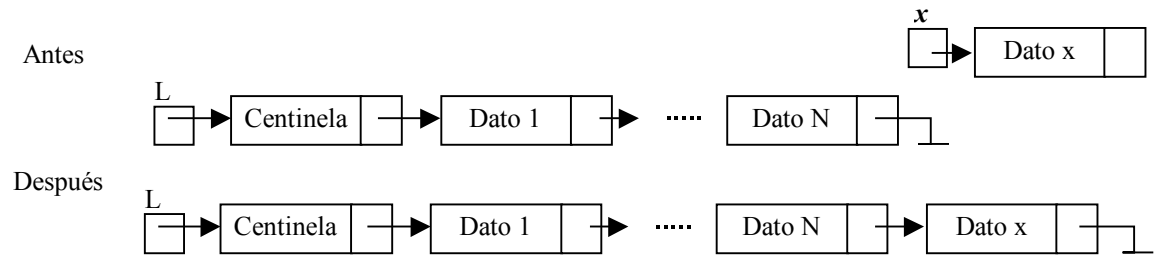
Tiempo: 100 Minutos (8:05 a 9:45), todas las preguntas tienen igual puntaje.

1. Se desea implementar la inserción de un nodo al final de una lista simplemente enlazada con centinela. Escriba en C la función ***insertar*** que ponga un nodo ***x*** al final de la lista ***L*** según se ilustra en la figura adjunta. Cada nodo de la lista tiene estructura:

```
typedef struct nodo_lista {
     struct nodo_lista * next;
     int dato;
} NODO_LISTA;
```




```
void insertar(NODO_LISTA * L, NODO_LISTA *x) {     7/7
/* Pre-condición x != NULL */
 while (L->next != NULL)
      L = L->next;                                  } 7/7
   L->next = x;                                     }
   x->next = NULL;                                    6/6
}
```

2. Se desea replicar (o clonar) un árbol de búsqueda binaria. La réplica contiene la misma información y con igual estructura. Escriba en C la función ***replicar*** que toma como argumento un puntero a la raíz de un árbol y retorna la dirección de la raíz de la réplica generada.
Suponga que cada nodo del árbol tiene la siguiente estructura:

```
typedef struct nodo_arbol {
    struct nodo_arbol * p;     /* puntero al padre */
    struct nodo_arbol * left;  /* hijo izquierdo */
    struct nodo_arbol * right; /* hijo derecho */
    ELEMENTO elemento;
} NODO_ARBOL;
```

```
NODO_ARBOL * replicar (NODO_ARBOL * T) {
  return replicarNodos(T, NULL);
}
NODO_ARBOL * replicarNodos (NODO_ARBOL * T, NODO_ARBOL * p) {
   NODO_ARBOL * R;
   if (T == NULL) return NULL;
   R = (NODO_ARBOL *) malloc(sizeof(NODO_ARBOL));
   R->elemento = T->elemento;
   R->p = p;
   R->left = replicarNodos(T->left, R);
   R->right = replicarNodos(T->right, R);
```

```
    return R;
}

Otra solución tomada de sus respuestas (por Eduardo González)

NODO_ARBOL * replicar (NODO_ARBOL * T) {
    NODO_ARBOL * padre;
    if (T == NULL) return NULL;
    padre = (NODO_ARBOL *) malloc(sizeof(NODO_ARBOL));
    padre->elemento = T->elemento;
    padre->p = NULL;
    padre->left = replicar(T->left);
    padre->right = replicar(T->right);
   if (padre -> left != NULL) /* luego con subárboles terminados */
        padre->left->p = padre; /* vinculo sub-árbol con su padre */
  if (padre -> right != NULL)  /* idem lado derecho */
        padre->right->p = padre;
    return padre;
}
```

Buena definición de prototipo de función. *7/7*
Buena implementación recursiva o no de la solución *8/8*
Liga al padre bien hecha *5/5*

3. En la implementación del algoritmo de Kruskal se ocupan las operaciones MakeSet(***v***), FindSet(***v***), y Union(***u,v***), donde ***v*** y ***u*** son vértices. Una implementación particular en el contexto de Kruskal es utilizar un arreglo de enteros. Se asume que los nodos están numerados de 0 a n-1. Cada vértice se asocia al índice ***v*** de modo que el contenido A[***v***] es el representante se su conjunto.
   En su fase de iniciación Kruskal invoca MakeSet para cada vértice, con lo cual el arreglo inicial queda:

vértice / Indice	0	1	2	3	.....	n-1
Representante	0	1	2	3	.....	n-1

a) Con esta estructura de datos, implemente FindSet(A,***v***) donde A es el arreglo descrito y ***v*** un vértice.
b) Con esta estructura de datos, implemente Union(A, n, ***u,v***) donde A es el arreglo descrito, n su tamaño y ***u, v*** son dos vértices.

a) *8/8*

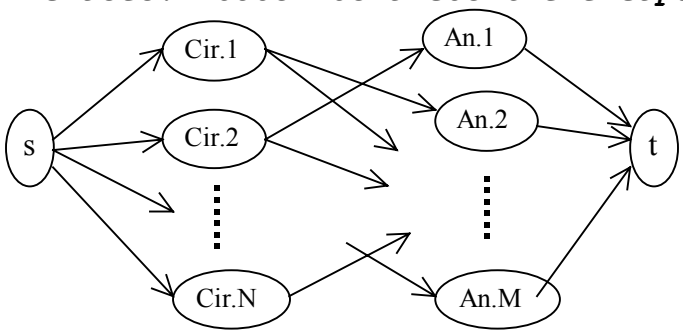
```
int FindSet(int A[], int v) {
    return A[v];
}
```

b) *12/12*

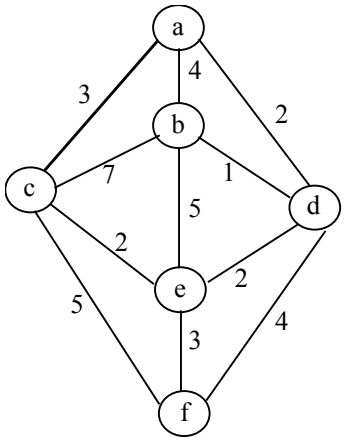
```
void Union( int A[], int n, int u, int v) {
    int i, representante = A[v];

    for (i=0; i < n; i++)
        if (A[i] == representante)
            A[i] = A[u];
}
```

4. ~~Se~~ En una misión multinacional se desea formar equipos compuestos de un cirujano y un anestesista que hablen el mismo idioma. Entre los médicos hay chilenos, franceses, rusos, estadounidenses, y alemanes, por nombrar algunos. Varios médicos hablan más de una lengua. A usted le piden que organice la mayor cantidad de equipos cirujano-anestesista que hablen un mismo idioma. Indique cómo resolvería el problema.

El problema puede ser visto como uno de asociación bipartita máxima. Por un lado ponemos a los cirujanos y por otro a los anestesistas. Establecemos un arco de capacidad 1 entre dos nodos siempre que ambos tengan algún idioma común. Luego se agregan nodos auxiliares fuente y destino con arcos de capacidad uno desde la fuente a cada cirujano y arcos de cada anestesista al destino. Luego se resuelve con Ford-Fulkerson visto en clases. Todos los arcos tiene capacidad uno.



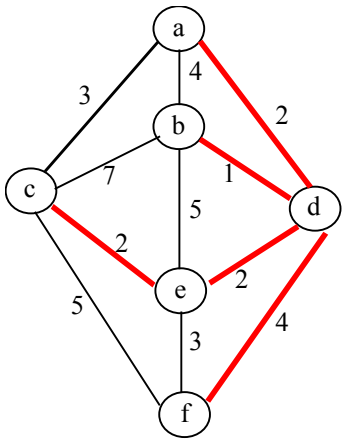
5. Suponiendo que los arcos son procesados en **orden alfabético** de sus vértices:



a) Mostrar valor inicial del arreglo d[v] (distancia) y p[v] (padre), y luego sus valores para cada iteración sobre los arcos en el algoritmo Bellman-Ford aplicado con origen ***b***.
b) Mostrar el árbol de expansión mínima obtenido por Prim empezando en ***b***. Muestre el valor final del arreglo p[v] (padre) y key[v] (peso de conexión).

a)
Si ponemos los arcos en orden alfabético, tenemos (si lo vemos como grafo dirigido, ponemos arcos de en un sentido y en otro por cada arco nodirigido):
(a,b), (a,c), (a,d), (b,c), (b,d), (b,e), (c,e), (c,f), (d,e), (d,f), (e,f)

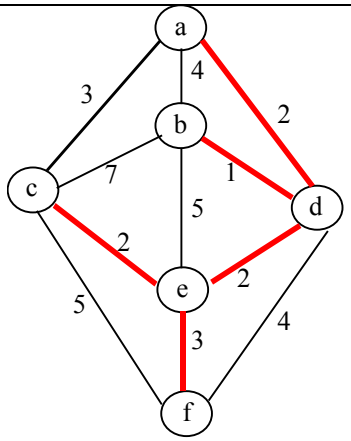
```
            a     b     c     d     e     f
d0 = [    ∞,    0,    ∞,    ∞,    ∞,    ∞ ]
p0 = [null, null, null, null, null, null]  2/2

d1 = [    4,    0,    7,    1,    3,    5 ]
p1 = [    b, null,    a,    b,    d,    d ]

d2 = [    3,    0,    5,    1,    3,    5 ]
p2 = [    d, null,    e,    b,    d,    d ]

d3 = [    3,    0,    5,    1,    3,    5 ]
p3 = [    d, null,    e,    b,    d,    d ]  8/8
```



Las restantes configuraciones se repiten hasta completar d_5 y p_5, y el algoritmo termina.

b)

```
              a     b     c     d     e     f
key0 = [    ∞,    0,    ∞,    ∞,    ∞,    ∞ ]
p0   = [null, null, null, null, null, null]
El mínimo es b.

key1 = [    4,    0,    7,    1,    5,    ∞ ]
p1   = [    b, null,    b,    b,    b, null]
El mínimo es d.

key2 = [    2,    0,    7,    1,    2,    4 ]
p2   = [    d, null,    b,    b,    d,    d ]
El mínimo es a (o e).

key3 = [    2,    0,    3,    1,    2,    4 ]
p3   = [    d, null,    a,    b,    d,    d ]
El mínimo es e.

key4 = [    2,    0,    2,    1,    2,    3 ]
p4   = [    d, null,    e,    b,    d,    e ]
El mínimo es c.

key4 = [    2,    0,    2,    1,    2,    3 ]
p4   = [    d, null,    e,    b,    d,    e ]  6/6
El mínimo es f, y no hay más cambios.
```



4/4