Solución Examen de Programación 3 y III (12/02/2010)

Instituto de Computación, Facultad de Ingeniería

Parte Teórica Obligatoria

Ejercicio 1 (10 puntos)

Parte 1)
El mejor caso se da cuando la condición de la línea 6 retorna siempre *falso*, por lo que la asignación de la línea 7 nunca llega a ejecutarse. La asignación de la línea 3 se ejecuta una vez por cada vez que ingresa al *for* de la línea 2.

$$T_B = \sum_{i=0}^{L-1} 1 = L$$

El peor caso es cuando la condición de la línea 6 retorna siempre *verdadero*, por lo que la asignación de la línea 7 se ejecuta siempre. La asignación de la línea 3 se ejecuta una vez por cada vez que ingresa al *for* de la línea 2.

$$T_W = \sum_{i=0}^{L-1}\left(1 + \sum_{j=i+1}^{L-1} 1\right) = \sum_{i=0}^{L-1} 1 + \sum_{i=0}^{L-1}\sum_{j=i+1}^{L-1} 1 = L + \sum_{i=0}^{L-1}(L-1-i) = L + \sum_{i=0}^{L-1}(L-1) - \sum_{i=0}^{L-1} i$$

$$T_W = L + L(L-1) - \frac{L(L-1)}{2} = \frac{L^2+L}{2}$$

Para el caso promedio se tiene en cuenta que la asignación de la línea 3 se ejecuta con probabilidad 1, mientras que la asignación de la línea 7 depende de la condición de la línea 6 que devuelve *true* con probabilidad ½.

$$T_A = \sum_{i=0}^{L-1} 1 + \sum_{i=0}^{L-1}\sum_{j=i+1}^{L-1} 1 * \frac{1}{2} = L + \frac{1}{2}\sum_{i=0}^{L-1}(L-1-i) = L + \frac{1}{2}\sum_{i=0}^{L-1}(L-1) - \frac{1}{2}\sum_{i=0}^{L-1} i = L + \frac{1}{2}L(L-1) - \frac{1}{2}\frac{L(L-1)}{2}$$

$$T_A = \frac{L^2+3L}{4}$$

Parte 2)

a. $\forall k \in R^+, \forall j \in R^+, k > j\left(n^k \in O(n^j)\right)$

$\forall k \in R^+, \forall j \in R^+, k > j, \left(n^k \in O(n^j)\right)$
Si $k > j \Rightarrow k - j > 0$

Aplicando la def de orden

$$\left(\exists c_0 \in R^+\right)\left(\exists n_0 \in N\right)\left(\forall n \in N\right)\left(n \geq n_0 \rightarrow n^k \leq c_0 n^j\right)$$

Entonces, $\frac{n^k}{n^j} \leq c_0$ sii $n^{k-j} \leq c_0$ (1)

$$\lim_{n \to +\infty} n^{k-j} \to +\infty \Leftrightarrow \forall c \in R, \exists m \in R, n \geq m, n^{k-j} \geq c$$

Como se cumple para todo c, en particular se debe cumplir para c = c_0 +1 entonces:

$\exists m \in R, n \geq m, n^{k-j} \geq c_0 + 1 > c_0$ (2)

De las afirmaciones (1) y (2) se llega a un absurdo, luego se demuestra que es **FALSO**.

b. Si $f(n) \in O(n)$ y $g(n) \in \Theta(f(n))$ entonces $f(n) \times g(n) \in O(n^2)$

Si $f(n) \in O(n)$ entonces $(\exists c_0 \in R^+)(\exists n_0 \in N)(\forall n \in N)(n \geq n_0 \to f(n) \leq c_0 n)$

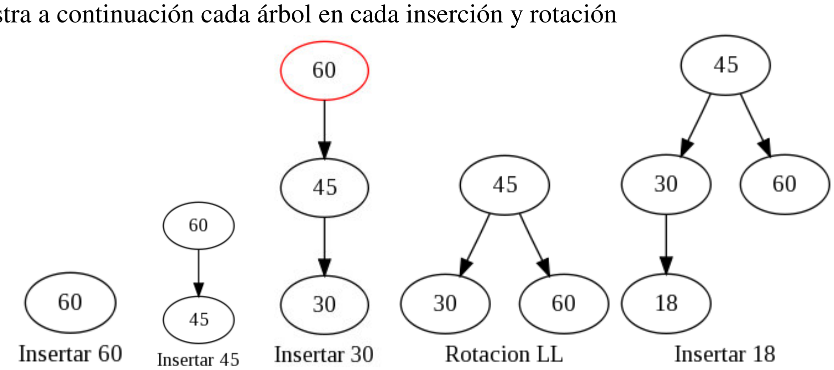
Si $g(n) \in \Theta(f(n))$ entonces $g(n) \in O(f(n))$ sii
$(\exists c_1 \in R^+)(\exists n_1 \in N)(\forall n \in N)(n \geq n_1 \to g(n) \leq c_1 f(n))$

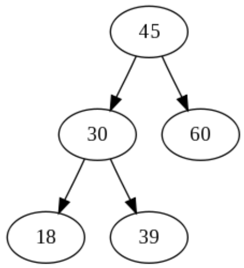
Multiplicando las relaciones…
$(f(n)g(n) \leq c_1 c_0 n f(n)) \leq c_1 c_0 n(c_0 n) = c_1 c_0^2 n^2$

Tomando $c_1 c_0^2$ como nuestra nueva constante (siendo mayor que cero), y aplicando la definición de orden $(f(n)g(n) \in O(n^2))$. Entonces es **VERDADERO**.

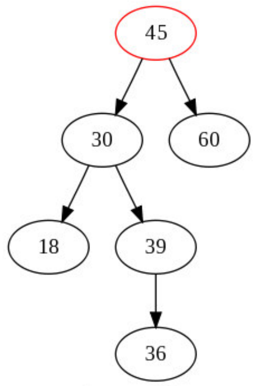
Ejercicio 2 (10 puntos)

1. Ver teórico.
2. Se muestra a continuación cada árbol en cada inserción y rotación

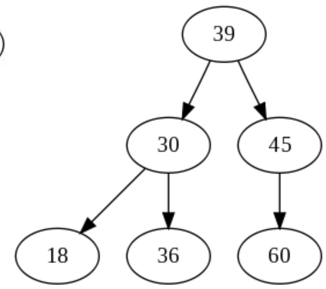




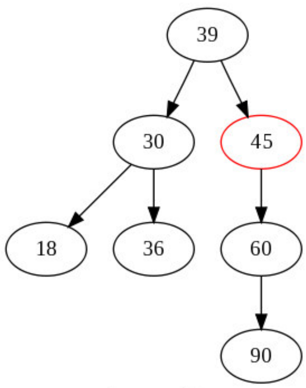
Insertar 39



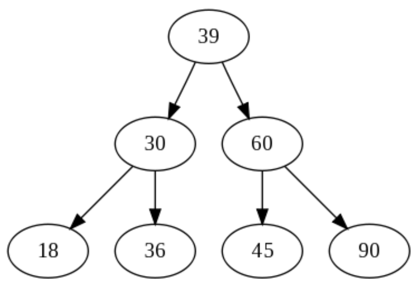
Insertar 36



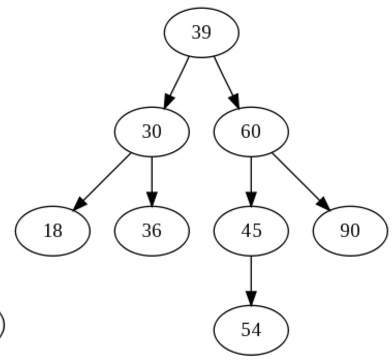
Rotacion LR



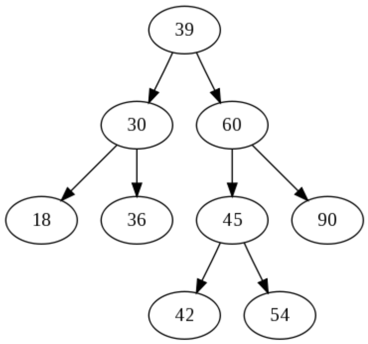
Insertar 90



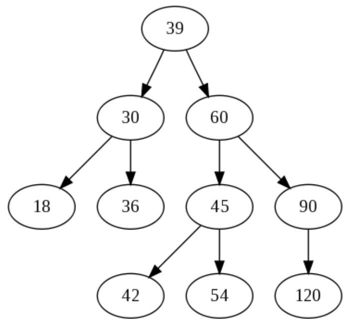
Rotacion RR



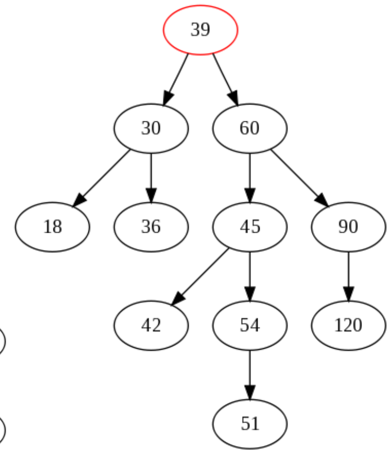
Insertar 54



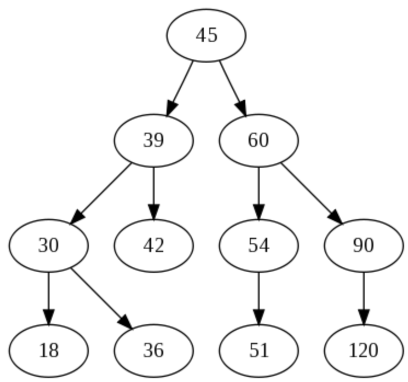
Insetar 42



Insetar 120



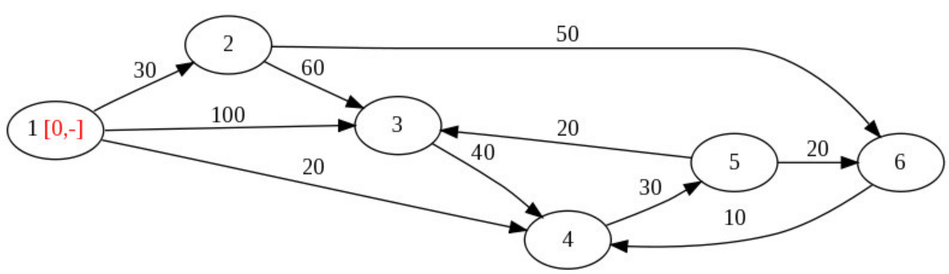
Insetar 51



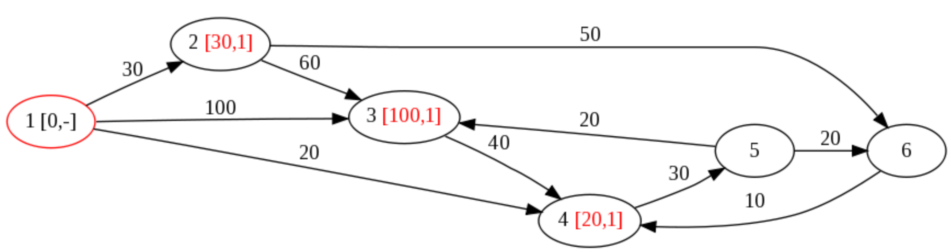
Rotacion RL

Ejercicio 3 (10 puntos)

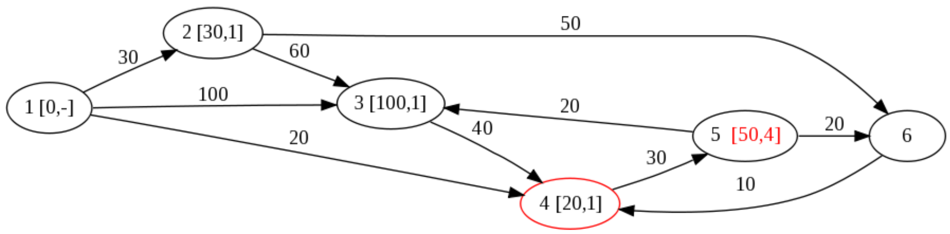
1. Ver teórico.
2. A continuación se muestra el grafo, paso a paso. En cada nodo se tiene la etiqueta *[a,b]*donde *a* indica el costo actual, y *b* el nodo por el cual se llega al mismo.



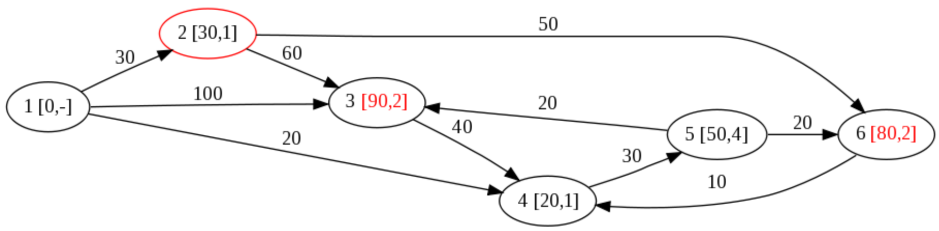
Inicio



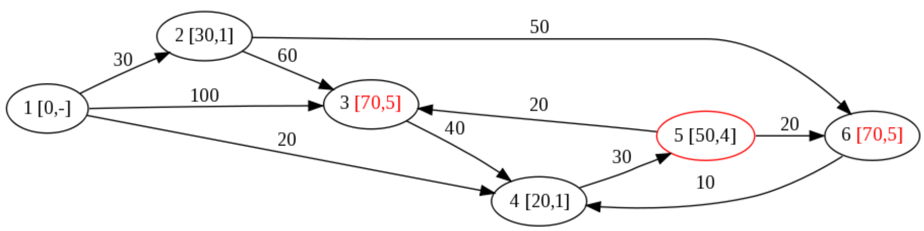
Iteración 1



Iteración 2



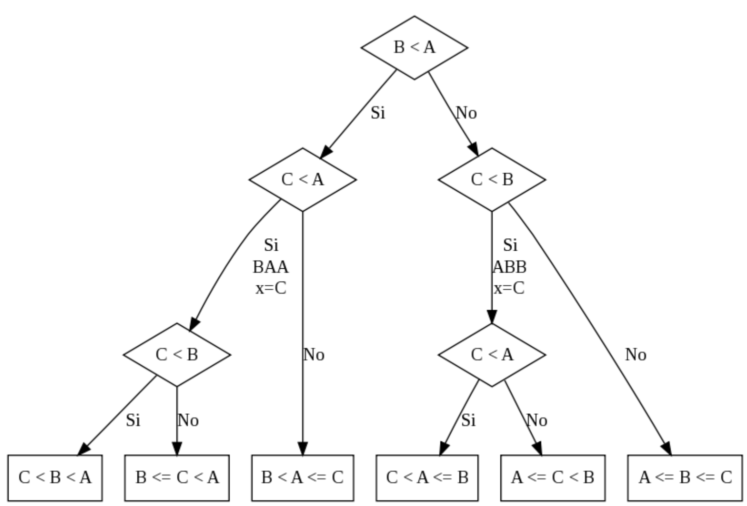
Iteración 3



Iteración 4

Ejercicio 4 (10 puntos)

1. Ver teórico.
2. Árbol de Decisión:



Problemas

Problema 1. (30 puntos)

SOLUCIÓN

Descripción

Se recorre el grafo G según DFS. Las aristas que no son árbol se categorizan según las propiedades de las relaciones de los tiempos de pre y post-procesamiento según casos:

a) si el vértice adyacente que la define no tiene marca de postprocesamiento entonces es arista de regreso a un vértice ancestro.

b) si el vértice adyacente que la define tiene marca de postprocesamiento, se debe determinar si hay relación de ancestría (caso de que la marca de preprocesamiento de vértice en cuestión menor que la marca de preprocesamiento del vértice adyacente) o pertenece a otra rama del árbol (caso de que la marca de preprocesamiento del vértice en cuestión es mayor que la marca de preprocesamiento del vértice adyacente). Además, se utilizan dos vectores adicionales para almacenar dichos tiempos.

Implementación

```
// Clasificación de aristas según recorrido DFS en grafo dirigido G
#define TAM ...
typedef int Grafo[TAM][TAM];

void dfs_clasifica(Grafo G, int v, int* marcados, int &time, int* pre, int*
post, Grafo M) {
	marcados[v] = 1;
	pre[v] = time;
	time++;
	for (int i = 0; i < TAM; i++) {
		if (G[v][i] == 1) {
			if (marcados[i] == 0) {
				M[v][i] = 1; // Arista arbol
				dfs_clasifica(G, i, marcados, time, pre, post, M);
			} else {
				if (post[i] == 0)
					M[v][i] = 2; // Arista regreso
				else {
					if (pre[v] < pre[i])
						M[v][i] = 3; // Arista directa
					else
						M[v][i] = 4; // Arista cruzada
				}
			}
		}
	}
	post[v] = time;
	time++;
}

void main() {
	int marcados[TAM] = {0};   // Indicador de marcados
	int time = 0;// Marca secuencial de tiempo
	int pre[TAM] = {0};  // Tiempos de preprocesamiento
	int post[TAM] = {0};       // Tiempos de postprocesamiento
	Grafo G = ...;       // Grafo dado
	Grafo M = {0};  // Resultado

	// Genera clasificacion de aristas
	for (int i = 0; i < TAM; i++)
```

```
            if (marcados[i] == 0)
                dfs_clasifica(G, i, marcados, time, pre, post, M);
}
```

Problema 2. (30 puntos)

Formalización

Forma de la solución

Tupla de largo fijo M donde M es la cantidad de semillas disponibles, de la forma <x0, x1,…, x M-1>.

Restricciones explícitas

$x_i \in \{0,1\} \forall i, 0 \le i \le n-1$.

Todos los elementos de la tupla deben pertenecer al conjunto {0,1}, si se cultiva un sector con la i-ésima semilla $x_i = 1$, sino $x_i = 0$.

Restricciones implícitas

El costo total de todos los cultivos no debe superar el presupuesto P:

$$\sum_{i=0}^{i=M-1} c_i x_i \le P$$.

La cantidad de horas totales de trabajo no debe superar H:

$$\sum_{i=0}^{i=M-1} h_i x_i \le H$$.

La cantidad de semillas sembradas no debe superar la cantidad de sectores K:

Función objetivo

Definimos la función auxiliar 'Ganancia (t)' como la suma de las ganancias mensuales generadas por las semillas incluidas en la tupla solución t=<x0, x1,…, x M-1>:

$$Ganancia(t) = \sum_{i=0}^{i=M-1} (v_i - c_i) x_i$$

La función objetivo es:

$f = máx_{t \in T} (Ganancia(t))$, donde T = { t = <x0, x1, …, xM-1> / t es factible }.

Implementación

```
Solucion* maxGanancia(int M, int K, int P, int H, int* costoSemilla, int* horasSemilla,
        int* precioVentaSemilla)
{
        Solucion* mejor = crearSolucion(M);
        Solucion* actual = crearSolucion(M);
        r_maxGanancia(M, K, P, H, costoSemilla, horasSemilla, precioVentaSemilla, mejor, actual);
        destruirSolucion(actual);
        return mejor;
}

r_maxGanancia(int M, int K, int P, int H, int* costoSemilla, int* horasSemilla,
        int* precioVentaSemilla, Solucion* mejor, Solucion* actual)
{
        if(actual->ganancia > mejor->ganancia)
        {
                destruirSolucion(mejor);
                mejor = copiarSolucion(actual);
        }//if

        for(int i = 0; i < M; i++)
        {
                if(actual->tupla[i] == 0 &&//si no plante ya la semilla i
                   actual->costo + costoSemilla[i] <= P &&//si no supero el presupuesto
                   actual->horas + horasSemilla[i] <= H &&//si no supero las horas
                   actual->cultivos + 1 < K)//si ya no plante todos los sectores
                {
                        //agrego el cultivo i a la solución
                        actual->costo += costoSemilla[i];
                        actual->horas += horasSemilla[i];
                        actual->cultivos++;
                        actual->ganancia += (precioVentaSemilla[i] - costoSemilla[i]);
                        actual->tupla[i] = 1;

                        r_maxGanancia(M, K, P, H, costoSemilla, horasSemilla, precioVentaSemilla,
                                mejor, actual);

                        //remuevo el cultivo i de la solución
                        actual->costo -= costoSemilla[i];
                        actual->horas -= horasSemilla[i];
                        actual->cultivos--;
                        actual->ganancia -= (precioVentaSemilla[i] - costoSemilla[i]);
                        actual->tupla[i] = 0;
                }//if
        }//for
}//r_maxGanancia
```