Solución Examen de Programación 3 y III (09/01/2013)

Instituto de Computación, Facultad de Ingeniería, UdelaR

Parte Teórica Obligatoria

Ejercicio 1 (10 puntos)

a)

i)

Es falso. Sea $f(n) = 3n$. Se cumple que $f(n) \in O(n)$ pero sin embargo también se cumple que $\lim_{n\to\infty} \frac{2^n}{2^{3n}} = 0$, por lo cual se concluye que $2^{3n} \notin O(2^n)$.

ii)

Es falso. Sea $f(n) = (\frac{1}{2})n$. Se cumple que $f(n) \in O(n)$ pero sin embargo también se cumple que $\lim_{n\to\infty} \frac{2^{(1/2)n}}{2^n} = 0$, por lo cual se cumple que $2^n \notin O(2^{(1/2)n})$.

Recordando que se cumple la siguiente propiedad:

$f(n) \in O(g(n)) \leftrightarrow g(n) \in \Omega(f(n))$

Se concluye que $2^{(1/2)n} \notin \Omega(2^n)$.

b)

$f \in O(g) \Rightarrow \exists c1 \in R+, \exists n1 \in N, \forall n \in N, n > n1 , f(n) \leq c1*g(n)$

$g \in O(h) \Rightarrow \exists c2 \in R+, \exists n2 \in N, \forall n \in N, n > n2 , g(n) \leq c2*h(n)$

Se debe demostrar que:

$\exists c \in R+, \exists n0 \in N, \forall n \in N, n > n0 , f(n) \leq c*h(n)$

Con los valores vistos de c1, c2, n1 y n2 se sabe que se cumplen

$f(n) \leq c1*g(n) \quad g(n) \leq c2*h(n) \Rightarrow$

$f(n) \leq c1*g(n) \leq c1*c2*h(n)$ a partir de un valor de n

Entonces: $n0 \in N$, $n0 = \max(n1, n2)$ y $c \in R+$, $c=c1*c2$

Ejercicio 2 (10 puntos)

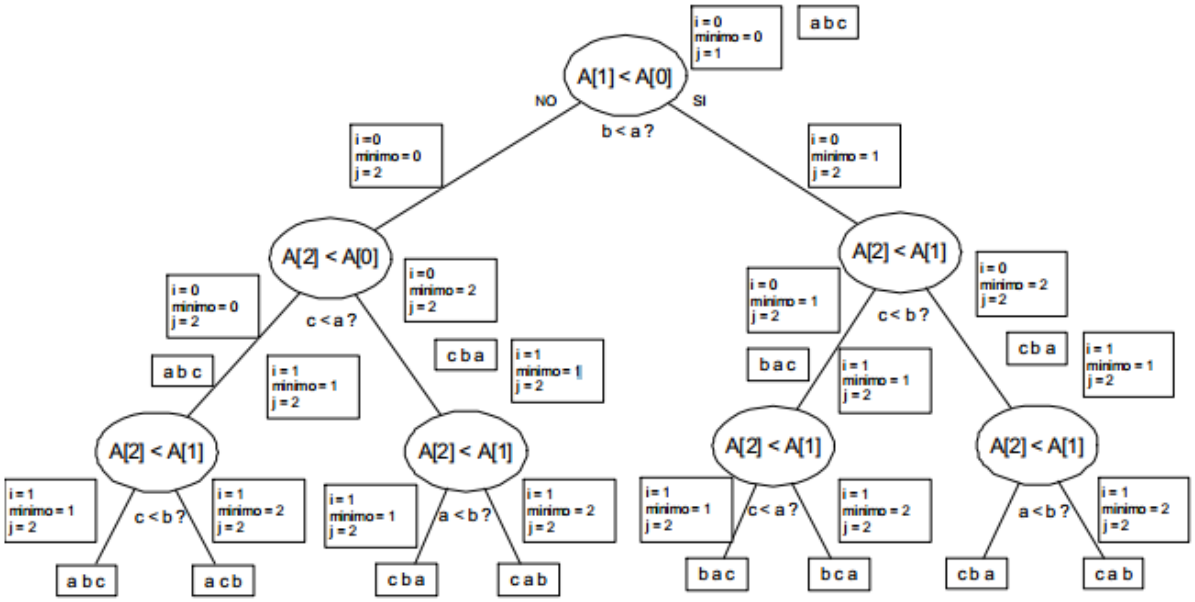
Teórico de Programación Dinámica pág. 5 - 6.

Ejercicio 3 (10 puntos)

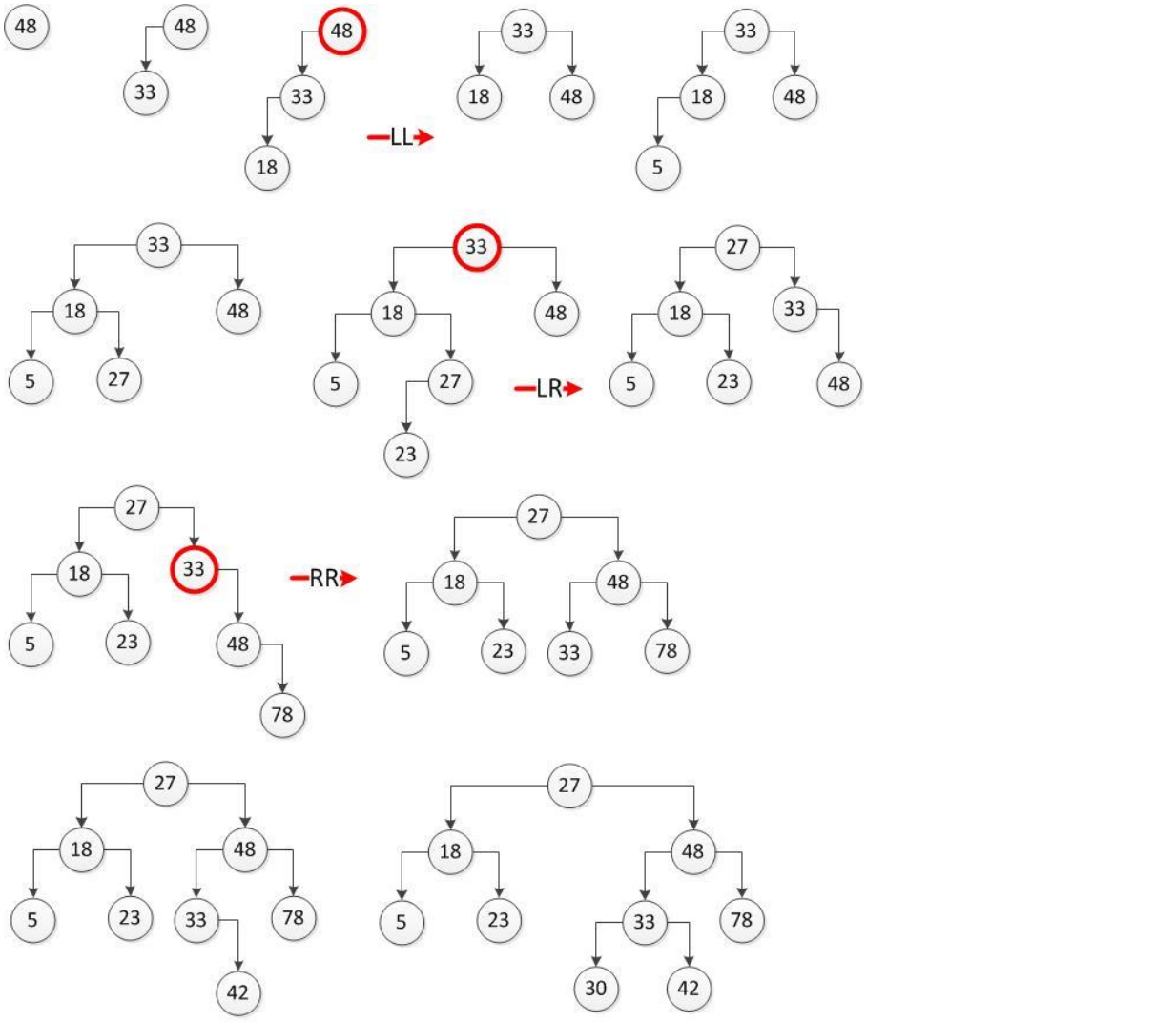
a)

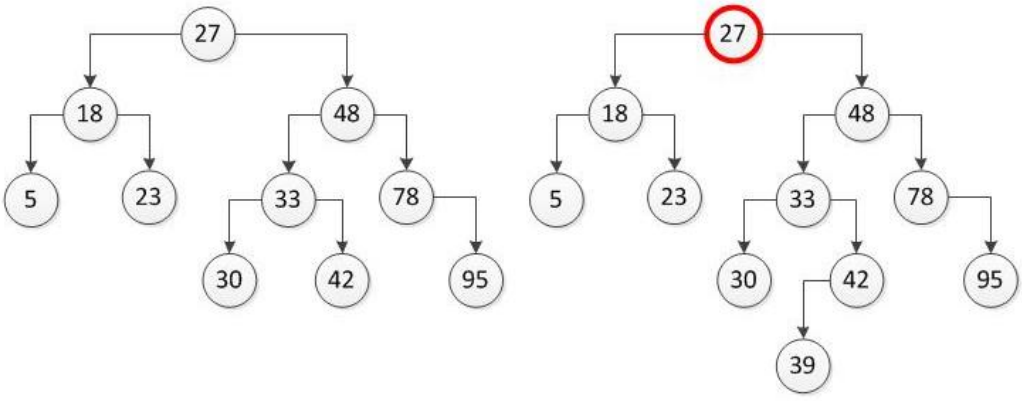
```
para i=0 hasta n-2
   minimo = i;
   para j=i+1 hasta n-1
      si lista[j] < lista[minimo] entonces
         minimo = j
      fin si
   fin para
   intercambiar(lista[i], lista[minimo])
fin para
```

b)

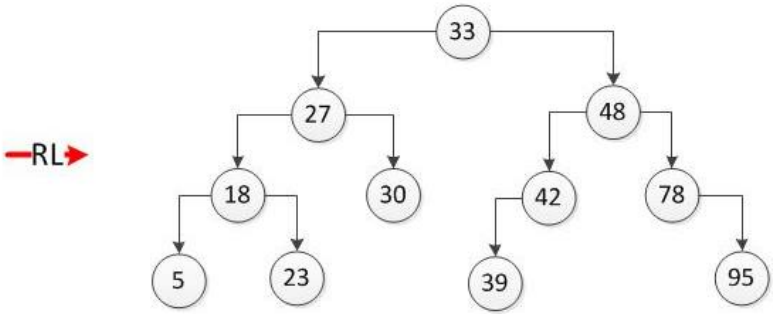


Ejercicio 4 (10 puntos)




27
18
48
5
23
33
78
30
42
95

27
18
48
5
23
33
78
30
42
95
39


RL
33
27
48
18
30
42
78
5
23
39
95

Problemas

Problema 1 (30 puntos)

a) Este problema se puede modelar considerando las secuencias de conversiones como caminos de una moneda de un origen a otro, donde las monedas de distinto origen representan los nodos de un grafo. Existe una arista entre dos nodos del grafo, si existe una conversión directa entre las monedas de distinto origen que representan dichos nodos.

Dado que las conversiones entre monedas de distinto origen son bidireccionales, por cada par de nodos hay dos aristas, cada una en un sentido. El costo de cada arista es 1 ya que se quiere saber la secuencia de conversiones de longitud mínima.

Este problema se resuelve aplicando el algoritmo de Floyd visto en el teórico que permite calcular los caminos mínimos entre cualquier par de nodos de un grafo.

Para poder aplicar el algoritmo de Floyd el grafo no puede contener ciclos de costo negativo. Dado que todas las aristas tienen costo 1, el grafo no contendrá ciclos de costo negativo.

Por lo tanto, $f^k(i,j)$ representa la cantidad de conversiones necesarias (mínimas) para convertir de la moneda de origen *i* a la moneda de origen *j* usando a lo sumo *k* conversiones intermedias. En caso de no existir solución, se devuelve $+\infty$.

Pasos base:

- $f^0(i, j) = 0$ si $i = j$. Este caso representa que se quiere convertir una moneda así misma, por lo tanto, se necesitan 0 conversiones.
  Nota: al momento de la corrección se consideran como válidas las soluciones que incluyen o no éste caso.

- $f^1(i,j) = 1$ *si* $ExisteConversionDirecta(i, j)$ *y* $1 \leq i, j \leq M$. Esto implica que existe una conversión directa entre las monedas de origen *i* y *j*.

- $f^1(i,j) = +\infty$ *si no* $ExisteConversionDirecta(i, j)$ *y* $1 \leq i, j \leq M$. Esto implica que no existe una conversión directa entre las monedas de origen *i* y *j*.

Pasos recursivos:
Se tienen las siguientes situaciones:

- el camino de menor costo entre los vértices *i* y *j* usando los vértices *{1..k}* como vértices intermedios y no pasando por el vértice *k*, en este caso: $f^k(i,j) = f^{k-1}(i,j)$

- el camino de menor costo entre los vértices *i* y *j* usando los vértices *{1..k}* como vértices intermedios y pasando por el vértice *k*, en este caso:
  $f^k(i,j) = f^{k-1}(i,k) + f^{k-1}(k,j)$

Por lo tanto,

$$f^{k}(i,j) = \begin{cases} f^{0}(i,j) = 0 \text{ si } i = j \\ f^{1}(i,j) = 1 \text{ si ExisteConversionDirecta}(i,j) \text{ con } 1 \le i,j \le M \\ f^{1}(i,j) = +\infty \text{ si no ExisteConversionDirecta}(i,j) \text{ con } 1 \le i,j \le M \\ \min\left\{ f^{k-1}(i,j),\ f^{k-1}(i,k) + f^{k-1}(k,j) \text{ con } 1 \le i,j \le M \right\} \end{cases}$$

El problema se resuelve con la invocación $f^{M}(a, b)$ siendo *a* y *b* las monedas de distinto origen a realizar la conversión.

b) Ver apuntes de teórico del algoritmo de Floyd.

Problema 2 (30 puntos)

a)
Forma:
Tupla de largo fijo T = NxM del tipo $<x_0,..x_i,..x_{k-1}>$ que donde xi representa el culito asignado la hectárea i del campo.
R.E:
Los elementos de la tupla deberán ser los identificadores de los cultivos [0..S-1].
$0 \le xi < S$ con $0 \le i < T$;
R.I:
Para todo cultivo i no se podrán plantar más de K[i] hectáreas.
Sea $I(c,x_i)$ la función indicatriz que indica si el cultivo c esta asignado a x_i se cumple que:

Sea:

$$I(c, x_i) = \begin{Bmatrix} 1 \, si \, c = x_i \\ 0 \, otro \, caso \end{Bmatrix}$$

R.I: $\sum_{xi=0}^{xi=T-1} I(c, x_i) \le K[c], \forall c \in [0.S-1]$

F.O:
Minimizar la sumatoria de los costos de los cultivos considerando el impacto de los linderos.
Sea

$$impactoF(x_i) = \{P[i] * impacto()\}$$

FO(t) = min $\sum_{xi=0}^{xi=T-1} impactoF(x_i)$

b)

```
int[N][M] Campo;
int[N][M] Costo;
int[S] P;
int[S] K;

void optimizarProduccion(int **Campo, int** Costo) {
	int costoMin=MAX_INT;
	int[N][M] CampoP;
	int[N][M] CostoP;
	optProduccion(int** CampoP, CostoP, costoMin, 0);
}

void optProduccion(int** CampoP, int** CostoP, int& costoMin, int p){
	int fila = p div N;
	int col = p % M;

	if(p == N*M-1){
		//Solucion completa
```

```
		if (costoActual(CostoP) < costoMin){
			// Mejor Solucion
			for (int f=0;f<=N;f++)
				for (int c=0;c<=M;c++){
					Campo[f][c]+=CampoP[f][c];
					Costo[f][c]=CostoP[f][c];
				}
				costoMin = CostoActual(CostoP);
		}
	}

	for (int c=0;c<S;c++){
		if (K[c] > 0){
			//c es cultivo candidato
			K[c]--;
			C[fila][col] = c;

			if (fila>0) //actualizo impacto de c en adyacente al oeste
				CostoP[fila-1][col]*impacto(CampoP[fila-1][col], c);
			if (col>0){//actualizo impacto de adyacente al norte
				CostoP[fila][col-1]*impacto(CampoP[fila][col-1], c);

				CostoP[fila][col] = P[c]*impacto(CampoP[fila][col-1],
c)*impacto(CampoP[fila-1][col], c);

				// Poda en base a funcion obj.
				if (costoActual(CostoP) < costoMin)
					optimizarProduccion(CampoP,CostoP, p+1);

				// Rollback
				if (fila>0)
					CostoP[fila-1][col]/impacto(CampoP[fila-1][col], c);
				if (col>0)
					CostoP[fila][col-1]/impacto(CampoP[fila][col-1], c);
				K[c]++;
				C[fila][col] = -1;
				CostoP[fila][col] = -1;
			}
		}
	}
}

int costoActual(int** Costo){
	int costo = 0;
	for (int q=0;q<=fila;q++)
		for (int h=0;h<=col;h++)
			costoParcial+=Costo[q][h];
	return costo;
}
```