Universidad Nacional de Córdoba

Informe Trabajo Final

Programación Concurrente

Simulación de una red flexible de producción mediante Redes de Petri.

Autores:
Esteban Morales (35.104.714)
Marino Uboldi (34.140.737)
Santiago Salamandri (35.177.596)

Supervisor:
Mgs. Orlando Micolini

13 de marzo de 2014

Índice

1. Introducción

En el presente informe se expone el desarrollo del trabajo final para la cátedra Programación Concurrente que se cursa en el primer cuatrimestre del cuarto año de la carrera Ingeniería en Computación en la Facultad de Ciencias Exáctas Físicas y Naturales de la Universidad Nacional de Córdoba, Argentina. Este trabajo tiene por consigna la simulación de una celda flexible de producción mediante el modelado por redes de petri y la implementación concurrente programada en Java. Tal implementación incluye la programación de los procesos productivos que ejecutan sus tareas de forma paralela y el diseño de un monitor para garantizar la sincronización de la adquisición y devolución de los recursos compartidos.

2. Consigna

Sea una planta industrial donde se intenta producir tres tipos de piezas distintas mediante el uso de una celda de producción flexible que incluye tres robots, cuatro máquinas, tres almacenes de materiales y tres almacenes de piezas producidas. Debe tener en cuenta las siguientes reglas de producción para cada pieza:

- Pieza 1: Para producir una unidad de pieza (1), una unidad de material del almacen I_1 debe someterse a los porcesos de la máquina M_1 y la máquina M_2. Finalmente se colocan las piezas producidas en el almacen = O_1.

$$I_1 \rightarrow M_1 \rightarrow M_2 \rightarrow O_1$$

- Pieza 2: Existen dos reglas para producir una unidad de pieza (2), una unidad de material del almacen I_2 debe someterse a los porcesos de la máquina M_4 ,la máquina M_3 y la máquina M_1 ó bien pasar por las máquinas M_2, M_3 y M_1. Finalmente se colocan las piezas producidas en el almacen = O_2.

$$I_2 \rightarrow M_4 \rightarrow M_3 \rightarrow M_1 \rightarrow O_2$$

$$I_2 \rightarrow M_2 \rightarrow M_3 \rightarrow M_1 \rightarrow O_2$$

- Pieza 3:Para producir una unidad de pieza (1), una unidad de material del almacen I_3 debe someterse a los porcesos de la máquina M_3 y la máquina M_4. Finalmente se colocan las piezas producidas en el almacen = O_3.

$$I_3 \rightarrow M_3 \rightarrow M_4 \rightarrow O_3$$

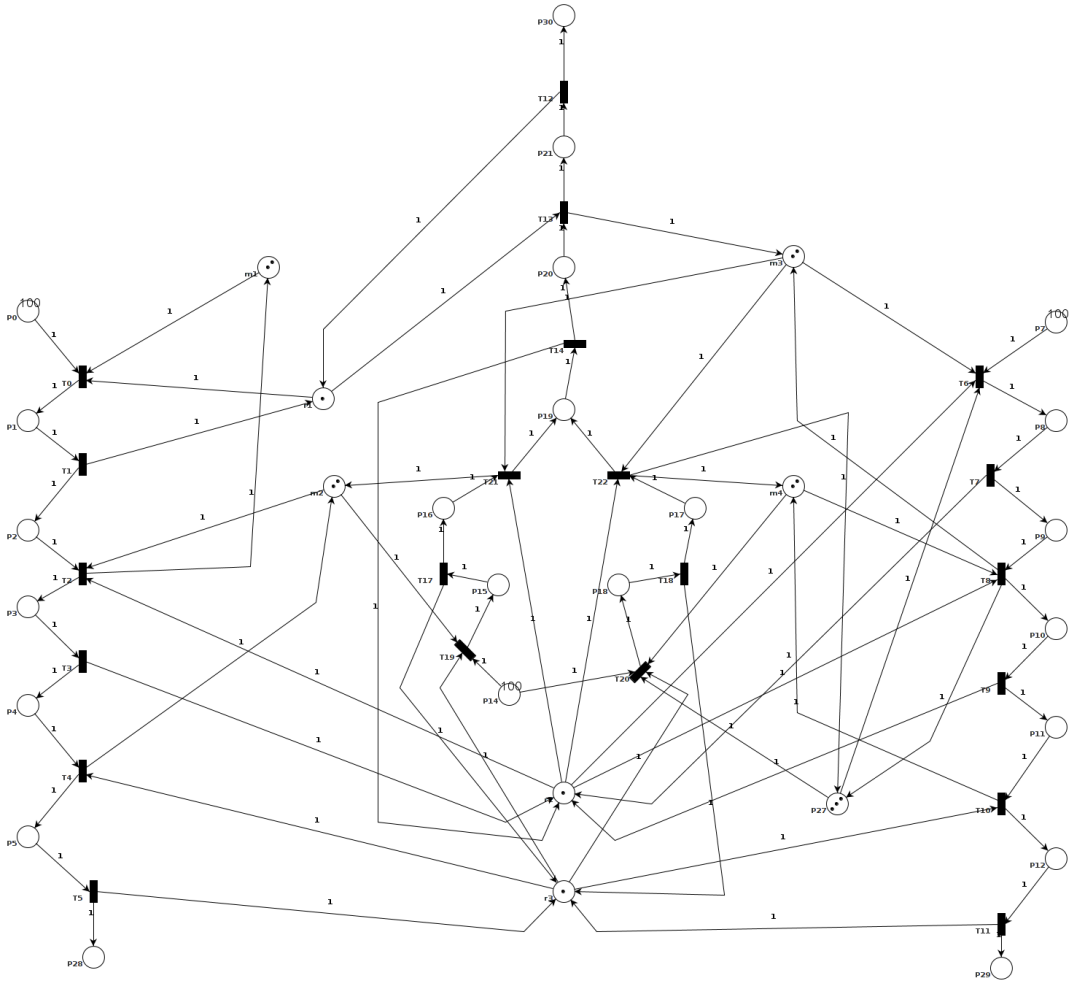
Los encargados de transportar los materiales, productos intermedios y finales son los robots, en este caso se cuenta con tres y cuyas responsabilidades se detallan a continuación:

- Robot 1 (R1):$[I_1, M_1, M_3, O_2]$
  - Carga la máquina M_1.
  - Libera la máquina M_3.
  - Quita materiales del almacen I_1.
  - Deposita las piezas (2) en el almacen O_2.
- Robot 2 (R2):$[I_3, M_1, M_2, M_3, M_4]$
  - Libera la máquina M_1.
  - Libera la máquina M_2.
  - Libera la máquina M_4.
  - Carga la máquina M_3.
  - Carga la máquina M_4.
  - Quita materiales del almacen I_3.
- Robot 3 (R3):$[M_2, M_4, 0_1, 0_3]$
  - Carga la máquina M_2.
  - Carga la máquina M_4.
  - Libera la máquina M_2.
  - Libera la máquina M_4.
  - Deposita las piezas (1) en el almacen O_1.
  - Deposita las piezas (3) en el almacen O_3.

Se pide como consigna modelar el sistema con una red de petri, luego debe programarse la solución y emplearse un monitor para la sincronización de los procesos concurrentes.

3. Red de Petri

Una Red de Petri es uno de varios lenguages de modelado matemático para la descripción de sistemas distribuidos, paralelos y concurrentes. Una Red de Petri es un grafo bipartito, en el cual los nodos representan transiciones(eventos) y plazas(condiciones o estados), los arcos dirigidos indican cuales plazas son precondiciones y/o poscondiciones para la ejecución de una transición. Mediante la interpretación semántica de la consigna se produjo 3 redes de petri que describen los procesos productivos del sistema propuesto en la consigna. 3 IMAGENES La ejecución de estas redes no presenta complicación alguna. Sin embargo al unirlas para construir la red completa del sistema aparecen deadlocks en la ejecución, interbloqueos producidos por el uso compartido de los recursos productivos. 1 IMAGEN Seguidamente se analiza la circunstancia de interbloqueo de la red y es posible observar, con cierta facilidad, que el deadlock se produce cuando el segundo y el tercer proceso intentan producir cuatro piezas en simultáneo. Naturalmente surge la idea de incorporar una restricción, la más obvia, limitar la producción simultanea de ambos procesos a sólo tres piezas. Esta solución acaba con los interbloqueos y en ejecuciones posteriores no se observan deadlocks independientemente de la cantidad de pasos simulados. Se muestra el modelo final de la red completa del sistema: 1 IMAGEN



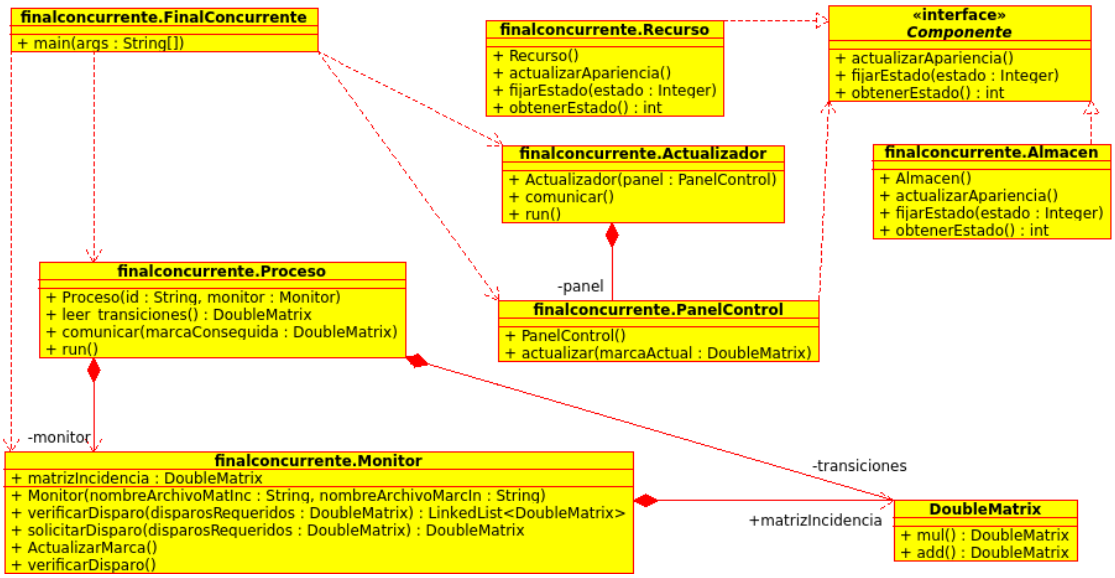
4. Clases

Con el objetivo de modularizar el código y procurar el encapsulamiento de cada parte, se decidió la implementación de las siguientes clases:

- Actualizador.java
- Componente.java
- FinalConcurrente.java
- Monitor.java

- PanelControl.java
- Proceso.java
- VentanaPanel.java

Se muestra a continuación el diagrama de clases y luego se desarrollará una breve descripción de las clases más importantes y se detallan los métodos más relevantes.



4.0.1. Administrador

Esta clase se encarga de administrar la simulación. Puede ver y modificar absolutamente todos los parámetros de la red Routers, computadoras, conexiones. Tiene la funcionalidad de calcular caminos óptimos con la implementación del algoritmo de Dijkstra para grafos direccionados y que admite ciclos no recurrentes. Adicionalmente es capáz de leer la configuración de la red desde un archivo y dibujar el grafo de la red haciendo uso de la librería graphvz mediante la aplicación dot de los repositorios de ubuntu. Cabe mensionar que se determinó que todos routers tienen la misma cantidad de maquinas conectadas.

4.0.2. Arco

Esta clase representa el arco o la arista de un grafo. Es la entidad que conecta dos nodos o vértices y que se le asigna un peso o distancia.

4.0.3. Biblioteca_paquetes

Esta clase es la estructura de datos que garantiza justicia en el envío de paquetes, esto es, igual atención a las solicitudes de todos los routers y para todas las páginas. En su interiror se implementa una colección de tipo anidada de tres niveles que se usa para organizar los paquetes. Esta colección es del tipo cola y utiliza el campo especial duenio para identificar a qué router y página pertenecen los paquetes encolados.

4.0.4. Computadora

Esta clase representa las computadoras o terminales conectadas a los routers en la red a simular. Incluye un puntero al router al que está conectada con el objetivo de solicitarle el envío de una nueva página. Adicionalmente posee una lista de páginas recibidas que podrá ser usada para medir la eficiencia de la red.

4.0.5. Conexion

Esta clase representa la vía que comunica dos routers en la red y hace referencia al arco del grafo de la red a simular. Consta de dos nros de ip, el origen y destino así como la especificación del ancho de banda y una cola para emular las limitaciones del canal.

4.0.6. Etiqueta

Esta clase implementa las triplas de tres enteros que utilizará la clase router para determinar hacia dónde debe enviar los paquetes.

4.0.7. Pagina

Esta clase representa las páginas que generan las computadoras y que contiene una identificación su tamaño en cantidad de paquetes y las direcciones de origen y de destino en forma de duplas de enteros.

4.0.8. Paquete

Esta clase representa las que componen una página y que contiene un número de órden, el tamanio de la página a la que pertenece y las direcciones de origen y de destino en forma de duplas de enteros.

4.0.9. Router

Esta clase representa los nodos de la red y opera como enrutador, tiene las funcionalidades de enviar y recibir paquetes así como recibi páginas completas desde las computadoras que están conectadas a este router. La recepción del paquete se realiza mediante la lectura de sus conexiones de recepción y el envío mediante la carga de las conexiones de envío. Cada router tiene una identificación entera, una colección de computadoras, una lista de conexiones de envío y otra de recepción, además cuenta con una tabla de enrutamiento que contiene las etiquetas para cada destino.

5. Segunda Fase: Implementación del Algoritmo

Algoritmo Implementado: Dijkstra

El algoritmo de Dijkstra resuelve el problema de los caminos más cortos y orígen único en un grafo dirigido y ponderado $G = (V, E)$ tal que todos los pesos de los arcos son no negativos. Con una buena implementación el timepo de ejecución del algoritmo de Dijkstra es menor al de Bellman-Ford.

¿Para qué sirve el algoritmo?

La función del algoritmo es calcular los caminos mínimos que hay entre el nodo origen y todos los nodos del grafo. *¿Cuál es la lógica del algoritmo?*

El algoritmo parte de un nodo inicial, y se va desplazando a través de la red, guiándose por los nodos adyacentes al nodo actual. Conforme el algoritmo evoluciona se encuentran mejores caminos para determinado nodo destino por lo que se actualiza el resultado. Finalmente el algoritmo termina cuando no quedan nodos sin camino mínimo encontrado.

Dijkstra mantiene un conjunto S de nodos para los que ya se calcularon los caminos mínimos. El algoritmo cíclicamente selecciona un nodo $u \in (V - S)$ cuya

distancia es la mínima entre los que quedan sin procesar, agrega el nodo seleccionado u a S y RELAJA todos los vértices adyacentes a u.

```
    DIJKSTRA(G, nodo_inicio)
1   INICIALIZAR(G)
2   S = ∅
3   Q = G.V
4   while (Q ≠ ∅)
5       nodo_sel=MAS CERCANO(Q)
6       S = S ∪ nodo_sel
7       for each nodo_i ∈ G.Adyacentes[nodo_sel]
8           RELAJAR(u,v)
```

¿Como fue implementada dicha lógica? Los elementos que utilizamos son:

- Conjunto Q: conjunto de nodos no chequeados
- Conjunto S: conjunto de nodos chequeados
- Conjunto Adyacentes: conjunto de nodos adyacentes al nodo inicio
- Vector Predecesores: guarda los valores de los router predecesores a cada router
- Vector Distancias: almacena las distancias de los caminos mas cortos a cada router
- Lista de Etiquetas: contiene las etiquetas correspondiente a cada uno de los routers, con los valores en el siguiente formato
  `{ r_destino, r_siguiente, peso del trayecto }`

Primeramente, cargamos en conjunto Q , Adyacentes, las etiquetas, haciendo distinción para la etiqueta del nodo inicio `{ orden_nodo_inicio, 0,0 }` y las demás `{ orden_nodo_x, -1, INF }`. Luego recorremos en conjunto Q buscando en las etiquetas de cada uno de los routers, la que contenga el menor trayecto. Con esta condición, seleccionamos el nodo actual o elegido, y lo pasamos al conjunto S. Luego recorremos los arcos, buscando aquellos que tengan origen en el nodo actual, y que su destino no corresponda a un nodo contenido en S. Es entonces,

cuando realizamos la comparación de distancias (trayectos almacenados en las etiquetas), con la distancia resultante de la suma de la distancia del trayecto desde el nodo inicio, al nodo actual, las la del trayecto del nodo actual al siguiente para determinar si actualizar o no los valores tanto en el peso de trayectoria, perteneciente a la etiqueta del router siguiente, como en el Vector Distancias y Predecesores.

Seguidamente, normalizo y actualizo los valores en el Vector Predecesor, con los correspondientes valores de nodos adyacentes al nodo inicio, lo que nos permite posteriormente, armar las tablas para dicho router. Finalmente, Se actualiza el campo `r_siguiente` de la etiqueta con el valor correspondiente, calculado en dicho Vector de Predecesores. Terminando así la ejecución del algoritmo, devolviendo la tabla con los nuevos valores de redireccionamiento.

¿Qué diferencia tiene con Floyd-Warshall? Ambos algoritmos tienen la misma finalidad, obtener el calculo del camino mas corto. Pero la diferencia es que Dijkstra calcula los caminos partiendo desde un único nodo base, hasta los demás del grafo, mientras Floyd aplica lo mismo, pero para todos los nodos de la red. Es decir, si el costo de calculo de Dijkstra es $n \times n$ operaciones, el costo de FFloyd-Warshall será $n \times n \times n$

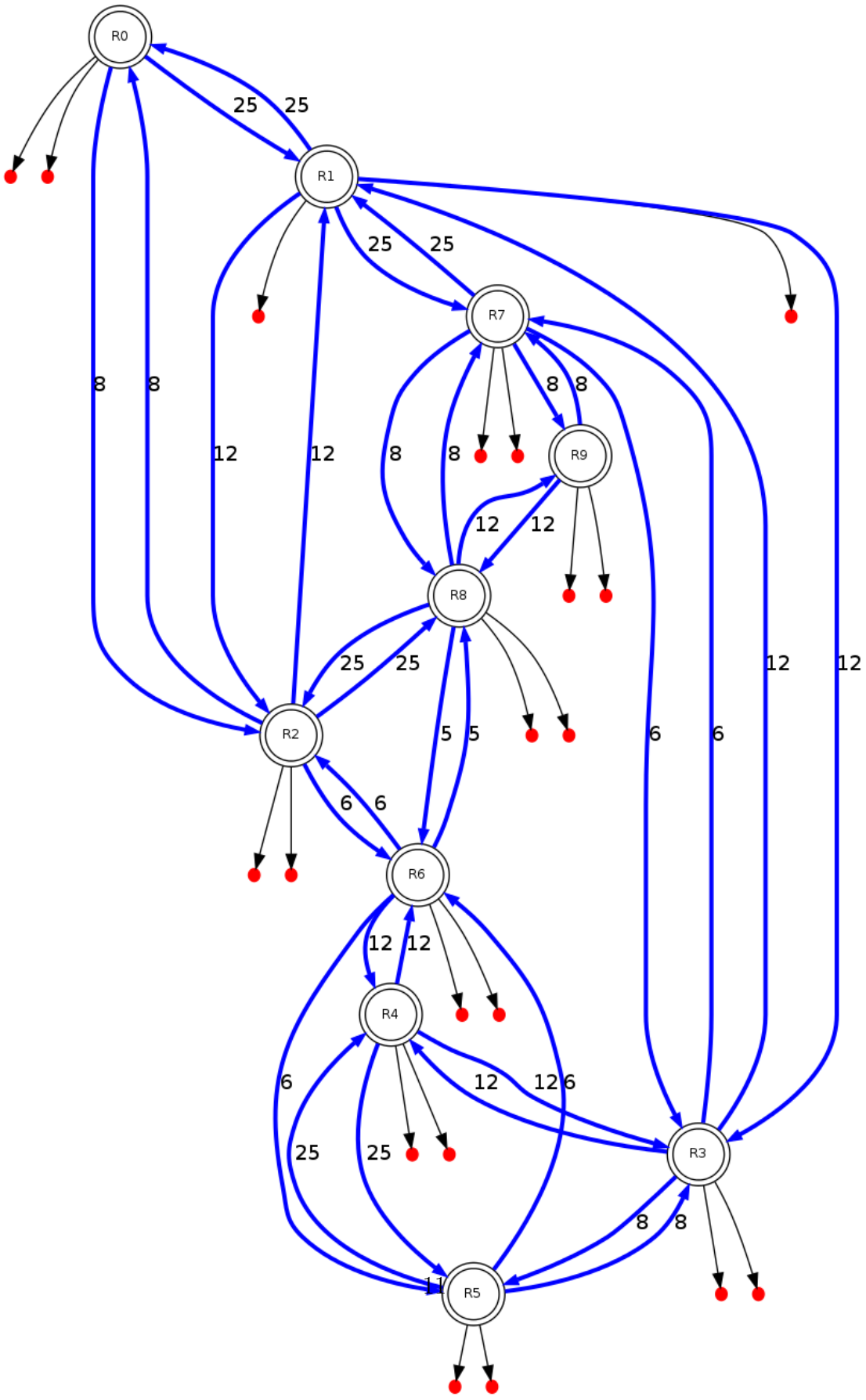
6. Tercera Fase: Pruebas

Para la validacion del modelo, se realizaron corridas del programa para dos configuraciones con 5, y 10 routers. Realizamos capturas de pantallas de (a)Red y (b)Configuración inicial de Arcos para evidenciar la correcta recepción de información a partir del archivo de texto leído.

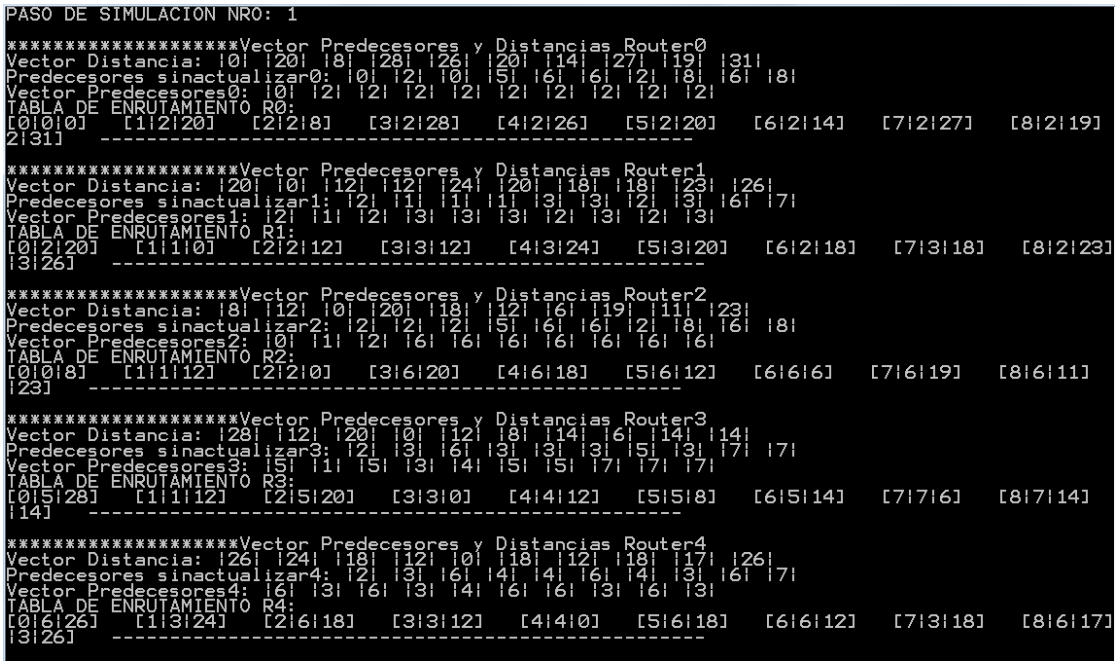
También capturamos los resultados arrojados por el algoritmo: (c)en cuanto a distancias → (c)Vector Distancia (d)respecto al router siguiente a direccionar → (d)Vector Predecesores (e)el resultado final para cada router → (e)Tablas.

Finalmente, se realiza el seguimiento visual de los paquetes de la página (en amarillo), a lo largo de las conexiones de la red hasta que llega a Destino.

Caso 1: Red de 10 Routers, se realiza el seguimiento de la Pagina 5, de tamaño 5, que se dirige desde el Router 7 al Router 4.


R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
25
25
25
25
8
8
8
8
8
8
12
12
12
12
12
12
25
25
5
5
6
6
6
6
12
12
12
12
6
6
25
25
8
8
11

La trayectoria del paquete, según dijkstra, será a través de los siguientes routers: $7 \rightarrow 3 \rightarrow 4$ con un peso de 18. Lo podemos ver en las tablas calculadas del direccionamiento

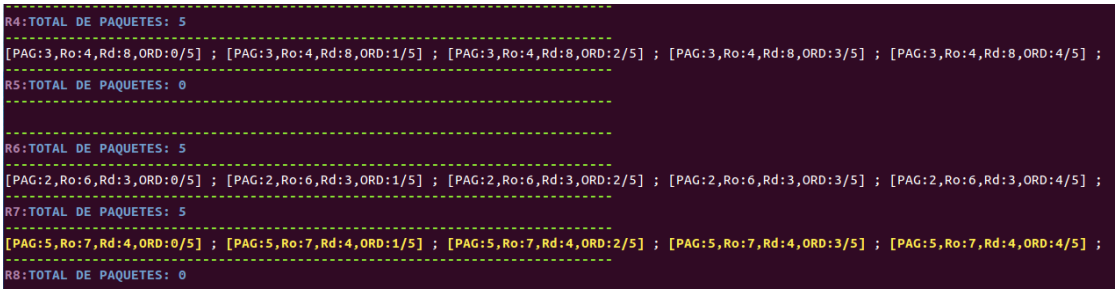


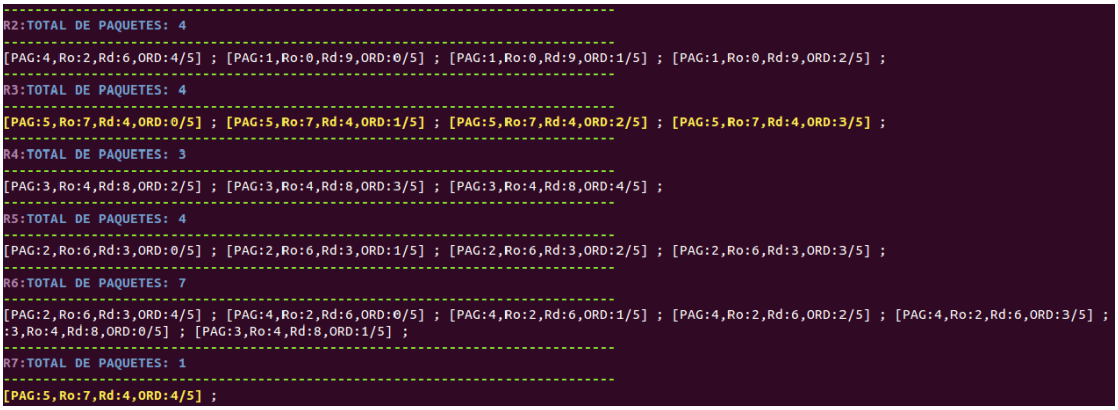
```
********************Vector Predecesores y Distancias Router5
Vector Distancia: |20| |20| |12| |8| |18| |0| |6| |14| |11| |22|
Predecesores sinactualizar5: |2| |3| |6| |5| |6| |5| |5| |3| |6| |7|
Vector Predecesores5: |6| |3| |6| |3| |6| |5| |6| |3| |6| |3|
TABLA DE ENRUTAMIENTO R5:
[0|6|20]   [1|3|20]   [2|6|12]   [3|3|8]   [4|6|18]   [5|5|0]   [6|6|6]   [7|3|14]   [8|6|11]
|22]    ------------------------------------------------

********************Vector Predecesores y Distancias Router6
Vector Distancia: |14| |18| |6| |14| |12| |6| |0| |13| |5| |17|
Predecesores sinactualizar6: |2| |2| |6| |5| |6| |6| |6| |8| |6| |8|
Vector Predecesores6: |2| |2| |2| |5| |4| |5| |6| |8| |8| |8|
TABLA DE ENRUTAMIENTO R6:
[0|2|14]   [1|2|18]   [2|2|6]   [3|5|14]   [4|4|12]   [5|5|6]   [6|6|0]   [7|8|13]   [8|8|5]
17]    ------------------------------------------------

********************Vector Predecesores y Distancias Router7
Vector Distancia: |27| |18| |19| |6| |18| |14| |13| |0| |8| |8|
Predecesores sinactualizar7: |2| |3| |6| |7| |3| |3| |8| |7| |7| |7|
Vector Predecesores7: |8| |3| |8| |3| |3| |3| |8| |7| |8| |9|
TABLA DE ENRUTAMIENTO R7:
[0|8|27]   [1|3|18]   [2|8|19]   [3|3|6]   [4|3|18]   [5|3|14]   [6|8|13]   [7|7|0]   [8|8|8]
|8]    ------------------------------------------------

********************Vector Predecesores y Distancias Router8
Vector Distancia: |19| |23| |11| |14| |17| |11| |5| |8| |0| |12|
Predecesores sinactualizar8: |2| |2| |6| |7| |6| |6| |8| |8| |8| |8|
Vector Predecesores8: |6| |6| |6| |7| |6| |6| |6| |7| |8| |9|
TABLA DE ENRUTAMIENTO R8:
[0|6|19]   [1|6|23]   [2|6|11]   [3|7|14]   [4|6|17]   [5|6|11]   [6|6|5]   [7|7|8]   [8|8|0]
|12]    ------------------------------------------------

********************Vector Predecesores y Distancias Router9
Vector Distancia: |31| |26| |23| |14| |26| |22| |17| |8| |12| |0|
Predecesores sinactualizar9: |2| |3| |6| |7| |3| |3| |8| |9| |9| |9|
Vector Predecesores9: |8| |7| |8| |7| |7| |7| |8| |7| |8| |9|
TABLA DE ENRUTAMIENTO R9:
[0|8|31]   [1|7|26]   [2|8|23]   [3|7|14]   [4|7|26]   [5|7|22]   [6|8|17]   [7|7|8]   [8|8|12]
|9|0]    ------------------------------------------------
```

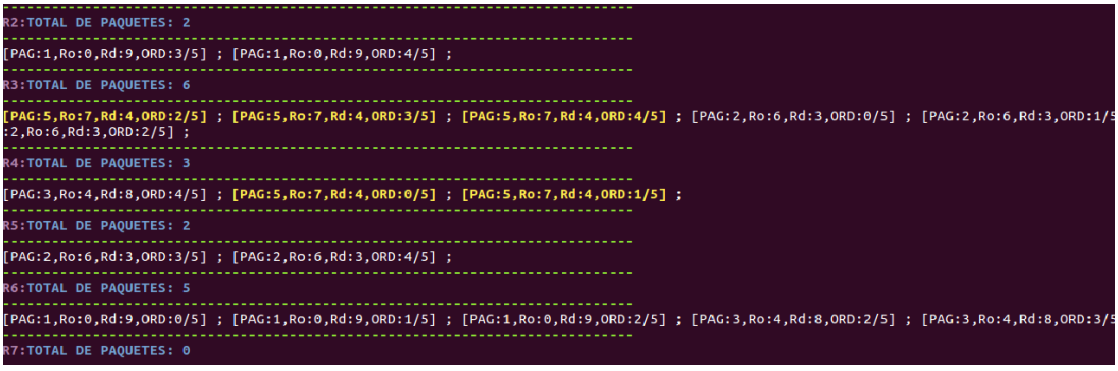
Comenzamos el seguimiento de los paquetes de color amarillo.

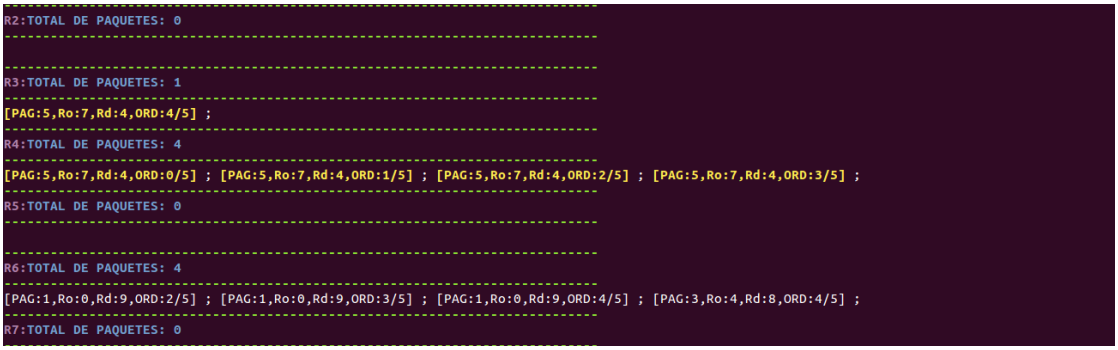
```
Pagina: 5
tamanio: 5
ip origen: 7--1
ip destino: 4--1
Elija una página de las creadas para seguir el trayecto de sus paquetes.
Ingrese el id de la página:
5
```

```
-----------------------------------------------ESTADO INICIAL-----------------------------------------

--------------------------------------------------------------------------------
R0:TOTAL DE PAQUETES: 5
--------------------------------------------------------------------------------
[PAG:1,Ro:0,Rd:9,ORD:0/5] ; [PAG:1,Ro:0,Rd:9,ORD:1/5] ; [PAG:1,Ro:0,Rd:9,ORD:2/5] ; [PAG:1,Ro:0,Rd:9,ORD:3/5] ; [PAG:1,Ro:0,Rd:9,ORD:4/5] ;
--------------------------------------------------------------------------------
R1:TOTAL DE PAQUETES: 0
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
R2:TOTAL DE PAQUETES: 5
--------------------------------------------------------------------------------
[PAG:4,Ro:2,Rd:6,ORD:0/5] ; [PAG:4,Ro:2,Rd:6,ORD:1/5] ; [PAG:4,Ro:2,Rd:6,ORD:2/5] ; [PAG:4,Ro:2,Rd:6,ORD:3/5] ; [PAG:4,Ro:2,Rd:6,ORD:4/5] ;
--------------------------------------------------------------------------------
```

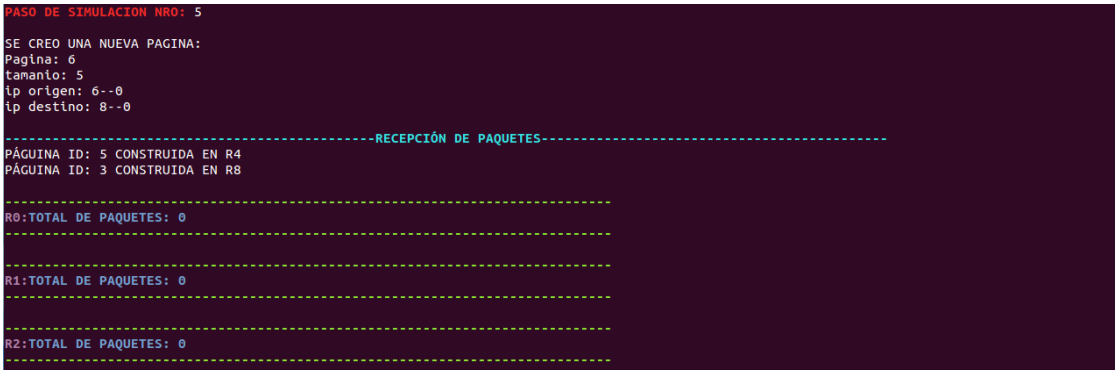

R4:TOTAL DE PAQUETES: 5
[PAG:3,Ro:4,Rd:8,ORD:0/5] ; [PAG:3,Ro:4,Rd:8,ORD:1/5] ; [PAG:3,Ro:4,Rd:8,ORD:2/5] ; [PAG:3,Ro:4,Rd:8,ORD:3/5] ; [PAG:3,Ro:4,Rd:8,ORD:4/5] ;
R5:TOTAL DE PAQUETES: 0
R6:TOTAL DE PAQUETES: 5
[PAG:2,Ro:6,Rd:3,ORD:0/5] ; [PAG:2,Ro:6,Rd:3,ORD:1/5] ; [PAG:2,Ro:6,Rd:3,ORD:2/5] ; [PAG:2,Ro:6,Rd:3,ORD:3/5] ; [PAG:2,Ro:6,Rd:3,ORD:4/5] ;
R7:TOTAL DE PAQUETES: 5
[PAG:5,Ro:7,Rd:4,ORD:0/5] ; [PAG:5,Ro:7,Rd:4,ORD:1/5] ; [PAG:5,Ro:7,Rd:4,ORD:2/5] ; [PAG:5,Ro:7,Rd:4,ORD:3/5] ; [PAG:5,Ro:7,Rd:4,ORD:4/5] ;
R8:TOTAL DE PAQUETES: 0


R2:TOTAL DE PAQUETES: 4
[PAG:4,Ro:2,Rd:6,ORD:4/5] ; [PAG:1,Ro:0,Rd:9,ORD:0/5] ; [PAG:1,Ro:0,Rd:9,ORD:1/5] ; [PAG:1,Ro:0,Rd:9,ORD:2/5] ;
R3:TOTAL DE PAQUETES: 4
[PAG:5,Ro:7,Rd:4,ORD:0/5] ; [PAG:5,Ro:7,Rd:4,ORD:1/5] ; [PAG:5,Ro:7,Rd:4,ORD:2/5] ; [PAG:5,Ro:7,Rd:4,ORD:3/5] ;
R4:TOTAL DE PAQUETES: 3
[PAG:3,Ro:4,Rd:8,ORD:2/5] ; [PAG:3,Ro:4,Rd:8,ORD:3/5] ; [PAG:3,Ro:4,Rd:8,ORD:4/5] ;
R5:TOTAL DE PAQUETES: 4
[PAG:2,Ro:6,Rd:3,ORD:0/5] ; [PAG:2,Ro:6,Rd:3,ORD:1/5] ; [PAG:2,Ro:6,Rd:3,ORD:2/5] ; [PAG:2,Ro:6,Rd:3,ORD:3/5] ;
R6:TOTAL DE PAQUETES: 7
[PAG:2,Ro:6,Rd:3,ORD:4/5] ; [PAG:4,Ro:2,Rd:6,ORD:0/5] ; [PAG:4,Ro:2,Rd:6,ORD:1/5] ; [PAG:4,Ro:2,Rd:6,ORD:2/5] ; [PAG:4,Ro:2,Rd:6,ORD:3/5] ;
:3,Ro:4,Rd:8,ORD:0/5] ; [PAG:3,Ro:4,Rd:8,ORD:1/5] ;
R7:TOTAL DE PAQUETES: 1
[PAG:5,Ro:7,Rd:4,ORD:4/5] ;


R2:TOTAL DE PAQUETES: 2
[PAG:1,Ro:0,Rd:9,ORD:3/5] ; [PAG:1,Ro:0,Rd:9,ORD:4/5] ;
R3:TOTAL DE PAQUETES: 6
[PAG:5,Ro:7,Rd:4,ORD:2/5] ; [PAG:5,Ro:7,Rd:4,ORD:3/5] ; [PAG:5,Ro:7,Rd:4,ORD:4/5] ; [PAG:2,Ro:6,Rd:3,ORD:0/5] ; [PAG:2,Ro:6,Rd:3,ORD:1/5
:2,Ro:6,Rd:3,ORD:2/5] ;
R4:TOTAL DE PAQUETES: 3
[PAG:3,Ro:4,Rd:8,ORD:4/5] ; [PAG:5,Ro:7,Rd:4,ORD:0/5] ; [PAG:5,Ro:7,Rd:4,ORD:1/5] ;
R5:TOTAL DE PAQUETES: 2
[PAG:2,Ro:6,Rd:3,ORD:3/5] ; [PAG:2,Ro:6,Rd:3,ORD:4/5] ;
R6:TOTAL DE PAQUETES: 5
[PAG:1,Ro:0,Rd:9,ORD:0/5] ; [PAG:1,Ro:0,Rd:9,ORD:1/5] ; [PAG:1,Ro:0,Rd:9,ORD:2/5] ; [PAG:3,Ro:4,Rd:8,ORD:2/5] ; [PAG:3,Ro:4,Rd:8,ORD:3/5
R7:TOTAL DE PAQUETES: 0



A consecuencia del ancho de banda de las Conexiones involucradas en la trayectoria y la demanda de paquetes, en 5to pasos de simulación se logra completar la página 5.



7. Conclusion

Finalizado el desarrollo del trabajo, de la ejecución del algoritmo de dijkstra pudimos apreciar que es una herramienta de gran utilidad y eficiencia. Aplicamos la propiedad de popularización, junto con un controlador de versiones, la tarea de programación fue más ligera, con el diseño previamente establecido. Además Pudimos dimensionar la potencialidad de las estructuras utilizadas: Nodo, Lista, Cola, Grafo. Inclusive, fue empleamos la estructura Cola, de manera iterativa, con el fin de conseguir igualdad al momento de selección de paquetes para enviar.

Referencias

[1] Cormen, T. C ; Leiserson, C.E ; Rivest, R.L and Stein, C *Introduction to Algorithms*, Third Edition The MIT Press. Cambridge, Massachusetts London, England, 2009 *Single-Source Shortest Paths*, 24:658–662.

[2] http://www.cplusplus.com/forum/articles/10627/ *Headers and Includes: Why and How* 2009

[1] http://choorucode.com/2010/07/22/c-fixing-cyclic-dependencies/ *C++: Fixing Cyclic Dependencies* 2010