Application of trees Graphical Representation of data structures

CS 293 Final Project Demo

25/11/2012

Abhishek Gupta, 110040067

Mridul Ravi Jain , 110040083

Outline (for a total of 30 mins)

- Aim of project (1 min)
- Demo (5 mins)
- Teamwork Details (0.5 min)
- Design Details –Algorithm (5 mins)
- Design Details – Implementation (8 mins)
- Viva (9 mins)
- Transition time to next team (2 mins)

Aim of the project

- Our Project mainly consists of two parts
  - Application Of Trees
    - Huffman Coding – data compression for text files using huffman coding
    - Trie – preprocessing a large text file in form of a trie to search all the words starting with the given string, extra features like suggesting a word etc are added
  - Graphical Representation of data structures
    - In this part we are representing some of the data structures graphically(stack,queue,list and binary search tree)

Demo (for huffman coding)

- In Huffman Coding we have made two programs one is encoder which would create the compressed files from the text file and the other is decoder which would give us the original text file using the compressed files
- In the encoder part I read the text from the file and store the characters and their frequencies in a vector

Demo (for huffman coding)

- Then I write the vector in a file because this would be used while reading
- Now using this vector I create huffman tree
- I visit all the leaves of the huffman tree and assign the characters their codes(if I go left then I add 0 else 1)
  - So if e is at depth 2 and leftmost then the code assigned to it would be 00

Demo (for huffman coding)

- Now I have all the codes for the characters , now I read the text file and whenever a character occurs then I apend to the string its new code
- So the string is the whole text in binary form but each 0 and 1 in string takes 1 bytes , we have to write them such that each 0 and 1 takes 1 bit

Demo (for huffman coding)

- We have the final string , we only have to write it now , for that I have created a “write” function which would write the string passed to it in a file in a compressed way
- For writing to file I have created a structure and defined a member function of it which takes only 1 bit.I have created 8 objects of this structure in the write function.

Demo (for huffman coding)

- Do the following till end of the string is reached
  - store the 8 characters(0's or 1's) of the string in the member of the 8 object's created earlier.
  - Now I store the 8 bits ,which I have in the member of the 8 objects created earlier , in a character and write this character in the file.
- So now the file has been compressed.

Demo (for huffman coding)

- The decoder function decompresses the compressed file and give us the original text file
- It first reads from the encoded vector file and recreates the vector
- Using this vector, a huffman tree is created
- Now this huffman tree is used for decompressing the compressed file

Demo (for huffman coding)

- In read function, I read the 1st byte from the compressed file in a character and then assign the 8 bits of the character to the members of the 8 objects of the structure created earlier using bitwise shift operations
- If the bit is 0 then I go left in the tree else right, as soon as I reach a leaf I print the character associated with it and again start from the root of the tree

Demo (for huffman coding)

- So using the encoder function we are able to compress the text file
- Using the decoder function e are able to get the original text file using the compressed files

Building a Tree

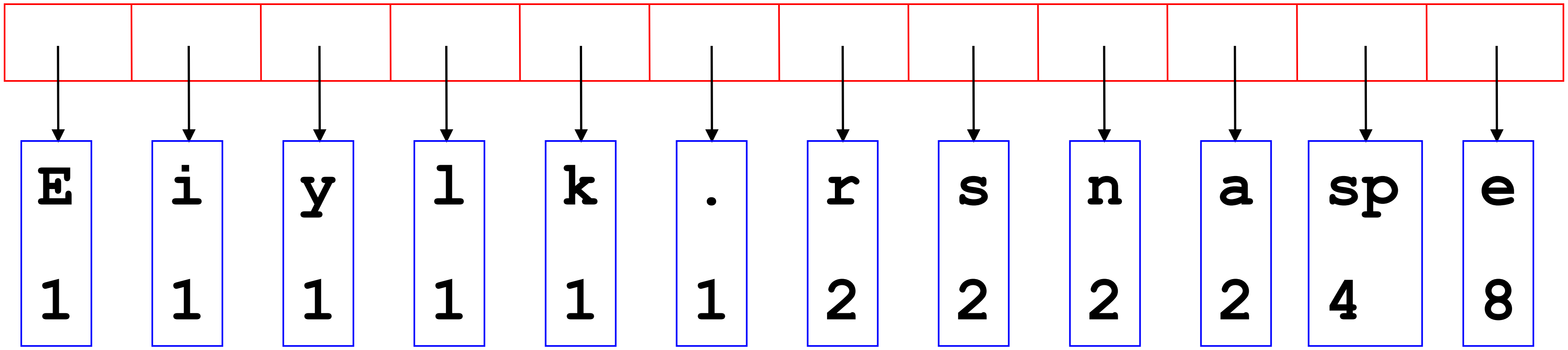
Scan the original text

Eerie eyes seen near lake.

Char	Freq.	Char	Freq.	Char	Freq.
E	1	y	1	k	1
e	8	s	2	.	1
r	2	n	2		
i	1	a	2		
space	4	l	1		

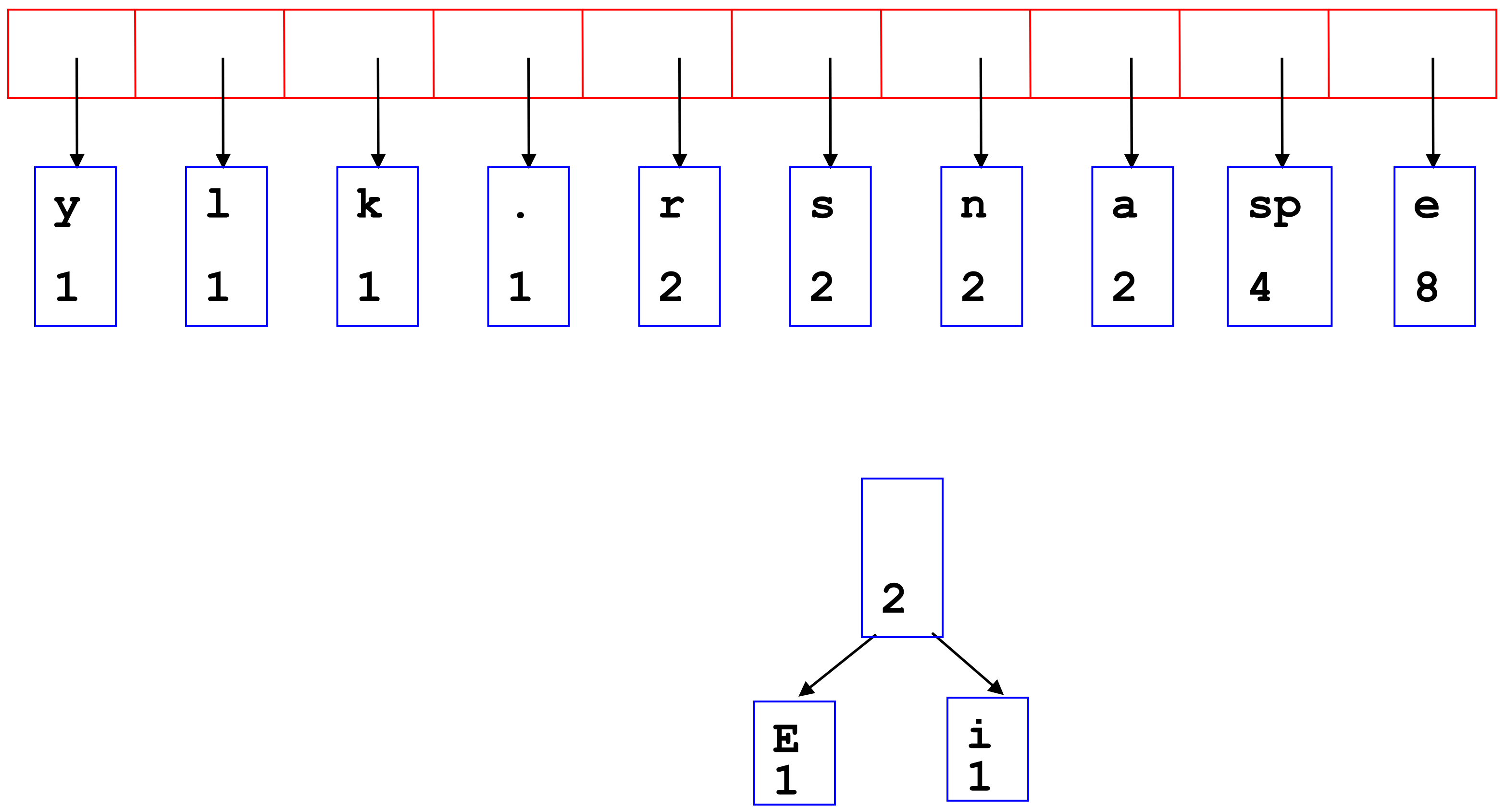
Building a Tree

The queue after inserting all nodes

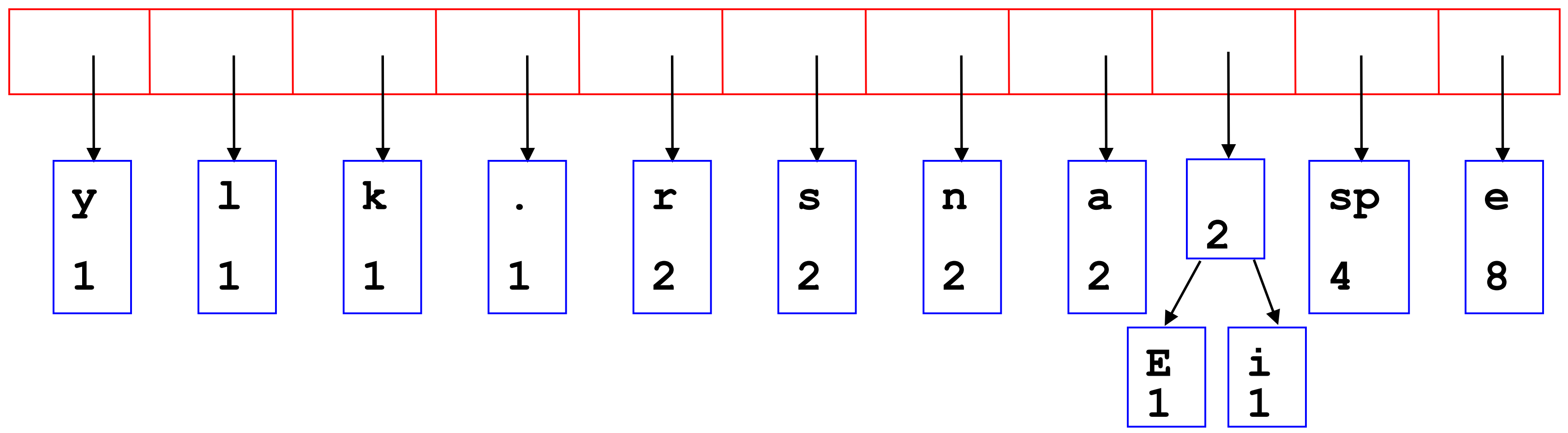


Null Pointers are not shown

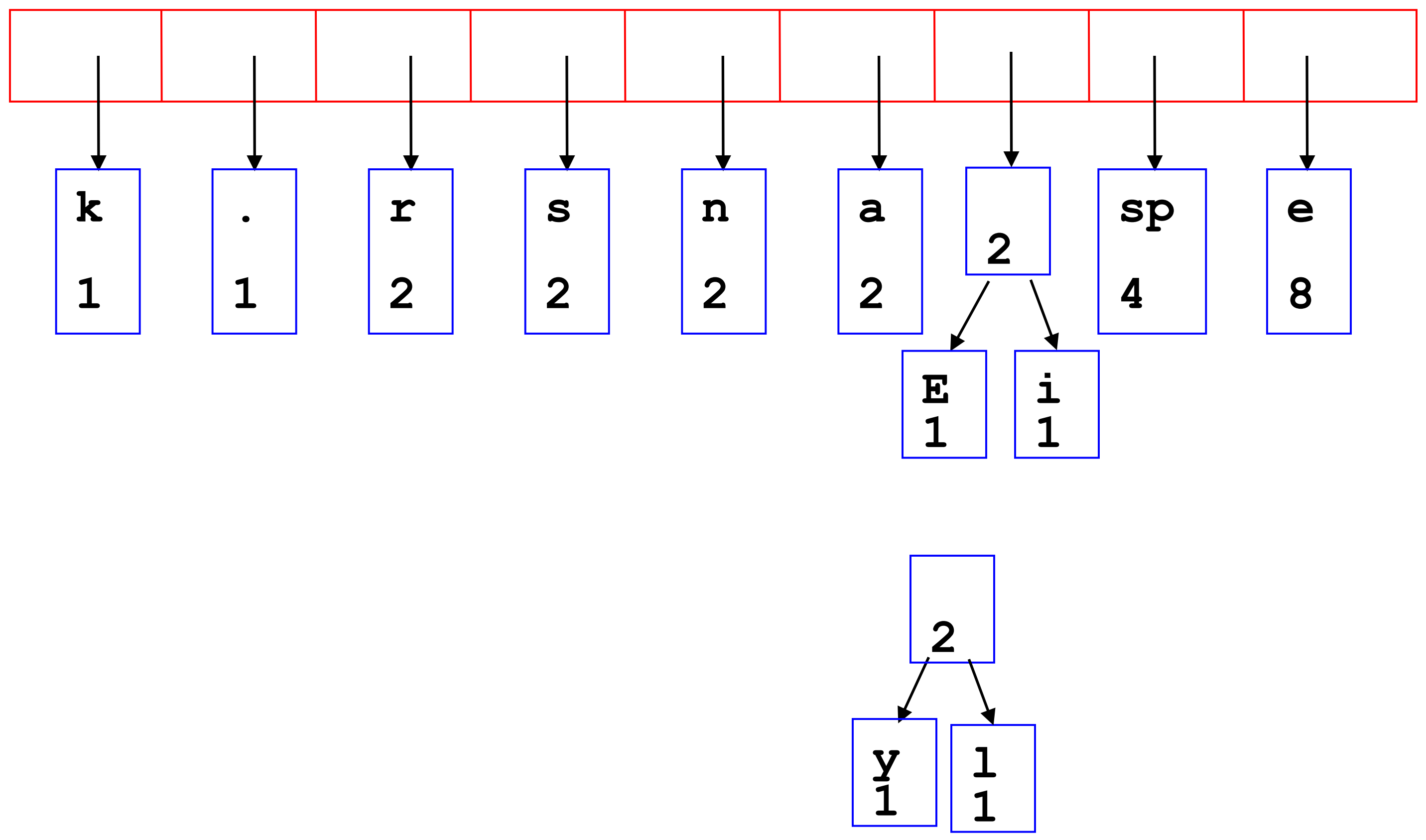
Building a Tree



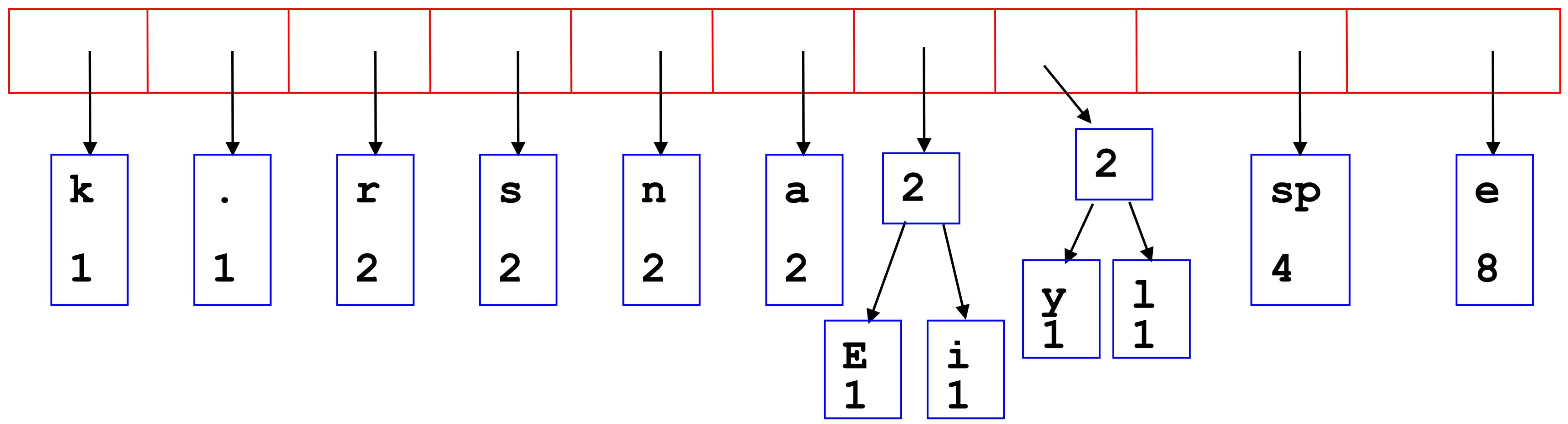
Building a Tree



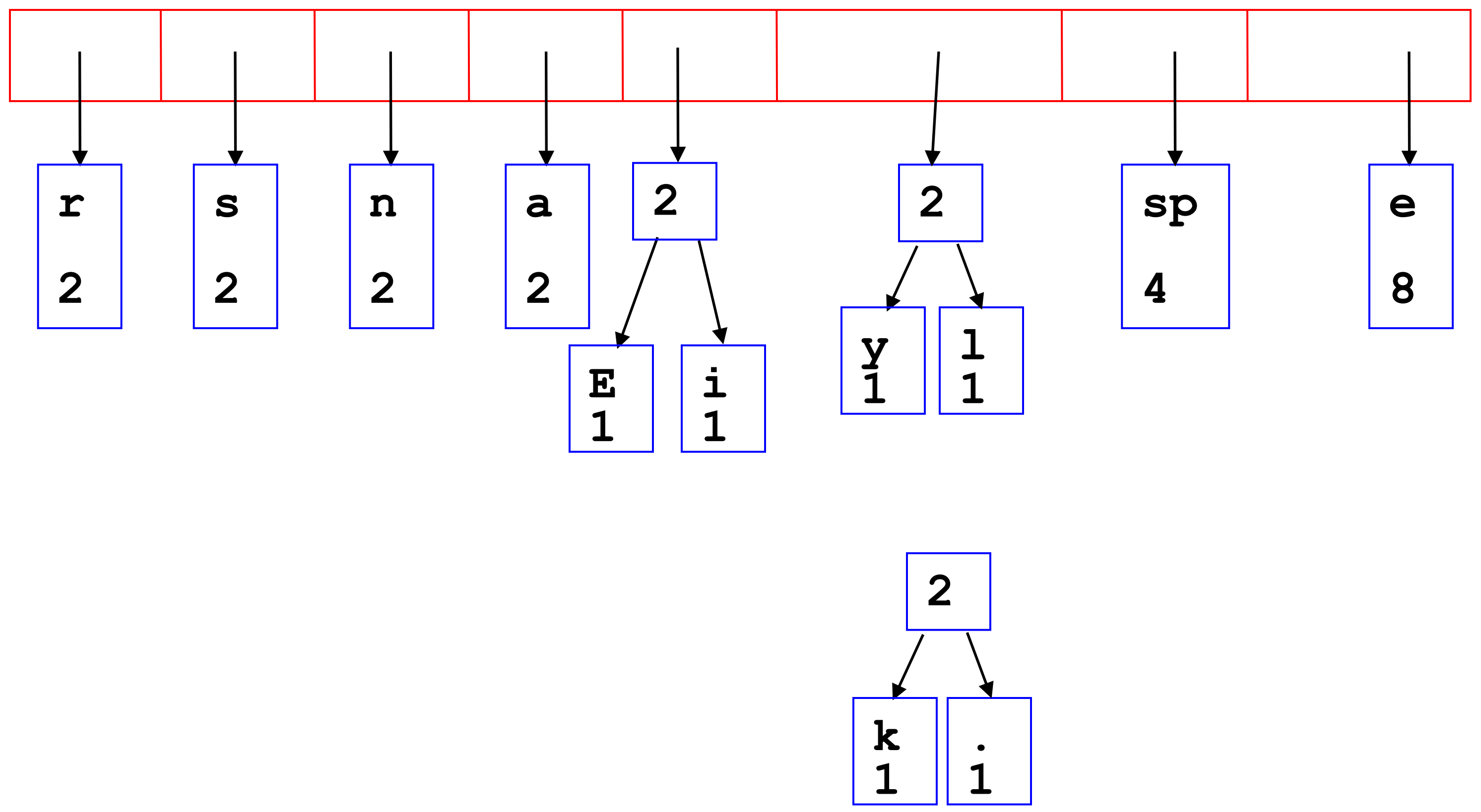
Building a Tree



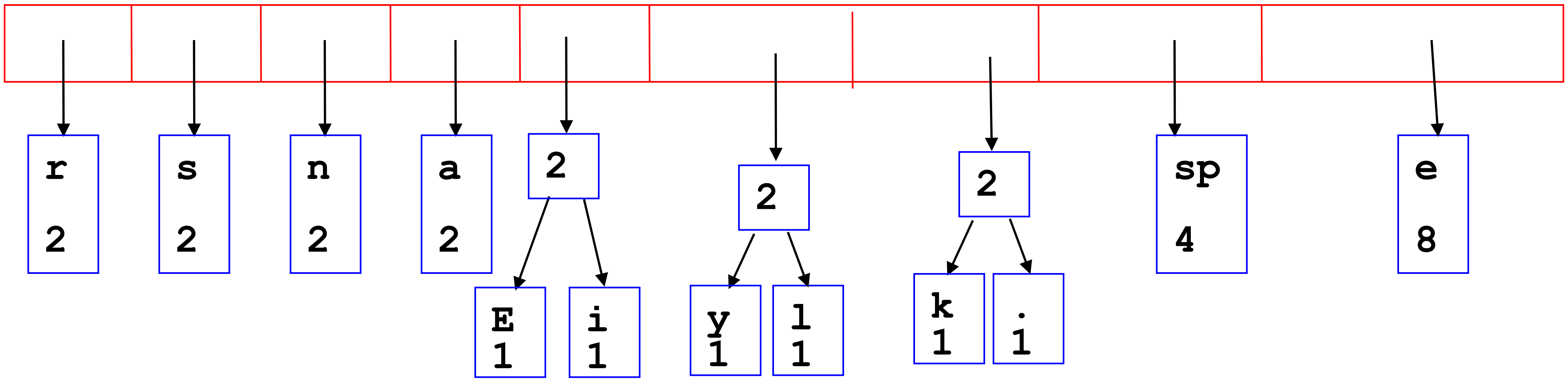
Building a Tree



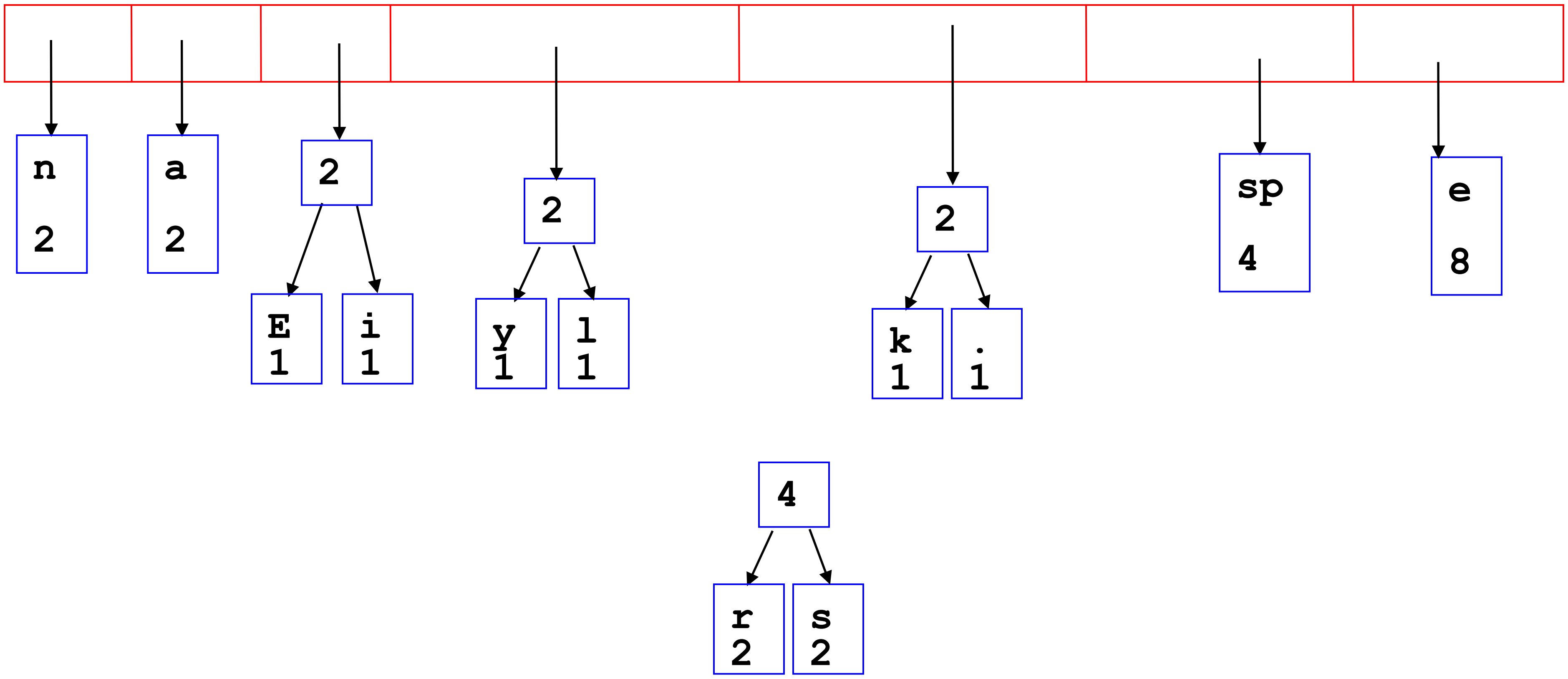
Building a Tree



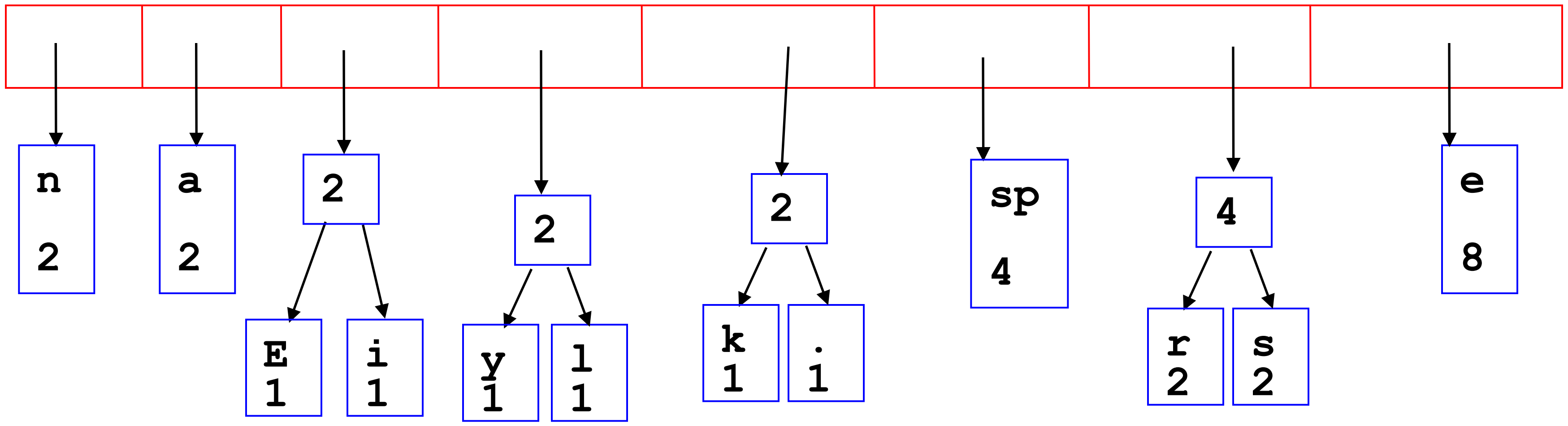
Building a Tree



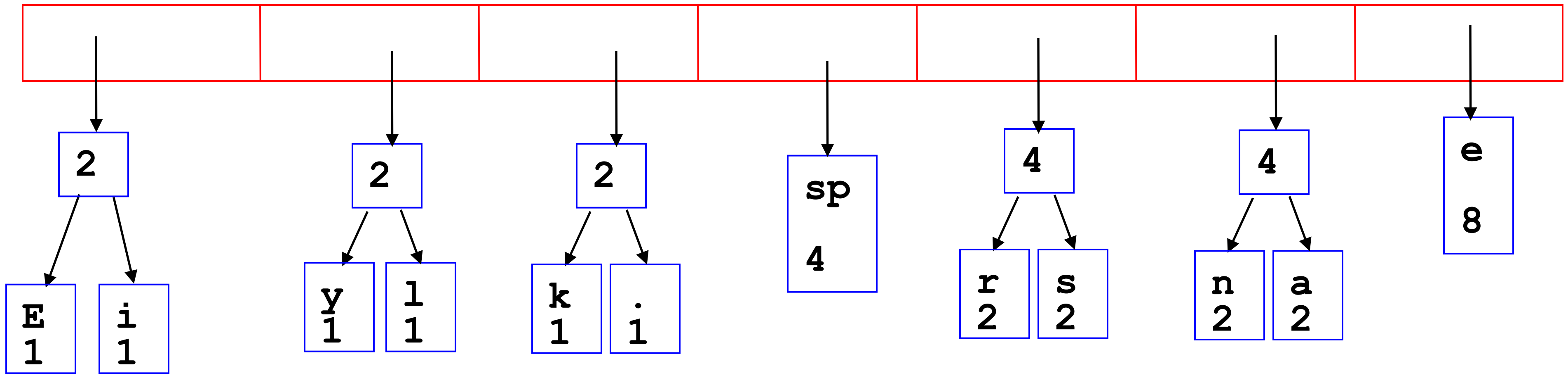
Building a Tree



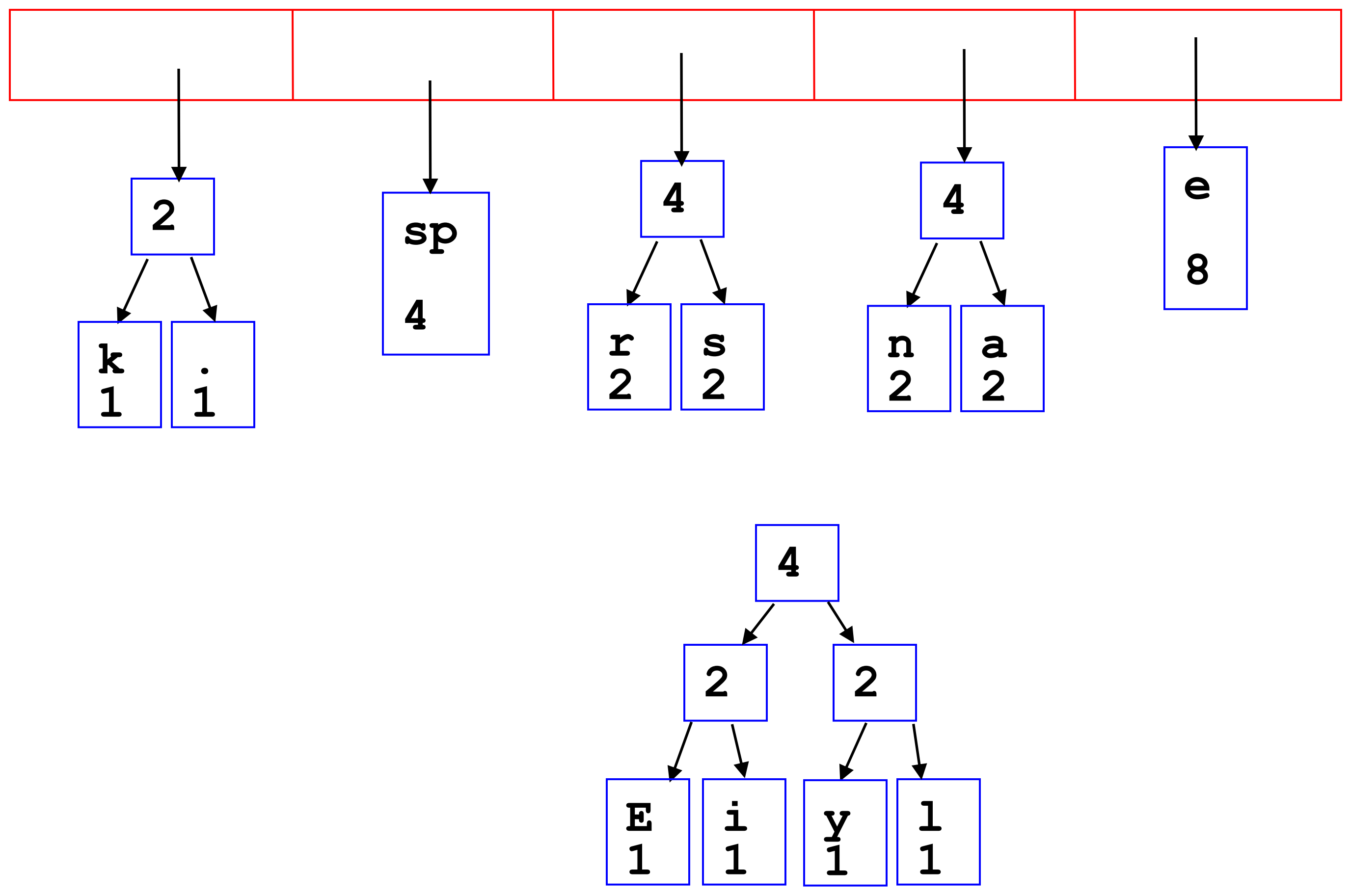
Building a Tree



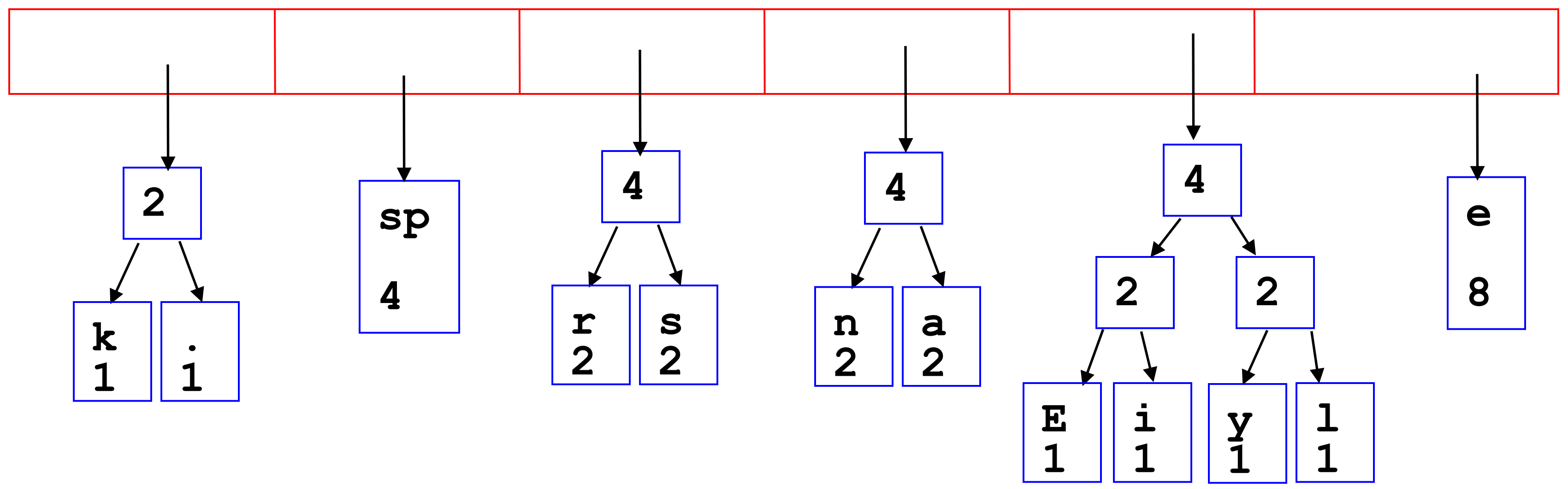
Building a Tree



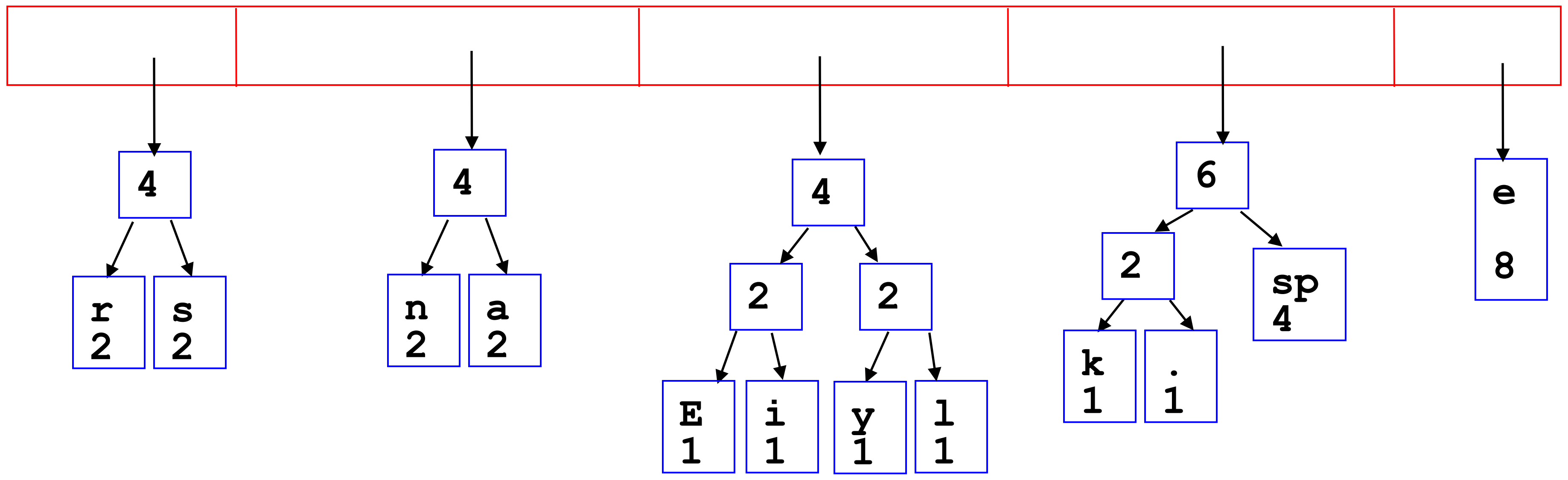
Building a Tree



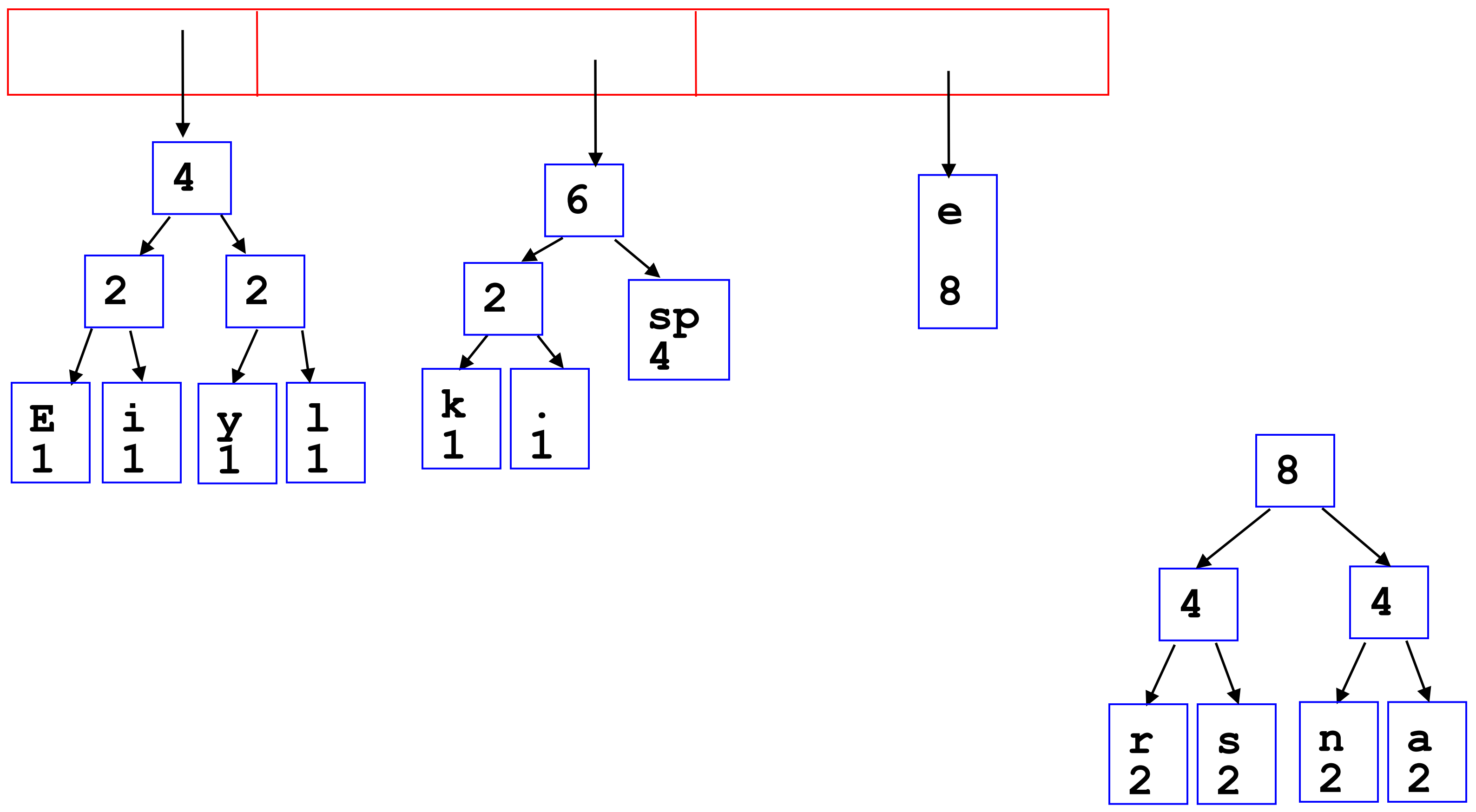
Building a Tree



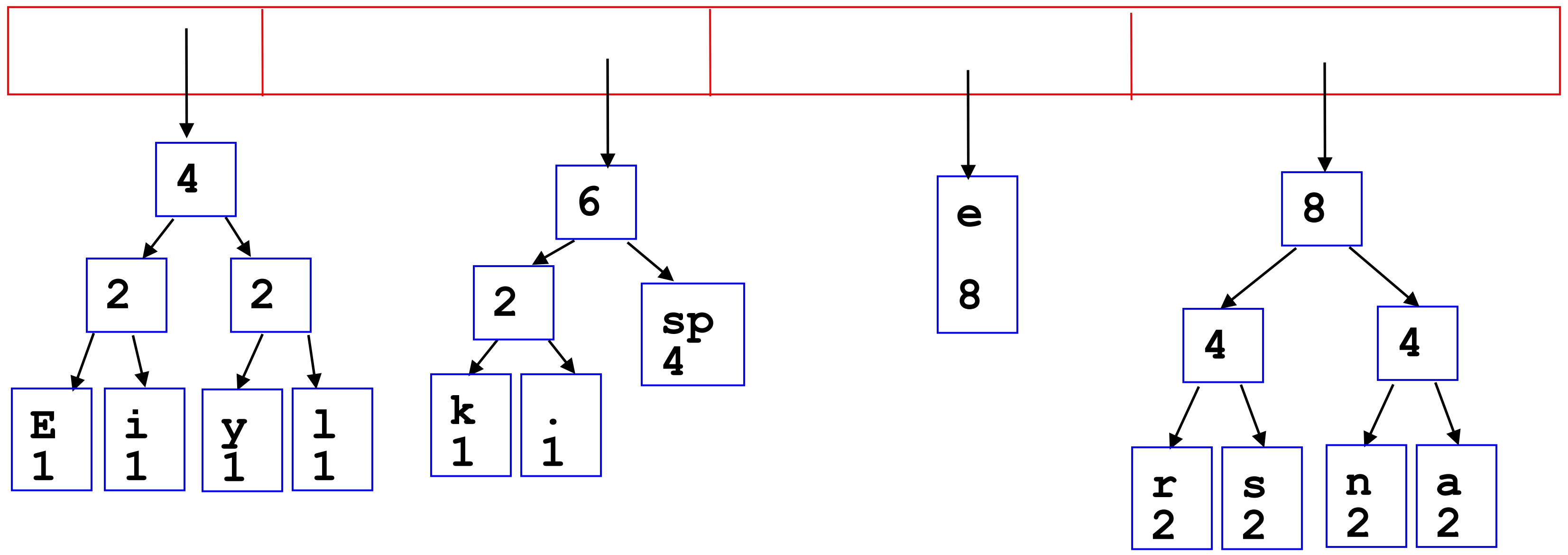
Building a Tree



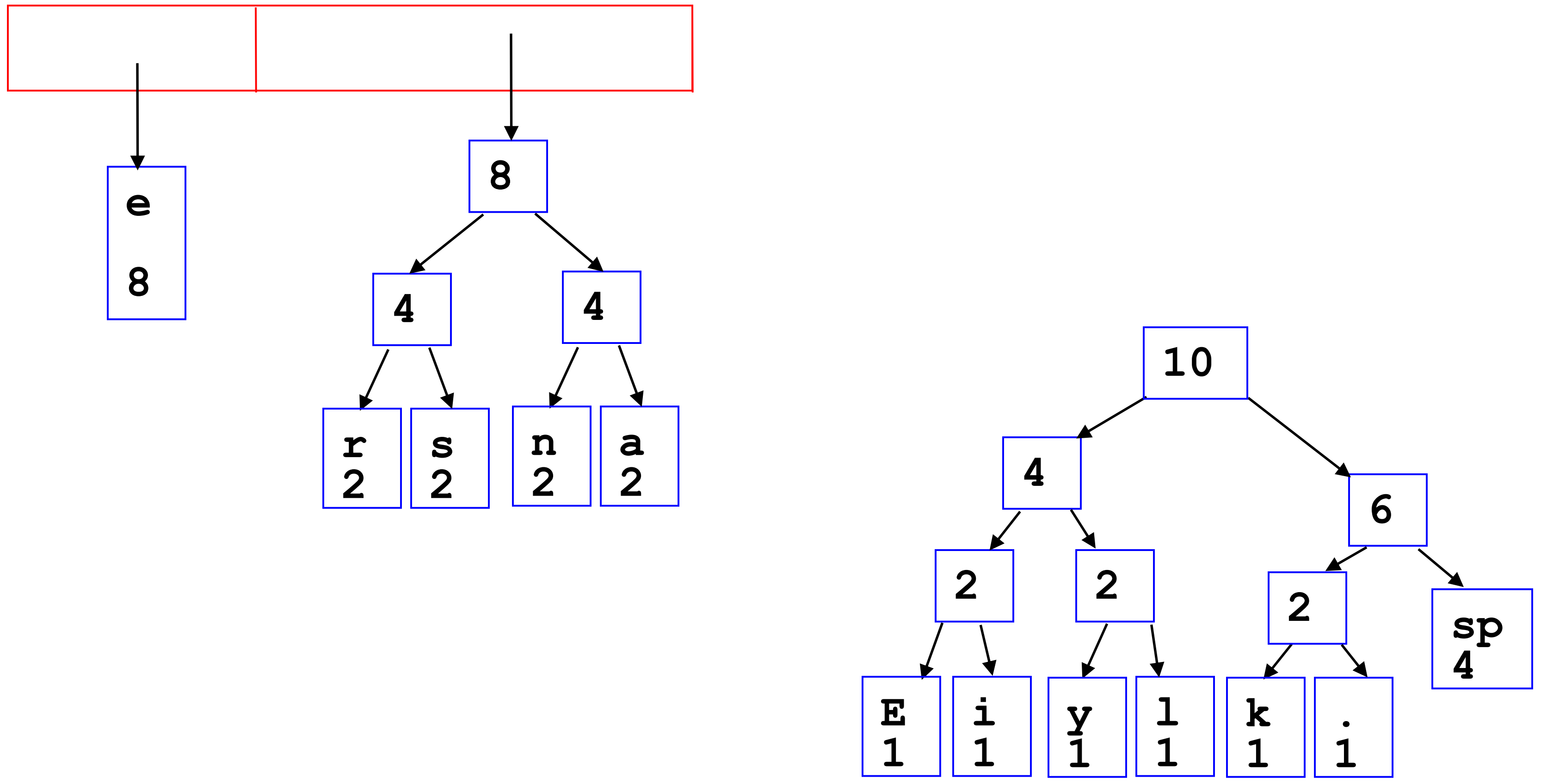
Building a Tree



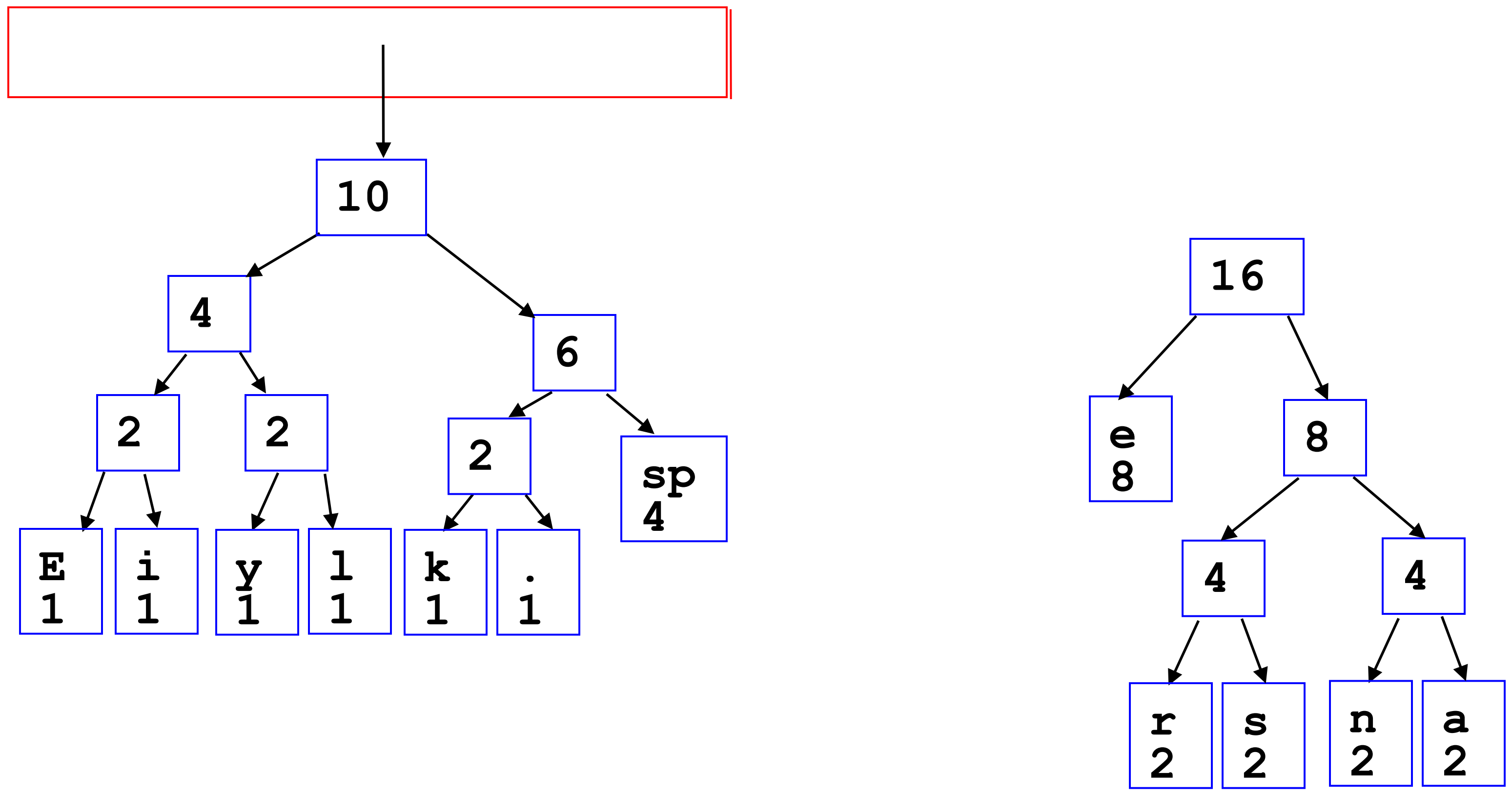
Building a Tree



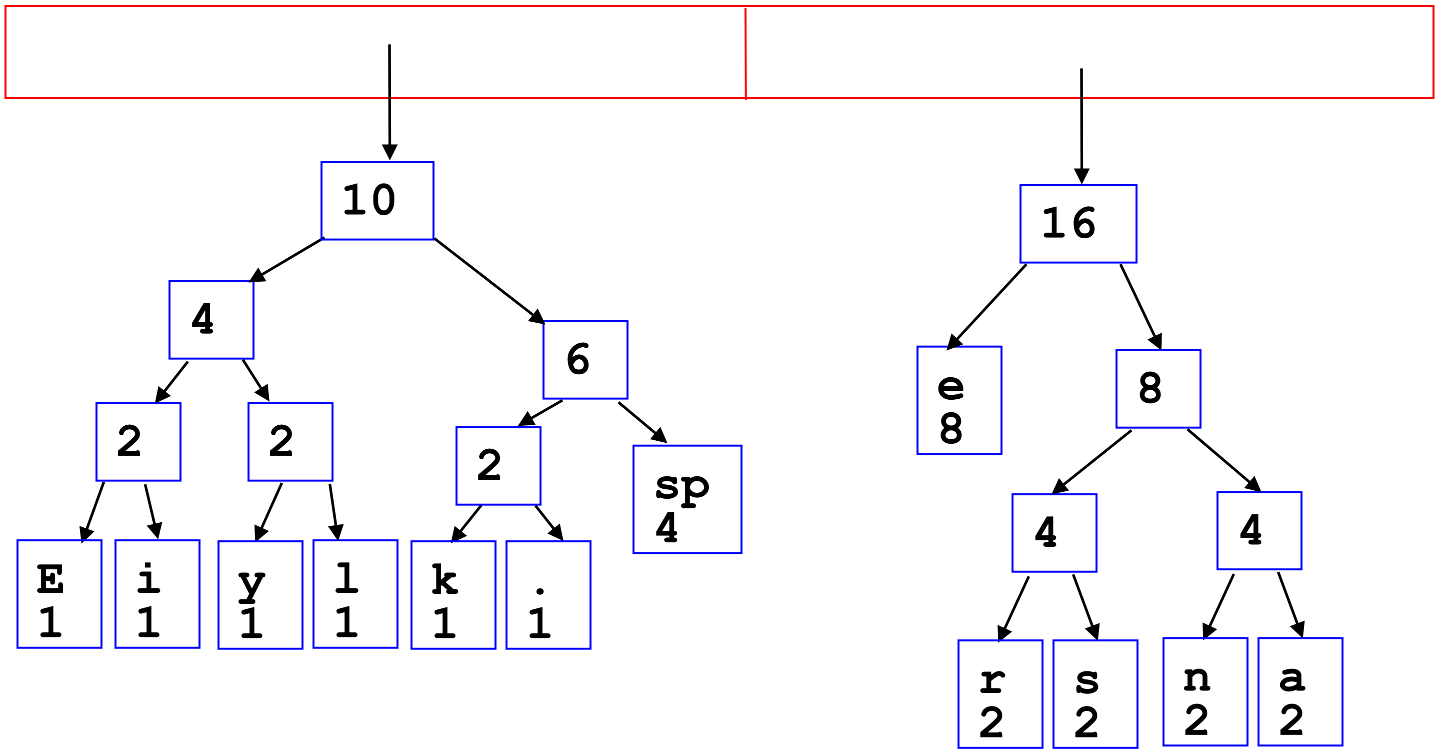
Building a Tree



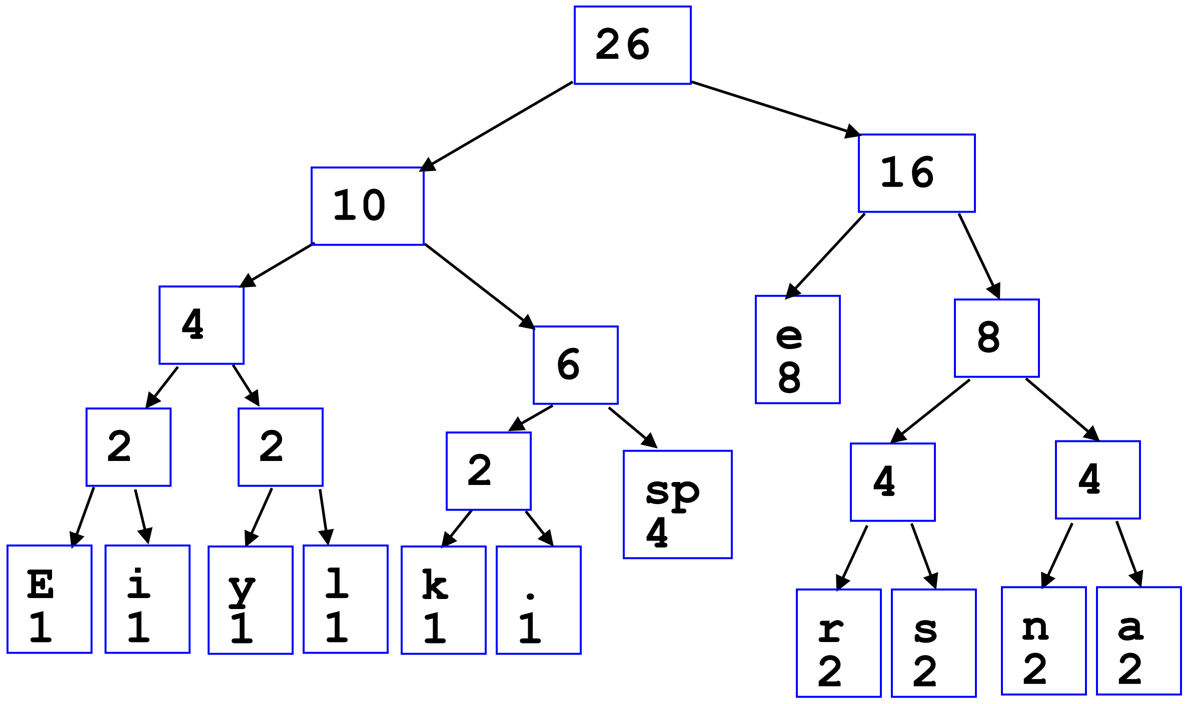
Building a Tree



Building a Tree

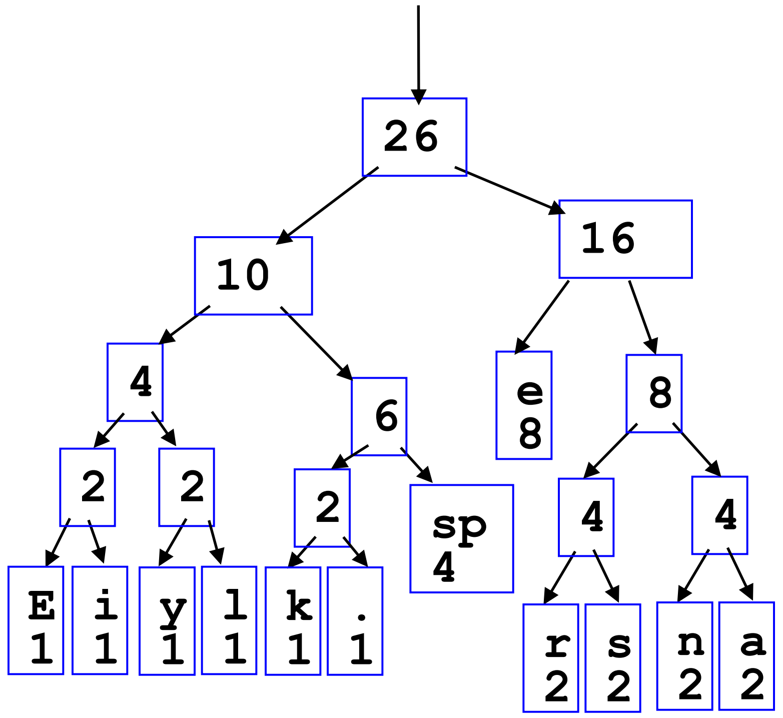


Building a Tree



Encoding the File
Traverse Tree for Codes

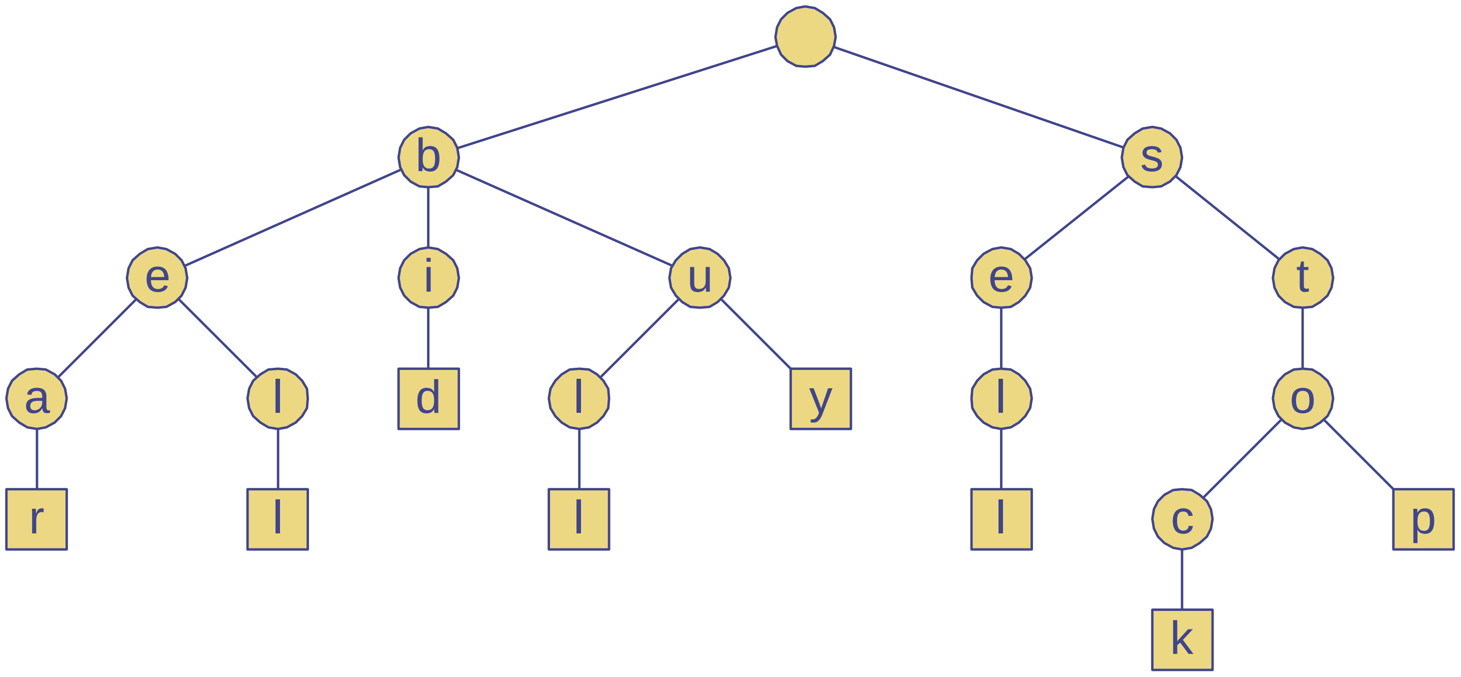
Char	Code
E	0000
i	0001
y	0010
l	0011
k	0100
.	0101
space	011
e	10
r	1100
s	1101
n	1110
a	1111



Trie implementation-1

The standard trie implementation done by us:

S = { bear, bell, bid, bull, buy, sell, stock, stop }



Trie implementaion-1

Each node is described by :

1. an character element,
2. pointers to 26 nodes corresponding to each of its children, and
3. a flag which is set to 1 if any word ends at this node,otherwise 0

Trie implementaion-1

The implementation is divided into 2 modules:

1. Insertion :

   Preprocessing the input file to form a trie

   - We have done an advanced implementation where we would preprocess any general text file to form a trie.

2. Searching :

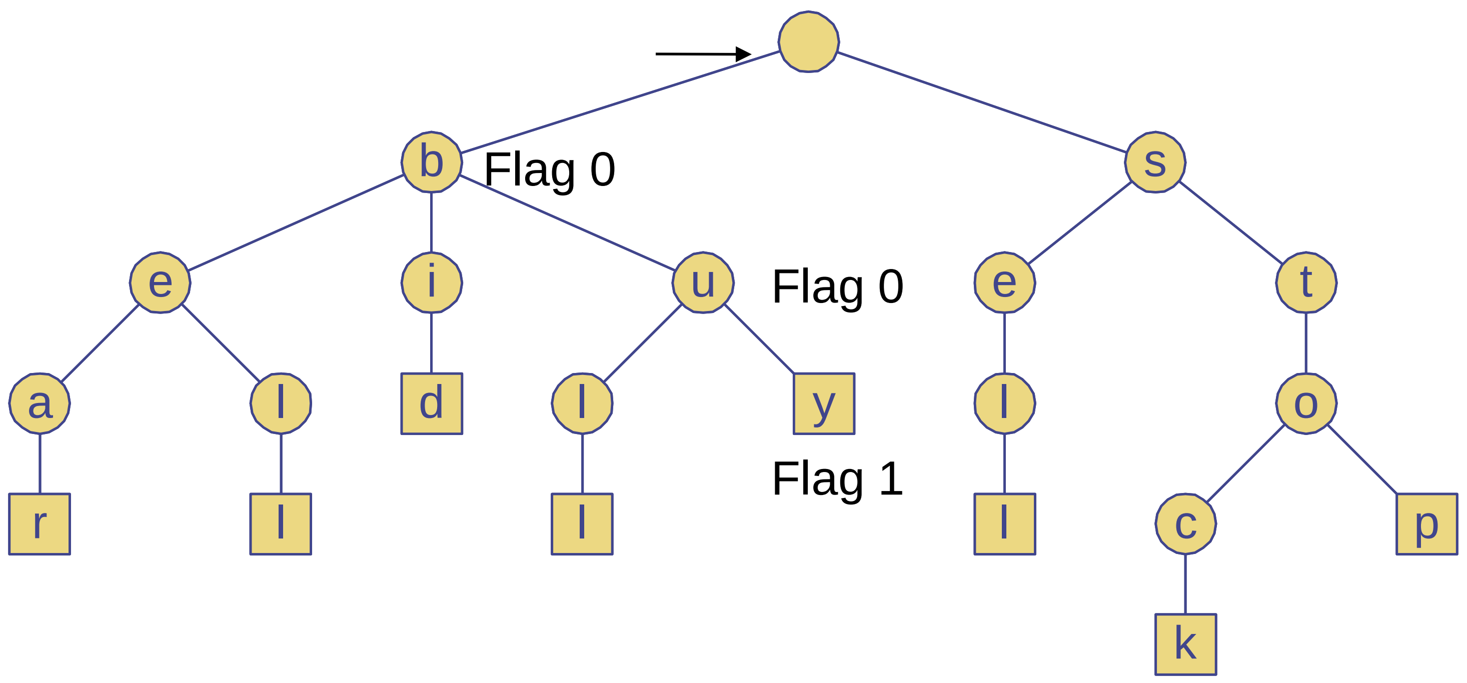
   Finding a word in a trie, with following features implemented :

   1. Removed case sensitivity
   2. Backspace(<)
   3. “Did you mean :” feature

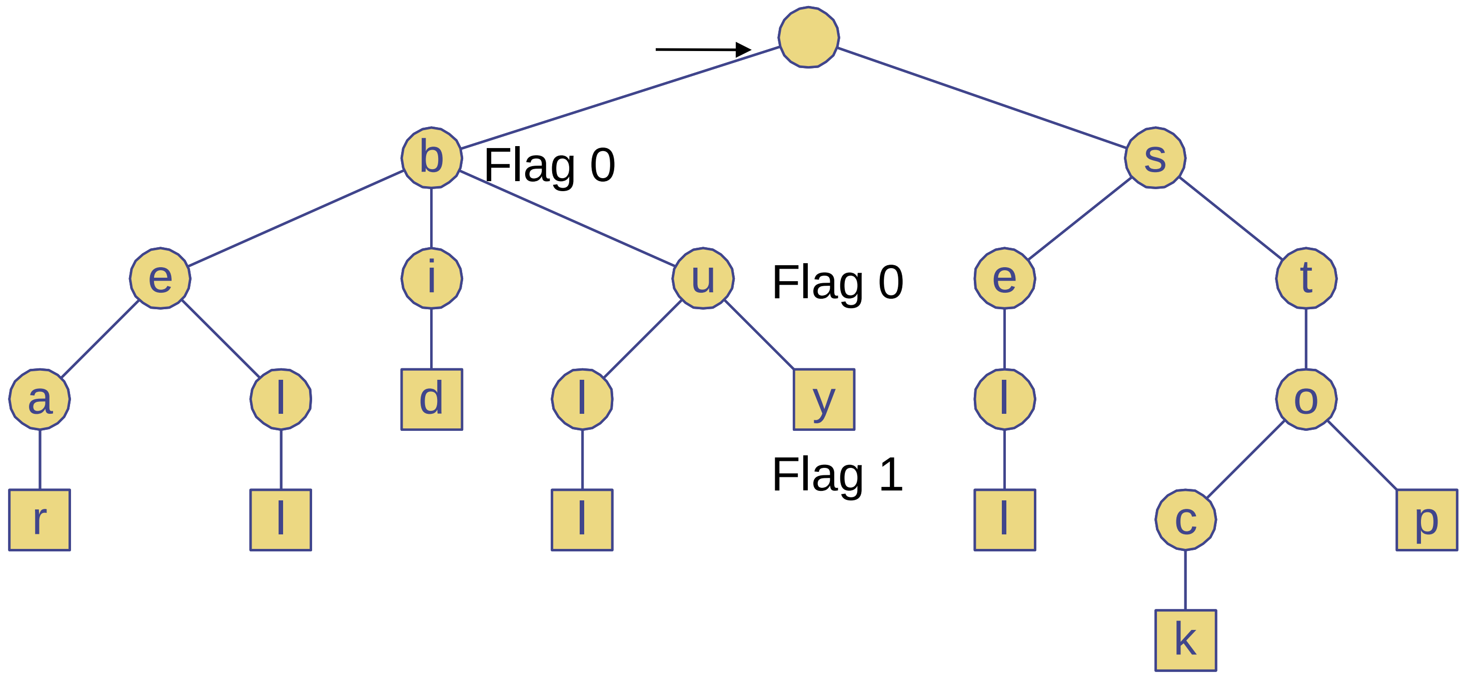
Insertion

- We maintain a pointer initially poining at the root node
- Only charaters in the range a-z, A-Z are inserted into the TRIE.
- When a character is inserted, our pointer moves to the newly added node
- When any special character is encountered, it is considered the end of a word, and our pointer moves back to the root because now any new character inserted will be the start of a new word

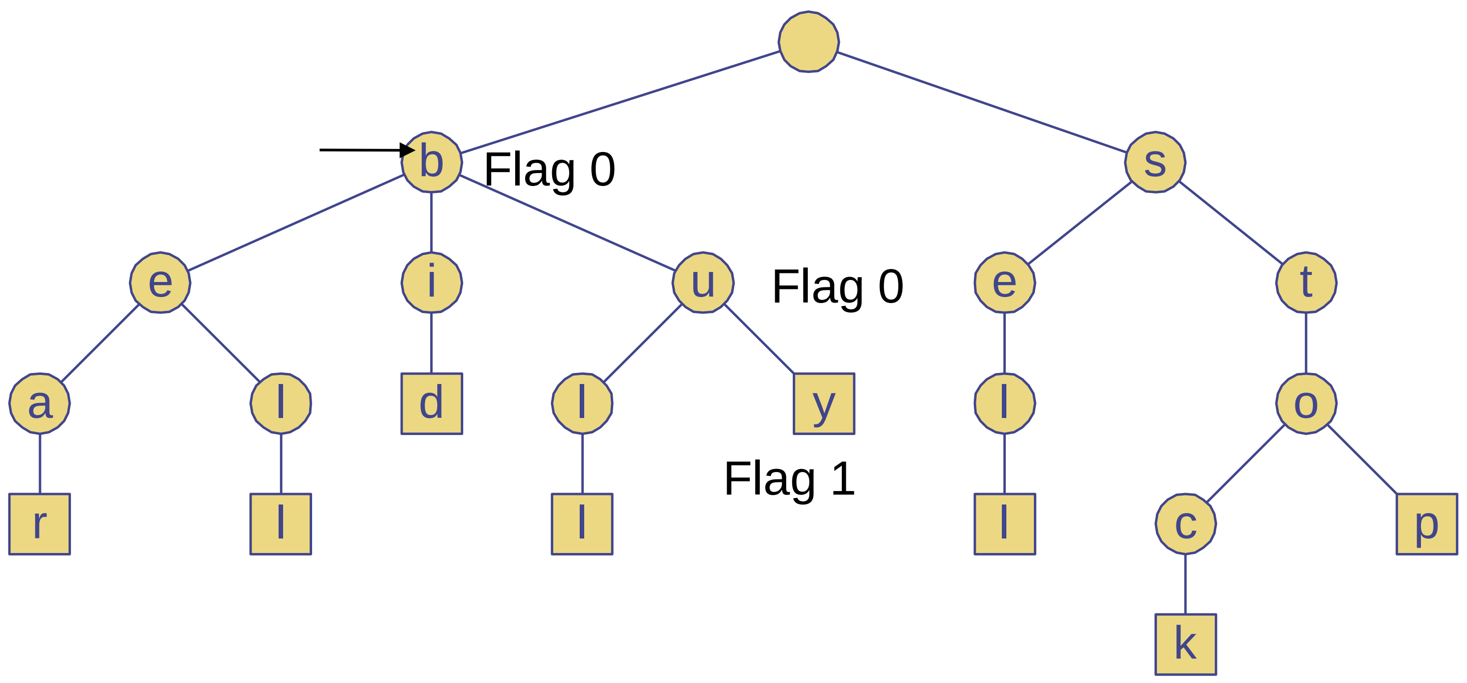
Insert 'buyer'



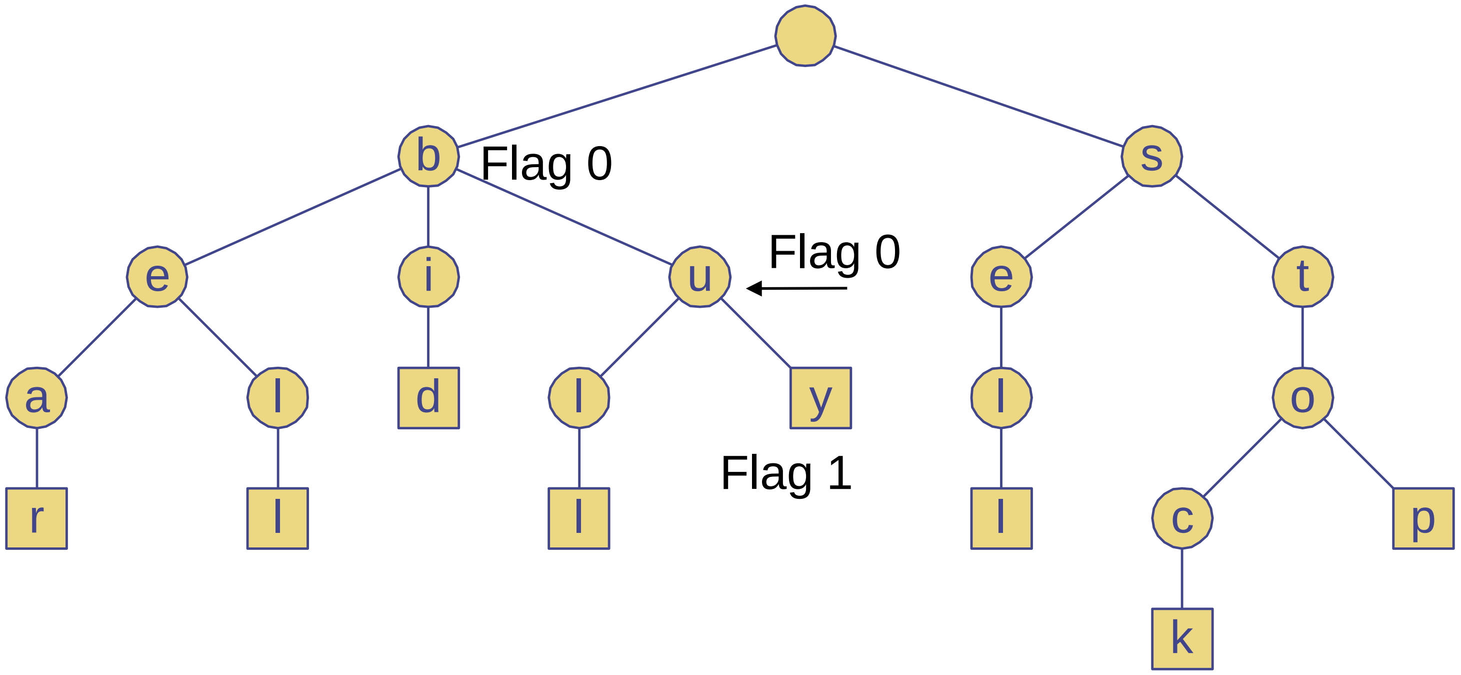
Insert 'buyer'



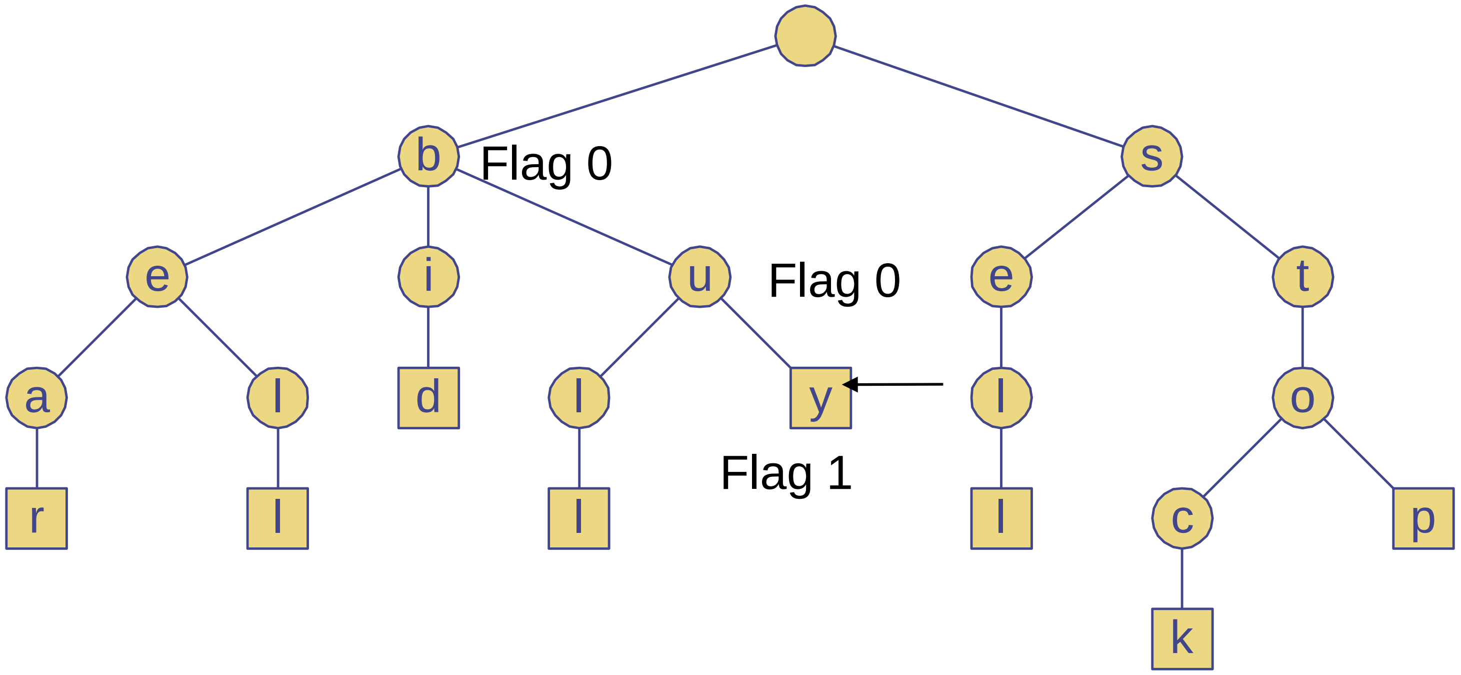
Insert 'buyer'



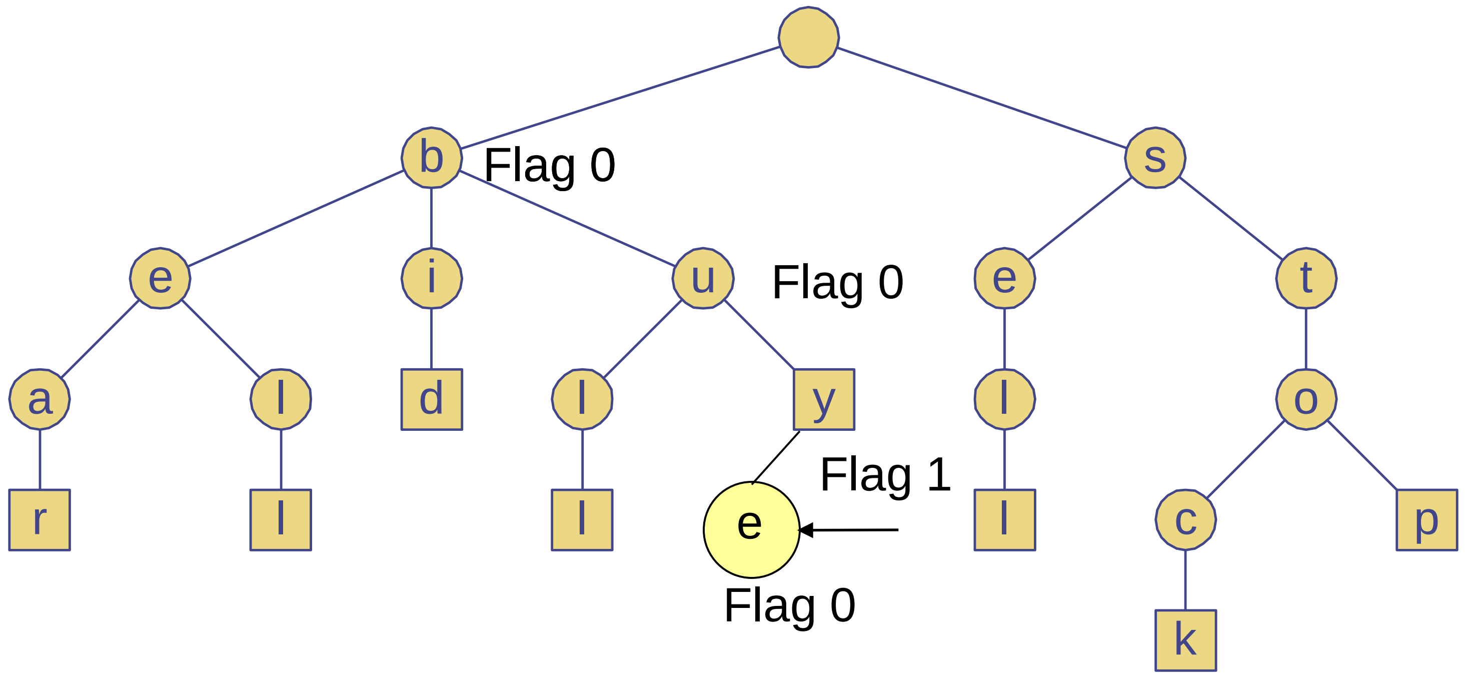
Insert 'buyer'



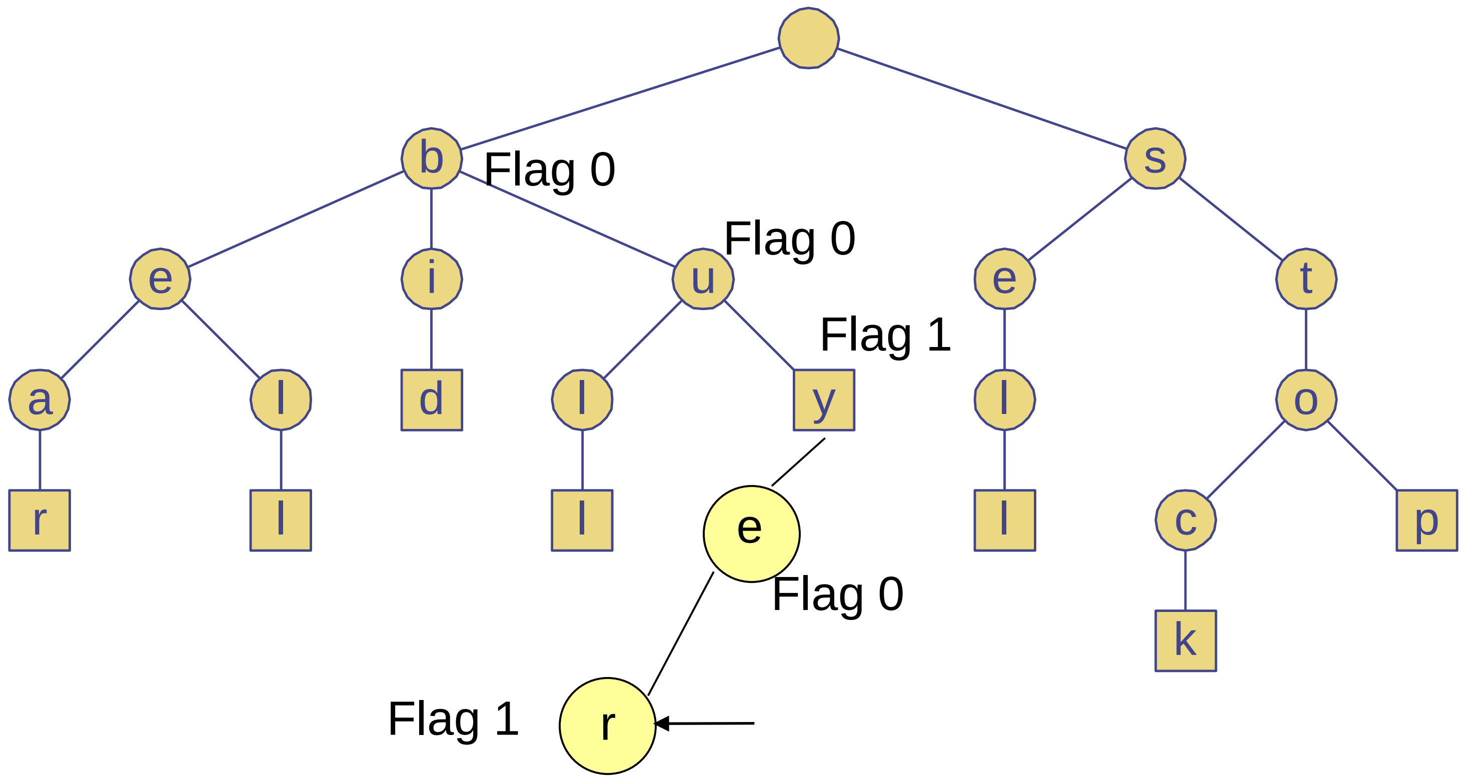
Insert 'buyer'



Insert 'buyer'



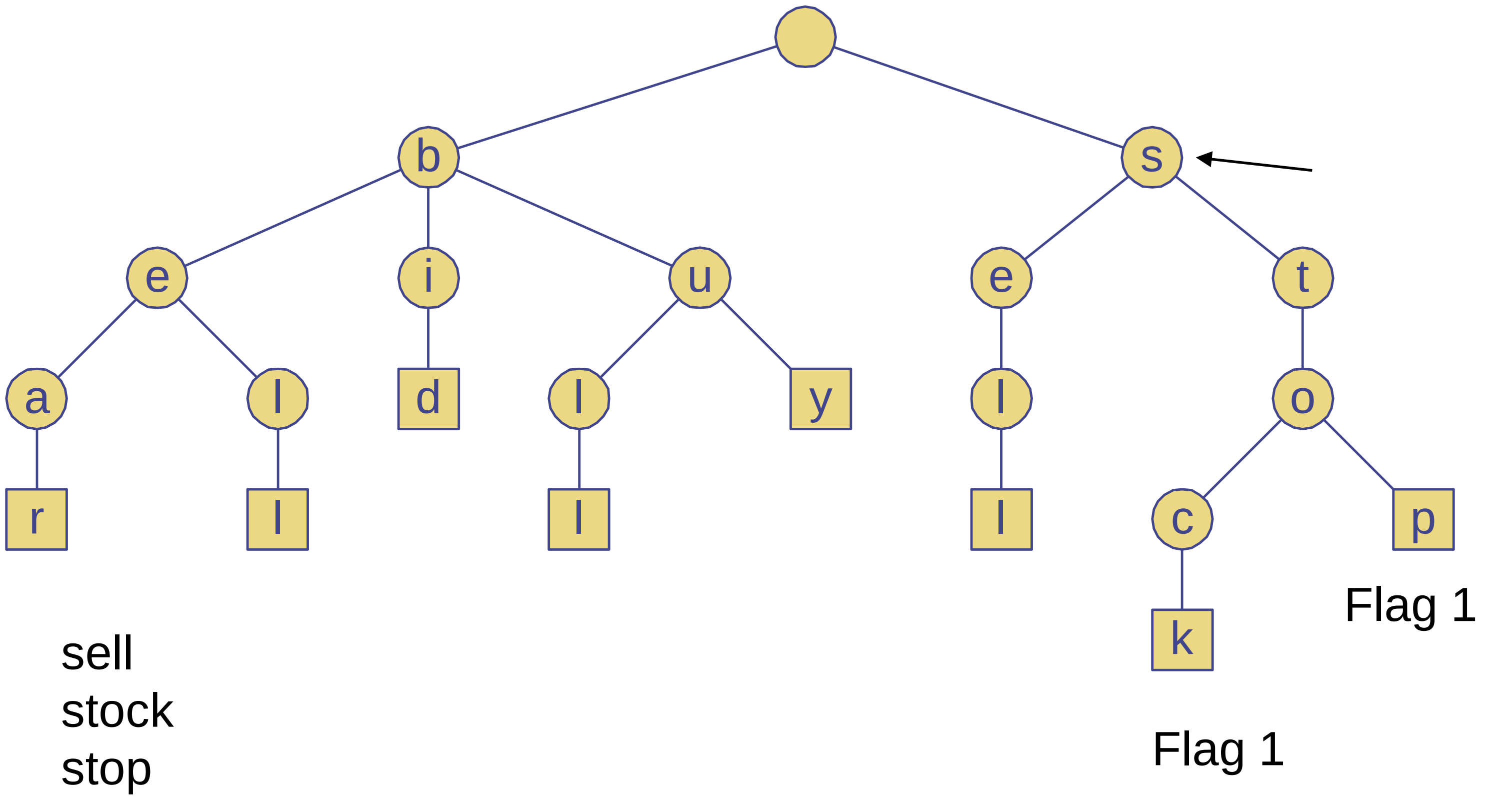
Insert 'buyer'



Searching

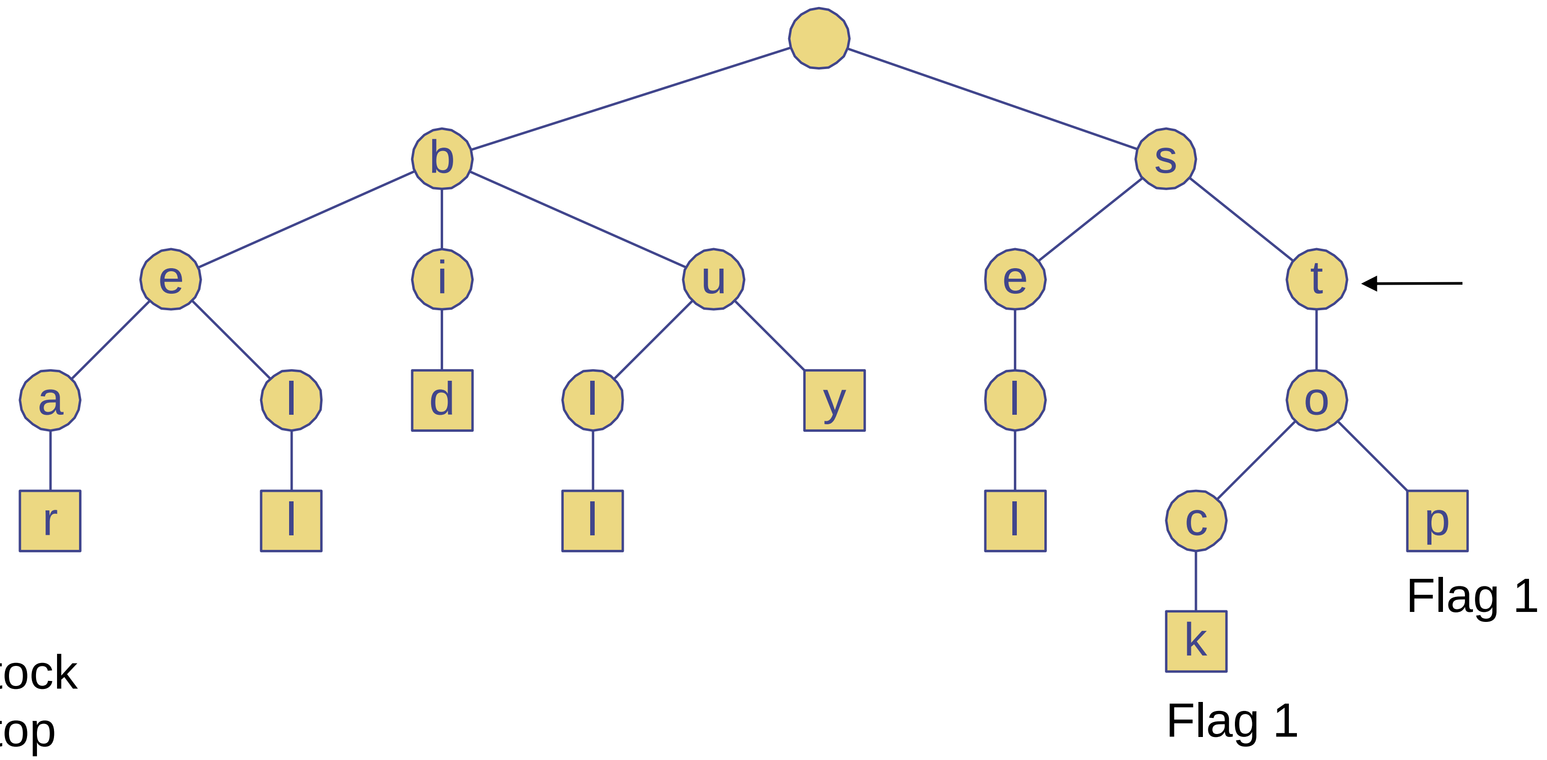
- Again we maintain a pointer initially pointing to root node.
- As a character is read from the input, we look for that child (if it is NULL or not) of current node.
    - If not NULL, then found, so current pointer moves to corresponding child, and is pushed into a stack.
    - Else, not found, so terminate with an error message.

Words starting with 's'



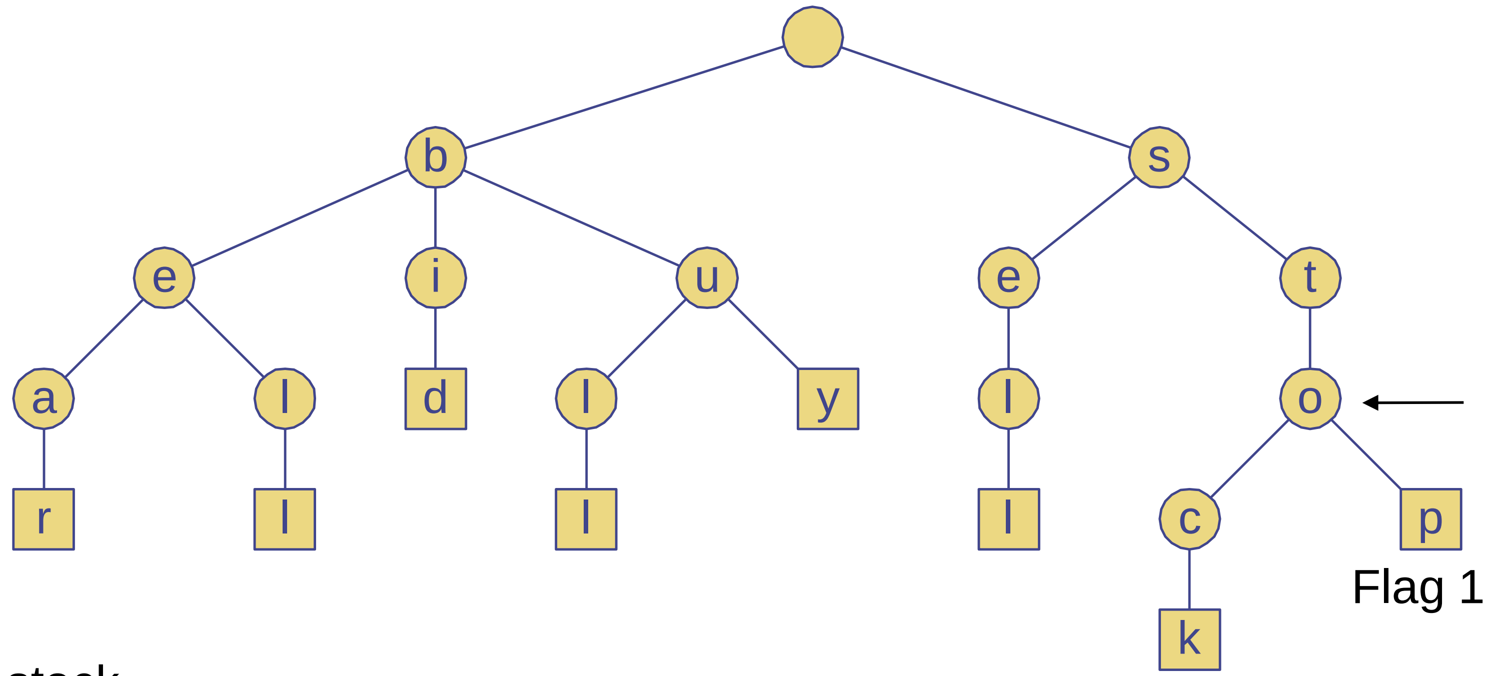
sell
stock
stop

Words starting with 'st'



stock
stop

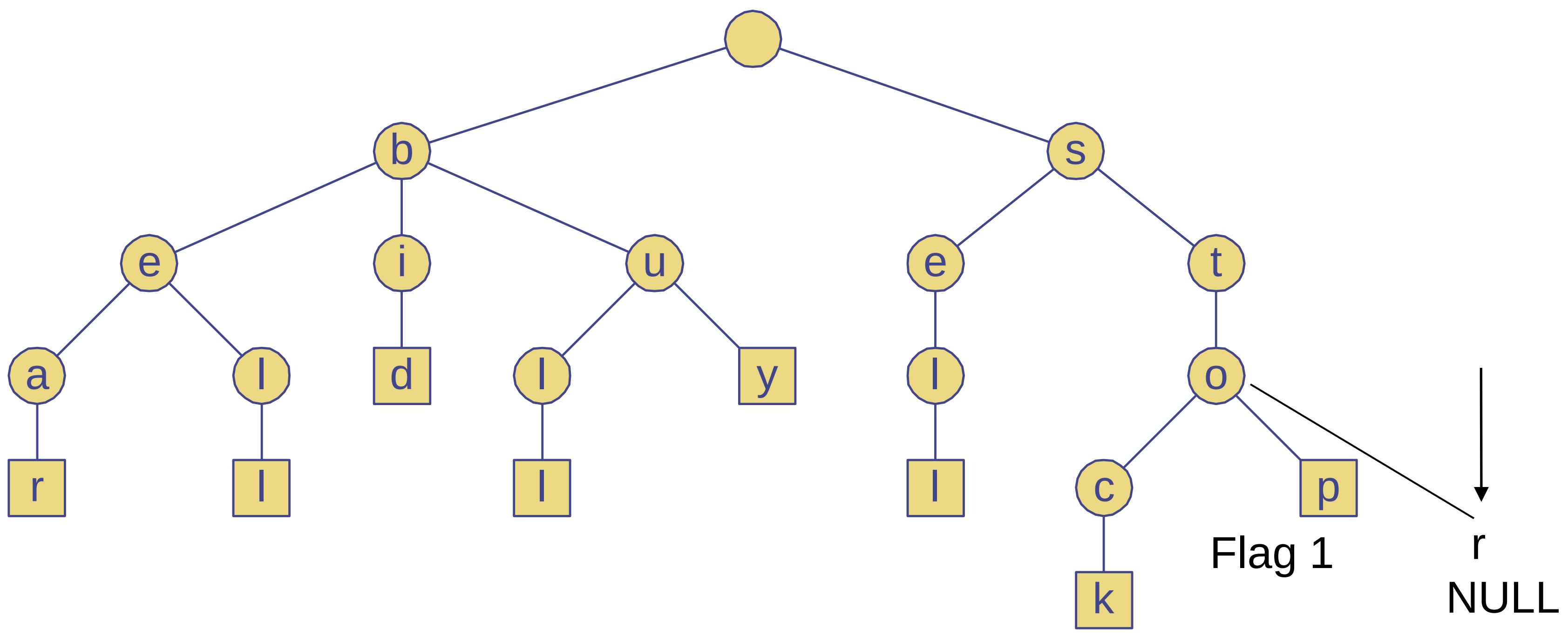
Words starting with 'sto'



Flag 1

stock
stop

Search for words starting with 'stor'



No words starting with "stor"

How does backspace work ?

- Remember, while searching as we crossed a node, we pushed pointer to it into a stack.
- Now is the time to use it !
- Every time we press backspace(<) we pop out of the stack. Thus reaching to the parent node.
- This enables us to perform a search just like a contact list search in a mobile phone.

'Did you mean : ' feature How does it work ?

- We use the same stack to implement this feature as well.
- Any time, user enters a character that on being appended with the word 'str' already entered, gives a word that does not exists in our trie. We pop the stack and print all words with 'str' as prefix.
- These are our suggestions (did you mean) which will be printed on the screen

Trie implementation 2

- Trie's implementation
  - A tree with more than 2 child
  - A node stores the character and an unordered map which stores a pair <char,Node *> i.e. pair which stores the character of the child and the pointer pointing to its node

Trie implementation 2

- The program read from the file a string and insert it till the end of the file is not reached
- Then the user is asked to enter the string which he wants to search
- Then we convert the string entered in lowercase and then pass this string to the find string function

Trie implementation 2

- Insert function inserts the string passed as argument in the trie,all strings are converted to lowercase before entering in the trie
- find_string finds the string passed as argument
  - if string is found then it calls the print function which prints all the strings starting with this string
  - If not found then it calls print function with the substring till which the match was found,it also calls suggestion function with the original string which suggests the string

Trie implementation 2

- Print fnction – this function prints all the strings starting with the string passed as argument
- Suggestion function – this function suggests string similar to the one entered (in case the original string was not found)

Graphical Representation of stack

- In this we would show a graphical stack model where input to the stack would be done by push ,pop operation would remove the last inserted element and top would be showing the last inserted element
- Stack is LIFO implementation

  LIFO - Last In First Out
- All the inputs would be given through the buttons in the graphical window

Graphical Representation of stack

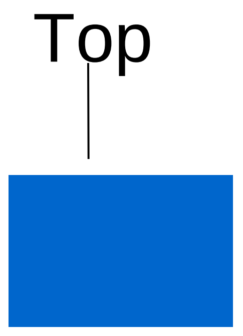
- We have created a window class which would create a window and add all the buttons to it
- In the constructor of this class we have added all the buttons that are needed to the main window and also connected all the buttons with the appropriate data members of renderarea class to store the input

Graphical Representation of stack

- In the RenderArea class we have a switch statement which would check which input is entered and perform the required operation
- For drawing and writing text on the window we have used the QPainter class of the Qt4 library

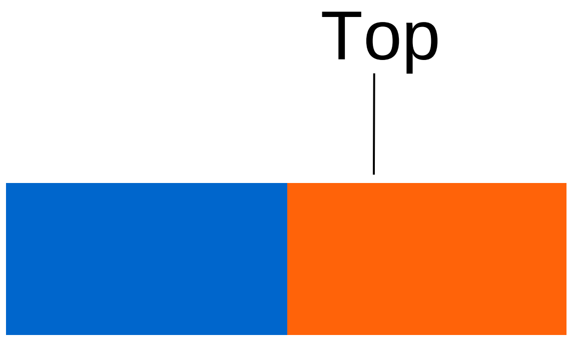
Graphical Representation of stack

Push Operation



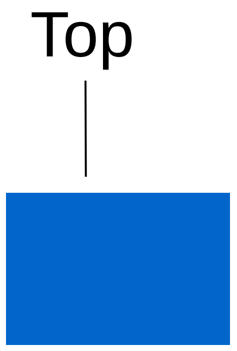
Graphical Representation of stack

Push Operation



Graphical Representation of stack

Pop Operation (Last inserted elemented is removed)



Graphical Representation of Queue

- In this we would show a graphical queue model where input to the queue would be done by enqueue , dequeue operation would remove the first inserted element and front would be showing the first inserted element and back would show the last inserted element
- Queue is a FIFO (First In First Out) implementation

Graphical Representation of Queue

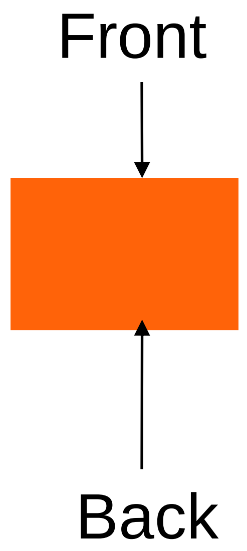
- All the inputs would be given through the buttons in the graphical window
- We have created a window class which would create a window and add all the buttons to it
- In the constructor of this class we have added all the buttons that are needed to the main window and also connected all the buttons with the appropriate data members of renderarea class to store the input

Graphical Representation of queue

- In the RenderArea class we have a switch statement which would check which input is entered and perform the required operation
- For drawing and writing text on the window we have used the QPainter class of the Qt4 library

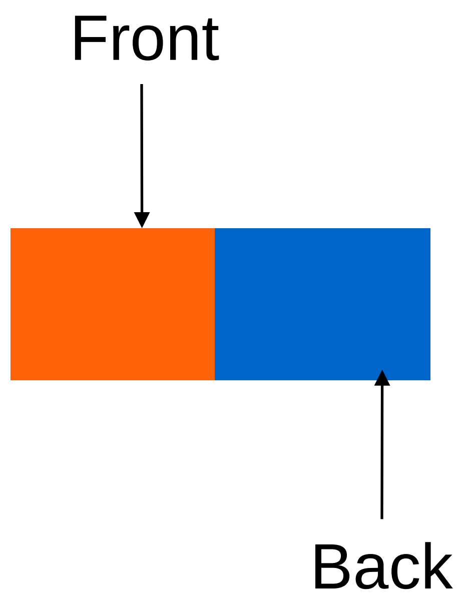
Graphical Representation of Queue

Enqueue



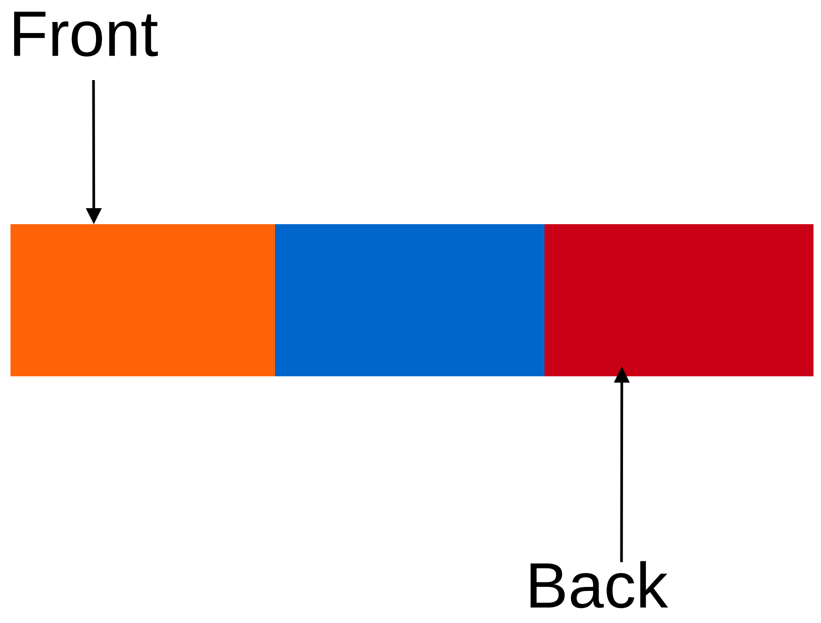
Graphical Representation of Queue

Enqueue



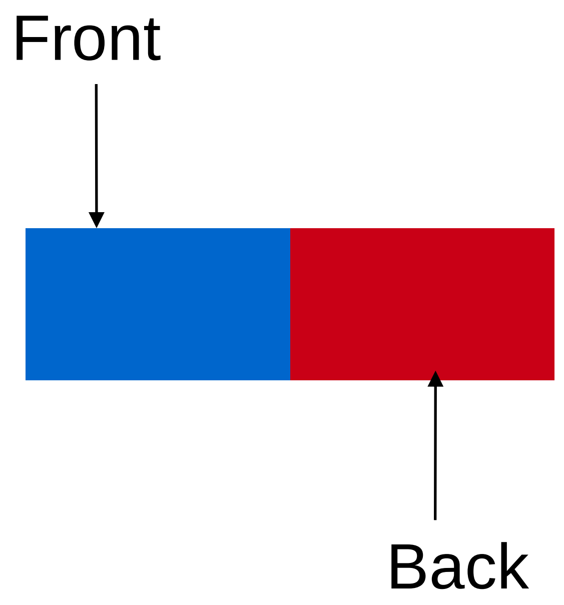
Graphical Representation of Queue

Enqueue



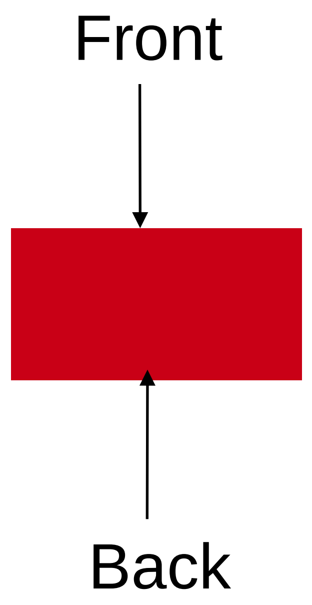
Graphical Representation of Queue

Dequeue



Graphical Representation of Queue

Dequeue



Graphical Representation of list

- In this we would show a graphical list ADT where input to the list would be done by push_back & push_front , pop_back and pop_front would remove the last and the first element respectively ,delete will remove the entered element from the list and insert after and insert before would insert the new element after or before the given element respectively

Graphical Representation of list

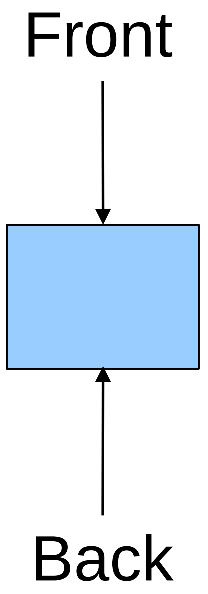
- All the inputs would be given through the buttons in the graphical window
- We have created a window class which would create a window and add all the buttons to it
- In the constructor of this class we have added all the buttons that are needed to the main window and also connected all the buttons with the appropriate data members of renderarea class to store the input

Graphical Representation of list

- In the RenderArea class we have a switch statement which would check which input is entered and perform the required operation
- For drawing and writing text on the window we have used the QPainter class of the Qt4 library

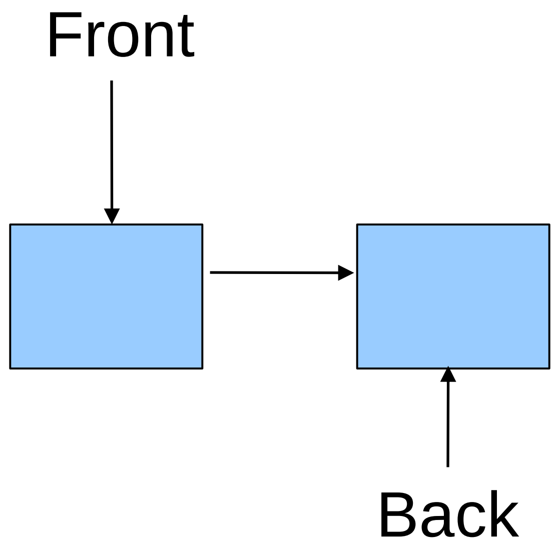
Graphical Representation of list

push_back



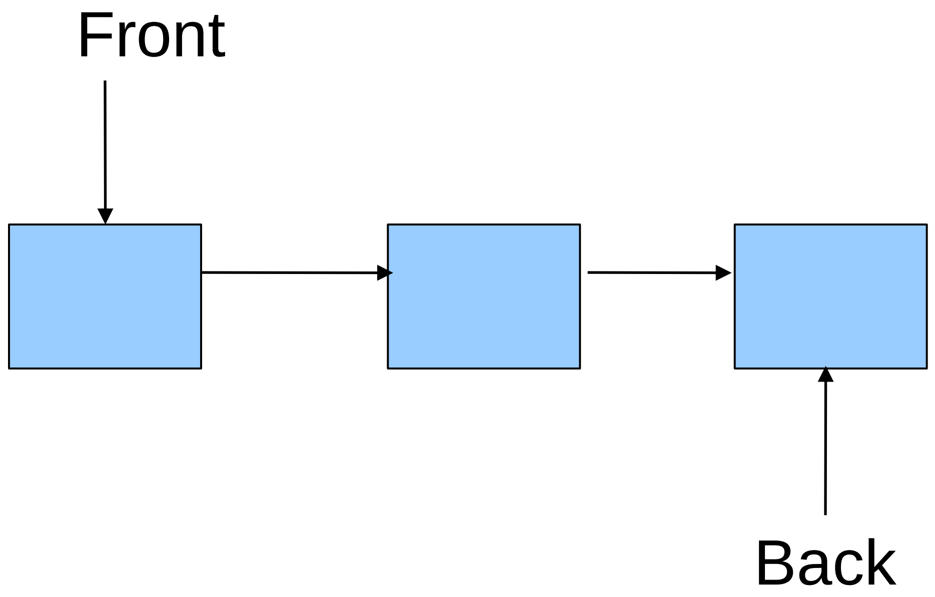
Graphical Representation of list

push_back



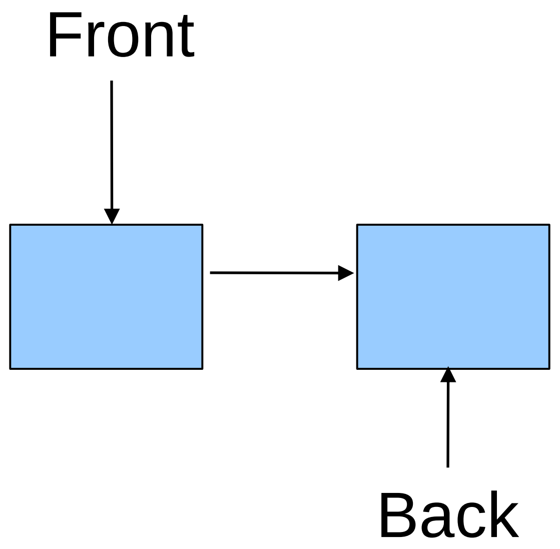
Graphical Representation of list

push_front



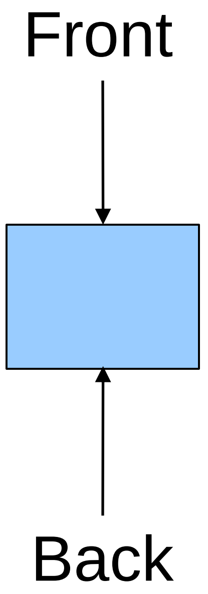
Graphical Representation of list

pop_front



Graphical Representation of list

pop_back



Graphical Representation of Binary Search Tree

- In this we would show a graphical binary search tree where input to the tree would be given by insert , remove would remove the given element , find_maximum and find_minimum would search the maximum and minimum elements respectively.

Graphical Representation of Binary Search Tree

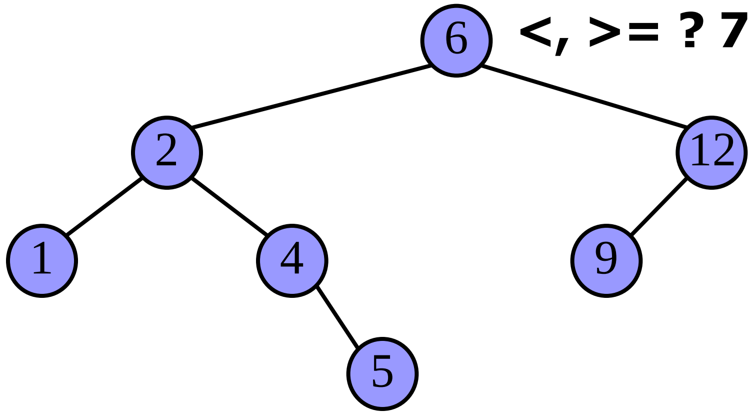
- All the inputs would be given through the buttons in the graphical window
- We have created a window class which would create a window and add all the buttons to it
- In the constructor of this class we have added all the buttons that are needed to the main window and also connected all the buttons with the appropriate data members of renderarea class to store the input

Graphical Representation of Binary Search Tree

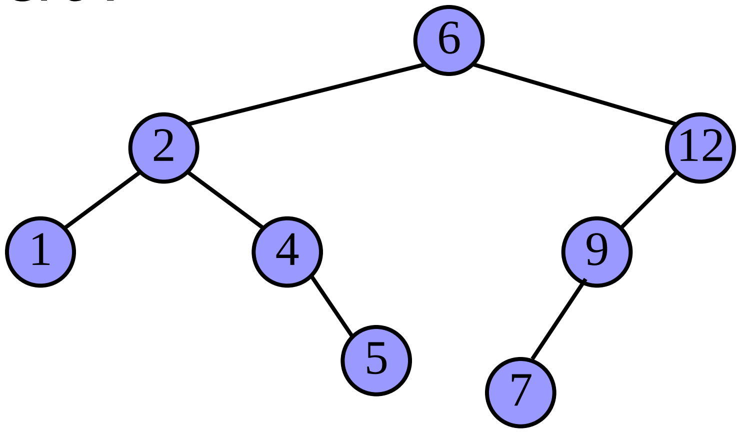
- In the RenderArea class we have a switch statement which would check which input is entered and perform the required operation
- For drawing and writing text on the window we have used the QPainter class of the Qt4 library

Graphical Representation of Binary Search Tree

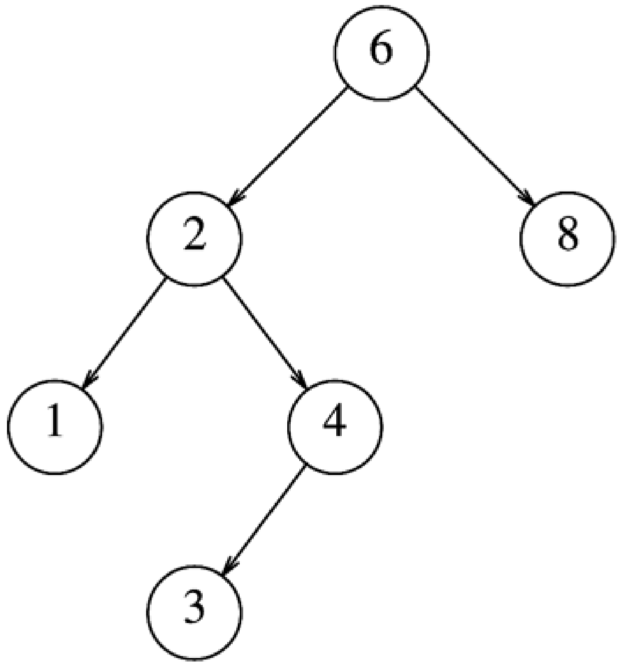
Insert 7

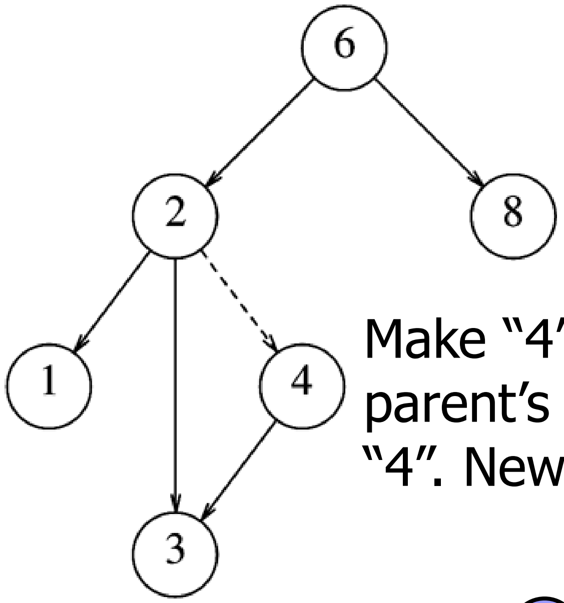


Insert 7

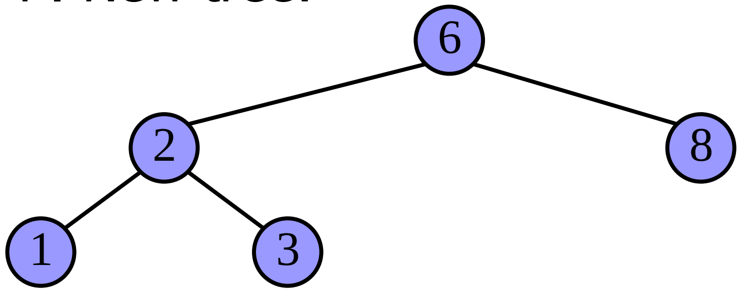


Deletion node with one child



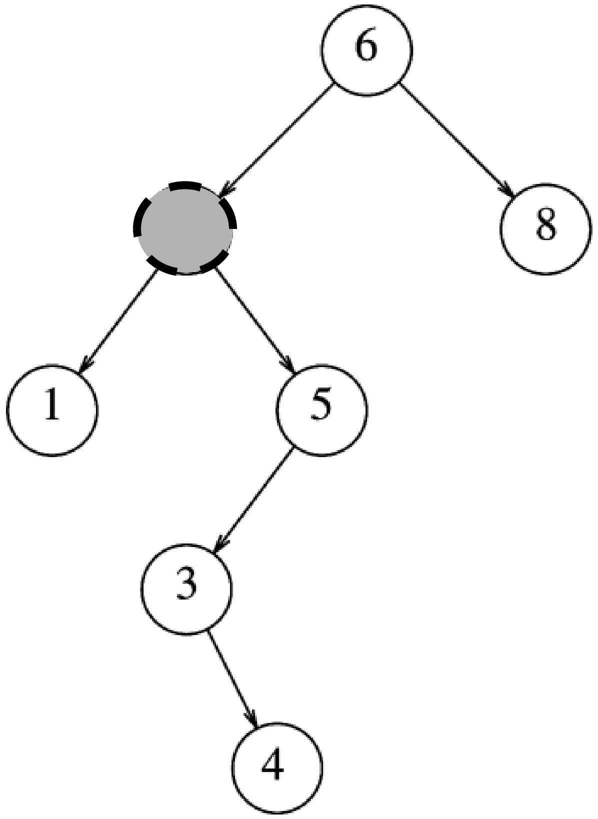


Make "4"'s child, it's parent's child, delete "4". New tree:



Delete 4

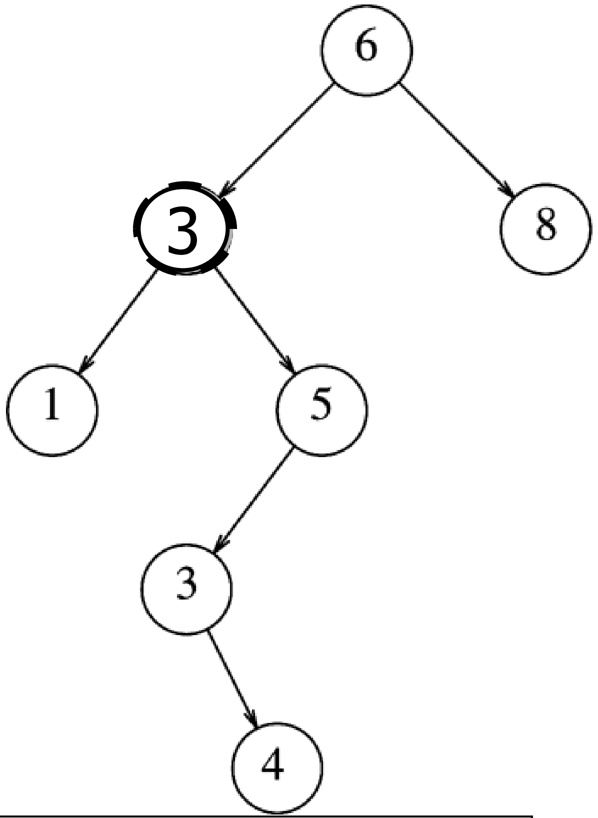
Deletion – node with two children



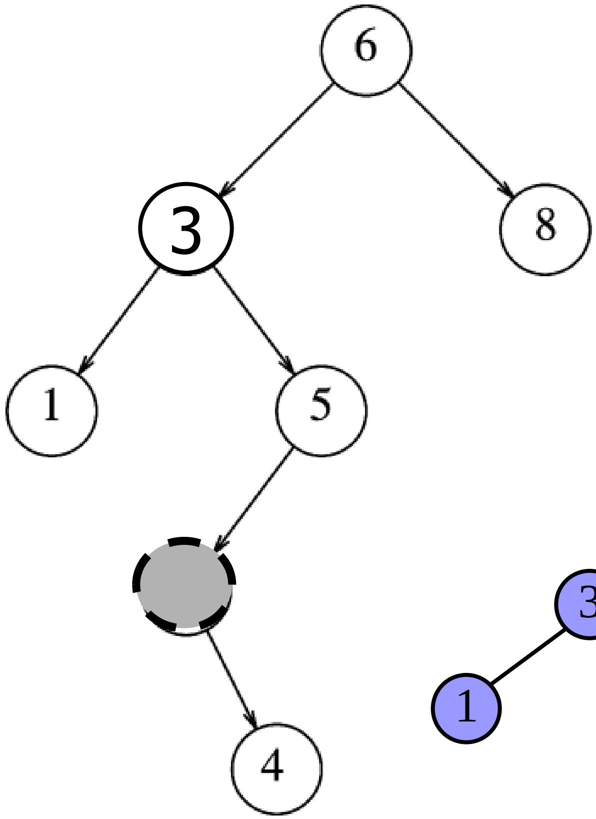
Either largest of left subtree, or smallest of right subtree, will satisfy this property. Move it to the "hole" and then recursively delete that element

Delete "2"

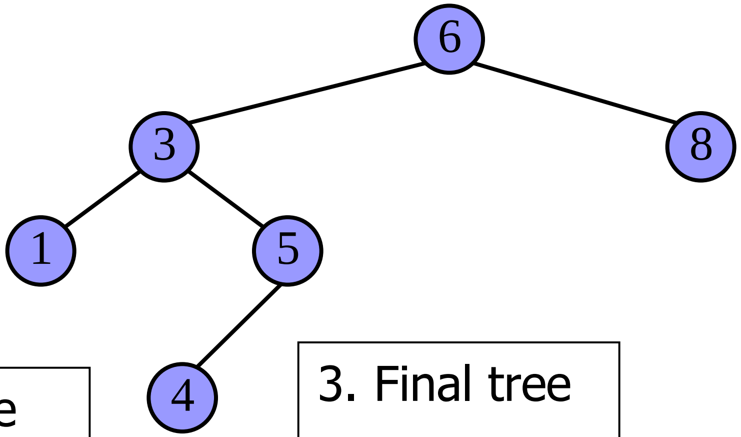
Deletion – node with two children



1. Find smallest element of right subtree, move it to the deleted node place



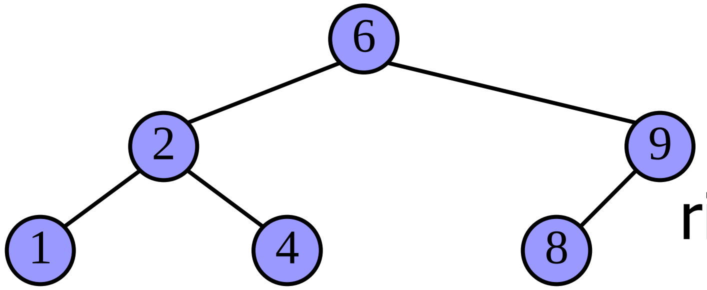
2. Recursively delete that element



3. Final tree

Graphical Representation of Binary Search Tree

Findmin, Findmax



rightmost node: max

Leftmost node: min

Teamwork Details

- **Abhishek Gupta**: Did major part of implementation in huffman coding, and also did Trie-2 Implementation
- **Mridul Ravi Jain**: Did complete Trie-1 implentation and helped in huffman coding.
- **Both** did an equal contribution to Graphical implementation of data structures, with Mridul coming up with the stack implementation and then both together advanced it to queue, list and BST

Overall Contribution by Abhishek	**Overall Contribution By Mridul**
52%	48%

Class Design – Huffman coding

- We have used a vector of size 256 for storing the frequency of the characters , so extra memory is wasted as there may be only 30-40 distinct character in a text file
- Before writing to the file we store the whole text in the form of their new codes in a string and then write the string in the file in a compressed form, so extra space is wasted.

Class Design – Trie-1 Implementation

- In our first implementation of trie, a node is described by a character element, a flag and 26 pointers for its children.
- Although it is expensive in terms of space usage, but very efficient in time.
- Because at node we directly look for the particular child, thus O(1) time.
- Also this design is more extendible.

Class Design – Trie-2 Implementation

- In order to come up with a more efficient space algorithm we did another trie implementation that uses an unordered map for each node
- However it is a liitle expensive in terms of time usage, but very efficient in space.
- At every node, finding a child is
  - O(constant) : average
  - O(linear) : worst case

Window Class Design – Graphical implementation of data structures

- We declare the various widgets,
- In the constructor we create and initialize the various widgets appearing in the main application window.
- Then we connect the parameter widgets with their associated slots using the static QObject::connect() function, ensuring that the RenderArea widget is updated whenever the user changes any of the parameters.
- Finally, we add the various widgets to a layout, and initialize the application.

RenderArea Class Design – Graphical implementation of data structures

- In the constructor we initialize some of the widget's variables.
- Next we perform the required operation as instruction is given by the user, this is done by paintEvent function.
- So, push,pop, pop_front, pop_back are treated as an event and accordingly a we construct different shapes.
- However, this event mechanism is O(n) as everytime an element is deleted or added to the data structure, we need to again construct all the previous elements also.

Class Design - Details

Class Name	Brief Description
Window Class	This class is basically used to create a window and all the required buttons to it. For eg :- it creates a window on which we would draw the lines or write the text, then all the buttons like combobox, spinbox etc are created and displayed on the main window using this class This class also connects the buttons added on the main window for input with the data members of the other classes where input is used

Class Design - Details

Class Name	Brief Description
RenderArea	This class is basically used to take the input given by the user through the buttons in the main window and process the input and perfom the operation For Eg:- if I write 5 and then give insert command then it will first store the input in its variable then it will process the command and then draw the final structure After processing it draws the new modified data structure

Data Structures Used in Trie-1

Purpose for which data structure is used	Data Structure Used	Whether Own Implementation or STL
Storing the order of search	Stack	STL

Data Structures Used in Trie-2

Purpose for which data structure is used	Data Structure Used	Whether Own Implementation or STL
Storing the list of children	unordered_map	STL

Data Structures Used in Huffman Coding

Purpose for which data structure is used	Data Structure Used	Whether Own Implementation or STL
Creating the huffman tree	Priority_queue	STL
Storing the frequency of character in the text file	vector	STL
For storing new huffman codes of characters	vector	STL

Source Code Information Trie-1

File Name	Brief Description	Author (Team Member)
trie.h	Declares trie class and methods	Mridul
trie.cpp	Implements trie class methods	Mridul
main.cpp	Main file to start execution & perform insertion and searching	Mridul
input1.txt,input2.txt	Input file	

Source Code Information Trie-2

File Name	Brief Description	Author (Team Member)
trie2.cpp	Complete implementation of trie using unordered map	Abhishek
input.txt	Input file	

Source Code Information Huffman coding

File Name	Brief Description	Author
huffman_write.cpp	Creates a compressed file from the original text file	Abhishek
huffman_read.cpp	Creates the original text file from the compressed file	Abhishek
input1.txt	Input file	
input2.txt	Input file	

Source Code Information Graphical representation

File Name	Brief Description	Author
Windows.h	Declares window class & its methods (common to list, queue, stack, BST)	Mridul & Abhishek
Windows.cpp	Implements Window class methods (common to list, queue, stack, BST)	Mridul & Abhishek

The Window class inherits QWidget, and is the application's main window displaying a RenderArea widget in addition to several parameter widgets.

Source Code Information Graphical representation

File Name	Brief Description	Author
RenderArea.h	Declares RenderArea class & its methods (common to list, queue, stack, BST with slight modifications)	Mridul & Abhishek
RenderArea.cpp	Implements RenderArea class methods (common to list, queue, stack, BSTwith slight modifications)	Mridul & Abhishek

The RenderArea class inherits QWidget, and renders shapes with given parameters at any event.

Source Code Information Graphical representation

File Name	Brief Description	Author
main.cpp	Main file that intantiates a Window object, draws the window on screen, and starts program execution	Mridul & Abhishek
iitbOrderedSet.h	Implements the binary search tree which is used in the graphical implementation	Mridul

Brief Conclusion

- We learnt a lot about the graphics library qt4 which was one of the exciting part of the assignment
- Writng to the file in bitwise was also a big challenge ,but thanks to T.A. Ashish who helped us in sorting it out
- In Trie we had to face trade-offs and thus we submitted 2 implementations

Thank You – Questions?