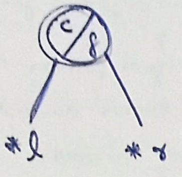
Hullman Encoding Text file Compression

what is node?



contains : a character
its frequency
and a left and right
pointer to other
child node.

what is a pointer : A value which refers to address of an object

eg: a is a variable
&a is address of a

then P be pointer of a
ie, P = &a

∴ *P = a ; * : dereference operator

ie, *&a = a

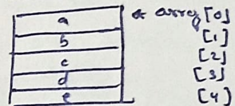
what is struct min-heap?

Contains : size ; no: of elements in heap (unique characters)

Node** array ; here array is a pointer which points to other node pointers → ie, array is a pointer which points to a set of pointers of type Node*

eg: min-heap: size = 5
array = 3200

3200 have 5 x size of node*
ie, *3200 have



and a, b, c, d, e are pointers (Node*) to each node of unique character.

[Before we continue,

consider a text file, open using FILE* and a pointer to begining of file is provided
It's run along each character to obtain freq

count-freq → freq1 [i]++ for each time char i read

ie, if 'a' is read once then ASCII of a to be 37 then freq1 [37]++

∴ finally freq1 [0] = 0
[1] = 0
⋮ ⋮
[37] = 1 ← no: of counts
⋮ ⋮
[255] = d

[Freq [ASCII index value]]

to form character array and frequency array of only present characters node
count = 0;
i run from 0 → 255 if freq1 [i] > 0 append freq[count] = freq1 [i];
charac[count] = (char) i ;
count ++

∴ if a : 5 time , b : 3 time , d ; 9 times

then freq = { a , b , d }
character = { 5 , 3 , 9 }

Now min-heap to form nodes and allocate all nodes in a sp/c memory location. (pointer 8136936 location - te)

struct min_heap* createheap (freq, characters, count)

struct min heap* minheap = (struct (min heap*) malloc (sizeof (struct min-heap));

↑ type of value returned is pointer to a minheap struct.

↑ our min-heap bind size 888 location 2000?? and pointer of location

min-heap → size = count;

↑ indicate min-heap is a pointer of struct ←

(if min-heap was struct itself I can use min-heap.size) the same functionality is)

min-heap → array = (struct node**) malloc (min-heap → size * sizeof (struct Node*));

↑ count * size of pointer of node

eg: 5 x pointer [illegible] [illegible] [illegible] pointer

now, we read each character [i] and

return [illegible] to array.

node* new node (char [i], freq [i]) is stored at → array [i]

[struct node* new node (chara [i], freq [i]) {
struct node* node = (struct node*) malloc (size of struct node);
node → character = chara [i]; node → l = NULL
node → freq = freq [i]; node → r = NULL
return node }

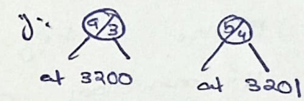
Now we have; min heap: size = 5

array

3200	3201	-	-	.
[0]	[1]	[2]	[3]	[4]

contain pointers to each node of single character.

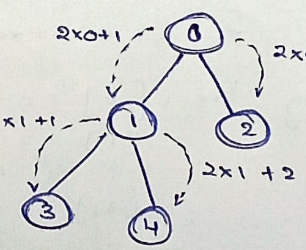
eg:



we need to rearrange this in an ascending order in order to extract min frequency node.

we need to arrange array now based to frequency order.

[note that: in any tree order is given as



parent = i
∴ l: 2i+1
r: 2i+2

[void Heapify (min-heap, i)

↓ start with i = (count-1)/2

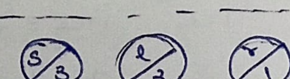
then i-- after each heapify. this helps in going through all nodes and heapify recursively as required.

smallest = i
l = 2i+1
r = 2i+2
] indexes of l, r array [i]

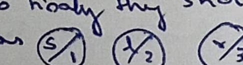
if (min heap → array [l] → freq < min heap → array [smallest] → freq)
smallest = l;

if (- - - - - - - [r] → freq < - - - - - - - -)
smallest = r;

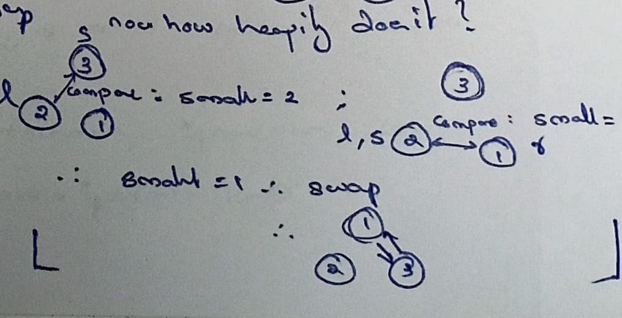
if (smallest != i) ← means smallest frequency char was not at index i in array then, swap
then swap array [smallest] and array [i]
recursively heapify (min-heap, smallest)
[



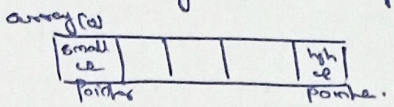
So finally they should be arranged as



now how heapify does it?



Now we got min heap: size = 5



Now. we need to build tree

struct *node tree (min heap) {

node * l, * r, * top

while (!isSizeOne(min-heap)) { runs until only one element left in minheap ary

smallest ary[0] → l = extract min (min-heap);

next smalst ary[1] → r = extract min (min-heap)

insert ie heapify so that ary[1] is the new ary[0]

top = new node ($, l→freq + r→freq)

random any character

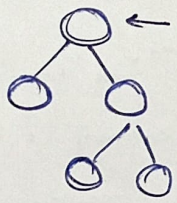
freqs y sum

top → l = l

top → r = r

insert into heap(min-heap, top)

[basically we extracted 2 min node, added them up to form a new node and placed it back in heap and do this repeated by until only one node left in minheap and its pointer is returned to tree



]

[extract min (min-heap)

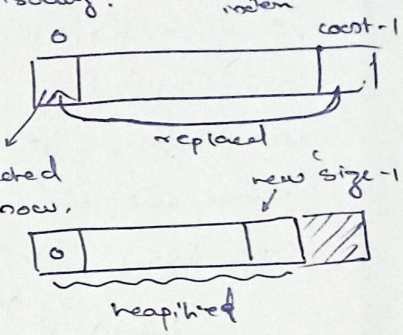
return minheap → array[0]

and reach minheap→array[0] with minheap→array[size-1]

size--

and heapify it. ie, we always traverse →(minheap, 0)

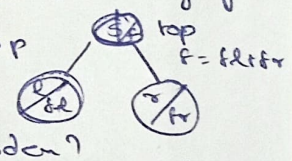
[visually:



]

[insert into heap (min-heap, top)

we need to find proper index of where to place it according to the freqency of node top

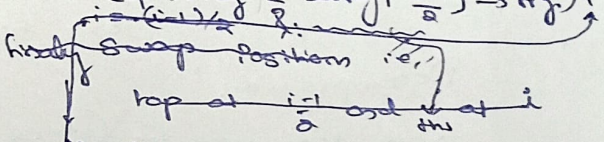


how to find index?

increase const to fit new node.

let i = const - 1 ; ie, last blank position.

while (top→freq < ary[(i-1)/2] → freq) {



minheap→array[i] = minheap→ary[(i-1)/2]

} i = (i-1)/2

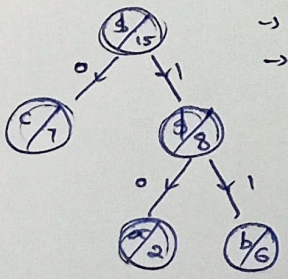
minheap→array[i] = top;

]

[ie, we place array with freqency greater than top to the bottom positions (last)

until we have appropriate index for place top]

So now we have to generate code words for each characters.

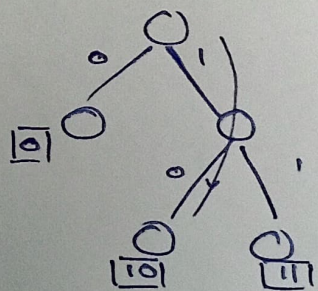


→ char code intermittently stores code

→ char codes [↑][→] finally goes to

ASCII of characters at leaf node ie end node

∴ codes have codeword at index as ASCII words of each chars.



void generate code (tree, code, codes){

if (!l and !r then \0 added to code

and code goes to codes [ch] [ ]
NULL

if (tree → l)

len = strlen(code)   (initially zero)

code [len] = '0'

code [len+1] = '\0'

generate code (tree → l, code, codes)

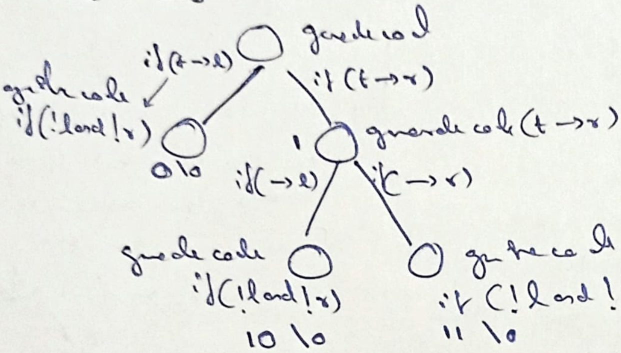
code [len] = '\0'.

if (tree → r)

for if (tree → r)
replace
with
code [len] = '1'

note: if (tree → l / → r)
is always if (1) ie, it will
run for sure cause
tree → l is a pointer value
of a new node unless it
an end node and values
>0 are all considered YES
∴ if (1) means sure run.
it will traverse along all
nodes recursively.

[adding \0 to next index doesn't
really do any good in b/w as it's
not considered as a real character
it easily replace while recursively
carrying along the tree]

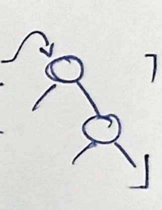


Now just read i/p file and obtain int ch ie ⇒ ASCII value and
print codes [ch] in o/p file.

to decode
our binary tree is already provided

Decompression:
void decompression (tree, o/p file)   (pointer to read)
As pointer of file move along
it read bit = 0 ; current = current → l
bit = 1 ; current = current → r

[initially current = tree ie
and we replace current* as we traverse]
(struct node data type)



if (!current → l and !current → r)
print (current → character) to decompressed file.
and reset pointer ie, current = tree