ANEXO IV: ALGORITMO DE DIJKSTRA:

Nuestra situación es la siguiente: tenemos un conjunto conexo de nodos (siempre hay, al menos, un camino que une un par de nodos cualquiera).

A la hora de calcular la distancia entre dos nodos, estamos interesados en el camino más corto.

Desde un punto de vista matemático, disponemos de un conjunto finito de nodos *(V)*, un conjunto de conexiones o aristas *(E)*, y un peso (en nuestro caso, *longitud*) para cada una de las aristas *(vector w)*. Es decir, tenemos un grafo ponderado *(V, E, w)* , y el problema de calcular el camino más corto desde un origen (dado un nodo inicial $i_0 \in V$, determinar para cada otro vértice $i \in V$, $i \neq i_0$ el camino más corto de i_0 a i).

El procedimiento para ello es ir construyendo una sucesión de árboles $t_1 \subset t_2 \subset \ldots \subset t_n$, de modo que cada árbol t_j se construye a partir del anterior t_{j-1}, añadiendo a este último, el nodo que menos diste de i_0 en ese momento (y que no esté en t_{j-1}).

El algoritmo para construir el camino más corto es entonces como sigue:

1) Fijar el árbol inicial t_1 con i_0 únicamente y asignar la distancia $d_1 = 0$.

2) Para $m=2,3,\ldots,n$.

   2.1) Encontrar $i \in t_{m-1}$ y $j \notin t_{m-1}$, tales que:

   $$d_i + l_{ij} = \min (d_l + l_{lk}),$$

   de entre todos los nodos $l \in t_{m-1}$ y $k \notin t_{m-1}$.

   2.2) Asignar la distancia $d_j \leftarrow d_i + l_{ij}$.

   2.3) Etiquetar $l_j = i$.

Es claro que con este procedimiento, en cada caso determinamos la distancia mínima y el camino más corto a un nuevo nodo, pues si hubiese una distancia d'_j menor que $d_j = d_i + l_{ij}$ al nodo j, debería ser a través de un camino encontrado posteriormente, pero en el momento en que dicho camino abandonase t_{m-1}, ya sería su longitud más larga que d_j, pues esta se ha elegido

como la longitud más pequeña con que se puede incrementar cualquier camino de t_{m-1} a un nodo fuera de t_{m-1}.

Sin embargo, tal y como está el algoritmo, tiene excesiva repetición de operaciones innecesarias. En cada paso se vuelven a inspeccionar elementos que ya habían sido inspeccionados en el caso anterior.

Fue el holandés Dijkstra quien sugirió como evitar esta repetición de operaciones. Si escribimos la matriz de distancias por filas como:

$$A = \begin{bmatrix} a_1^T \\ \vdots \\ \vdots \\ a_n^T \end{bmatrix},$$

la idea del algoritmo de Dijkstra es, en cada paso tomar el mínimo componente a componente del vector de distancias d y el vector $d_{ja} + a^T_{ja}$ donde ja es el último nodo etiquetado, esto es:

$$d \leftarrow \left[\min\left(\begin{bmatrix} d_1 \\ d_{ja} + a_{ja1} \end{bmatrix} \right), \min\left(\begin{bmatrix} d_2 \\ d_{ja} + a_{ja2} \end{bmatrix} \right), \ldots, \min\left(\begin{bmatrix} d_n \\ d_{ja} + a_{jan} \end{bmatrix} \right) \right]$$

donde j_a es nodo etiquetado en el paso anterior.

De esta manera se aprovechan las comparaciones hechas en el paso anterior, pues se almacena en d los mínimos calculados por columnas en el paso anterior. Dicho de otro modo: d lleva siempre las distancias mínimas hasta ahora encontradas.

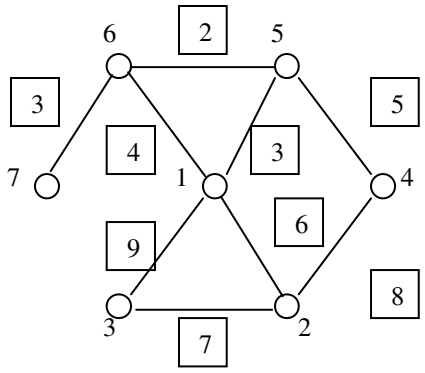
Para etiquetar un nuevo nodo se encuentra el mínimo en d de entre las componentes aún no etiquetadas.

Ahora bien, al comparar d con $d_{ja} + a^T_{ja}$, para tomar el mínimo, si el valor mínimo se encuentra en $d_{ja} + a^T_{ja}$ sabremos que la etiqueta del nuevo nodo será ja, pero si se alcanza en d, como d contiene los mínimos de todas las filas $d_j + f$ inspeccionadas anteriormente, no sabremos cómo etiquetar el nuevo nodo. Para evitar esta situación, podemos utilizar el vector l, haciendo que sus componentes l_i indiquen a que fila $d_j + a_j$ corresponde el correspondiente valor de d_i. Dicho de otro modo, en todo momento: $d_i = d_{lj} + a_{lj\,i}$ $\quad i = 1, \ldots, n.$

Se han creado dos funciones para facilitar la programación de este algoritmo. La primera es **'distancia_nodos'** que se encargará de recibir los dos argumentos imprescindibles en el algoritmo, éstos son:

- La matriz **S** (2 x n, donde n es el número total de enlaces), que representa entre qué nodos hay enlaces.
- El vector **w**, que guarda los pesos (en nuestro caso longitudes) de los lados o enlaces asignados en la matriz S anterior.

En el siguiente grafo ponderado en las aristas, podemos ver un ejemplo de cómo se introducirían la matriz S, y el vector w.



$$S = \begin{bmatrix} 1 & 1 & 3 & 2 & 4 & 5 & 5 & 6 & 6 \\ 3 & 2 & 2 & 4 & 5 & 1 & 6 & 1 & 7 \end{bmatrix}$$

w = [9 6 7 8 5 3 2 4 3]

A continuación , la función irá tomando uno a uno todos los nodos asignándoles, en cada iteración, la etiqueta de 'nodo inicial del arbol' (i_0), e invocando a la función **'dijkstra'** (a la que le envía los argumentos *S,w,* i_1), esta función siguiendo, el algoritmo anteriormente explicado, hallará los caminos más cortos entre i_0 y el resto de nodos del grafo, quedando guardados en el vector *d*, que es devuelto a la función **'distancia_nodos'** que se encargará de almacenarlo en la fila i_0 de la matriz *D*, que irá creándose poco a poco, completándose cuando sean analizadas las distancias entre todos los nodos del grafo, es decir, cuando i_0 sea igual al último nodo (en el ejemplo: $i_0 = 7$).

Se mostrarán ahora las funciones programadas:

Función **'distancia_nodos':**

```
function[distancias]=distancia_nodos(S,w)
% llama a dijkstra para calcular la matriz de distancias entre todos los nodos
n=max(max(S));
D=zeros(n,n);
for i=1:n
   i0=i;
   [l,d]=dijkstra(S,w,i0)
   D(i,:)=d
end
distancias=D;
```

Función **'Dijkstra':**

```
function [l,d]=dijkstra(S,w,i0)
% programa para encontrar un arbol generador
% minimal en el grafo con lista de lados S
% y pesos correspondientes w,
% mediante el algoritmo de dijkstra,
% tomando como nodo inicial del arbol i0.
% el etiquetado de dicho arbol se devuelve en el vector l.

% inicializando las variables.
n=max(max(S));

% creando la matriz de adyacencia con pesos
A=zeros(n,n);
A(:,:)=inf;
for k=1:length(S)
   i=S(1,k); j=S(2,k);
   A(i,j)=w(k);
   A(j,i)=w(k);
end
d=zeros(1,n);
```

```
d(:)=inf;
d(i0)=0;

l=zeros(1,n);
l(:)=inf;
l(i0)=0;

% N, lista de nodos etiquetados.
% Nr, nodos sin etiquetar.
N=[i0]
Nr=[1:i0-1,i0+1:n];

% el primer paso aparte
[amin,k]=min(A(i0,Nr));
% la columna j donde se alcanza el minimo correponde a Nr(k)
j=Nr(k);
if amin<inf
   l(j)=i0;
   d(j)=amin;
   N=[N,j];
   Nr=[Nr([1:k-1]),Nr([k+1:length(Nr)])];
   A(j,:)=A(j,:)+amin;
end
% lno, numero de nodos no etiquetados.
% lsiv, numero de nodos etiquetados en el paso anterior
lno=length(Nr);
lsiv=1;

% ciclo principal
while (length(N)>lsiv) & lno>0
   lsiv=length(N);

   % en vmin, los minimos de cada columna
   % y en l las filas donde se alcanza.
   [vmin, l]=min(A(N,Nr));

   % en amin el m´inimo de vmin,
```

```
    % y en k la columna donde se alcanza.
    [amin,k]=min(vmin);

    % como hemos mirado en la submatriz A(N,Nr)
    % recuperamos ahora los valores en A
    if(amin<inf)
       j=Nr(k);
       i=N(I(k));
       I(j)=i;
       d(j)=amin;
       A(j,:)=A(j,:)+amin;
    % aumentamos N y reducimos Nr
       N=[N,j];
       Nr=[Nr([1:k-1]),Nr([k+1:length(Nr)])];
       lno=lno-1;
    end
end
```