② El problema presentado es un problema de asignación de trabajos. La búsqueda mediante ramificación y poda nos evita buscar en subárboles que no contienen una solución óptima. En la búsqueda, elegimos un nodo con el menor coste.

	Contenedor 1	Contenedor 2	Contenedor 3
ciudad 1	30	40	70
ciudad 2	60	20	10
ciudad 3	40	90	30

Para cada ciudad, elegimos el contenedor con el menor tiempo de atención (menor coste) de la lista de contenedores. (tomamos la entrada mínima de cada fila)

Trataremos de calcular el costo prometedor cuando el contenedor 1 se le asigna la ciudad 1. Dado que el contenedor 1 se le asigna a la ciudad 1 (lo marcamos en rojo), el costo se convierte en 30 y el contenedor deja de estar disponible (marcado en negro).

Ahora asignamos el contenedor 3 a la ciudad 2, ya que tiene un coste mínimo de la lista de contenedores no asignados. El costo se convierte en $30 + 10 = 40$ y el contenedor 3 deja de estar disponible.

Por último, el contenedor 2 es asignado a la ciudad 3, pues es el único contenedor disponible. El costo total es $40 + 90 = 130$.

	C_1	C_2	C_3
ciudad 1	30	40	70
ciudad 2	60	20	10
ciudad 3	40	90	30

→

	30	40	70
→	60	20	10
	40	90	30

→

	30	40	70
	60	20	10
	40	90	30

Sin embargo, también tenemos una mejor solución:

	30	40	70
	60	20	10
	40	90	30

→

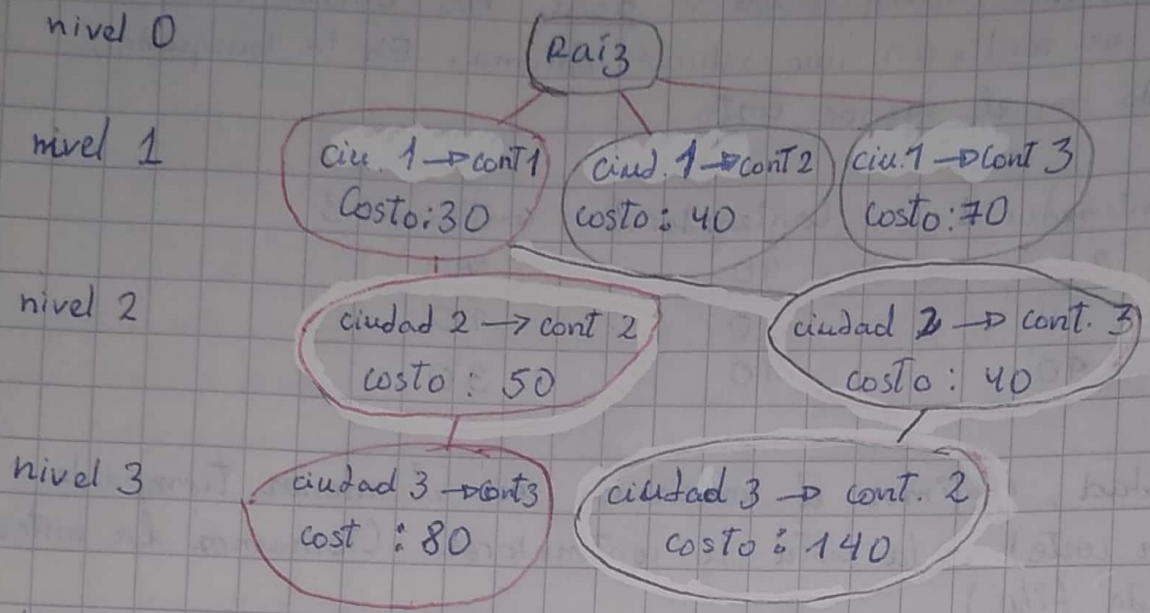
	30	40	70
	60	20	10
	40	90	30

→

	30	40	70
	60	20	10
	40	90	30

Cuyo costo total es $30 + 20 + 30 = 80$

El diagrama de búsqueda es el siguiente (ruta roja es la solución):



Algoritmo:
Antes de mostrar el algoritmo, detallaremos algunas funciones:

- findMinCost() usa Least() y Add para mantener la lista de nodos activos.
- Least() encuenta un nodo activo con el menor costo, lo elimina de la lista y lo retorna.
- Add() calcula el costo de "x" y lo agrega a la lista de nodos activos.

Implementamos la lista de nodos activos como un montículo mínimo, ó "min heap".
Ahora si, mostraremos el algoritmo:

```
// [illegible] del árbol de búsqueda
nodo {
    int: número de contenedor
    int: número de ciudad
    nodo: padre
    int: costo
}
```

```
// Entrada: Matriz de coste del problema de asignación de contenedores.
// Salida: Costo mínimo y contenedores asignados.

findMinCost ( costMatrix int[][] ) {

    // inicializamos la lista de nodos activos (min heap) con
    // La raíz del árbol de búsqueda

    mientras ( verdad) {
        // Encontramos un nodo activo de menor costo
        E = Least()

        // El nodo encontrado es borrado de la lista de
        // nodos activos.

        Si ( E es un nodo hoja) {
            imprimirSolucion()
            retornar
        }

        Para cada hijo "x" de E {
            Add (x)      // añadir "x" a la lista
                         // de nodos activos
            Padre (X) = E
        }
    }
}
```