Algoritmo recursivo para generar un código de Huffman dada su distribución probabilistica, junto el código asociado a cada una de sus reducciones.

Sean...

- $R$ un vector de objetos con las propiedades: $c, p, 0$ y opcionalmente 1, y representa una reducción.
- $R_{-1}$ la reducción anterior a $R$, y $R_2$ la reducción que sigue a $R$.
- $R[i].p$ la probabilidades de aparición del $i$-ésimo símbolo en $R$.
- $R[i].c$ la palabra código asignada al $i$-ésimo símbolo de $R$.
- $R[i].0$ el índice del símbolo en $R_{-1}$ del que salió $R[i].p$, es decir,

$$R[i].p = R_{-1}[R[i].0] \iff \nexists R[i].1$$

- Si $R[i].p$ es el resultado de la suma de dos probabilidades de la reducción anterior, entonces $\exists R[i].1$, y por lo tanto:

$$R[i].p = R_{-1}[R[i].0] + R_{-1}[R[i].1] \iff \exists R[i].1$$

El algoritmo asume que la distribución de probabilidades de la reducción para la que debe generar el código ya existe, por lo que, en base a ella, crea la distribución de probabilidades para la reducción siguiente, y vuelve a llamarse a si mismo, enviando como parámetro a esta distribución que creó, para encontrar el código de la siguiente reducción.

Una vez encontrada la última reducción (aquella que sólamente tiene 2 símbolos, cuyas palabras código son 0 y 1), hace backtrack... luego el algoritmo ya conoce los códigos de las reducciones siguientes, por lo que, de acuerdo a los índices guardados en $R_2[i].0$ y $R_2[i].1$, construye el código correspondiente al símbolo en $R[R_2[i].0]$ y, si existiera en $R[R_2[i].1]$.

```
1  function H (R, reducciones)
2  R_2 = []
3  a_{m_1} = min(R)
4  a_{m_1} = 2ndMin(R)
5  if |R| == 2 then
6  |  R[0].c = "0"
7  |  R[1].c = "1"
8  |  return { R, reducciones }
9  end
10 for r_i in R do
11 |  if i ∉ {m_1, m_2} then
12 |  |  R_2.push({0 : i,  p : r_i.p})
13 |  end
14 end
15 R_2.push({0 : m_1,  1 : m_2,  p : a_{m_1} + a_{m_2}})

16 H(R_2, reducciones)

17 for R_{2_i} in R_2 do
18 |  if ∃R_{2_i}.1 then
19 |  |  R[R_{2_i}.0].c = R_{2_i}.c + "0"
20 |  |  R[R_{2_i}.1].c = R_{2_i}.c + "1"
21 |  else
22 |  |  R[R_{2_i}.0].c = R_{2_i}.c
23 |  end
24 end
25 reducciones.push(R_2)
26  R, reducciones
```