CS653

Text File Compression Using Huffman Algorithm

PROJECT REPORT

Group Members
Rohan Jingar
Vimal Sharma

Huffman Algorithm :.

- First we need to scan the input text file and record the number of occurences of each character as a list of tuples

- Example:
  [(123,'a'),(43,'g'),(15,'n'),(124,'r')]

- We have declared a data type Priorityqueue to implement priority queue methods required for the construction of Huffman Tree.

```
data Ord k => Priorityqueue k a = Null
   |Node k a (Priorityqueue k a) (Priorityqueue k a)
```

- Note that 'a' is polymorphic and we will be using this to construct the huffman tree when we need to insert a node where 'a' can be a huffman tree.

- The "fromList" function takes a list of the form mentioned above and constructs the min priority queue.

```
fromList::Ord k=>[(k,a)]->Priorityqueue k a
      fromList xs = foldl (\acc x -> merge acc x) Null nodes
        where nodes = map (\(x,y) -> Node x y Null Null) xs
```

Huffman Tree:

Huffman tree has the following data type:

```
data HuffmanTree a = Empty
                   | Node (HuffmanTree a) (HuffmanTree a)
                   | Leaf a
```

Construction of Huffman Tree:

- Take the queue constructed above and extract the two minimum weight nodes.
- Let w and w' be their respective weights.
- Insert a new node with weight (w+w').
- Repeat this procedure until we are left with a single node queue.
- This single node is the Huffman Tree.

Generation of Codewords

To generate the codewords, traverse the Huffman Tree adding a '0' for each left move and a '1' for each right move.The codeword corresponding to a character at the leaf node will be the list of 0's and 1's we have generated upto that node.

encode.hs

- Given the name of an input text file it will generate the character frequency list and the huffman tree for that file.

- It will store the frequency list to <filename>.key

- And then using the list it will compress it to <filename>.cmp

- It can be confirmed by using the diff unix command.

- First we read the entire file and generate a frequency list and based on that a priority queue pq is generated and using pq a huffmantree tree is generated.
- By traversing the whole tree we generate a codelist which contains the bit code of each character

```
codelist :: [ ( Char , [ Bit ] ) ]
```

- Encoding:
  - Now since we have the code list we read the file again and store a list containing the encoded Bits of the character we encounter.
  - Then we take first 8 Bits from the above list and convert it into word8 (unsigned 8 bit integer) and then into corresponding bytestring.
  - The bytestring is then written into the binary compressed file.

Decode.hs

- Given file names of the compressed file and the key file it will first generate the huffman tree from the frequency list stored in the key file.

- We read from the compressed file a list of byte words (word8). Then convert each byte into binary (a list of 0's and 1's).

- Based on the above bits (0 or 1) we traverse the huffman tree generated for the original file.

- If at any point we get a leaf we put that character into the output file.

Problems faced:

- Reading/Writing Binary file was a little tricky. We can not directly write bits into a file. First we have to wrap the list of bits in groups of 8 (a byte). Then convert these 8 bits into a bytestring which can then be written to a file.

- Maintaining the height of the priority queue after merge operation such that insert and extract operations are O(log n) was a bit tricky. We made the left subtree of the original tree as the new right subtree. The new left subtree is the tree obtained by merging right subtree and the other tree.

The project has a darcs repository at patch-tag.com:
to get the latest patch:

```
$darcs get http://patch-tag.com/r/svimal/huffman_coding
```

To install it in your ~/bin directory:

```
$cabal install --prefix=$HOME -user
```

or

```
$runhaskell Setup configure --user --prefix=$HOME
$runhaskell Setup build
$runhaskell Setup install
```