Nén dữ liệu

Nén dữ liệu

- Dữ liệu trong bộ nhớ sử dụng dung lượng cố định để biểu diễn
- Để truyền dữ liệu, phương pháp biểu diễn này không hiệu quả.
- Nhằm nâng cao tốc độ và khả năng lưu trữ, các ký hiệu dữ liệu nên sử dụng số lượng bit ít nhất có thể để biểu diễn.
- Các phương pháp sử dụng để nén:
  - Mã hóa các kí hiệu phổ biến với ít bit
    - Shannon-Fano, Huffman, UNIX compact
  - Mã hóa chuỗi các kí hiệu với vị trí của chuỗi lưu trong một từ điển
    - PKZIP, ARC, GIF, UNIX compress, V.42bis
  - Nén mất mát
    - JPEG và MPEG

Mã hóa bit độ dài thay đổi

- Giả sử ‘A’ xuất hiện 50 lần trong văn bản, nhưng ‘B’ chỉ xuất hiện 10 lần
- Mã ASCII cung cấp 8 bit cho mỗi kí tự, ví thế tổng số bit cho ‘A’ và ‘B’ là 60 * 8 = 480
- Nếu ‘A’ dùng mã 4-bit và ‘B’ dùng mã 12-bit, thì tổng số bit là 50 * 4 + 10 * 12 = 320

Quy tắc nén:

- Sử dụng ít số bit nhất
- Không mã nào là tiền tố của mã khác
- Cho phép giải mã không nhập nhằng từ trái-qua-phải

Mã hóa bit độ dài thay đổi

- Không mã nào là tiền tố của mã khác
  - VD: không mã hóa ‘A’ thành 10 và ‘B’ thành 100, bởi vì 10 là tiền tố của 100.
- Cho phép giải mã không nhập nhằng từ trái-qua-phải
  - Nếu ta quan sát thấy 10, ta biết đó là ‘A’, không phải là phần bắt đầu của kí tự khác.

Mã Huffman

- Tạo ra bởi một cây mã hóa, nhưng bắt đầu từ lá
- Mã nhị phân Huffman

Giải thuật mã hóa Huffman

① Tạo một nút lá cho mỗi mã kí hiệu
   - Thêm vào xác suất sinh ra của mỗi kí hiệu vào nút lá

② Lấy hai nút lá có xác suất nhỏ nhất và tạo một nút mới
   - Thêm 1 hoặc 0 vào mỗi nhánh
   - Xác suất của nút mới tạo là tổng xác suất của hai nút thành phần

③ Nếu chỉ còn một nút, quá trình xây dựng mã hoàn thành. Nếu không, quay lại (2)

Demo

- 65demo-huffman.ppt

Nén một văn bản

- Xét văn bản ngắn sau:

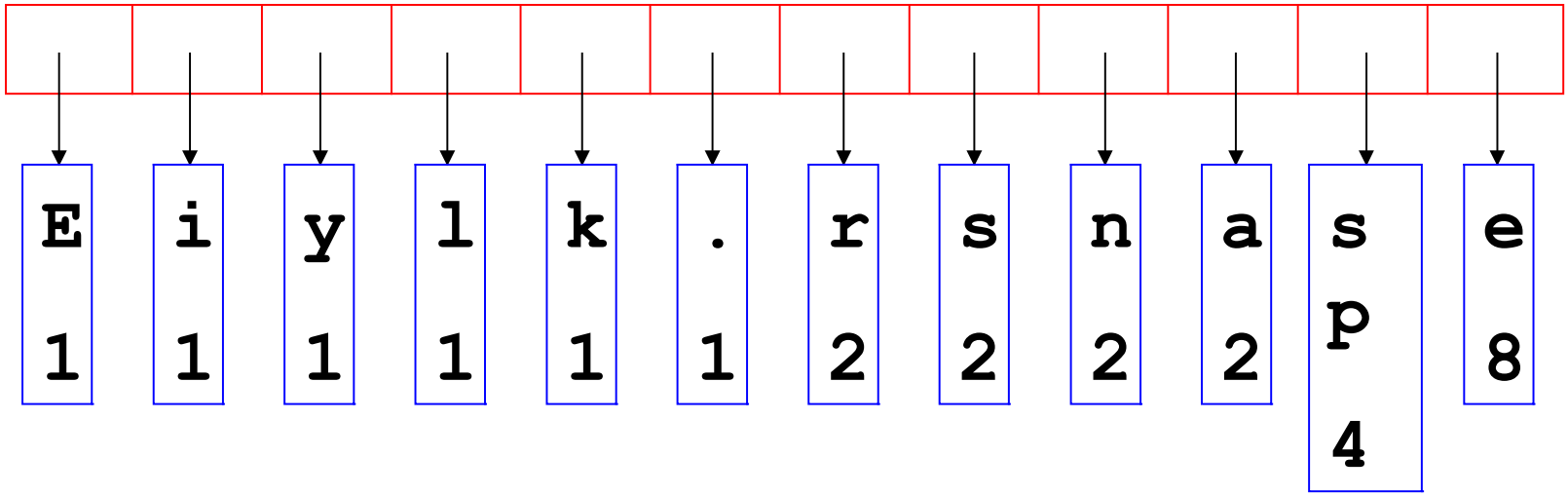
  Eerie eyes seen near lake.

- Đếm số lần xuất hiện của các kí tự trong văn bản

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		

Xây dựng cây

- Hàng đợi sau khi thêm tất cả các nốt

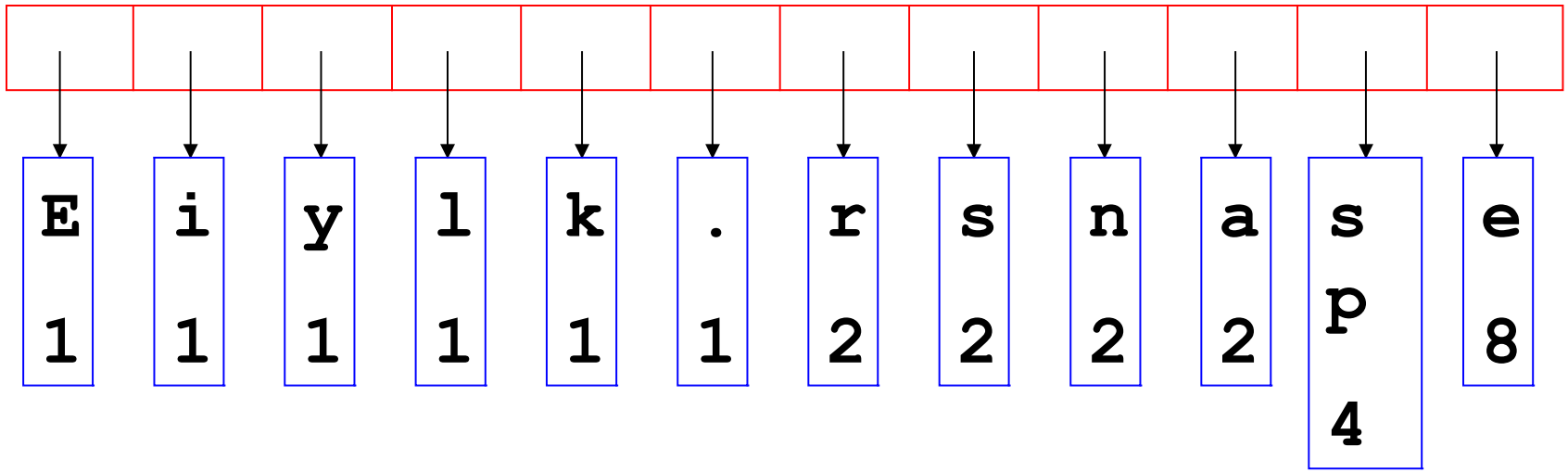


- Con trỏ NULL không được hiển thị

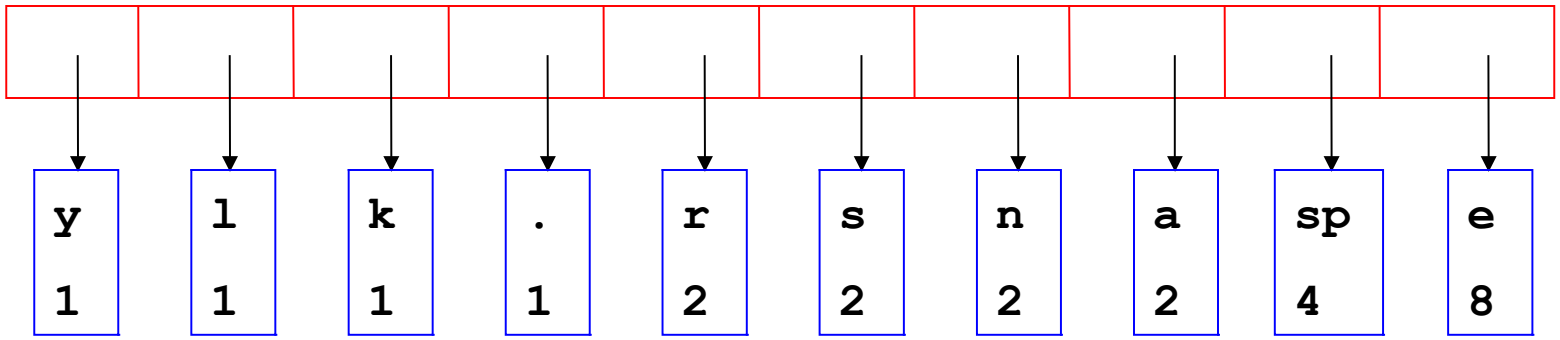
Xây dựng cây

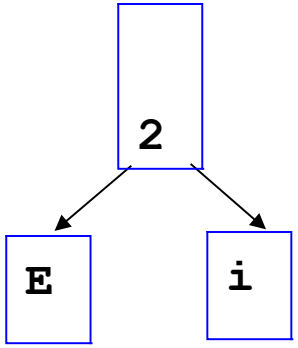
- Khi hàng đợi ưu tiên chứa hai hay nhiều nút
  - Tạo ra một nút mới
  - Lấy nút ra và gán thành cây con trái
  - Lấy nút tiếp theo và gán thành cây con phải
  - Tần xuất của nút mới bằng tổng tần xuất của hai nút con
  - Đưa nút mới vào hàng đợi

Xây dựng cây

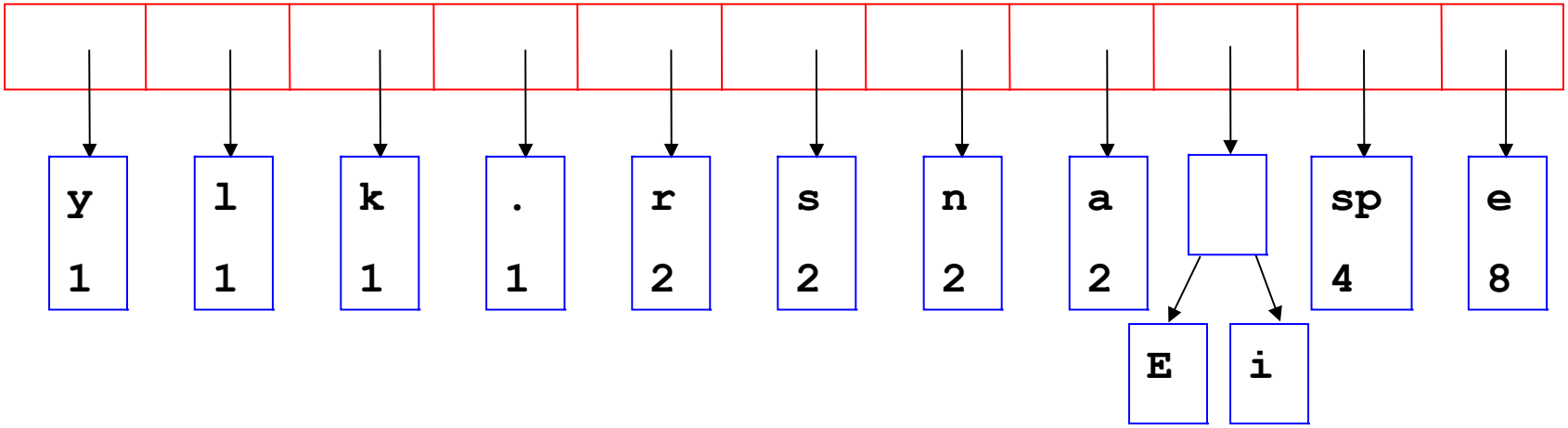


Xây dựng cây

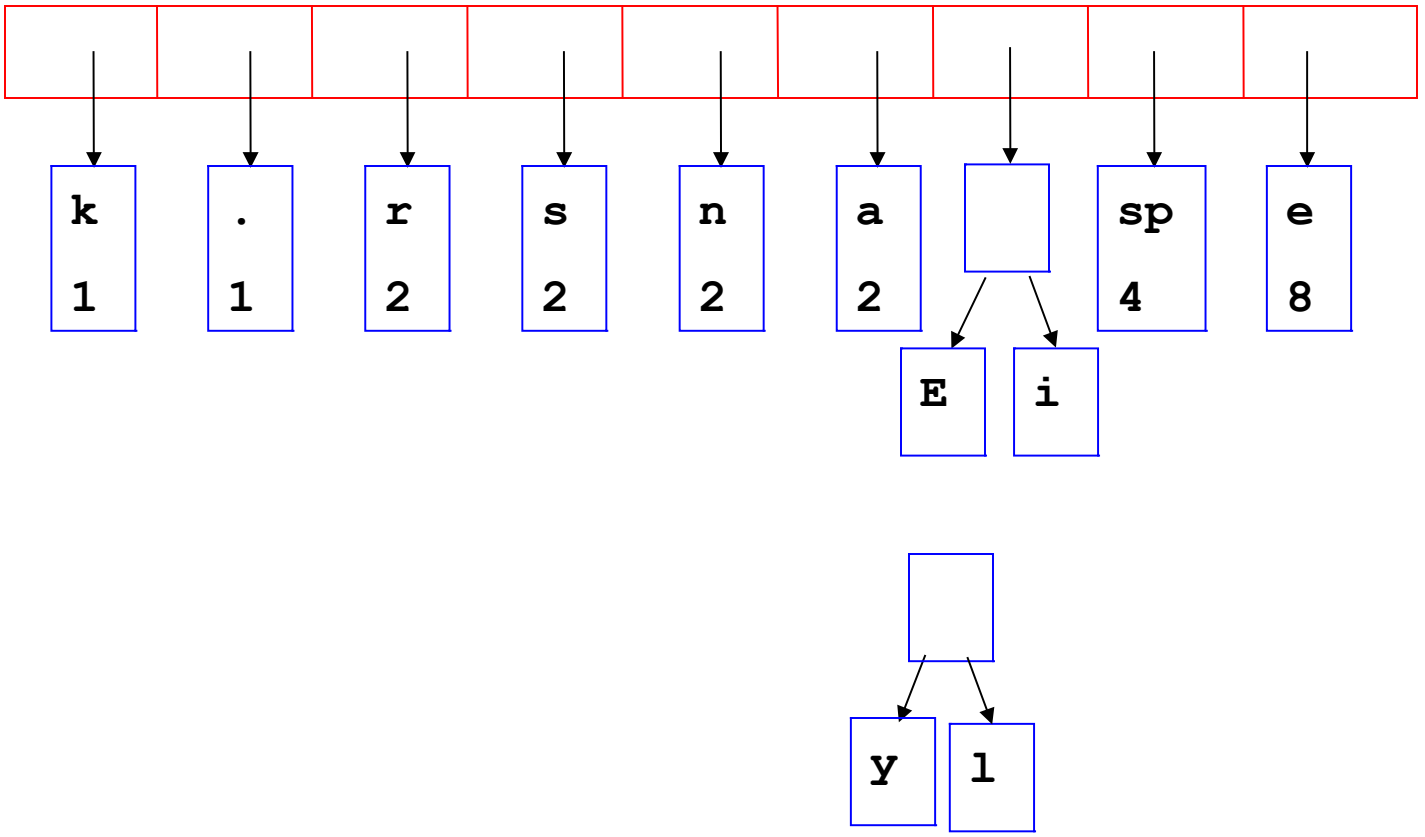




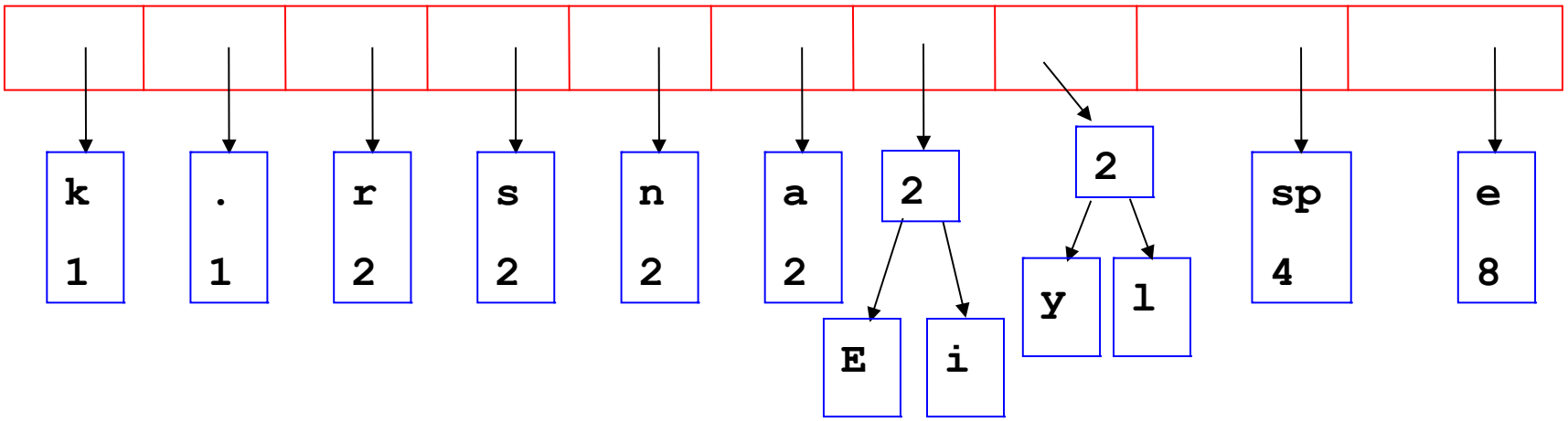
Xây dựng cây



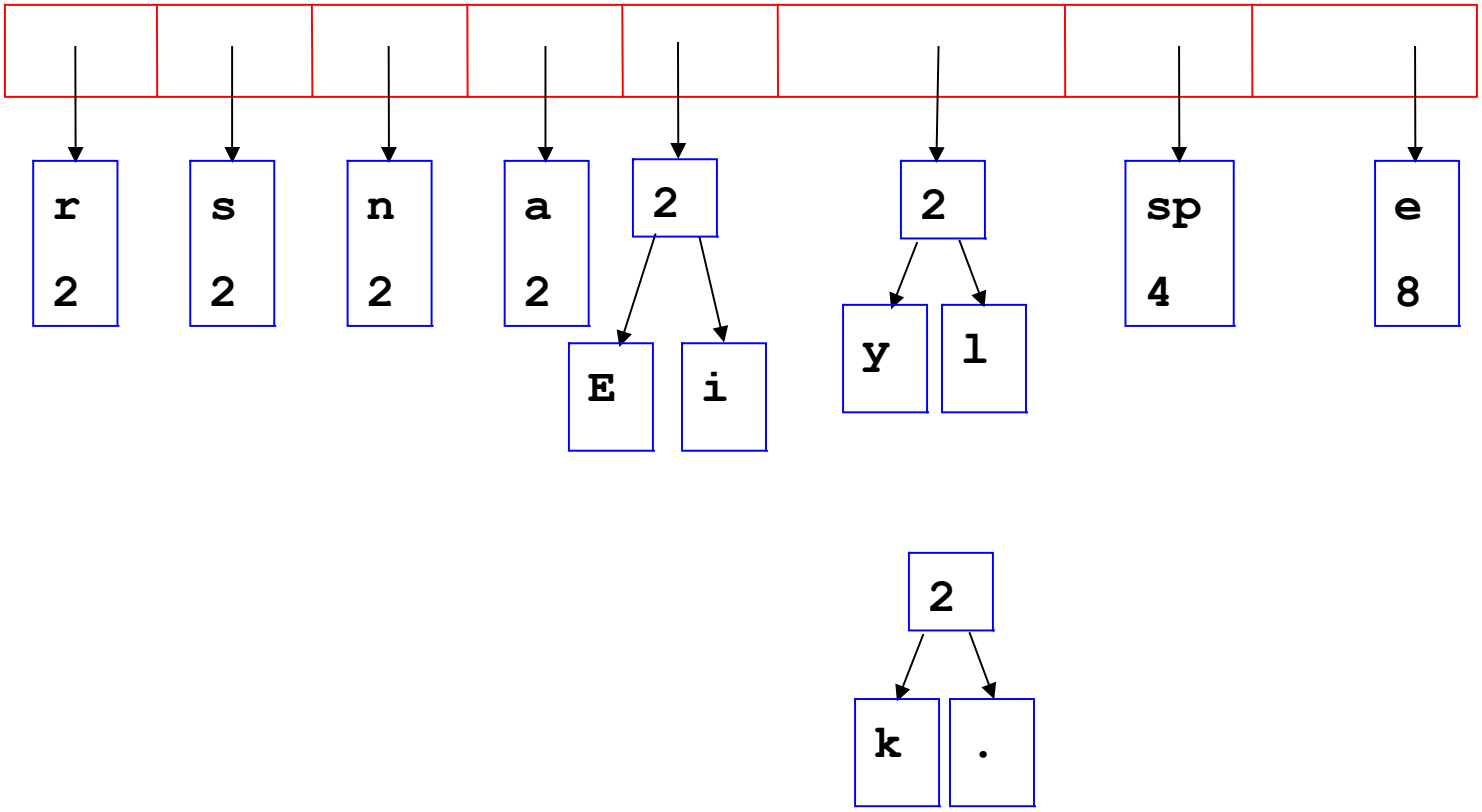
Xây dựng cây



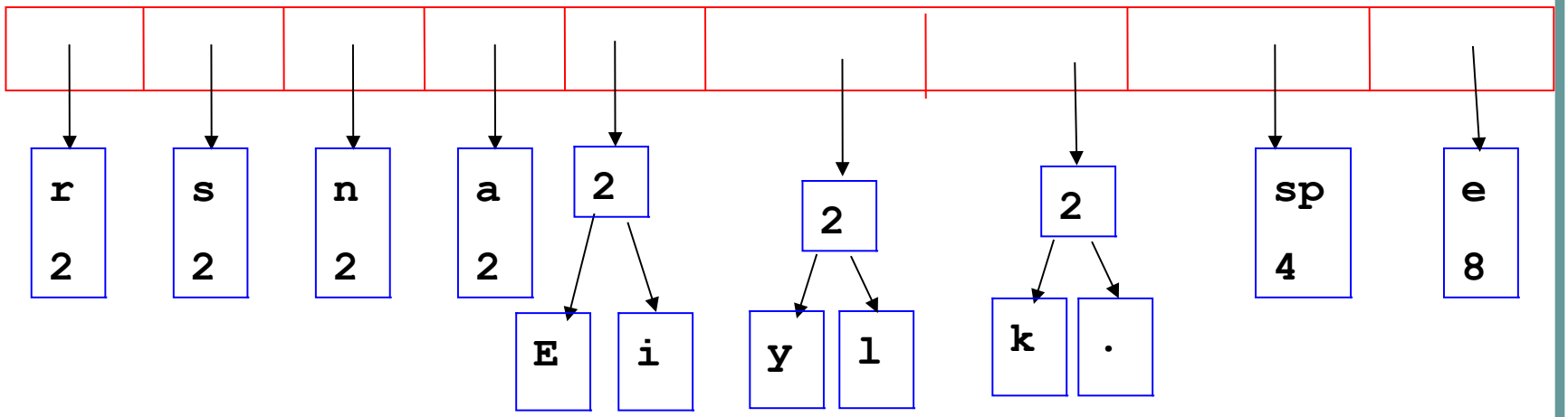
Xây dựng cây



Xây dựng cây

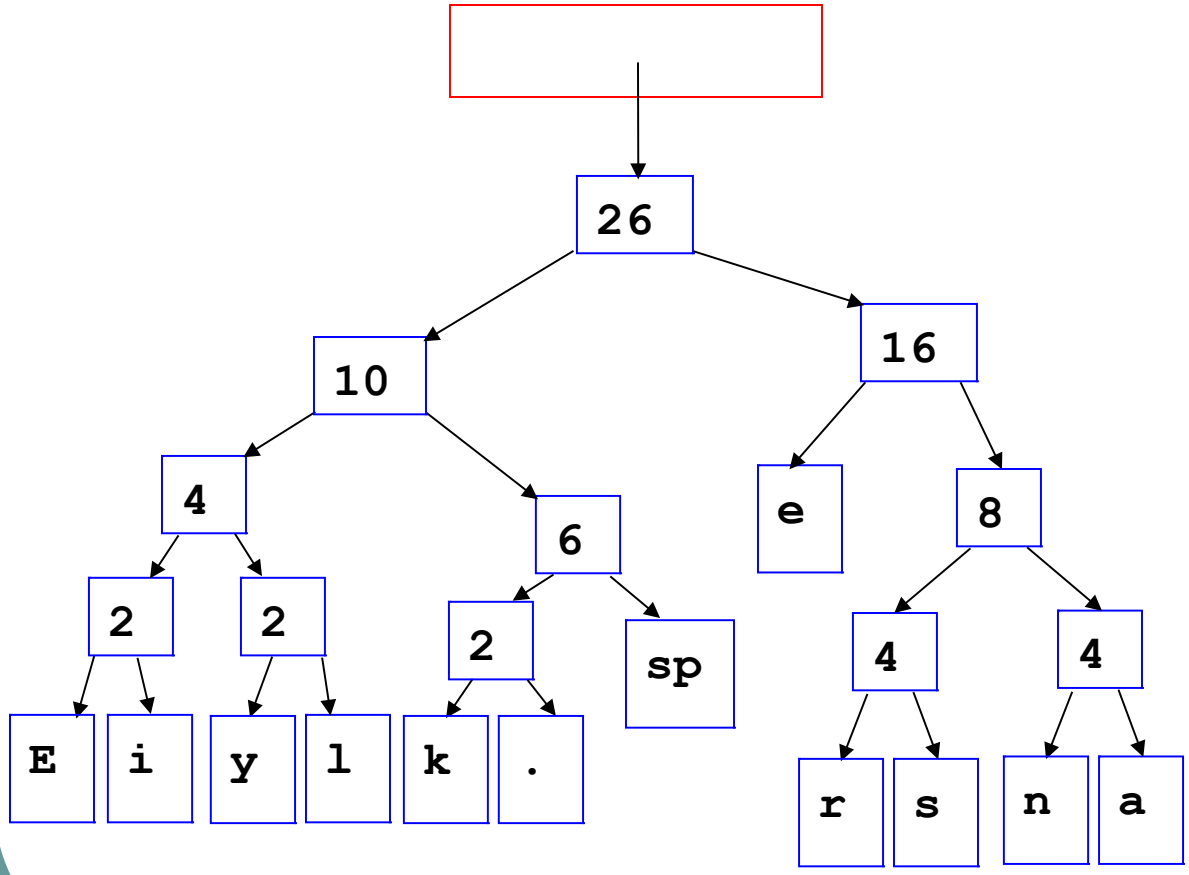


Xây dựng cây



- Còn tiếp…

Cuối cùng



Sau khi lấy nút này ra, chỉ còn lại một nút trong hàng đợi ưu tiên

Cài đặt ntn?

- Sử dụng lại JRB để biểu diễn cây
  - Mỗi nút mới được tạo bởi một nút JRB
  - Cạnh hướng từ cha xuống con
  - Hai cạnh được tạo và được đánh dấu 0 hoặc 1 khi một nút cha được tạo
- Sử dụng lại Dllist hoặc JRB để biểu diễn hàng đợi ưu tiên
  - Một nút hàng đợi chứa một khóa là tần suất của nút tương ứng trong cây
  - Giá trị của nút hàng đợi là con trỏ trỏ đến nút tương ứng trong cây

Bài tập 1

- Sử dụng lại API đồ thị để xây một cây Huffman từ một chuỗi như sau:

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

Bảng mã Huffman

- Để nén dữ liệu chuỗi, ta cần xây dựng bảng mã từ cây Huffman. Cấu trúc dữ liệu sau được sử dụng để biểu diễn bảng mã:

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

- huffmanTable['A'] cho mã của 'A'. Nếu kích thước của mã = 0, kí tự 'A' không có trong văn bản. Các bit chứa mã huffman (chuỗi bit) của kí tự tương ứng.

Bài tập 2

- Viết hàm tạo bảng mã từ cây Huffman
  - void createHuffmanTable(HuffmanTree htree, Coding* htable);
- Viết hàm nén một chuỗi kí tự thành một mã Huffman.
  - void compress(char * buffer, int size, char* huffman, int* nbit);
- Chuỗi chứa *size* kí tự. Sau khi nén, mã huffman chứa nbit bit.
- Để viết hàm này, bạn nên viết một hàm để thêm một kí tự mới vào mã huffman như sau
  - void addHuffmanChar(char * ch, Coding* htable, char* huffman, int* nbit);