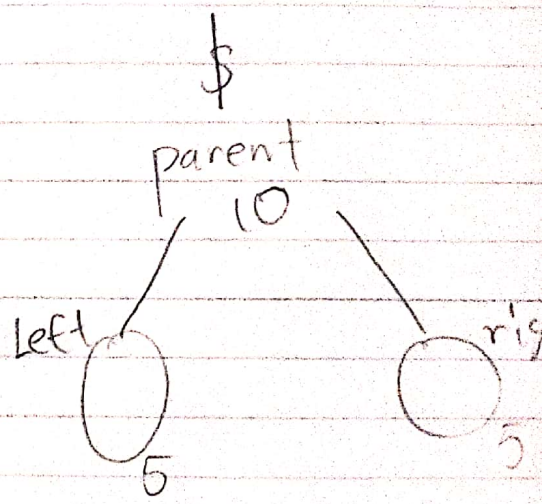
Design

Node

Node* left
Node* right
uint8_t symbol
uint64_t frequency

Node_create

Left = 0 / NULL
Right = 0 / NULL
symbol = symbol
frequency = frequency



Void node_delete

free left and right
free * to node
*node = NULL

Node *node_join(left, right)

parent = node_create('$', left + right frequency)

parent's left = left

parent's right = right

Priority Queues

Have a
- tail
- head
- size
- capacity
- Node **arr

PriorityQueue create(capacity)
- head, tail = 0
- q->capacity = capacity
- calloc(capacity) for queue.

PriorityQueue - delete(**pq)
- free arr
- free *pq
- q = NULL

pq-empty(*q)

If tail = 0
- true

else
- false

pq-full

if head = ALPHABET = 256

true

else false

pq-size(*q)

return size of queue

enqueue(*q, *n)

if queue is full then false

Place node at tail

tail += 1

size += 1

true.

dequeue(*q, **n)

if queue is empty then false

nothing to dequeue.

take from head.

tail -= 1

size -= 1

Code

top = 0

for loop to set 0 bits 2^3 to 0.

code_size (Code *c)

return the c->top

code_empty (Code *c)

if code_size(c) = 0 then true

if not false

code_full (Code *c)

if code_size = ALPHABet then true

if not false.

code_set_bit

shift the bit

then or

want to be 1

```
0 0 0 1
or 0 1 0 1
-----------
0 1 0 1
```

code_clr_bit.

shift bit → ~()

then & operator.

Code_get_bit

And with 1 then shift.

Code_push_bit

if full, false no space to push-

c->bits array Eat the top] = bit.

Increment top- by1

Code_pop_bit

if empty then false nothing to pop.

c -> top decrement by 1

*bit = bits array at top minus 1

Stack

Stat_create(capacity)

calloc' to allocate memory

top = top

capacity = capacity

stack_delete(stack *s)

free items

free pointer to stack

stack = NULL
pointer.

Stack empty ( )

if top = 0 true

Stack full

if top is == to capacity

Stack_size

for loop through the stack.
to count.

Stack_push

array index at top = Node

Stapek_pop

Pops node passing through double pointer