Greedy And DFS In Huffman Coding

Raydhitya Yoseph 13509092
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13509092@std.stei.itb.ac.id
raydhitya.yoseph@gmail.com

***Abstract*—*This paper contains explanation of Huffman coding algorithm in encoding and decoding. The explanation sequence are Huffman coding, greedy algorithm, Huffman encoding, deep first search algorithm, and Huffman decoding. All the explanation show Huffman encoding uses greedy algorithm and Huffman decoduing uses deep first search algorithm.*

***Index Terms*—*algorithm, coding, DFS, greedy, Huffman.*

I. Huffman Coding

Huffman coding is a very popular coding to represent data with minimum memory needed to store the data. Huffman coding was created to reduce data redundancy in a file. Huffman coding was developed by David A. Huffman while he was a Ph.D., student at MIT, and published in the 1952 paper "A Method for the Construction of Minimum-Redundancy Codes.

A. Character Encoding

One file compromised of many bits to represent its content. That bits is grouped into sets of bits. This grouped bits represent something that understandable in human language. For example the character 'a' is represented by 01000001. Changing the representation of bits into character is called character encoding scheme. Representation of the human readable character is highly dependent at the coding used. For example ASCII and UNICODE have different representation of the character 'a' in bits.

B. Statistical Data Redundancy

A character can be inside the file multiple times. For example in the string "sunny", the data of character 'n' is inside the string twice and therefore stored twice. Storing a character more than once is what we called statistical data redundancy. It is redundant and wasting space to store the same data more than once, even though, the character needs to be more than once to complete the information. This what the Huffman coding accomplishes which is storing an information in a efficient way using minimum memory.

C. Huffman Coding

The idea of Huffman coding is using the frequency of data inside a file. Data which has the highest frequency being assigned by the shortest bit and data which has the lowest frequency being assigned by the longest bit. This means using variable-length data as opposed to fixed-length data in the raw information. Fixed-length all data have same length whether the data is frequent or not. This will cause an information needs larger space.

Bit assigning is determined by making a tree structure data first. Each node from top to bottom will contain data from the most frequent to the least frequent data. The consensus is left child assigned by '0' and right child by '1'. The tree will continue to span until it reaches a leaf that contains one data. So, a node in Huffman tree will always have two children.

Bit assigning also depends on the amount of unique data. If there are two data, we only need '0' and '1'. If there are three data, we need '0', "10", and "11". The more the data are, the longer the bit needed. Compared to 8 bit extended ACII binary code which often used, Huffman coding save many wasted space.

Given a string there are 4 step to do the coding:

1. Count the frequency of each character in the string;
2. Form the Huffman tree based on the character frequency from bottom up using greedy algorithm;
3. Assign each character in the tree according to the convention bit;
4. Convert the given string old bits representation into new bits representation using the Huffman code for each charcter.

II. Greedy Algorithm

A. Definition

Given a problem and find its best solution. One from many ways to do that is by greedy algorithm which solve the problem step by step. Greedy algorithm is an algorithm that choose the best solution at each stage, which called local optimum, in hoping to get the best solution for the problem, called global optimum.

The choices made during each stage is the best solution for the current stage and without calculating future

consequences. This what made greedy algorithm will not always give the best solution for the problem because the best solution for the current stage is not always the best solution.

The choices made cannot be changed, once an element choosen as the best solution at the current stage and be added to the solution it will remain the solution until the end. The best solution at each stage can be made by all information given for each element in candidate set. For example in a candidate set each of the elements have information A and information B. The best choices can be made calculating information A hence greedy by A or calculating information B hence greedy by B.

B. Property

Generally greedy algorithm has 4 properties:

1. Candidate Set
   A set contains element from which a solution is created.
2. Solution Set
   A set contains solution elements from the best candidate. This set is a subset of candidate set.
3. Selection Function
   A funtion which select the best candidate from each step.
4. Feasibilty Function
   A function which check whether the best candidate can be added to the solution or not given the constraints.
5. Objective Function
   A function which add a value to the solution

Using the general property of greedy algorithm the algorithm can be constructed as follows:

1. initialize solution set with empty set;
2. select a value from the candidate set using the selection function;
3. remove the choosen value from the candidate set;
4. check if the choosen value can be added to the solution or not using feasibility function;
5. if the choosen value can be added to the solution, add it to the solution;
6. repeat number 2 until candidate set is empty;
7. if candidate set is empty the solution set is the solution for the problem.

III. ENCODING HUFFMAN CODE

Forming huffman tree after calculating the frequency of all the characters appearing in the given string is one example of greedy algorithm. Huffman tree is formed from bottom to top by joining two least frequent data into one and continue up.

Greedy properties of forming Huffman tree procedure

1. Candidate Set
   Trees each of which is one node containing each of the characters which appeared in the given string and its frequency.
2. Solution Set
   The Huffman tree which contains all elements from the candidate set.
3. Selection Function
   A function which gives two trees with the minimum root node character frequency.
4. Feasibility Function
   Not exist because all elements in the candidate set will be in the solution set.
5. Objective Function
   A procedure which combines two trees given by the selection function.

Using above properties forming Huffman tree algorithm can be constructed as follows:

1. initialize solution set with candidate set;
2. select two trees from the candidate set;
3. combine the two tree into one with root node frequency is the sum of the two trees' root node frequency
4. Repeat number 2 and 3 until one tree left

An example of the algorithm using following information

Table 1 Huffman Table 1

Symbol	A	B	C	D	E
Frequency	15/39	7/39	6/39	6/39	5/39

With the selection function two trees which have the lowest frequency are choosen. Two trees with the lowest frequency are D and E. Those two trees combined together forming a tree with root node frequency equals to D and E frequency which is 11/39. The first step will make the table as follows

Table 2 Huffman Table 2

Symbol	A	B	C	DE
Frequency	15/39	7/39	6/39	11/39

With the selection function two trees which have the lowest frequency are choosen. Two trees with the lowest frequency are B and C. Those two trees combined together forming a tree with root node frequency equals to B and C frequency which is 13/39. The second step will make the table as follows

Table 3 Huffman Table 3

Symbol	A	BC	DE
Frequency	15/39	13/39	11/39

Two trees with the lowest root node frequency are BC and DE. Those two trees are selected by the selection function and combined together forming a tree with root node frequency equals to BC and DE root nodes frequencies which is 24/39. The third step will make the table as follows

Table 4 Huffman Table 4

Symbol	A	BCDE
Frequency	15/39	24/39

Both tree A and tree BCDE will be choosen to form tree ABCDE with root node frequency is 39/39. The algorithm will stop because the amount of remaining tree is one.

Above algorithm will be easier to understand represented using tree

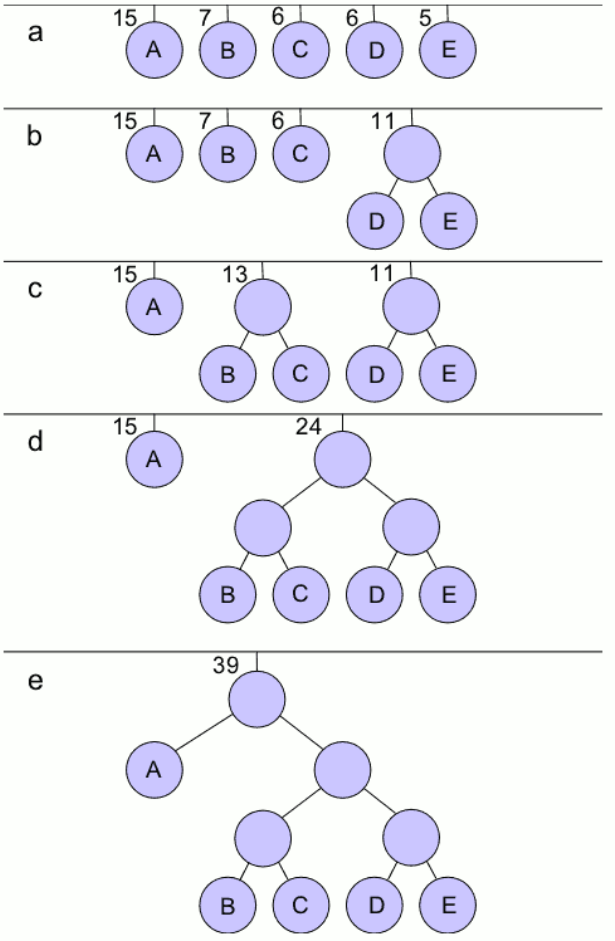

Figure 1 Huffman Tree

The algorithm will make symbol given with following code

Table 5 Huffman Coding Result

Symbol	ASCII Code	Huffman Code
A	0100 0001	0
B	0100 0010	100
C	0100 0011	101
D	0100 0100	110
E	0100 0101	111

If we used fixed-length data, we need 8*39 = 312 bit. If we used variable-length data as the Huffman code, we only need 1*15 + 3*7 + 3*6 + 3*6 + 3*5 = 87 bit. Huffman coding saved 312-87 = 225 bit, which is a huge 72.11% of space. Averagely Huffman coding will save 20% to 30% of space

VI. Deep First Search

Given a tree there are two ways to traverse the graph, which are breadth first search and deep first search.

Breadth first search is a searching algorithm which begins at root node and explores all the neighboring nodes. For all the nearest neighboring nodes, the search explores its neighboring nodes and so on until the goal.

Deep first search is a searching algorithm which begins at root node and explores each branch as far as possible until the goal. The algorithm can be modified using bactracking which traverses the previous node when the goal is not reached.

Deep first search can be implemented recursively and non-recursively. The algorithm recusively is as follows:

1. start at root node;
2. check whether the node is the goal or not;
3. if the node is the goal, stop searching;
4. if the node is not the goal, choose one child;
5. recursively traverse the child tree;
6. do number 2 to 4 until the goal.

The algorithm non-recursively using a queue as follows:

1. start at root node;
2. add the root node to queue;
3. take the first node of the queue out;
4. check whether the node is the goal or not; if yes, stop searching;
5. add all neighboring nodes to the queue;
6. repeat number 3 to 4 until queue is empty.

VI. Decoding Huffman Code

Above process is called coding which is to convert information into another form of representation, in Huffman code case into a tree structure data. This process is done for the information to get communicated. To recognize the information again, we need the reverse process which is called decoding. Decoding converts the information back into its original representation so it could be recognized by the receiver.

Decoding an information coded with Huffman coding can be achieved by two means. First is to traverse the tree, write the symbol found during the traversal, traverse again from the tree's root. The first method is a modified version of deep first search algorithm. Second is to use a lookup table which contains all symbols and its bit code. To compare the two methods we'll use previous example and try to decode “01100111” into “ADAE”.

A. Traversing Huffman Tree

The idea of the first method is to traverse the Huffman tree bit by bit and write the symbol found. To decode

“01100111” we traverse the string and find '0'. Like DFS we begins from root node. In here the implementation is better to implements recursive DFS. We only doing recursive into the child node with coresponding bit with the string. So we are doing recursive into the root's child assigned by '0'. There we find a symbol which is A therefore we write 'A' and back again to the tree's root.

Second traversal we find the bit '1' so we do recursive to the right child from the root node. At the right child we don't find a symbol therefore we traverse into the next bit which is '1' and again move to the right child. The second right child also doesn't contain a symbol so we move to the next bit which is '0'. Moving to the left child will get us a symbol which is 'D'. We write 'D' and again come back to tree's root.

Repeating the process will get us 'A' and 'E' as the next symbol. After the last symbol, there is no next bit so the bit stream “01100111” finished decoded into “ADAE”.

This search is better implemented recursively because we will always find the goal in the branch we are traversing and do not need the other branch.

Notice that we don't record anything from the bit stream we simply traversing it which is different with the second method.

B. Lookup Table

The idea of the second method is to provide a lookup table which contains all the symbols and the bit assigned to that symbol. The process is to traverse only the bit stream not including the Huffman tree.

First we make a lookup table that contains all symbol with their respective bit assignment. The table basically is Table 5. Next we traverse the bit stream and traversing the bit stream will net us '0' as the first bit. We record the bit and then search the table whether there is or not a symbol assigned by the bit '0'. We found 'A' is assigned by '0' in the table so we write the symbol 'A' and can search for the next symbol.

The difference between the first method we only traversing to the next bit and don't come back to the tree's root. The next bit we net is '1', we record it and search the table. There is no symbol assigned by '1' so we traverse the next bit. Again we find '1', we record it which made our current bit stream become “11”. Do the second search with “11” will also net us with nothing. Traversing the next bit our current bit stream become “110”. Searching that will get us the 'D' symbol. We write the symbol and can start to search for the next symbol. Repeating the process will make us decode “01100111” bit stream into “ADAE”. Notice that the second difference between the first method is we record the bit we currently search the symbol for.

Modifying the second method, we could also save the bit assigned to a symbol length. We then can use that information to modify the traversal. Traversing our bit stream again will get us the first '0' and the first 'A'. The next bit will be '1'. Notice that all bit assigned to a symbol beside the symbol 'A' have the length 3. So, when we get a '1' after we found a symbol, we traverse the stream three times while recording each bit. After that, we search the table to find the symbol. Back to the bit stream we will get “110” and can then search the table to find the symbol 'D'. Repeating the process, once again we decode the “01100111” bit stream into “ADAE”.

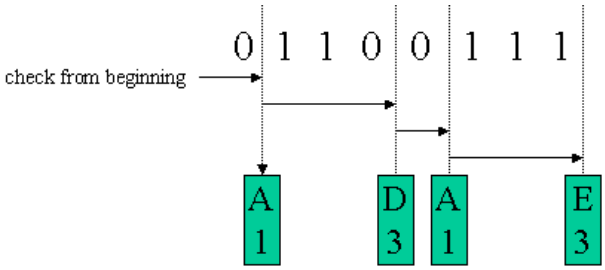

Figure 2 Huffman Decoding

Both code will have their use in different ways. Traversing the bit stream and Huffman tree is more complex than traversing only the bit stream. Searching a table with too many data is also troublesome. In the end what we want to decode that matters. Whether the information is long or not and whether the information is being build by many data or not.

VII. CONCLUSION

The conclusion of the explanation:

1. Huffman encoding uses greedy algorithm by minimum frequency;
2. Huffman decoding uses BFS implemented in recursive;
3. BFS in Huffman decoding only doing recursive to one of the child node from two child nodes.

REFERENCES

[1] http://en.wikipedia.org/wiki/Data retrieved at 8th December 12:17

[2] http://en.wikipedia.org/wiki/ASCII retrieved at 8th December 12:19

[3] http://en.wikipedia.org/wiki/Character_encoding retrieved at 8th December 12:20

[4] http://en.wikipedia.org/wiki/Redundancy_(information_theory) retrieved at 8th December 12:26

[5] http://en.wikipedia.org/wiki/Greedy_algorithm retrieved at 8th December 14:24

[6] Rinaldi Munir, *Diktat Kuliah IF3051 Strategi Algoritma*, Bandung: Teknik Informatika ITB, 2009, pp 36-31, 108-113.

[7] Raydhitya Yoseph, “Huffman Coding For Your Digital Music Hearing Pleasure”, unpublished, 2009.

[8] http://en.wikipedia.org/wiki/Depth-first_search retrieved at 8th December 22:51

[9] http://manoftoday.wordpress.com/2007/11/12/algorithm/ retrieved at 8th December 22:54

[10] http://en.wikipedia.org/wiki/Breadth-first_search retrieved at 8th December 23:11

DECLARATION

I hereby declare the paper is my own writing, not an adaptation, onr translation from another person paper, and not a form of plagiarism.

Bandung, 9th December 2011

Raydhitya Yoseph 13509092