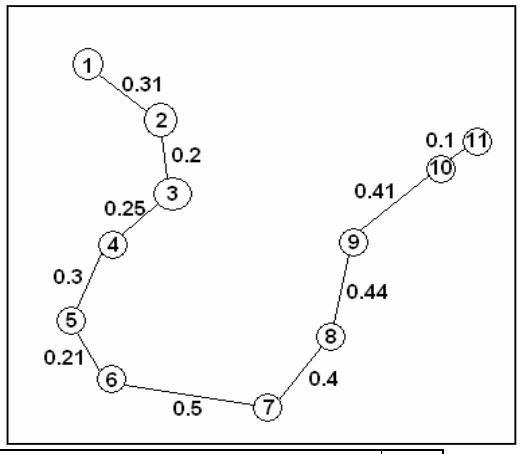
COMPUTACIÓN 1
Instituto de Computación

1er Parcial - Setiembre de 2009 - 2ª parte

- No se podrá utilizar ningún tipo de material (apuntes, libro, calculadora, etc). Apague su teléfono celular.
- **Sólo** se contestarán preguntas sobre interpretación de la letra hasta 30 minutos antes de la finalización del mismo.
- Escriba las hojas de un solo lado
- Las partes no legibles del examen se considerarán no escritas
- En la primer hoja a entregar ponga con letra clara, en el ángulo superior derecho, salón en el cual desarrolló la prueba, su nombre, número de cédula de identidad y cantidad de hojas -en ese orden-; las demás hojas es suficiente con nombre, número de cédula y número de página.

Problema 1	10 (4,0.5,5.5) ptos	

Una cooperativa de transporte colectivo está realizando un análisis de la utilidad de las líneas de ómnibus que tiene en funcionamiento. Para realizar parte de este estudio, interesa conocer las distancias entre las paradas que cubre una línea determinada.
Dada una línea con *n+1* paradas, se tiene un vector *v* de tamaño *n*, el cual contiene en la i-ésima posición la distancia desde la parada *i* a la parada *i+1*. En la figura se muestra un ejemplo de una línea con 11 paradas y la correspondiente distancia entre las paradas, expresada en kilómetros. El vector *v* para la línea del ejemplo sería:




```
v = [0.31 0.2 0.25 0.3 0.21 0.5 0.4 0.44 0.41 0.1]
```

Si dos paradas consecutivas están muy cercanas, el servicio se ve perjudicado, ya que el ómnibus tendría que parar muy seguido sin que esto sea de utilidad para los viajantes.
a) Escriba una función llamada `paradas_cercanas` que, dado un vector v de distancias entre paradas y un cierto criterio de cercanía `d_cerca`, devuelva un vector con los números de paradas cuya siguiente parada está a menos de `d_cerca` kilómetros.

```
paradas_cercanas(v,0.25) = [2,5,10]
paradas_cercanas(v,0.075) = []
```

Si dos paradas consecutivas están muy separadas, es posible que haya viajantes perjudicados, pues ambas paradas lo podrían dejar lejos de su destino.
b) Escriba el **cabezal** (no es necesario implementar la función) de una función llamada `paradas_lejanas` que, dado un vector *v* de distancias entre paradas y un cierto criterio de lejanía `d_lejos`, devuelva un vector con todas las paradas cuya siguiente parada está a `d_lejos` o más kilómetros de distancia.

Se tiene una matriz *M* de dimensiones *mxn*, la cual contiene la información de distancias entre paradas para *m* líneas de *n+1* paradas cada línea.
c) Escriba una función `buscar_linea_con_defectos` que reciba esta matriz *M* como parámetro junto con dos criterios de distancia `d_cerca` y `d_lejos`, y devuelva el número de fila de la matriz *M* que contiene la primera línea de ómnibus con al menos una parada muy cercana a su siguiente, o bien, al menos una parada muy lejana a su siguiente. Si no se encuentra ninguna línea con estas condiciones, la función debe devolver cero.

Nota: Para la resolución de este ejercicio ***NO*** se permite el uso de ninguna función de Matlab que por su naturaleza, resuelva trivialmente el problema.
Para la parte c) puede ser conveniente utilizar las funciones implementadas en las partes a y b.
La función de la parte b puede ser utilizada sin implementarla !!!

Solución Problema 1:

a)

```
function p = paradas_cercanas(v, d_cerca)
    p = [];
    for i = 1:length(v)
        if v(i) < d_cerca
            p = [p i];
        end
    end
```

b)

```
function p = paradas_lejanas(v, d_lejos)
```

c)

```
function r = buscar_linea_con_defectos(M, d_cerca, d_lejos)
    [m n] = size(M);
    i = 1;
    r = 0;
    while i <= m & r == 0
        paradas = paradas_cercanas(M(i,:), d_cerca);
        if length(paradas) > 0
            r = i;
        else
            paradas = paradas_lejanas(M(i,:), d_lejos);
            if length(paradas) > 0
                r = i;
            end
        end
        i= i+1;
    end
```

Problema 2	10 (6, 4) ptos	

a) Para calcular la distancia entre un origen y un destino (por ejemplo, dos paradas de ómnibus) se utiliza una matriz cuadrada de distancia. En la primera columna de esta matriz se guardan los orígenes y en la primera fila los destinos. El resto de las celdas contienen las distancias entre orígenes y destinos (ver ejemplo). Escriba una función **iterativa** `darDistancia` en *Matlab* que tome como entrada la matriz de distancia, un valor de origen y un valor de destino y devuelva la distancia entre las dos ubicaciones. Si no se encuentra una de las dos ubicaciones en la matriz, devuelve -1.

Por ejemplo, si la matriz *M* es:

0	3024	2358	2688	3201
3024	0	-1	-1	-1
2358	0.31	0	-1	0.4
2688	0.45	-1	0	0.8
3201	0.22	0.33	0.55	0

```
darDistancia(M,2358,3024) = 0.31
darDistancia(M,2688,2358) = -1
darDistancia(M,1111,2358) = -1
darDistancia(M,1111,4444) = -1
```

b) Implementar una función iterativa en Matlab "`darDistanciaCam`" que dada una matriz de distancia y un vector que representa un recorrido calcule la distancia asociada a dicho recorrido.

```
darDistanciaCam(M,[])= 0          darDistanciaCam(M,[3201,3024]) = 0.22
darDistanciaCam(M,[3024])= 0      darDistanciaCam(M,[3201,2358,3201])=0.33+0.4
darDistanciaCam(M,[3024,3024])=0  darDistanciaCam(M,[3201,3024,3201])=-1
```

Notas: En este ejercicio ***NO*** se permite utilizar `find` ni ninguna función de *Matlab* que, por su naturaleza, resuelva trivialmente el problema.

Solución Problema 2:

a)
```
function d = darDistancia(M, origen, destino)
[filas cols] = size(M);
% Valor de retorno si no encuentro nada
d = -1;
% Busco la fila del origen
fila = 0;
i = 2;
while i <= filas && fila == 0
      if M(i,1) == origen
            fila = i;
      end
      i = i + 1;
end
if fila > 0
      % Busco la columna del destino
      i = 2;
      columna = 0;
      while i <= cols && columna == 0
            if M(1,i) == destino
                  columna = i;
            end
            i = i + 1;
      end
      if columna > 0
            d = M(fila, columna);
      end
end
```

b)
```
function d_camino = darDistanciaCam(M, v)
d_camino = 0;
n = length(v);
i = 1;
while i < n && d_camino >= 0
      d = darDistancia(M, v(i), v(i+1));
      if d >= 0
            d_camino = d_camino + d;
      else
            d_camino = -1;
      end
      i = i + 1;
end
```

Problema 3	10 (6, 4) ptos	

a) Escriba una función **iterativa** "`pertenece`" en *Matlab* que tome como entrada un vector "*vect*" y un valor entero "*elem*" y retorne un vector de largo 3 de la siguiente forma:

$$[cant, primerIndice, ultimoIndice]$$

donde *cant* es la cantidad de ocurrencias de "*elem*" en "*vect*", *primerIndice* es la posición del primer elemento "*elem*" en "*vect*", y *ultimoIndice* es la posición del último elemento "*elem*" en "*vect*". Si "elem" no se encuentra en "*vect*", el resultado a devolver es *[0, 0, 0]*.

b) Escriba una función **iterativa** "`contarElementos`" en *Matlab* que tome como entrada un vector, y retorne una matriz de *N* filas y 4 columnas de la siguiente forma:

$$\begin{bmatrix} elem1 & cant1 & primerIndice1 & ultimoIndice1 \\ & & \vdots & \\ elemN & cantN & primerIndiceN & ultimoIndiceN \end{bmatrix}$$

donde *elemX* es un elemento del vector "*vect*", *cantX* es la cantidad de veces que ocurre el elemento *elemX* en el vector "*vect*", *primerIndice* es la posición del primer elemento *elemX* en el vector "*vect*", y *ultimoIndiceX* es la posición del último elemento *elemX* en el vector "*vect*" (para *X* de 1 a *N*).

Notas: Para la resolución de este ejercicio no se permite el uso de funciones *Matlab* que permitan resolver de manera trivial el problema. Puede asumir que el vector "*vect*" tiene por lo menos un elemento en ambas partes.

Solución Problema 3:

a)
```
function v = pertenece(vect, elem)
v = [0 0 0];
n = length(vect);
for i= 1:n
     if vect(i) == elem
          v(1) = v(1) + 1;
          if v(2) == 0
               v(2) = i;
          end
          v(3) = i;
     end
end
```

b)
```
function M = contarElementos(vect)
M = [];
n = length(vect);
for i= 1:n
     v = pertenece(vect, vect(i));
     M = [M; vect(i) v];
end
```