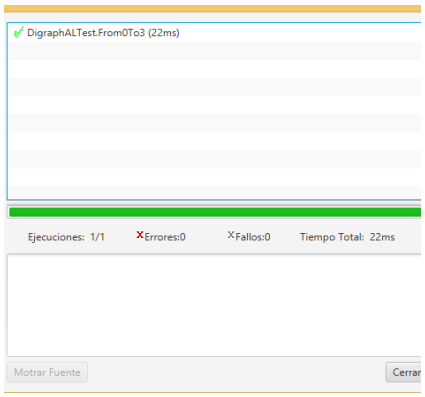
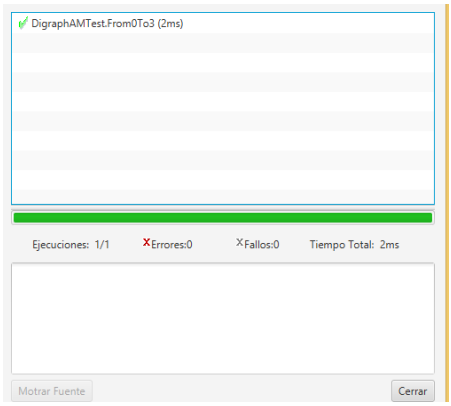
Laboratorio Nro. 6: implementación de grafos

Manuela Valencia Toro
Universidad Eafit
Medellín, Colombia
mvalenciat@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

1. 



2. **Ambos funcionan de manera similar, uno está implementado con listas y el otro con matrices, en ambos se desarrollaron métodos para insertar arcos, obtener peso y obtener sucesores, en uno simplemente se creó una matriz de tamaño ingresada por el usuario, y se fue llenando conforme ingresaba datos el usuario, y en la otra, se crearon 4 listas, y dos de estas se añadieron a las otras dos, y se hizo lo mismo que con el anterior.**
3. **Cuando se tienen una gran cantidad de datos es más conveniente usar listas de adyacencia, una de las razones es que esta ocupa menos espacio que la matriz, pero en situaciones en los que la cantidad de datos no es muy alta, puede usarse una matriz de adyacencia.**
4. **Es mejor usar listas de adyacencia, debido a la gran cantidad de datos que deben almacenarse en el grafo.**
5. **En mi consentimiento, es mejor usar listas de adyacencia, ya que tal y como lo mencioné anteriormente el espacio usado es menor que el usado por matrices de adyacencia, y aparte de esto, permite un rápida ejecución.**

6. **Es mejor usar listas de adyacencia, debido a que la cantidad de datos podría llegar a ser alta.**

7.
```
class Bipartite
{
  static int V=-1;
   public static void recibir(){
      String a= "";
       Scanner s = new Scanner (System.in);
      int matriz [] [];
      System.out.println("Ingrese nro de vertices");
      V = s.nextInt();
      while(V!=0){
      System.out.println("Ingrese nro de arcos");  // c + n
       s = new Scanner (System.in);       // c + n
      int arista=s.nextInt();                        // c + n
     System.out.println("Ingrese relaion");   // c + n
      s = new Scanner (System.in);   // c + n
      String conexiones = s.nextLine ();  // c + n
      matriz= new int [V] [V];   // c + n

      while(conexiones.length()>1){

matriz[(Integer.parseInt(conexiones.substring(0,1)))][(Integer.parseInt(conexiones.substring(2)))]=1;  //
c + n*m
           System.out.println("Ingrese relacion"); // c + n*m
          conexiones = s.nextLine ();   // c + n*m
      }
      if(isBipartite(matriz,0))   // c + n +O(m+n)
      a=a + "BICOLORABLE \n";else  // c + n
      a=a + "NOT BICOLORABLE \n"; // c + n
      V=Integer.parseInt(conexiones); // c + n
   }
   System.out.println(a);
} O(m*n)

   static boolean isBipartite(int G[][],int src)
   {
            int colorArr[] = new int[V];
      for (int i=0; i<V; ++i)
         colorArr[i] = -1;   // c + n


      colorArr[src] = 1;

      LinkedList<Integer>q = new LinkedList<Integer>();
      q.add(src);


      while (q.size() != 0)
      {

         int u = q.poll();  // c + n


         if (G[u][u] == 1) // c + n
```

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

```
            return false;  // c + n

          for (int v=0; v<V; ++v)
          {

            if (G[u][v]==1 && colorArr[v]==-1)
            {
              colorArr[v] = 1-colorArr[u];  // c + n*m
              q.add(v);     // c + n*m
            }

             else if (G[u][v]==1 && colorArr[v]==colorArr[u])
              return false;  // c + n *m
          }
        }
        return true;
      }
  } O(m*n)
```

O(m*n)

8. m representa la cantidad de vértices, y m representa la diferencia entre las variables v y V.

4) Simulacro de Parcial

	0	1	2	3	4	5	6	7
0				1	1			
1	1		1			1		
2					1		1	
3								1
4			1					
5								
6			1					
7								

1. .

2. 0 → 3,4
1 →0,2,5
2 → 4, 6
3 → 7
4 →2
5 →
6 →2
7 →

3. *A) 0,3,7 y 0,4,2,6,1,5*
B)0,3,4,7,1,2,5,6

4. *a*

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co