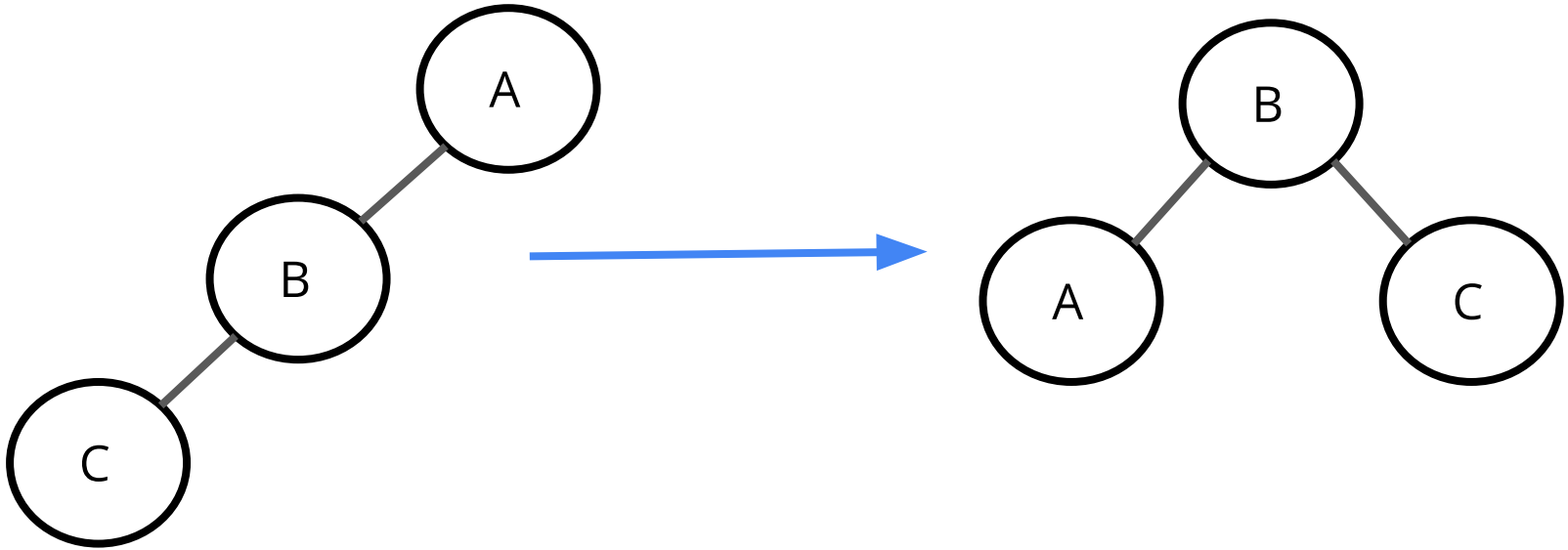
Ayudantía 8: Repaso I2

¡No se olviden que pueden llevar un resumen!

Repaso Árboles

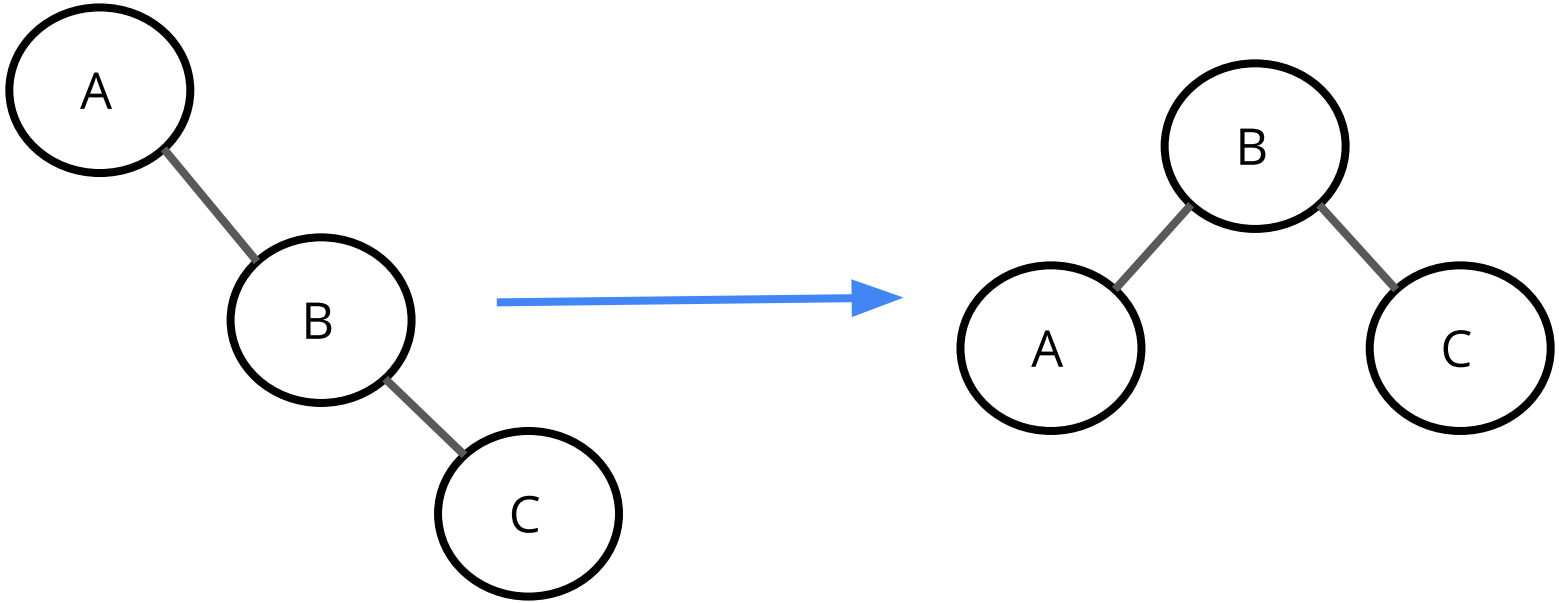
AVL

1. **Right Rotation**. Nodo es insertado en el subárbol izquierdo de un subárbol izquierdo



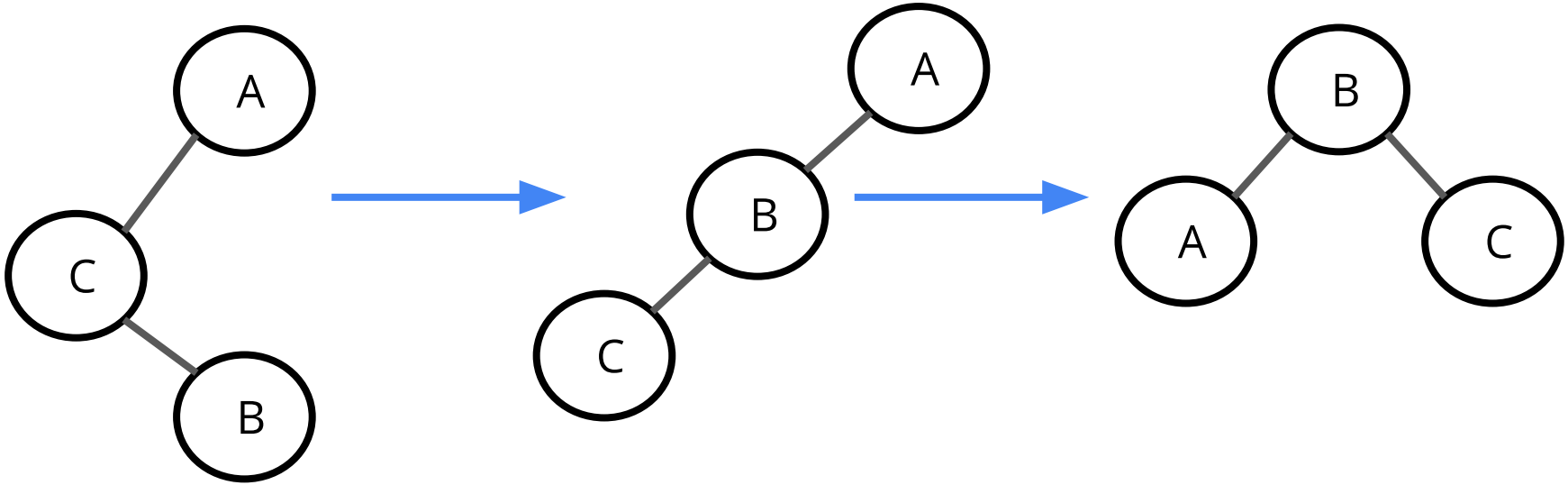
AVL

2. **Left Rotation**. Nodo es insertado en el subárbol derecho de un subárbol derecho



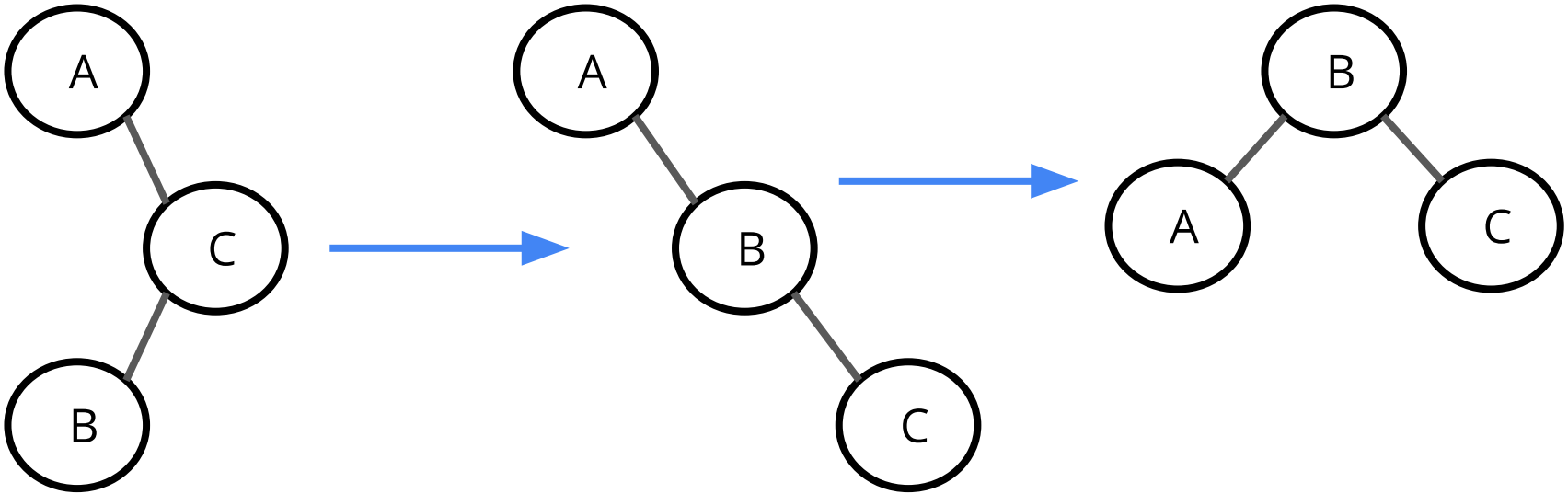
AVL

3. **Left-Right Rotation**. Nodo es insertado en el subárbol derecho de un subárbol izquierdo



AVL

4. **Right-Left Rotation**. Nodo es insertado en el subárbol izquierdo de un subárbol derecho



Árbol 2-3

Al insertar llaves:

- Se inserta como llave múltiple en una hoja existente
- Si la hoja era 2-nodo, queda como 3-nodo y terminamos
- Si la hoja era 3-nodo, queda como 4-nodo (con 3 llaves) por ahora
- La llave central del 4-nodo sube como llave múltiple al padre (**split**)
- Se repite la modificación de forma recursiva hacia la raíz

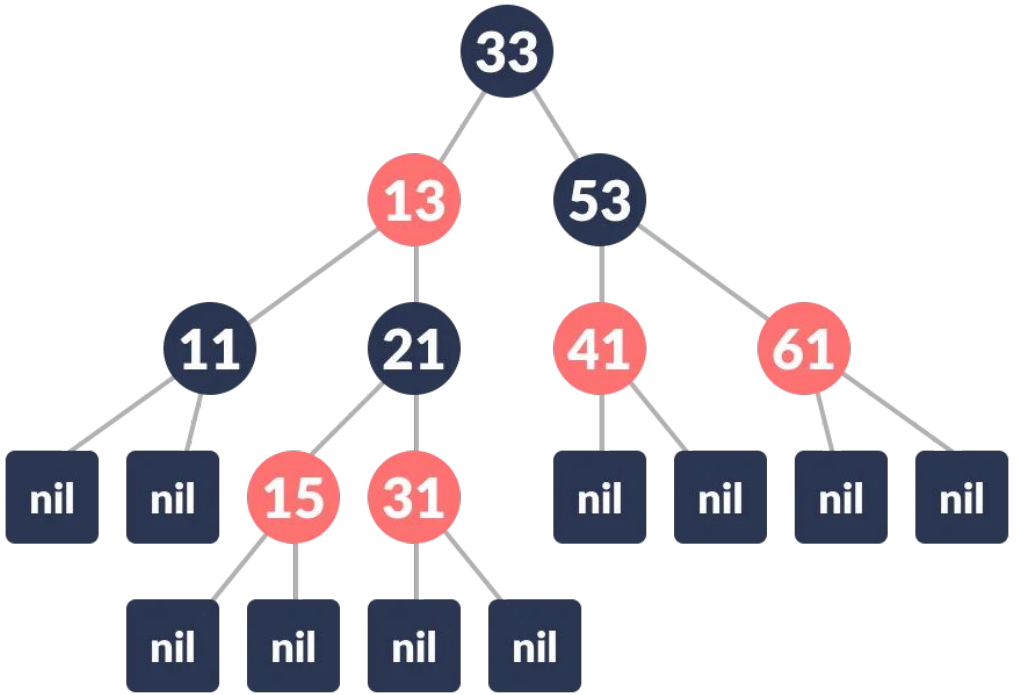
Árbol Rojo-Negro

Propiedades

- Un nodo es **rojo** o **negro**
- Los nodos raíz o hojas son **negros**
- Si un nodo es **rojo**, entonces sus hijos son **negros**
- Todas las rutas de un nodo a sus descendientes hoja contienen el mismo número de nodos **negro**

Árbol Rojo-Negro

```
FixBalance (x):
    while x.p ≠ ∅ ∧ x.p.color = rojo :
        t ← x.uncle   ▷ tío de x
        if t.color = rojo :
            x.p.color ← negro
            t.color ← negro
            x.p.p.color ← rojo
            x ← x.p.p
        else:
            if x es hijo interior de x.p :
                Rotation(x.p, x)
                x ← x.p
            x.p.color ← negro
            x.p.p.color ← rojo
            Rotation(x.p.p, x)
    A.root.color ← negro
```



Ejercicio Árboles (I2 - 2022-1)

a) Considera una secuencia de inserciones de claves distintas que se ejecuta tanto en un árbol AVL como en un rojo-negro, ambos inicialmente vacíos. La secuencia es tal que mantiene los árboles balanceados tanto como sea posible. ?¿Cuál de los dos árboles se desbalancea primero?

b) Dibuja un árbol rojo-negro, tal que si nos olvidamos de los colores **No** es un AVL.

c) Demuestra que cualquier AVL, sus nodos pueden ser pintados tal que sea un arbol rojo-negro.

Repaso Hash

Ejercicio Hash (I2 - 2022-1)

La Universidad ha desarrollado una aplicación que permite verificar en línea si un estudiante es o no alumno regular, con su número de alumno de 7 cifras. Para esto utilizan un ABB donde se guardan los datos de aprox 25.000 alumnos regulares.

a) Proponga una mejora a la estructura de datos. Justifique.
b) Escriba el pseudocódigo de búsqueda, inserción y eliminación.
c) Proponga un algoritmo para migrar los datos del ABB a la nueva solución.

Repaso Backtracking

Backtracking

```
is solvable(X, D, R):
    if X = ∅, return true
    x ← alguna variable de X
    for v ∈ D_x:
        if x = v viola R, continue
        x ← v
        if is solvable(X − {x}, D, R):
            return true
        x ← ∅
    return false
```

Poda

Se deducen restricciones a partir de las restricciones o asignaciones anteriores que pueden ser agregadas al problema.

En otras palabras, estamos podando parte del conjunto de caminos a soluciones posibles.

```
is solvable(X, D, R):
    if X = ∅, return true
    x ← alguna variable de X
    for v ∈ D_x:
        if x = v no es válida, continue
        x ← v
        if is solvable(X − {x}, D, R):
            return true
        x ← ∅
    return false
```

Propagación

Cuando a una variable se le asigna un valor, se puede propagar esta información para luego poder reducir el dominio de valores de otras variables.

```
is solvable(X, D, R):
    if X = ∅, return true
    x ← alguna variable de X
    for v ∈ D_x:
        if x = v viola R, continue
        x ← v, propagar
        if is solvable(X − {x}, D, R):
            return true
        x ← ∅ , propagar
    return false
```

Heurística

Una heurística es una aproximación al mejor criterio para abordar un problema.

```
is solvable(X, D, R):
    if X = ∅, return true
    x ← la mejor variable de X
    for v ∈ D_x, de mejor a peor:
        if x = v viola R, continue
        x ← v
        if is solvable(X − {x}, D, R):
            return true
        x ← ∅
    return false
```

Ejercicio Backtracking (I2 - 2022-1)

Viendo un pasillo como una matriz de 0s y 1s, representando zonas inseguras y seguras respectivamente, un robot (de tamaño 1x1) debe avanzar por el pasillo. Solo puede avanzar por fila o por columna.

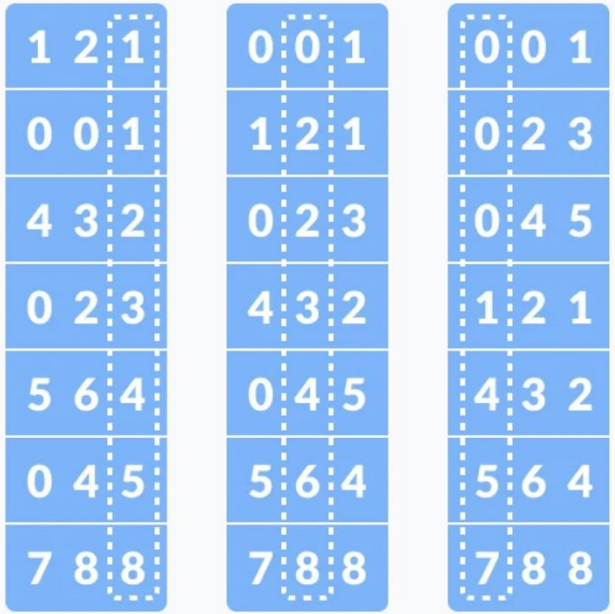
Dada cualquier matriz de tamaño n y un número m de celdas inseguras distribuidas en la matriz se busca encontrar un camino que llegue desde alguna celda en la izquierda a alguna celda en la derecha, sin pasar por celdas inseguras ni adyacentes a inseguras.

Ejercicio Backtracking (I2 - 2022-1)

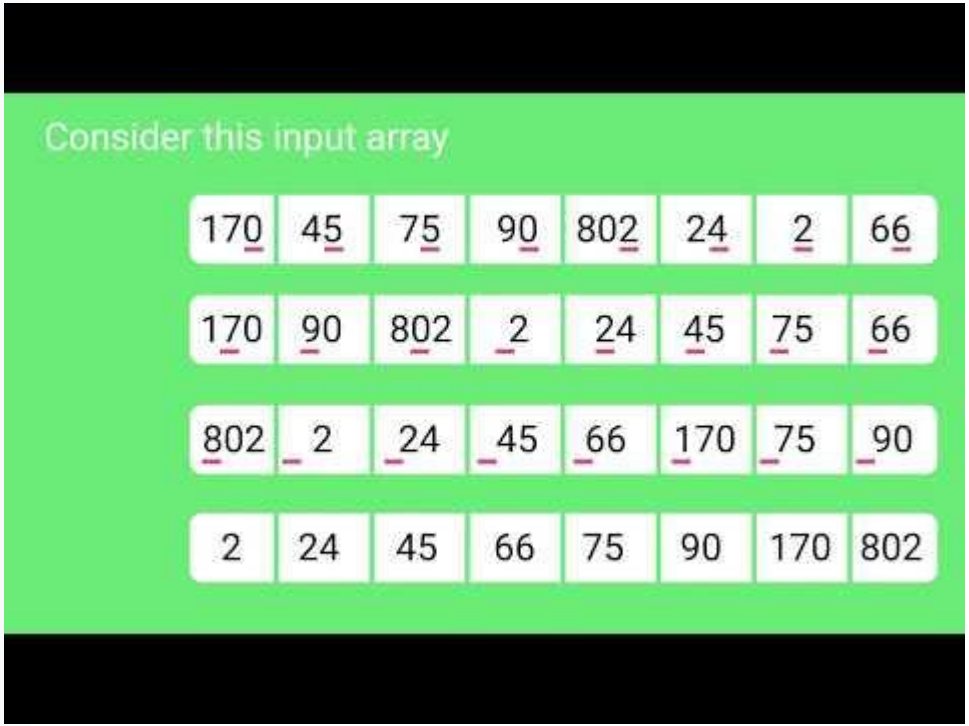
a) Identificar variables, dominios y restricciones.
b) Proponer un algoritmo que resuelva el problema, marcando con X las celdas del camino e indicando su largo
c) Modificar el algoritmo de b) para que encuentre el camino más corto.

Repaso Sorting O(n) (Radix)

Radix Sort



sorting the integers according to units, tens and hundreds place digits



Radix Sort

```
radixSort(array)
  d <- maximum number of digits in the largest element
  create d buckets of size 0-9
  for i <- 0 to d
    sort the elements according to ith place digits using countingSort

countingSort(array, d)
  max <- find largest element among dth place elements
  initialize count array with all zeros
  for j <- 0 to size
    find the total count of each unique digit in dth place of elements and
    store the count at jth index in count array
  for i <- 1 to max
    find the cumulative sum and store it in count array itself
  for j <- size down to 1
    restore the elements to array
    decrease count of each element restored by 1
```

Ejercicio Radix Sort

Una central telefónica registra en un archivo LOG las llamadas realizadas durante un mes dado. En cada fila registra: fecha_llamada, hora_llamada, numero_origen, numero_destino, duracion_llamada. Se desea utilizar LOG para realizar la facturación del servicio telefónico, indicando para cada número de origen el detalle en orden cronológico de cada llamada realizada, y su costo total (duracion_llamada x P).

Ejercicio Radix Sort

a) Proponga una solución para encontrar la facturación indicada de la manera más eficiente en tiempo posible.
b) Muestre que su solución cumple con los requisitos indicados.
c) Un compañero le propone usar quickSort en LOG y separar por origen. Compare su solución con la de su compañero en términos de eficiencia.

¡Muchas gracias y suerte en la prueba!