Assignment 6
Huffman Code
DESIGN.pdf

Reuben T. Chavez

March 2, 2022

Pseudeocode

```
#Libaries
import pq
import huffman
import stdlib
import stdint

# Psuedocode for encode.c

## Usage ##
function usage is
    input: executable
    output: void

    print(
    "Synopsis of encode\n"

    "Usage of encode\n"

    "Options for encode\n"
    )

function main is
    input : argument count argc and argument vector argv
    output: zero to exit program

    intialize opt to 0
    initialize input to standard input
    initialize output object to standard output
    initalize compression statistics as false

    while getting commands from command line do
        switch command:
            case h:
                call usage funcntion
                break
            case i:
                if file exists:
                    input is set to read file
                    break
                print that the file does not exist
                stop running
```

```
        case o :
            if file exists:
                output is set to read file
                break
            print that the file doe not exist
            stop running
        case compression stat:
            set compression stat to true
        default help:
            prints usage and ends program

    Compute histogram of input file, and count occurences of
        euniquew symbol in file

    Construct Huffman tree with histogram using Priority Queue

    Construct Code Table

    Emit encodeing to given file, through a post-order traversal,
        calling huffman's tree dump

    Step Through each symbol of the input file, emit code to output
        file


    if compression stat:
        print(
                Uncompressed File Size:
                Compressed File Size:
                Space Saving:
             )

    close all opened files
    return 0
```

```
#Libaries
import pq
import huffman
import stdlib
import stdint

# Psuedocode for decode.c
```

```
## Usage ##
function usage is
    input: executable
    output: void

    print(
    "Synopsis of decode\n"

    "Usage of decode\n"

    "Options for decode\n"
    )

function main is
    input : argument count argc and argument vector argv
    output: zero to exit program

    intialize opt to 0
    initialize input to standard input
    initialize output object to standard output
    initalize compression statistics as false

    while getting commands from command line do
        switch command:
            case h:
                call usage funcntion
                break
            case i:
                if file exists:
                    input is set to read file
                    break
                print that the file does not exist
                stop running
            case o :
                if file exists:
                    output is set to read file
                    break
                print that the file doe not exist
                stop running
            case compression stat:
                set compression stat to true
            default help:
                prints usage and ends program

    Read tree dump from given input file
```

```
    Read rest of input file bit by bit , traversing down huffman
        tree one link at a time,
    reading zero go left reading one go right

    if compression stat:
        print(
               Compressed File Size:
               Decompressed File Size:
               Space Saving:
             )

    close all opened files
    return 0
```

```
#Pseudocode for Nodes

##Libraries

import stdint.h

define type Node;

Initialize struct Node with:
    - left Node
    - right Node
    - unsigned interger of 8 bits that's a symbol
    - unsigened inter of 64 bit to frequncy

Function Node Create:
    input: two unsigened integer, one a 8 bit for the node's symbol
           the second a 64 bit for frequncy
    output: a pointer to a Node type

    Allocate space for Node n with a size of Node

    Node n's Symbol = symbol
    Node n's Frequncy = frequncy

    return Node n



Function Node Delete:
```

```
    Input: Double pointer N
    Output: None since functioin is void

    free memory of given input
    Previous Node is set to Null

Function Node Join:
    Input: left node and riht node
    Output: node pointer

    set unsigned integer of 8 bits to dollar-sign

    set frequncy to sum of left and right frequncy

    Initalize parent node with function Node Create and symbol and
        frequncy as input

    set parent left to left child's left
    set parent's right to right child's right

    return parent node

Function print node:
    input: pointer to node
    output: Nothing function is void


    print the data item withn the pointer node
```

```
#Pseudocode for pq.c

#Libraries
import node
import stdbool
import stdint


struct PriorityQueue
    - contains head
    - contains tail
    - contains capacity
    - contains Node array

Function Create Priority Queue:
```

```
    Input: An unsigned intger of 32 bits
    Output: Priorit Queue pointer

    - Allocate space for a Priority Queue pointer
    - Initialize head, tail, capacity, and the Node array if the pq
        is not NULL

Function Insertion Sort:
    Input: A Priority Queue and Node
    Output: Nothing function is void
    - For iteration of Priorty queue:
        - set j to current index
        - create temp of current index in PQ array
        - While j is greater than 0 and the temp is greater than the
            last index
            - set array at index j to the last insex
            - subtract j by 1
        - set the Priorty Queue on index j to temp

Function pq delete:
    Input : Double pointer to Priority Queue
    Output : Nothing function is void

    - If the input is not null, free double pointer and set
        previous node to NULL

Function pq full:
    Input : Double pointer to Priority Queue
    Output: boolean

    - rturn that given pq is either full or not


Function pq empty:
    Input : Double pointer to Priority Queue
    Output: boolean

    - rturn that given pq is either empty or not


Function pq size:
    Input : Priorty Queue pointer
    Output: Unsigned 32 interger

    - return the the top node in pq
```

```
Function enqueue:
    Input: Priorty Queue pointer and Node pointer
    Output: boolean

    - if Priorty Queue not null,
        - if empty return false
        - add node to head of pq
        - Resort the tail node to the in the pq using a sorting
            algorthim
    -return true to signify that the pq was succefully enqued

Function dequeue:
    Input: Priorty Queue double pointer and Node pointer
    Output: boolean

    - if Priorty Queue not null,
        - if full return false
        - remove node from tail of pq
        - Resort the tail node to the in the pq using a sorting
            algorthim
        - sub top by 1
    -return true to signify that the pq was succefully enqued

Function pq print:
    Input: Priorty Queue Node
    Output: Nothing the function is void

    - Print all items with pq
```

```
#Pseudocode for stack.c

#Libraries
import node
import stdbool
import stdint
import stdlib

Stack struct:
    - contains top
    - contains capacity
    - contains double pointer node array
```

```
Function stack create
    Input: unsigned 32 bit integer
    Output: Pointer to Stack

    - Allocate memory fro STACK object
    - Initilize items within Stack struct
    - return stack pointer

Function stack delete:
    Input: Stack pointer
    Output: Nothing function is void

    - Delete specifed stack, and set previous stack to NULL

Function stack empty:
    Input : Stack pointer
    Output: boolean

    - return if the stack is empty

Function stack full:
    Input : Stack pointer
    Output: boolean

    - return if the stack is full

Function stack push:
    Input: Stack pointer and pointer to node
    Output: boolean

    - Check if the stack is not full, if its add more space to stack
    - Add stack top and set node pointer equal to


Function stack pop:
    Input: Stack pointer and double pointer to node
    Output: boolean
```

#Pseudocode for code.c

#Pseudocode for huffman.c

```
#Pseudocode for
```