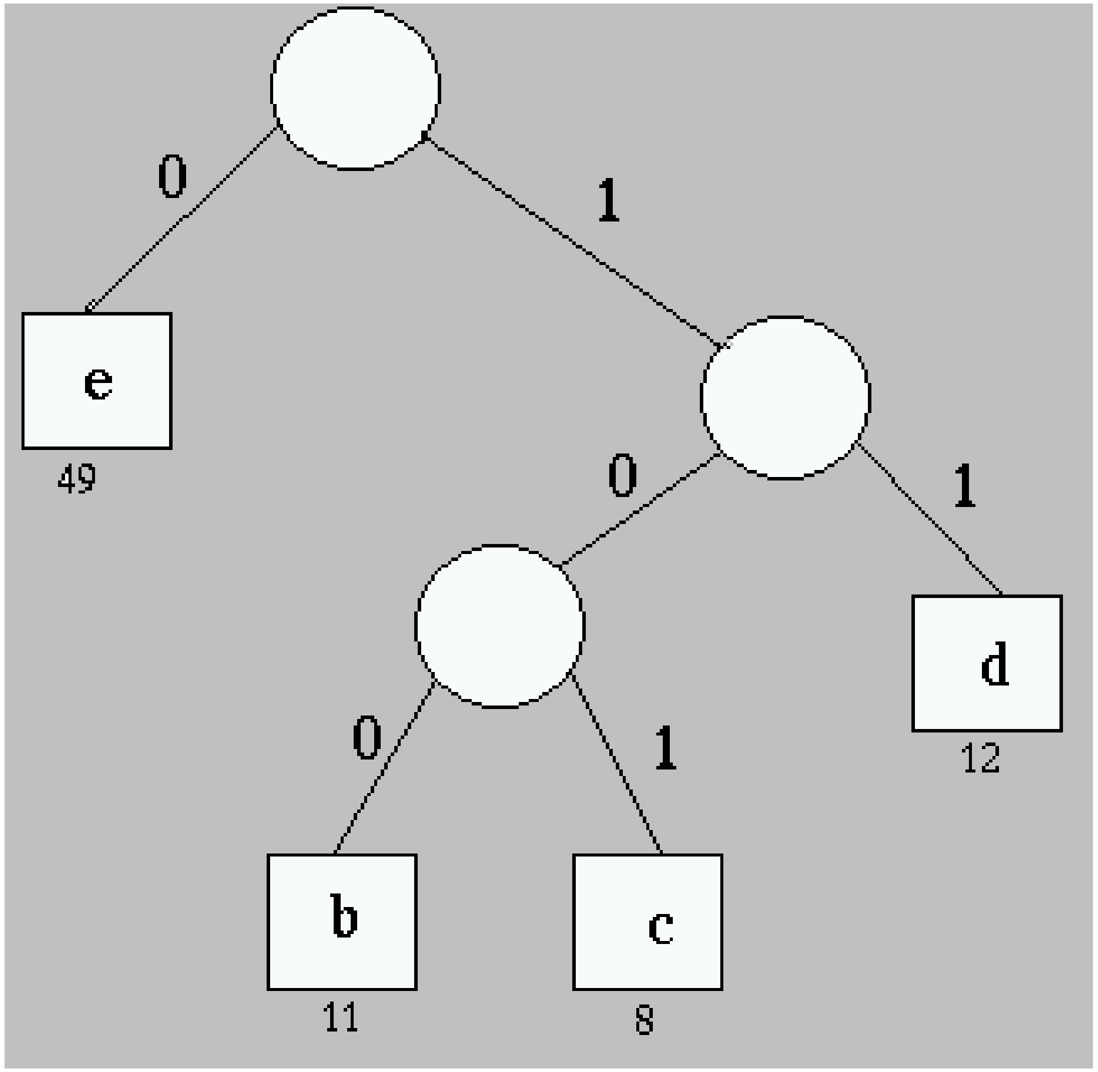
HW #6
(Huffman Tree Encoding / Decoding)

- The goals of this project are to practice Huffman Trees, polymorphism, and recursion.

- You are expect to do the work in Java (or C++ if you prefer).

- A Huffman Tree is a special type of binary tree used for data compression.
- A small Huffman Tree appears below:

HW #6 (2)



HW #6 (3)

- Each leaf in the tree contains a symbol (in this case a Character) and an integer value.

- Each non-leaf (internal node) contains just references to its left and right children.

- The 0’s and 1’s that you see are not stored anywhere, we just mentally associate 0 with any left branch and 1 with any right branch.

HW #6 (4)

- **Each character that appears in the tree is assigned a unique code (a sequence of 0's and 1's) obtained by following the path from the root of the tree to the leaf containing that character.**

- **Below are the codes for the four characters found in this sample tree:**

HW #6 (5)

- e is coded by “0”
- b is coded by “100”
- c is coded by “101”
- d is coded by “11”

HW #6 (6)

- **A sequence of characters can be "encoded" into a String of 0's and 1's by using the codings as described above.**

- **For example, the String "eddbc" would be encoded into the sequence "01111100101".**

  (That is good compression – we went from 5 characters down to 11 bits.)

HW #6 (7)

- A sequence of 0’s and 1’s can be “decoded” into a Sting of Characters by using the codings backwards.

- For example, the sequence “11010111” could be decoded into the String “decd”.

HW #6 (8)

- Given a “sample” stream of Characters, one can create a Huffman Tree which will do a very good job of compressing other streams whose characters share the same relative frequency as those in the sample stream used to create the Tree.

HW #6 (9)

1. The first step is to count the frequency of each character in the “sample” stream of Characters.

- Suppose your sample stream is:

  “eebbeecdebeeebecceeeddebbbeceedebeed deeeecceeeedeeedeeebeedeceedebeeedecee edebee”

HW #6 (10)

- Below are the counts for each character that occurs in the stream:

- 49 e’s
- 11 b’s
- 8 c’s
- 12 d’s

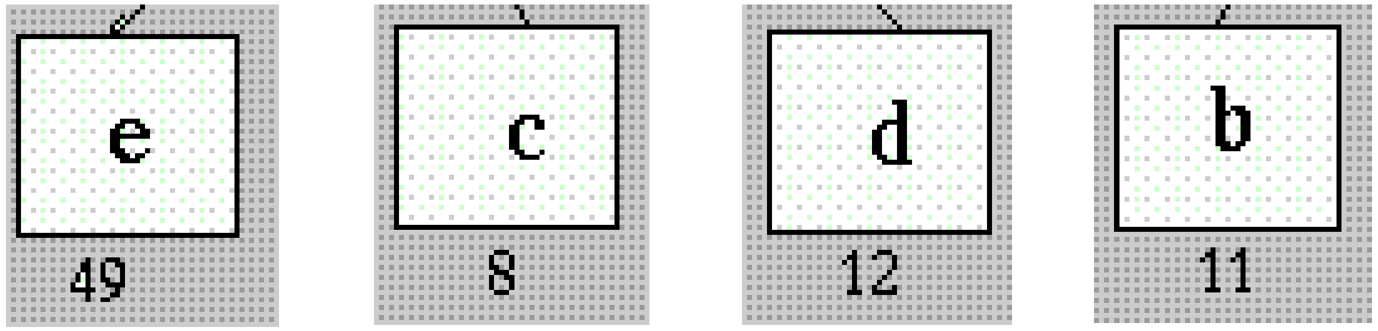
HW #6 (11)

2. In this example, the next step is to create four little trees – one for each character – each consisting of just a single leaf.

- The little trees should be put into some kind of list.

HW #6 (12)

- **Below is a picture of the list of four little trees:**



HW #6 (13)

3. The next step is to select two trees with the SMALLEST frequencies.

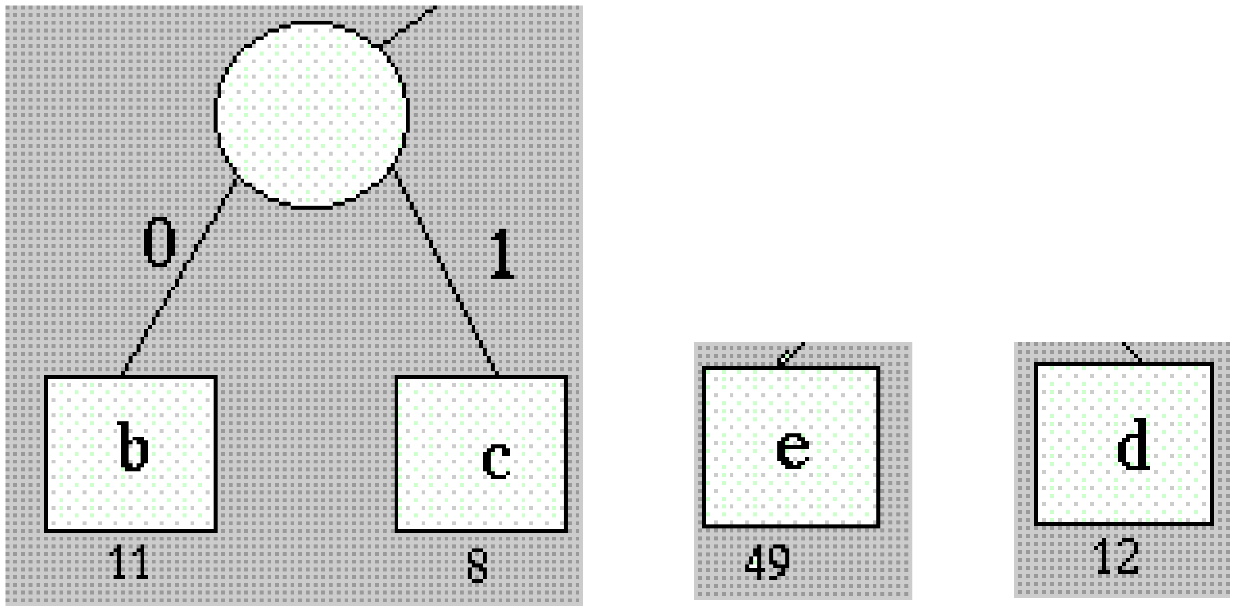
- In this example, that would be 8 and 11.

- Remove them from the list of trees, and then join them as children of a new node to make a new tree.

- Then add this new tree to the list of trees.

HW #6 (14)

- **Below is the list of trees after this step has been accomplished.**
- **There are now three trees in the list.**



HW #6 (15)

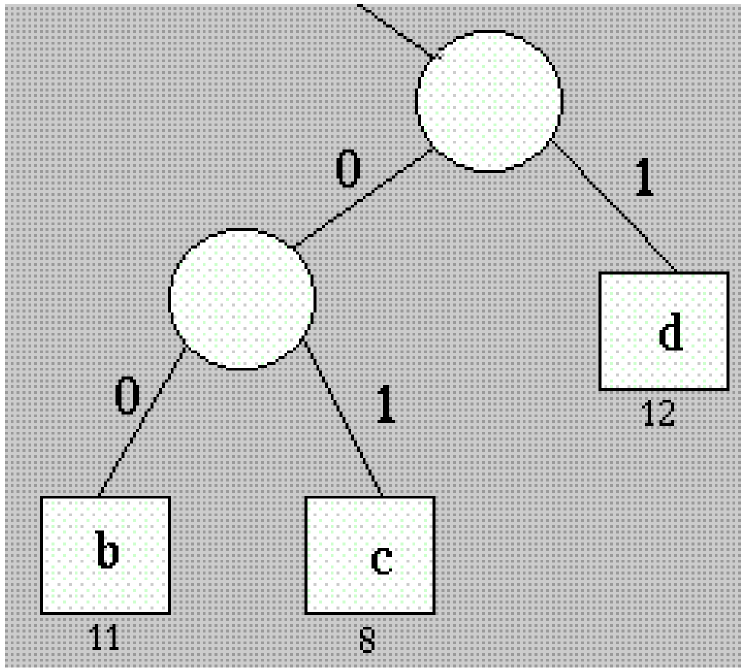
- The “frequency” of a tree that is more than just a single leaf is the sum of the frequencies of all leaves below it.

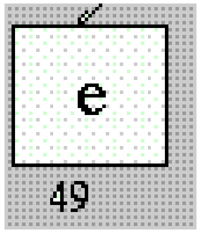
- So the frequency of the first tree is 19, the sum of its leaves.

- Now we repeat step 3. (This time, the two smallest frequencies are 19 and 12.)

HW #6 (16)

- Below is the list of trees after we join them. (We are down to TWO trees now.)



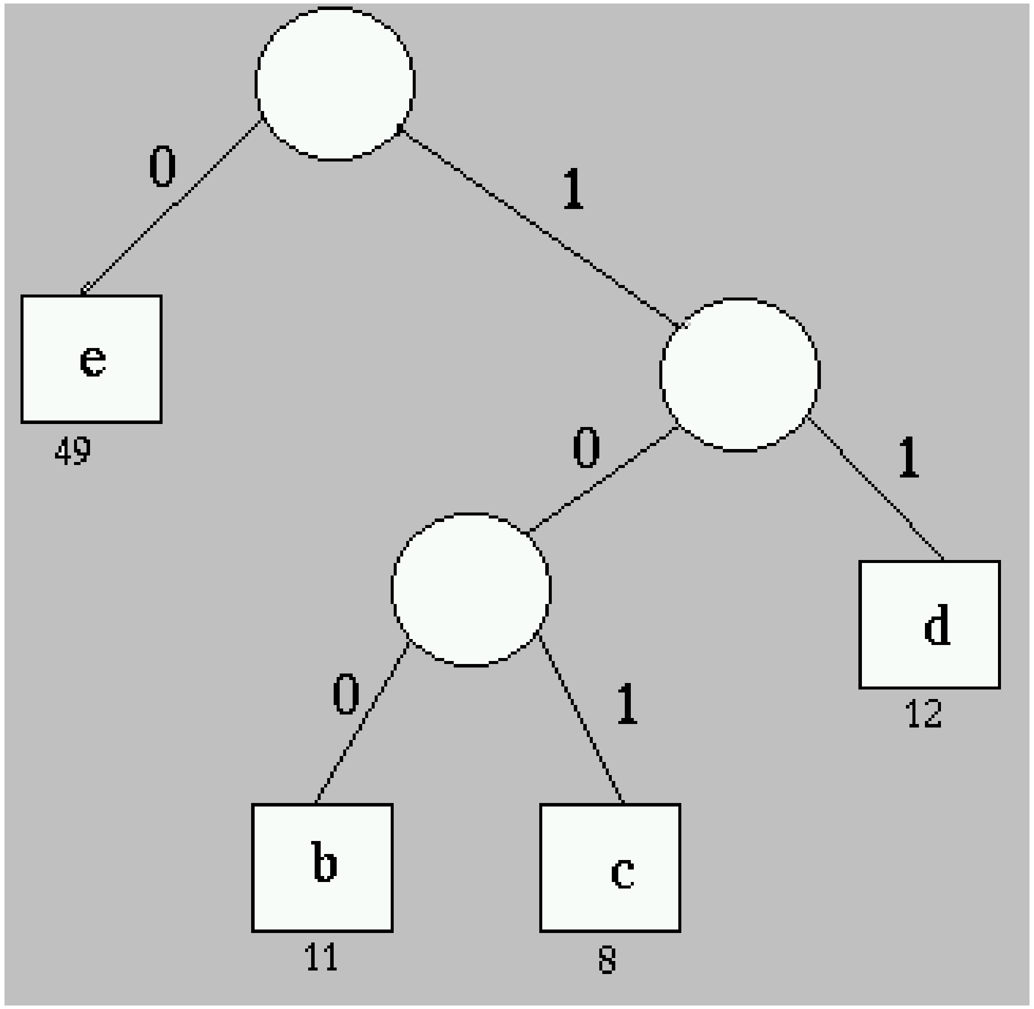


HW #6 (17)

- Now we repeat step 3 again. (This time, the two smallest frequencies are obvious – 31 and 49.)

- On the next slide is the list of trees after we join them. (Just one tree now.)
- Since the list now contains just one tree, we are done – this is the Huffman Tree generated by the “sample” sequence of characters.

HW #6 (18)



HW #6 (19)

- You will write a program that will first construct a Huffman Tree from a sequence of Characters.
- Each leaf at the bottom of the tree will represent just a single Character.
- We will allow letters, digits, punctuation, and all other type of Character.

HW #6 (20)

- Once the Huffman Tree has been built, your program will be able to do two things:
- "encode" a sequence of Characters into a String of 0's and 1's using the Huffman Tree.
- "decode" a sequence of 0's and 1's into a String of Characters using the Huffman Tree.

HW #6 (21)

- **HuffmanTree is an interface that will be implement by two classes:**

  HuffmanInternalNode and HuffmanLeaf.

HW #6 (22)

- A HuffmanLeaf is a node at the bottom of the Tree. You may not add any other fields to this class:

1. private Character data – the Character that this leaf holds.
2. private int count – the relative frequency of this character (how many times it occurred in the sequence of Characters used to generate the Huffman Tree.)

HW #6 (23)

- A HuffmanInternalNode is a node in the tree that is not a leaf. It has two fields that are self-explanatory. You may **not** add any other fields to this class:

-
  ```
  private HuffmanTree leftChild, rightChild;
  ```

HW #6 (24)

- There is a class called HuffmanTools that will contain the three static methods that make this project work. You will be writing these methods. They are:

1. public static HuffmanTree buildHuffmanTree (Iterator symbols) – The iterator "symbols" will provide a sequence of Character objects.

- These are used to construct a Huffman Tree, which is returned by the method.

HW #6 (25)

2. **public static String encode(HuffmanTree tree, Iterator symbols)** – The first parameter (tree) is the root of an existing Huffman Tree.

- The second parameter is an Iterator that will provide a sequence of Character objects.

- The return value will be a String of ‘0’ and ‘1’ characters that represent the encoding of the Characters in the Iterator.

HW #6 (26)

- For example, if “tree” is the Huffman Tree from the example pictured above, and the Iterator gives you the sequences ‘e’, ‘d’, ‘d’, ‘b’, ‘c’ (five separate Character objects), then the return value would be the String “01111100101”.

  (These are not really bits, they are the characters ‘0’ and ‘1’.)

HW #6 (27)

3. **public static String decode(HuffmanTree tree, Iterator bits) – The first parameter is the root of a Huffman Tree.**

- **The second parameter is an Iterator that will provide a sequence of Character objects that are all either ‘0’ or ‘1’.**

- **The return value will be a String of characters that represent the decoding of the sequence of 0’s and 1’s.**

HW #6 (28)

- For example, if “tree” is the Huffman Tree from the example pictured above, and the Iterator gives you the sequence ‘1’, ‘1’, ‘0’, ‘1’, ‘0’, ‘1’, ‘1’, ‘1’ (eight separate Character objects), then the return value should be the String “decd”.

HW #6 (29)

- 如果用 C++，Iterator symbols (or bits) 可以改成 String, i.e., a null terminated character array.

想要用 C++ 的 STL 所支援的 Iterator 請見：

- Iterator – From Wikipedia, the free encyclopedia
- 迭代器 – 維基百科，自由的百科全書