MANUAL TÉCNICO - Programa de manipulación de grafos
Miguel Diaz Diaz - Santiago Monroy
Para realizar este programa se necesitó de Html, Css, Javascript y también se utilizó el framework de python llamado Django.

Con Html se encuentran los botones para activar y desactivar las funcionalidades del programa, se encuentra dispuesto un elemento html canvas para visualizar los grafos, este canvas es un elemento fundamental en el programa debido a que con ayuda del lenguaje Javascript nos aportará la interactividad gracias a los escuchadores de eventos que este lenguaje dispone, como por ejemplo cuando se de click izquierdo dentro de la zona del canvas creará un nodo con cierto id, name y sus respectivas posiciones que estarán guardadas en un objeto llamado Circle, para crear este objeto se creó una clase de Javascript definiendo sus respectivos métodos para dibujarse como un círculo (cabe aclarar que para dibujar en un canvas se debe obtener el contexto del canvas el cual nos servirá como un lienzo para dibujar en el canvas, para dibujar los círculos se dispone de una función del contexto llamada arc, en el cual teniendo los argumentos indicados se puede lograr esa circunferencia).

```
ctx.arc(this.x ,this.y , 20, 0, 2 * Math.PI, false);
```

this.x this.y siendo las posiciones en donde el usuario dio click para crear el nodo, el argumento 20 es el radio con esto se controlará claramente que tan grande es el círculo, con 2*Math.PI dará la forma circular y con el booleano false se controla si el elemento es “draggable” básicamente que si será un elemento arrastrable, pero este no debe importar porqué en el programa se usa un propio sistema “drag and drop” lo que hace que el usuario pueda arrastrar y mover los nodos del grafo a su gusto, también se debe mencionar que todos los objetos Circle creados serán guardados en un array para manipularlos posteriormente.

Así como hay una clase llamada Circle que nos ayuda a controlar los nodos del grafo, también existe una clase llamada Line esta clase se utiliza para las aristas, en este objeto se guardarán el peso de la arista, y los objetos Circle que se involucran, esto para conocer de qué nodo a qué nodo se dirige la arista, evidentemente esta clase tiene un método dibujar con el que se dibujó la flecha que indica la dirección de la arista, su relación y el respectivo peso, hay que mencionar que estos objetos que controlan las aristas también se guardaran en un array para su posterior manipulación.

Una función importante para la interactividad será la función actualizar la cual se debe invocar cuando se deba refrescar el canvas con nuevos nodos, nuevas aristas o la eliminación de estos mismos, su edición (por ejemplo cuando se edita el peso de las aristas o se edita el nombre de los nodos), está función “limpia” el canvas y lo redibuja con los elementos nuevos que se han agregado,eliminado o editado, esto dibuja de nuevo los elementos que estén guardados y ya modificados en los arrays dispuestos para los nodos y aristas, luego se llaman a sus métodos “draw()” dando esa sensación de actualización constante.

```
const actualizar=()=>{
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    circles.forEach((circle)=>{

        circle.draw();
    })
    lines.forEach((line)=>{

        line.draw();
    })
}
```

Para seleccionar los elementos se utilizan funciones de intersección que nos ayudan saber si el usuario dio click en los nodos o si selecciono los pesos de las aristas para hacer algún tipo de modificación, esto se verifica comparando el nuevo click que el usuario acaba de realizar con todos los nodos que existen actualmente, se realiza este sistema también para las aristas pero un poco modificado debido a que el peso, el cual será el target al que el usuario debe clickar se encuentra entre la mitad de los nodos por lo que se hace una pequeña variante, pero en si el mecanismo es igual. La comparación se hará mediante una fórmula matemática en el que las posiciones de los círculos y la nueva posición del click del usuario, nos indican si algún nodo o arista fue seleccionado.

```
const isIntersect=(point, circle)=>{

    return Math.sqrt((point.x-circle.x) ** 2 + (point.y - circle.y) ** 2) < 20;
}

const isIntersectLine=(point, line)=>{

    return Math.sqrt((point.x-(line.fromx+line.tox)/2) ** 2 + (point.y - (line.fromy+line.toy)/2) ** 2) < 20;
}
```

Para la eliminación o edición de los elementos del grafo se dispondrá de un menú que se despliega cuando el usuario da click derecho en un elemento, para los nodos se puede editar el nombre de los nodos, más no el Id de estos, también se podrá eliminar los nodos, evidentemente si este nodo que se desea seleccionar tiene aristas conectadas estás se borraran; para las aristas se puede editar el peso de las mismas o también existe la posibilidad de eliminarlas. Para esto se usan dos funciones mencionadas anteriormente las cuales son isIntersect tanto del nodo o de la arista y la función de actualizar, esto es más sencillo con actualizar debido a que como se mencionó antes se redibuja el canvas con respecto a los elementos que existan en el array de aristas o de nodos, cuando se editan,eliminan o crean elementos se modifican los objetos dentro de los arrays y como ya se dispone de todas la información guardada en estos arreglos se actualiza y se visualizan los cambios.

```
buttonEliminar.addEventListener('click',(e)=>{

    circles.forEach((circle)=>{
        if(isIntersect(posible ,circle)){


            deleteCircleInLines(circle);
            circles.splice(circles.indexOf(circle),1);
        }
    })

    actualizar();

})
```

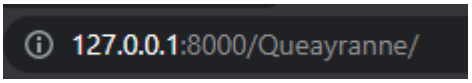
Este es un ejemplo para realizar la eliminación de nodos se hace uso de la función intersección para saber si se selecciono cualquier nodo, si es así primero que todo se eliminan las aristas (en el array) que tengan conexión con este nodo, para luego eliminar del array para nodos, el nodo que se ha seleccionado con el método splice, ya realizado esto se llama la función actualizar para mostrar el elemento canvas con los nuevos cambios.
Hablando sobre las conexiones entre el framework Django, el Html y Javascript. desde el html se pueden llamar funciones de django desde objetos html como el anclaje (<a>) en donde su atributo “href” se llama a una ruta (ya explicaré como funcionan las rutas en django) o también con formularios se debe aclarar que cuando se hace un submit de un formulario, este envía información a cierto lugar en el cual se trata la información, en nuestro caso será una ruta.

```
<form id="formularioQ" action="/Queayranne/" method="POST">{% csrf_token %}
  <button id="passJsonJsToPy"   class="nav-link active">Queayranne</button>
</form>
```

En el atributo action se escribe el lugar donde queremos que vaya la información tratada esta información debe ir en elementos html inputs, en nuestro caso el lugar donde queremos que vaya la información es la ruta por tanto se pone la ruta que en el caso es “/Queayranne/”

```
from django.urls import path


from grafo.views import plantillaCanvas, guCanvas , mostrarGraphsBD, Queayranne, volverPrincipal,PartirUser

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',plantillaCanvas),


    path('guCanvas/',guCanvas),

    path('mostrarGraphsBD/',mostrarGraphsBD),

    path("Queayranne/",Queayranne),

    path("Queayranne/volverPrincipal/",volverPrincipal),
```

en django existe un archivo llamado urls.py en donde iran todas las rutas y las funciones a las que llama cuando se dirige información a dichas rutas, por ejemplo



en este momento se debe llamar a la función Queyranne

```
path("Queayranne/",Queayranne),
```

estas funciones se escriben en el documento views.py, cabe mencionar que en el archivo url se deben importar todas las funciones que se requieran del archivo views.py

```
def Queayranne(request):
```

esta función recibe como argumento un request debido a que es una solicitud a la que django debe responder

```
dictMenorAristas={
    "mAristas":menorAristas
}

return render(request,"canvasQ.html",{"j":dumps(circlesJson),"menorAristas":dumps(dictMenorAristas) })
```

esta función debe retornar el request, una plantilla de html y también se puede enviar un “contexto” esto es la información que queremos que el html reciba (se puede enviar diccionarios, arreglos, entre otros), ya cuando esta en nuestro html se puede tratar de todas las formas que necesitemos, es decir html será como un puente para traspasar información de Javascript a django y viceversa.

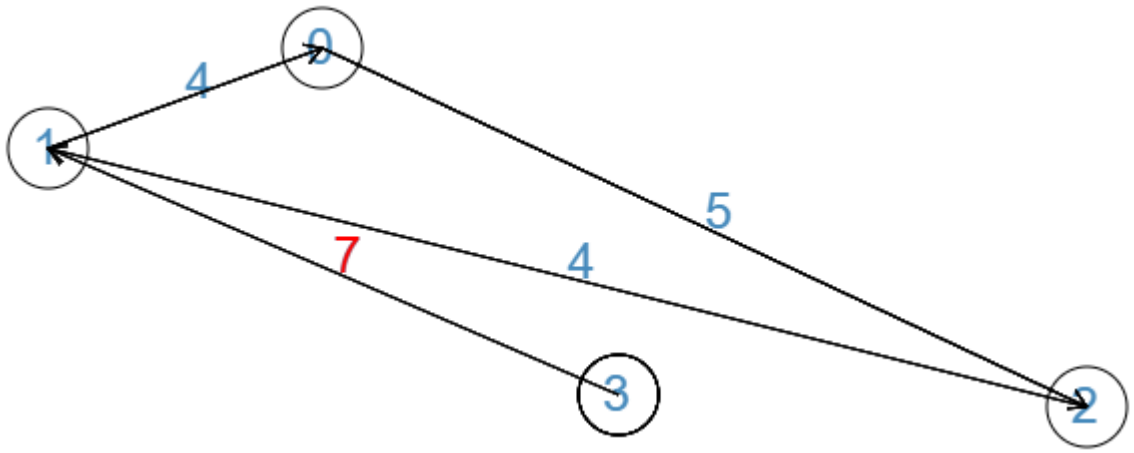
Para la implementación de estos dos algoritmos se realizan desde django con el lenguaje python, con el algoritmo Queayranne se recibe el grafo en formato Json desde javascript hacía la vista de python (la función), se adquieren los nodos del grafo, se hallan todas las biparticiones posibles y se hace un proceso de búsqueda para encontrar la partición donde se pierda la mínima cantidad de aristas con respecto al peso, esto se envía a una plantilla html y después al script de Javascript la cual grafica este nodo subrayando las aristas con el peso en rojo indicando así, que son las aristas que se deben romper.

La forma de adquirir las aristas con menor peso se realiza recorriendo una lista en donde están guardadas todas las posibles combinaciones del grafo partido en dos por ejemplo un grafo con nodos : A,B,C esta lista será [[[A,B] , [C]], [[B,C] , [A]], [[A,C] , [B]]] se verifican cuáles nodos tienen aristas con la función retornaAristasDeVertx, se debe ingresar el objeto json y dos nodos los cuales se quiere verificar si existe una arista que relacione ambos nodos

```python
def retornaAristasDeVertx(json,id1,id2):

    for i in json["graph"][0]["data"]:
        q=0
        if(i["id"]==id1 ):
            while q < len(i["linkedTo"]):
                if(i["linkedTo"][q]["nodeId"]==id2):
                    return i["linkedTo"][q]
                q+=1

    for i in json["graph"][0]["data"]:
        q=0
        if(i["id"]==id2 ):
            while q < len(i["linkedTo"]):
                if(i["linkedTo"][q]["nodeId"]==id1):
                    return i["linkedTo"][q]
                q+=1
```

Se compara el peso de las aristas que tengan relación y se realiza un proceso de comparación para encontrar el mínimo y se guardan los nodos de la arista y su arista

```
el menor [[1, 3], [{'nodeId': 1, 'peso': 7}]]
```



para este ejemplo los nodos 1 y 3 con su respectiva arista será la información que se envía a Javascript para graficar el grafo y pintar la arista de un color diferente.

Para el algoritmo de partir por usuario es algo similar al algoritmo anterior debido a que también se debe verificar la existencia de las aristas entre los nodos de los dos grupos con la función retornaAristasDeVertx y enviar un diccionario con estas aristas y ambos grupos de nodos para después pintar las aristas indicadas