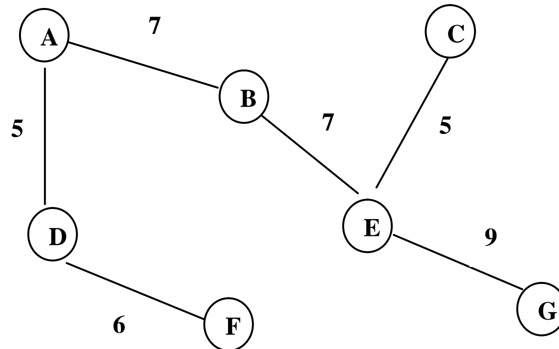
# Solución Examen de Programación 3 y III (14/12/2010)

Instituto de Computación, Facultad de Ingeniería, UdelaR

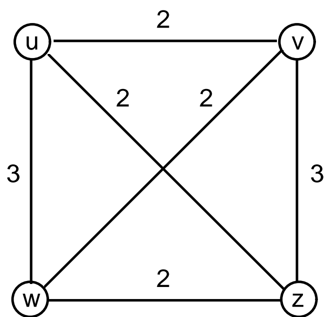
## Parte Teórica Obligatoria

### Ejercicio 1 (10 puntos)

1.1) El árbol de cubrimiento de costo mínimo, resultado de aplicar el algoritmo de Prim, partiendo del vértice A es:



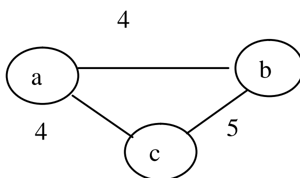

1.2) El enunciado es falso.



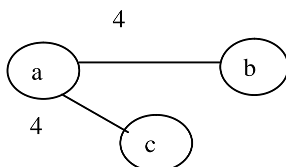

El algoritmo de Dijkstra, para cualquier vértice devuelve un árbol de cubrimiento de costo 7, que está formado por las aristas incidentes al vértice. Sin embargo el árbol de cubrimiento de costo mínimo es 6.

### Ejercicio 2 (10 puntos)

2.1) El enunciado es falso.

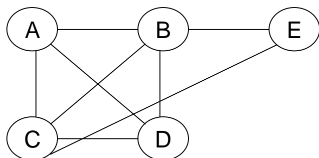



El único árbol de cubrimiento de costo mínimo es:

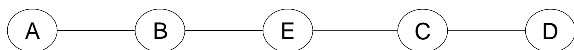



El camino de menor costo entre los vértices *b* y *c* es la arista de costo 5 y no esta incluida en el árbol de cubrimiento de costo mínimo.

2.2) El enunciado es falso.




Los vértices A, B, C y D forman un clique. En DFS (A) se puede obtener el siguiente árbol:




Por lo tanto, no necesariamente todos los vértices del clique aparecen consecutivos.

## Ejercicio 3 (10 puntos)

En el primer bloque lo que se hace es comparar los elementos de la posición *i-1* con el de la siguiente posición *i*, en caso de que el de la posición *i-1* sea menor que el de la posición *i*, lo que se hace es cambiar el valor de la posición *i-1* por la suma de todos los siguientes valores.

Por otro lado, en el segundo bloque se puede observar un algoritmo de ordenación (que dado un arreglo lo ordena de forma ascendente).

Las operaciones básicas se pueden identificar en las líneas:

***Primer bloque:***

- *a[i-1] < a[i]* (comparación)
- *a[i-1] = 0* (asignación)
- *a[i-1] = a[i-1] + 2*a[j]* (asignación)

***Segundo bloque:***

- *a[i] >= a[j]* (comparación)
- *a[i] = a[j]* (asignación)
- *a[j] = temp* (asignación)

**a)**

a1) **Peor caso**

Considerando el primer bloque, el peor caso se da cuando el ***`if`*** siempre evalúa en ***`true`***. Esto corresponde al caso de una secuencia ordenada en forma ascendente (estricto).
Para el segundo bloque el peor caso también se da cuando el ***`if`*** siempre evalúa en ***`true`***. Esto corresponde a una secuencia ordenada en forma descendente (no necesariamente estricto).

***Considerando todo el algoritmo***, se debe observar que al ejecutar el peor caso del primer bloque, el arreglo termina siendo modificado de forma que queda ordenado en forma descendente estricto. La entrada del segundo bloque será su peor caso. Se concluye que el peor caso del algoritmo es que el arreglo de entrada venga ordenado en forma ascendente estricto.

a2) **Mejor caso**

Considerando sólo el primer bloque, su mejor caso se da cuando el ***`if`*** siempre evalúa en ***`false`***. Esto corresponde al caso de una secuencia ordenada en forma descendente (no necesariamente estricto). Después de ejecutar el primer bloque para este caso, el arreglo queda sin cambios. Notar que la entrada del segundo bloque será uno de sus peores casos.
Para el segundo bloque (aislado) el mejor caso se daría cuando el ***`if`*** siempre evalúa en ***`false`***. Esto corresponde a una secuencia ordenada en forma ascendente estricta. Esta entrada no es posible que se dé como salida del primer bloque en ningún caso.

***Considerando todo el algoritmo***, se tomó como solución válida aquellas ***entradas que son el mejor caso del primer bloque*** y por lo tanto el peor caso del segundo.

**NOTA**: Este no es el mejor caso para **todo** el algoritmo (considerando la ejecución secuencial de ambos bloques). Sin embargo se aceptó dicha simplificación para la corrección del examen y por esa razón se publica la solución que toma en cuenta este razonamiento.

b) **Costo del mejor caso:**

***Primero se va a calcular el costo del segundo bloque:***
*(El costo del segundo bloque es siempre el mismo ya que luego del primer bloque el arreglo llega ordenado en forma descendente)*

$$T^{(2)}(n) = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} c_1 + 2c_2$$

$$T^{(2)}(n) = (c_1 + 2c_2) \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} 1$$

$$T^{(2)}(n) = (c_1 + 2c_2) \sum_{i=0}^{n-1} (n - i - 1)$$

$$T^{(2)}(n) = (c_1 + 2c_2) \sum_{i=0}^{n-1} n - (c_1 + 2c_2) \sum_{i=0}^{n-1} i - (c_1 + 2c_2) \sum_{i=0}^{n-1} 1$$

$$T^{(2)}(n) = (c_1 + 2c_2)n^2 - (c_1 + 2c_2)\frac{(n-1)n}{2} - (c_1 + 2c_2)n$$

como $\dfrac{(n-1)n}{2} = \dfrac{n^2 - n}{2}$,

$$T^{(2)}(n) = (c_1 + 2c_2)n^2 - \frac{(c_1 + 2c_2)}{2}n^2 + \frac{(c_1 + 2c_2)}{2}n \; - (c_1 + 2c_2)n$$

$$T^{(2)}(n) = \frac{(c_1 + 2c_2)}{2}n^2 - \frac{(c_1 + 2c_2)}{2}n$$

Sustituyendo por los valores de las constantes:

$$T^{(2)}(n) = 5n^2 - 5n$$

***Mejor caso primer bloque:*** si el arreglo a se encuentra ordenado de forma descendente no estricto, la comparación *a[i-1] < a[i]* siempre va a ser falsa. Resultado:

$$T_B^{(1)}(n) = (n - 1)c_1$$

Sustituyendo por los valores de las constantes:

$$T_B^{(1)}(n) = 2n - 2$$

Uniendo los cálculos de los dos bloques se concluye que:

***Mejor caso:***

$$T_B(n) = 5n^2 - 5n + 2n - 2 = 5n^2 - 3n - 2$$

c)

***Peor caso:***

$$T_W(n) = \sum_{i=1}^{n-1}(c_1 + c_2 + \sum_{j=i}^{n-1} c_2) + \sum_{i=0}^{n-1}\sum_{j=i+1}^{n-1} c_1 + 2c_2$$

El cálculo no se pedía.

## Ejercicio 4 (10 puntos)

1) Hallar la cota

La definición de costo en el caso medio es: $T_A(n) = \sum_{E \in D'_n} p(E)T(E)$ donde E∈ D'$_n$ y p(E) es la probabilidad que tiene E de ser entrada del algoritmo.

Se debe transformar dicha expresión de forma que resulte útil para el cálculo, teniendo en cuenta los datos del ejercicio.

1. Toda entrada del algoritmo hace que éste ejecute de una manera particular, esto en términos del árbol de decisión se ve como que dicha entrada termina en un determinado nodo externo.
2. La ejecución del algoritmo y por lo tanto el nodo externo que le corresponde en el algoritmo es el mismo para toda entrada que tenga el mismo orden relativo entre sus elementos. Pueden agruparse entonces las entradas en dominios $D'_{1n}$, $D'_{2n}$,…$D'_{qn}$ que representan conjuntos de secuencias con el mismo orden relativo. Supongamos que la cantidad de subconjuntos es ***q***, puede determinarse rápidamente que al ser secuencias de n elementos sin repetición, entonces ***q=n!***
3. Como todas las secuencias son equiprobables, la probabilidad de que una secuencia de un determinado subconjunto sea entrada es: **p = 1/n!**
4. Como secuencias de subconjuntos distintos deben terminar en distintos nodos externos se tiene que **m = n!**
5. También se cumplirá que la probabilidad de caer en determinado nodo externo es también 1/n! o sea **p(i) =1/n!**
6. Por definición de AD, el costo de la ejecución de un algoritmo que termina en el nodo externo i es $\mathbf{k_i}$

Incorporando todos estos datos a la fórmula general del costo se tendrá:

$$T_A(n) = \sum_{i=nodoexterno} p(i) * k_i \quad \textbf{entonces} \quad T_A(n) = \sum_{i=nodoexterno} \frac{1}{n!} * k_i$$

$$T_A(n) = \frac{1}{n!} \sum_{i=nodoexterno} k_i$$

Aplicando el lema 2 del teórico y dato de este ejercicio, como m= n!:

$$\sum_{i=nodoexterno} k_i \geq m \log m \quad \textbf{entonces} \quad \sum_{i=nodoexterno} k_i \geq n! \log n!$$

Retomando T:

$$T_A(n) \geq \frac{1}{n!} * n! \log n! \text{ por lo tanto } T_A(n) \geq \log n!$$

Usando el dato:

$$T_A(n) \geq n \log n - 1.5n$$

Como la expresión de la derecha acota inferiormente el costo en el caso medio de un algoritmo cualquiera, esa puede ser considerada la cota buscada:

$$F_A(n) = n \log n - 1.5n$$

en este punto no se puede afirmar que ésta sea la mejor cota inferior, pero sí que el problema de sorting en su caso medio tiene un costo **Ω(n log n),** eventualmente podría ser mayor si no se demuestra otra cosa.

2) Complejidad de sorting en caso medio

Supongamos que $F_A$ no es la mejor cota, llamemos $F^B_A$ (n) a la posible mejor y representará la complejidad del sorting en el caso medio.

En el Teórico se determinó que el Algoritmo Mergesort en su caso medio es **O(n log n),** y por ser Mergesort uno de los algoritmos considerados en la primera parte, cumple también que es **Ω(n log n).**

o sea: $T^{MS}_A(n) \in O(n \log n)$ y $T^{MS}_A(n) \in \Omega(n \log n)$

Al igual que $F_A$, $F^B_A$ es una cota inferior y deberá ser menor (a partir de cierto n) que el costo de cualquier algoritmo, en particular Mergesort, por lo tanto:

Si $F^B_A$ es mejor que $F_A$, entonces $\mathbf{F^B_A} \in$ **Ω(n log n)** y

Si $F^B_A(n) \leq T^{MS}_A(n)$, entonces $\mathbf{F^B_A} \in$ **O(n log n)** y

$\mathbf{F^B_A}(n) \in$ **Θ(n log n)**

Entonces el problema de sorting en su caso medio tiene una complejidad **Θ(n log n).**

# Problemas

## Problema 1 (30 puntos)

**Parte a)**

No. Si se definen los siguientes vectores k[6, 5, 5] y d[8, 5, 5], y G = 10, entonces se tiene como solución 8, ya que se carga el animal i = 0, y no se cargan los animales de peso 5. Esta solución no es la óptima global ya que se puede cargar los animales i = 1e i = 2, obteniendo un ganancia de 10.

**Parte b)**

No. Se define el valor por unidad de peso del animal *i* como el cociente $c_i = [d_i / k_i]$. La idea del algoritmo ávido consiste en ir examinando los objetos en orden decreciente de valor por unidad de peso, es decir, aprovechando al máximo el valor por cada unidad de peso que ingresa al barco.

Para los vectores: k[6, 5, 5] y d[8, 5, 5] se tiene el vector C de valores por unidad de peso es: C = [4/3, 1, 1].

Si G = 10 entonces se tiene como solución 8, ya que se carga el animal i = 0, y no se cargan los animales de peso 5. Esta solución no es la óptima global ya que se puede cargar los animales i = 1 e i = 2, obteniendo un ganancia de 10 y respetando la restricción del peso.

**Parte c)**

$f(i, g)$ indica la mayor cantidad de denarios que puede obtener el comerciante por la venta de los animales que puede transportar en su barco, considerando hasta *i* tipos de animales y soportando *g* kilos sin hundirse.

A continuación se describen dos alternativas para resolver el problema, cabe mencionar que pueden existir otras alternativas para resolver el problema. Para la corrección se tomó como válidas las soluciones que consideraban hasta G kilos al igual que aquellas soluciones que consideraban G kilos exactamente a cargar en el barco.

**Alternativa 1:**

Pasos base:

$$f(i, g) = \begin{cases} -\infty & si \quad i < 0 \\ 0 & si \quad i \geq 0, g = 0 \end{cases}$$

Pasos recursivos:

$$f(i, g) = \begin{cases} f(i-1, g) & si \quad 0 < g < k[i] \quad y \quad 0 \leq i < n \\ \max\{f(i-1, g), f(i-1, g-k[i]) + d[i]\} & si \quad g \geq k[i] > 0 \quad y \quad 0 \leq i < n \end{cases}$$

donde k[i] indica el peso del animal i-ésimo.

El problema se resuelve con $f(n-1, G)$, donde G es el peso máximo en kilos que puede soportar el barco sin hundirse.

**Alternativa 2:**

La función $f(i, g)$ tiene solución trivial cuando $i = n$, donde *n* es el total de animales. La función retorna cero porque se evaluaron todos los animales del comerciante.

En general, se tienen dos opciones:

- incluir el animal *i* en el barco: $f(i+1, g-k[i]) + d[i]$, donde k[i] es el peso en kilos del animal *i*, y *d*[i] la cantidad de denarios que cuesta el animal *i* siempre y cuando $g \geq k[i]$.
- no incluir el animal *i* en el barco: $f(i+1, g)$

La función recursiva que resuelve el problema es:

$$f(i,g)=\begin{cases}0 \quad si \quad i=n \\ f(i+1,g) \quad si \quad g<k[i] \quad y \quad 0\le i<n \\ \max\{f(i+1,g), f(i+1,g-k[i])+d[i]\} \quad si \quad g\ge k[i] \quad y \quad 0\le i<n\end{cases}$$

El problema se resuelve con $f(0,G)$, donde G es el peso máximo en kilos que puede soportar el barco sin hundirse.

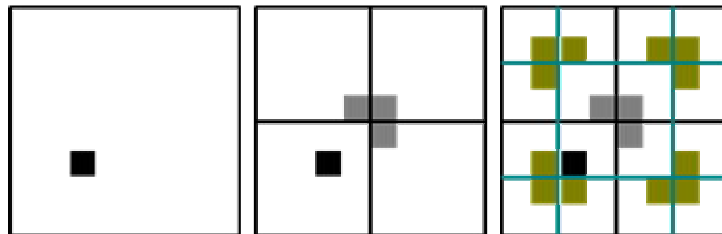
**Parte d)**

En el paso recursivo se debe almacenar si el animal de tipo *i* es cargado o no al barco. Para esto es suficiente mantener un vector de tamaño *n* donde los índices del vector representan los *i* tipos de animales. Si el animal es cargado al barco se guardará el valor 1 en la celda del vector correspondiente para dicho animal. El vector se inicializa con todos los elementos en 0.

# Problema 2 (30 puntos)

1) $(n^2-1) / 3$

2) Siguiendo D&C la solución al problema consta de subdividir el problema en 4 de (N/2xN/2), siendo N potencia de 2, cada subproblema también tendrá dimensión potencia de 2. En cada división se ubica una baldosa de la siguiente manera, considerando dejar la parte "interna" de la baldosa hacia la posición ya cubierta. (Primero se ubica la gris, luego las mostaza, etc.)



Se subdivide el problema invocando para cada cuadrante: su posición inicial (relativa a la división), dimensión (N/2) y posición cubierta (monumento o baldosa ubicada). Al llegar al caso base en que se tiene un cuadrado de 2x2 con una celda ya cubierta, ya sea por el monumento u otra baldosa previamente puesta, solo hay una manera de ubicar la baldosa en ese caso.

```
void cubrirPlaza(Posicion p, int dim, Posicion cubierta){
	// Problema base en area de 2x2
	Posicion p1, p2, p3,p4;
	if (dim==2){
		p1= p;
		p2= Posicion(p.fila, p.col+1)
		p3= Posicion(p.fila+1, p.col);
		p4= Posicion(p.fila+1, p.col+1);
		if (estaCubierta(p)){ //m2 superior izq
			ubicarBaldosa(p2,p3,p4)
		} else if (estaCubierta (p2){ // m2 superior derecho
			ubicarBaldosa(p1,p3,p4)
		} else if (estaCubierta (p3){// m2 inferior izquierdo
			ubicarBaldosa(p1,p2,p4)
		} else { // m2 inferior derecho
			ubicarBaldosa(p1, p2, p3);
		}
	}
	else { // Division del problema NxN en subproblemas N/2 x N/2 (cuadrantes
de la matriz)
		int mitad = dim/2;
		/* se ubica una baldosa de manera de cubrir un m2
		 * de cada cuadrante con excepción del que ya tiene 1 cubierto
```

```
            */
           int cuadrante=0
           if (cubierta.fila < p.fila + mitad && cubierto.col < p.col+mitad){
                 // el cuadrante esta en cuadrante sup izq
                 cuadrante=1;
           } else if (cubierto.fila < p.fila+mitad && cubierta.col >= p.col +
mitad){
                 //el monumento esta en cuad sup der
                 cuadrante=2;
           }  else if (cubierto.fila >= p.fila + mitad && cubierta.col < p.col
+ mitad){
                 //el monumento esta en cuad inf izq
                 cuadrante=3;
           }  else if (cubierto.fila >= p.fila + mitad && cubierta.col >= p.col
+ mitad){
                 //el monumento esta en cuad inf der
                 cuadrante=4;
           }
           p1=Posicion(p.fila + mitad -1, p.col + mitad - 1);
           p2=Posicion(p.fila + mitad - 1, p.col + mitad);
           p3=Posicion(p.fila + mitad, p.col + mitad - 1);
           p4=Posicion(p.fila + mitad, p.col + mitad);

           switch (cuadrante){
                 case 1:
                       p1=cubierta;
                       ubicarBaldosa(p2,p3,p4); break;
                 case 2:
                       p2=cubierta;
                       ubicarBaldosa(p1,p3,p4); break;
                 case 3:
                       p3=cubierta;
                       ubicarBaldosa(p1,p2,p4); break;
                 default:
                       p4=cubierta;
                       ubicarBaldosa(p1,p2,p3); break;
           }
           //cuadrante sup izq
           cubrirPlaza(p, mitad, p1);
           //cuadrante sup der
           cubrirPlaza(Posicion(p.fila,p.col+mitad), mitad, p2);
           //cubrir cuadrante inf izq
           cubrirPlaza(Posicion(p.fila+mitad,p.col), mitad, p3);
           //cubrir cuadrante inf der
           cubrirPlaza(p4, mitad, p4);
      }
}
```