Uso de R

Asignatura: **Investigación Operativa II**, 2° Grado en Estadística

Autor: Miguel Rodríguez Rosa

Caminos mínimos. Algoritmo de Dijkstra

Cargamos la librería “igraph”:

```
library(igraph)
```

Creamos las etiquetas para los nodos:

```
etiquetas<-c("New York","Cleveland","St. Louis", "Nashville","Phoenix","Dallas","Salt Lake City","Los Angeles")
```

Creamos la matriz de adyacencia (0 indica que no hay un arco):

```
matriz<-matrix(c(
  0, 400, 950, 800,    0,   0,    0,    0,
  0,   0,   0,   0, 1800, 900,    0,    0,
  0,   0,   0,   0, 1100, 600,    0,    0,
  0,   0,   0,   0,    0, 600, 1200,    0,
  0,   0,   0,   0,    0,   0,    0,  400,
  0,   0,   0,   0,  900,   0, 1000, 1300,
  0,   0,   0,   0,    0,   0,    0,  600,
  0,   0,   0,   0,    0,   0,    0,    0
),ncol=length(etiquetas),byrow=TRUE)
```

Ponemos las etiquetas a las filas y las columnas de la matriz de adyacencia:

```
rownames(matriz)<-colnames(matriz)<-etiquetas
```

Creamos el grafo ponderado a partir de la matriz de adyacencia:

```
grafo<-graph_from_adjacency_matrix(matriz, weighted=TRUE)
```

Si algunos de los pesos es faltante, se transforma en 0:

```
E(grafo)[is.na(E(grafo)$weight)]$weight<-0
```

Coloreamos los nodos y los arcos (por ejemplo en gris):

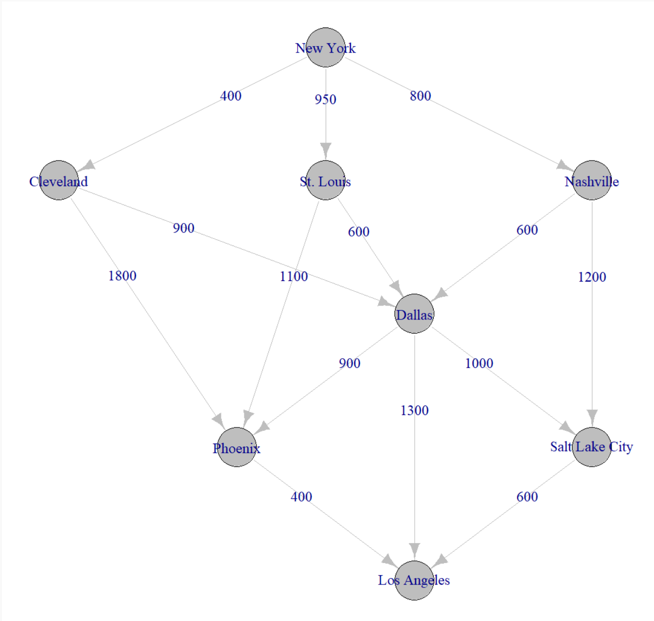
```
V(grafo)$color<-"grey"
E(grafo)$color<-"grey"
```

Especificamos que queremos que disponga el grafo en forma de árbol descendente desde el nodo origen al destino (estillo llamado “Sugiyama”):

```
disposicion<-layout_with_sugiyama(grafo)
```

Dibujamos el grafo, donde las etiquetas de los arcos serán sus pesos:

```
plot(grafo,edge.label=E(grafo)$weight,layout=disposicion)
```



Calculamos el camino mínimo con el algoritmo de Dijkstra, desde “New York” hasta “Los Angeles”, especificando que nos devuelva tanto los nodos como los arcos de la solución:

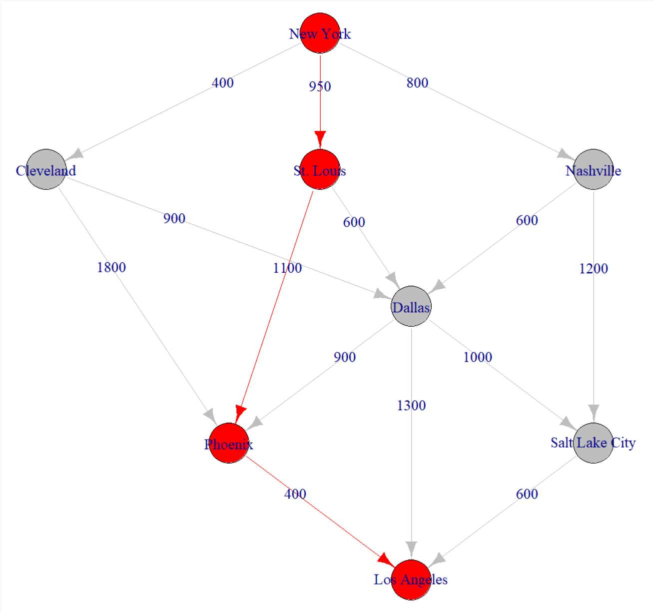
```
camino<-shortest_paths(grafo,from="New York",to="Los Angeles",
    algorithm="dijkstra",output="both")
```

Coloreamos los nodos y los arcos que forman la solución (por ejemplo en rojo):

```
V(grafo)[camino$vpath[[1]]]$color<-"red"
E(grafo)[camino$epath[[1]]]$color<-"red"
```

Dibujamos el grafo, ahora con la solución superpuesta:

```
plot(grafo, edge.label=E(grafo)$weight, layout=disposicion)
```



Calculamos la longitud del camino mínimo:

```
sum(camino$epath[[1]]$weight)
```

```
[1] 2450
```

Flujo máximo. Algoritmo de Ford-Fulkerson

Cargamos la librería “igraph”:

```
library(igraph)
```

Creamos las etiquetas para los nodos:

```
etiquetas<-c("New York","Chicago","Memphis","Denver","Dallas","Los Angeles")
```

Creamos la matriz de adyacencia (0 indica que no hay un arco):

```
matriz<-matrix(c(
  0, 500, 400,   0,   0,   0,
  0,   0,   0, 300, 250,   0,
  0,   0,   0, 200, 150,   0,
  0,   0,   0,   0,   0, 400,
  0,   0,   0,   0,   0, 350,
  0,   0,   0,   0,   0,   0
),ncol=length(etiquetas),byrow=TRUE)
```

Ponemos las etiquetas a las filas y las columnas de la matriz de adyacencia:

```
rownames(matriz)<-colnames(matriz)<-etiquetas
```

Creamos el grafo ponderado a partir de la matriz de adyacencia:

```
grafo<-graph_from_adjacency_matrix(matriz,weighted=TRUE)
```

Si algunos de los pesos es faltante, se transforma en 0:

```
E(grafo)[is.na(E(grafo)$weight)]$weight<-0
```

Coloreamos los nodos y los arcos (por ejemplo en gris):

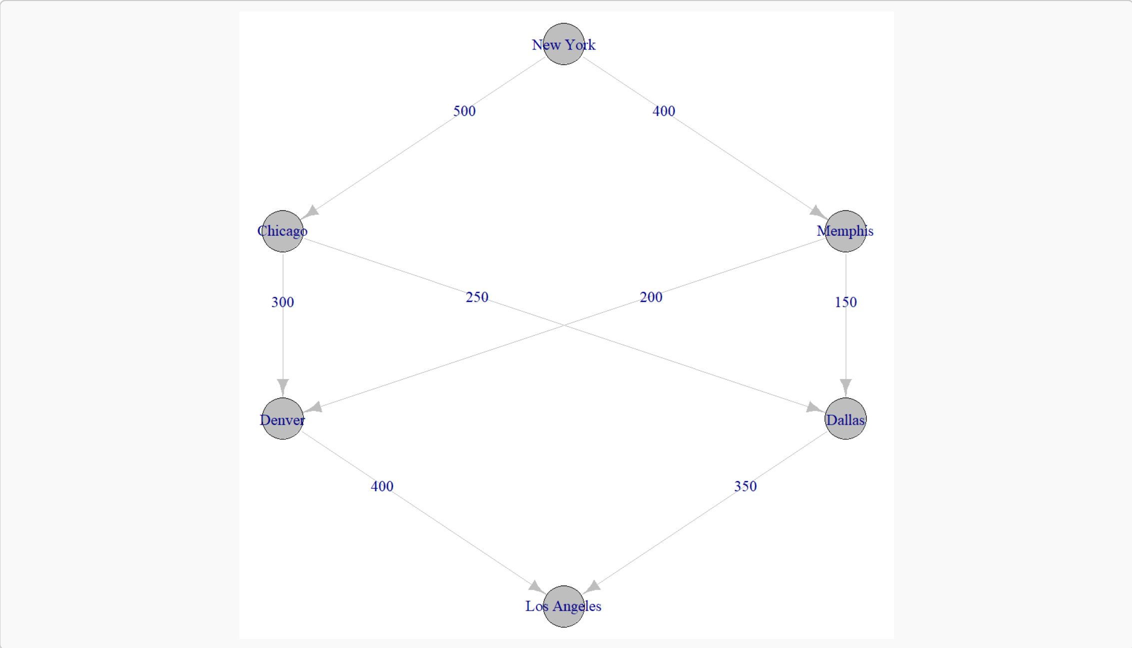
```
V(grafo)$color<-"grey"
E(grafo)$color<-"grey"
```

Especificamos que queremos que disponga el grafo en forma de árbol descendente desde el nodo origen al destino (estillo llamado “Sugiyama”):

```
disposicion<-layout_with_sugiyama(grafo)
```

Dibujamos el grafo, donde las etiquetas de los arcos serán sus pesos:

```
plot(grafo,edge.label=E(grafo)$weight,layout=disposicion)
```



Calculamos la distribución del flujo máximo por los arcos, desde “New York” hasta “Los Angeles”, especificando que los pesos de los arcos son las capacidades máximas:

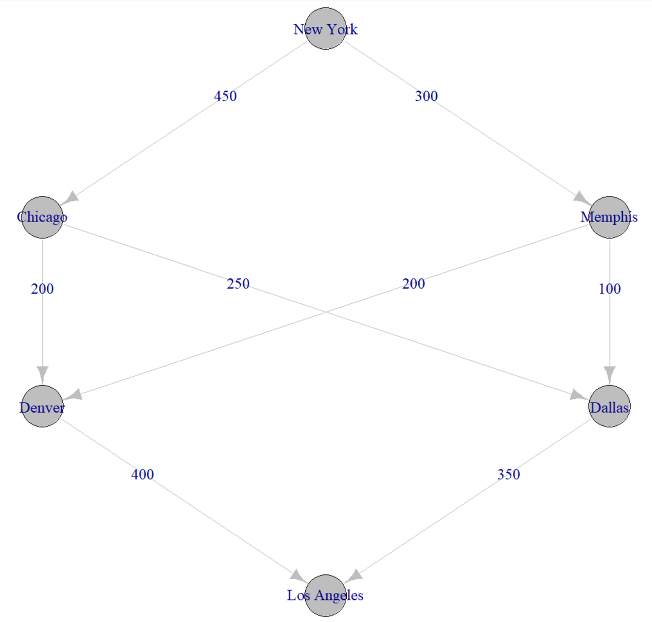
```
flujo<-max_flow(grafo,"New York","Los Angeles",capacity=E(grafo)$
    weight)
```

Escribimos la solución en las etiquetas de los arcos:

```
E(grafo)$weight<-flujo$flow
```

Dibujamos el grafo, ahora con la solución superpuesta:

```
plot(grafo,edge.label=E(grafo)$weight,layout=disposicion)
```



Calculamos el flujo máximo:

```
flujo$value
```

```
[1] 750
```

Planificación de proyectos. Algoritmo CPM

Cargamos las librerías “igraph” y “critpath”:

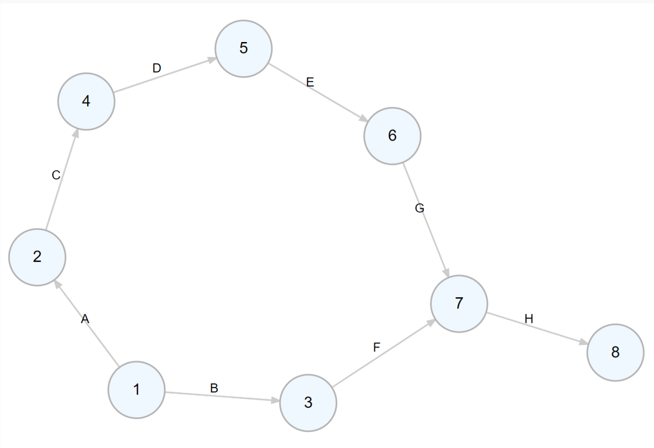
```
library(igraph)
library(critpath)
```

Creamos la tabla con las actividades, las predecesoras de cada una, y sus duraciones:

```
matriz<-data.frame(
  "actividades"=  c("A", "B", "C", "D", "E", "F", "G",   "H"),
  "predecesoras"= c(NA,  NA,  "A", "C", "D", "B", "E", "F,G"),
  "duracion"=     c( 6,   5,   3,   2,   3,   4,   4,     2))
```

Dibujamos la red AOA, especificando que tenemos datos de predecesoras:

```
plot_graphAOA(matriz,TRUE)
```



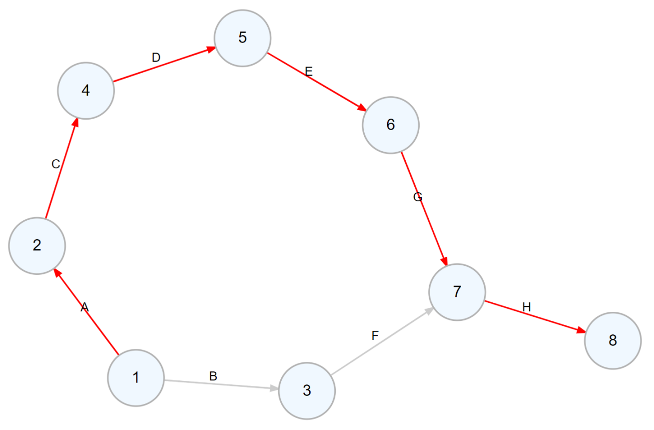
Calculamos el camino crítico con el algoritmo CPM, especificando que tenemos datos de predecesoras y que las duraciones son conocidas (por eso llevamos a cabo un CPM) y el tiempo de terminación del proyecto:

```
camino<-solve_pathAOA(matriz,TRUE,TRUE)
```

```
[1] Completion time: 20
```

Dibujamos la red AOA, ahora con la solución superpuesta:

```
plot_graphAOA(solved=camino)
```



Planificación de proyectos. Algoritmo PERT

Cargamos las librerías “igraph” y “critpath”:

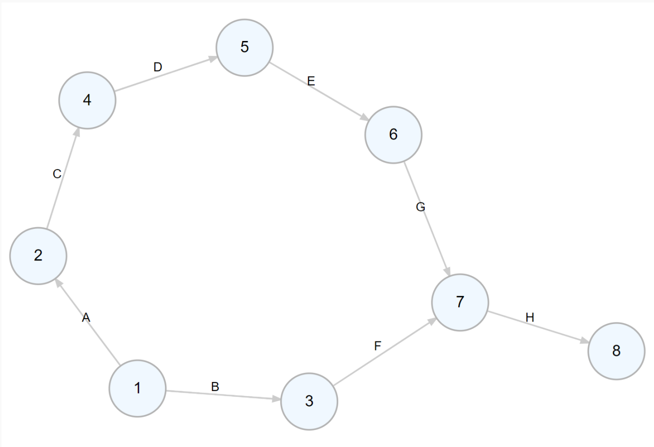
```
library(igraph)
library(critpath)
```

Creamos la tabla con las actividades, las predecesoras de cada una, y sus duraciones más optimistas, más probables y más pesimistas:

```
matriz<-data.frame(
  "actividades"=  c("A",  "B",  "C",  "D",  "E",  "F",  "G",    "H"),
  "predecesoras"= c(NA,   NA,   "A",  "C",  "D",  "B",  "E",  "F,G"),
  "optimista"=    c( 2,    4,    2,    1,    1,    3,    2,      0),
  "probable"=     c( 6,    5,    3,    2,    3,    4,    4,      2),
  "pesimista"=    c(10,    6,    4,    3,    5,    5,    6,      4))
```

Dibujamos la red AOA, especificando que tenemos datos de predecesoras:

```
plot_graphAOA(matriz,TRUE)
```



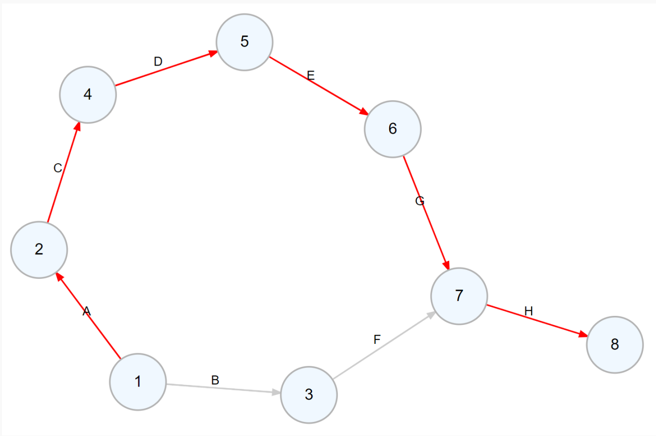
Calculamos el camino crítico con el algoritmo PERT, especificando que tenemos datos de predecesoras y que las duraciones son inciertas (por eso llevamos a cabo un PERT) y la distribución del tiempo de terminación del proyecto:

```
camino<-solve_pathAOA(matriz,FALSE,TRUE)
```

```
[1] Expected compl. time distribution: N( 20 , 1.825742 )
```

Dibujamos la red AOA, ahora con la solución superpuesta:

```
plot_graphAOA(solved=camino)
```



Calculamos el tiempo de antelación con el que hay que comenzar el proyecto para tener una confianza del 99 % de completarlo a tiempo:

```
invisible(PERT_newtime(0.99,camino))
```

```
[2] New expected compl. time: 24.24731
```

Calculamos la probabilidad de terminar el proyecto antes de 12 unidades de tiempo:

```
invisible(PERT_newprob(12,camino))
```

```
[3] Prob. of completion: 5.88567e-06
```

Árbol de expansión mínima (o máxima). Algoritmo de Prim

Cargamos la librería “igraph”:

```
library(igraph)
```

Creamos las etiquetas para los nodos:

```
etiquetas<-c("New York","Cleveland","St. Louis", "Nashville","Phoenix","Dallas","Salt Lake City","Los Angeles")
```

Creamos la matriz de adyacencia (0 indica que no hay un arco):

```
matriz<-matrix(c(
  0, 400, 950, 800,    0,   0,    0,    0,
  0,   0,   0,   0, 1800, 900,    0,    0,
  0,   0,   0,   0, 1100, 600,    0,    0,
  0,   0,   0,   0,    0, 600, 1200,    0,
  0,   0,   0,   0,    0,   0,    0,  400,
  0,   0,   0,   0,  900,   0, 1000, 1300,
  0,   0,   0,   0,    0,   0,    0,  600,
  0,   0,   0,   0,    0,   0,    0,    0
),ncol=length(etiquetas),byrow=TRUE)
```

Ponemos las etiquetas a las filas y las columnas de la matriz de adyacencia:

```
rownames(matriz)<-colnames(matriz)<-etiquetas
```

Creamos el grafo ponderado a partir de la matriz de adyacencia, especificando que es un grafo no dirigido donde los pesos están solamente en la parte triangular superior de la matriz de adyacencia:

```
grafo<-graph_from_adjacency_matrix(matriz,mode="upper",weighted=TRUE)
```

Si algunos de los pesos es faltante, se transforma en 0:

```
E(grafo)[is.na(E(grafo)$weight)]$weight<-0
```

Coloreamos los nodos y los arcos (por ejemplo en gris):

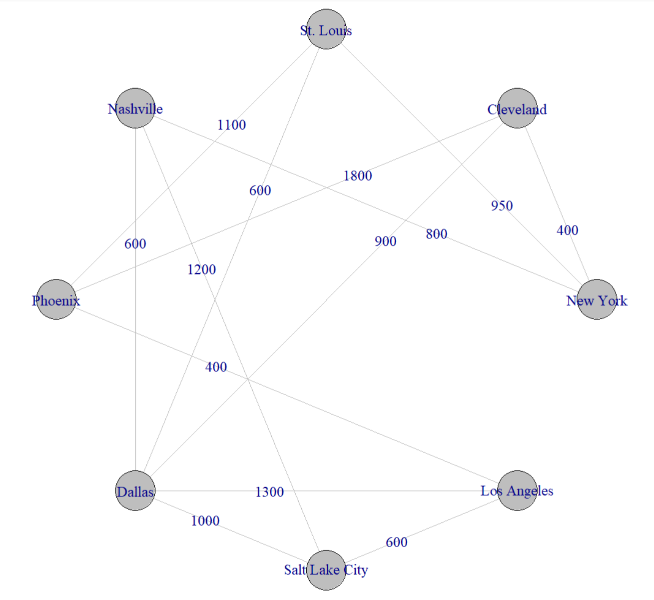
```
V(grafo)$color<-"grey"
E(grafo)$color<-"grey"
```

Especificamos que queremos que disponga el grafo en forma de círculo:

```
disposicion<-layout_in_circle(grafo)
```

Dibujamos el grafo, donde las etiquetas de los arcos serán sus pesos:

```
plot(grafo,edge.label=E(grafo)$weight,layout=disposicion)
```



Calculamos el árbol de expansión mínima con el algoritmo de Prim:

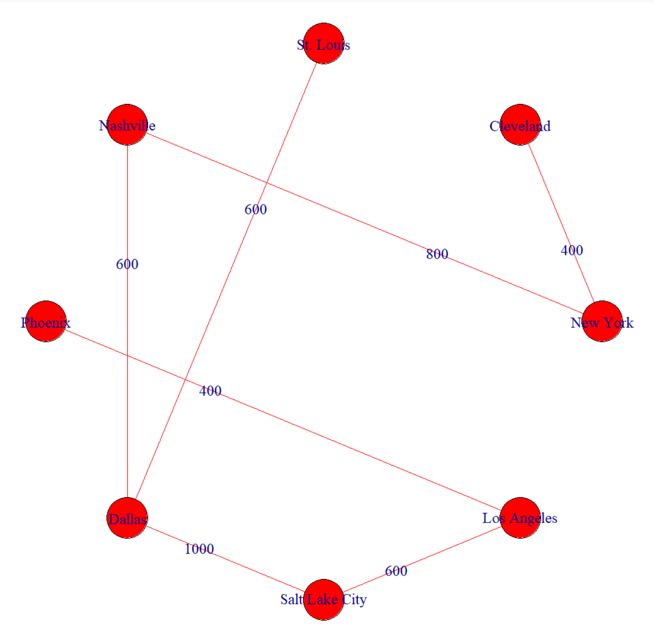
```
arbol<-mst(grafo,algorithm="prim")
```

Coloreamos los nodos y los arcos que forman la solución (por ejemplo en rojo):

```
V(arbol)$color<-"red"
E(arbol)$color<-"red"
```

Dibujamos el árbol de expansión mínima:

```
plot(arbol,edge.label=E(arbol)$weight,layout=disposicion)
```



Calculamos la longitud del árbol de expansión mínima:

```
sum(E(arbol)$weight)
```

```
[1] 4400
```