# CAPÍTULO 5

**5. APLICACIÓN**

En el presente capítulo se realiza una descripción de la implementación del algoritmo en el lenguaje de programación C++ y se ilustra el funcionamiento del algoritmo de la ruta sucesiva más corta mediante un ejemplo.

**5.1. Representación de la Red**

Para representar la red en el computador se empleó un arreglo de tamaño N, en la celda i-ésima del mismo se almacena lo siguiente:

1. La oferta o demanda del nodo .
2. El desequilibrio del nodo .
3. El potencial del nodo .
4. Distancias más cortas desde un nodo seleccionado por el algoritmo al nodo i . de menor costo desde el nodo seleccionado por el algoritmo al nodo i.
5. Un puntero que señala a una lista enlazada en una dirección que representa la lista de adyacencia correspondiente al nodo i.

Para cada nodo j perteneciente a la lista de adyacencia del nodo i, existirá una celda en la lista enlazada mediante puntero, cuyos campos serán:

1. El número del nodo j.
2. El costo original de transportar una unidad de flujo a lo largo del arco .
3. Capacidad de flujo en el arco .
4. El costo reducido del arco respecto a los potenciales de los nodo i y j.
5. El flujo inicial entre el nodo i y el nodo j.

**5.2. Ilustración del Algoritmo**

Cabe resaltar que el algoritmo resuelve el problema dual de flujo de costo mínimo, maximizando el flujo en la red residual correspondiente. El primer paso consiste en inicializar en cero los potenciales de cada nodo y calcular la oferta o demanda de los nodos, .

A continuación se presenta una red con sus respectivos costos, capacidades y flujo inicial en cada arco.



En la red se observa que el nodo 1, es un nodo de oferta y el nodo 4

es un nodo de demanda. Los desequilibrios de los nodos son inicializados con sus correspondientes ofertas o demandas. La siguiente figura es la red residual inicial para un flujo , por lo que es igual a la red original. Además se observan los potenciales inicializados en cero y los desequilibrios de cada nodo inicializados con sus respectivos .



En esta red los costos reducidos son iguales a los costos originales debido a que los potenciales son iguales a cero.



Una vez determinados los valores de los desequilibrios de cada nodo es posible inicializar los conjuntos E y D, por lo tanto:

 y 

Luego se selecciona un nodo k perteneciente al conjunto E y un nodo l perteneciente al conjunto D, es decir k=1 y l=4. Posteriormente, mediante el algoritmo de Dijkstra se determinan las distancias de las rutas más cortas desde el nodo K a todos los otros nodos en G(x) con respecto a los costos reducidos y se obtiene el vector . Se define a P como la ruta de menor costo entre los nodos k y l, por lo que P=1-3-4. Conocidas las distancias más cortas desde el nodo k a los demás nodos se deben actualizar los nodos potenciales, por lo tanto se tiene que . A continuación se presenta la red residual con los potenciales y los

costos reducidos actualizados.

Se define a  como: . En este caso  sería igual a 2. Finalmente se aumentan  unidades de flujo a lo largo de la ruta P, por lo que se tienen que actualizar los desequilibrios restando las dos unidades de  si el arco sale del nodo i y sumando las dos unidades a  si el arco llega al nodo i. Finalmente se debe actualizar la red residual, por lo que al término de la iteración se obtiene la siguiente red residual.



El procedimiento se repite hasta que todos los desequilibrios sean iguales a cero. En la última iteración se obtienen las siguientes redes residuales:



A partir de la red residual resultante se debe obtener la solución del problema primal. Se debe tener en cuenta que para un arco  de la red original



y



Por lo tanto el flujo óptimo es el que consta en la siguiente figura:



Se observa que el costo asociado a la solución es de 14 unidades, menor que el de la solución factible inicial, cuyo costo era de 18 unidades. Cabe mencionar que para la implementación del algoritmo se diseñó un clase denominada *Red, q*ue consta de una estructura llamada *celda*, y contiene información correspondiente a cada arco como: costo, capacidad, flujo y un puntero tipo *celda* que permite construir la lista de adyacencia para cada nodo. Además la clase consta de una estructura denominada *nodo*, que se utilizará para almacenar la información de cada nodo, como el potencial, su desequilibrio, la distancia de un nodo fuente hasta el nodo, un campo llamado *pred*, para almacenar la ruta desde el nodo fuente hasta el nodo, y un puntero tipo *celda*, que apuntará al inicio de la lista de adyacencia correspondiente.

Finalmente, la clase consta de las funciones miembro necesarias para acceder y manipular la información de los nodos y arcos, así como una función que corresponde al algoritmo de Dijkstra para calcular las distancias más cortas desde un nodo fuente hasta los demás nodos. La implementación se diseñó bajo el supuesto que desde un nodo de oferta siempre será posible hallar una ruta hasta un nodo de demanda, es decir que la red sea fuertemente conexa. Es posible encontrar en internet problemas para probar este tipo de algoritmos, una dirección para los mismos la podemos encontrar en la bibliografía del presente trabajo investigativo.