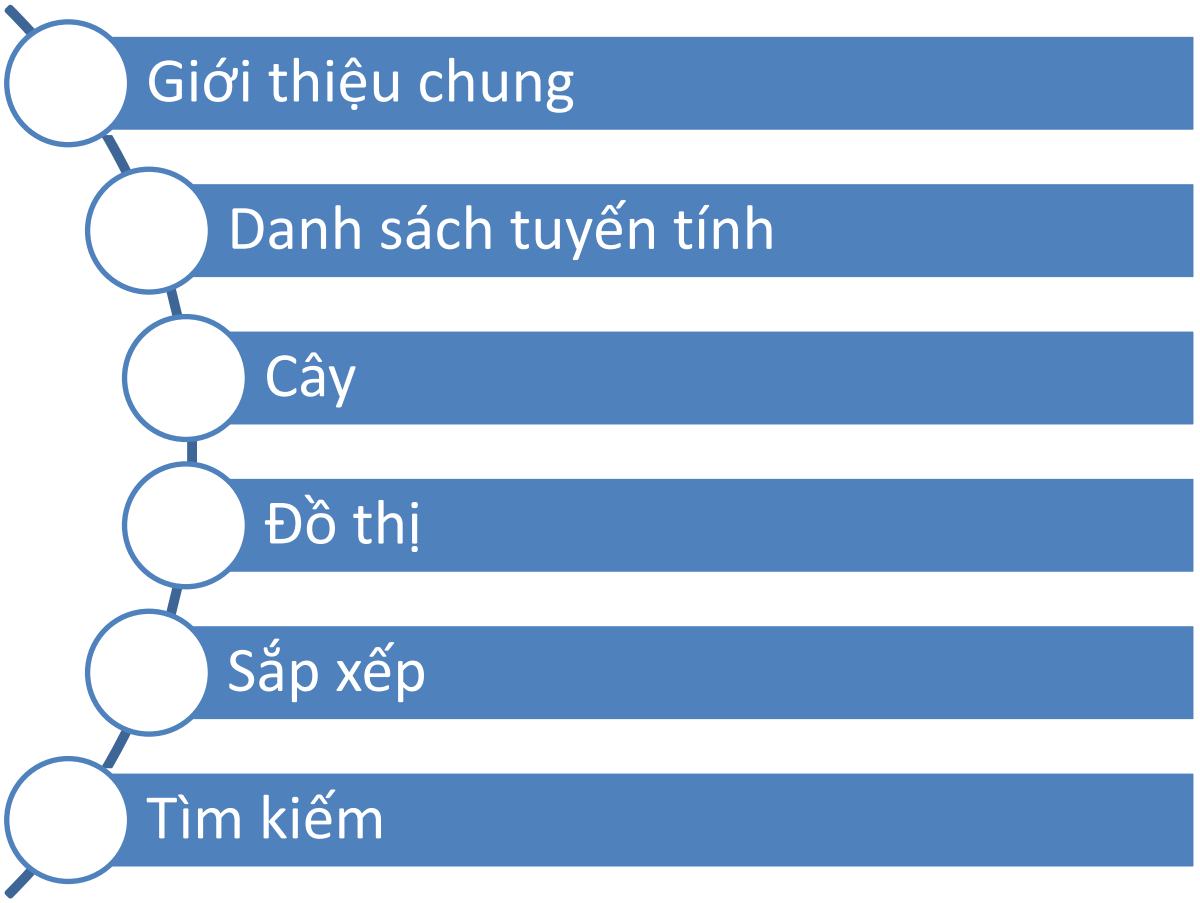
CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Vũ Song Tùng

NỘI DUNG


Giới thiệu chung
Danh sách tuyến tính
Cây
Đồ thị
Sắp xếp
Tìm kiếm

I. Giới thiệu chung

- Các kỹ thuật lập trình
- Ngôn ngữ giả lập trình
- Sơ lược về CTDL và giải thuật

1.1. Các kỹ thuật lập trình

- Lập trình hàm
  - Chia các giai đoạn của chương trình vào các hàm
  - Thể hiện những đoạn mã tương tự trong một hàm
  - Ưu điểm
    - Dễ quản lý
    - Tiết kiệm dung lượng chương trình

1.1. Các kỹ thuật lập trình

- Lập trình hướng đối tượng
  - Trừu tượng hóa các thành phần của chương trình
  - Đặc điểm
    - Tính đóng gói – các thuộc tính của một thành phần và các hàm xử lý các thuộc tính đó được định nghĩa trong cùng một khối
    - Tính đa hình – cho phép tạo ra nhiều hình thái của một loại thành phần
    - Tính kế thừa – cho phép sử dụng lại các định nghĩa đã có của một thành phần để tạo ra một thành phần mới

1.2. Ngôn ngữ giả lập trình

- Các quy ước

Các kiểu dữ liệu	Mô tả
$\mathbb{R}, \mathbb{Z}, \mathbb{N}$	Tập các số thực, nguyên và tự nhiên
$Type$	Kiểu bất kỳ
Array$[\ell..r]$**of** $Type$	Mảng chứa $r - \ell + 1$ phần tử kiểu T
Sequence of $Type$	Dãy
Set of $Type$	Tập hợp

1.2. Ngôn ngữ giả lập trình

- Các quy ước

Toán tử	Mô tả
$:=$	Gán
$=, \neq, <, >, \leq, \geq$	So sánh
$\wedge$, $\vee$, $\neg$	And, Or, Not logic
$a{+}{+}/a{-}{-}$	$a := a \pm 1$
$\lfloor x \rfloor, \lceil x \rceil$	Làm tròn xuống và lên
$(C ? I : J)$	Nếu C đúng thực hiện I, C sai thực hiện J
$\langle x_1, x_2, \ldots, x_n \rangle$	Dãy các phần tử
$\{x_1, x_2, \ldots, x_n\}$	Tập hợp các phần tử
;	Ngắt các biểu thức

1.2. Ngôn ngữ giả lập trình

- Biểu thức

Loại	Cú pháp
Khai báo biến	$x = v : Type$
Điều kiện	**if** C **then** I **if** C **then** I **else** J
Lặp xác định	**for** $i := a$ **to** b **do** I **foreach** $a \in s$ **do** I
Lặp không xác định	**while** C **do** I **repeat** I **until** C

1.2. Ngôn ngữ giả lập trình

- Biểu thức

Loại	Cú pháp
Tạo con trỏ	**pointer to** $Type$
Cấp phát bộ nhớ	$p \coloneqq$ **allocate** $Type$ $p \coloneqq$ **allocate Array**$[\ell..r]$**of** $Type$
Giải phóng bộ nhớ	**dispose** p
Lấy địa chỉ	**address of** p
Lấy nội dung trong con trỏ	$(* p)$
Truy cập thành viên của con trỏ	$p \rightarrow c$

1.2. Ngôn ngữ giả lập trình

- Hàm và thủ tục

```
procedure Power(x : ℝ; n : ℕ; var r : ℝ)
    r = 1 : ℝ
    for i := 1 to n do
        r := r * x
```

```
function Power(x : ℝ; n : ℕ) : ℝ
    if n = 0 then return 1
    else return x * Power(x, n − 1)
```

1.2. Ngôn ngữ giả lập trình

- Hướng đối tượng

class $Complex(x, y : \mathbb{R})$
    $r = x : \mathbb{R}$
    $i = y : \mathbb{R}$
    function modul : $\mathbb{R}$ **return** $\sqrt{r^2 + i^2}$
    function add $(c : Complex) : Complex$
        return $Complex(r + c.r, i + c.i)$

1.3. Sơ lược về CTDL và giải thuật

- Cấu trúc mảng
  - Tập hợp các phần tử cùng kiểu
    - được sắp xếp liên tiếp trong bộ nhớ
    - được xác định vị trí bằng chỉ số
  - Thường dùng để lưu trữ các danh sách tuyến tính có kích thước cố định

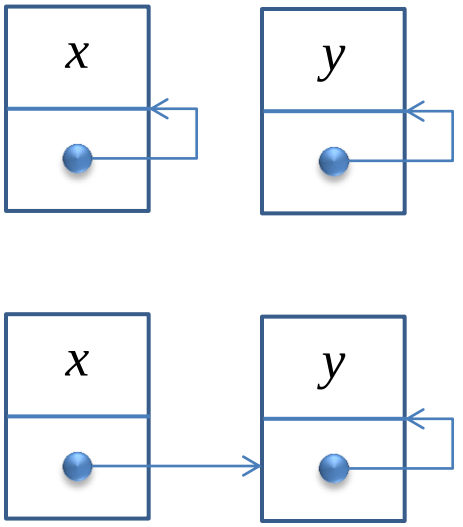
$$a : \textbf{Array}[\ell..r] \textbf{ of } Type$$
$$c \coloneqq \textbf{size of } Type$$
$$L_0 \coloneqq \textbf{address of } a[\ell]$$
$$L_i \coloneqq \textbf{address of } a[i]$$

$$\Rightarrow L_i = L_0 + (i - \ell)c$$

1.3. Sơ lược về CTDL và giải thuật

- Cấu trúc liên kết
  - Tập hợp các phần tử (item) lưu trữ dữ liệu (info) và địa chỉ liên kết (link) đến phần tử khác
  - Dùng để lưu trữ danh sách động hoặc các cấu trúc phi tuyến

class $Handle =$ **pointer to** $Item$
class $Item(i)$ **of** $Type$
    $info : Type$
    $link =$ **this** $: Handle$
...
$a \coloneqq Item(x)$
$b \coloneqq Item(y)$
$a.link \coloneqq$ **address of** b



1.3. Sơ lược về CTDL và giải thuật

- Giải thuật đệ quy
  - Giải thuật gọi lại chính nó, gồm 2 thành phần:
    - Điều kiện dừng đệ quy
    - Gọi đệ quy
  - Áp dụng giải thuật đệ quy khiến cho thuật toán trở nên mạch lạc

1.3. Sơ lược về CTDL và giải thuật

- Độ phức tạp của thuật toán $O(\ldots)$
  - Số lượng trung bình (T_{cp}) của phép toán phức tạp nhất trong thuật toán
  - Độ phức tạp của các thuật toán sắp xếp và tìm kiếm trong dãy n phần tử:

Thuật toán	Độ phức tạp
SelectSort	$O(n^2)$
InsertSort	$O(n^2)$
BubbleSort	$O(n^2)$
QuickSort	$O(n\log_2 n)$
HeapSort	$O(n\log_2 n)$
LinearSearch	$O(n)$
BinarySearch	$O(\log_2 n)$

II. Danh sách tuyến tính

- Ngăn xếp
- Hàng đợi
- Danh sách liên kết

2.1. Ngăn xếp (Stack)

- Dãy các phần tử được thêm vào (push) và lấy ra (pop) tại cùng một đầu của dãy

$$S = \langle e_1, e_2, \dots, e_{t-1}, e_t \rangle$$
$$e \mapsto S \Rightarrow S := \langle e_1, e_2, \dots, e_t, e_{t+1} \rangle$$
$$e \mapsfrom S \Rightarrow (e, S) := (e_t, \langle e_1, e_2, \dots, e_{t-1} \rangle)$$

2.1. Ngăn xếp (Stack)

- Định nghĩa ngăn xếp bằng mảng

class Stack($n : \mathbb{N}$) **of** $Type$
    $t = 0 : \mathbb{N}$
    s : **Array**$[1..n]$ **of** $Type$

    procedure Push(e : $Type$) $t{+}{+}; s[t] \coloneqq e$
    function Pop : $Type$
        $t' = t; t{-}{-}$
        return $s[t']$
    function IsEmpty : $\{0,1\}$ **return** $(t < 1 ? 1 : 0)$

2.2. Hàng đợi (Queue)

- Dãy các phần tử được thêm vào (enqueue) và lấy ra (dequeue) tại hai đầu khác nhau của dãy

$$Q = \langle e_f, e_{f+1}, \dots, e_r \rangle$$
$$e \mapsto Q \Rightarrow Q := \langle e_f, e_{f+1}, \dots, e_r, e_{r+1} \rangle$$
$$e \leftarrow Q \Rightarrow (e, Q) := \left(e_f, \langle e_{f+1}, \dots, e_r \rangle\right)$$

2.2. Hàng đợi (Queue)

- Định nghĩa hàng đợi bằng mảng

```
class Queue(n : ℕ) of Type
    f = r = 0 : ℕ
    c = 0 : ℕ
    q : Array[1..n] of Type

    procedure Inc(var i : ℕ) (i = n? i := 1 : i++)
    procedure Enqueue(e : Type)
        (c = 0? f := r := 1: Inc(r)); q[r] := e; c++
    function Dequeue : Type
        f' = f; (c = 1? f := r := 0 : Inc(f)); c--
        return q[f']
    function Count : ℕ; return c
```

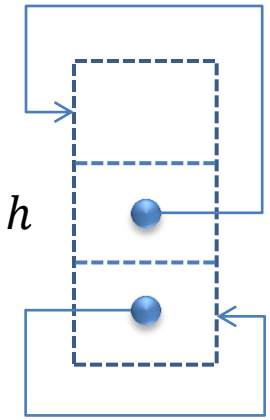
2.3. Danh sách liên kết

- Danh sách liên kết 2 chiều
  - Dãy các phần tử liên kết với phần tử phía sau (next) và phía trước (prev) trong dãy

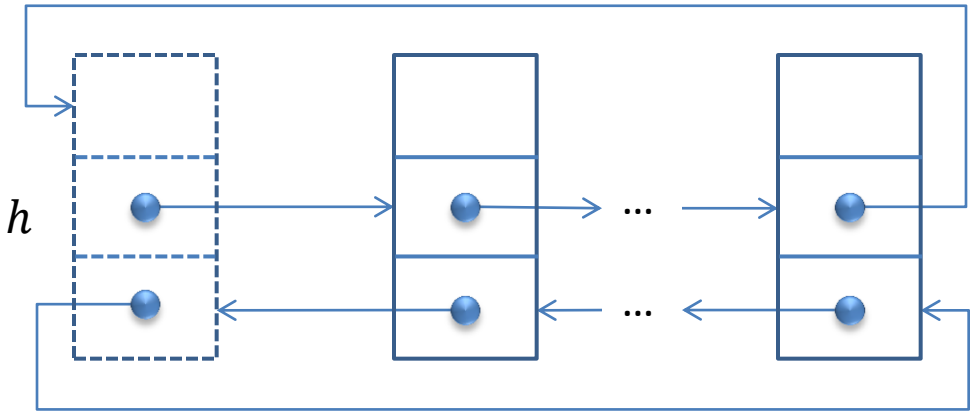
$$L = \langle first, \dots, p, q, \dots, last \rangle$$
$$p \rightarrow next = q$$
$$q \rightarrow prev = p$$

2.3. Danh sách liên kết

- Mô hình



Danh sách rỗng



$h.next$ – phần tử đầu danh sách
$h.prev$ – phần tử cuối danh sách

2.3. Danh sách liên kết

- Định nghĩa – **class** $Item$

class $Handle$ **of** $Type$
class $Item(i)$ **of** $Type$

$info = i : Type$

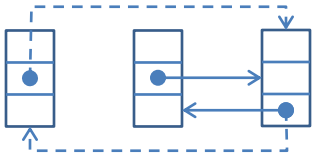
$next =$ **this** $: Handle$

$prev =$ **this** $: Handle$

procedure insertAfter$(p : Handle)$

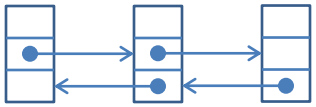
$p \rightarrow next := next$

$next \rightarrow prev := p$



$next := p$

$p \rightarrow prev :=$ **this**



2.3. Danh sách liên kết

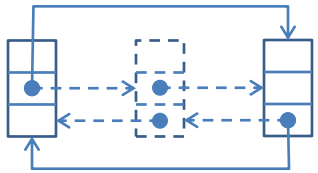
- Định nghĩa – **class** $Item$

procedure remove(p : $Handle$)

p → $prev$ → $next$:= p → $next$

p → $next$ → $prev$:= p → $prev$

dispose p



procedure makeEmpty

p := $next$

while $p \neq$ **this**

$next$:= p → $next$

dispose p

p := $next$

$prev$:= **this**

2.3. Danh sách liên kết

- Định nghĩa – **class** $List$

class $List$ **of** $Type$

$h(\emptyset) : Item$

function newItem$(e : Type) : Handle$ **return allocate** $Item(e)$

procedure pushBegin$(e : Type)$ $h.$insertAfter$\big($newItem$(e)\big)$

procedure pushBegin$(e : Type)$ $h.prev \rightarrow$ insertAfter$\big($newItem$(e)\big)$

procedure popBegin $p := h.next;$ $h.$remove(p)

procedure popEnd $p := h.prev;$ $h.$remove(p)

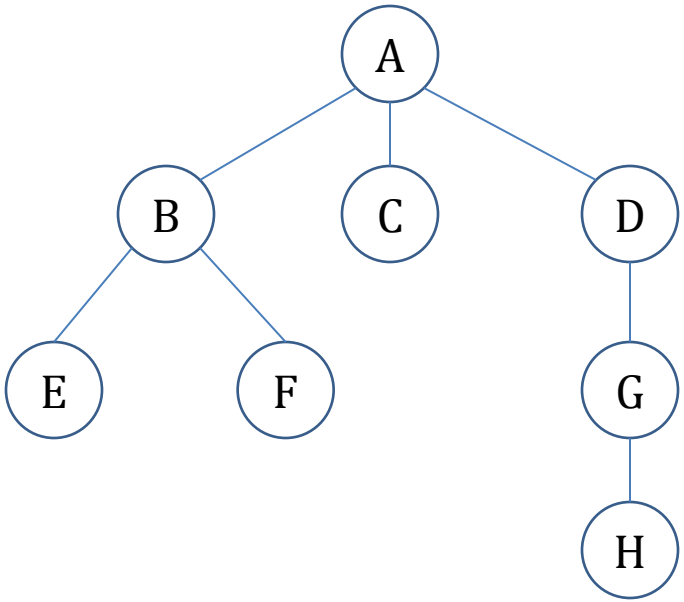
procedure removeAll $h.$makeEmpty

III. Cây

- Các khái niệm
- Cây nhị phân
- Một vài ứng dụng

3.1. Các khái niệm

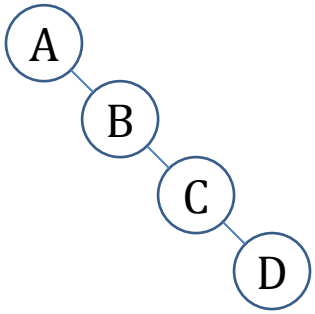
$$T : \{T_1, T_2, \dots, T_n\} \mid_{T_i - subtree}$$



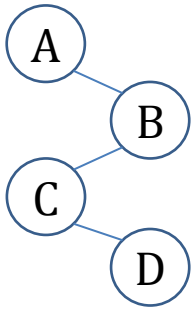
Các khái niệm	
Gốc	A
Cành	B, C, D, G
Lá	C, E, F, H
Cấp	3
Chiều cao	4
Mức 0	A
Mức 1	B, C, D
Mức 2	E, F, G
Mức 3	H

3.2. Cây nhị phân

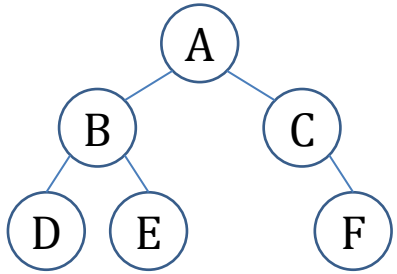
$$T : \{T_{left}, T_{right}\}$$



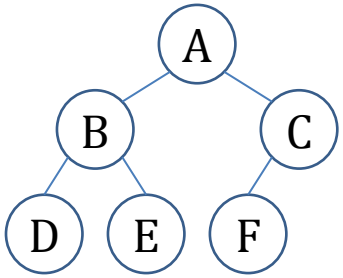
Cây lệch phải



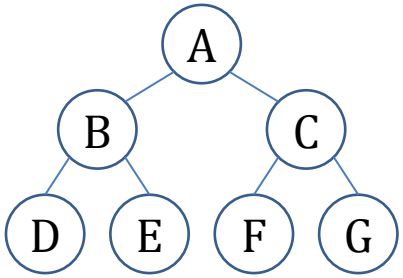
Cây zíc-zắc



Cây gần đầy



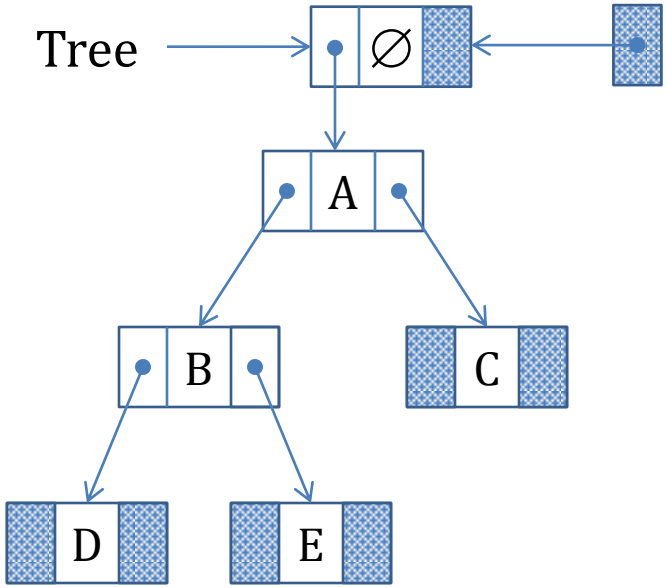
Cây đầy đủ



Cây hoàn chỉnh

3.2. Cây nhị phân

- Mô hình lưu trữ
  - Mỗi nút trên cây được coi là một cây con gồm một trường chứa dữ liệu (info) và hai địa chỉ liên kết với cây con trái (left) và phải (right)



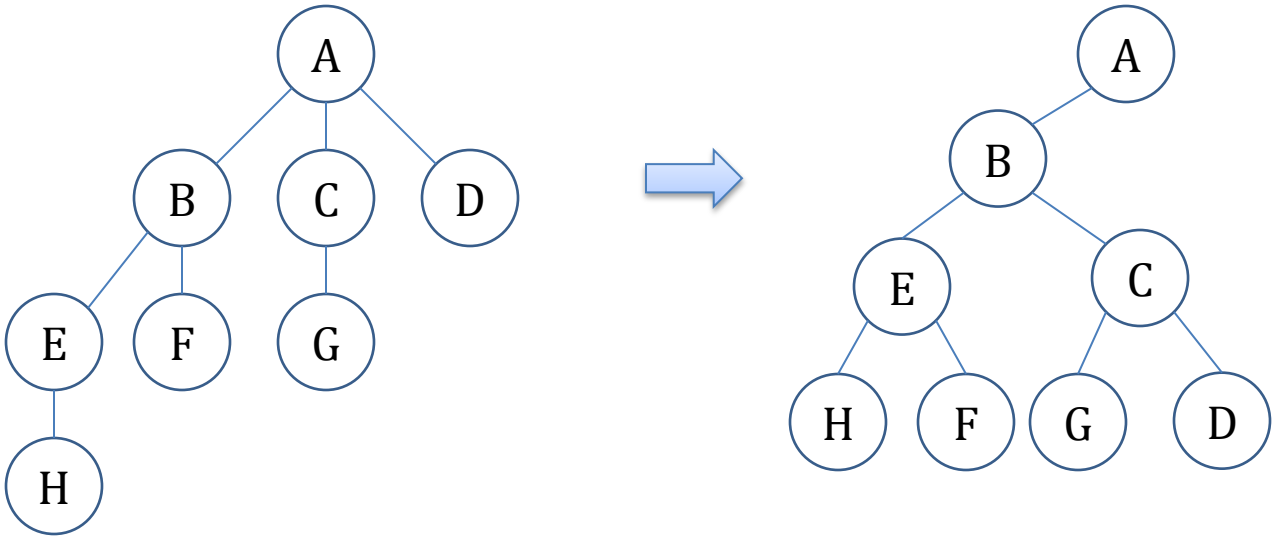
3.2. Cây nhị phân

- Định nghĩa – **class** $BinaryTree$

class $Handle =$ **Pointer to** $BinaryTree$
class $BinaryTree(i : Type; t : Handle)$ **of** $Type$
    $e = i : Type$
    $left = t : Handle$
    $right = t : Handle$
    function $isEmpty : \{0, 1\}$ **return** $(e = \emptyset ? 1 : 0)$
    procedure postOrderTraverse
        if $\neg isEmpty$ **then**
            $left \rightarrow$ postOrderTraverse;
            $right \rightarrow$ postOrderTraverse;
            do something

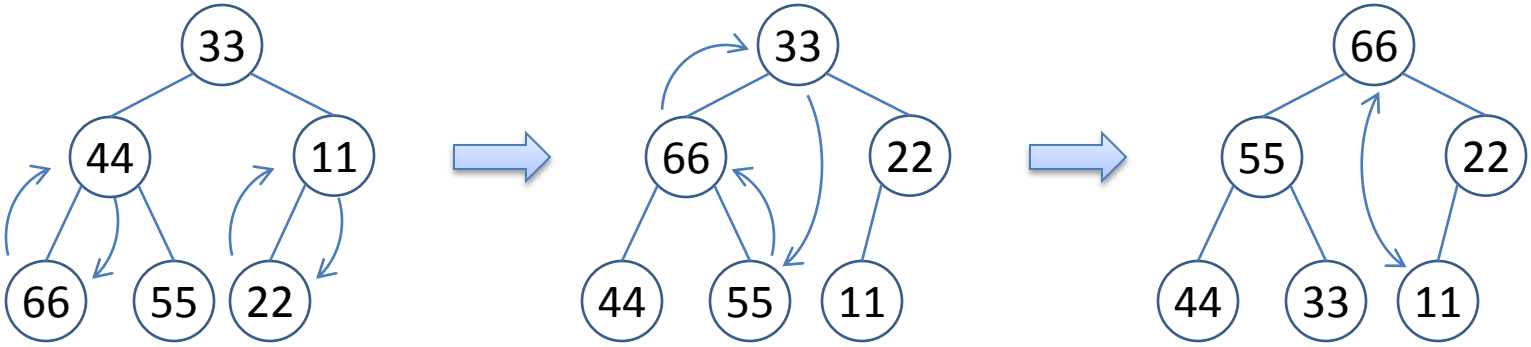
3.3. Một vài ứng dụng

- Biểu diễn cây tổng quát
  - Con đầu tiên (First Child) → left
  - Em liền kề (Next Sibling) → right



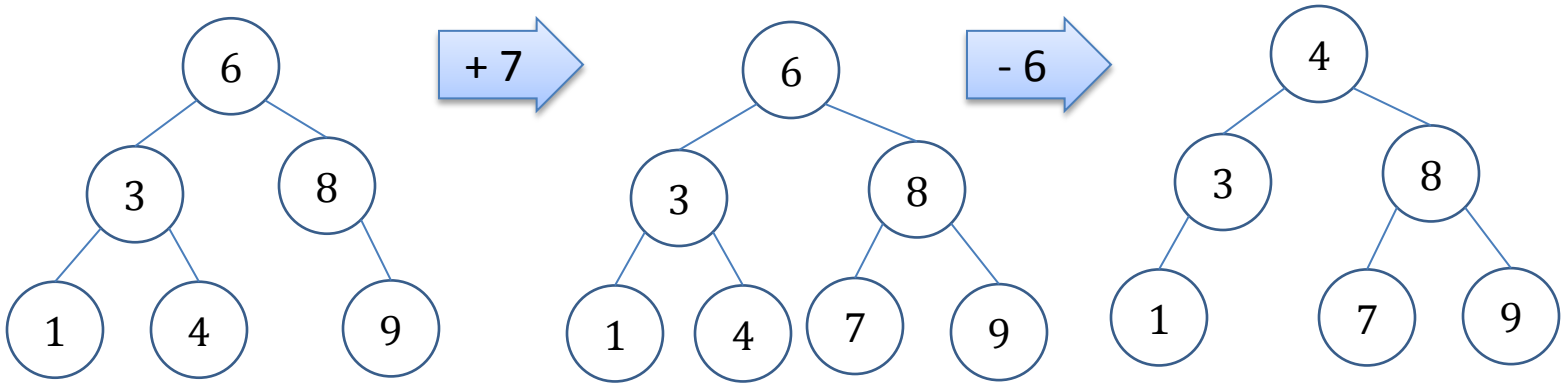
3.3. Một vài ứng dụng

- Sắp xếp vun đống
  - Có thể coi mảng là cây nhị phân đầy đủ (hoặc hoàn chỉnh)
  - Đối với mảng $a[\ell..r] = \langle 33,44,11,66,55,22 \rangle$
    - $a[\ell]$ là nút gốc
    - $a[i]$ có các con $a[j], a[j+1], j = 2i - \ell + 1$
  - Vun đống là thuật toán để cây có tính chất của đống (heap – nút cha > nút con)



3.3. Một vài ứng dụng

- Cây nhị phân tìm kiếm
  - Tất cả các nút của cây con trái phải nhỏ hơn gốc và nhỏ hơn tất cả các nút của cây con phải
  - Sử dụng trong các danh sách thường xuyên phải xử lý các thao tác thêm, xóa dữ liệu



IV. Đồ thị

- Khái niệm
- Lưu trữ đồ thị
- Duyệt đồ thị
- Các ứng dụng

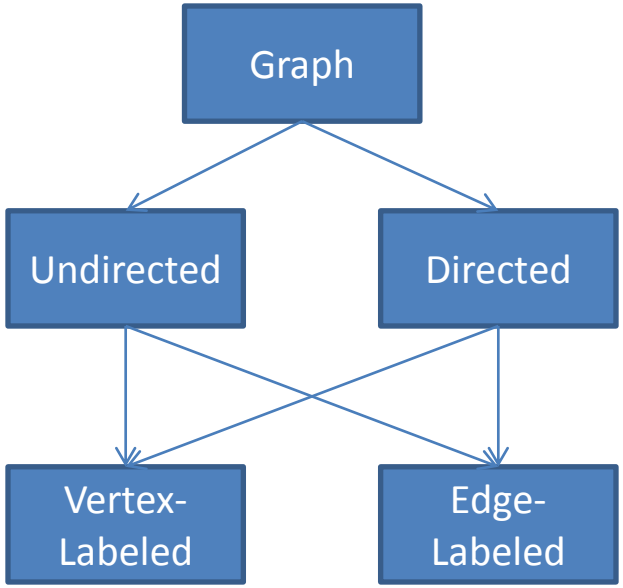
4.1. Khái niệm

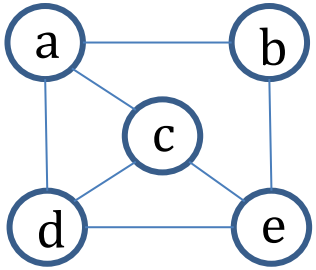
- Đồ thị (graph) được xác định bằng tập các đỉnh (vertex) và tập các cung (edge) giữa các đỉnh

$$G = (V, E)$$
$$e = (u, v)|_{u,v \in V; e \in E}$$

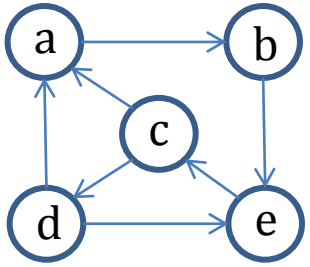
4.1. Khái niệm

- Phân loại

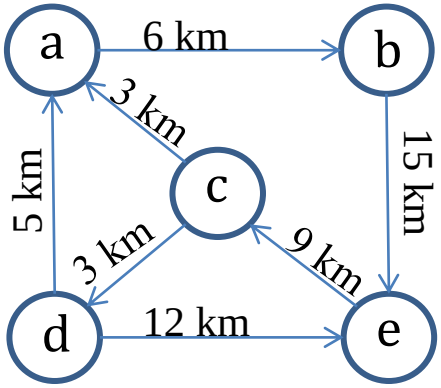




Undirected graph



Directed graph

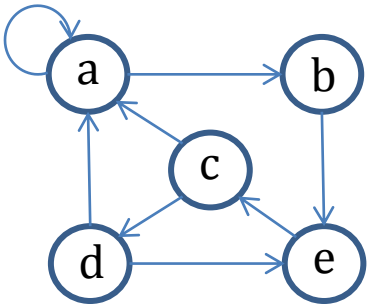


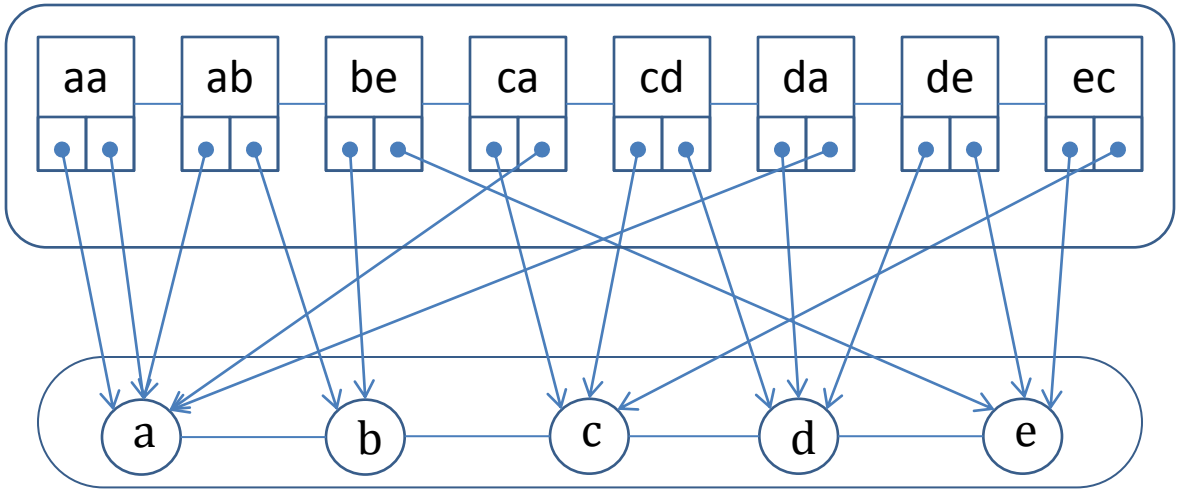
Edge-Labeled graph

4.2. Lưu trữ đồ thị

- Danh sách các cung (Edge List)

$$G : List \textbf{ of } Edge$$
$$Edge(u, v : \textbf{Pointer to } Vertex)$$

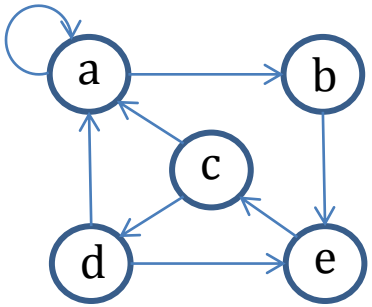




4.2. Lưu trữ đồ thị

- Ma trận lân cận kề (Adjacency Matrix)

$$G : \textbf{Array}[1..n] \textbf{ of } \textbf{Array}[1..n] \textbf{ of } Edge$$
$$Edge(u, v) = G[u, v]$$

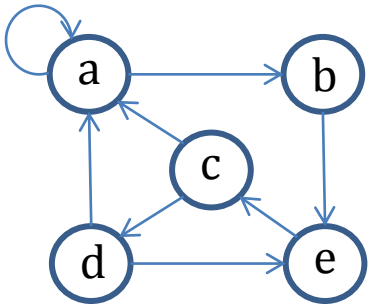


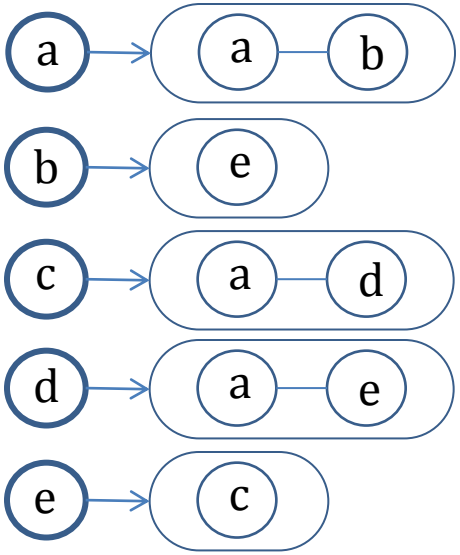
	a	b	c	d	e
a	1	1	0	0	0
b	0	0	0	0	1
c	1	0	0	0	1
d	1	0	0	0	1
e	0	0	1	0	0

4.2. Lưu trữ đồ thị

- Danh sách lân cận kề (Adjacency List)

$$G : \textbf{Array}[1..n] \textbf{ of } List \textbf{ of } Vertex$$





4.3. Duyệt đồ thị

- Tìm theo chiều sâu (Depth-First Search)

```
procedure DFS(v : Vertex)
    if v is not marked then
        mark v
        foreach (v, w) ∈ E do
            if w is not marked then DFS(w)
```

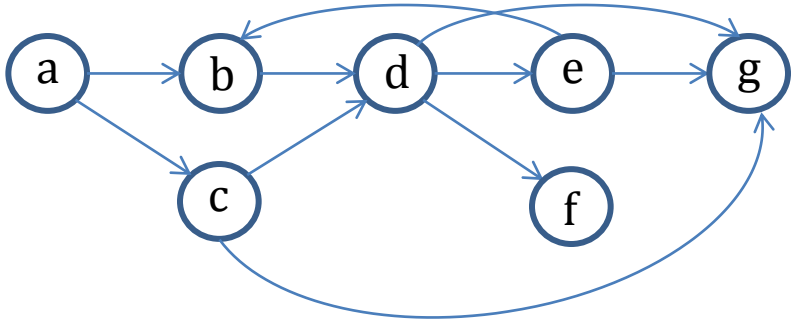
4.3. Duyệt đồ thị

- Tìm theo chiều rộng (Breadth-First Search)

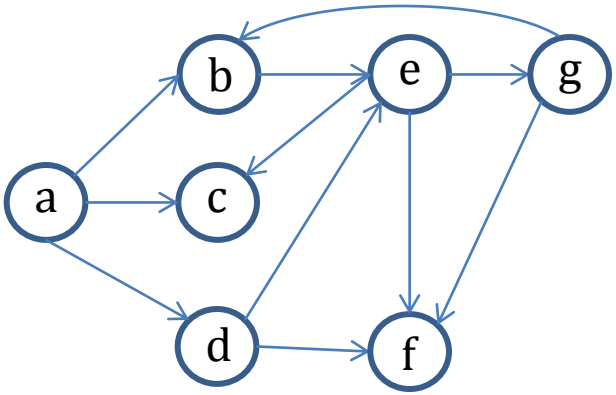
```
procedure BFS(v : Vertex)
    q : Queue(n) of Vertex
    mark v
    q.Enqueue(v)
    while ¬q.IsEmpty do
        u := q.Dequeue
        foreach (u, w) ∈ E do
            if w is not marked then
                mark w
                q.Enqueue(w)
```

4.3. Duyệt đồ thị

- Ví dụ



$DFS(a) \Rightarrow$ a, b, d, e, g, f, c



$BFS(a) \Rightarrow$ a, b, c, d, e, f, g

4.4. Các ứng dụng

- Kiểm tra kết nối giữa hai đỉnh (Connectedness)
- Tìm đường đi ngắn nhất giữa hai đỉnh
- Tìm chi phí thấp nhất (Min Cose Spanning Tree)
- Sắp xếp topo (Topological Sorting)

V. Sắp xếp

- Các thuật toán đơn giản
- Sắp xếp nhanh
- Sắp xếp vun đống
- Thuật toán hòa nhập

5.1. Các thuật toán đơn giản

- Thuật toán lựa chọn

```
procedure SelectSort(a : Array[ℓ..r] of Type)
    for i := ℓ to r − 1 do
        m := i
        for j := i + 1 to r do
            if a[j] < a[m] then m := j
        if m ≠ i then Swap(a[i], a[m])
```

5.1. Các thuật toán đơn giản

- Ví dụ - SelectSort

i	m	a					
		1	2	3	4	5	6
		33	44	11	66	55	22
1	1, 3	11	44	33	66	55	22
2	2, 3, 6	11	22	33	66	55	44
3	3	11	22	33	66	55	44
4	4, 5, 6	11	22	33	44	55	66
5	5	11	22	33	44	55	66

$$T = \frac{6 \times (6-1)}{2} = 15$$

5.1. Các thuật toán đơn giản

- Thuật toán chèn

```
procedure InsertSort(a : Array[ℓ..r] of Type)
    for i := ℓ + 1 to r do
        e := a[i]
        j := i − 1
        while j ≥ ℓ ∧ a[j] > e do a[j + 1] := a[j]; j−−
        a[j + 1] := e
```

5.1. Các thuật toán đơn giản

- Ví dụ - InsertSort

i	e	j	a					
			1	2	3	4	5	6
			33	44	11	66	55	22
2	44	1	33	44				
3	11	$2 \rightarrow 0$	11	33	44			
4	66	3	11	33	44	66		
5	55	$4 \rightarrow 3$	11	33	44	55	66	
6	22	$5 \rightarrow 1$	11	22	33	44	55	66

$$T = 11$$

5.1. Các thuật toán đơn giản

- Thuật toán nổi bọt

```
procedure BubbleSort(a : Array[ℓ..r] of Type)
    for i := ℓ to r − 1 do
        c = 0 : {0, 1}
        for j := r downto i + 1 do
            if a[j − 1] > a[j] then
                Swap(a[j − 1], a[j]); c := 1
        if c = 0 then return
```

5.1. Các thuật toán đơn giản

- Ví dụ - BubbleSort

i	j và $j-1$	a					
		1	2	3	4	5	6
		33	44	11	66	55	22
1	6↔5↔4, 3↔2↔1	11	33	44	22	66	55
2	6↔5, 4↔3↔2	11	22	33	44	55	66
3	6, 5, 4, 3						

$$T = 12$$

5.2. Sắp xếp nhanh

- Tương quan giữa các phần tử của mảng đã được sắp xếp tăng dần

$$a_\ell < \cdots < a_{j-1} < a_j < a_{j+1} < \cdots < a_r$$

procedure QuickSort(a : **Array of** $Type$; ℓ, r : $\mathbb{N}$)
    if $\ell < r$ **then**
        $j \coloneqq \text{Part}(a, \ell, r)$
        QuickSort$(a, \ell, j-1)$
        QuickSort$(a, j+1, r)$

5.2. Sắp xếp nhanh

- Thuật toán phân đoạn

$$\langle a_\ell, \dots, a_{j-1} \rangle < a_j < \langle a_{j+1}, \dots, a_r \rangle$$

```
function Part(a : Array of Type; ℓ, r : ℕ)
    i := ℓ + 1; j := r
    while i ≤ j do
        while i ≤ j ∧ a[i] < a[ℓ] do i++
        while i ≤ j ∧ (i = j ∨ a[j] > a[ℓ]) do j−−
        if i < j then
            Swap(a[i], a[j]) ; i++; j−−
    Swap(a[ℓ], a[j])
    return j
```

5.2. Sắp xếp nhanh

- Ví dụ - Part

i	j	a					
		1	2	3	4	5	6
		33	44	11	66	55	22
2	6		22				44
$3 \rightarrow 4$	$3 \leftarrow 5$	11	22	33	66	55	44

$$T = 5$$

5.3. Sắp xếp vun đống

- Sắp xếp $\langle a_1, \ldots, a_n \rangle$ sao cho

$$a_i > \langle a_{2i}, a_{2i+1} \rangle \; \forall \, i \in \left[1, \left\lfloor \frac{n}{2} \right\rfloor\right]$$

```
procedure HeapSort(a : Array[1..n] of Type)
    for i = ⌊n/2⌋ downto 1 do BuildHeap(a, i)
    while n > 1 do  // sorting
        Swap(a[1], a[n]) ; n−−
        while n > 1 then BuildHeap(a, 1)
```

5.3. Sắp xếp vun đống

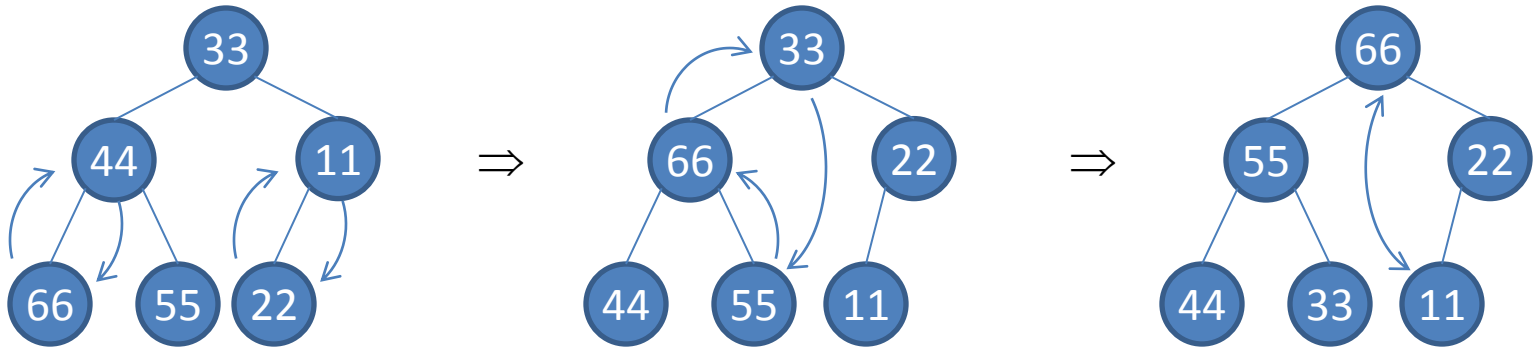
- Thuật toán vun đống

```
procedure BuildHeap(a : Array[1..n] of Type; i : ℕ)
    e := a[i]; j := i × 2
    while j ≤ n do
        if j + 1 ≤ n ∧ a[j] < a[j + 1] then j++
        if a[j] > e then
            a[i] := a[j]
            i := j; j := i × 2
        else exit while
    a[i] := e
```

5.3. Sắp xếp vun đống

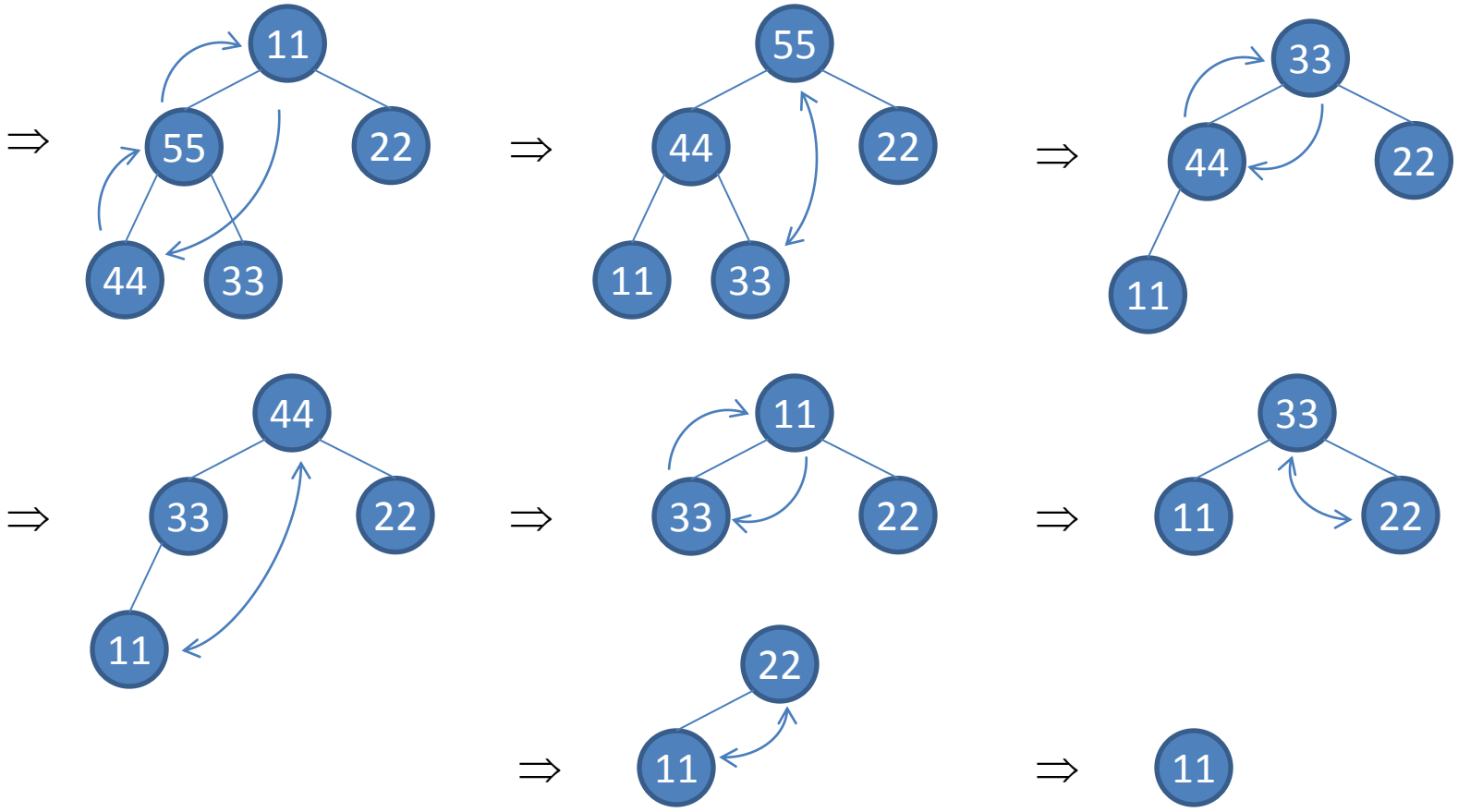
- Ví dụ – $\text{BuildHeap}(a, i|_{i=3;2;1})$

n	i	e	j	a					
				1	2	3	4	5	6
				33	44	11	66	55	22
6	3, 6	11	6, 12			22			11
6	2, 4	44	4, 8		66		44	55	
6	1, 5	33	2, 4+1, 10	66	55	22	44	33	11



5.3. Sắp xếp vun đống

- Ví dụ – sorting



VI. Tìm kiếm

- Tìm kiếm tuần tự
- Tìm kiếm chia đôi
- Cây nhị phân tìm kiếm

6.1. Tìm kiếm tuần tự

- Tìm kiếm tuần tự (Linear Search)
  - Áp dụng cho các mảng chưa sắp xếp

```
function LinearSearch(a : Array[1..n] of Type; x : Type) : ℕ
    i := 1
    while i ≤ n ∧ a[i] ≠ x do i++
    if i ≤ n return i
    else return 0
```

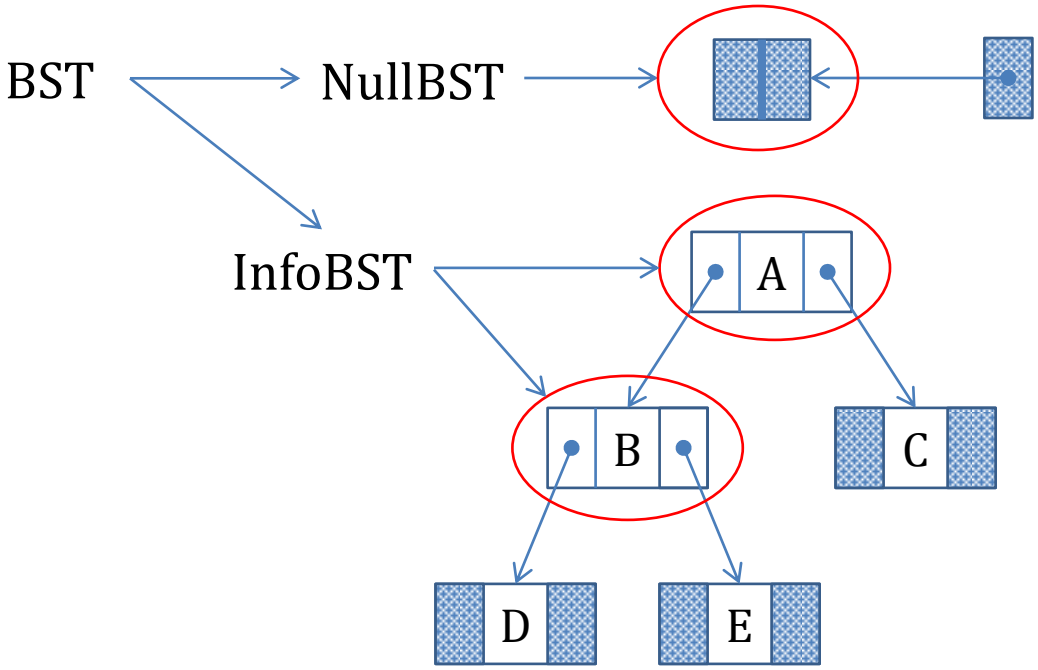
6.2. Tìm kiếm chia đôi

- Tìm kiếm chia đôi (Binary Search)
  - Áp dụng cho mảng đã được sắp xếp tăng dần

```
function BinarySearch(a : Array[1..n] of Type; x : Type) : ℕ
    ℓ := 1; r := n
    while ℓ ≤ r do
        m := ⌊(ℓ+r)/2⌋
        if a[m] = x then return m
        if a[m] > x
            then r := m − 1
            else ℓ := m + 1
    return 0
```

6.3. Cây nhị phân tìm kiếm

- Mô hình kế thừa



6.3. Cây nhị phân tìm kiếm

- Định nghĩa **class** BST và **class** $NullBST$

class $Handle =$ **poiter to** BST
class $BST(\ell, r : Handle)$ **of** $Type$
    $left = \ell : Handle$
    $right = r : Handle$
    function isEmpty : $\{0,1\}$ **return** 1
    function search($e : Type$) : $\{0,1\}$ **return** 0
    function insert($e : Type$) : $Handle$ **return this**
    function remove($e : Type$) : $Handle$ **return this**
    function findMax : $Handle$ **return** 0

class $NullBST$ **inherit** BST **of** $Type$
    function insert($e : Type$) : $Handle$
        return allocate $InfoBST(e,$ **this**, **this**$)$

6.3. Cây nhị phân tìm kiếm

- Định nghĩa **class** $InfoBST$

class $InfoBST(e : Type;\ \ell, r : Handle)$ **inherit** BST **of** $Type$
    $info = e : Type$
    function isEmpty : $\{0, 1\}$ **return** 0
    function search$(e : Type) : \{0, 1\}$
        if $e = info$ **then** **return** 1
        else if $e < info$ **then** **return** $left \rightarrow$ search(e)
        else **return** $right \rightarrow$ search(e)
    function insert$(e : Type) : Handle$
        if $e < info$ **then** $left = left \rightarrow insert(e)$
        else if $e > info$ **then** $right = right \rightarrow insert(e)$
        return this

6.3. Cây nhị phân tìm kiếm

- Định nghĩa **class** $InfoBST$

```
function remove(e : Type) : Handle
    if e < info then          left = left → remove(e)
    else if e > info then     right = right → remove(e)
    else    r := left → findMax
            if r = 0 then     r := right
                              dispose this
                              return r
            else              info := r → info
                              left := left → remove(info)
    return this
function findMax  : Handle
    return (right → isEmpty? this : right → findMax)
function findMin  : Handle
    return (left → isEmpty? this : left → findMin)
```