asgn5

dump tree

parent node   section notes

*recursion

*could use dfs

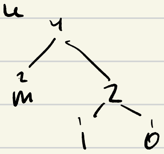
huffman.c

alphabet 256 - 80 256 possible combos (8bit)

-Length of string (string / 8)

Huffman coding

- 1st count how many times a character is used & put it in order (priority queue)
- very last ones will be the lower branches
  - how often they use = frequency
- add them up w/ sum of frequencys
  - add sum back into list where it needs to be, repeat
  - when one sum is the same as frequency add them up: example



- at the end will end with one big thing / starting
  - it also shows how to convert back
  - store tree

Node.c

```
Node *create (symbol, frequency)
    Node *n=(Node*) malloc (node);   //allocating memory
    if(n) {
        n->symbol= symbol
        n-> frequency= frequency
        n->left =Null
        n->right=Null
    }
    return n;

void Delete_node
    if(n){
        free(n)
        n= null
    }
Node *node_join (Node *left, Node *right){
    x= left->frequency + right->frequency
    Node *n= node create ($, x);  // $ parent Node
    n->left= left
    n->right=right
    return n;
}
```

priority Queue

like stack but opposite. So I plan on using the same concept

```
Struct PriorityQueue {
    uint32_t capacity;
    uint32_t first;
    uint32_t len; //for current items
    Node *queue;   //maybe
};

PriorityQueue *pq_create (uint32_t capacity){
    PriorityQueue *p = (PriorityQueue *) malloc(sizeof(PriorityQueue));
    if(p){
        p->len = 0;
        p->capacity = capacity;
        p->first = 0;
    }
}

pq_empty(*p){
    return (p->length=0)
}

pq_full(*p){
    return(p->length = capacity)
}

pq_size(*p){
    return(p->length)
}

pq_enqueue(*p, *n){   needs edit (low frequency = high priority)
    if(pq_full){
        Status=false;
    }else{
        insert sort first
        p->items[p->length] = n  // following stack
        length+=1
        Status=true
    }
pq_dequeue(
    if(empty){
        Status=false
    }else{
        length-1
        p->items[p->length *n
        length+1
    }
```

Codes

```
code_init(void) {
    c->top = 0;   *no need to malloc
    c->bits[c->top] = 0;
    return c
}
code_size(*c){
    return top;
}
code_empty(q
    return (top == 0)
}
code_full(*c){
    return (top = MAX_CODE)
}
code_set_bit(*c, i)
    if (i < MAX_CODE)
        bits[i] = 1
        return true
}
code_clr_bit(*c, i){
    if (i < Max){
        bits[i] = 0
    return true
    }
}
code_get_bit(*c, i){
    if(i < Max){
        b = bit[i]
        if(b = 0){
            false;
        true
        }
    }
```

*A a lil confused about bit in pop & push. am i changing the array?

I/O

* need to go to section / a little confused on what true purpose

-needs to be able to read all

read_bytes will be used when reading any info

-I assume in main instead of open()

*ask if what i used in asgn4 good to use