Grafo Simpático

Funciones de apoyo:

```
public static String pre_Procesado(String cadena){
    cadena = cadena.toLowerCase();
    cadena = cadena.replace(";", "");
    cadena = cadena.replace(",", "");
    cadena = cadena.replace(".", "");
    return cadena;
}
public static void aumentoEdgePeso(String origen, String destino,GraphLA<String, String> graf ){
    for (Vertex<String, String> v : graf.getVertices()) {
        if (v.getData().equals(origen)) {

            for (Edge <String, String> e: v.getEgdes() ){
                if(e.getDestino().getData().equals(destino)){
                   e.setPeso(e.getPeso()+1);
                }
            }
        }
    }

}
```

Función principal:

```
public static GraphLA generarGafoSintactico(String cadena){
    String cadenaP = pre_Procesado(cadena);
    String[] palabras = cadenaP.split(" ");
     GraphLA<String, String> grafo = new GraphLA<>(true);
    if(palabras.length<2){
        System.out.println("No se puede crear un grafo de texto con 1 o 0 palabras");
        return null;
    }
    System.out.println(palabras.length);//18
    for (int i = 0; i < palabras.length-1; i++) {
        String origen= palabras[i];
        String destino= palabras [i+1];
        //la funccion add agrega en caso de que no exista el vertice
        //y si si existe, simplemente no hara nada, por este motivo
        //no se necesita validar los vertices
         grafo.addVertex(origen);
         grafo.addVertex(destino);
         boolean indicador=grafo.connect(origen,destino, null, 1);
         //devuelve false en caso de que el edge exista..
         if(indicador==false){
             //En ese caso lo unico que debemos hacer es
             //buscar el edge y aumentar su peso en 1
             aumentoEdgePeso(origen,destino,grafo);
         }
    }
    return grafo;
}
```

Sistema Académico

Función principal:

```
public static Map<String, Map<String,List<Student>>> getRegistros (GraphLA<Student, Map<String,String> > grafo){
     Map<String, Map<String,List<Student>>> elmapa= new HashMap();
     for (Vertex<Student, Map<String,String>> v: grafo.getVertices()){
        for(Edge<Map<String,String>,Student> e: v.getEgdes()){
            for(String clave: e.getData().keySet()){
                if(elmapa.containsKey(clave)){
                    if(elmapa.get(clave).containsKey(e.getData().get(clave))){
                        if(!elmapa.get(clave).get(e.getData().get(clave)).contains(e.getOrigen().getData())){
                            elmapa.get(clave).get(e.getData().get(clave)).add(e.getOrigen().getData());
                        }
                        if(!elmapa.get(clave).get(e.getData().get(clave)).contains(e.getDestino().getData())){
                            elmapa.get(clave).get(e.getData().get(clave)).add(e.getDestino().getData());
                        }
                    }
                    else{
                        String paraleloN = e.getData().get(clave);
                        List<Student> listaN =new ArrayList();
                        listaN.add(e.getOrigen().getData());
                        listaN.add(e.getDestino().getData());
                        elmapa.get(clave).put(paraleloN, listaN);
                    }
                }
                else{
                    Map<String,List<Student>> contenido = new HashMap();
                    List<Student> lista =new ArrayList();
                    lista.add(e.getOrigen().getData());
                    lista.add(e.getDestino().getData());
                    String paralelo = e.getData().get(clave);
                    contenido.put(paralelo, lista);
                    elmapa.put(clave, contenido);
                }
            }
        }
     }
    return elmapa;
}
```