Taller 1 Racket y recursividad Parte 3
Racket is love, is life. The final challenge!
Fundamentos de Lenguajes Programación

Carlos Andres Delgado S
carlos.andres.delgado@correounivalle.edu.co

Marzo de 2023

1. Reglas

- Realizar en grupos de 2 a 4 personas, en caso de presentar individualmente o un grupo de 2 o 3 personas no permitan el ingreso de alguien que lo requiera, se sancionará con 0.5 en la nota del taller por incumplimiento de esta regla.
- Debe usarse recursión para resolver los problemas, no se permiten funciones como map o sort que resuelven el problema directamente
- Debe usarse sintaxis de Racket paréntesis en caso de entregar en papel, en computador debe ser funcional en lenguaje Racket.
- Las funciones deben definirse con **lambda**

1.1. Indicadores de logro

1. IL 1.1.1 Construye datos recursivamente utilizando métodos de inducción y gramáticas BNF
2. IL 1.1.2 Utiliza gramáticas BNF para guiar la construcción de programas recursivos

2. Enunciado del taller

De acuerdo a la gramática de listas:

```
<lista>      ::= '()
             ::= <numero> <lista>
```

Una lista de listas

```
<lista-s>      ::= '()
               ::= <lista> <lista-s>
```

Y la gramática de árboles

```
<arbol-b>          ::= '()
                   ::= <numero>
                   ::= <numero> <arbol-b><arbol-b>
```

1. Un árbol binario de búsqueda es aquel que dado un nodo x sus hijos izquierdos son menores a él, y sus hijos derechos son mayores o iguales que el, diseñe la función **insercion-cosmica** la cual recibe un árbol y una lista de números a insertar, esta función retorna el árbol con los valores insertados respetando esta regla. Ejemplo:

```
(define arbol1
  '(10
    (7 (4 3 6) (9 8 () ))
    (15 (12 11 13) (17 16 20))
  )
 )

(define datosInsertar
  '(2 3 4 10 22 32 23)
  )

  (insercion-cosmica arbol1 datosInsertar)
```

   Da como resultado

```
(10
    (7
        (4
            (3 2 3)
            (6 4 ()))
        (9 8 ()))
    (15
        (12 (11 10 ()) 13)
         (17
            16
             (20
                ()
                (22
                    ()
                    (32 23 ()
    ))))))
```

   El ejemplo gráfico lo puede ver en la página 4 de este enunciado.

2. La distancia a dos puntos n-dimensionales está dado por:

$$\sqrt{(y_1 - x_1)^2 + (y_2 - x_2)^2 + .. + (y_n - x_n)^2} \tag{1}$$

Ecuación de distancia en línea recta

   Diseñar la función **distancia-superpro** la cual recibe una lista con n elementos que indican un punto y una lista de listas de n elementos, la cual indica los puntos a los cuales vamos a medir la

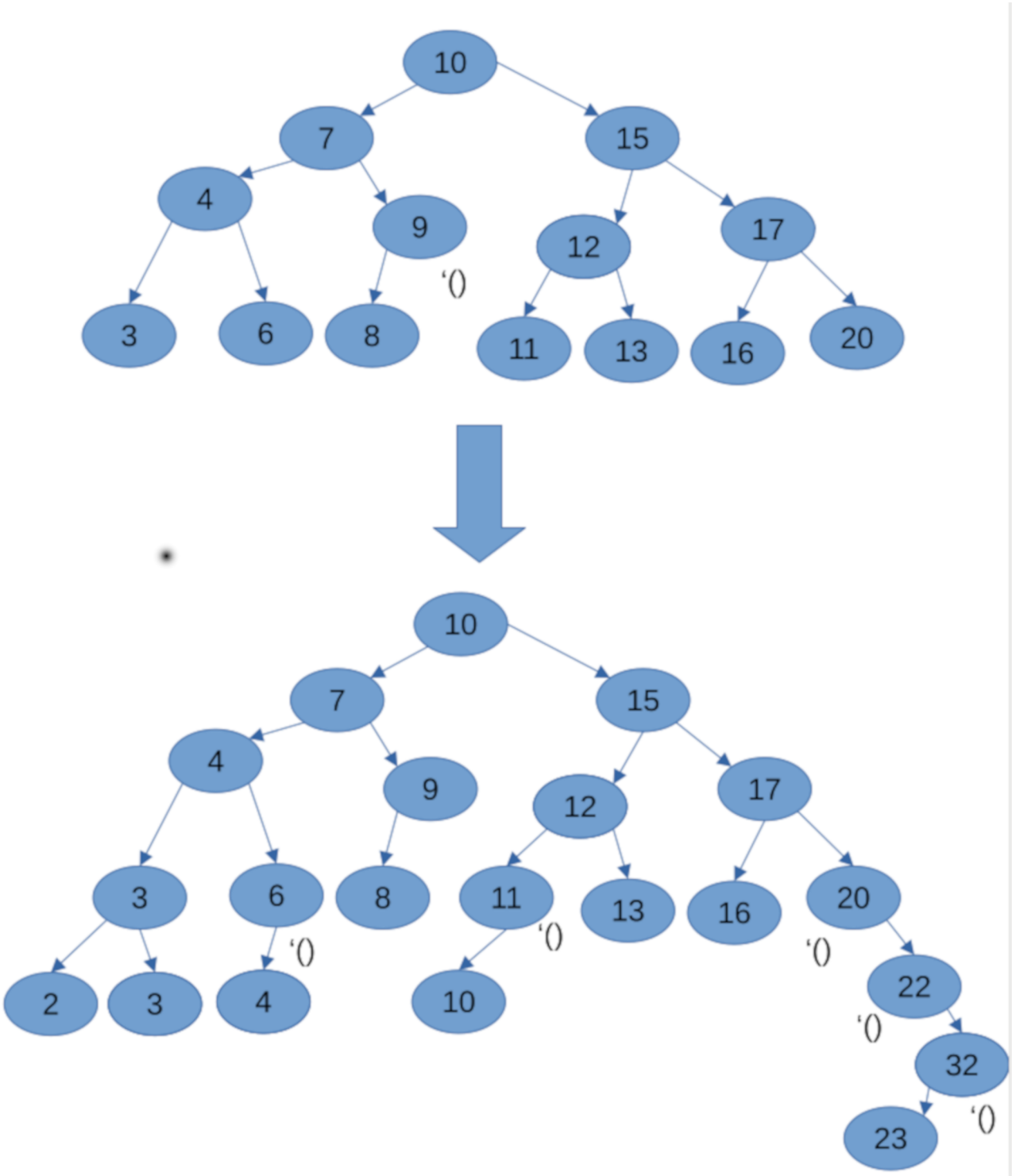
distancia. Esta debe retornar una lista de tuplas, cuya primera posición es una lista con n elementos y su segunda posición es el valor de la distancia, esta debe estar ordenada de menor a mayor. **Nota:** No se puede usar ninguna función como map o sort o alguna función que resuelva el problema, debe resolver todo recursivamente. Su uso afectará en gran medida la nota del punto.

```
(define punto
  '(0 0 0 0 0 0 0 0))

(define listapuntos
  '(
    (1 1 2 2 3 3 4 4)
    (9 9 2 3 1 2 3 2)
    (4 2 1 2 3 2 4 0)
    (3 9 1 2 8 2 3 5)
    (-3 3 -4 2 2 2 -3 1)
    (1 2 3 5 4 5 -2 2)
 ))
(distancia-superpro punto listapuntos)
```

Debe retornar

```
(((4 2 1 2 3 2 4 0) 7.3484692283495345)
 ((-3 3 -4 2 2 2 -3 1) 7.483314773547883)
 ((1 1 2 2 3 3 4 4) 7.745966692414834)
 ((1 2 3 5 4 5 -2 2) 9.38083151964686)
 ((9 9 2 3 1 2 3 2) 13.892443989449804)
 ((3 9 1 2 8 2 3 5) 14.035668847618199))
```


10
7
15
4
9
12
17
3
6
8
11
13
16
20
10
7
15
4
9
12
17
3
6
8
11
13
16
20
2
3
4
10
22
32
23