Báo cáo DSA

# Mở đầu

Trong bài viết này nhóm chúng mình sẽ nói về Binary Heap, ….

Chúng ta sẽ nói chủ yếu về cây nhị phân

…

Chưa nghĩ ra văn, mà có nên viết phần mở đầu để giới thiệu không ta.

# Lịch sử hình thành hàng đợi ưu tiên

* John William Joseph Williams là một nhà khoa học máy tỉnh nổi tiếng với việc phát minh ra thuật toán heapsort và cấu trúc dữ liệu binary heap vào năm 1963 trong khi đang công tác tại Eliot Bros (London).
* Đống nhị phân (Binary Heap): là một cấu trúc dữ liệu đồng có dạng cây nhị phân. Nó là một cách phổ biến để triển khai hàng đợi ưu tiên.
* Đống nhị phân được JWJ Williams giới thiệu vào năm 1964 như một cấu trúc dữ liệu để triển khai thuật toán heapsort.

# Một số khái niệm

## Cây

### Định nghĩa về cây

Cây bao gồm các nút, có một nút đặc biệt được gọi là nút gốc (root) và các cạnh nối các nút. Cây được định nghĩa đệ quy như sau:

* Bước cơ sở: một nút được gọi là cây và được gọi là gốc cây.
* Bước đệ quy: Giả sử là các cây với gốc là , ta có thể xây dựng cây mới bằng cách đặt làm nút cha (parent) của các nút độc . Trong cây mới tạo ra là gốc và là các cây con của gốc . Các nút được gọi là con của nút .

A diagram of a triangle with a circle and a point

Description automatically generated

Hình minh họa định nghĩa đệ quy của cây

### Các thuật ngữ

* Gốc (root): Nút đầu tiên tạo nên cây.
* Tổ tiên (ancestors): Nút là tổ tiên của nút nếu đường đi từ nút gốc đến phải đi qua
* Cha (parent): Nút là cha của nút khi đường đi từ nút gốc đến phải đi qua , và được nối với
* Con (child): Nút có một nút cha.
* Lá (leaf): Nút không có nút con.
* Hậu duệ (descendants): là hậu duệ của khi là tổ tiên của .
* Anh em (sibling): Các nút có chung cha.
* Mức (level): Số bước từ gốc đến một nút.
* Chiều cao (height): Số cạnh từ gốc đến nút xa nhất.
* Nút trong (internal node)

A diagram of a network

Description automatically generated

Nút xanh thẫm là nút gốc, các nút xanh lá là nút lá

Cây này có chiều cao là 4

A diagram of a network

Description automatically generated

Các nút b,c,d là các nút anh em

A diagram of a network with Silverstone Circuit in the background

Description automatically generated

Cây con của a có gốc b

**Ở trong bài viết này, chúng mình sẽ chỉ tìm hiểu và làm việc với cây nhị phân**

## Cây nhị phân

### Định nghĩa

Cây nhị phân là một cấu trúc dữ liệu mà mỗi node cha có tối đa 2 node con.

Các thuật ngữ về cây nhị phân giống như các thuật ngữ về cây đã giới thiệu ở trên.

### Các kiểu cây nhị phân

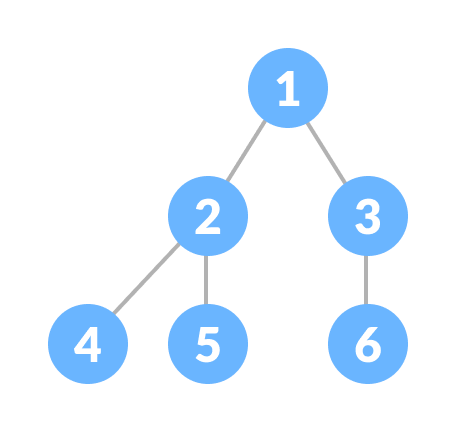
* Cây nhị phân đầy đủ (full binary tree): Mỗi node cha có 0 hoặc 2 node con.



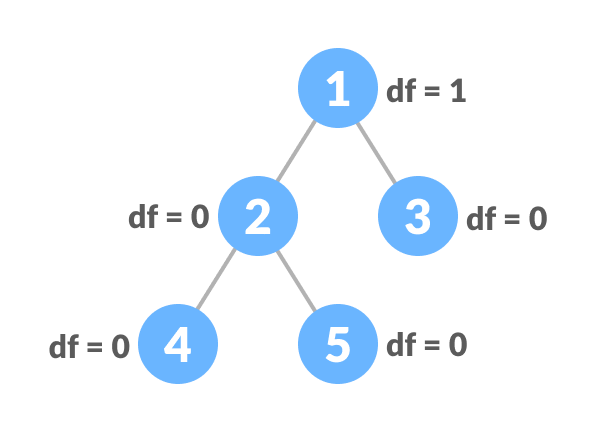
* Cây nhị phân hoàn hảo (Perfect binary tree): Mỗi node trung gian đều có 2 con và mọi node lá đều cùng mức.



* Cây nhị phân hoàn chỉnh (Complete binary tree): Tất cả các node lá phải tinh gọn về bên trái, Node lá cuối cùng không nhất thiết phải có anh em bên phải.



* Cây nhị phân cân bằng (Balanced binary tree): Mọi node trên cây có độ cao của cây con bên trái và độ cao của cây con bên phải chênh lệch nhau không quá 1.



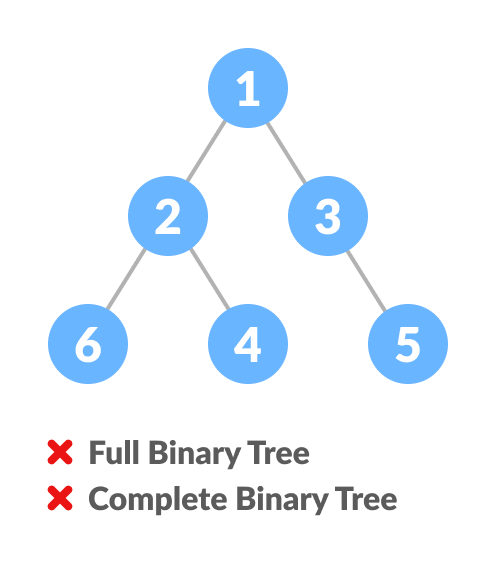
Cây nhị phân cân bằng

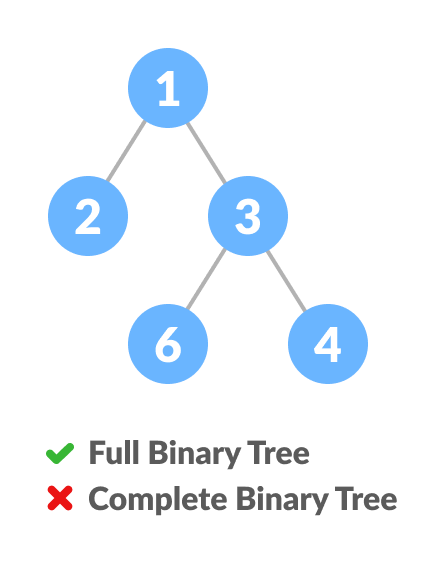
A diagram of numbers and equations

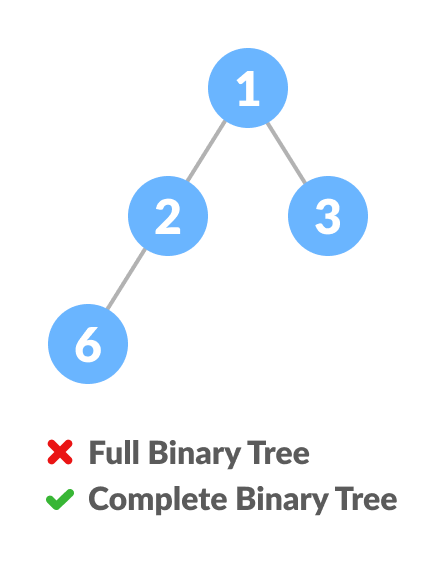
Description automatically generated

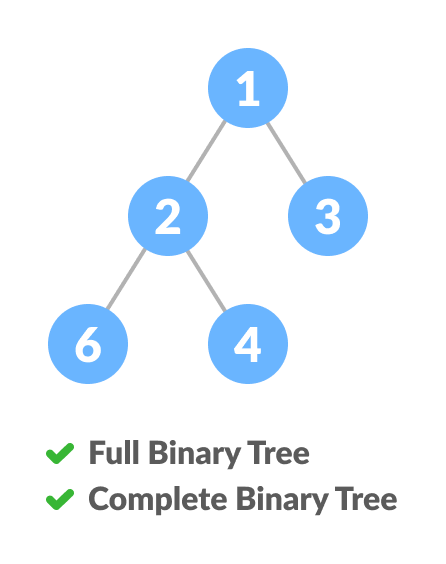
Cây nhị phân không cân bằng

### Phân biệt các loại cây nhị phân







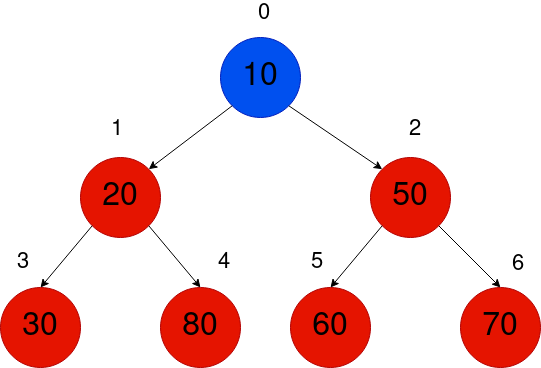


**Như đã nói ở trên, ở trong bài viết này, chúng mình chỉ làm việc với cây nhị phân và cụ thể hơn là cây nhị phân hoàn chỉnh (complete binary tree)**

## Binary Heap

### Định nghĩa

* Binary Heap là một cây nhị phân hoàn chỉnh được sử dụng để lưu trữ dữ liệu một cách hiệu quả nhằm lấy được phần tử lớn nhất hoặc nhỏ nhất dựa trên cấu trúc của nó.



* Binary Heap thường được biểu diễn dưới dạng của một mảng.

### Các loại Binary Heap

* Max Heap: Cho dãy số là một max-heap nếu thỏa mãn:
* Min Heap: Cho dãy số là một max-heap nếu thỏa mãn:

### Một số thao tác trên Binary Heap

Ở đây, ta xét Binary Heap này là một Min Heap, các theo tác trên Max Heap cũng tương tự như vậy. Binary Heap được xét ở đây đang lưu trong mảng và chỉ số được bắt đầu từ 0.

* heapify(): Tạo ra một min heap dựa trên một dãy số cho trước. Độ phức tạp của hàm này là

|  |
| --- |
| shift(a[], l, r){  i = l, j = 2 \* i + 1;  while (j <= r){  if (j < r)  if (a[j] > a[j + 1]) j++;  if (a[i] <= a[j]) break;  swap(a[i], a[j]);  i = j; j = 2 \* i + 1;  }  }  heapify(a[], heap\_size){  k = [heap\_size / 2] – 1;  while (k >= 0) shift(a, k--, heap\_size - 1);  } |

* getMin(): Trả về phần tử gốc của Min Heap. Độ phức tạp của hàm này là
* extractMin(): Xóa phần tử nhỏ nhất khỏi Min Heap và duy trì thuộc tính của Heap. Độ phức tạp của hàm này là Mã giả:

|  |
| --- |
| extractMin(a[], heap\_size){  swap(a[--heapsize], a[0]);  shift(a, 0, heap\_size - 1);  } |

* remove(i): Loại bỏ phần tử thử i ở trong mảng

|  |
| --- |
| remove(a[], heap\_size, i){  swap(a[--heapsize], a[i]);  shift(a, i, heap\_size - 1);  } |

* insert(p): thêm phần tử p vào Binary Heap

|  |
| --- |
| remove(a[], heap\_size, p){  a[heap\_size++] = p;  k = [(p – 1) / 2];  while (k >= 0){  shift(a, k, heap\_size - 1);  k = [(k – 1) / 2];  }  } |

## Ứng dụng của Binary Heap

### Heap Sort

Ý tưởng:

### Hàng đợi ưu tiên

# Hàng đợi ưu tiên

Ở trong mục này, ta xét đang làm việc với hàng đợi ưu tiên có phần tử ở đỉnh hàng đợi là lớn nhất.

## Hàng đợi ưu tiên được biểu diễn bởi mảng

Các theo tác này, ta đang xét trên một hàng đợi ưu tiên pq.

* size(pq): kiểm tra kích thước hiện tại trong hàng đợi ưu tiên. Độ phức tạp của thao tác này là

|  |
| --- |
| size(pq){  return pq.size();  } |

* empty(pq): kiểm tra rỗng ở trong hàng đợi ưu tiên.

|  |
| --- |
| bool empty(pq){  return (size(pq) == 0);  } |

* top(pq): truy cập phần tử trên đỉnh ở tron hàng đợi ưu tiên.

|  |
| --- |
| top(pq){  return pq[0];  } |

* delete(pq): xóa phần tử trên đỉnh ở trong hàng đợi ưu tiên.

|  |
| --- |
| delete(pq){  swap(a[0], a[size(pq) – 1]);  pq.resize(size(pq) - 1);  maxHeapify(a, 0, size(pq) -1);  } |

* insert(pq, x): thêm một phần tử x vào hàng đợi ưu tiên pq.

|  |
| --- |
| insert(pq, x){  pq.resize(size(pq) + 1);  a[size(pq) – 1] = x;  j = size(pq) - 1;  k = [(j – 1) / 2];  while (k >= 0){  if (pq[k] >= pq[j]) break;  swap(pq[j], pq[k]);  j = k;  k = [(j – 1) / 2];  }  } |

## Hàng đợi ưu tiên được biểu diễn bởi danh sách liên kết

Cấu trúc này sử dụng các node được liên kết bằng con trỏ. Mỗi node chứa giá trị, con trỏ đến cha, con trái và con phải.

**Thêm phần tử (push)**

1. Tạo node mới với giá trị x.
2. Nếu heap rỗng, đặt node này làm gốc (root).
3. Nếu không, tìm vị trí chèn tiếp theo trong cây.

* Nếu chưa có con trái, gán node mới vào con trái.
* Nếu chưa có con phải, gán node mới vào con phải.

1. Thiết lập quan hệ cha-con giữa node mới và node cha.
2. Duy trì tính chất heap bằng cách heapify up.

|  |
| --- |
| function push(x):  node = createNode(x)  if root == null:  root = node  else:  parent = findInsertionPoint()  if parent.left == null:  parent.left = node  else:  parent.right = node  node.parent = parent  heapifyUp(node) |

**Xóa phần tử lớn nhất (pop)**

1. Hoán đổi giá trị của gốc và node cuối cùng.

2. Xóa node cuối cùng khỏi cây.

3. Cập nhật lại node cuối cùng mới.

4. Duy trì tính chất heap bằng cách heapify down.

|  |
| --- |
| function pop():  if root == null:  throw Error("Heap is empty")  if size == 1:  delete root  root = null  else:  swap(root.value, last.value)  deleteLastNode()  updateLastNode()  heapifyDown(root) |

heapifyUp

Duy trì tính chất của heap sau khi thêm phần tử mới vào cây. Thực hiện hoán đổi giá trị giữa node hiện tại và node cha nếu vi phạm tính chất của heap (cha nhỏ hơn con).

|  |
| --- |
| function heapifyUp(node):  while node.parent != null and node.parent.value < node.value:  // Hoán đổi giá trị của node hiện tại với node cha  swap(node.value, node.parent.value)  // Di chuyển lên node cha  node = node.parent |

heapifyDown

Duy trì tính chất của heap sau khi thêm phần tử mới vào cây. Thực hiện hoán đổi giá trị giữa node hiện tại và node cha nếu vi phạm tính chất của heap (cha nhỏ hơn con).

|  |
| --- |
| function heapifyDown(node):  while node != null:  largest = node  // Kiểm tra con trái  if node.left != null and node.left.value > largest.value:  largest = node.left  // Kiểm tra con phải  if node.right != null and  node.right.value > largest.value:  largest = node.right  // Nếu node hiện tại đã là lớn nhất, thoát vòng lặp  if largest == node:  break  // Hoán đổi giá trị của node hiện tại với node lớn hơn  swap(node.value, largest.value)  // Di chuyển xuống node con  node = largest |

# Tài liệu tham khảo

Tài liệu tham khảo phải là sách hoặc là paper – bài báo khoa học chứ không thể là web nhé : v

* [Geeksforgeeks](https://www.geeksforgeeks.org/binary-heap/)
* [Wiki](https://en.wikipedia.org/wiki/Binary_heap)
* [Programiz](https://www.programiz.com/dsa/balanced-binary-tree)