COP 5536 Advanced Data Structures

Assignment #1

(Programming Project)

REPORT

Implementation of Huffman Encoder and Decoder

Submitted by: Vasu Raj Jain

UFID: 9219-1234

UF Email: vasujain@ufl.edu

Working Environment:

Operating System: Windows

Compiler: g++ Compiler

Integrated Environment: Dev C++ providing a GUI for the compiler and the execution shell.

Function prototype showing the structure of the program :-

Here i will list the functions which i used in my code.For complete documentation, read the comments in the code.

Classes Used in encoder.cpp

- **Class: MinNode**

Data Members: String data , int freq , Pointer to Node left and right.
Public functions :
MinNode(string data, int freq):
(A Constructor to initialize the MinNode Object.)

MinNode* getChild(int child):
(A function which returns the child according to index)

int isLeaf():
(A function which checks if a node is leaf or not)

int getFreq():
(A function which returns the frequency of the node)

void set_l_r(MinNode *l, MinNode* r):
(function which sets the left right pointer of the node)

bool isChild(int child):
(A function which checks whether a node is child or not and return the boolean value)

string getData():
(A function which returns the data field of the node)

- **Class: DHeap :**

Data Members: int d , int currentSize , int size , Pointer of Pointer of array.

Public functions :

DHeap(int capacity)
(A Constructor to initialize the 4way Heap of given capacity.)

bool isEmpty():
(A function which checks if a 4way heap is empty or not. In this case 4way cache optimised heap)

bool isFull():
(A function which checks if a 4 way cache optimised heap is full or not)

void makeEmpty():
(A function which makes the 4way heap empty by changing its size)

int parent(int i):
(A function which return the parent index of 4 way cache optimised heap)

int kthChild(int i, int k):
(A function which returns the index of kth child of ith node of 4 way heap)

void insert(MinNode* temp):
(A function to insert element in the 4 way cache optimised heap)

MinNode* findMin():
(A function which returns the min node of 4 way heap)

MinNode* deleteMin():
(A function which deletes the min element of 4way heap i.e root)

void buildHeap():
(A function which builds the heap using 4way heap algorithm)

void pDown(int hole):
(A function to satisfy the heap property when element is deleted so that to move the element to its actual position)

int smallestChild(int hole):
(A function which returns the smallest child node of the ith index node)

void pUp(int hole):
(A function which satisfy the heap property after inserting the element in the bottom of the heap.)

void printHeap()
(A function to print the heap)

void set_array(int i, string data, int freq):
(A function which sets the value in the node of ith index with the values passed in arguments)

void set_size(int size):
(A function to set the size of the heap)

int isSizeOne():
(A function which check the size of the heap if it's one or not in cache optimised case if it's equal to the root index i.e 4 or not)

Function to build Huffman and Encoding :

- **DHeap* CreateBuildFourWayHeap(map<string, int> m) :**

(This function takes the argument as map and stores the value pair from the frequency table)

- **MinNode* buildHuffmanTreeVasu(map<string, int> m) :**

(This function selects the two node with minimum frequency and merge them and make left right pointers and add the node back to the Heap structure)

- **void printingCode(MinNode* root, int arr[], int top) :**

(This function generates the code by traversing the heap on the fact that left traverse will give '0' code and right traverse will give '1' code)

- **void GenerateCodeTable() :**

(This function generates the code table and writes it in the file name code_table.txt)

- **void GenerateEncodedData(vector<string> v) :**

(This function generates the encoded.bin file)

- **void HuffmanCodesVasu(map<string, int> m) :**

(This utility function calls the other functions define and calls above mentioned function inside this call)

- **struct datafrequency read_file(char *file) :**

(This function reads the data and its frequency from the file and store it in the map)

Classes Used in decoder.cpp

- **Class: MinNode**

Public :

Data Members: String data , Pointer to Node left and right.

MinNode(string data, int freq):

(A Constructor to initialize the MinNode Object with NULL values)

Independent function used in Decoding the encoded.bin :

- **vector<int> FileReadDecode(string filename) :**
  (This function reads the code_table from the file)

- **MinNode *buildHuffmanVasu(string filename) :**

(This function reconstruct the huffman tree according the code_table which we read from file and place the data part in leaf nodes)

- **void DecodeVasu(MinNode* root, vector<int> v) :**

(This function decodes the tree and return the leaf element according to the encoded.bin)

Performance analysis :

In performance analysis in the initial part i have implemented the huffman tree using three different internal data structure to find the best data structure to generate the huffman encoded code.

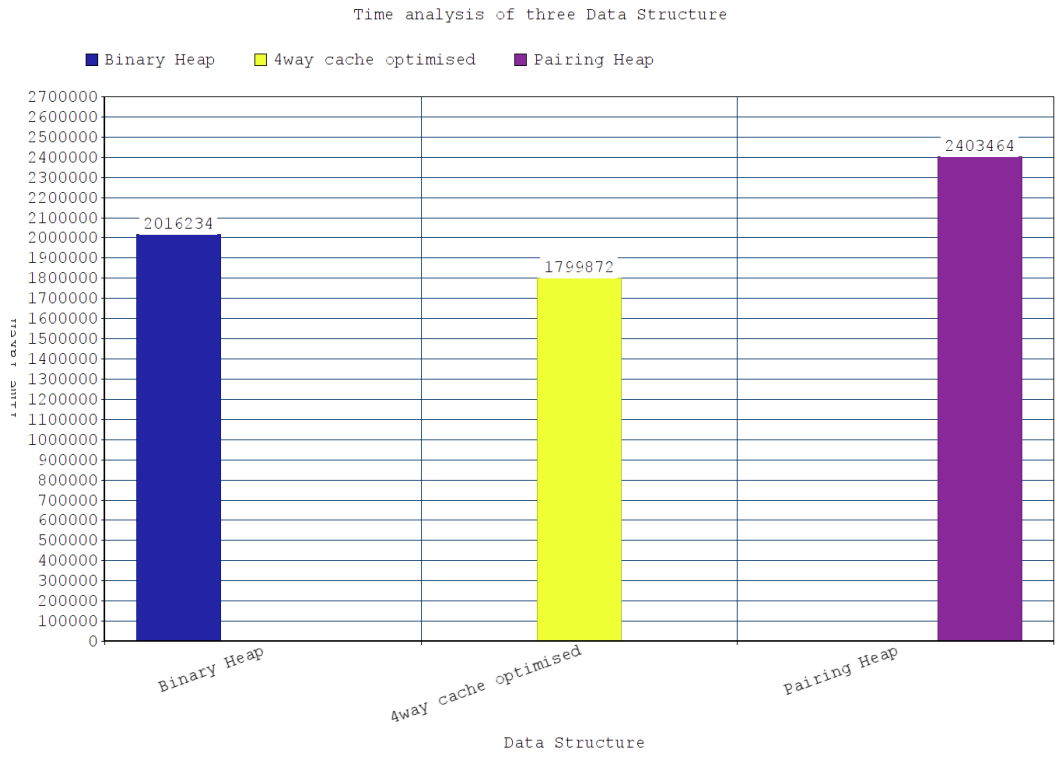
In the analysis i found that best data structure after comparing time taken by all three DS (Binary heap, 4-way Cache optimised,Pairing heap).

The minimum time was taken by the 4-way Cache optimised Heap followed by Binary heap and then Pairing heap.

Time analysis was done using the sample_input_large.txt file and average of 10 runs were taken.

```cpp
int main() {
    string filename = "sample2/sample_input_large.txt";

    vector<int> freq_table = parseFile(filename);

    // binary heap
    start_time = clock();
    for(int i = 0; i < 10; i++) {
        //run 10 times on given data set
        build_tree_using_binary_heap(freq_table);
    }
    cout << "Average time using binary heap (microsecond):  " << (clock() - start_time)/10 << endl;

    // 4-way heap
    start_time = clock();
    for(int i = 0; i < 10; i++) { //run 10 times on given data set
        build_tree_using_4way_heap(freq_table);
    }
    cout << "Average time using 4-way heap (microsecond):  " << (clock() - start_time)/10 << endl;

    // pairing heap
    start_time = clock();
    for(int i = 0; i < 10; i++) { //run 10 times on given data set
        build_tree_using_pairing_heap(freq_table);
    }
    cout << "Average time using pairing heap (microsecond):  " << (clock() - start_time)/10 << endl;

    return 0;
}
```

```
Average time using binary heap (microsecond): 2016234
Average time using 4 way (microsecond): 1799872
Average time using pair heap (microsecond): 2403464
Press any key to continue . . .
```



Decoding Algorithm and its complexity :

Here in my implementation of huffman decoding i am generating the decoded text file from the code_table.txt using the deep first search algorithm.

Decoding Algorithm : DFS (Deep First Search)
Complexity of Decoding Algorithm : O(N) where N is the number of nodes in the Huffman tree to be traversed.

Deep first search algorithm starts from the root and explores each branch until the goal node is reached.
Here in my decoding algorithm we traverse the tree from the root and return the symbol which we found during traversal and starts again from the root. this is little bit modified version of the deep first search.
The logic behind decoding is to traverse the huffman tree bit by bit and return the symbol found.