**IT003.N21.CTTN – Cấu trúc dữ liệu và giải thuật**

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\\

**Họ tên:** Võ Trần Minh

**MSSV:** 22520894

\\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//

Link code: [DSA/tree at main · tranminhprvt01/DSA (github.com)](https://github.com/tranminhprvt01/DSA/tree/main/tree)

**A picture containing text, screenshot, plot, line

Description automatically generated**

**Biểu đồ thể hiện chiều cao của cây AVL và cây Red-Black so với log2(n) và 1.45\*log2(n) trên 10 bộ dữ liệu khác nhau**

**Nhận xét:**

* Dựa vào các số liệu ở biểu đồ trên, ta có thể thấy rằng chiều cao của cây AVL tương đối ổn định qua các bộ dữ liệu khác nhau. Còn cây đỏ đen thì lại không được ổn định bằng.
* Mặc dù cả hai cây AVL và cây đỏ đen đều là cây nhị phân tìm kiếm tự cân bằng (self-balancing BST). Nhưng thực tế, dựa vào các đặc tính (properties) để xây dựng cây AVL, ta có thể thấy rằng cây AVL sẽ tự cân bằng nghiêm ngặt hơn vì mỗi khi thêm 1 nút mới vào cây AVL, chỉ cần chiều cao của cây con trái của 1 nút chênh nhau quá 1 so với chiều con của cây con phải của nút đó là sẽ thực hiện cân bằng lại (re-balance).
* Tức là, ở cây AVL sẽ thực hiện nhiều lần rotate hơn so với cây đỏ đen. Dẫn đến nhược điểm là mỗi khi thêm 1 nút mới vào cây AVL sẽ lâu hơn so với cây đỏ đen.
* Tuy nhiên do cây AVL đã tự cân bằng nghiêm ngặt nên việc thực hiện tìm kiếm trên cây AVL sẽ nhanh hơn rất nhiều so với cây đỏ đen
* Từ đó, cây AVL sẽ thích hợp hơn để sử dụng trong các database, khi mà việc tìm kiếm một dữ liệu nào đó diễn ra rất thường xuyên. Còn cây AVL sẽ được sử dụng nhiều cho các dạng cấu trúc dữ liệu như map, multimap, … để tối ưu hóa tốc độ xử lí trong quá trình thêm bớt các dữ liệu.