Bài 11: Hàng ưu tiên

Giảng viên: Hoàng Thị Điệp

Khoa Công nghệ Thông tin – Đại học Công Nghệ

Nội dung chính

1. KDLTT hàng ưu tiên
2. Các phương pháp cài đặt
3. Ứng dụng: xây dựng mã Huffman

diepht@vnu

KDLTT hàng ưu tiên (priority queue)

- Là tập hợp trong đó mỗi phần tử là một cặp (giá trị ưu tiên, đối tượng)
  - ta có thể so sánh được các giá trị ưu tiên
- Các phép toán
  - insert(k, o) xen vào hàng ưu tiên đối tượng o có giá trị ưu tiên k.
  - findMin() tìm đối tượng có giá trị ưu tiên nhỏ nhất. Thực hiện được nếu hàng không rỗng
  - removeMin() loại bỏ và trả về đối tượng có giá trị ưu tiên nhỏ nhất. Thực hiện được nếu hàng không rỗng.
  - findMinKey() tìm giá trị ưu tiên nhỏ nhất .Thực hiện được nếu hàng không rỗng
  - size()
  - isEmpty()
- Ứng dụng
  - Quản lý băng thông
  - Sử dụng trong thiết kế các thuật toán (Huffman…)

diepht@vnu

Minh họa

Phép toán	Output	Hàng ưu tiên
insert(5,A)	-	{(5,A)}
insert(9,C)	-	{(5,A), (9,C)}
insert(3,B)	-	{(3,B), (5,A), (9,C)}
insert(7,D)	-	{(3,B), (5,A), (7,D), (9,C)}
findMin()	B	{(3,B), (5,A), (7,D), (9,C)}
findMinKey()	3	{(3,B), (5,A), (7,D), (9,C)}
removeMin()	-	{(5,A), (7,D), (9,C)}
size()	3	{(5,A), (7,D), (9,C)}
findMin ()	A	{(5,A), (7,D), (9,C)}
removeMin()	-	{(7,D), (9,C)}
removeMin()	-	{(9,C)}
removeMin()	-	{}
removeMin()	“error”	{}
isEmpty()	true	{}

diepht@vnu

Wikipedia: priority queue

Contents [hide]

diepht@vnu

NIST's DADS: priority queue

Definition: An *abstract data type* to efficiently support finding the item with the highest priority across a series of operations. The basic operations are: insert, find-minimum (or maximum), and delete-minimum (or maximum). Some implementations also efficiently support join two priority queues (*meld*), delete an arbitrary item, and increase the priority of a item (decrease-key).

Formal Definition: The operations new(), insert(v, PQ), find-minimum or min(PQ), and delete-minimum or dm(PQ) may be defined with *axiomatic semantics* as follows.

1. new() returns a priority queue
2. min(insert(v, new())) = v
3. dm(insert(v, new())) = new()
4. min(insert(v, insert(w, PQ))) = if priority(v) < priority(min(insert(w, PQ))) then v else min(insert(w, PQ))
5. dm(insert(v, insert(w, PQ))) = if priority(v) < priority(min(insert(w, PQ))) then insert(w, PQ) else insert(v, dm(insert(w, PQ)))

where PQ is a priority queue, v and w are items, and priority(v) is the priority of item v.

diepht@vnu

Standard Template Library

			Container Adaptors		
Headers			<stack>	<queue>	
Members			**stack**	**queue**	**priority_queue**
	constructor	*	constructor	constructor	constructor
capacity	size	O(1)	size	size	size
	empty	O(1)	empty	empty	empty
element access	front	O(1)		front	
	back	O(1)		back	
	top	O(1)	top		top
modifiers	push	O(1)	push	push	push
	pop	O(1)	pop	pop	pop

diepht@vnu

Nội dung chính

1. KDLTT hàng ưu tiên
2. Các phương pháp cài đặt
3. Ứng dụng: xây dựng mã Huffman

diepht@vnu

Các phương pháp cài đặt

	findMin	insert	removeMin
Mảng được sắp	?	?	?
Mảng không sắp	?	?	?
DSLK được sắp	?	?	?
DSLK không sắp	?	?	?
Cây TKNP	?	?	?
Cây thứ tự bộ phận (heap)	?	?	?

diepht@vnu

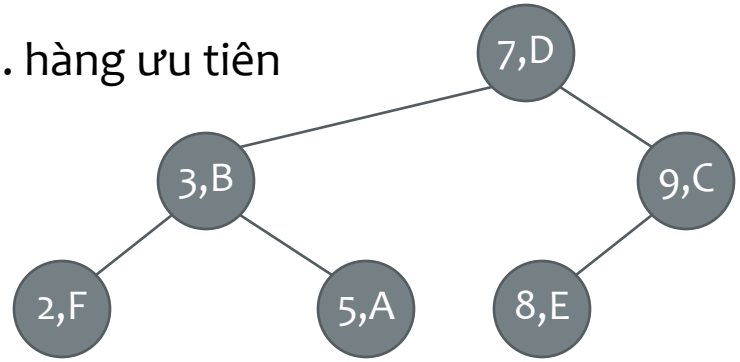
Cài đặt hàng ưu tiên bởi mảng

	Ví dụ	Minh họa mảng
Mảng được sắp	Q = {(3,B), (5,A), (7,D), (9,C)}	
	findMin()	
	insert(8, E)	
	removeMin()	
Mảng không sắp	Q = {(7,D), (3,B), (9,C), (5,A)}	
	findMin()	
	insert(8, E)	
	removeMin()	

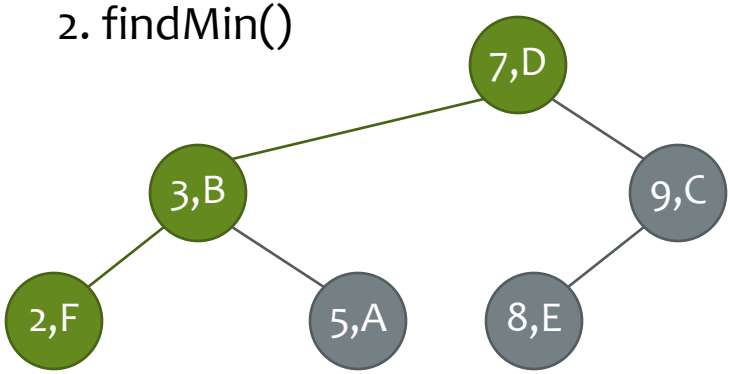
diepht@vnu

Cài đặt hàng ưu tiên bởi cây tìm kiếm nhị phân

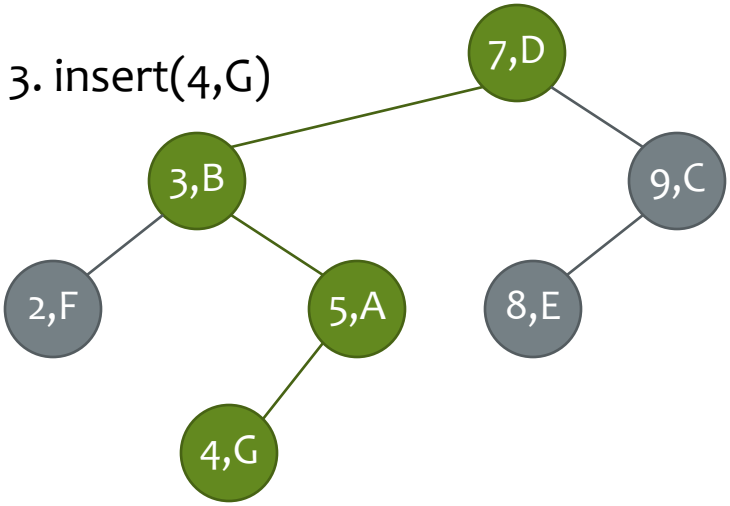
1. hàng ưu tiên



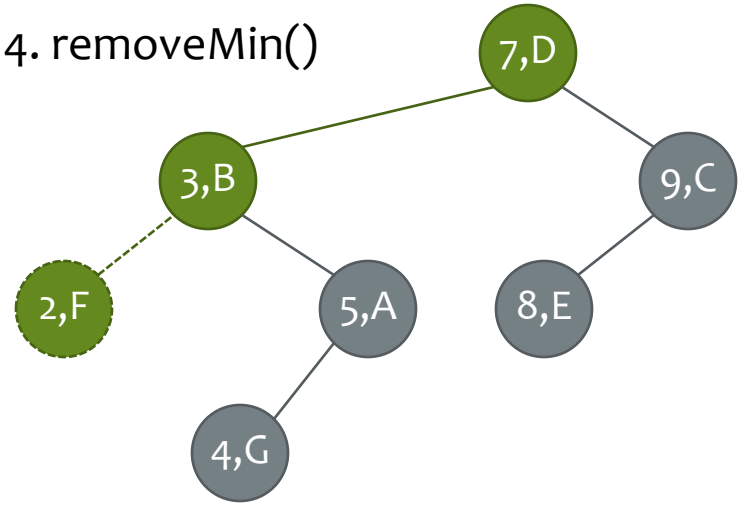
2. findMin()



3. insert(4,G)



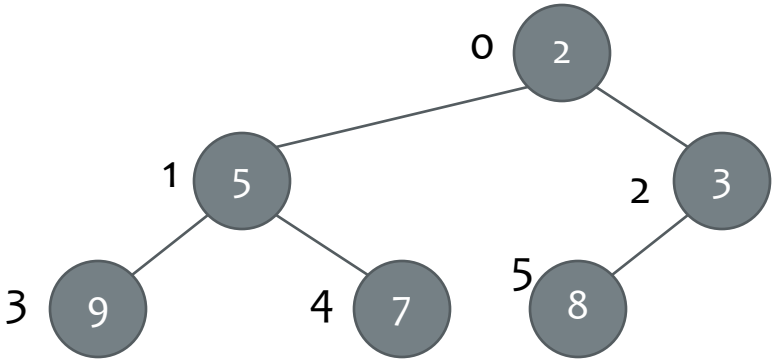
4. removeMin()



diepht@vnu

Cây thứ tự bộ phận (heap)

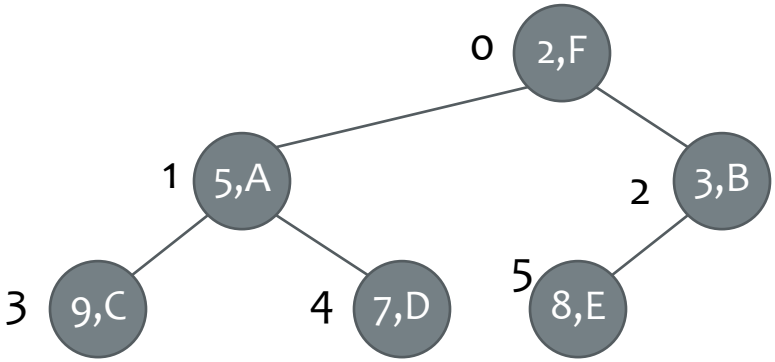
- Cây nhị phân hoàn toàn
  - có tất cả các mức của cây đều không thiếu đỉnh nào, trừ mức thấp nhất được lấp đầy kể từ bên trái
- Min heap là một cây nhị phân hoàn toàn với tính chất
  - khóa của một đỉnh bất kỳ nhỏ hơn hoặc bằng khóa của các đỉnh con của nó
- Max heap
- Cài đặt heap
  - có thể dùng cấu trúc liên kết (con trỏ)
  - có thể dùng mảng
- Độ cao của heap là log(n)



2	5	3	9	7	8

diepht@vnu

Cài đặt hàng ưu tiên bởi heap



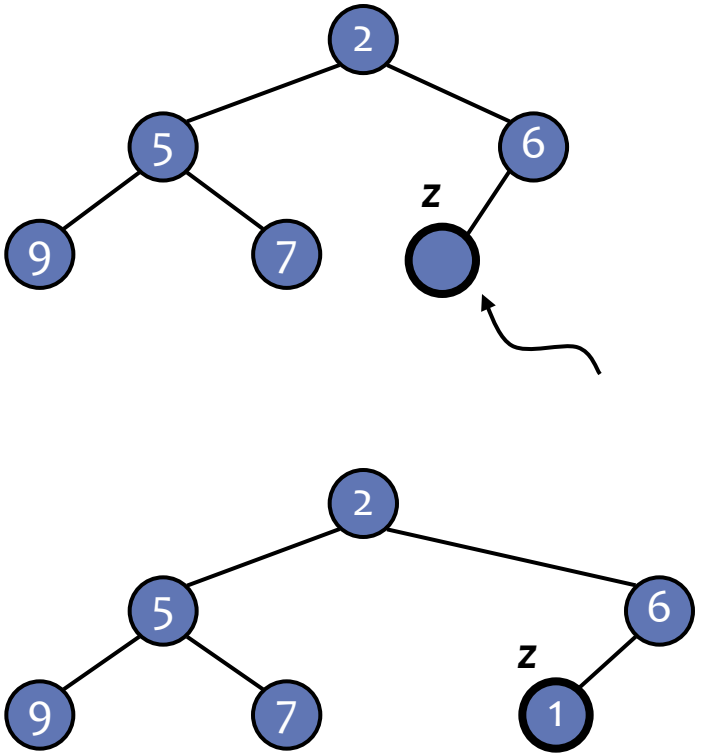
2,F	5,A	3,B	9,C	7,D	8,E

- findMin?
- insert?
- removeMin?

diepht@vnu

Xen thêm vào heap

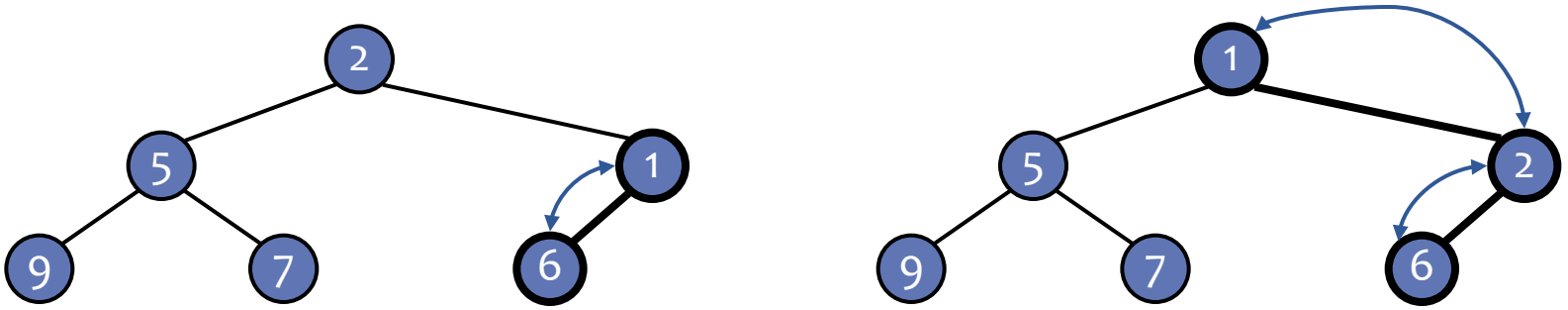
- Phép insert của hàng ưu tiên tương ứng với phép xen thêm một phần tử có khóa ***k*** vào heap
- Thuật toán
  - Tìm tới vị trí ***z*** để đưa phần tử mới vào (là đỉnh cuối mới)
  - Lưu phần tử có khóa ***k*** vào ***z***
  - Khôi phục tính chất thứ tự bộ phận của heap



diepht@vnu

Thuật toán upheap

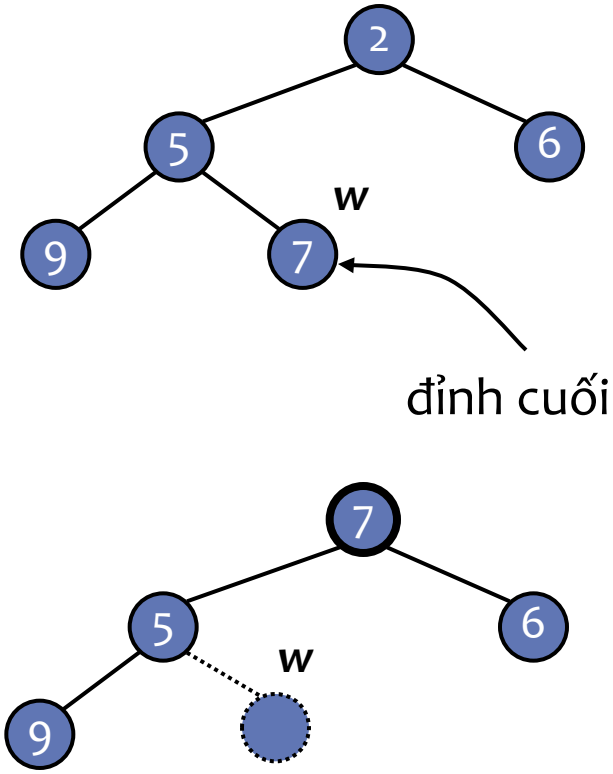
- Sau khi xen thêm một khóa ***k*** mới, heap có thể mất đi tính chất thứ tự bộ phận
- Thuật toán upheap khôi phục lại tính chất này bằng cách đảo chỗ ***k*** dọc theo đường đi từ đỉnh mới hướng tới gốc
- Upheap dừng lại khi ***k*** tiến tới gốc hoặc một đỉnh có khóa của cha ≤ ***k***
- Vì heap có độ cao ***O***(log ***n***), upheap thực hiện trong thời gian ***O***(log ***n***)



diepht@vnu

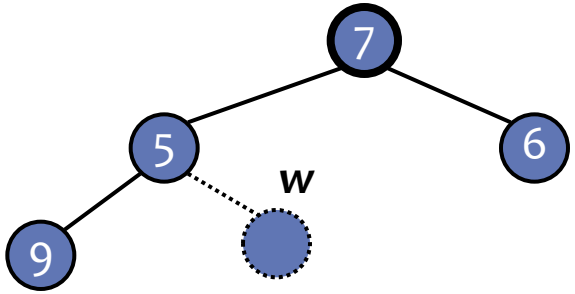
Loại một phần tử khỏi heap

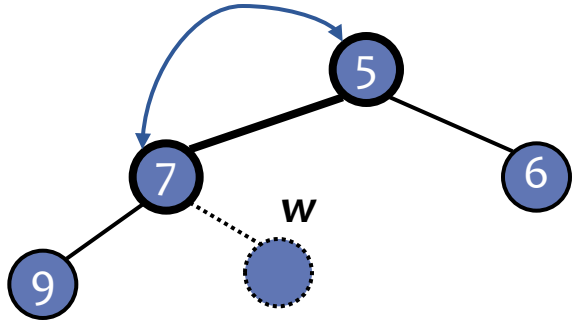
- Phép removeMin của hàng ưu tiên tương ứng với phép loại gốc của heap
- Thuật toán
  - Thay thế gốc bằng đỉnh cuối cùng ***w***
  - Giải phóng đỉnh ***w***
  - Khôi phục tính chất thứ tự bộ phận của heap



Thuật toán downheap

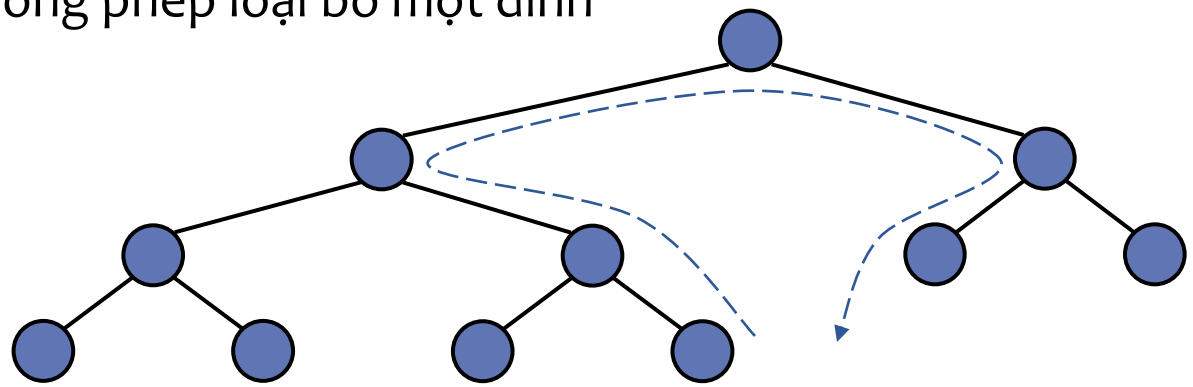
- Sau khi thay thế đỉnh gốc với đỉnh cuối (có khóa ***k***), heap có thể mất đi tính chất thứ tự bộ phận
- Thuật toán downheap khôi phục lại tính chất này bằng cách đảo chỗ đỉnh có khóa ***k*** dọc theo đường đi từ gốc xuống
- Downheap dừng khi ***k*** tiến tới một lá hay một đỉnh có các khóa con ≥ ***k***
- Do heap có độ cao ***O***(log ***n***), downheap thực hiện trong thời gian ***O***(log ***n***)





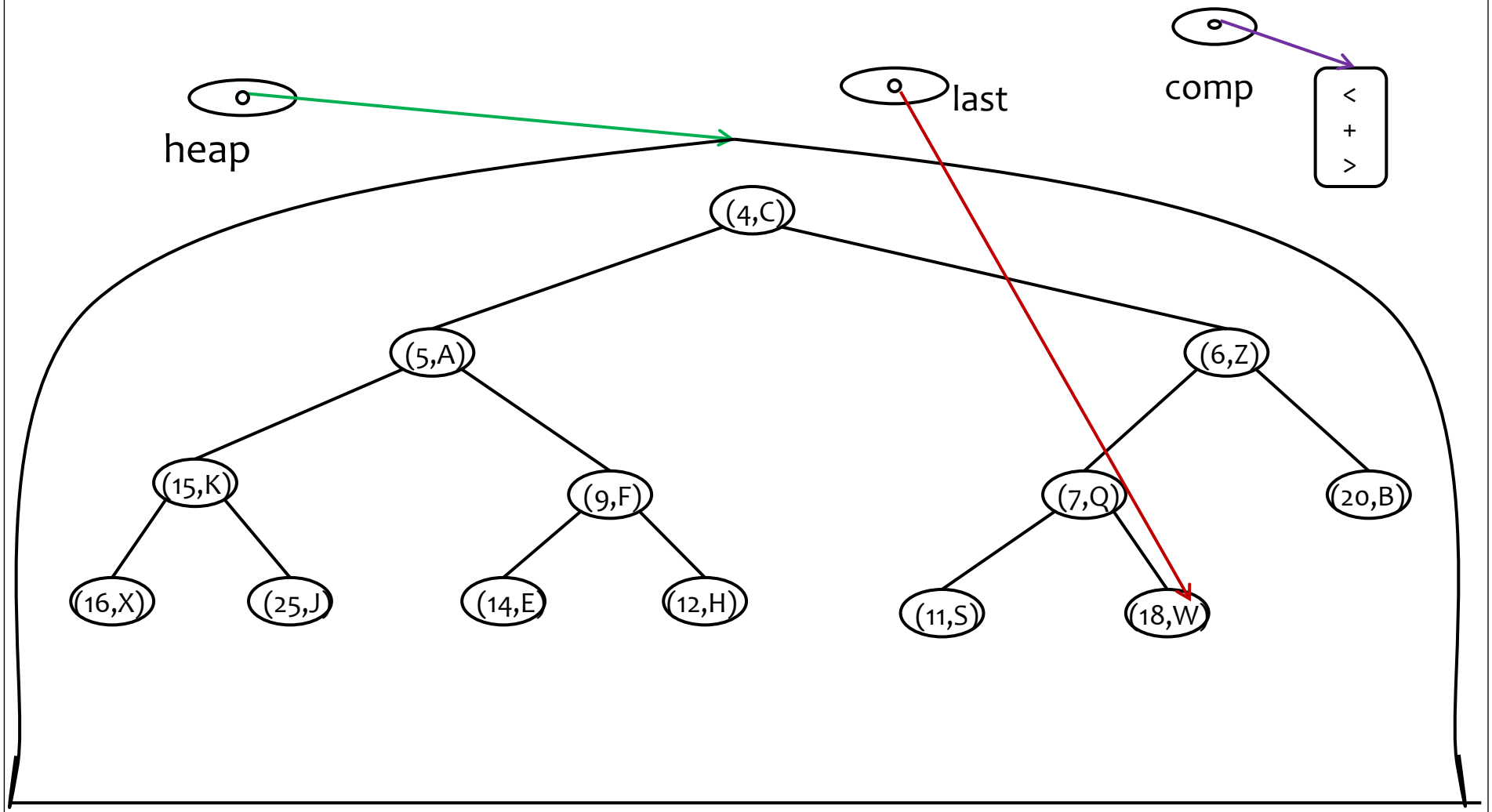
Cập nhật con trỏ tới đỉnh cuối

- Dùng trong cài đặt heap bằng cấu trúc liên kết
- Có thể tìm chỗ cho đỉnh sẽ thêm vào bằng cách đi theo hành trình gồm $O(\log n)$ đỉnh
  - Đi lên tới khi gặp một con trái hoặc gặp gốc
  - Nếu gặp một con trái thì chuyển sang con phải
  - Đi xuống tới khi gặp một lá
- Áp dụng thuật toán tương tự cho cập nhật con trỏ tới đỉnh cuối trong phép loại bỏ một đỉnh



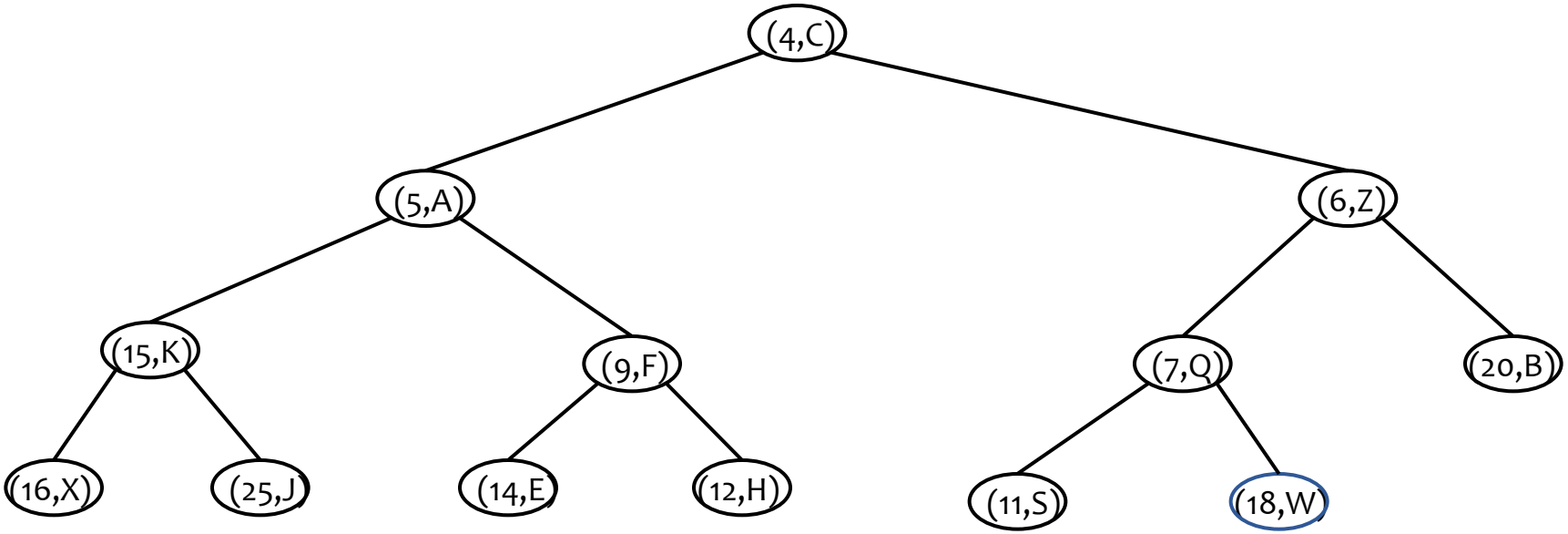
diepht@vnu

Minh họa cài đặt hàng ưu tiên



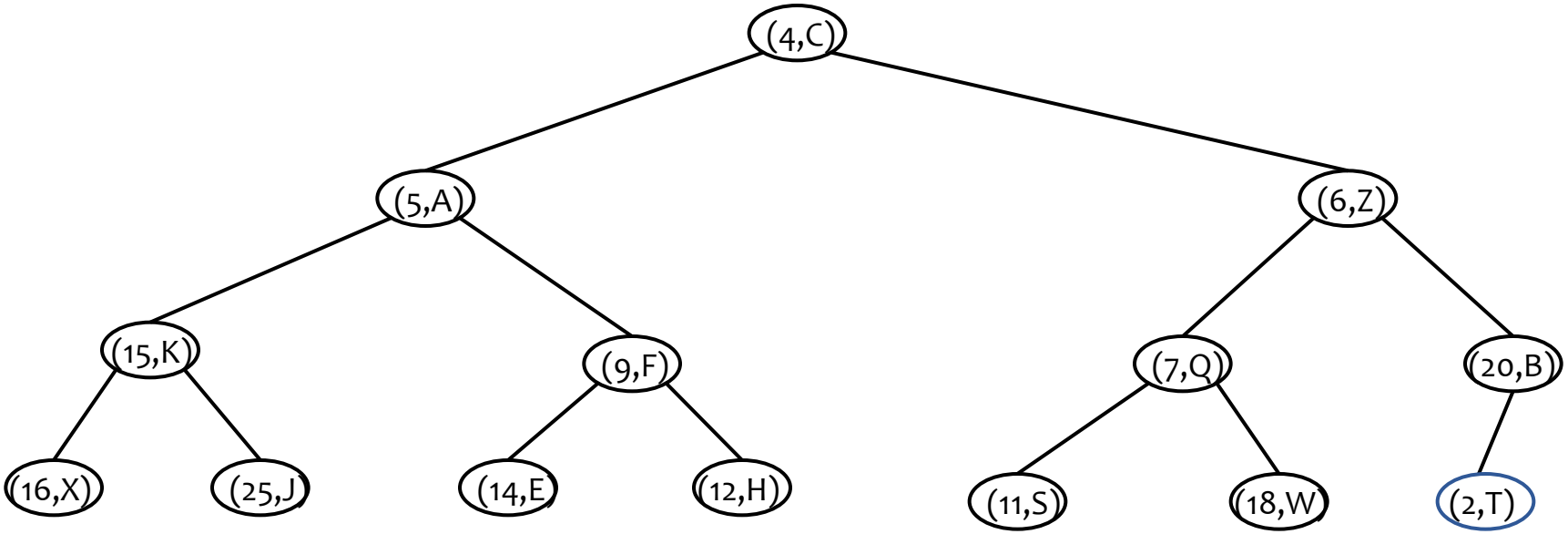
diepht@vnu

insert(2,T)



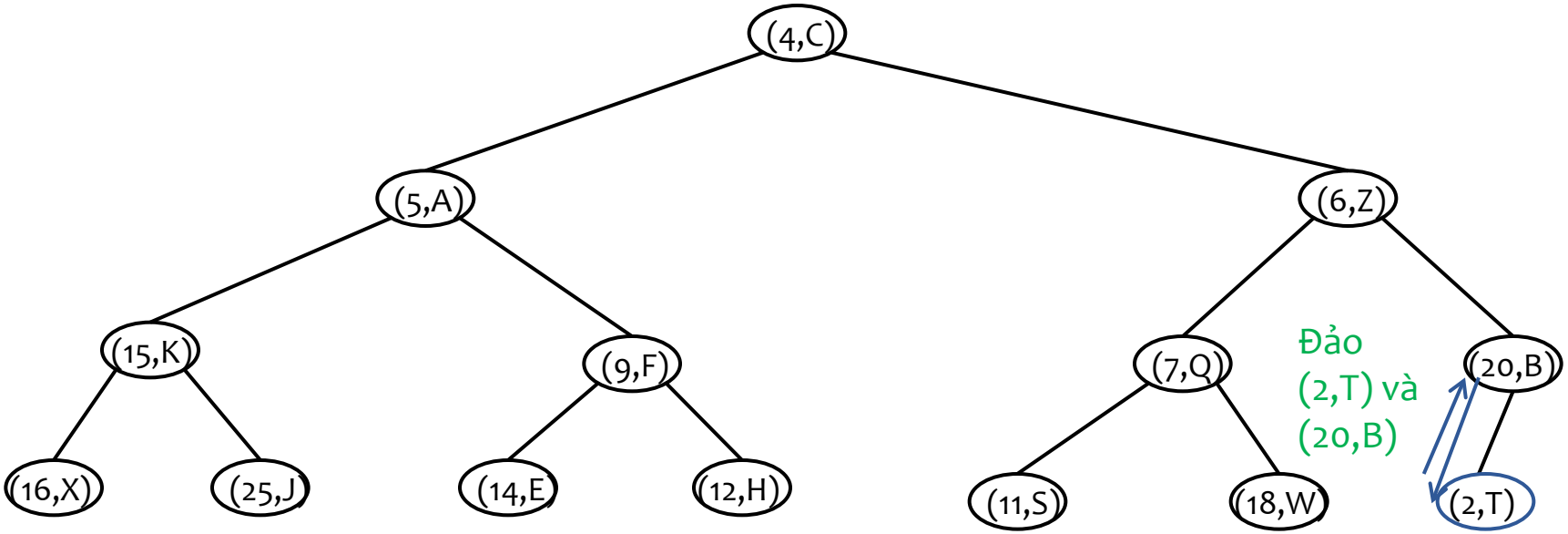
diepht@vnu

insert(2,T)



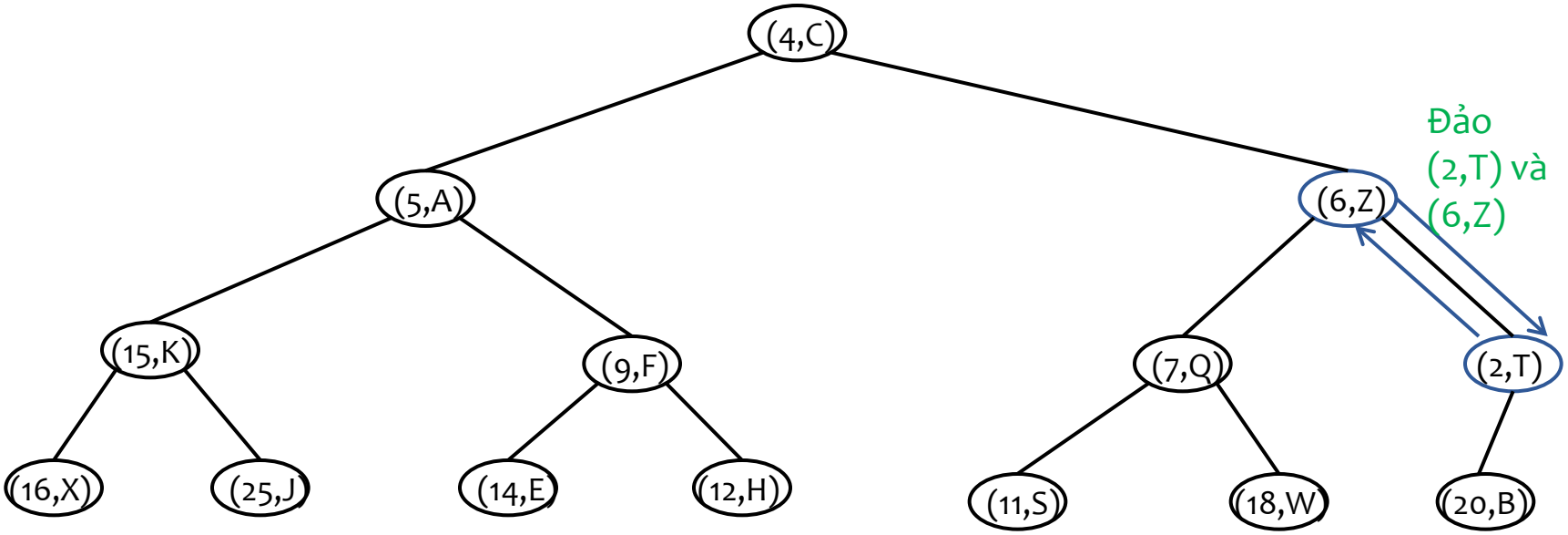
diepht@vnu

insert(2,T)

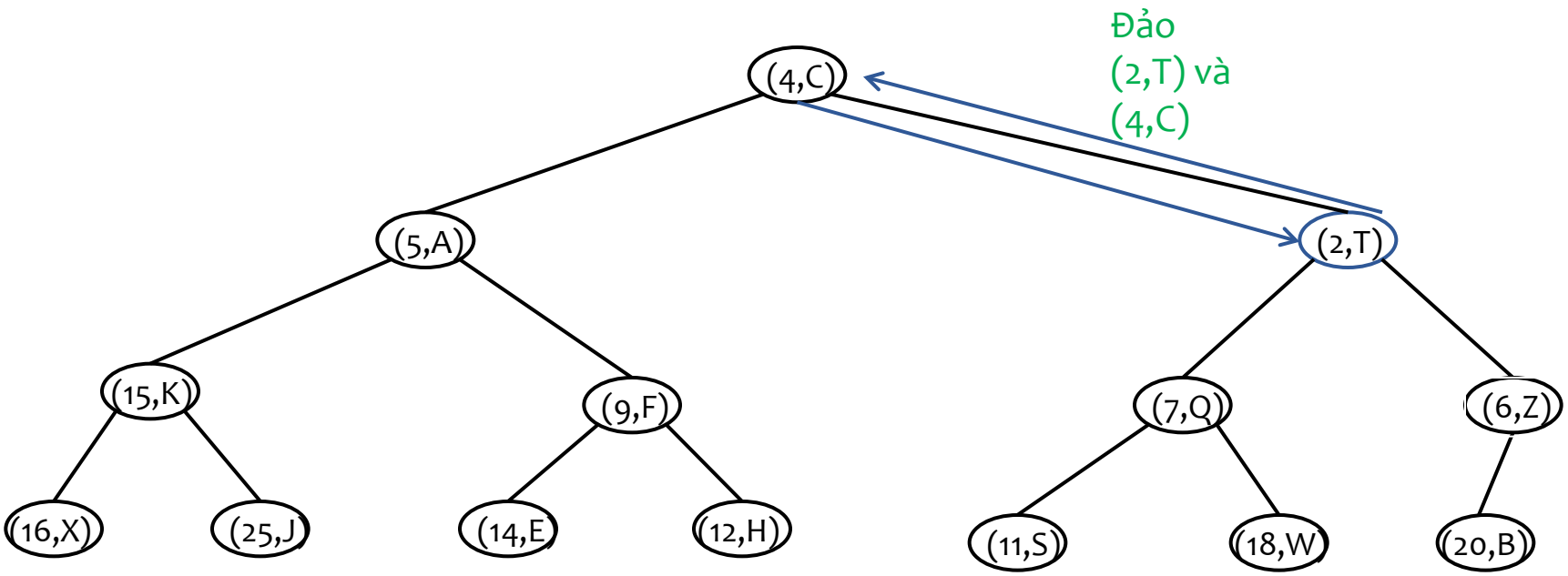


diepht@vnu

insert(2,T)

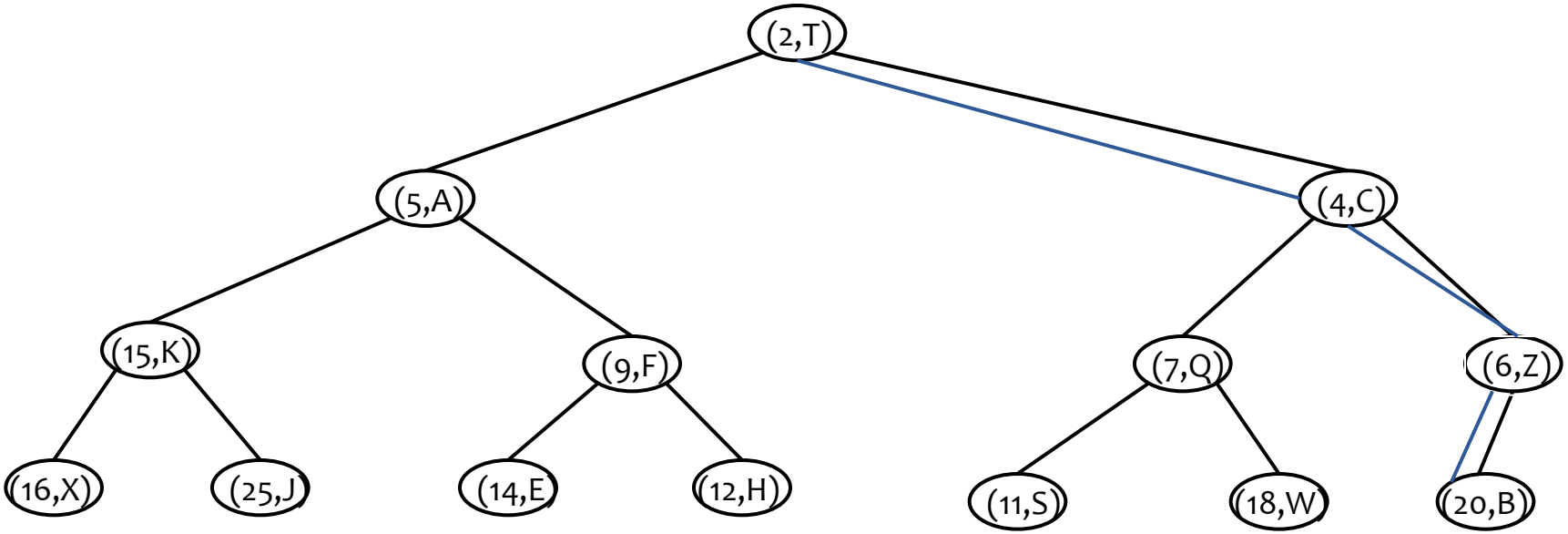


insert(2,T)



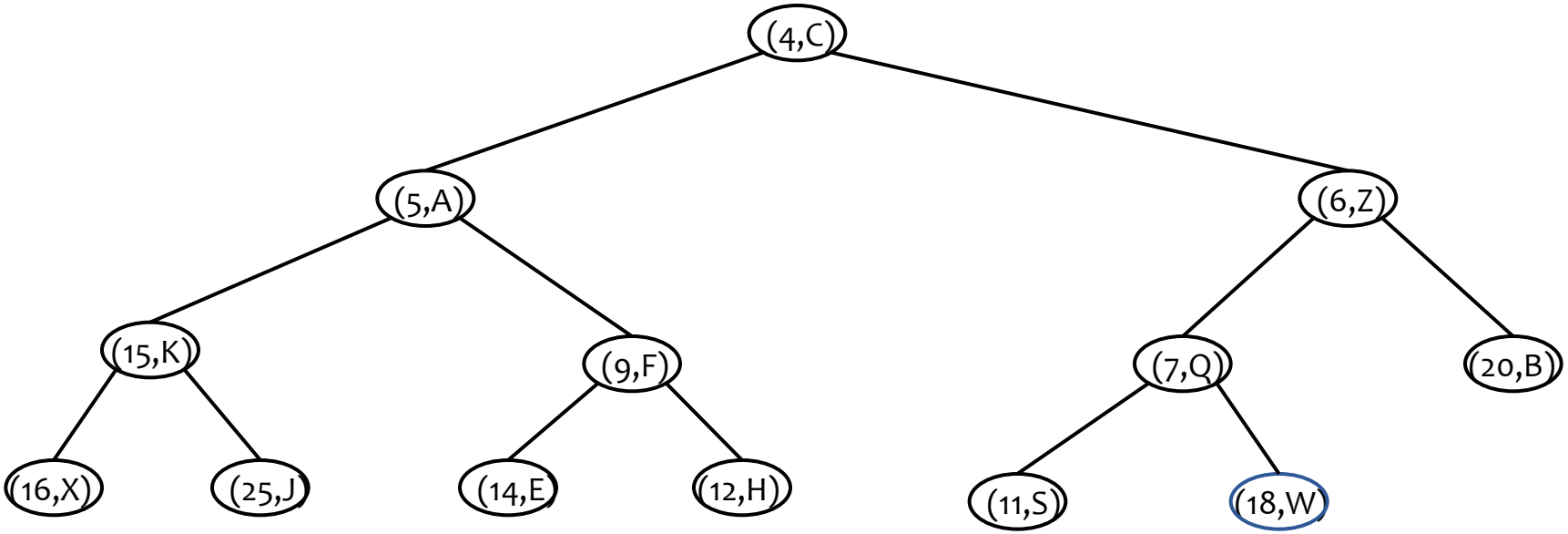
diepht@vnu

insert(2,T)

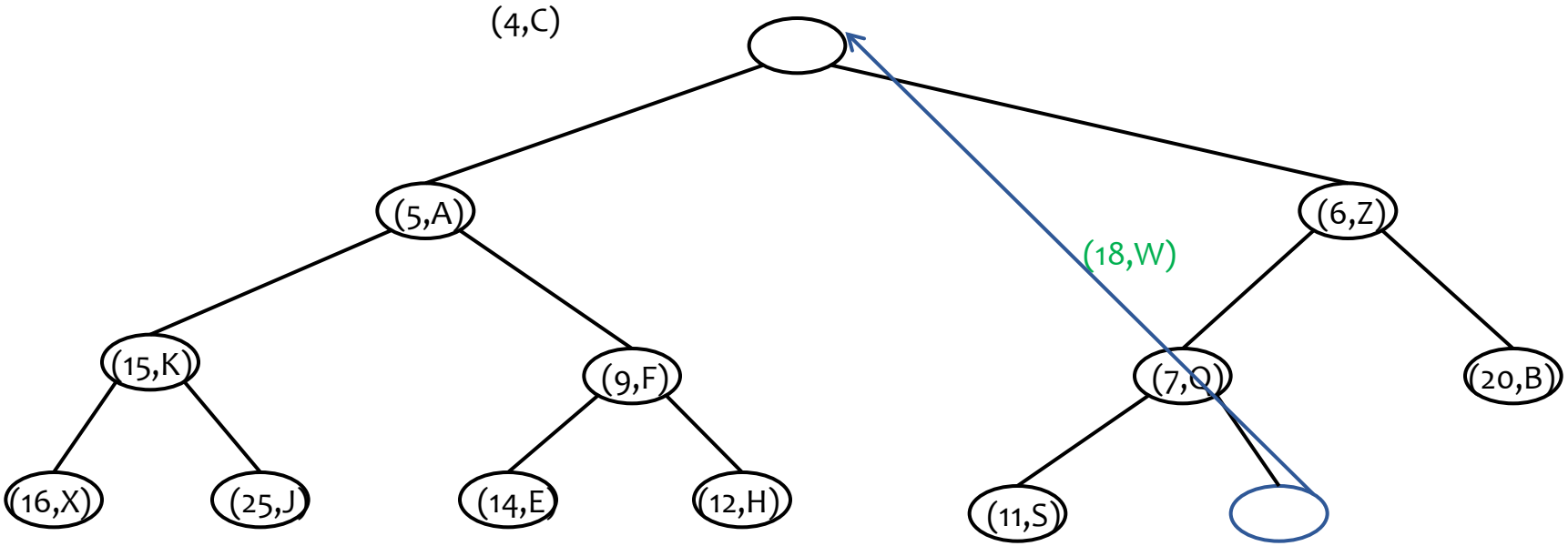


Sau thời gian O (log n) thì cây lại thành một heap

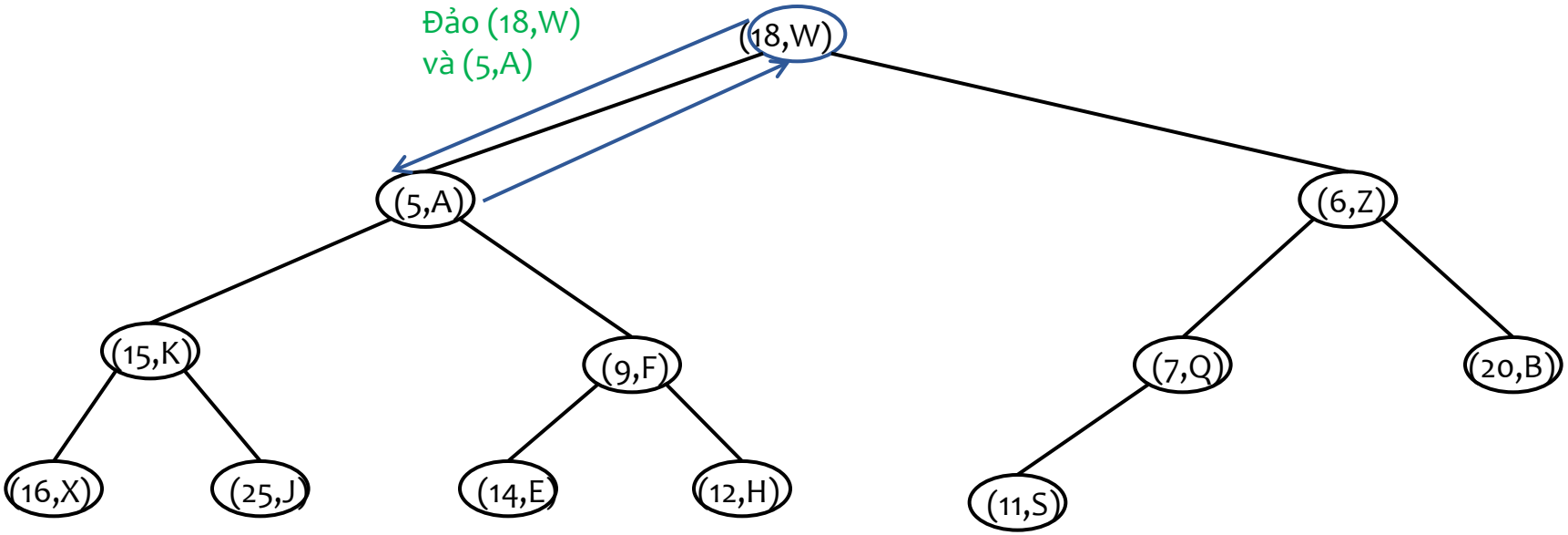
removeMin()



removeMin()

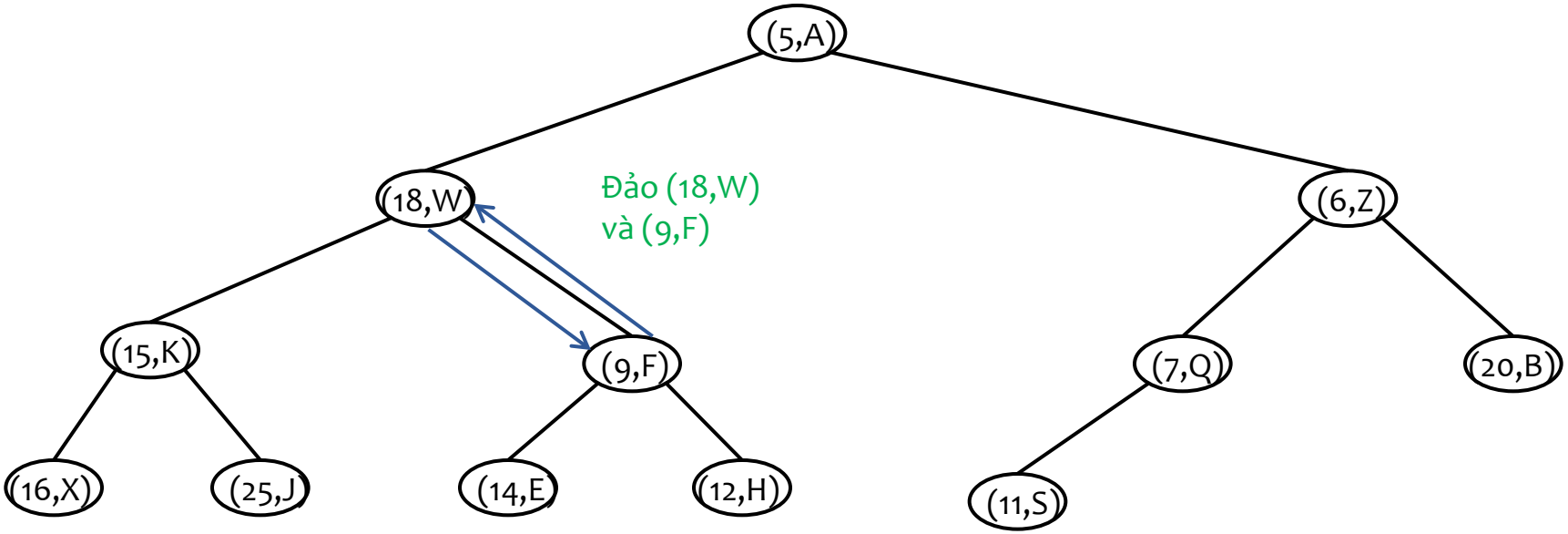


removeMin()

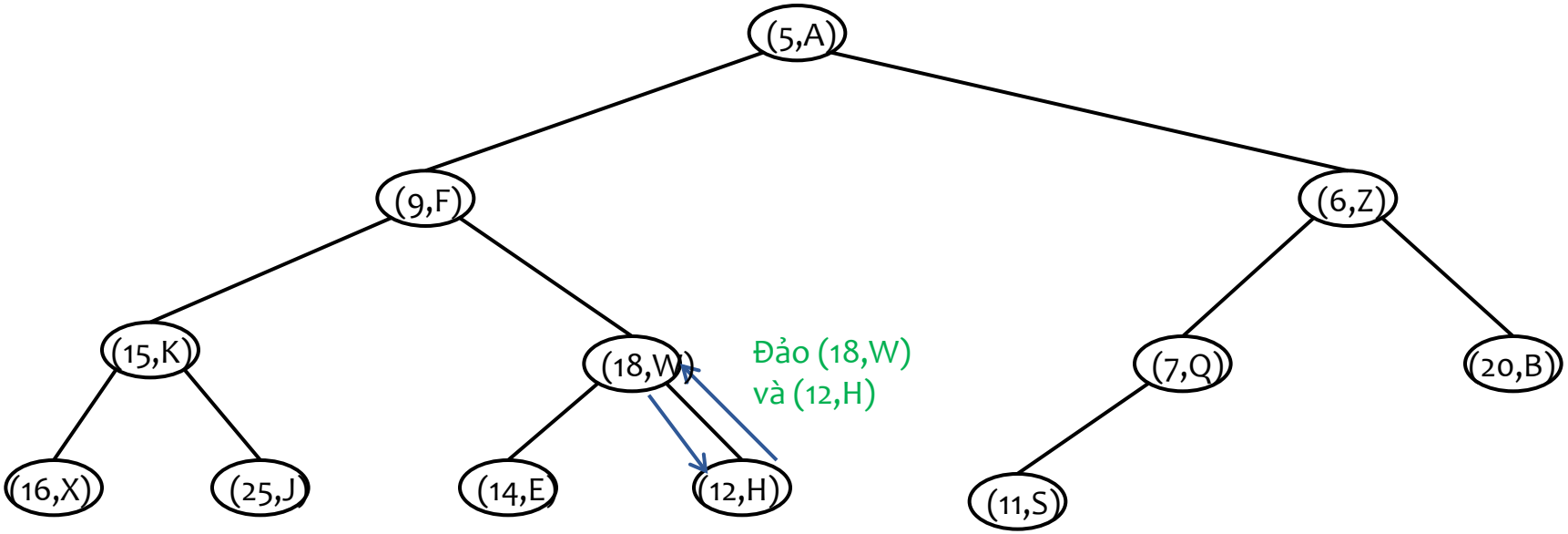


diepht@vnu

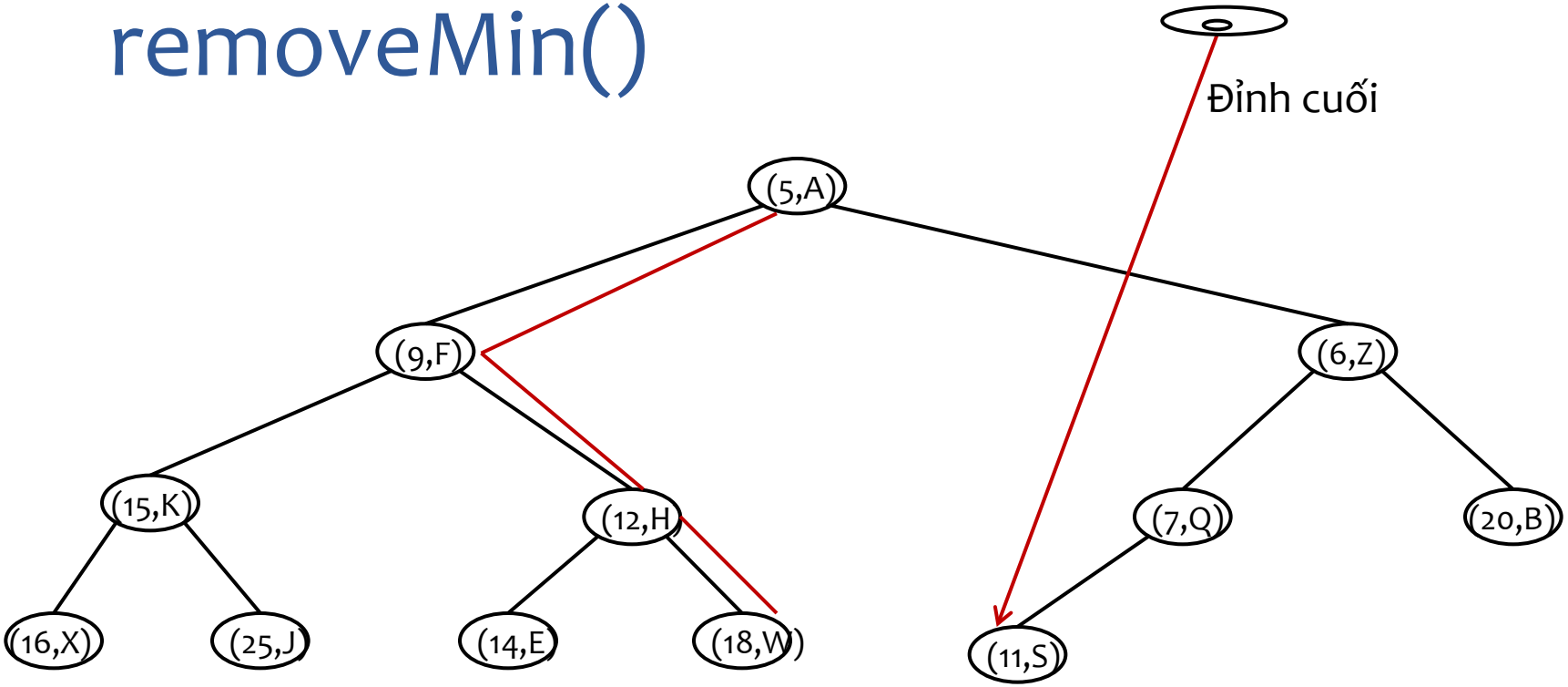
removeMin()



removeMin()



removeMin()



Độ phức tạp

- Độ phức tạp không gian
  - Cài bằng cấu trúc liên kết (dùng con trỏ): *O*(n)
  - Cài bằng cấu trúc vector (mảng): tỉ lệ với N (cỡ của mảng)
- Độ phức tạp thời gian

Phép toán	Thời gian
size, isEmpty	O (1)
findMin, findMinKey	O (1)
insert	O (log n)
removeMin	O (log n)

diepht@vnu

Tổng kết

	findMin	insert	removeMin
Mảng được sắp	O(1)	O(n)	O(1)
Mảng không sắp	O(n)	O(1)	O(n)
DSLK được sắp	O(1)	O(n)	O(1)
DSLK không sắp	O(n)	O(1)	O(n)
Cây TKNP	O(h)	O(h)	O(h)
Cây thứ tự bộ phận (heap)	O(1)	O(logn)	O(logn)

diepht@vnu

Nội dung chính

1. KDLTT hàng ưu tiên
2. Các phương pháp cài đặt
3. Ứng dụng: xây dựng mã Huffman

Nén dữ liệu

- Giả sử cần nén một tệp dữ liệu chứa 100000 ký tự từ bảng 6 chữ cái (từ a đến f).
- Mã độ dài giống nhau (1)
  - biểu diễn mỗi chữ cái bởi 3 bit (thay vì 8 bit như thường lệ)
  - tỉ lệ nén = 3/8
- Mã độ dài khác nhau (2)
  - dùng khi ta biết tần suất của các chữ cái
  - gán mã ngắn nhất cho chữ cái xuất hiện nhiều nhất
  - kích thước file nén:

(45×1 + 13×3 + 12×3 + 16×3 + 9×4 + 5×4) × 1000 = 224 000 bits

  - tỉ lệ nén = 0.28

(1)

Chữ cái	a	b	c	d	e	f
Từ mã	000	001	010	011	100	101

(2)

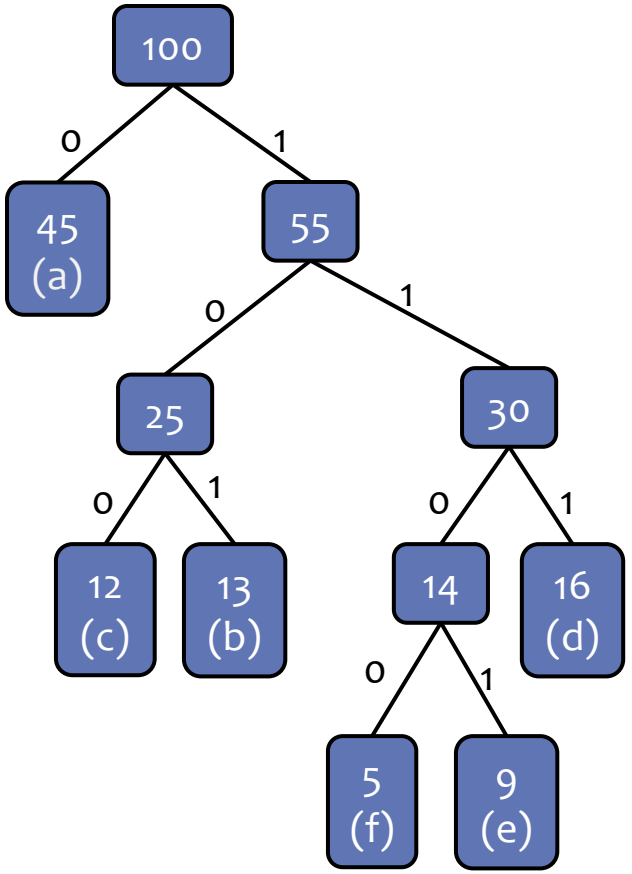
Chữ cái	a	b	c	d	e	f
Tần suất (K)	45	13	12	16	9	5
Từ mã	0	101	100	111	1101	1100

- Chú ý: không có mã nào được làm tiền tố của mã khác
  - gọi là mã tiền tố (prefix code)
  - mục đích: phục vụ giải nén
- Bài toán: xây dựng mã tiền tố với tỉ lệ nén thấp nhất
  - Lời giải: mã Huffman

diepht@vnu

Mã Huffman

- Biểu diễn mã tiền tố dưới dạng cây nhị phân
  - tại mỗi đỉnh, nhánh trái được gắn nhãn là 0, nhánh bên phải được gắn nhãn là 1
  - mỗi ký tự được lưu trong một đỉnh lá
  - từ mã của mỗi ký tự là xâu bit tạo thành từ các nhãn trên đường đi từ gốc tới đỉnh lá chứa ký tự đó
- Thuật toán Huffman sử dụng hàng ưu tiên để xây dựng mã tiền tố dưới dạng cây nhị phân
  - Mã sinh ra gọi là mã Huffman



Chữ cái	a	b	c	d	e	f
Tần suất (K)	45	13	12	16	9	5
Từ mã	0	101	100	111	1101	1100

diepht@vnu

Thuật toán Huffman

- Với mỗi ký tự xuất hiện trong xâu nguồn, ta tạo ra một đỉnh chứa ký tự đó
  - gắn với giá trị ưu tiên bằng tần suất
- Từ tập các cây chỉ có một đỉnh, tại mỗi bước ta kết hợp hai cây thành một cây
  - đỉnh cha sẽ gắn với giá trị ưu tiên bằng tổng độ ưu tiên các con
  - ta cần chọn hai cây nhị phân có mức ưu tiên nhỏ nhất để kết hợp thành một ⇨ dùng hàng ưu tiên.

Algorithm *HuffmanCoding(S,F)*

Input: Bảng chữ cái S và bảng các tần suất F

Output: cây Huffman

Tạo ra một đỉnh cho mỗi ký tự trong S

Khởi tạo hàng ưu tiên P chứa các đỉnh này

for *i*=0 to *n*-1 do

$v_1 \leftarrow$ P.removeMin()

$v_2 \leftarrow$ P.removeMin()

Tạo ra đỉnh v mới với

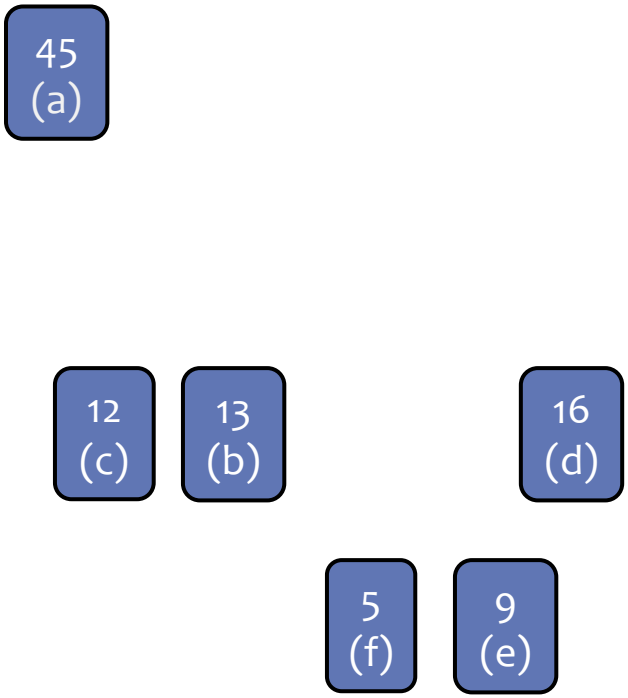
v.leftChild $\leftarrow v_1$

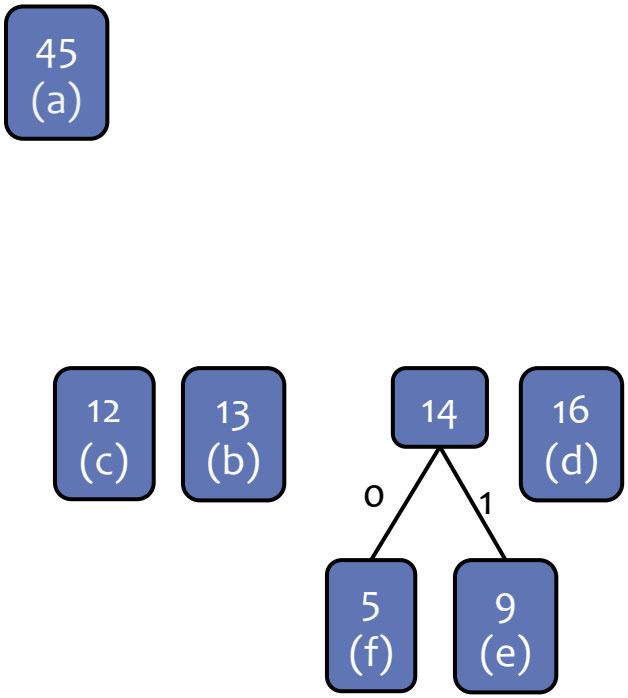
v.rightChild $\leftarrow v_2$

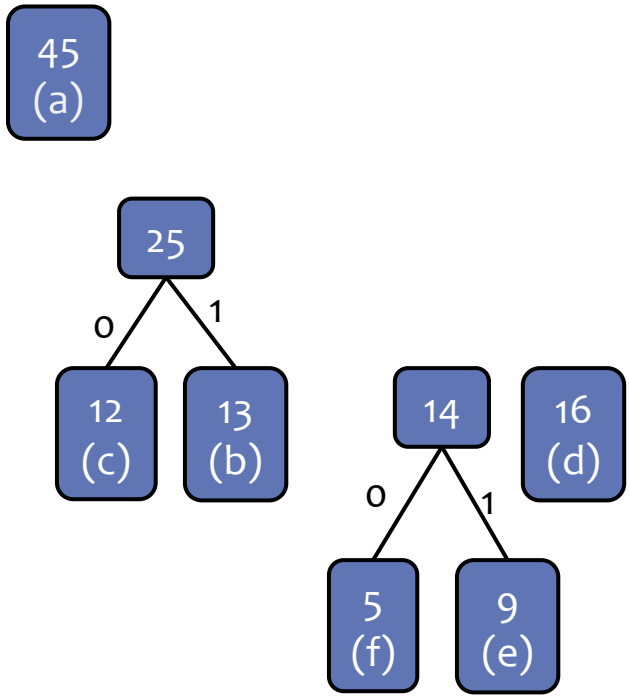
v.f $\leftarrow v_1$.f + v_2.f

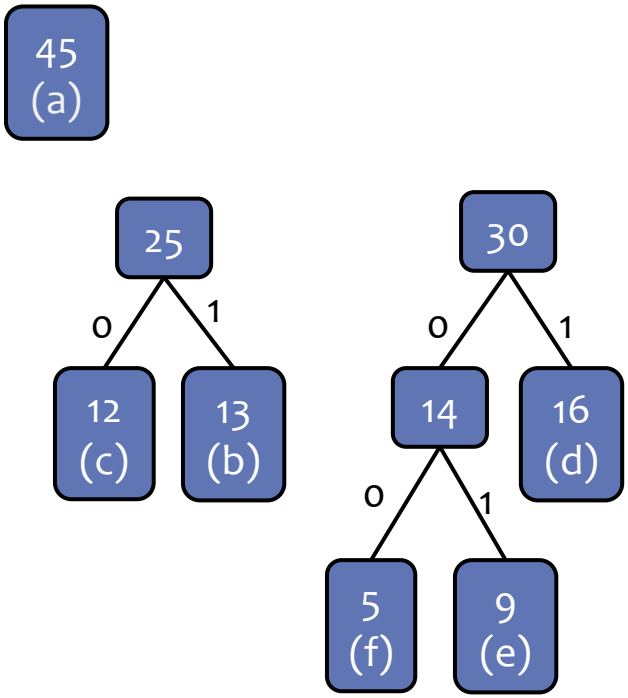
P.insert(v)

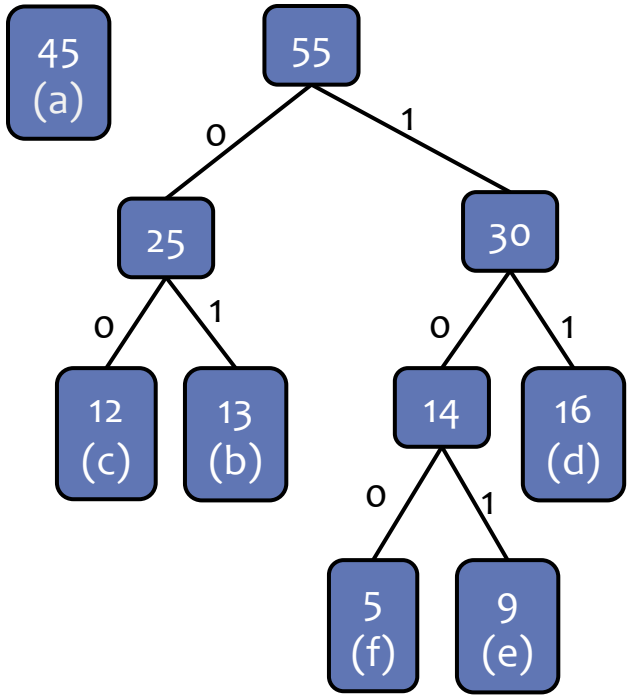
diepht@vnu


45
(a)
12
(c)
13
(b)
16
(d)
5
(f)
9
(e)

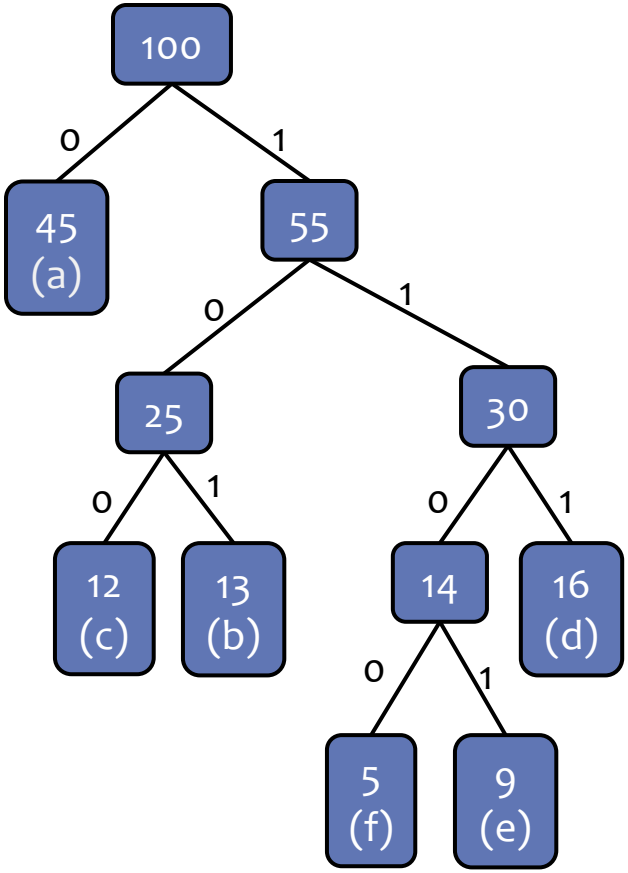

45
(a)
12
(c)
13
(b)
14
16
(d)
0
1
5
(f)
9
(e)


45
(a)
25
0
1
12
(c)
13
(b)
14
16
(d)
0
1
5
(f)
9
(e)


45
(a)
25
0
1
12
(c)
13
(b)
30
0
1
14
16
(d)
0
1
5
(f)
9
(e)



diepht@vnu


100
0
1
45
(a)
55
0
1
25
30
0
1
0
1
12
(c)
13
(b)
14
16
(d)
0
1
5
(f)
9
(e)

Mục tiêu bài học

1. KDLTT hàng ưu tiên
2. Các phương pháp cài đặt
3. Ứng dụng: xây dựng mã Huffman

Chuẩn bị tuần tới

- Lý thuyết: Đọc chương 16 giáo trình (Thiết kế thuật toán)
- Thực hành: Bảng băm