Data compression

anhtt-fit@mail.hut.edu.vn

Data Compression

- Data in memory have used fixed length for representation
- For data transfer (in particular), this method is inefficient.
- For speed and storage efficiencies, data symbols should use the minimum number of bits possible for representation.
- Methods Used For Compression:
  - Encode high probability symbols with fewer bits
    - Shannon-Fano, Huffman, UNIX compact
  - Encode sequences of symbols with location of sequence in a dictionary
    - PKZIP, ARC, GIF, UNIX compress, V.42bis
  - Lossy compression
    - JPEG and MPEG

Variable Length Bit Codings

- Suppose ‘A’ appears 50 times in text, but ‘B’ appears only 10 times
- ASCII coding assigns 8 bits per character, so total bits for ‘A’ and ‘B’ is 60 * 8 = 480
- If ‘A’ gets a 4-bit code and ‘B’ gets a 12-bit code, total is 50 * 4 + 10 * 12 = 320

Compression rules:

- Use minimum number of bits
- No code is the prefix of another code
- Enables left-to-right, unambiguous decoding

Variable Length Bit Codings

- No code is a prefix of another
  - For example, can't have 'A' map to 10 and 'B' map to 100, because 10 is a prefix (the start of) 100.
- Enables left-to-right, unambiguous decoding
  - That is, if you see 10, you know it's 'A', not the start of another character.

Huffman code

- Constructed by using a code tree, but starting at the leaves
- A compact code constructed using the binary Huffman code construction method

Huffman code Algorithm

① Make a leaf node for each code symbol
   - Add the generation probability of each symbol to the leaf node

② Take the two leaf nodes with the smallest probability and connect them into a new node
   - Add 1 or 0 to each of the two branches
   - The probability of the new node is the sum of the probabilities of the two connecting nodes

③ If there is only one node left, the code construction is completed. If not, go back to (2)

Demo

- 65demo-huffman.ppt

Compress a text

- Consider the following short text:

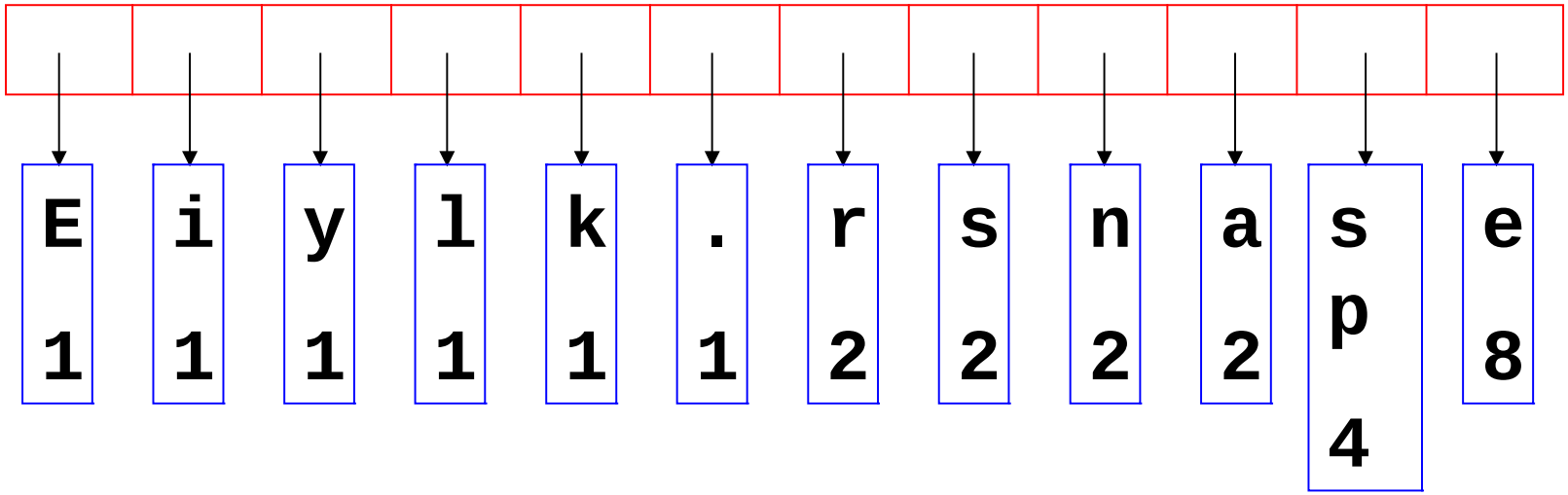
  Eerie eyes seen near lake.

- Count up the occurrences of all characters in the text

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

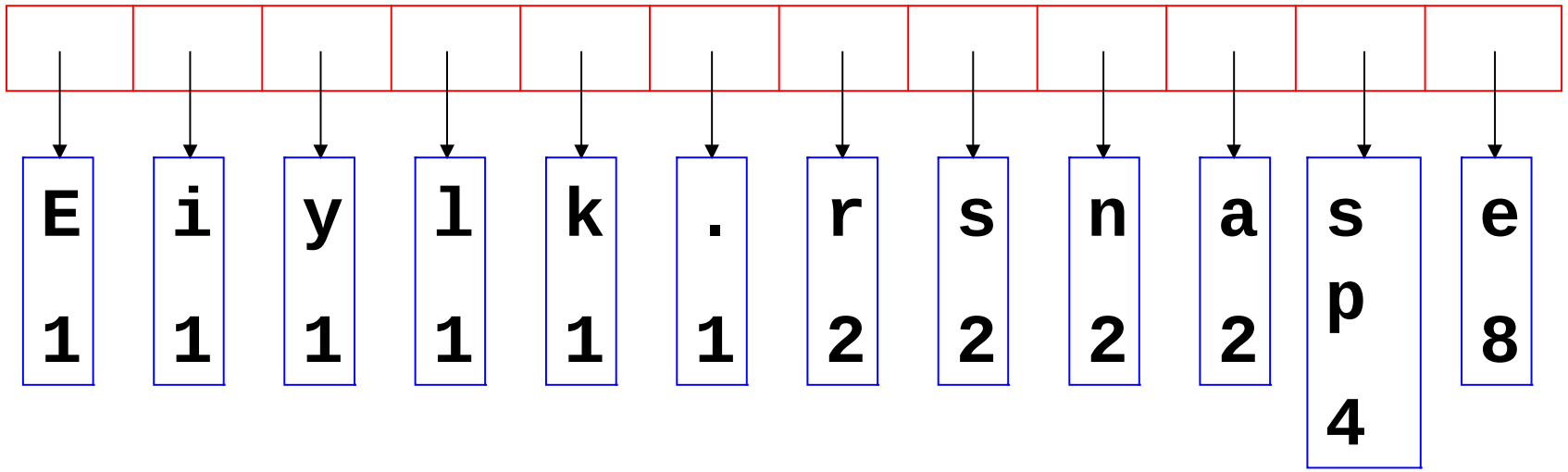


- Null Pointers are not shown

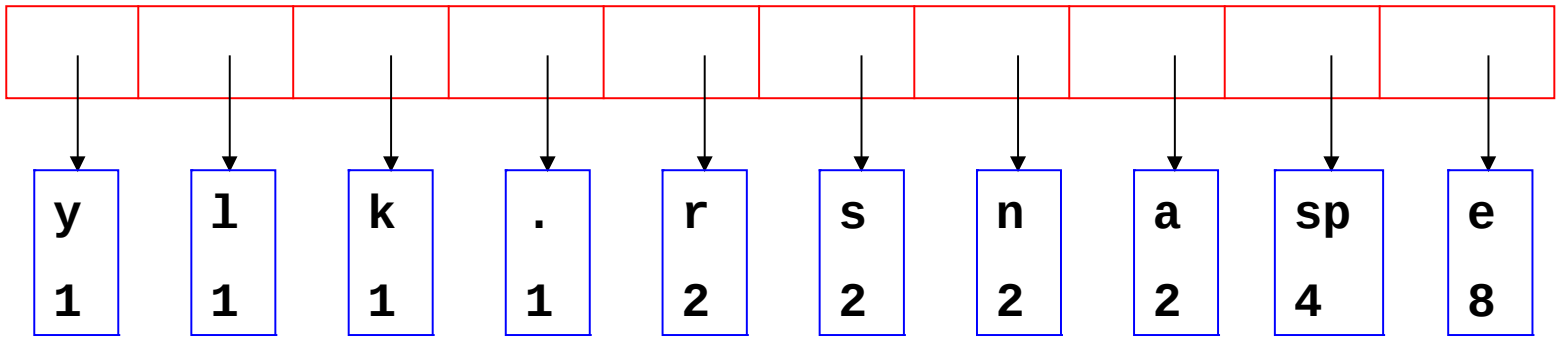
Building a Tree

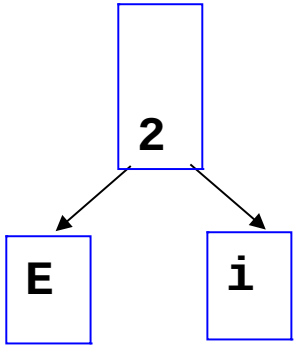
- While priority queue contains two or more nodes
  - Create new node
  - Dequeue node and make it left subtree
  - Dequeue next node and make it right subtree
  - Frequency of new node equals sum of frequency of left and right children
  - Enqueue new node back into queue

Building a Tree

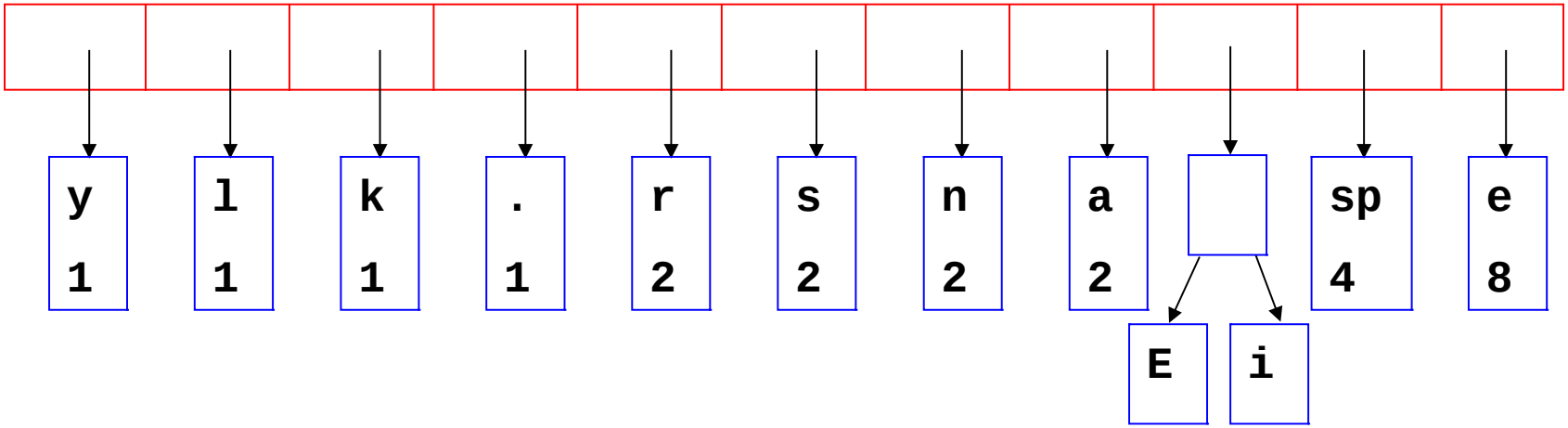


Building a Tree

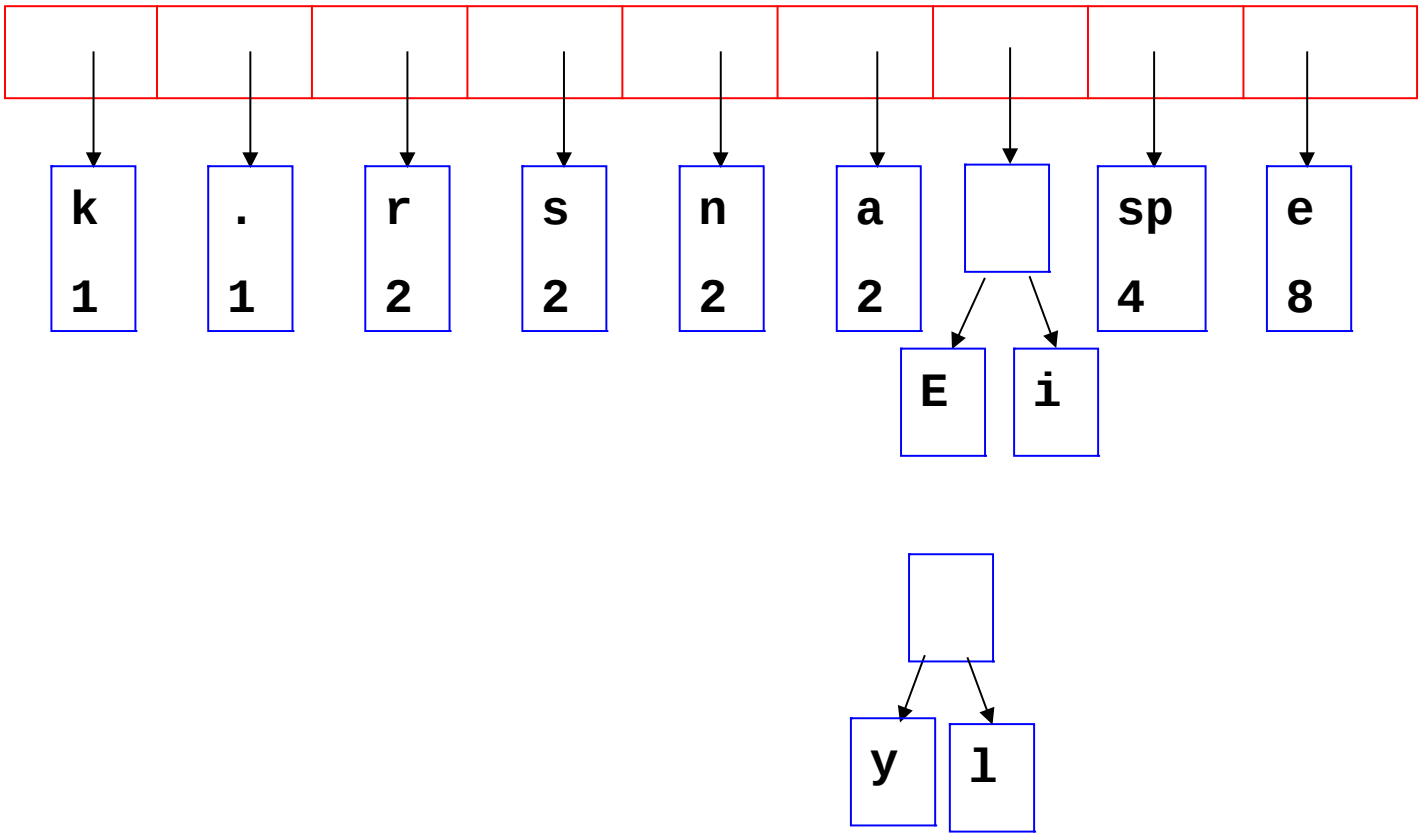




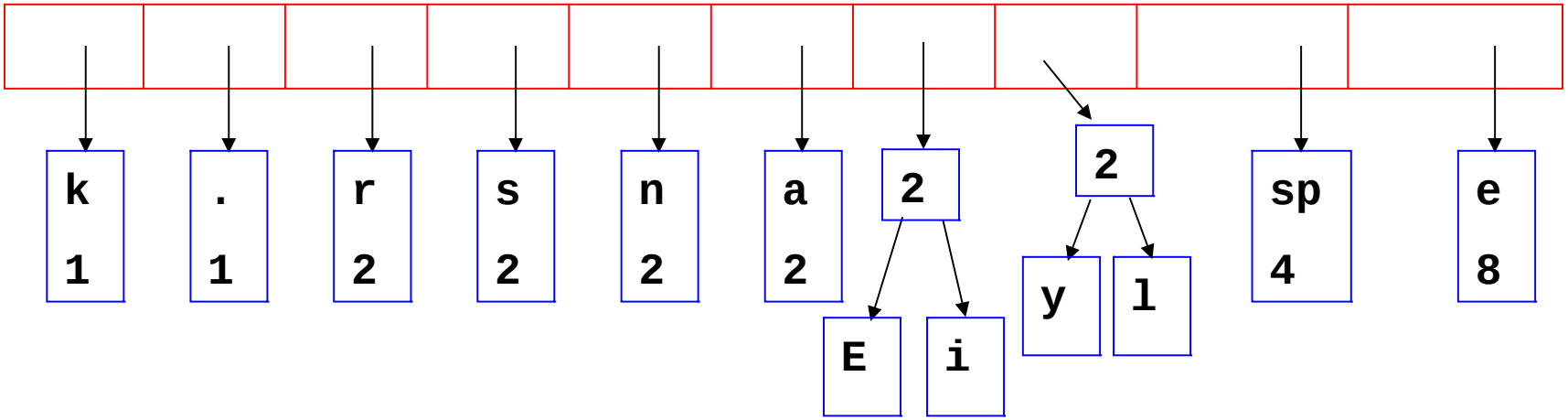
Building a Tree



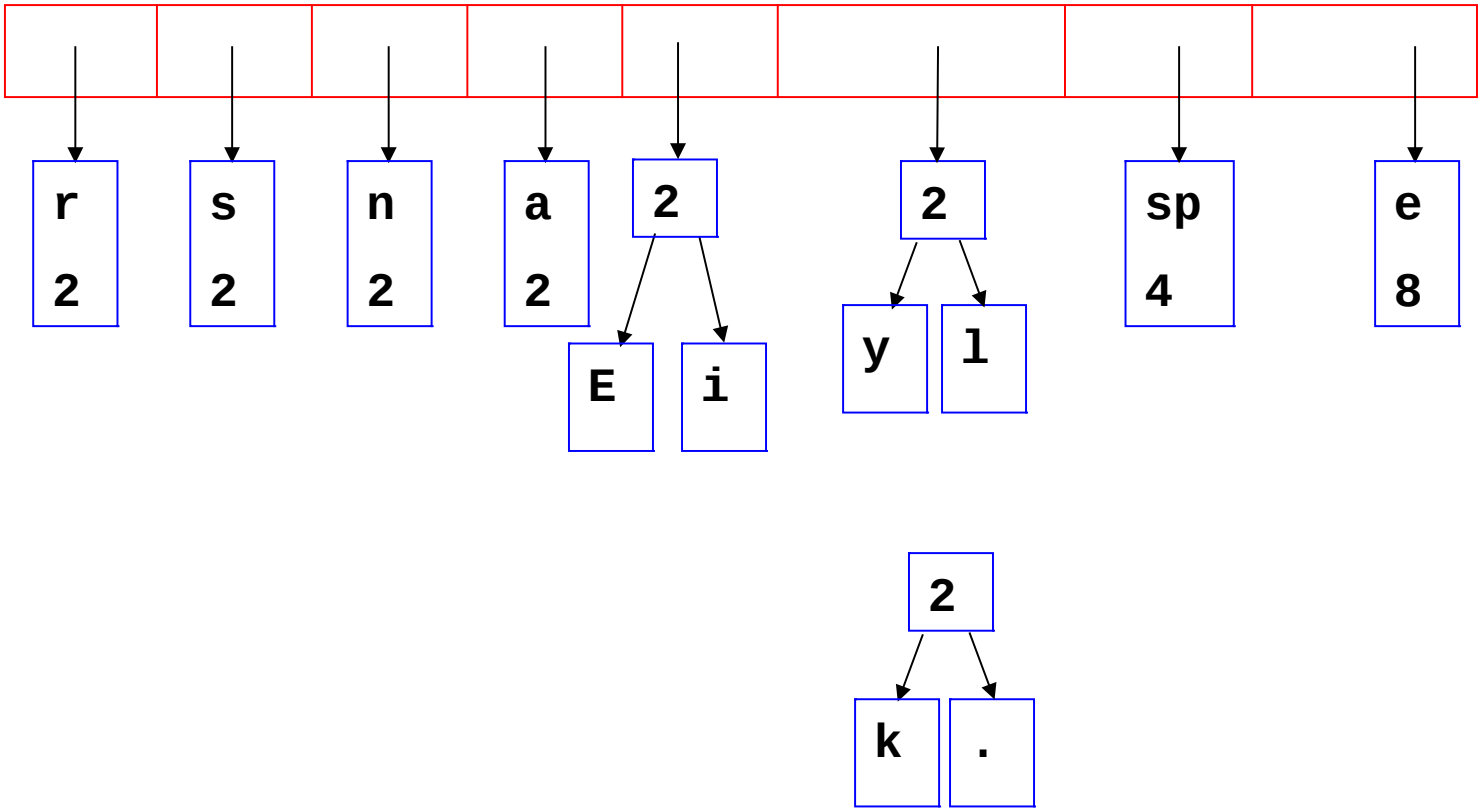
Building a Tree



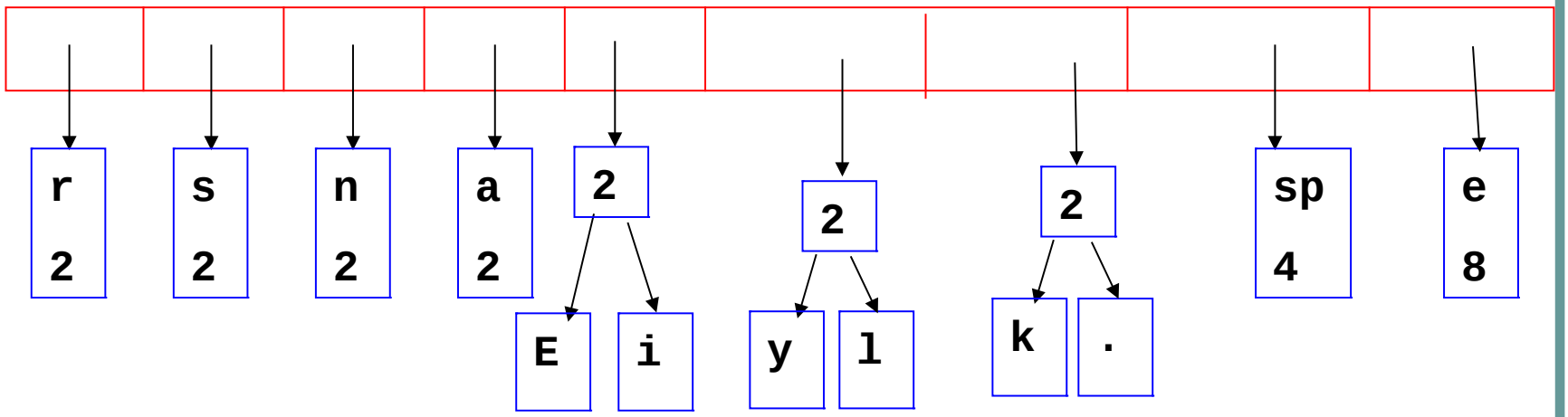
Building a Tree



Building a Tree

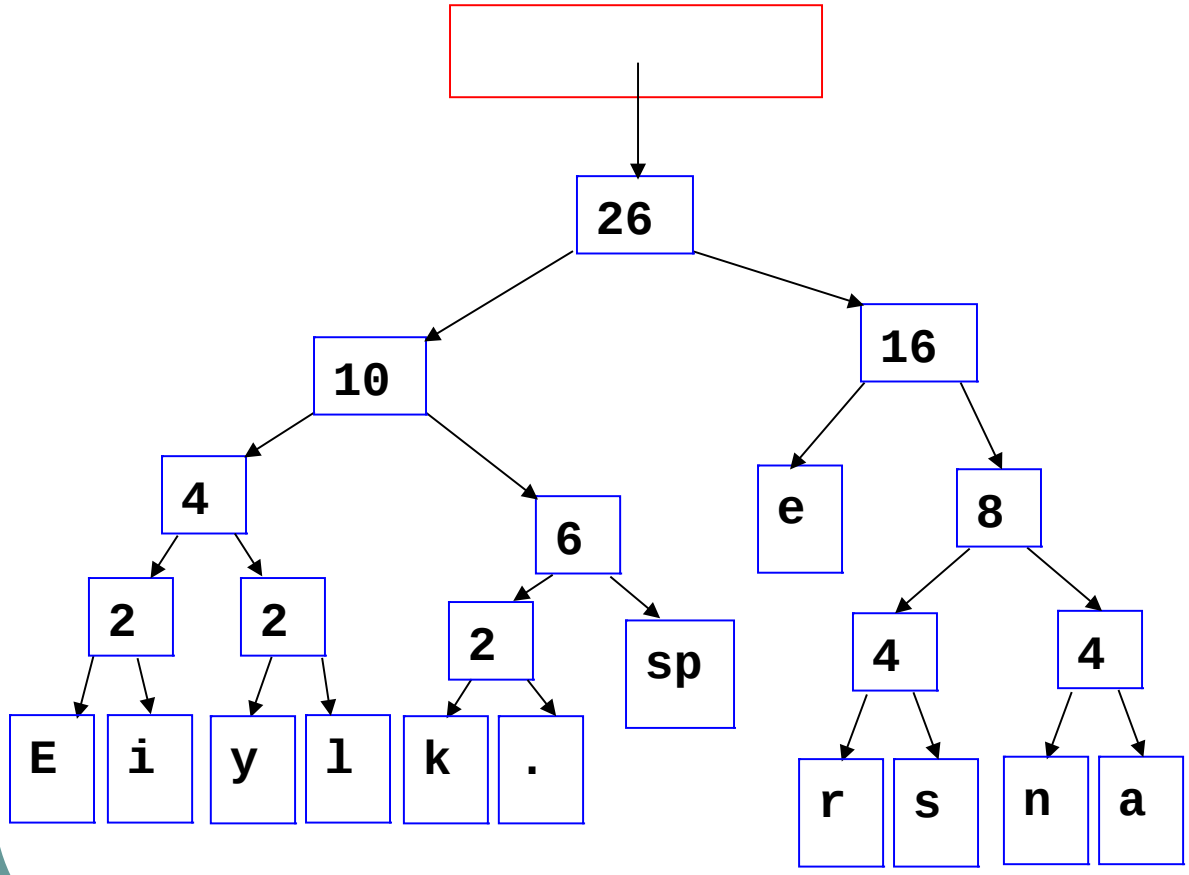


Building a Tree



- To continue …

At the end



After enqueueing this node there is only one node left in priority queue.

How to implement ?

- Reuse JRB to represent the tree
  - Each new node is created as a JRB node
  - The edges are directional from the parents to the children.
  - Two edges are created and marked using label 0 or 1 when a parent node is created.
- Reuse Dllist or JRB to represent the priority queue
  - A queue node contains a key as the frequency of the related node in the tree
  - The queue node’s value is a pointer referencing to the node in the tree

Quiz 1

- Reuse the graph API defined in previous class to write a function that builds a Huffman tree from a string as the following

```
typedef struct {
   Graph graph;
   JRB root;
} HuffmanTree;
HuffmanTree makeHuffman (char * buffer, int size);
```

Huffman code table

- In order to compress the data string, we have to build a code table from the Huffman tree. The following data structure is used to represent the code table

```
typedef struct {
    int size;
    char bits[2];
} Coding;
Coding huffmanTable[256];
```

- huffmanTable['A'] give the coding of 'A'. If the coding's size = 0, the character 'A' is not present in the text. bits contains the huffman code (sequence of bits) of the given character.

Quiz 2

- Write a function to create the Huffman code table from a Huffman tree
  - void createHuffmanTable(HuffmanTree htree, Coding* htable);
- Write a function to compress a text buffer to a Huffman sequence.
  - void compress(char * buffer, int size, char* huffman, int* nbit);
- The buffer contains *size* characters. After compressing, the huffman buffer contains nbit bits for output.
- In order to write this function, you should create a function to add a new character into the huffman buffer as the following
  - void addHuffmanChar(char * ch, Coding* htable, char* huffman, int* nbit);