx.lineWidth = 0;
  ctx.fillStyle = color;
  ctx.closePath();
  ctx.fill();
}

/****/

function network(nodes,edges,container,options){
  var data = {
    nodes: new vis.DataSet(nodes),
    edges: new vis.DataSet(edges)
  };
  if(options==null){
    options = {
      autoResize:false,
      interaction:{
        dragNodes:false,
        dragView:false,
        hoverConnectedEdges:false,
        zoomView:false,
      },
      physics:{
        enabled:false
      },
      nodes:{
        shape:'circle',
        size:5
      }
    };
  }
  var s = new vis.Network(container, data, options);
  var x = s.body.container.clientWidth/2;
  var y = s.body.container.clientHeight/2;
  s.moveTo({position:{x:x,y:y}});
  return s;
}

/******/
function distance(p1,p2){
  return Math.round(Math.sqrt(Math.pow(p2.x-p1.x,2)+Math.pow(p2.y-p1.y,2)));
}

function color_rand(){
   var hexadecimal = new Array("0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F")
   var color = "#";
   for (i=0;i<6;i++){
      posarray = aleatorio(0,hexadecimal.length)
      color += hexadecimal[posarray]
   }
   return color;
}

function aleatorio(inferior,superior){
   numPosibilidades = superior - inferior
   aleat = Math.random() * numPosibilidades
   aleat = Math.floor(aleat)
   return parseInt(inferior) + aleat;
}

function instersect(l1,edges){
  var n_int=0;
  for(var i=0; i<edges.length;i++){
    var l2 = edges[i];
    var p1 = l1.from;
    var p2 = l1.to;
    var p3 = l2.from;
    var p4 = l2.to;
    var lado1 = lado(p1,p2,p3,p4);
    var lado2 = lado(p3,p4,p1,p2);
    if(lado1<0 && lado2<0)
      n_int++;
  }
  return n_int;
}

function lado(p1,p2,p3,p4){
  var dx = p2.x -p1.x;
  var dy = p2.y -p1.y;
  var dx1 = p3.x-p1.x;
  var dy1 = p3.y-p1.y;
  var dx2 = p4.x-p2.x;
  var dy2 = p4.y-p2.y;
  var res = (dx*dy1 - dy*dx1) * (dx*dy2 - dy*dx2);
  return res;
}

function same_polygon(n1,n2,polygons){
  var a = -1;
  var b = -1;
  for(var i=0;i<polygons.length;i++){
    for(var j=0;j<polygons[i].nodes.length;j++){
      if(polygons[i].nodes[j].id == n1)
        a= i;
      if(polygons[i].nodes[j].id == n2)
        b= i;
    }
  }
  if(a!=-1 && a==b)
    return true;
  return false;
}

function getMousePos(canvas, evt) {
  var rect = canvas.getBoundingClientRect();
  return {
    x: evt.clientX - rect.left,
    y: evt.clientY - rect.top
  };
}

function writeMessage(canvas, message) {
  var context = canvas.getContext('2d');
  context.clearRect(0, 0, 85,20);
  context.font = '10pt Calibri';
  context.fillStyle = 'black';
  context.fillText(message,5,10);
}
```

Repositorio git-hub

https://github.com/david195/tree-search