# Solución del examen de Programación 3 y III

**12 de diciembre de 2015**

## Parte Obligatoria (Ejercicios 1 a 4)

### Ejercicio 1 (10 puntos)

Suponga que tiene una estructura de árbol binario definida de la siguiente forma:

```
1 struct TreeB {
2    int info;
3    struct TreeB* der, izq;
4 }
```

A su vez considere el siguiente algoritmo:

```
1  TreeB* algoritmo (int k, TreeB* A)
2  {
3     TreeB* retorno;
4     if (A == NULL) {
5        retorno = new TreeB;
6        retorno->info = k;
7        retorno->der = NULL;
8        retorno->izq = NULL;
9     else if (k > A->info) {
10       retorno = algoritmo(k, A->der);
11    } else {
12       retorno = algoritmo(k, A->izq);
13    }
14    return retorno;
15 }
```

Teniendo en cuenta que la operación básica es la comparación entre enteros y que $n$ es la cantidad de nodos, responda:

a. Suponiendo que el parámetro de entrada `A` es un ABB y `k` es mayor estricto que todos los elementos del árbol, determine el costo $T(n)$ del algoritmo en el peor caso en función de $n$. Especifique cuándo se da este caso.

b. Suponiendo que el parámetro de entrada `A` **NO** es un ABB y `k` es mayor estricto que todos los elementos del árbol, determine el costo $T(n)$ del algoritmo en el peor caso en función de $n$. Especifique cuándo se da este caso.

c. Considerando también la asignación de enteros como una operación básica, determine si el orden de complejidad de $T(n)$ cambia en alguno de los dos casos. Justifique brevemente.

### Solución

a. El peor caso se da cuando se deba realizar la comparación sobre todos los elementos del ABB. Para que esto suceda, y como `k` es mayor estricto a todos los elementos del ABB, se ejecutará la comparación de la línea 9. Entonces la recurrencia se ejecutará sobre el subárbol derecho siempre. Esto hace que la estructura del ABB deforma en una lista, en la que los hijos de todos los elementos se encuentran a la derecha y está ordenada por ser un ABB. Por lo dicho anteriormente, y como se ejecuta la comparación para todos los elementos, $T(n) = n$.

b. El peor caso coincide con el de la parte anterior con la diferencia que la lista en la que deforma no necesariamente estará ordenada.

c. El orden de complejidad no cambia debido a que solo se hace una asignación. Por lo tanto, como estamos sumando una constante a $n$, el orden de complejidad de $T(n)$ sigue siendo $n$.

## Ejercicio 2 (10 puntos)

a. Defina la variante del problema de la mochila que en el curso se resuelve utilizando Programación Dinámica.

b. Demuestre el principio de optimalidad para dicho problema.

c. Plantee la recurrencia.

d. Muestre mediante un contraejemplo, lo más sencillo posible, que la estrategia Greedy utilizada para resolver de forma óptima la otra variante del mismo problema no siempre resuelve de manera óptima la variante de la parte a.

## Solución

a. Se resuelve el problema de la mochila 0,1 donde no se pueden fraccionar objetos.

Maximizar $\sum_{i=1}^{n} p_i x_i$

sujeto a $\sum_{i=1}^{n} w_i x_i \leq M$

con $x_i \in 0, 1 \quad \forall\, i\, 1 \leq i \leq n$

b. Sea $y_1, ..., y_n$ una solución óptima al problema `Knap(1,n,M)`.

La primer decisión es el valor de $y_1$:

Si $y_1$=0 $\Rightarrow$ $y_2, ..., y_n$ debe ser una solución óptima para el problema `Knap(2,n,M)`, si no lo fuera entonces $y_1, y_2, ..., y_n$ no sería una solución óptima a `Knap(1,n,M)`.

Si $y_1$=1 $\Rightarrow$ $y_2, ..., y_n$ debe ser una solución óptima para el problema `Knap(2,n,M-`$w_1$`)`, si no lo fuera, existe otra solución $z_2, ..., z_n$ mejor, por lo tanto $y_1, z_2, ..., z_n$ tendría más ganancia que $y_1, y_2, ..., y_n$ lo cual es absurdo porque $y_1, y_2, ..., y_n$ es una solución óptima.

c. Sea $g_j(Y)$ el valor de la ganancia de la solución óptima al problema `Knap(j+1, n, Y)`. Entonces la recurrencia es la siguiente:

$g_j(Y) = \text{máx}\{g_{j+1}(Y), g_{j+1}(Y - w_{j+1}) + p_{j+1}\} \quad \forall\, Y, 0 \leq Y \leq M,\ \forall\, j, 0 \leq j < n$

$g_n(Y) = 0 \quad \forall\, Y,\ 0 \leq Y \leq M$

d. **Contraejemplo.**

$n = 2$ y $x_0,\ x_1$ los objetos a considerar,

$M = 10$,

$(p_0, p_1) = (10, 8)$,

$(w_0, w_1) = (15, 8)$,

$\frac{p_0}{w_0} = \frac{10}{15}, \frac{p_1}{w_1} = \frac{8}{8}$,

$\Rightarrow$ el orden en el que se consideran los objetos es: $(x_1, x_0) = (\frac{8}{8}, \frac{10}{15})$

La ganancia obtenida siguiendo esta estrategia es 8, cuando la ganancia máxima es 10.

## Ejercicio 3 (10 puntos)

a. Sea G un grafo sin costos en sus aristas. ¿Qué tipo de recorrida de grafos permite obtener el camino más corto (cantidad de aristas) desde un nodo dado de G a todos los demás? Justifique brevemente y escriba el algoritmo correspondiente a dicha recorrida.

b. Si en cambio G tuviera costos no negativos asociados a sus aristas, ¿el algoritmo anterior resuelve el problema de obtener los caminos de costo mínimo (sumando los costos de las aristas) desde un nodo dado de G a todos los demás? Justifique. En caso negativo indique cuál algoritmo lo resuelve.

## Solución

a. Una recorrida BFS permite obtener el camino más corto desde la raíz (nodo en el que se inicia la recorrida) hacia cada uno de los demás nodos. Esto se debe a que el algoritmo construye un árbol de cubrimiento que considera todos los adyacentes del nodo en el que está parado antes de agregar un nuevo nivel.

Algoritmo:

```
Procedimiento BFS (v: évrtice)
  Var Q: cola de évrtices
  u,w: vertice
  Comienzo
     CrearCola(Q)
     Marcar(v)
     InsBack(Q,v) // encolar
     Mientras No-Vacia(Q)
        u = primero(Q)
        Q = resto(Q)
        Para cada w adyacente a u
           Si w no marcado
              Marcar(w)
              InsBack(Q,w)
           Fin Si
        Fin Para
     Fin Mientras
  Fin
```

b. No, si el grafo tiene costos en sus aristas, una recorrida BFS no resuelve el problema pedido. El algoritmo que sí resuelve este problema es el de Dijkstra.

## Ejercicio 4 (10 puntos)

Dado el algoritmo Selection Sort, cuya implementación es:

```
void selectionSort(int n, int * & A) {
    for (int i = 0; i < n-1; i++) {
        int iMin = i;
        for (int j = i+1; j < n; j++) {
            if (A[j] < A[iMin]) {
                iMin = j;
            }
        }
        int tmp = A[i];
        A[i] = A[iMin];
        A[iMin] = tmp;
    }
}
```

a. Explique cómo se construye el árbol de decisión de este algoritmo.

b. Si se desea utilizar el algoritmo para ordenar tres números enteros, ¿cuál es el árbol de decisión asociado al mismo?

c. ¿Existe alguna hoja del árbol construido en b. que se corresponda con una ejecución del algoritmo que nunca podría darse? Si existe, ¿cuál es y por qué?

## Solución

a. Un árbol de decisión es una forma de representar el funcionamiento de un algoritmo para todos los datos posibles de un tamaño $n$ dado. Se trata de un árbol binario estricto.

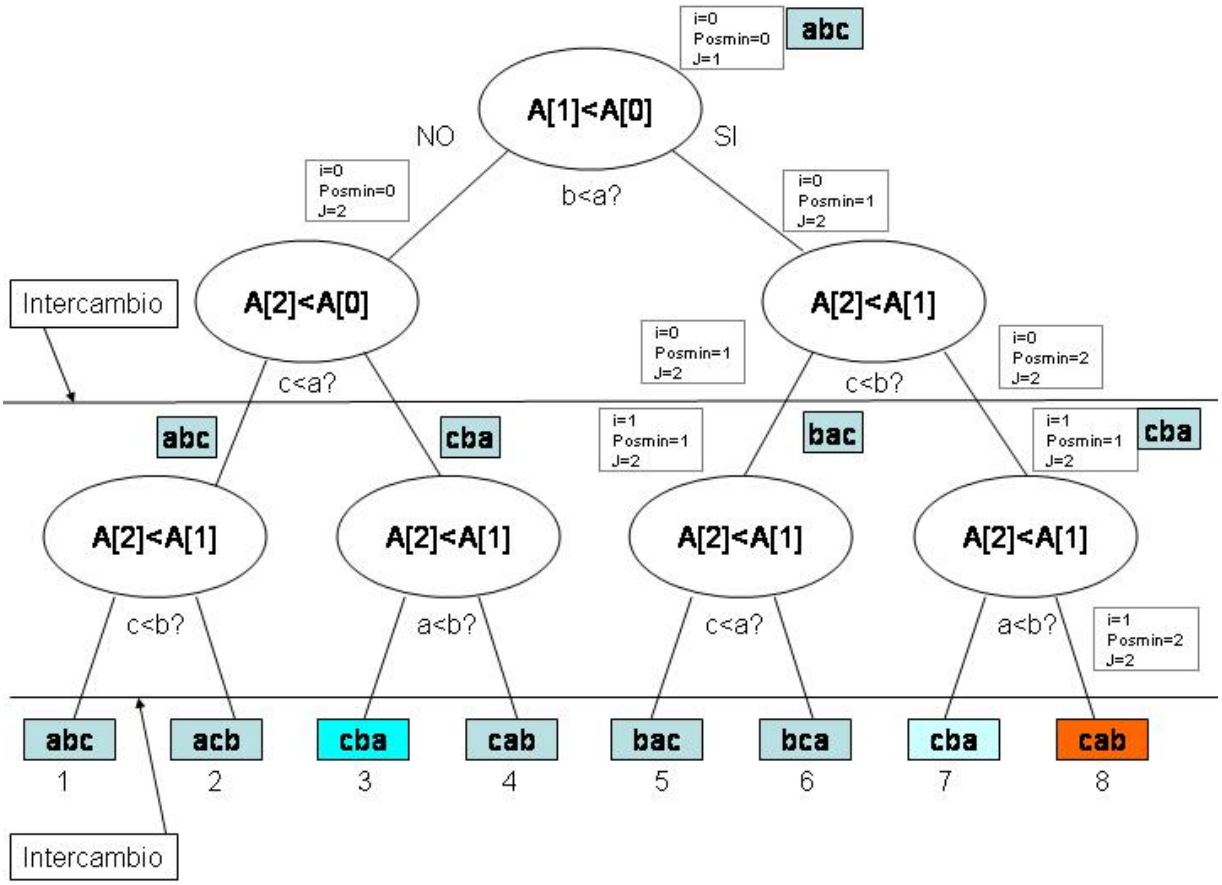
Como cada nodo interno contiene una pregunta que se aplica a los datos, y como la operación básica en este caso es la comparación entre dos elementos, los nodos internos contendrán comparaciones de dos elementos del arreglo.

Los hijos de cada nodo representan las diferentes posibilidades de respuesta, según los datos que se tengan en la entrada. En este caso hay dos posibles respuestas: verdadero o falso. La ejecución del algoritmo sobre los datos reflejará el resultado de la comparación.

Por otro lado, las hojas tendrán las ordenaciones posibles para las distintas entradas.

Finalmente, cada camino desde la raíz a una hoja representa una posible ejecución del algoritmo para una entrada particular.

b. El árbol de decisión es el siguiente:




c. La hoja 8 no puede darse nunca. Como podemos ver en su camino de ejecución en el árbol, en la raíz se hace la pregunta $b < a$ y luego más abajo se pregunta por $a < b$. Sabemos que ambas condiciones no se pueden dar en simutáneo, por eso es que nunca puede darse.

Por otro lado, el nodo 3 no se puede dar si no se admiten repetidos, ya que se tiene que $b < a$ es falso y $a < b$ también. Por ende se cumple que $b \geq a$ y que $a \geq b$, implicando que $a = b$.

# Problemas

## Problema A (30 puntos)

Sea $G = (V, E)$ un grafo simple no dirigido con $|V| = n$. Se dice que una de sus aristas es **puente** si al removerla el grafo resultante tiene mayor cantidad de componentes conexas que el original.

a. Sean $v_1, v_2 \in V$ puntos de articulación. Si existe $(v_1, v_2) \in E$ ¿es una arista puente? Demuestre o dé un contraejemplo.

b. Sea $(v_1, v_2) \in E$ una arista puente. Muestre un ejemplo en el que $v_1$ o $v_2$ NO son puntos de articulación.

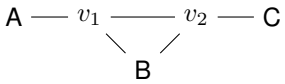
c. ¿Qué cantidad máxima de puentes puede haber en $G$?

d. ¿Qué relación hay entre puentes y ciclos en $G$?

e. Escribir un seudocódigo, basado en DFS, que dados $G$ y una arista $(v_1, v_2)$ permita determinar si dicha arista es puente.

## Solución

a. No tiene por qué serlo. Sean $V = \{A, B, C, v_1, v_2\}$ y $E = \{(v_1, v_2), (A, v_1), (v_1, B), (B, v_2), (v_2, C)\}$.




En este ejemplo $v_1$ y $v_2$ son puntos de articulación y el grafo es conexo. Si se remueve la arista $(v_1, v_2)$ el grafo sigue siendo conexo por lo que la arista no es puente.

b. Si se remueve una arista puente entonces sus vertices incidentes pasan a integrar componentes conexas separadas. Si dichos vértices tienen grado mayor que uno (antes de remover la arista) entonces son puntos de articulación, por que si se los removiera, junto a sus aristas incidentes (el puente incluido), aumentaría la cantidad de componentes conexas del grafo. En el caso de que el grado de los vértices fuera uno (la única arista es el puente) no aumentaría la cantidad de componentes conexas.

Por lo tanto no siempre $v_1$ y $v_2$ son puntos de articulación. El contraejemplo más sencillo es aquel en el que el grafo consiste solamente de los vértices $v_1$ y $v_2$ y la arista $(v_1, v_2)$.




La arista es puente porque al removerla quedan dos componentes conexas pero ninguno de los vertices era punto de articulación.

c. En grafos de $n$ vértices la cantidad máxima de aristas que se puede tener sin formar ciclos es $n-1$ (propiedad de árbol en grafo conexo). Debido al ítem anterior, estas aristas son puentes por definición.

d. Una arista es puente sii no integra ciclos del grafo, por que si los integrara al removerla habría un camino alternativo que conectaría a sus vértices incidentes, por lo que removerla no aumenaría a cantidad de componentes conexas.

e. El seudocódigo busca determinar de forma equivalente si la arista en cuestión integra un ciclo en el grafo. Para lo cual se realiza una recorrida DFS del grafo inicializada en uno de sus vértices incidentes, asumiendo que es puente (no hay ciclo). Si antes de finalizar la recorrida se llega al otro vértice (sin pasar por la arista) entonces existe un ciclo, por lo que la arista no es puente. Si la recorrida se finaliza sin alcanzar el otro vértice entonces no existe ciclo, por lo que la arista es puente. Para simplificar la implementación se recorre sobre el grafo resultante de remover la arista.

```
bool DFSEsAristaPuente(grafo G', vertice v, vertice v2, bool visitado[], bool &espuente) {
  visitado[v] = true;
  Si (espuente) {
    Para cada w adyacente a v en G' {
      Si (!visitado[w]) {
         Si w == v2
           espuente = false;
         Sino
           DFSEsAristaPuente(G', w, v2, visitado, &espuente);
      }
    }
  }
}

bool EsAristaPuente(grafo G, vertice v1, vertice v2) {
  vertice v;
  bool espuente = true;
  gafo G' = removerarista(G,v1,v2);  // La arista (v1,v2) se asume que cierra el ciclo

  Para cada v: visitado[v] = false;
  DFSEsAristaPuente(G', v1, v2, visitado[], espuente);  //  Inicia DFS en v1
  return espuente;
}
```

## Problema B (30 puntos)

Un turista está planificando un viaje en su vehículo. Dispone de un mapa de la zona representado en una tabla $D$. Cada sitio se identifica con un entero entre 1 y $n$. En $D[i, j]$ (fila $i$ y columna $j$ de $D$) se encuentra la duración del viaje en el tramo que va desde el sitio $i$ hasta el sitio $j$. Si no existe un tramo desde $i$ a $j$, entonces $D[i, j] = \infty$. Los tramos son flechados.

Debido a diferentes condiciones de los tramos (calidad del pavimento, clima, cuestas, interrupción por el tráfico, etc.) el consumo de combustible en cada tramo no es proporcional a la duración del viaje en el mismo. El consumo se encuentra en la tabla $C$. Esto es, $C[i, j]$ es el consumo en el viaje por el tramo que va desde $i$ hasta $j$. La capacidad del tanque de combustible del vehículo es $T$ y se asume que está lleno al comenzar el viaje. En algunos sitios hay surtidores de combustibles en los cuales el viajero puede llenar el tanque. Esto se representa en la tabla $S$. El valor de $S[i]$ es 'SÍ' o 'NO' dependiendo de si en el sitio $i$ hay o no un surtidor.

El turista quiere determinar el camino que lo lleve desde el sitio $A$ al sitio $B$ en el menor tiempo posible. Para esto, además de las duraciones de los viajes por los tramos, debe tener en cuenta que cada vez que decida recargar el tanque en un surtidor la duración se incrementa $R$ unidades de tiempo.

En el siguiente **ejemplo** se ven las tablas $S$, $D$ y $C$, y los valores $T$, $R$, $A$ y $B$.

| Sitio | S |
|---|---|
| 1 | NO |
| 2 | SÍ |
| 3 | NO |
| 4 | SÍ |
| 5 | SÍ |

Duración (D)

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | $\infty$ | 5 | $\infty$ | $\infty$ |
| 2 | $\infty$ | 0 | $\infty$ | $\infty$ | 4 |
| 3 | $\infty$ | 3 | 0 | $\infty$ | 8 |
| 4 | 6 | $\infty$ | $\infty$ | 0 | $\infty$ |
| 5 | $\infty$ | $\infty$ | $\infty$ | 7 | 0 |

Consumo (C)

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | $\infty$ | 30 | $\infty$ | $\infty$ |
| 2 | $\infty$ | 0 | $\infty$ | $\infty$ | 45 |
| 3 | $\infty$ | 35 | 0 | $\infty$ | 65 |
| 4 | 15 | $\infty$ | $\infty$ | 0 | $\infty$ |
| 5 | $\infty$ | $\infty$ | $\infty$ | 55 | 0 |

$T = 70$
$R = 2$
$A = 3$
$B = 4$

La solución óptima es partir desde 3, pasar por 5 (recargando el tanque) y llegar a 4, con una duración 17, generada por: 8 en el tramo $(3, 5)$, 7 en el tramo $(5, 4)$ y 2 ($= R$) en la recarga necesaria en el sitio 5. La otra solución es partir desde 3, pasar por 2 (recargando el tanque), pasar por 5 (recargando el tanque) y llegar a 4. En esta la duración en los tramos es 14, menor que en la solución anterior, pero debido a que hacen falta 2 recargas (en los sitios 2 y 5) la duración total es 18. En otro ejemplo, si el destino es el sitio 1 la solución óptima es partir desde 3, pasar por 5 (recargando el tanque), pasar por 4 (SIN recargar el tanque) y llegar a 1, con una duración 23 $= 8 + 2 + 7 + 6$. No hace falta recargar en el sitio 4 ya que al llegar al sitio 1 se puede ver que el último sitio donde se recargó fue en el segundo sitio del camino, y desde ahí los consumos de los tramos son $55 + 15 \leq 70$.

**Se pide:**

Resuelva el problema usando la técnica **Backtracking**:

a. Exprese en lenguaje natural y en lenguaje formal la forma de la tupla, restricciones explícitas, restricciones implícitas y función objetivo en el caso que correspondan.

b. Dibuje el árbol del espacio de soluciones correspodiente al ejemplo anterior (con $A = 3$, $B = 4$). En cada nodo se debe mostrar el cumplimiento o no de las distintas restricciones, el valor de la función objetivo, y si corresponde o no a una solución factible o solución óptima.

## Solución

a. **Forma de la tupla**

Tupla de largo variable $t = \langle (t_1, s_1) \dots (t_p, s_p) \rangle$, que representa el camino para llegar desde $A$ hasta $B$. Los componentes son pares. El primer elemento de cada par es el identificador del sitio y el segundo elemento indica si se utilizó el surtidor de combustible. Si no hay solución la tupla debe ser vacía.

**Restricciones explícitas**

(a) Los primeros elementos de cada componente de la tupla son identificadores de sitio.

$$t_i \in \{1, \dots, n\}, \text{ para } 2 \leq i \leq p-1.$$

(b) Los segundos elementos de cada componente de la tupla son booleanos.

$$s_i \in \{0, 1\}, \text{ para } 1 \leq i \leq p.$$

(c) El inicio y el fin de la tupla son $A$ y $B$ respectivamente.

$$t_1 = A, t_p = B.$$

(d) Sólo se puede cargar combustible en los sitios en los que hay surtidor.

$$\text{Si } s_i = 1 \text{ entonces } S[t_i] = \text{'SÍ'}, \text{ para } 2 \leq i \leq p-1.$$

(e) Se asume que al comienzo el tanque está lleno y que no se recarga al final.

$$s_1 = 0, s_p = 0.$$

**Restricciones implícitas**

(a) Existe un tramo que lleva desde un sitio al inmediatamente siguiente de la tupla.

$$D[t_i, t_{i+1}] \neq \infty, \text{ para } 1 \leq i \leq p-1.$$

(b) Para cada posición $i$ se define la posición anterior a $i$ en la tupla en que se hizo la última recarga:

$$u_i = \text{máx}\{j, 1 \leq j < i \mid s_j = 1\}, \text{ para } 2 \leq i \leq p.$$

Entonces, el consumo desde la última recarga no puede superar la capacidad del tanque.

$$\sum_{j=u_i}^{i-1} C[t_j, t_{j+1}] \leq T, \text{ para } 2 \leq i \leq p.$$

(c) No puede haber ciclos.

$$\text{Si } i \neq j \text{ entonces } t_i \neq t_j, \text{ para } 1 \leq i, j \leq p.$$

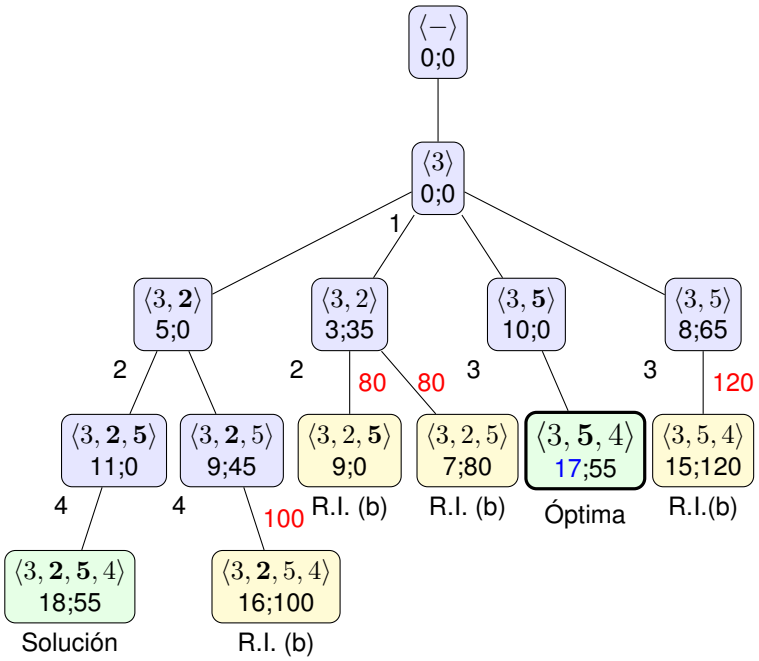
**Función objetivo** Se quiere minimizar la duración del viaje. Se suma la duración de cada tramo recorrido y se suma $R$ cada vez que se utilizó el surtidor.

$$f = \min_{t \in Sol} \left( \sum_{i=1}^{p-1} D[t_i, t_{i+1}] + R \sum_{i=2}^{p-1} s_i \right),$$

donde $Sol$ es el conjunto de tuplas solución.

b. **Árbol de soluciones**

En cada nodo del árbol se muesta la tupla, la duración y el consumo de combustible desde la última recarga. Por simplicidad del diagrama, de cada componente de la tupla sólo se muestra el elemento que representa el sitio. Si se elige cargar el tanque el sitio se destaca en negrita. En las aristas que llevan a los nodos donde no se cumple la restricción implícita (b) se destaca que el consumo supera la capacidad $T(= 70)$. No se muestran los nodos en los que no se cumplen las restricciones explícitas (por ejemplo, no se incluyen los hijos de la raíz que no cumplen la R.E (c)). Debajo del diagrama se listan las tuplas que no cumplen las restricciones implícitas (a) y (a).




1. Hijos de $\langle(3,0)\rangle$
   - que no cumplen la R.I. (a):
     $\langle(3,0),(1,0)\rangle$, $\langle(3,0),(4,?)\rangle$
   - que no cumplen la R.I. (c):
     $\langle(3,0),(3,0)\rangle$
2. Hijos de $\langle(3,0),(2,?)\rangle$
   - que no cumplen la R.I. (a):
     $\langle(3,0),(2,?),(1,0)\rangle$, $\langle(3,0),(2,?),(3,0)\rangle$, $\langle(3,0),(2,?),(4,?)\rangle$
   - que no cumplen la R.I. (c):
     $\langle(3,0),(2,?),(2,?)\rangle$, $\langle(3,0),(2,?),(3,0)\rangle$
3. Hijos de $\langle(3,0),(5,?)\rangle$
   - que no cumplen la R.I. (a):
     $\langle(3,0),(5,?),(1,0)\rangle$, $\langle(3,0),(5,?),(2,?)\rangle$, $\langle(3,0),(5,?),(3,0)\rangle$
   - que no cumplen la R.I. (c):
     $\langle(3,0),(5,?),(3,0)\rangle$, $\langle(3,0),(5,?),(5,?)\rangle$
4. Hijos de $\langle(3,0),(2,1),(5,?)\rangle$
   - que no cumplen la R.I. (a):
     $\langle(3,0),(2,1),(5,?),(1,0)\rangle$, $\langle(3,0),(2,1),(5,?),(2,?)\rangle$, $\langle(3,0),(2,1),(5,?),(3,0)\rangle$
   - que no cumplen la R.I. (c):
     $\langle(3,0),(2,1),(5,?),(2,?)\rangle$, $\langle(3,0),(2,1),(5,?),(3,0)\rangle$,$\langle(3,0),(2,1),(5,?),(5,?)\rangle$