Contents

1 Data Structure bdnode

For the assignment the "bdnode" data structure is used.

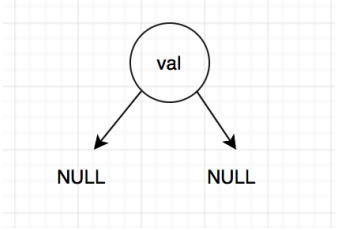
```
struct bdnode {
  std::string val;
  bdnode *left;
  bdnode *right;

};

typedef bdnode *bdt;
```

It consists of a std::string "val" variable and two pointers to other "bdnode" structures.

For the creation of this data structure thougout the code the "newNode" function is called, which fills the "val" field with the value string and creates two pointers left and right pointing to NULL.

```
bdt newNode(std::string value) {
    bdt temp = new bdnode;
    temp->val = value;
    temp->left = NULL;
    temp->right = NULL;
    return temp;
}
```

Figure 1: Bdnode Structure



2 Build The Tree

The function's goal is to build a simplified binary tree of a boolean function, given the inputs for which the function is 1.
My implementation is inspired by the "Divide and conquer" algorithm.
It uses a vector of all the leaf values of the original full tree. Then recursively checks for possible simplifications and creates the tree.

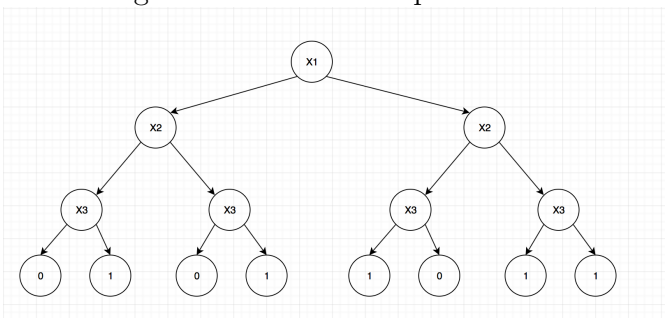
2.1 Buildcompactbdt Function

To understand better the algorithm, an example will be used along the way.

Let the input vector fvalues consists of the strings :

"001", "011", "100","110","111"
These values create the unsimplified binary tree in Figure 2.

Figure 2: Initial Unsimplified Tree



The first thing the function does, is to create the vector without any simplifications yet. It calculates the maximum depth that the tree will reach which is equal to the length of any input string. In our case 3 (e.g. "001").
Then calculates the number of leaves the final tree will have which is pow(2,max depth). In our example, the tree will have 8 leaves.
It creates the bool vector "vectorLeafValues" with size equal to the number of the

leaves and it fills the "vectorLeafValues" with zeros.

Figure 3: Initial vectorValues Vector

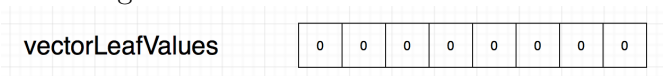
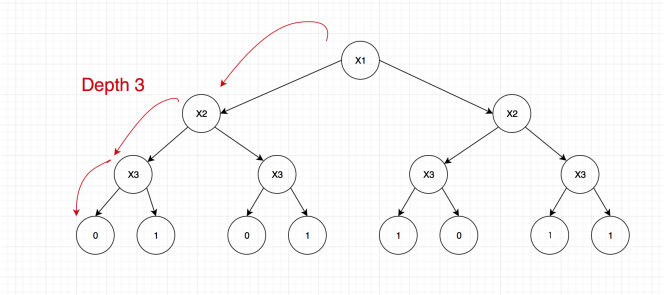


Figure 4: Depth in the Unsimplified Tree



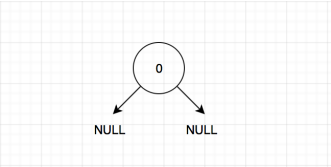
The last thing the function does is to call the following two functions before it returns the pointer to the root of the simplified tree.

```
setVectorLeafValuesTo1;

bdt root = reduceTree(vectorLeafValues,maxDepth);
```

The functions will be explained in the the next subsections. But before that, it worth mentioning a special case that the "buildcompactbdt" function handles.
Up until this point, the function operates on data from the input vector to create the necessary vectors. But what happens if the input vector is empty ?
When the input vector is empty, that means the function is always zero. The following condition in the beginning of the function creates a node with the value "0" and return a pointer to that node.

```
if (fvalues.size()==0){
    return newNode("0");
}
```

Figure 5: Simplified Tree for empty input vector



2.2 setVectorLeafValuesTo1 Function

As the name indicates this function will set the correct values of vectorLeafValues to logical 1. It takes a reference to the input vector fvalues and a reference to the vectorLeafValues. It converts each element of the input vector from string binary to an integer with the following command:

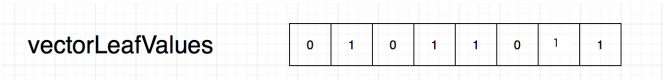
```
index_set1_leaf = std::stoi(input[i], nullptr, 2);
vectorLeafValues[index_set1_leaf] = 1;
```

and sets the corresponding elements of the vectorLeafValues to logical 1 (True).

In our example:

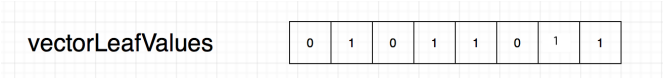
fvalues = [”001”, ”011”, ”100”,”111”] so it will set the 2nd,4th,5th and 8th elements to 1. (In c++ the first element is indexed 0.)

Figure 6: vectorLeafValues after setVectorLeafValuesTo1



2.3 reduceTree Function

Figure 7: Vectors before Reduce



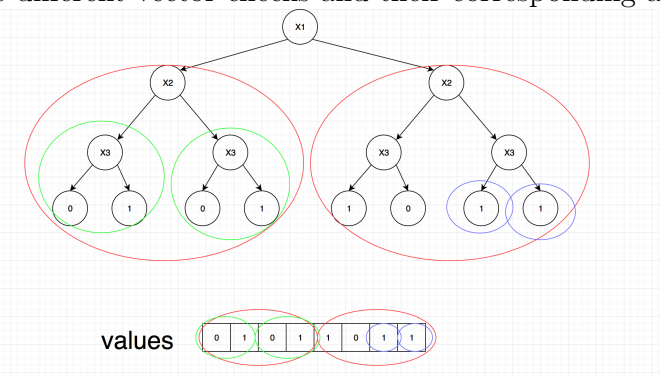
The "reduceTree" function has the following arguments:

1)A reference to the values vectors which initially is the vectorLeafValues vector.

2)An integer "maxIndex" which is the maxDepth the orginal tree could reach.

The main idea behind the "reduceTree" function is to divide and check vectors. The function finds the middle of the "values" vector, "midValue". It divides the "values" vector at this "midValue" and checks if the two halves are equal. This is equivalent as to check if the subtrees of an X node are the same.

Figure 8: Three different vector checks and their corresponding areas in the tree



The first thing the function does is to check if "midValue" is zero. This is equivalent as to check if it is checking only a leaf.
If it is zero (it has reached a leaf value), it creates a bdt pointer filling the "val" field with the first element of the current values (values[0]).

If it is not zero it means that it is currently checking subtrees to an X node. It splits the vector in the middle and checks if the two halves are equal. There are two cases here:

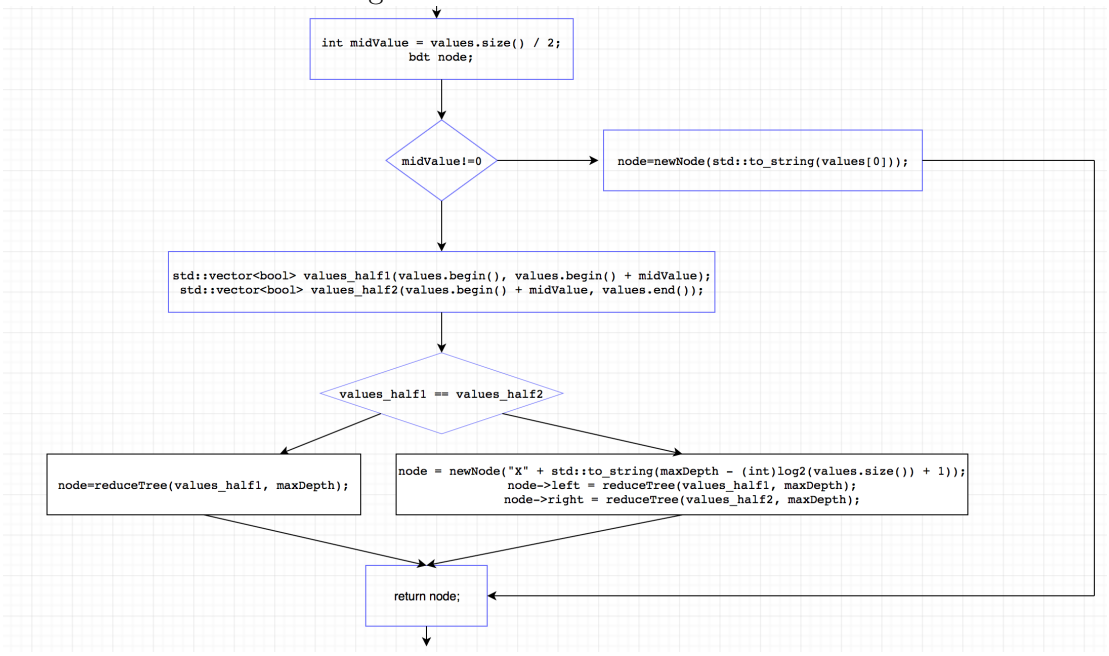
1)If the two halves are not equal then it creates a new node with the right X node "val" field. While it calls the "reduceTree" function twice now, once to assign the returned bdt pointer to the left of that node and once to the right of that node. The following command is used to create the right X node.

```
node = newNode("X" + std::to_string(maxDepth - (int)log2(values.size()) + 1));
```

2)If they are equal, it calls the "reduceTree" function again but passing this time the first half of the current value vector. Effectively, this step skips creating the X node as in case 1 hence it simplifies the tree.

Since the function is called recursively and every time it returns the "node" pointer, it recursively builds the simplified tree.

Figure 9: reduceTree Flow chart



The figures below demonstrate the "reduceTree" function for our example step by step. The blue circle demonstrates the parts of the original "values" vector that the function currently checks.

Figure 10: reduceTree step 1 Example

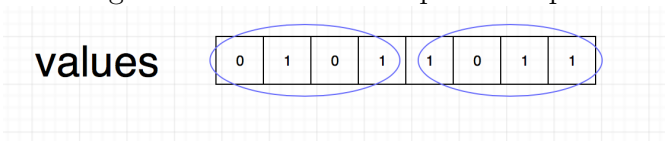


Figure 11: reduceTree step 2 Example

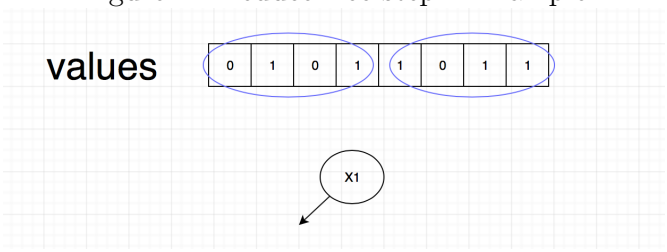


Figure 12: reduceTree step 3 Example

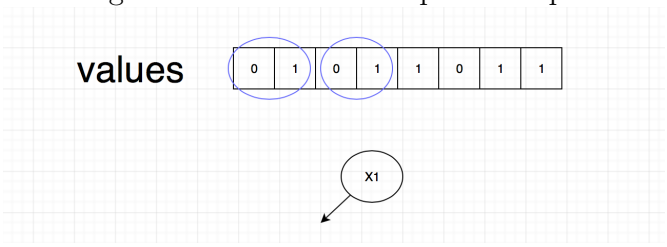


Figure 13: reduceTree step 4 Example

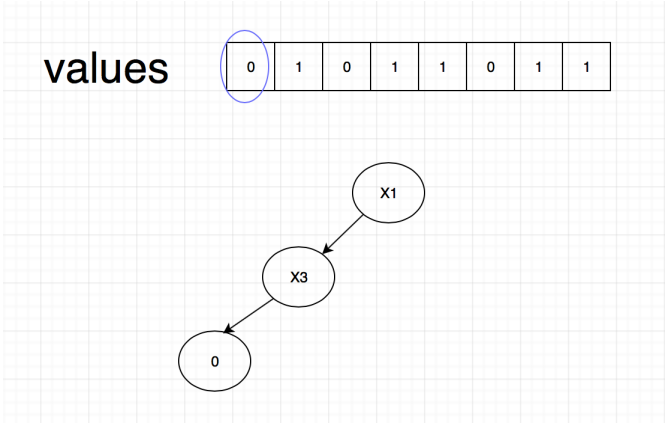


Figure 14: reduceTree step 5 Example

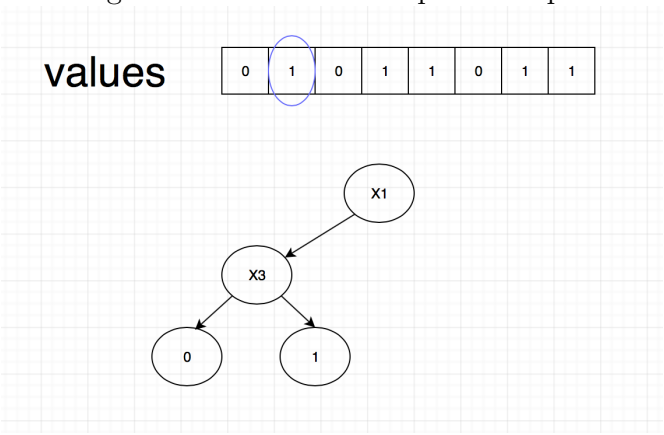


Figure 15: reduceTree step 6 Example

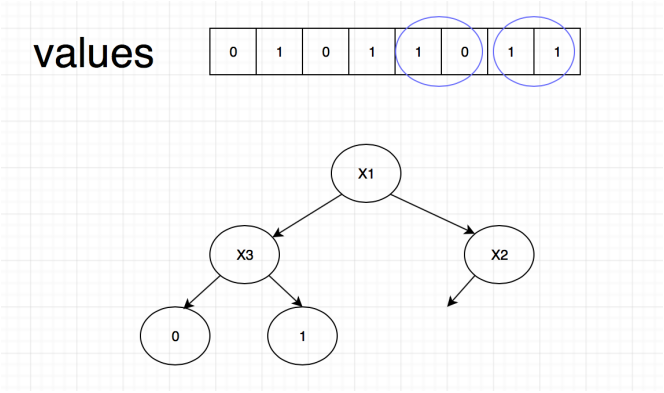


Figure 16: reduceTree step 7 Example

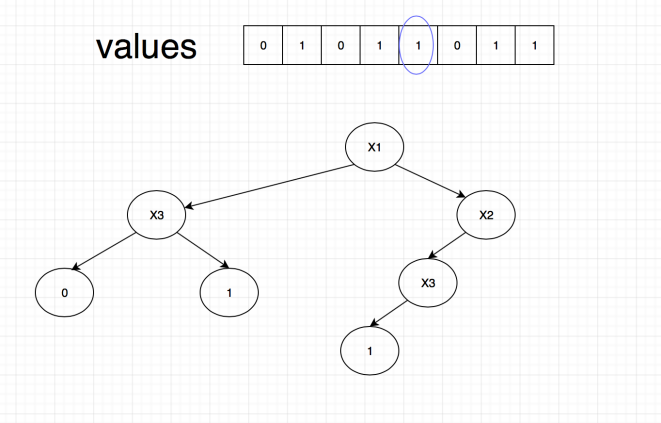


Figure 17: reduceTree step 8 Example

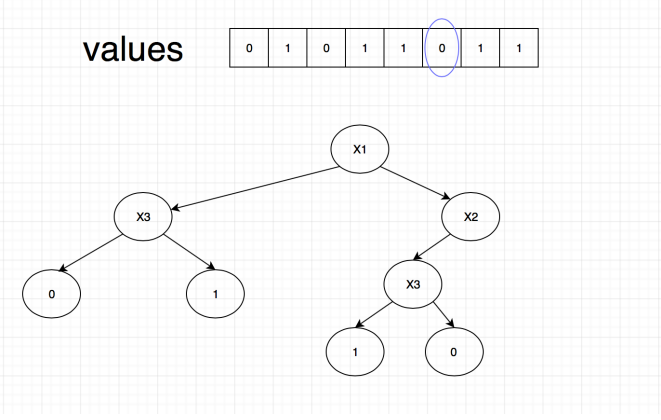


Figure 18: reduceTree step 9 Example

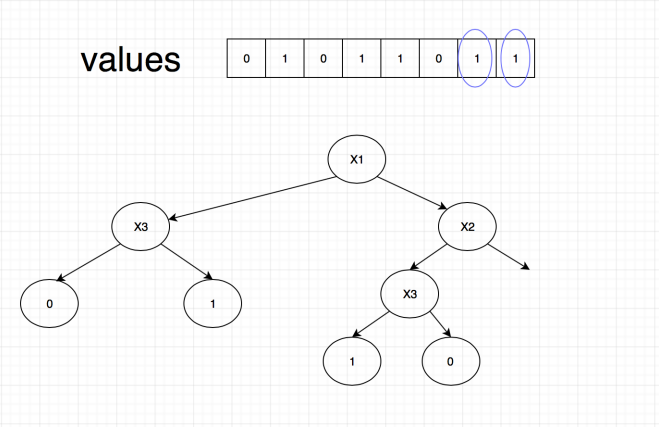
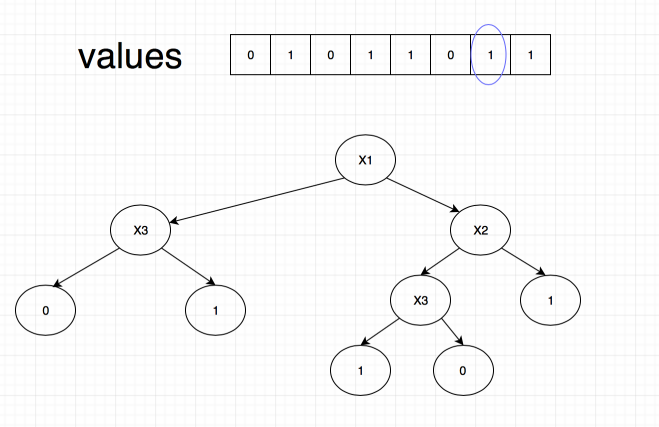


Figure 19: reduceTree step 10 Example



3 Evaluate From the Tree

For the evaluation of a value given a string input, two functions are used:

1)

```
std::string evalbdt(bdt t, const std::string &input)
```

2)

```
bdt movePointer(bdt t, char c)
```

3.1 Evalbdt Function

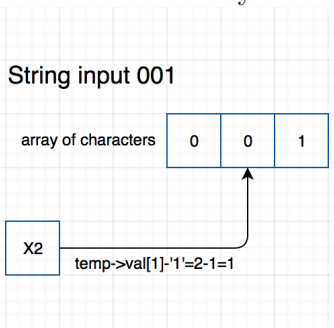
The function turns the input string into an array of characters. Then using the pointer to a node (initially to the root) the function checks if the "val" field is 0,1 or X node. If it is 1 or 0, it returns it to the main function. If it is X node (e.g. X2), it passes the corresponding element of the array of characters along with the pointer to the "movePointer" in order to move the pointer correctly.

```
int arrayCharIndex = std::stoi(temp->val.substr(1), nullptr,10) - 1;
temp = movePointer(temp, array_of_characters[arrayCharIndex]);
```

How does this work ?

The temp-¿val.substr(1) is the characters in the X node string val after X. For example, in X2 temp-¿val.substr(1) is '2'. By turning it into an integer and subtracting 1 , the "arrayCharIndex" points to the 1 which corresponds to X2 (since in C++ first elements of arrays has index 0).

Figure 20: X2 in the array of characters

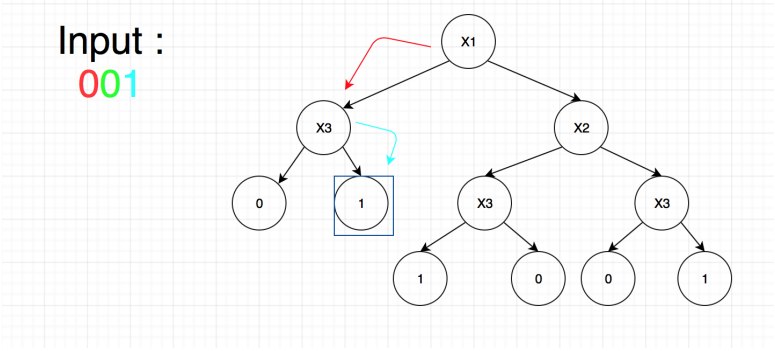


3.2 movePointer Function

The "movePointer" function is simple. Depending on the c value, it moves the pointer either left (c is 0) or right (c is 1).

```
bdt movePointer(bdt t, char c) {
    if (c == '0') {
        return t->left;
    } else {
        return t->right;
    }
}
```

Figure 21: Example for evaluation input="001" and output 1



4 Tests

To test the build functionality of the code, the three following print functions were used to print the three vectors and the final tree.

```
  void PrintTree(bdt root) {
    if (root == NULL) {
        return;
    } else {
        PrintTree(root->left);
        std::cout << root->val << " ";
        PrintTree(root->right);
    }
}
```

In addition to that, the execution time of the program was measured using c++11 Chrono library and the following commands in the main function:

```
int main(){
auto start = std::chrono::high_resolution_clock::now();

<MAIN FUNCTION'S BODY>

auto elapsed = std::chrono::high_resolution_clock::now() - start;
long long microseconds = std::chrono::duration_cast
   <std::chrono::microseconds>(elapsed).count();
std::cout << microseconds << std::endl;
return 0;
}
```

The tests, that were performed, consist of all possible input combinations for two X nodes X1 and X2 and all possible input combinations for three X nodes X1, X2 and X3. In addition below, there is one four Variable example with (Unsimplified: 0101010111001100) see Figures 22 and 23. The last example was used to enter all possible inputs to check Evaluation function. Since the first test has even number X nodes and the second odd, I believe to the best of my ability that the algorithm is functioning universally (See subsection "Correctness Tests Results" for the measured results).

Figure 22: 4 Variables Example Unsimplified

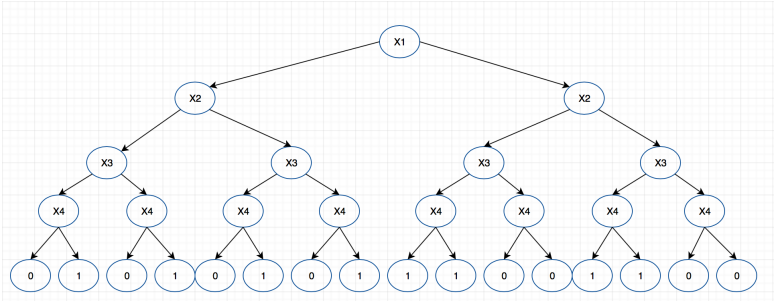
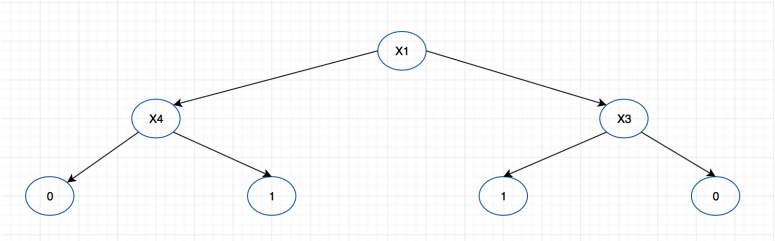


Figure 23: 4 Variables Example Simplified

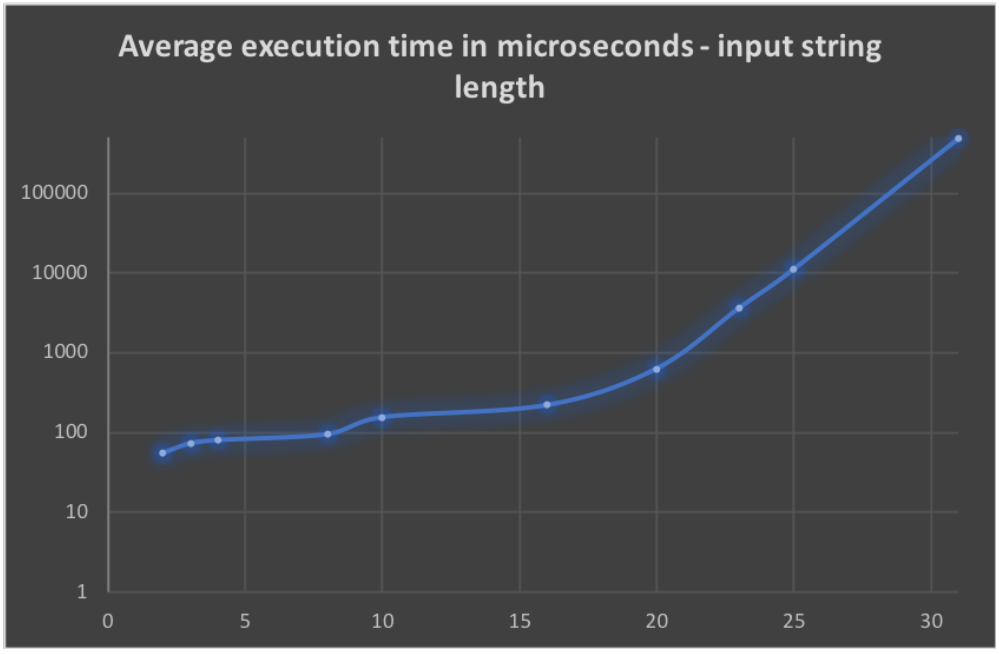


4.1 Implementation's Limitation

There is one limitation of my implementation. Inside the code, a integer is used to calculate the number of leaves and create a bool vector of that number. If the input string is long (more than 31 characters) the unsigned int variable in c++ can not handle it and this results in an exception error.

4.2 Speed Analysis

Figure 24: Speed Analysis



Different lengths of input string were tested.The execution time of different combinations was measured and averaged to provide a speed analysis graph. An increase is noticed as the length of the input string increases. But still the code runs even for 31 length in less than 1.3 seconds.

4.3 Correctness Tests Results

The correctness tests for the two functions (”buildcompactbdt” and ”evalcompactbdt”) are displayed below:

Evaluation Test

Four Variables

Unsimplified	Time (ms):	Outputs:
Example	80	0 X4 1 X1 1 X3 0

Eval - input	output
0	0
1	1
10	0
11	1
100	0
101	1
110	0
111	1
1000	1
1001	1
1010	0
1011	0
1100	1
1101	1
1110	0
1111	0

Build Tree Tests

Two Variables

Unsimplified	Time (ms):	Outputs:
0	22	0
1	55	0 X1 0 X2 1
10	54	0 X1 1 X2 0
11	52	0 X1 1
100	54	0 X2 1 X1 0
101	48	0 X2 1
110	69	0 X2 1 X1 1 X2 0
111	50	0 X2 1 X1 1
1000	60	1 X2 0 X1 0
1001	60	1 X2 0 X1 0 X2 1
1010	52	1 X2 0
1011	58	1 X2 0 X1 1
1100	50	1 X1 0
1101	55	1 X1 0 X2 1
1110	56	1 X1 1 X2 0
1111	42	1

Three Variables

Unsimplified	Time (ms):	Output:
0000 0000	22	0
0000 0001	70	0 X1 0 X2 0 X3 1
0000 0010	61	0 X1 0 X2 1 X3 0
0000 0011	65	0 X1 0 X2 1
0000 0100	73	0 X1 0 X3 1 X2 0
0000 0101	63	0 X1 0 X3 1
0000 0110	70	0 X1 0 X3 1 X2 1 X3 0
0000 0111	71	0 X1 0 X3 1 X2 1
0000 1000	71	0 X1 1 X3 0 X2 0
0000 1001	73	0 X1 1 X3 0 X2 0 X3 1
0000 1010	66	0 X1 1 X3 0
0000 1011	69	0 X1 1 X3 0 X2 1
0000 1100	62	0 X1 1 X2 0
0000 1101	71	0 X1 1 X2 0 X3 1
0000 1110	67	0 X1 1 X2 1 X3 0
0000 1111	61	0 X1 1
0001 0000	69	0 X2 0 X3 1 X1 0
0001 0001	64	0 X2 0 X3 1
0001 0010	73	0 X2 0 X3 1 X1 0 X2 1 X3 0
0001 0011	71	0 X2 0 X3 1 X1 0 X2 1
0001 0100	72	0 X2 0 X3 1 X1 0 X3 1 X2 0
0001 0101	73	0 X2 0 X3 1 X1 0 X3 1
0001 0110	72	0 X2 0 X3 1 X1 0 X3 1 X2 1 X3 0

0001 0111	70	0 X2 0 X3 1 X1 0 X3 1 X2 1
0001 1000	73	0 X2 0 X3 1 X1 1 X3 0 X2 0
0001 1001	69	0 X2 0 X3 1 X1 1 X3 0 X2 0 X3 1
0001 1010	69	0 X2 0 X3 1 X1 1 X3 0
0001 1011	75	0 X2 0 X3 1 X1 1 X3 0 X2 1
0001 1100	72	0 X2 0 X3 1 X1 1 X2 0
0001 1101	75	0 X2 0 X3 1 X1 1 X2 0 X3 1
0001 1110	71	0 X2 0 X3 1 X1 1 X2 1 X3 0
0001 1111	68	0 X2 0 X3 1 X1 1
0010 0000	64	0 X2 1 X3 0 X1 0
0010 0001	70	0 X2 1 X3 0 X1 0 X2 0 X3 1
0010 0010	63	0 X2 1 X3 0
0010 0011	73	0 X2 1 X3 0 X1 0 X2 1
0010 0100	77	0 X2 1 X3 0 X1 0 X3 1 X2 0
0010 0101	70	0 X2 1 X3 0 X1 0 X3 1
0010 0110	73	0 X2 1 X3 0 X1 0 X3 1 X2 1 X3 0
0010 0111	74	0 X2 1 X3 0 X1 0 X3 1 X2 1
0010 1000	70	0 X2 1 X3 0 X1 1 X3 0 X2 0
0010 1001	72	0 X2 1 X3 0 X1 1 X3 0 X2 0 X3 1
0010 1010	68	0 X2 1 X3 0 X1 1 X3 0
0010 1011	77	0 X2 1 X3 0 X1 1 X3 0 X2 1
0010 1100	74	0 X2 1 X3 0 X1 1 X2 0
0010 1101	73	0 X2 1 X3 0 X1 1 X2 0 X3 1
0010 1110	76	0 X2 1 X3 0 X1 1 X2 1 X3 0
0010 1111	77	0 X2 1 X3 0 X1 1
0011 0000	65	0 X2 1 X1 0
0011 0001	72	0 X2 1 X1 0 X2 0 X3 1
0011 0010	71	0 X2 1 X1 0 X2 1 X3 0
0011 0011	63	0 X2 1
0011 0100	70	0 X2 1 X1 0 X3 1 X2 0
0011 0101	67	0 X2 1 X1 0 X3 1
0011 0110	77	0 X2 1 X1 0 X3 1 X2 1 X3 0
0011 0111	75	0 X2 1 X1 0 X3 1 X2 1
0011 1000	67	0 X2 1 X1 1 X3 0 X2 0
0011 1001	72	0 X2 1 X1 1 X3 0 X2 0 X3 1
0011 1010	70	0 X2 1 X1 1 X3 0
0011 1011	73	0 X2 1 X1 1 X3 0 X2 1
0011 1100	60	0 X2 1 X1 1 X2 0
0011 1101	77	0 X2 1 X1 1 X2 0 X3 1
0011 1110	78	0 X2 1 X1 1 X2 1 X3 0
0011 1111	63	0 X2 1 X1 1
0100 0000	69	0 X3 1 X2 0 X1 0
0100 0001	73	0 X3 1 X2 0 X1 0 X2 0 X3 1
0100 0010	78	0 X3 1 X2 0 X1 0 X2 1 X3 0
0100 0011	77	0 X3 1 X2 0 X1 0 X2 1

0100 0100	63	0 X3 1 X2 0
0100 0101	78	0 X3 1 X2 0 X1 0 X3 1
0100 0110	72	0 X3 1 X2 0 X1 0 X3 1 X2 1 X3 0
0100 0111	68	0 X3 1 X2 0 X1 0 X3 1 X2 1
0100 1000	69	0 X3 1 X2 0 X1 1 X3 0 X2 0
0100 1001	77	0 X3 1 X2 0 X1 1 X3 0 X2 0 X3 1
0100 1010	74	0 X3 1 X2 0 X1 1 X3 0
0100 1011	69	0 X3 1 X2 0 X1 1 X3 0 X2 1
0100 1100	71	0 X3 1 X2 0 X1 1 X2 0
0100 1101	67	0 X3 1 X2 0 X1 1 X2 0 X3 1
0100 1110	73	0 X3 1 X2 0 X1 1 X2 1 X3 0
0100 1111	63	0 X3 1 X2 0 X1 1
0101 0000	57	0 X3 1 X1 0
0101 0001	66	0 X3 1 X1 0 X2 0 X3 1
0101 0010	78	0 X3 1 X1 0 X2 1 X3 0
0101 0011	59	0 X3 1 X1 0 X2 1
0101 0100	67	0 X3 1 X1 0 X3 1 X2 0
0101 0101	70	0 X3 1
0101 0110	69	0 X3 1 X1 0 X3 1 X2 1 X3 0
0101 0111	66	0 X3 1 X1 0 X3 1 X2 1
0101 1000	66	0 X3 1 X1 1 X3 0 X2 0
0101 1001	67	0 X3 1 X1 1 X3 0 X2 0 X3 1
0101 1010	50	0 X3 1 X1 1 X3 0
0101 1011	71	0 X3 1 X1 1 X3 0 X2 1
0101 1100	63	0 X3 1 X1 1 X2 0
0101 1101	73	0 X3 1 X1 1 X2 0 X3 1
0101 1110	73	0 X3 1 X1 1 X2 1 X3 0
0101 1111	60	0 X3 1 X1 1
0110 0000	75	0 X3 1 X2 1 X3 0 X1 0
0110 0001	74	0 X3 1 X2 1 X3 0 X1 0 X2 0 X3 1
0110 0010	75	0 X3 1 X2 1 X3 0 X1 0 X2 1 X3 0
0101 0011	68	0 X3 1 X2 1 X3 0 X1 0 X2 1
0110 0100	73	0 X3 1 X2 1 X3 0 X1 0 X3 1 X2 0
0110 0101	68	0 X3 1 X2 1 X3 0 X1 0 X3 1
0110 0110	54	0 X3 1 X2 1 X3 0
0110 0111	76	0 X3 1 X2 1 X3 0 X1 0 X3 1 X2 1
0110 1000	74	0 X3 1 X2 1 X3 0 X1 1 X3 0 X2 0
0110 1001	72	0 X3 1 X2 1 X3 0 X1 1 X3 0 X2 0 X3 1
0110 1010	72	0 X3 1 X2 1 X3 0 X1 1 X3 0
0110 1011	70	0 X3 1 X2 1 X3 0 X1 1 X3 0 X2 1
0110 1100	78	0 X3 1 X2 1 X3 0 X1 1 X2 0
0110 1101	67	0 X3 1 X2 1 X3 0 X1 1 X2 0 X3 1
0110 1110	75	0 X3 1 X2 1 X3 0 X1 1 X2 1 X3 0
0110 1111	70	0 X3 1 X2 1 X3 0 X1 1
	69	
0111 0000	65	0 X3 1 X2 1 X1 0

0111 0001	69	0 X3 1 X2 1 X1 0 X2 0 X3 1
0111 0010	67	0 X3 1 X2 1 X1 0 X2 1 X3 0
0111 0011	68	0 X3 1 X2 1 X1 0 X2 1
0111 0100	69	0 X3 1 X2 1 X1 0 X3 1 X2 0
0111 0101	70	0 X3 1 X2 1 X1 0 X3 1
0111 0110	70	0 X3 1 X2 1 X1 0 X3 1 X2 1 X3 0
0111 0111	64	0 X3 1 X2 1
0111 1000	66	0 X3 1 X2 1 X1 1 X3 0 X2 0
0111 1001	72	0 X3 1 X2 1 X1 1 X3 0 X2 0 X3 1
0111 1010	68	0 X3 1 X2 1 X1 1 X3 0
0111 1011	74	0 X3 1 X2 1 X1 1 X3 0 X2 1
0111 1100	67	0 X3 1 X2 1 X1 1 X2 0
0111 1101	73	0 X3 1 X2 1 X1 1 X2 0 X3 1
0111 1110	71	0 X3 1 X2 1 X1 1 X2 1 X3 0
0111 1111	66	0 X3 1 X2 1 X1 1
1000 0000	63	1 X3 0 X2 0 X1 0
1000 0001	69	1 X3 0 X2 0 X1 0 X2 0 X3 1
1000 0010	72	1 X3 0 X2 0 X1 0 X2 1 X3 0
1000 0011	71	1 X3 0 X2 0 X1 0 X2 1
1000 0100	69	1 X3 0 X2 0 X1 0 X3 1 X2 0
1000 0101	68	1 X3 0 X2 0 X1 0 X3 1
1000 0110	73	1 X3 0 X2 0 X1 0 X3 1 X2 1 X3 0
1000 0111	79	1 X3 0 X2 0 X1 0 X3 1 X2 1
1000 1000	63	1 X3 0 X2 0
1000 1001	73	1 X3 0 X2 0 X1 1 X3 0 X2 0 X3 1
1000 1010	72	1 X3 0 X2 0 X1 1 X3 0
1000 1011	68	1 X3 0 X2 0 X1 1 X3 0 X2 1
1000 1100	72	1 X3 0 X2 0 X1 1 X2 0
1000 1101	75	1 X3 0 X2 0 X1 1 X2 0 X3 1
1000 1110	68	1 X3 0 X2 0 X1 1 X2 1 X3 0
1000 1111	74	1 X3 0 X2 0 X1 1
1001 0000	79	1 X3 0 X2 0 X3 1 X1 0
1001 0001	75	1 X3 0 X2 0 X3 1 X1 0 X2 0 X3 1
1001 0010	75	1 X3 0 X2 0 X3 1 X1 0 X2 1 X3 0
1001 0011	67	1 X3 0 X2 0 X3 1 X1 0 X2 1
1001 0100	72	1 X3 0 X2 0 X3 1 X1 0 X3 1 X2 0
1001 0101	66	1 X3 0 X2 0 X3 1 X1 0 X3 1
1001 0110	76	1 X3 0 X2 0 X3 1 X1 0 X3 1 X2 1 X3 0
1001 0111	80	1 X3 0 X2 0 X3 1 X1 0 X3 1 X2 1
1001 1000	74	1 X3 0 X2 0 X3 1 X1 1 X3 0 X2 0
1001 1001	52	1 X3 0 X2 0 X3 1
1001 1010	78	1 X3 0 X2 0 X3 1 X1 1 X3 0
1001 1011	79	1 X3 0 X2 0 X3 1 X1 1 X3 0 X2 1
1001 1100	71	1 X3 0 X2 0 X3 1 X1 1 X2 0
1001 1101	74	1 X3 0 X2 0 X3 1 X1 1 X2 0 X3 1
1001 1110	74	1 X3 0 X2 0 X3 1 X1 1 X2 1 X3 0

1001 1111	75	1 X3 0 X2 0 X3 1 X1 1
1010 0000	55	1 X3 0 X1 1 X2 0 X3 1
1010 0001	65	1 X3 0 X1 0 X2 0 X3 1
1010 0010	68	1 X3 0 X1 0 X2 1 X3 0
1010 0011	62	1 X3 0 X1 0 X2 1
1010 0100	72	1 X3 0 X1 0 X3 1 X2 0
1010 0101	54	1 X3 0 X1 0 X3 1
1010 0110	71	1 X3 0 X1 0 X3 1 X2 1 X3 0
1010 0111	74	1 X3 0 X1 0 X3 1 X2 1
1010 1000	69	1 X3 0 X1 1 X3 0 X2 0
1010 1001	68	1 X3 0 X1 1 X3 0 X2 0 X3 1
1010 1010	69	1 X3 0
1010 1011	76	1 X3 0 X1 1 X3 0 X2 1
1010 1100	65	1 X3 0 X1 1 X2 0
1010 1101	70	1 X3 0 X1 1 X2 0 X3 1
1010 1110	68	1 X3 0 X1 1 X2 1 X3 0
1010 1111	58	1 X3 0 X1 1
1011 0000	72	1 X3 0 X2 1 X1 0
1011 0001	70	1 X3 0 X2 1 X1 0 X2 0 X3 1
1011 0010	68	1 X3 0 X2 1 X1 0 X2 1 X3 0
1011 0011	69	1 X3 0 X2 1 X1 0 X2 1
1011 0100	69	1 X3 0 X2 1 X1 0 X3 1 X2 0
1011 0101	68	1 X3 0 X2 1 X1 0 X3 1
1011 0110	71	1 X3 0 X2 1 X1 0 X3 1 X2 1 X3 0
1011 0111	76	1 X3 0 X2 1 X1 0 X3 1 X2 1
1011 1000	68	1 X3 0 X2 1 X1 1 X3 0 X2 0
1011 1001	72	1 X3 0 X2 1 X1 1 X3 0 X2 0 X3 1
1011 1010	68	1 X3 0 X2 1 X1 1 X3 0
1011 1011	58	1 X3 0 X2 1
1011 1100	68	1 X3 0 X2 1 X1 1 X2 0
1011 1101	70	1 X3 0 X2 1 X1 1 X2 0 X3 1
1011 1110	73	1 X3 0 X2 1 X1 1 X2 1 X3 0
1011 1111	74	1 X3 0 X2 1 X1 1
1100 0000	58	1 X2 0 X1 0
1100 0001	70	1 X2 0 X1 0 X2 0 X3 1
1100 0010	71	1 X2 0 X1 0 X2 1 X3 0
1100 0011	67	1 X2 0 X1 0 X2 1
1100 0100	78	1 X2 0 X1 0 X3 1 X2 0
1100 0101	71	1 X2 0 X1 0 X3 1
1100 0110	67	1 X2 0 X1 0 X3 1 X2 1 X3 0
1100 0111	65	1 X2 0 X1 0 X3 1 X2 1
1100 1000	78	1 X2 0 X1 1 X3 0 X2 0
1100 1001	73	1 X2 0 X1 1 X3 0 X2 0 X3 1
1100 1010	66	1 X2 0 X1 1 X3 0
1100 1011	68	1 X2 0 X1 1 X3 0 X2 1

1100 1100	48	1 X2 0
1100 1101	76	1 X2 0 X1 1 X2 0 X3 1
1100 1110	69	1 X2 0 X1 1 X2 1 X3 0
1100 1111	61	1 X2 0 X1 1
1101 0000	66	1 X2 0 X3 1 X1 0
1101 0001	68	1 X2 0 X3 1 X1 0 X2 0 X3 1
1101 0010	69	1 X2 0 X3 1 X1 0 X2 1 X3 0
1101 0011	66	1 X2 0 X3 1 X1 0 X2 1
1101 0100	68	1 X2 0 X3 1 X1 0 X3 1 X2 0
1101 0101	63	1 X2 0 X3 1 X1 0 X3 1
1101 0110	78	1 X2 0 X3 1 X1 0 X3 1 X2 1 X3 0
1101 0111	67	1 X2 0 X3 1 X1 0 X3 1 X2 1
1101 1000	60	1 X2 0 X3 1 X1 1 X3 0 X2 0
1101 1001	77	1 X2 0 X3 1 X1 1 X3 0 X2 0 X3 1
1101 1010	68	1 X2 0 X3 1 X1 1 X3 0
1101 1011	72	1 X2 0 X3 1 X1 1 X3 0 X2 1
1101 1100	70	1 X2 0 X3 1 X1 1 X2 0
1101 1101	59	1 X2 0 X3 1
1101 1110	69	1 X2 0 X3 1 X1 1 X2 1 X3 0
1101 1111	68	1 X2 0 X3 1 X1 1
1110 0000	73	1 X2 1 X3 0 X1 0
1110 0001	67	1 X2 1 X3 0 X1 0 X2 0 X3 1
1110 0010	65	1 X2 1 X3 0 X1 0 X2 1 X3 0
1110 0011	70	1 X2 1 X3 0 X1 0 X2 1
1110 0100	78	1 X2 1 X3 0 X1 0 X3 1 X2 0
1110 0101	68	1 X2 1 X3 0 X1 0 X3 1
1110 0110	74	1 X2 1 X3 0 X1 0 X3 1 X2 1 X3 0
1110 0111	72	1 X2 1 X3 0 X1 0 X3 1 X2 1
1110 1000	72	1 X2 1 X3 0 X1 1 X3 0 X2 0
1110 1001	69	1 X2 1 X3 0 X1 1 X3 0 X2 0 X3 1
1110 1010	65	1 X2 1 X3 0 X1 1 X3 0
1110 1011	69	1 X2 1 X3 0 X1 1 X3 0 X2 1
1110 1100	64	1 X2 1 X3 0 X1 1 X2 0
1110 1101	73	1 X2 1 X3 0 X1 1 X2 0 X3 1
1110 1110	55	1 X2 1 X3 0
1110 1111	63	1 X2 1 X3 0 X1 1
1111 0000	55	1 X1 0
1111 0001	79	1 X1 0 X2 0 X3 1
1111 0010	77	1 X1 0 X2 1 X3 0
1111 0011	60	1 X1 0 X2 1
1111 0100	68	1 X1 0 X3 1 X2 0
1111 0101	62	1 X1 0 X3 1
1111 0110	71	1 X1 0 X3 1 X2 1 X3 0
1111 0111	69	1 X1 0 X3 1 X2 1
1111 1000	66	1 X1 1 X3 0 X2 0

1111 1001	70	1 X1 1 X3 0 X2 0 X3 1
1111 1010	56	1 X1 1 X3 0
1111 1011	67	1 X1 1 X3 0 X2 1
1111 1100	64	1 X1 1 X2 0
1111 1101	78	1 X1 1 X2 0 X3 1
1111 1110	72	1 X1 1 X2 1 X3 0
1111 1111	46	1