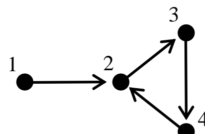
Solución Examen de Programación 3 y III (20/07/2013)

Instituto de Computación, Facultad de Ingeniería, UdelaR

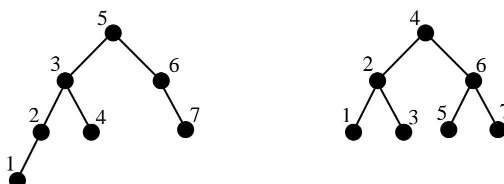
Parte Teórica Obligatoria

Ejercicio 1 (10 puntos)

1. **Falso**. Que uno de sus vértices tenga grado de entrada 0 no implica que el grafo no tenga ciclos. Un contraejemplo es el siguiente grafo:



2. **Falso**. Considerar los siguientes árboles:



   El de la izquierda es un AVL de altura 3 y el de la derecha es un ABB de altura 2 que tiene los mismos elementos.

3. **Falso**. Una secuencia de inserciones peor caso para el ABB es cualquiera que deje el árbol como una lista, por ejemplo, una secuencia ordenada. Una secuencia de inserciones peor caso para la Tabla de Hash es una en la que todos los elementos colisionen, lo que claramente depende de la función de Hash.

4. **Verdadero**. Puede hacerse utilizando variables auxiliares, como el siguiente algoritmo, que es claramente O(*n*):

```
int fib (int n)
{
      int f_i2 = 0, f_i1 = 1, f_i = n;

      for (int i = 2; i <= n; i++)
      {
            f_i  = f_i2 + f_i1;
            f_i2 = f_i1;
            f_i1 = f_i;
      }

      return f_i;
}
```

5. **Verdadero**. Ver teórico.

Ejercicio 2 (10 puntos)

a) El algoritmo es el Insertion Sort visto en clase.
Para analizar el algoritmo se requiere contar únicamente la cantidad de comparaciones entre los elementos de la secuencia, o sea cuántas veces se evalúa la segunda parte de la condición de la segunda iteración (while):
`Primero <= S[j].`

El Peor Caso viene dado cuando se tiene que colocar al principio del arreglo cada elemento a ordenar (o sea que la secuencia viene ordenada en forma descendente); en ese caso se realizarán en cada paso i comparaciones.

$$T(n) = \sum_{i=1}^{i=n-1} (i) = \frac{n.(n+1)}{2} - n = \frac{n^2 - n}{2} = \frac{n(n-1)}{2}$$

b) Como la comparación que se realiza es: Primero <= S[j], si hubiese elementos repetidos, cuando se encuentre un elemento que ya existe en la parte ordenada del arreglo será colocado al principio de los que son iguales a él, por lo tanto el orden de los elementos se invierte.

Si se tiene inicialmente el arreglo

2_1	4	2_2	1	2_3

después de ordenar los elementos "2" quedan juntos pero en orden inverso:

1	2_3	2_2	2_1	4

Ejercicio 3 (10 puntos)

Definimos $D^k(i, j)$ como el costo del camino de menor costo entre los vértices i y j utilizando únicamente los vértices $0..k$ como vértices intermedios.
Este valor se calcula de la siguente manera:

$$\begin{cases} D^k(i,j) = \min(D^{k-1}(i,j), D^{k-1}(i,k) + D^{k-1}(k,j)) \forall k \in 1..n-1 \\ D^0(i,j) = C(i,j) \end{cases}$$

Si k es un vértice intermedio que pertenece al camino de menor costo de i a j entonces los subcaminos de i a k y de k a j deben ser los de menor costo, si no fuera así, el camino de i a j no sería óptimo.

Por lo tanto el principio de optimalidad es aplicable.

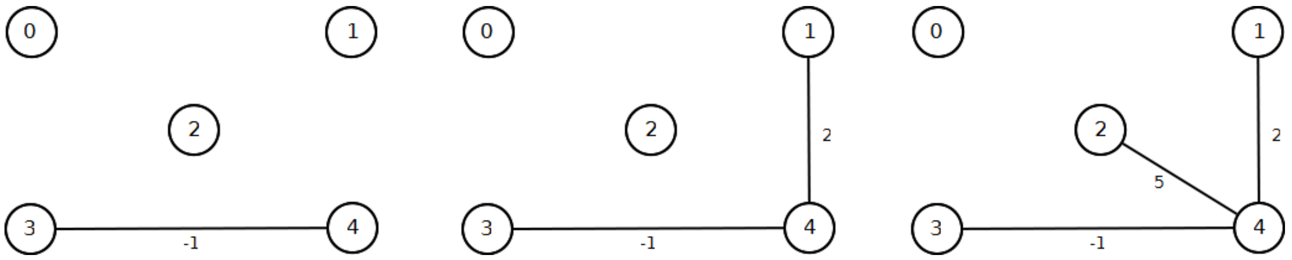
Ejercicio 4 (10 puntos)

a) Sea $G=(V,E)$ un grafo no dirigido conexo, un árbol de cubrimiento es un grafo $G'=(V,E')$, donde E' es un subconjunto de E y tal que G' es un árbol, o sea un grafo conexo y sin ciclos.
Además G' es de costo mínimo si $costo(G') = mín\{costo(G'') \mid G''$ es un árbol de cubrimiento de $G\}$, siendo $costo(G) = \sum_{e \in E} costo(e)$

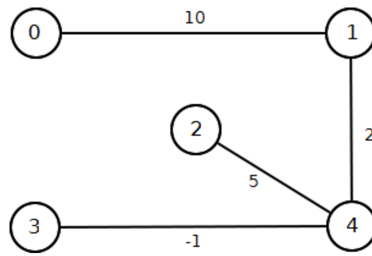
b) El algoritmo de Kruskal

c) Se ordenan las aristas por costos y se eligen en el siguiente orden hasta obtener un árbol de cubrimiento de costo mínimo.

Se agregan las aristas 3-4, 1-4 y 2-4



Se descartan las aristas 1-2 y 2-3, ya que generan un ciclo.
Luego se agrega la arista 1-0 y se obtiene un árbol de cubrimiento de costo mínimo.



Problemas

Problema 1 (30 puntos)

a)

$$f(i,j) = \begin{cases} i & si\ j = 0 \\ j & si\ i = 0 \\ \min\{f(i-1,j-1), f(i-1,j)+1, f(i,j-1)+1\} & si\ v_1[i] = v_2[j] \\ \min\{f(i-1,j-1), f(i-1,j), f(i,j-1)\}+1 & o.c \end{cases}$$

$f(i,j)$ representa el valor del costo de la alineación óptima, considerando los primeros i cubos de la secuencia v_1 y los primeros j cubos de la secuencia v_2.
El costo óptimo del problema original se obtiene invocando $f(n_1,n_2)$ donde n_i denota el largo de la secuencia de cubos v_i.
Los pasos bases denotan los casos en que la cantidad de cubos de una de las secuencias es 0 (es decir, ya fueron considerados todos los cubos de la secuencia). En esos casos, la cantidad de posiciones distintas corresponde a la cantidad de cubos restantes de la otra secuencia.
El primer caso recursivo corresponde a cuando se tienen cubos del mismo color en las posiciones consideradas. En dicho caso mantener la alineación no constituye necesariamente el paso óptimo, pues la inserción de comodines puede disminuir costos posteriores. Por lo tanto se deben evaluar a su vez las alternativas de inserción de comodines, lo cual se contempla en los dos últimos casos, según la secuencia donde se inserte el comodín.
El caso recursivo final corresponde a cuando se tienen cubos de distinto color en las posiciones consideradas. En dicho caso se deben explorar todas las alternativas:

1) Alinear los cubos tal cual como se encuentran, aunque no sean iguales
2) Agregar un espacio en la secuencia v_2
3) Agregar un espacio en la secuencia v_1

(Notar que agregar un espacio en ambas secuencias no mejora el costo de ninguna forma y degeneraría en una solución no recursiva).

b)

```
//secuencias de cubos (v1, v2) y largos asociados (n1, n2 respectivamente)
int alineacionOptima(cubo * v1, cubo * v2, int n1, int n2) {

	int i,j;

	int** f = new int*[n1 + 1];
	for (int i=0; i<=n1; i++) {
		f[i] = new int[n2 + 1];
	}

	//paso base 1
	for (i=0; i<=n1; i++) {
		f[i][0] = i;
	}

	//paso base 2
	for (j=0; j<=n2; j++) {
		f[0][j] = j;
	}

	for (i=1; i<=n1; i++) {
		for (j=1; j<=n2; j++) {

			if (v1[i] == v2[j]) {
				//paso recursivo 1
				f[i][j] = min(f[i-1][j-1], f[i-1][j] + 1, f[i][j-1]+1);
			}
			else {
```

```
                        //paso recursivo 2
                        f[i][j] = min(f[i-1][j-1], f[i-1][j], f[i][j-1]) + 1;
                    }
                }
            }

    int costoOptimo = f[n1][n2];

    for (int i=0; i<n1; i++) {
        delete [] f[i];
    }

    delete [] f;

    return costoOptimo;
}
```

Problema 2 (30 puntos)

Forma de la solución:
Tupla de largo fijo N de la forma <x_0, x_1, …, x_{N-1}> donde x_i indica si el i-ésimo video estará presente en el DVD (con un uno) o no (con un cero).

Restricciones explícitas:
Todos los elementos de la tupla pertenecen al conjunto {0, 1}

$$x_i \in \{0,1\} \quad \forall i,\ 0 \leq i \leq N-1$$

Restricciones implícitas:
La duración del DVD debe ser como máximo 60 minutos:

$$\sum_{i=0}^{n-1} Xi * duracion(i) \leq 3600$$

Debe haber exactamente 20 videos

$$\sum_{i=0}^{n-1} Xi = 20$$

Función objetivo:
Se debe obtener la mejor valoración, la cual se logra con los valores más bajos

$$\min\left(\sum_{i=0}^{n-1} Xi * valoracion(i)\right)$$

Predicado de Poda:
Al procesar el j-esimo elemento, si la cantidad de videos seleccionados más todos los que quedan por considerar es menor a 20, se poda:

$$\sum_{i=0}^{j-i} Xi + (N - j - 1) < 20$$

b)

```
Tupla t, sol;
  t.seleccion = new int[N];
  t.valoracion = 0;
  sol.seleccion = new int[N];
  sol.valoracion = MAX_INT;

  int* videos = new int[N]
  for(int i = 0; i < N; i++)
```

```
    videos[i] = 0;

  seleccionVideos(valoracion, duracion, videos, 0, 0, 0, 0, sol);
  //sol es la tupla solucion

void seleccionVideos (int * valoracion, int * duracion,
                       int * enConstruccion, int valorAcumulado,
                       int tiempoAcumulado, int cantVideos, int actual,
                       tupla & sol)
// Se cumplen las siguientes precondiciones:
//  valorAcumulado < sol.valoracion
//  tiempoAcumulado <= 3600
//  actual <= N
//  cantVideos <= 20
{
   if (cantVideos == 20) {
      int i;
      for (i = 0; i < actual; i++)
         sol.seleccion [i] = enConstruccion [i];
      for (i = actual; i < N; i++)
         sol.seleccion [i] = 0;
      sol.valoracion = valorAcumulado;
   } else {
      if (cantVideos + N - actual >= 20) { // predicado de poda
      // y como 20 > cantVideos, entonces actual < N
      // y en las siguientes llamadas se siguen cumpliendo
      // las precondiciones actual <= N y cantVideos <= 20
      // porque se va a pasar como par�tro actual`+ 1 y cantVideos + 1.

         // Las llamadas tal vez modifiquen enConstrucion desde actual hasta N-
1.
         // Pero no importa porque enConstruccion y actual se manejan como un
         // array con tope.

         // No se agrega el elemento actual.
         // Se podr  ser mas restrictivo y pedir que
         // actual < N - 1 (se sabe que se cumple actual < N)
         // porque como cantVideos < 20 y el actual elemento ser  el ltimo
         // y no se va a incluir, entonces no se obtendr  una soluci�
         enConstruccion [actual] = 0;
         seleccionVideos (valoracion, duracion, enConstruccion,
                           valorAcumulado, tiempoAcumulado,
                           cantVideos, actual + 1, sol);

         // Si se cumplen condici�mplicita y funcion objetivo se agrega el
elemento actual
         if ( (duracion [actual] + tiempoAcumulado <= 3600)
          && (valoracion [actual] + valorAcumulado < sol.valoracion)){
            enConstruccion [actual] = 1;
            seleccionVideos (valoracion, duracion, enConstruccion,
                              valoracion [actual] + valorAcumulado,
                             duracion [actual] + tiempoAcumulado,
                             cantVideos + 1, actual + 1, sol);

         }
      }
   }
};
```