Họ tên: Lê Trần Hữu Đắc

MSSV: 1712026

Lớp: 17CNTN

Bài tập thực hành

Cấu trúc dữ liệu và giải thuật

**BÁO CÁO XÂY DỰNG ỨNG DỤNG TỪ ĐIỂN**

# Sử dụng cấu trúc dữ liệu cây AVL.

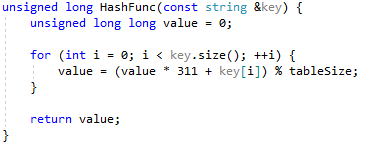
* Em sử dụng *cây AVL* để lưu trữ và xử lý dữ liệu. Với mỗi nút của cây, ta có trường *key* và trường *value* kiểu string để lưu từ và nghĩa tương ứng của từ. Khi so sánh các nút thì ta so sánh trường key của các nút đó.
* Tạo một lớp đặt tên là *Dictionary* để chứa cây AVL và cung cấp một số hàm cần thiết như *loadData*, *search*, *add*, *edit*, *remove*, *saveData*.
* Với việc load nội dung từ điển được lưu ở một file cho trước, em đọc từng dòng file dữ liệu và tách từ và nghĩa của từ bởi 2 khoảng trắng liên tiếp đầu tiên của dòng đó. Nếu không tách được thì chuyển sang dòng tiếp theo, ngược lại thêm từ này vào cây.
* Để tra nghĩa của một từ, ta thực hiện thao tác tìm nút có key trùng với từ đó trên cây và trả về value của nút đó chính là nghĩa của từ.
* Với yêu cầu thêm từ mới, chương trình yêu cầu người dùng nhập key và value sau đó thêm vào cây tương tự như lúc load nội dung.
* Thao tác xóa, ta thực hiện xóa nút có key tương ứng trong cây AVL.
* Với thao tác sửa nghĩa của một từ, trước tiên em xóa từ đó ra khỏi cây và thêm lại từ đó với nghĩa mới sau khi sửa.
* Để lưu từ điển, ta có thể dùng các phương pháp duyệt cây và lưu lại dữ liệu vào file.
* Để dễ dàng sử dụng, em có cài đặt menu với các chức năng cơ bản trên.
* Vì cài đặt bằng cây AVL nên phương pháp này có các tính chất của cây AVL như:
  + Độ cao trung bình của cây là *logN*, trong quá trình thực hiện các thao tác, cây luôn được cân bằng nên không có trường hợp suy biến về *N*.
  + Việc thêm, xóa hay tìm kiếm đều chỉ tốn chi phí thời gian là *O(logN \* L)* cho quá trình di chuyển từ gốc đến nút cần thao tác và so sánh key trên đường đi với *L* là độ dài key.
  + Để load từ điển ta cần thêm lần lượt các từ vào cây mà độ phức tạp cho mỗi lần thêm là *O(logN \* L)* nên độ phức tạp cho việc load từ điển là *O(NlogN\*L)*.
  + Vì chỉ cần duyệt cây để lưu dữ liệu nên thời gian việc lưu từ điển cũng tương đương với thời gian duyệt cây thêm với thời gian ghi dữ liệu là *O(N \* L)*.
  + Lượng bộ nhớ cần để lưu từ điển cũng là *O(N)* ứng với số lượng nút cần để lưu các từ trong từ điển, mỗi nút cần lưu hai chuỗi thể hiện từ và nghĩa của từ, thêm vào đó là 2 con trỏ đến 2 nút con và một biến nguyên lưu độ cao cây con tại nút đó.

# Sử dụng cấu trúc dữ liệu cây đỏ - đen.

* Cấu trúc dữ liệu này có cùng chức năng với cấu trúc dữ liệu cây AVL ở trên nên cách tổ chức dữ liệu tương tự nhau.
* Điểm khác biệt giữa cây đỏ - đen với cây AVL chính là thuật toán cài đặt cây, do đó dẫn đến hiệu năng của 2 cấu trúc dữ liệu này có sự khác biệt:
  + Cây AVL duy trì tính cân bằng ổn định hơn cây đỏ - đen. Chiều cao cây AVL tối đa chỉ khoảng *1.44log(N + 2),* trong khi đó ở cây đỏ - đen chiều cao có thể lên đến *2log(N + 1)* với *N* là số nút của cây.
  + Với thao tác thêm, cả 2 cấu trúc dữ liệu đều phải tìm nút thích hợp trước khi thêm vào tốn chi phí *O(logN \* L)* nhưng chiều cao của cây AVL thấp hơn cây đỏ - đen nên việc tìm trên cây AVL nhanh hơn. Ngoài ra tiêu chuẩn cân bằng của cây đỏ - đen thường đòi hỏi ít phép xoay hơn cây AVL nên chi phí để cân bằng cho cây đỏ - đen ít hơn cây AVL.
  + Với thao tác tìm kiếm, như đã nói ở trên, cây AVL sẽ tìm kiếm hiệu quả hơn cây đỏ - đen.
  + Với thao tác xóa, cả 2 cấu trúc dữ liệu đều phải tìm nút cần xóa, rồi tìm nút thay thế với chi phí tìm kiếm nêu trên và cuối cùng là xóa nút thay thế này. Để xóa một nút thì cây AVL có thể phải thực hiện *O(logN)* lần xoay cây để duy trì tính cân bằng, trong khi đó cây đỏ - đen chỉ cần *O(1)* lần xoay.
  + Với lượng dữ liệu được cung cấp thì chênh lệch thời gian giữa 2 cấu trúc dữ liệu cho các thao tác tìm kiếm, thêm, xóa một từ là không lớn và gần như bằng 0.
  + Thời gian load từ điển khi sử dụng cây đỏ - đen nhanh gấp đôi khi sử dụng cây AVL, còn thời gian lưu từ điển là tương đương nhau.
  + Số lương bộ nhớ sử dụng trong cả 2 cấu trúc dữ liệu gần như nhau, thay vì lưu độ cao như cây AVL thì mỗi nút của cây đỏ - đen sẽ lưu màu của nút đó và con trỏ trỏ đến nút cha nó.

# Sử dụng cấu trúc dữ liệu Bảng băm.

* Tương tự trên, ta cũng có một lớp Dictionry chứa bảng băm và xử lý các công việc cần thiết cho từ điển.
* Bảng băm bao gồm một mảng *table*, mỗi phần tử *table[i]* lưu một danh sách liên kết gồm các nút có *HashValue* = i. Ở đây em chọn mảng có kích thước *10­5 + 7* và hàm băm HashFuction được thực hiện như sau:



* Mỗi nút trong bảng băm lưu 2 chuỗi key, value tương ứng với từ và nghĩa tương ứng và một con trỏ trỏ đến nút kế tiếp trong danh sách liên kết các nút có cùng *HashValue* với nút đó.

# Tự đánh giá.

* Em đã hoàn thành các công việc nêu trên, chương trình có thể chạy tốt.
* Nếu file dữ liệu không tuân thủ những đặc điểm kể trên thì chương trình không nhận diện được chính xác từ và nghĩa của từ. Ví dụ như nếu một từ hay nghĩa của từ được ghi ở nhiều dòng thì chương trình chỉ đọc đến hết dòng và chuyển sang từ khác.
* Với những từ gồm nhiều từ được viết cách nhau bởi một khoảng trắng như “Affiliation order” thì chương trình vẫn rút trích được từ và nghĩa của từ.
* Để load dữ liệu từ file OxfordEnglishDictionary.txt mà thầy cung cấp, chương trình cần khoảng vài giây để load xong do phải đọc một lượng lớn dữ liệu và thêm vào cây, thời gian để load nếu sử dụng cây đỏ - đen nhanh gấp đôi so với cây AVL.
* Nếu tra cứu nghĩa của một từ chưa có trong từ điển thì chương trình sẽ trả về chuỗi rỗng, hay khi xóa từ không tồn tại trong từ điển thì không có gì thay đổi cả.
* Nếu người dùng sửa nghĩa một từ chưa có trong từ điển thì chương trình sẽ thêm từ đó và nghĩa tương ứng.