ARRAY, TREE, HASH TABLE

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Hướng dẫn: Trương Tấn Khoa, Phạm Trọng Nghĩa

MỤC LỤC

THÔNG TIN

Cá Nhân

MSSV	HỌ TÊN	EMAIL
19127517	Hồ Thiên Phước	htphuoc19@clc.fitus.edu.vn

BÀI LÀM

Bảng 1

	Array	AVL Tree	Hash Table
Load	1031.6	1546.09	2038.39
Save	50.989	0.005	..
Look up	0.118	0.13	0.073
Insert	380.821	0.149	0.068
Remove	14.01	0.089	0.117
Edit	0.111	0.082	0.088

Phân tích dữ liệu:

Array:
Thời gian tìm kiếm của mảng nhanh
Tốc độ load chậm
Thêm/ xoá phần tử trong mảng tốn nhiều thời gian vì phải dịch chuyển các phần tử khác

AVL:
Thời gian load gấp rưỡi lần so với mảng
Tốc độ load chậm
Thời gian tìm kiếm, chèn, xoá tương đối ngắn

Hash:
Thời gian tính toán nhanh
Tốc độ chèn xoá tương đương Array, AVL, tuy nhiên việc chèn xoá lại đơn giản hơn mảng mà không làm dịch chuyển các phần tử khác
Tốc độ nhanh nhất.

- Descript in detail how you make each implementation.

Đối với Array: load file .txt về thành kiểu dữ liệu string* Với mỗi phần tử của string* là các dòng khác rỗng
Cắt chuỗi ra gán vào các thuộc tính của struct

Đối với AVL: Cây phải được cân bằng lại sau khi chèn và xóa
Trong cây AVL, độ cao của hai cây con của bất kỳ nút nào khác nhau nhiều nhất 1 đơn vị. Cây AVL có thể bị mất cân bằng sau khi chèn và xóa từ vựng. Xoay là cơ chế cơ bản mà cân bằng lại cây không cân bằng. Xoay là một sự điều chỉnh cho cây, xung quanh một mục, duy trì thứ tự yêu cầu của các từ vựng.
Đối với HashTable: xem nó như là 1 mảng danh sách

Đối với Hash table: 1 mảng các danh sách liên kết (separate chaining), key băm được tính bằng ‘ép kiểu int từng kí tự của 1 Word và tính tổng của chúng, sau đó mod size của hash table’

- Comparing time complexities with implementation of other data structure.

Đối với Array: Vì không có sự khác biệt về thời gian giữa việc tìm kiếm phần tử thứ hai hoặc cuối cùng trong mảng, mảng có thời gian tìm kiếm liên tục O (1)- tương đối nhanh. Về cơ bản, sự phức tạp thời gian chạy tốt nhất có thể là O (1). Thêm/xoá phần tử trong mảng có độ phức tạp là O(n), trừ việc thêm xoá ở cuối O(1)

Đối với AVL: search = delete = add = O(logn)

Đối với HashTable: hàm băm mất O(1) thời gian tính toán, nếu hàm băm phân phối các key đồng đều quanh bảng sao cho khoảng cách giữa các key nhỏ thì toàn bộ quá trình kiểm tra diễn ra là O (1) (trung bình) up to O(n). Chèn/ xoá mất O(n) thời gian

- Analyze strong and weak points of each data structure.

Đối với Array:
+Ưu: thời gian tìm kiếm nhanh vì các phần tử lưu trữ mảng trong các vị trí bộ nhớ liên tục
+ Khuyết: hao tốn bộ nhớ, thêm/ xoá phần tử trong mảng tốn nhiều thời gian vì khi thêm và xoá, các phần tử từ vị trí sau đều bị dịch chuyển, trừ việc thêm/ xoá ở phần tử cuối

Đối với AVL:
+Ưu: không phải lo lắng về việc liên tục phân bổ và giải quyết bộ nhớ khi thêm/ xoá phần tử, khả năng duyệt cây có thứ tự
+Khuyết: hơi tốn kém chi phí, độ rộng của mạng lưới dữ liệu với bộ nhớ mà mình sử dụng

Đối với HashTable:
+Ưu: hữu ích về mặt dữ liệu truy cập, sử dụng bộ nhớ tốt hơn, phân bổ bộ nhớ hiệu quả hơn, chèn/ xoá nhanh hơn, dễ dàng hơn, ngăn chặn lãng phí bộ nhớ

+Khuyết: tìm kiếm chậm hơn mảng vì vị trí bộ nhớ không liên tục, yêu cầu bổ sung bộ nhớ cho mỗi phần tử khi chèn

- Your experience.

Thành thạo hơn với cách đọc file bằng cách cắt chuỗi thay vì trước đó dùng vector, stringstream, pushback (quen với hàm substr, compare (2 string), find, empty, vòng lặp while)
Ép kiểu string->ASCII hay dùng hàm toupper phải dùng vòng lặp với hàm length
Quen với cách tạo .txt, cách Debug từng dòng với breakpoint bằng XCode Quen với syntax cấp phát bộ nhớ new-delete
Chia nhỏ vấn đề lớn thành nhiều vấn đề nhỏ để dễ dàng xử lí: chia nhỏ cấu trúc file, chia hàm lớn thành nhiều hàm con
Cách phân bố đều thời gian để đạt 1 mục tiêu: Your future is created by what you do today not tomorrow

THAM KHẢO

[1] geeksforgeeks
[2] slide lí thuyết thầy Lê Ngọc Thành