Se implemento el algoritmo incluido en el articulo “An improved ant colony optimisation heuristic for graph colouring”.
Para ello se utilizaron las siguientes clases:

MatrizActualizacion:

Esta clase modela la matriz dij incluida en el articulo, ademas posee los métodos para su actualización y modificación.

MatrizRastro:

Esta clase modela la matriz tij incluida en el articulo, ademas posee los métodos para su actualización y modificación, **MatrizAdyacencias:**

Esta clase modela la matriz de adyacencias de la gráfica (aunque no se ocupa, pues se utiliza la lista de adyacencias de cada vértice).

Vertice:

Esta clase modela a un vértice de la gráfica, cada uno de estos tiene un identificador, un color asignado y una lista de adyacencias y los métodos para su manipulación.

AntCol:

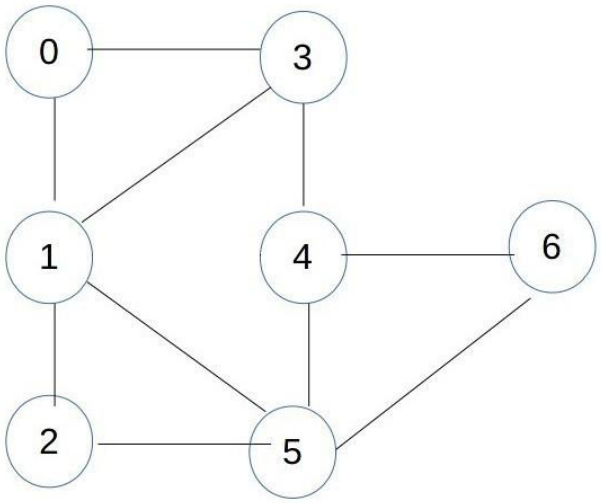
Esta es la clase encargada de implementar el algoritmo del articulo, llamando y construyendo objetos de las otras clases.

Ejecucion del Algoritmo:

Para ejecutar el algoritmo es necesario hacer desde una terminal:

java AntCol

La clase AntCol.java tiene definida una gráfica por defecto (el como cambiarla se explicara mas adelante).
La gráfica es:



Después de ejecutar el algoritmo obtenemos en la consola:

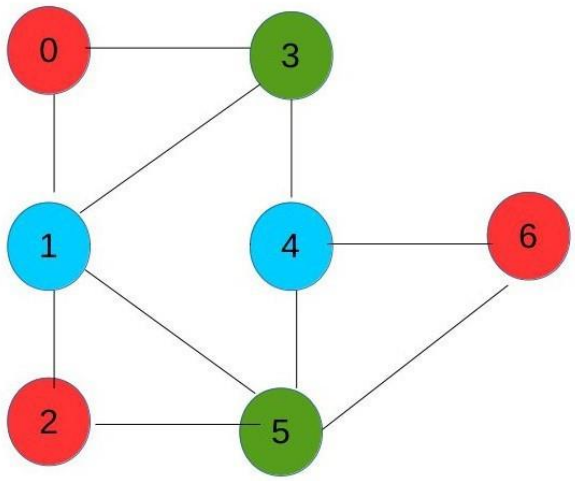
```
urxvt
# java AntCol


Al final la matriz delta dij fue :
0.0     0.0     2.5     0.0     2.333333333333333       0.8333333333333333      2.5
0.0     0.0     0.0     0.0     0.0     2.5     1.5833333333333333
1.3333333333333333      0.0     0.0     0.3333333333333333      0.8333333333333333      0.0     1.3333333333333333
0.0     0.0     1.5833333333333333      0.0     0.0     1.3333333333333333      1.3333333333333333
0.6666666666666666      1.9999999999999998      1.6666666666666665      0.0     0.0     0.0     2.333333333333333
0.5833333333333333      0.0     0.0     1.0     0.0     0.0     1.833333333333333
0.5     0.0     1.833333333333333       0.3333333333333333      0.0     0.0     0.0

Al final la matriz tij fue :
1.0E-323        0.0     1.0350665293349748      0.0     0.7423088937057336      0.4084935060638916      0.9400741116393598
0.0     1.0E-323        0.0     0.0     0.7675193895193559      0.5123564502749595      0.19719978528026072
1.0350665293349748      0.0     1.0E-323        0.5416162216902286      0.6091861780793967      0.0     1.2549657809219161
0.0     0.0     0.5416162216902286      1.0E-323        0.0     0.7065735772475594      0.4901764664426795
0.7423088937057336      0.7675193895193559      0.6091861780793967      0.0     1.0E-323        0.0     0.6153951030699212
0.4084935060638916      0.5123564502749595      0.0     0.7065735772475594      0.0     0.0     0.20712983323124226
0.9400741116393598      0.19719978528026072     1.2549657809219161      0.4901764664426795      0.6153951030699212      0.20712983323124226     1
.0E-323

Solucion final
[[v2, v0, v6], [v4, v1], [v5, v3]]
#
```

donde solución final es la ultima solución encontrada por el algoritmo.
Ademas muestra el estado final de las matrices definidas en el articulo (dij y tij).
Si coloreamos de acuerdo a esta solución final obtenemos:



que es una solución optima ya que la gráfica tiene triángulos, (no siempre regresa una solución optima, aunque generalmente si, se explicara el porque mas adelante).

Modificación de la Gráfica:
Para cambiar la gráfica a colorear es necesario modificar el archivo AntCol.java particularmente la sección de Vértices (crear los que hagan falta asegurándose que el identificador es único) y definiendo después sus adyacencias.
También en esa misma clases modificar el arreglo llamado arreglovertices para que incluya todos los vértices.
Después de esto hay que compilar haciendo: javac *.java

Parámetros.
En AntCol.java se usaron los siguientes (los definidos en el articulo):

```
numciclos=40;
numants=20;
p=0.8;
```

que definen el numero de ciclos, de hormigas y el rango de evaporación de las feromonas respectivamente.
Para modificarlos, hay que modificar AntCol.java y compilar.

Observaciones.
La implementación del algoritmo es fiel a la del articulo (incluyendo la actualizacion de las matrices) salvo la opción de seleccionPik (que en el articulo se explica de forma bastante vaga) en su lugar se uso la sugerencia de la maestra (usar una selección uniforme), esto impacta un poco de manera desfavorable al algoritmo ya que no toma en cuenta las soluciones anteriores (ni a las matrices de rastro) no asigna las probabilidades de selección de acuerdo a estas y es como si fuera el primer ciclo de ejecución.
Entonces se vuelve un algoritmo glotón sin memoria que puede corregirse (no se hizo por falta de tiempo) definiendo seleccionPik.

Nota
Se implemento el algoritmo en Slackware Linux 14, compilandolo con el OpenJDK 7 (por si sale un mensaje major minor versión o algo parecido, si ocurre mover los archivos .java a una nueva carpeta y hacer javac *.java).