huffman encoding

Michaela Elise (117) Sep 30, 2013 at 4:26pm

This is our code from a class assignment. I am posting it in case it helps anyone else out. I know it is a little messy, but it works (had to get it done quickly). I ran diff on the original and decoded files and got 0 differences.

I have plans to change the code around. Create a header struct, possibly read/write binary, restructure to use less memory, and just clean it up in general.

As you may notice, generating a trie using a priority queue is only a few lines of code, very simple and clean. The key is to use a custom compare function to reverse the PQ.

The biggest epiphany I had in doing this program was that after creating a string of bitcodes, I already had the 1's and 0's in the order I needed so the encoding (insertBits function) only needs an OR and a left shift.

```
#include <sstream>
#include <fstream>
#include <string>
#include <stdlib.h>

class bitChar{
public:
        unsigned char* c;
        int shift_count;
        std::string BITS;

        bitChar();
        void setBITS(std::string _X);
        int insertBits(std::ofstream& outf);
        std::string getBits(unsigned char _X);
        void writeBits(std::ofstream& outf);
        ~bitChar();
};

```

```
#include "bitChar.h"

bitChar::bitChar()
{
        shift_count = 0;
        c = (unsigned char*)calloc(1, sizeof(char));
}

void bitChar::setBITS(std::string _X)
{
        BITS = _X;
}

//Returns number of bits inserted
int bitChar::insertBits(std::ofstream& outf)
{
        int total = 0;

        while(BITS.length())
        {
                if(BITS[0] == '1')
                        *c |= 1;
                *c <<= 1;
                ++shift_count;
                ++total;
                BITS.erase(0, 1);

                if(shift_count == 7)
                {
                        writeBits(outf);
                        shift_count = 0;
                        free(c);
                        c = (unsigned char*)calloc(1, sizeof(char));
                }
        }

        //Account for any trailing bits and push them over
        if(shift_count > 0)
        {
                *c <<= (8 - shift_count);
                writeBits(outf);
                free(c);
                c = (unsigned char*)calloc(1, sizeof(char));
        }

        return total;
}

//Outputs a char in binary format
std::string bitChar::getBits(unsigned char _X)
{
        std::stringstream _itoa;

        int _size = sizeof(unsigned char) * 8;

        for(unsigned _s = 0; _s < _size - 1; ++_s)
        {
                _itoa << ((_X >> (_size - 1 - _s)) & 1);
        }

        return _itoa.str();
}

void bitChar::writeBits(std::ofstream& outf)
{
        outf << *c;
}

```

```
bitChar::~bitChar()
{
	if(c)
		free(c);
}
```

```
#include <queue>
#include <iostream>
#include <vector>
#include <fstream>
#include <iomanip>
#include "bitChar.h"

const std::string magicNum="7771234777";
const int asciiSize = 256;
int lCount[asciiSize];
std::string str_code[asciiSize];

struct node{
	char ch;
	int count;
	node* left;
	node* right;
};

class cmp{
public:
	bool operator()(const node* lhs, const node* rhs) const
	{
		return lhs->count > rhs->count;
	}
};

node* makeNode(char ch, int count)
{
	node* tmp = new node;
	tmp->ch = ch;
	tmp->count = count;
	tmp->left = NULL;
	tmp->right = NULL;
	return tmp;
};

typedef std::priority_queue<node*, std::vector<node*>, cmp> mypq;

void trie(mypq& _X)
{
	while(_X.size() > 1)
	{
		node* holder = new node;
		holder->left = _X.top(); _X.pop();
		holder->right = _X.top(); _X.pop();
		holder->count = holder->left->count + holder->right->count;
		holder->ch = -1;
		_X.push(holder);
	}
}

//Create bit codes by recursively traversing the trie, adding a 0 for left and 1 for right, the key is to remove the en
void code(node* _X)
{
	static std::string bits = "";
	if (_X->right != NULL)
	{
		bits += "1";
		code(_X->right);
		bits = bits.substr(0, bits.size() - 1);
	}
	if (_X->left != NULL)
	{
		bits += "0";
		code(_X->left);
		bits = bits.substr(0, bits.size() - 1);
	}
	if(!_X->left && !_X->right)
	{
		str_code[_X->ch] = bits;
	}
}

void count(std::string file, int& _X){
	char letter;
	std::ifstream inf(file.c_str());

	inf >> std::noskipws;

	//Clears array
	for(int i = 0;i < asciiSize; ++i)
		lCount[i] = 0;

	//Goes through text and counts
	while(inf >> letter){
		if(letter >= 0 && letter < asciiSize)
		{
			++lCount[letter];
			++_X;
		}
	}
	inf.close();
}

//Generates a string of the bit codes in the order they appear in the file
//Used during encoding
std::string BITSstring(std::string inFile)
{
	char input;
	std::string BITS = "";

	//Open input stream and create BITS string of entire file
	std::ifstream inf(inFile.c_str());
	inf >> std::noskipws;
```

```
106
107         while(inf >> input)
108         {
109                 BITS += str_code[input];
110         }
111
112         inf.close();
113
114         //Append ascii 3 EOT character to signify end of text
115         BITS += str_code[3];
116
117         return BITS;
118 }
119
120 int main(int argc, char** argv)
121 {
122         int rc;
123         char choice;
124         unsigned char inChar;
125         std::string inFile = "", outFile = "", BITS = "", BITSsub = "", mn = "";
126         std::ofstream outf;
127         std::ifstream inf;
128         mypq pq;
129         bitChar bchar;
130         int origSize = 0;
131
132         std::cout << "Menu..." << std::endl << "e) Encode file" << std::endl << "d) Decode file" << std::endl;
133         std::cin >> choice;
134
135         switch(choice)
136         {
137         case 'e':
138                 //Get input filename and set output filename
139                 std::cout<<"Enter File Name to Encode: "<<std::endl;
140                 std::cin>>inFile;
141
142                 outFile = inFile + ".mpc";
143
144                 std::cout << std::left << std::setw(17);
145                 std::cout << "Input filename: " << inFile << std::endl;
146                 std::cout << std::left << std::setw(17);
147                 std::cout << "Output filename:" << outFile << std::endl;
148                 std::cout << std::endl;
149
150                 //Open output streams
151                 outf.open(outFile.c_str());
152
153                 //count and populate array of letter occurrences (lCount) and add one EOT char
154                 count(inFile, origSize);
155                 if(lCount[3] == 0)
156                         lCount[3] = 1;
157
158                 //Output compressed file header
159                 outf<<magicNum<<std::endl;
160                 outf<<inFile<<std::endl;
161                 for(int i = 0; i < asciiSize; ++i)
162                 {
163                         outf << lCount[i] << " ";
164                 }
165                 outf << std::endl;
166
167                 //Create nodes based on the available ascii characters and push them into the priority queue
168                 for(int i = 0; i < asciiSize; ++i)
169                 {
170                         if(lCount[i] > 0)
171                         {
172                                 node* tmp = makeNode(i, lCount[i]);
173                                 pq.push(tmp);
174                         }
175                 }
176
177                 //Create trie and bit codes
178                 trie(pq);
179                 code(pq.top());
180
181                 //Create string of bitcodes for actual huffman encoding and do it
182                 BITS = BITSstring(inFile);
183                 outf << '#';
184                 bchar.setBITS(BITS);
185                 outf << std::noskipws;
186                 rc = bchar.insertBits(outf);
187
188                 if(rc == BITS.length())
189                 {
190                         std::cout << "Encoding succsessful! :)" << std::endl;
191                         std::cout << "The compression ration is: " << (float)rc / ((float)origSize * 8.0) * 100.0 << "%
192                 }
193                 else
194                 {
195                         std::cout << "There was an error writing the bits! :(" << std::endl;
196                         std::cout << "Expected: " << BITS.length() * 8 << " but got: " << rc << std::endl;
197                 }
198
199                 break;
200         case 'd':
201                 //Get input filename and set output filename
202                 std::cout<<"Enter File Name to Decode: "<<std::endl;
203                 std::cin>>inFile;
204
205                 inf.open(inFile.c_str());
206                 inf >> mn;
207                 if(mn != magicNum)
208                 {
209                         std::cout << "Magic number does not match, this is not a valid file..." << std::endl;
210                         return 1;
211                 }
212
213                 inf >> outFile;
214                 if(outFile != inFile.substr(0, inFile.length() - 4))
215                 {
216                         std::cout << outFile << " " << inFile.substr(0, inFile.length() - 4) << std::endl;
217                         std::cout << "File names do not match but will attempt to decode anyway..." << std::endl;
```

```
218                 }
219                 outf.open(outFile.c_str());
220 
221                 //Read in the letter count and add valid one to the priority queue
222                 for(int i = 0; i < asciiSize; ++i)
223                 {
224                         inf >> lCount[i];
225                         if(lCount[i] > 0)
226                         {
227                                 node* tmp = makeNode(i, lCount[i]);
228                                 pq.push(tmp);
229                         }
230                 }
231 
232                 //Create trie and bit codes
233                 trie(pq);
234                 code(pq.top());
235 
236                 while(inChar != '#')
237                 {
238                         inf >> inChar;
239                 }
240 
241                 inf >> std::noskipws;
242                 //Read in encoded chars and create BITS
243                 while(inf >> inChar)
244                 {
245                         BITS += bchar.getBits(inChar);
246                 }
247 
248                 inf.close();
249 
250                 for(int i = 0; i < BITS.length(); ++i)
251                 {
252                         BITSsub += BITS[i];
253                         for(int j = 0; j < asciiSize; ++j)
254                         {
255                                 if(BITSsub == str_code[j])
256                                 {
257                                         //End of text has been hit and file is over, write newline and exit
258                                         if(j == 3)
259                                         {
260                                                 outf << "\n";
261                                                 i = BITS.length();
262                                                 break;
263                                         }
264                                         outf << (char)j;
265                                         BITSsub = "";
266                                         break;
267                                 }
268                         }
269                 }
270 
271                 break;
272         default:
273                 std::cout << "Invalid choice...." << std::endl;
274                 break;
275         }
276 
277         outf.close();
278 
279         return 0;
280 }
```

Last edited on Sep 30, 2013 at 4:29pm

Topic archived. No new replies allowed.