

Garsia-Wachs Documentation

Application:

This can be used to build optimum search tables and to balance rope data structure in an optimal way (In text editing, rope data structure may be used to represent the string to be edited). A rope is a binary tree where each leaf node holds a string and a weight, and each node further up the tree holds the sum of the lengths of all the leaves in its left subtree.

Algorithm:

Main Function:

- 1) Define Node weights, left and right children , depth , working region, number of nodes as Arrays of size 64 or any random number globally, also initialise t and m where t is the size of the working region and m is the current node.
- 2) Take n as the size of the weights array and then take weights as input from user.
- 3) Now initialize all the values of left and right children with -1 and region as infinite (as first and last elements are compared with infinity) and initialise $t = 1$.
- 4) Now
 - a) for($k=1$ and $k \leq n$)
 - i) While $q[t-1]$ is less than equal to $w[k]$ do combine(t)

ii) Now increment t and replace the value of q[t] with w[k] here k will be n and the replace v[t] with k (this is done to move to the next weight and next node in the region).

b) while t>1 do combine(t);

c) call the mark function that it mark(v[1], 0); and make t = 0

d) Now call build function

e) Now run For loop with i =0 and i<= 2*n and initialize ans integer for storing the value of the weight sum, Now

i) if(l[i]!= -1) ans +=w[l[i]];

ii) else ans = w[i];

iii) if(r[i]!=-1) ans+= w[r[i]];

iv) Else ans = w[i];

v) W[i] = ans;

f) Now again run the for loop with i =2*n and i>=0 , basically run a reverse for loop

i) if(r[i]!= -1) then d[r[i]] = d[i] +1

ii) if(l[i] != -1) then d[l[i]] = d[i] +1

g) basically above two for loops are for adding weights and depth which are optimal

h)Now again run the for loop for printing the output._

Combine function :

- a) This recursive subroutine is the heart of the Garsia-Wachs algorithm. It combines two weights, shifts them left as appropriate, and maintains the 2-descending condition. Variables j and w are local, but variables k , m , and t are global.
- b) Now create a new node and at this point we have $k \geq 2$ so increment the m by 1 and put the value of left children as $v[k-1]$ and right children as $v[k]$
- c) Now for the weight as $W[m] = q[k-1] + q[k]$ and decrease the t
- d) Now shift the following nodes left Set $t \leftarrow t - 1$, then $P_j \leftarrow P_{j+1}$ for $k \leq j \leq t$ and then shift the preceding nodes right Set $j \leftarrow k - 2$; then while $\text{Weight}(P_j) < w$ set $P_{j+1} \leftarrow P_j$ and $j \leftarrow j - 1$
- e) Insert the new node and set $P_{j+1} \leftarrow X_m$
- f) Now if $j=0$ and $\text{weight}(P_{j-1}) > w$.exit this function
- g) Now restore Set $k \leftarrow j$, $j \leftarrow t-j$, and call this function recursively. Then reset $j \leftarrow t - j$ (note that t may have changed!) and return to step f
- h) It basically gives the optimum sequence

Mark Function :

- a) It is used to store the depth of every node recursively
- b) First initialize the depth with 0.
- c) Then if $l(k) \geq 0$ call $\text{mark}(l(k), p+1)$ and if $r[k] \geq 0$ then call $\text{mark}(r[k], p+1)$.recursively.
- d) The value of p here is used to store the depth value of every node.

Build Function :

Here basically we are calling the build function to generate the final tree that gives the optimal solution

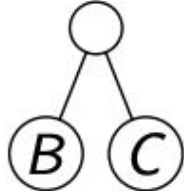
- a) Initialise $j = m$
- b) If m is equal to 0 then return
- c) If $d[t] == b$ then $l[j] = t++$ else $m--$, $l[j] = m$, $\text{build}(b+1)$;
- d) If $d[t] == b$ then $r[j] = t++$ else $m--$, $r[j] = m$, $\text{build}(b+1)$
- e) This way build function is called recursively to build a new tree that will be the optimal tree

Example : A, 3; B, 2; C, 1; D, 4; E, 5 [INPUT LIST]

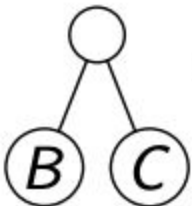
❑ Determine the smallest i such that $\text{weight}(t_{i-1}) \leq \text{weight}(t_{i+1})$

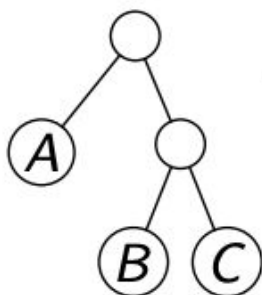
$\textcircled{A}, 3 \quad \textcircled{B}, 2 \quad \textcircled{C}, 1 \quad \textcircled{D}, 4 \quad \textcircled{E}, 5$ $i = 2$

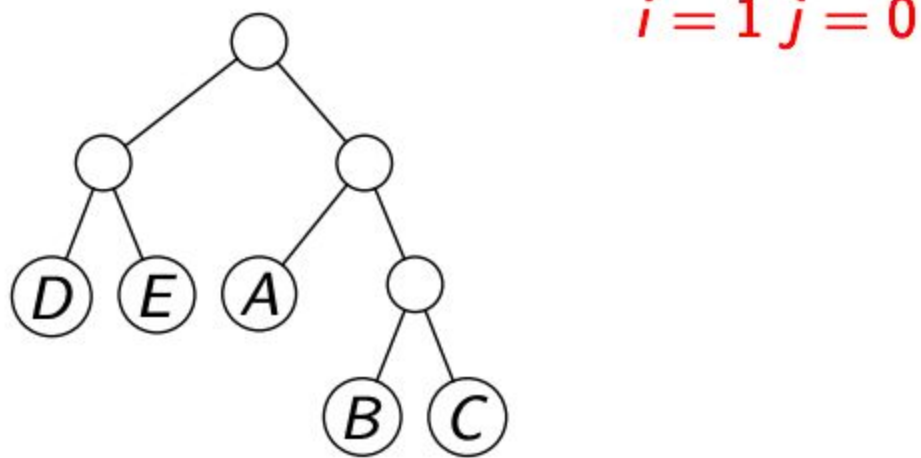
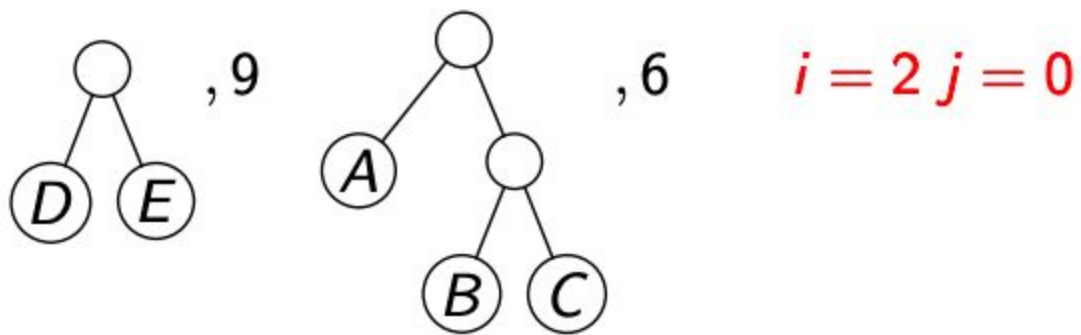
❑ Link t_{i-1} and t_i , with weight $w = \text{weight}(t_{i-1}) + \text{weight}(t_i)$

$\textcircled{A}, 3 \quad \textcircled{D}, 4 \quad \textcircled{E}, 5$ $t =$  $w = 3$

❑ Insert t at largest $j < i$ such that $\text{weight}(t_{j-1}) \geq w$

$\textcircled{A}, 3$  $, 3 \quad \textcircled{D}, 4 \quad \textcircled{E}, 5$ $j = 1$

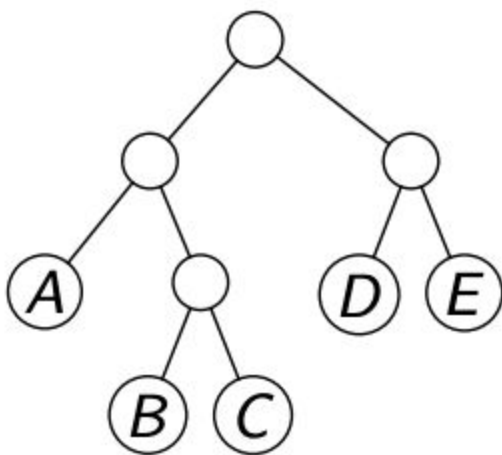
 $, 6 \quad \textcircled{D}, 4 \quad \textcircled{E}, 5$ $i = 1 \quad j = 0$



The depths (inorder) are : A B C D E : 2 3 3 2 2

The algorithm now uses the above tree and builds a tree with leaf nodes in order.

i.e



The Final tree