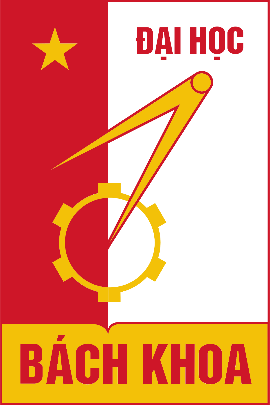
**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**



**BÁO CÁO MÔN HỌC PROJECT I – IT3150**

**THỰC HIỆN KIỂM TRA VÀ PHÂN TÍCH KHẢ NĂNG CODE CỦA SINH VIÊN**

Sinh viên thực hiện : Nguyễn Hoàng Hiệp

MSSV : 20205204

Mã Lớp : 721014

GVHD : KHMT - Phạm Đăng Hải

**MỤC LỤC**

1. **Giới thiệu chung**
2. **Bài tập số 1 : Bài toán sắp xếp**
   1. Giới thiệu chung
   2. Trình bày các dạng bài toán sắp xếp
   3. So sánh các phương thức so sánh
   4. Kết luận
3. **Bài tập số 2 : Bài toán quay lui**
   1. Giới thiệu chung
   2. Bài toán 8 hậu
   3. Bài toán người đi du lịch
   4. Kết luận
4. **Bài tập số 3 : Lập chỉ mục (index) cho file**
   1. Giới thiệu chung
   2. Thực hiện bài toán bằng các dạng cấu trúc dữ liệu
      1. Xây dựng chương trình
      2. Danh sách liên kết
      3. Bảng băm
      4. Cây nhị phân
5. **Giới thiệu chung**

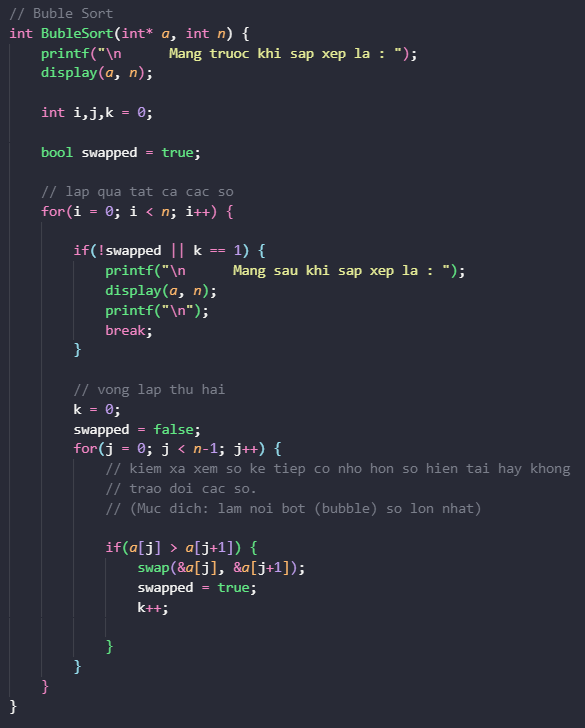
* **Thời lượng :** IT3120 2(0-0-4-8)
* **Mục tiêu :**
  + Hệ thống hóa một số kiến thức cơ sở cốt lõi
  + Phát triển các kĩ năng lập trình chuyên nghiệp
    - *Quy ước lập trình, gỡ rối, kiểm thử …*
  + Phát triển các kĩ năng mềm khác
    - *Làm việc nhóm, quản lý thời gian, báo cáo, thuyết trình …*
* **Nội dung chính :**
  + Cài đặt một số thuật toán từ đơn giản đến phức tạp;
    - *Sử dụng các cấu trúc dữ liệu khác nhau*
    - *Sử dụng các ngôn ngữ lập trình như C, C++, Java ...*
  + Nghiên cứu một chủ đề cụ thể
  + Viết báo cáo và thuyết trình kết quả
* **Thời gian :** 7-8 tuần
* **Nội dung :**
  + Bài 1 : Bài toán sắp xếp
    - *Thời gian dự kiến 1-2 tuần*
  + Bài 2 : Bài toán quay lui
    - *Thời gian dự kiến 1-2 tuần*
  + Bài 3 : Bài toán lập chỉ mục cho file văn bản
    - *Thời gian dự kiến 3-4 tuần*

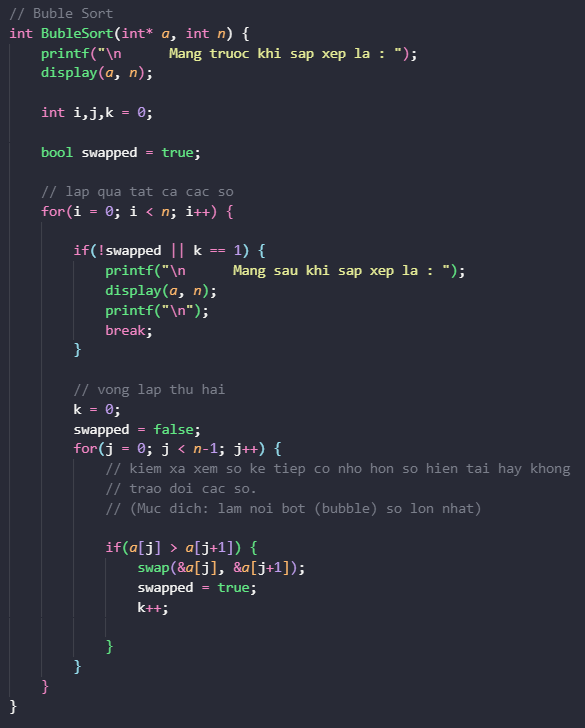
1. **Bài tập 1 : Bài toán sắp xếp**
   1. **Giới thiệu chung**

* **Đề bài :** Nhập dãy số, sắp xếp theo thứ tự tăng dần
  + **Input :** Bàn phím, File hoặc sinh ngẫu nhiên
  + **Output :** Màn hình hoặc File
  + **Algorithm :** Bubble Sort, Heap Sort, Quick Sort, Chèn, Trộn, …
* **Yêu cầu kết quả :**
  + **Chương trình :** Kiểm tra chương trình
  + **Báo cáo :** Đánh giá độ phức tạp
    - *So sánh thời gian thực hiện Lý thuyết và Thực tế.*
* **Yêu cầu nâng cao :**
  + Cài đặt một thuật toán sắp xếp song song (đa luồng)
  + Đánh giá thời gian thực hiện trên các máy Multicore
  1. **Trình bày các dạng bài toán sắp xếp**
     1. **Bubble Sort**

Sắp xếp nổi bọt (Bubble Sort) là một thuật toán sắp xếp đơn giản, với thao tác cơ bản là so sánh hai phần tử kề nhau, nếu chúng chưa đứng đúng thứ tự thì đổi chỗ. Có thể tiến hành từ trên xuống hoặc từ dưới lên. Sắp xếp nổi bọt còn có tên là sắp xếp bằng so sánh trực tiếp.

**Code:**

****

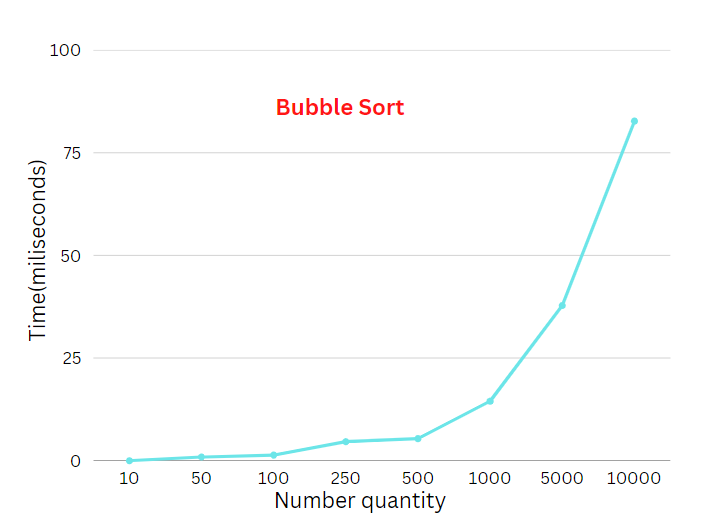
****

**Đánh giá độ phức tạp :**

Với mỗi i = 0,1,2,…, n-1 trong mỗi vòng lặp ta có n phép so sánh. Ở đây có 2 vòng lặp, vậy trong trường hợp tệ nhất ta có Độ phức tạp là **O(n2)**

**So sánh thời gian thực hiện trên lý thuyết và trên thực tế :**

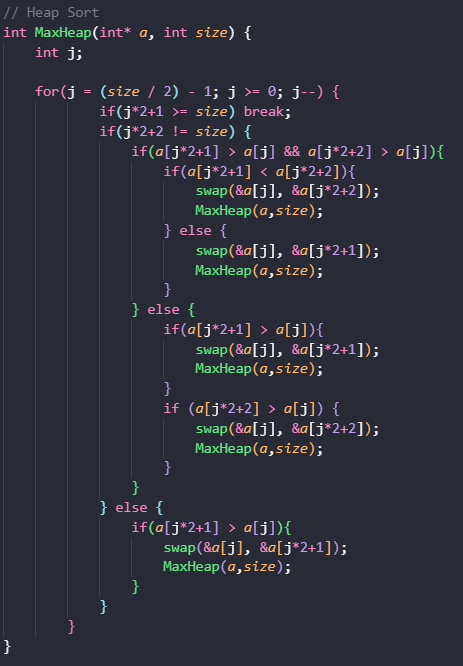
* Thực hiện chạy giải thuật ở các bộ test từ 10 đến 10000. Ta có đồ thị thống kê sau :

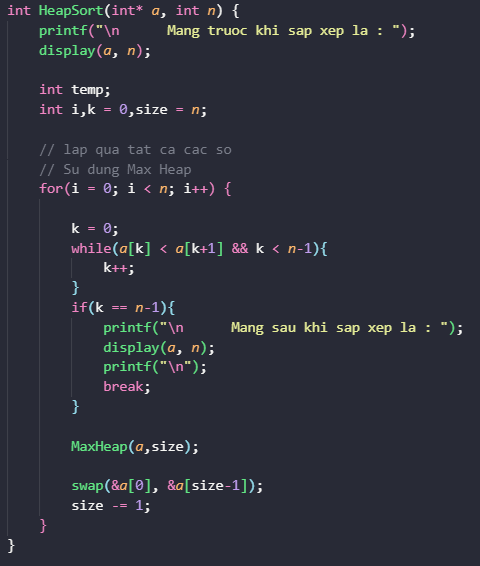
****

* + 1. **Heap Sort**

**Sắp xếp vun đống – heap sort** là một thuật toán sắp xếp nhanh sử dụng kĩ thuật phân loại dựa trên cấu trúc cây nhị phân đặc biệt gọi là đống nhị phân (**binary heap**). Thuật toán dựa vào sự đặc biệt của cây nhị phân để lựa chọn ra phần tử lớn nhất rồi lần lượt chèn phần tử này vào vùng sắp xếp.

**Code:**

****

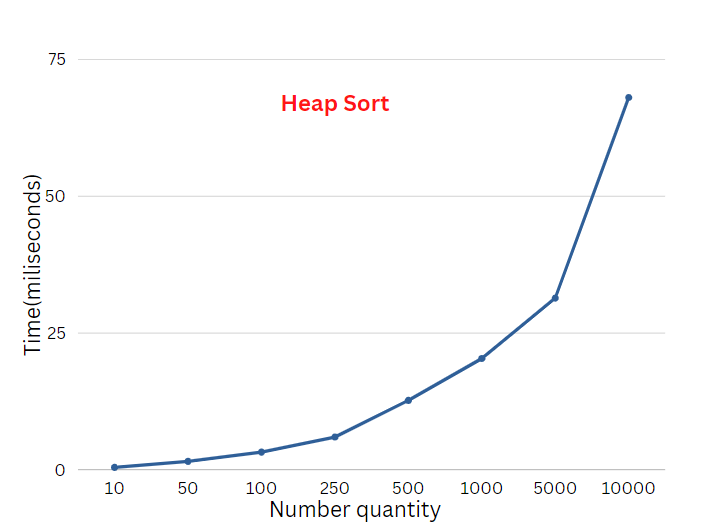
****

**Đánh giá độ phức tạp :**

* Để vun đống, ta sẽ phải vun từ dưới lên, tức là vun từ đỉnh cha cuối cùng của heap. Một mảng **n** phần tử sẽ có **n/2** node cha.
* Đổi chỗ đỉnh gốc đống với phần tử cuối cùng của mảng. Như vậy phần tử lớn nhất sẽ nằm ở cuối mảng
* Giảm số lượng phần tử (loại bỏ phần tử cuối cùng đúng vị trí)
* Lặp lại việc vun đống và lấy phần tử cho tới khi còn 1 phần tử thì dừng
* Một mảng **n** phần tử sẽ phải thực hiện vun đống **n** lần
* Hàm để build heapify có độ phức tạp là **O(logn)**
* Độ phức tạp thời gian của createAndBuildHeap là **O(n)**
* độ phức tạp thời gian tổng thể của Heap Sort là **O(nLogn)**.

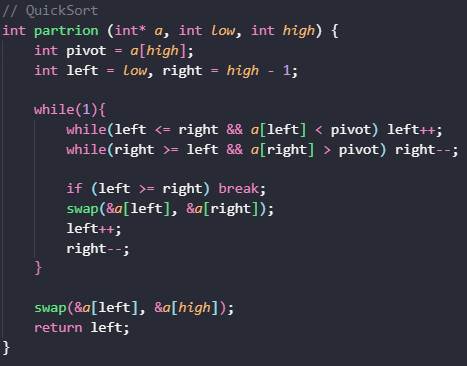
**So sánh thời gian thực hiện trên lý thuyết và trên thực tế :**

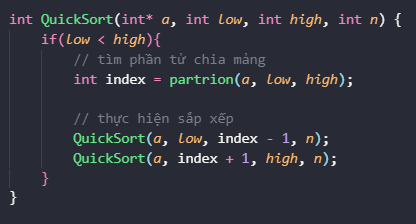
* Thực hiện chạy giải thuật ở các bộ test từ 10 đến 10000. Ta có đồ thị thống kê sau :

**\**

* + 1. **Quick Sort**

**Code:**

****

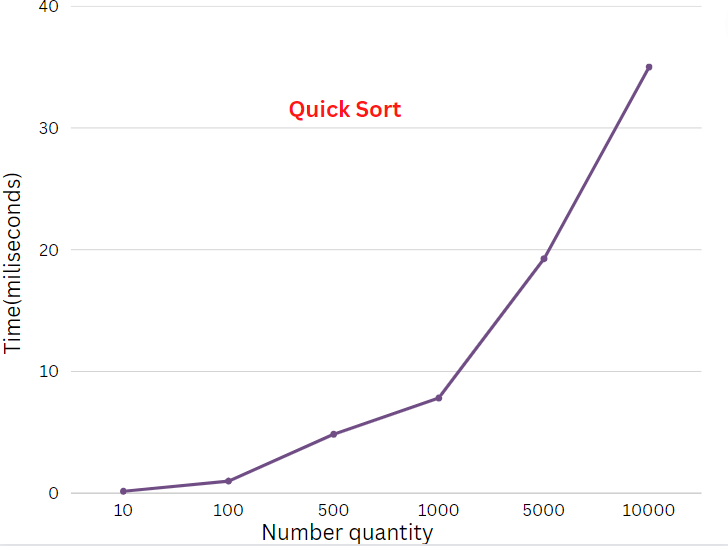
****

**Đánh giá độ phức tạp :**

* Phức tạp thời gian: Trung bình **O(n log n)**
* Xấu nhất: **O(n2)**

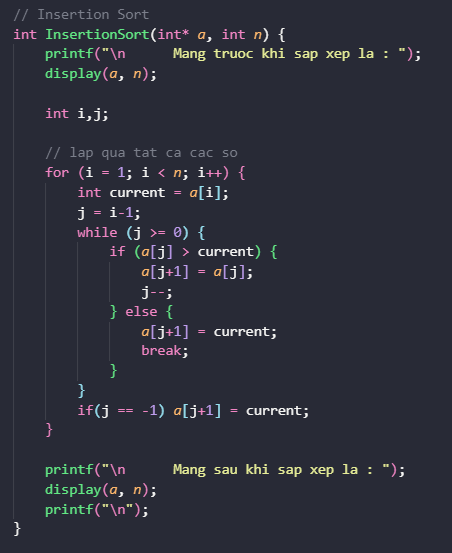
**So sánh thời gian thực hiện trên lý thuyết và trên thực tế :**

* Thực hiện chạy giải thuật ở các bộ test từ 10 đến 10000. Ta có đồ thị thống kê sau :

****

* + 1. **Insertion Sort**

**Code:**

****

**Đánh giá độ phức tạp :**

* + Hiệu suất trung bình: О(n2)
  + Hiệu suất trường hợp tốt nhất: O(n)

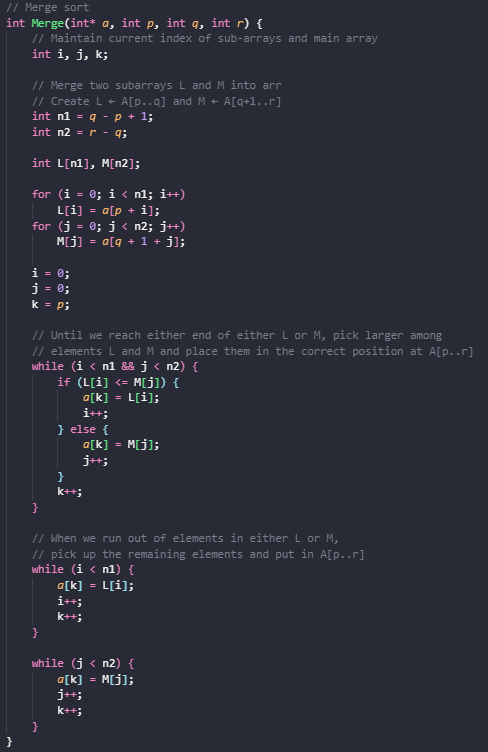
**So sánh thời gian thực hiện trên lý thuyết và trên thực tế :**

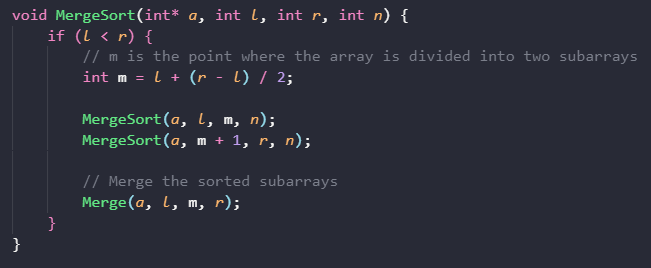
* Thực hiện chạy giải thuật ở các bộ test từ 10 đến 10000. Ta có đồ thị thống kê sau :

****

* + 1. **Merge Sort**

**Code :**

****

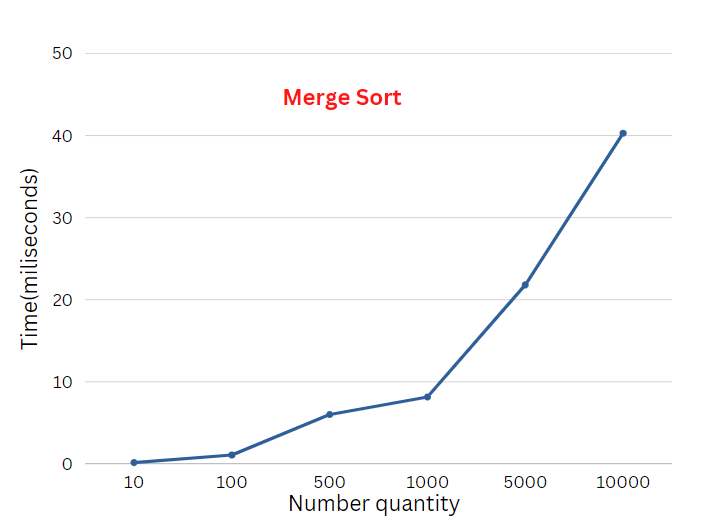
****

**Đánh giá độ phức tạp :**

* + Giả sử có hai danh sách đã được sắp xếp **a[1 ... m]** và **b[1 ... n].**
  + Ta có thể trộn chúng lại thành một danh sách mới c[1 ... m+n] được sắp xếp theo cách sau:
  + So sánh hai phần tử đứng đầu của hai danh sách, lấy phần tử nhỏ hơn cho vào danh sách mới. Tiếp tục như vậy cho tới khi một trong hai danh sách là rỗng.
  + Khi một trong hai danh sách là rỗng ta lấy phần còn lại của danh sách kia cho vào cuối danh sách mới.
  + Độ phức tạp thuật toán: **O(nlog(n))**

**So sánh thời gian thực hiện trên lý thuyết và trên thực tế :**

* Thực hiện chạy giải thuật ở các bộ test từ 10 đến 10000. Ta có đồ thị thống kê sau :

****

* 1. **So sánh các phương thức so sánh**

**1. Bubble sort - Thuật toán sắp xếp nổi bọt**

* Hiệu suất của bubble sort O(n):
  + - * Tốt nhất: n
      * Tệ nhất: n^2
* Ưu điểm: Code ngắn gọn nhất
* Nhược điểm: Hiệu suất thấp nhất

**2. Heap sort - Thuật toán sắp xếp vun đống**

* Hiệu suất của heap sort O(n):
  + - * Tốt nhất: n log n
      * Tệ nhất: n log n
* Ưu điểm: Hiệu suất của thuật toán cao
* Nhược điểm: Code phức tạp

**3. Quick sort - Thuật toán sắp xếp nhanh**

* Hiệu suất của quick sort O(n):
  + - * Tốt nhất: n log n
      * Tệ nhất: n^2
* Ưu điểm: Tuỳ cách chọn pivot mà tốc độ của thuật toán nhanh hay chậm
* Nhược điểm: Code khá phức tạp

**4. Insert sort - Thuật toán sắp xếp chèn**

* Hiệu suất của insert sort O(n):
  + - * Tốt nhất: n
      * Tệ nhất: n^2
* Ưu điểm: Chạy nhanh khi mảng nhỏ hay được sắp xếp một phần
* Nhược điểm: Hiệu suất thấp

**5. Merge sort - Thuật toán sắp xếp trộn**

* Hiệu suất của bubble sort O(n):
  + - * Tốt nhất: n log n
      * Tệ nhất: n log n
* Ưu điểm: Hiệu suất của merge sort rất cao
* Nhược điểm: Code thuật toán này khá phức tạp

1. **Bài tập 2 : Bài toán quay lui**
   1. **Giới thiệu chung**
      * + **Bài 1 :** Chọn 1 trong bài toán sau và thực hiện
          - Bài toán mã đi tuần
          - Bài toán 8 hậu
          - **Yêu cầu kết quả :**

Kết quả đưa ra trực quan

Báo cáo

Báo cáo cách tổ chức dữ liệu (dạng bảng)

Thuật toán

* + - * **Bài 2 :** Bài toán người du lịch
        + Người đi du lịch qua tất cả thành phố,

Mỗi thành phố đi 1 lần + trở về thành phố ban đầu khi hết hành trình

Tìm đường đi với chi phí nhỏ nhất

* + - * + **Yêu cầu chương trình:**

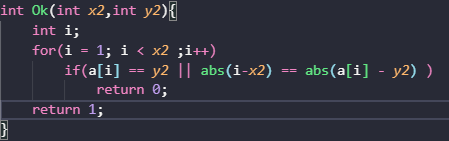
Input : Ma trận chi phí (Bàn phím, ngẫu nhiên)

Output : Đường đi và chi phí

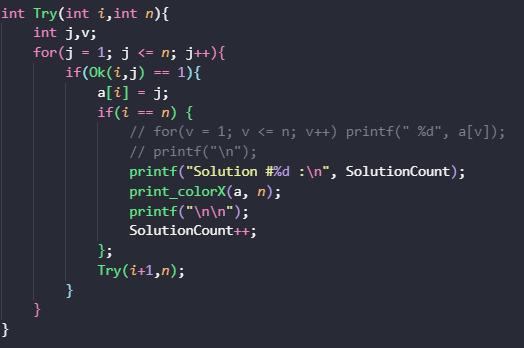
Algorithm : Vét cạn và Quay lui có nhánh cận

* + - * **Yêu cầu báo cáo :**
        + Cấu trúc dữ liệu và giải thuật
        + Đánh giá độ phức tạp
  1. **Bài toán 8 hậu**
     + - **Code :**

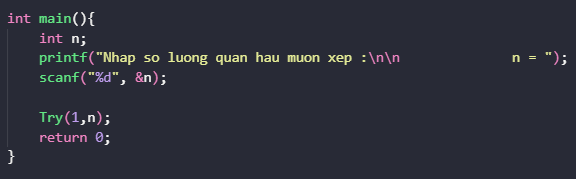
**Hàm check điều kiện**

****

**Hàm đệ quy**

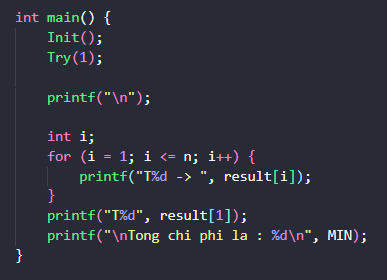
****

**Hàm main**

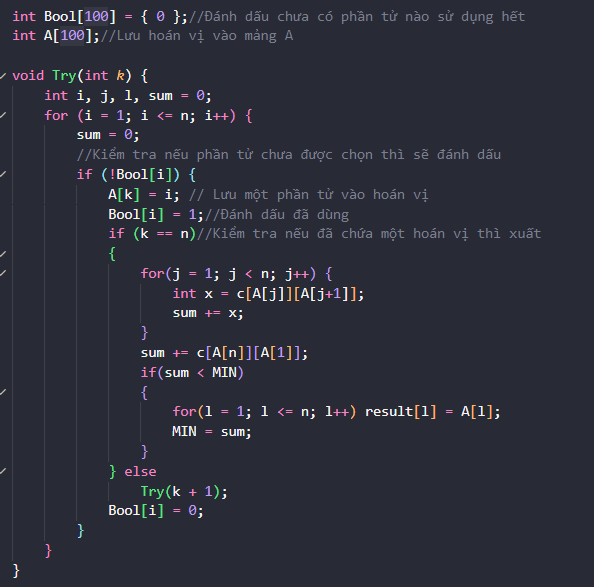
****

* + - * **Cấu trúc dữ liệu và giải thuật:**
        + Bài toán sử dụng giải thuật **Đệ quy Quay lui** tìm ra vị trí quan hậu trên từng hàng của bàn cờ, nếu thỏa mãn (tức hàm Ok(i,j) == 1 ) thì tiếp tục soát vị trí trên hàng tiếp theo. Thực hiện đến khi tìm được các đáp án phù hợp.
      * **Đánh giá độ phức tạp :**
  1. **Bài toán người đi du lịch**
     1. **Vét cạn**
        + **Code:**

**Hàm main()**



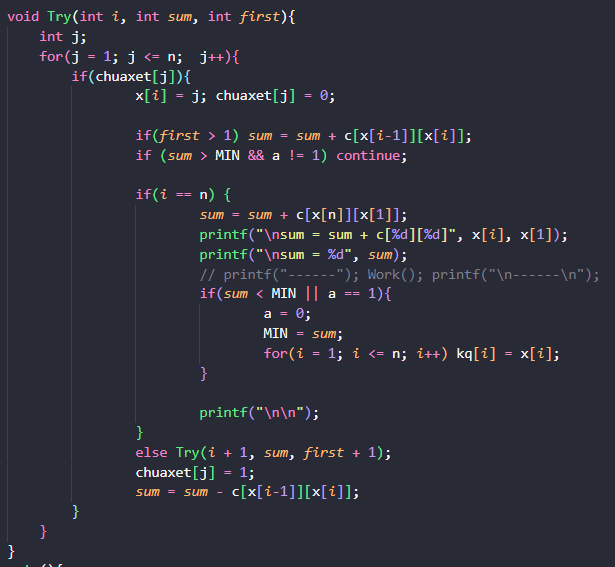
**Hàm hoán vị và tính chi phí theo từng hoán vị**

****

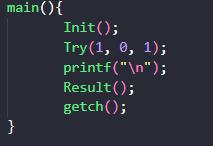
**Mô tả :**

* + - * Tạo một mảng Bool để đánh dấu xem các phần tử đã được sử dụng chưa. (0 : Chưa; 1 : đã sử dụng) và một mảng A để lưu hoán vị nếu nó đạt Chi phí thấp nhất (MIN).
      * Thực hiện đệ quy từ vị trí số 1 (hiện tại đang là 0,1,2,…,n) thực hiện đệ quy qua vòng for. Nếu hàm Bool[i] == 0 tức phần tử chưa được sử dụng. ta sẽ tiếp tục thực hiện đệ quy cho đến n. Nếu đã sử dụng ta chuyển qua đệ quy với vị trí tiếp theo. Thực hiện cho đến hết các hoán vị
    1. **Quay lui có nhánh cận**
       - **Code :**

**Hàm đệ quy**

****

**Hàm main()**

****

**Mô tả:**

* + - * Chuaxet[] là mảng input.
      * Hàm Try() thực hiện đệ quy trên từng số thứ tự, cho đến n thì check xem hành trình vừa có đạt MIN không, nếu có thực hiện lưu hành trình và MIN. Trong quá trình tính nếu Sum > MIN sẽ ngay lập tức trở về nhánh trước đó.
  1. **Kết luận**

**Thuật toán quay lui**

**- Ưu điểm:** Việc quay lui là thử tất cả các tổ hợp để tìm được một lời giải. Thế mạnh của phương pháp này là nhiều cài đặt tránh được việc phải thử nhiều trường hợp chưa hoàn chỉnh, nhờ đó giảm thời gian chạy.

**- Nhược điểm:** Trong trường hợp xấu nhất độ phức tạp của quay lui vẫn là cấp số mũ.

1. **Bài tập 3 : Lập chỉ mục (index) cho file**
   1. **Giới thiệu chung**

**Đề bài:**

* Đọc một tệp văn bản, hãy lập một bảng chỉ mục (index table) cho tệp văn bản đó.
* Bảng chỉ dẫn liệt kê tất cả các từ có ý nghĩa xuất hiện trong văn bản theo quy cách
  + Mỗi từ được liệt kê một lần cùng với số lần xuất hiện trong văn bản và dòng xuất hiện từ đó.
  + Các từ phải được sắp xếp theo thứ tự từ điển
* Không đưa vào bảng các từ
  + Không có ý nghĩa để tra cứu (for, the, an, …)

**Input :**

* File **VanBan.txt**
  + File văn bản tiếng Anh cần lập chỉ mục
    - Từ là dãy chữ cái (‘a’ .. ‘z’), phân biệt bởi ký tự phân cách
    - Không phân biệt chữ hoa, chữ thường
* File **Stopword.txt**
  + File chứa danh sách các không có ý nghĩa tra cứu
  + Mỗi từ khóa trên một dòng

**Output :**

* Ra màn hình hoặc ra file văn bản **Index.txt**
* Mỗi từ trên một dòng
  + Từ, Số lần xuất hiện, Danh sách dòng xuất hiện.

**Lưu ý :**

* Không có thông tin về
  + Số lượng từ trong file **Stopw.txt**
  + Số lượng từ trong file **Vanban.txt**
  + Số lượng dòng trong file **Vanban.txt**
* Không dùng kiểu mảng để lưu thông tin
  + Dùng danh sách liên kết
    - Danh sách các từ của Stopw
    - Danh sách các dòng
  + Dùng các CTDL để lưu danh sách các chỉ mục
    - DSLK, Cây nhị phân, Bảng băm
  1. **Thực hiện bài toán bằng các dạng cấu trúc dữ liệu**
     1. **Xây dựng chương trình**

**Thiết kế khung cho chương trình**

* Xây dựng danh sách StopW
* Đọc một từ, Kiểm tra từ có nghĩa
* Lưu từ vào danh sách được sắp xếp
* Xác định các thông tin cho từ có nghĩa
  + Số lần xuất hiện, chỉ số dòng

**Kiểm tra từ có ý nghĩa**

* Đọc từ trong file
  + Đọc từng ký tự cho tới khi gặp chữ cái (int isalpha())
  + Đọc tiếp các ký tự cho tới khi không phải chữ cái
    - Chuyển về chữ thường
  + Lưu chữ cái đọc vào mảng (*độ dài từ <12*), thêm ‘\0’
* Kiểm tra từ có nghĩa
  + Kiểm tra từ có trong danh sách StopW
    - Tồn tại trong danh sách 🡪 Bỏ qua, quay lại đọc từ tiếp
    - Không tồn tại trong danh sách 🡪 Thêm vào danh sách chỉ mục
* Lặp tới khi toàn bộ file Vanban.txt đã đọc xong

**Chèn từ vào danh sách**

* Nếu từ chưa có trong danh sách
  + - Chèn vào danh sách theo đúng vị trí
    - Số lần xuất hiện là 1
    - Thêm số dòng xuất hiện liên quan
      * Xác định chỉ số dòng xuất hiện (ký tự ‘\n’)
* Nếu từ đã có trong danh sách
  + Tăng số lần xuất hiện
  + Thêm chỉ số dòng (*nếu chưa có*)
    - Chỉ số dòng xuất hiện viết theo thứ tự
    - So sánh dòng của từ với phần tử cuối

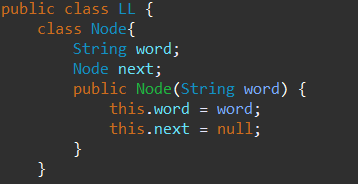
**Tạo nền cho chương trình**

* Nhận thấy 3 CTDL đều có thể dựng trên nền Danh sách liên kết (**LinkedList**) -> Dựng nền gồm 2 file :
  + **LL.java**
    - File **LL.java** có vai trò check một từ được thêm vào và build **Node** trong danh sách cho từ đó
  + **LL3.java**
    - File **LL3.java** có vai trò nhận những từ từ **Node** đã được build ở **LL.java** và **LL2.java** để check trong file **VanBan.txt** (hay file **testcase.txt** (file cá nhân) và in ra màn hình các dòng mà từ đó xuất hiện

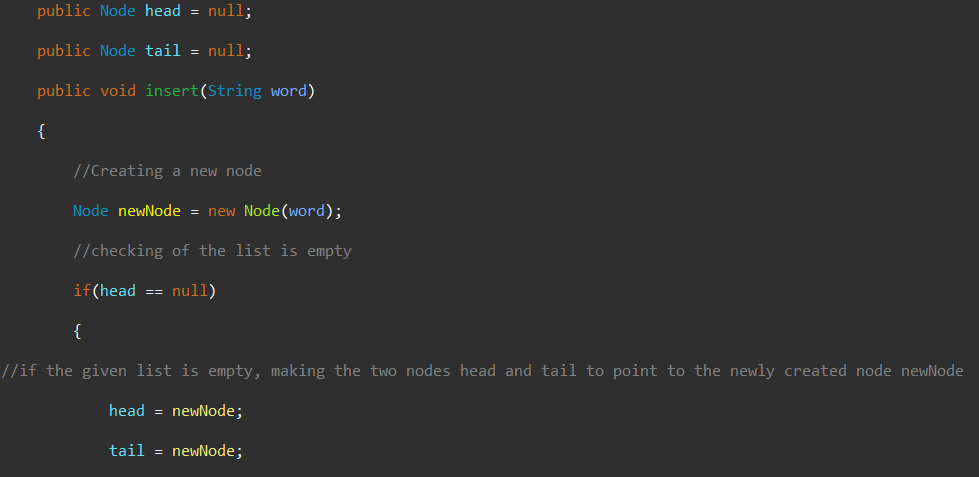
**Code:**

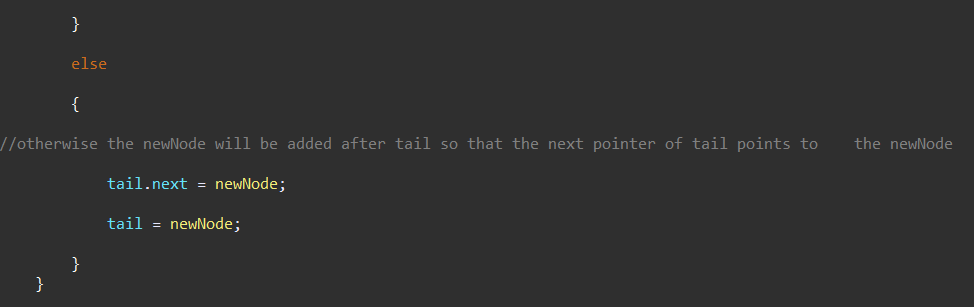
**File LL.java**

**Cấu trúc Node**

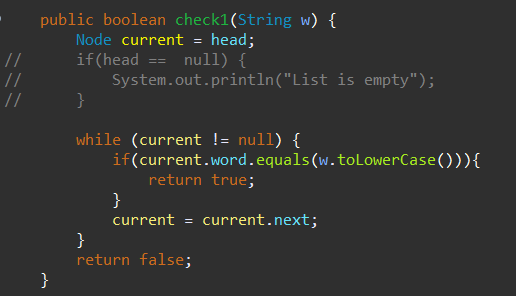
****

**Hàm Insert() thực hiện tạo Node mới**

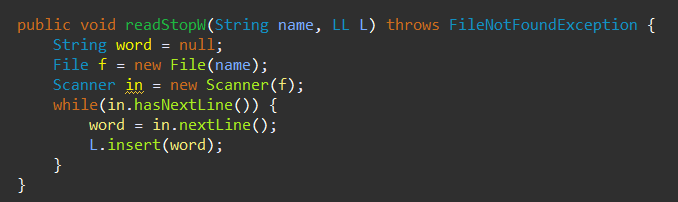
****

****

**Hàm check() để xét xem từ đó có trong danh sách chưa**

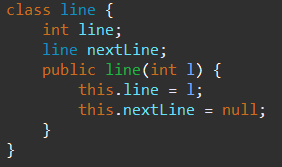
****

**Hàm readStopW() để thực hiện check word được đưa vào từ file bài làm**

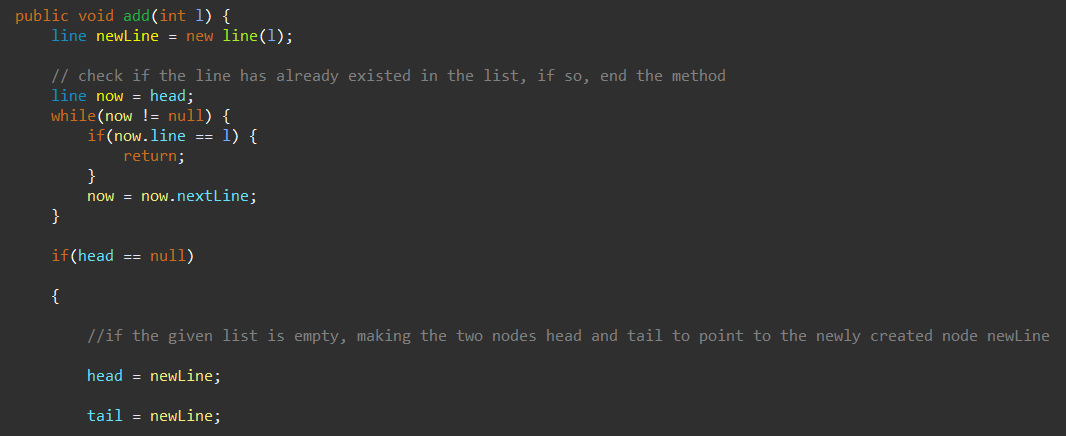
****

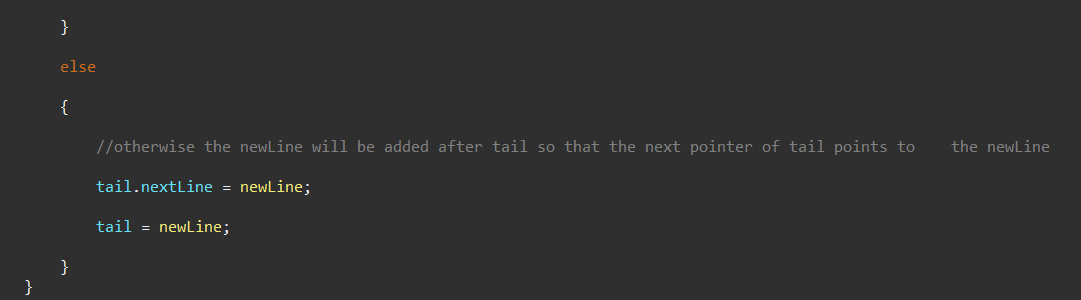
**File LL3.java**

**Cấu trúc Node line**

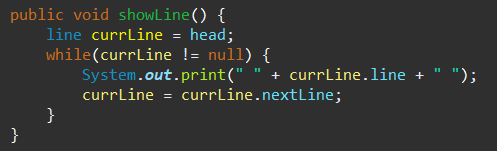
****

**Hàm Add() để thêm dòng (line) mới khi dòng đưa vào đã tồn tại**

****

****

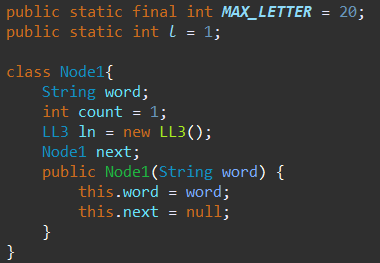
**Hàm Showline() để thực hiện in ra những dòng (Node) đã được thêm vào**

****

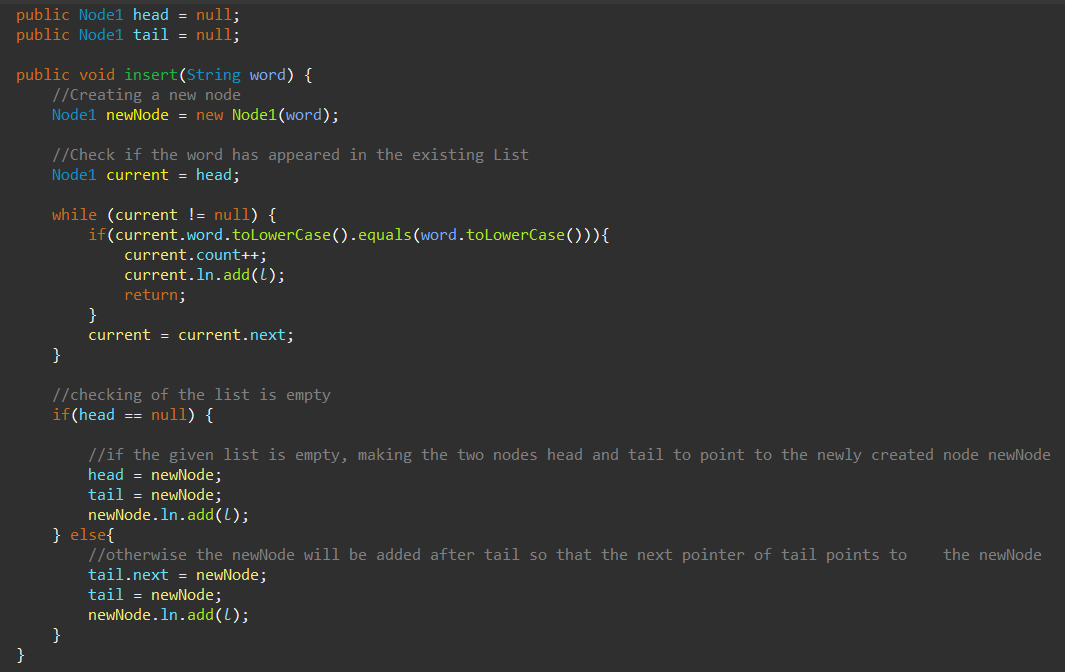
Sau khi build xong 2 file danh sách liên kết. Bắt đầu thực hiện viết chương trình cho các cấu trúc dữ liệu chính.

* + 1. **Danh sách liên kết**

**Code :**

****

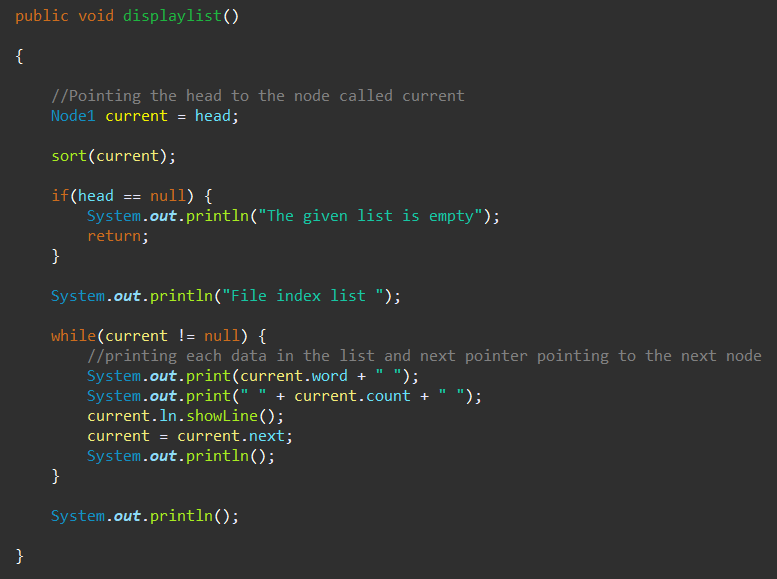
Thực hiện tạo cấu trúc **Node1** theo form của bài tập cho. (**LL3** đã được build ở trên)



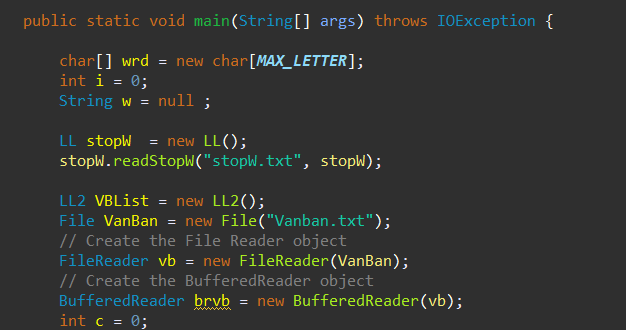
Hàm **insert()** để tạo Node mới khi gọi



Hàm **sort()** để sắp xếp theo các từ trong danh sách liên kết theo thứ tự bảng chữ cái



Hàm **displaylist()** In kết quả ra màn hình sau khi đã hoàn thành chương trình.





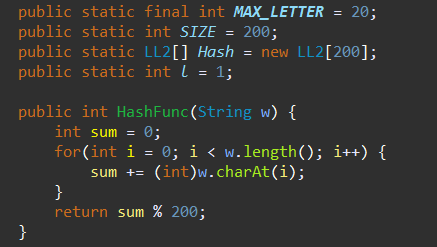
Hàm **main()** để check và lấy thông tin đầu vào cho **LL**, **LL3** và chính **LL2**.

* + 1. **Bảng băm**

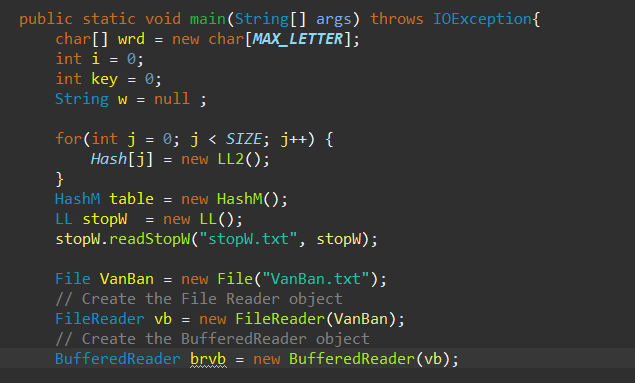
**Code :**

Trên nền danh sách liên kết của CTDL Danh sách liên kết **LL2** ở trên. Tái sử dụng **LL**, **LL2** và **LL3.**

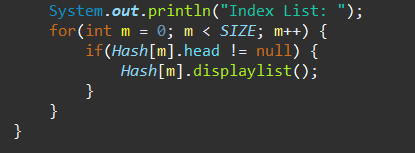
Tạo hàm **HashFunc()** tính tổng chia 200 để đưa vào bảng băm.



Thay vì hàm **main()** ở **LL2** như trên, LL2 không collect dữ liệu mà chỉ xử lý. Hàm **main()** của Bảng băm để check và lấy thông tin đầu vào cho **LL2**.

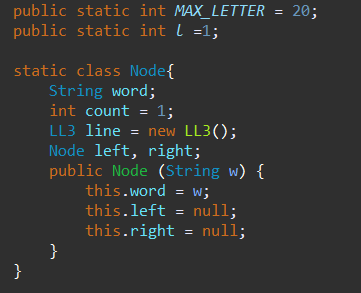






* + 1. **Cây nhị phân**

Trên nền danh sách liên kết của CTDL Danh sách liên kết. Tái sử dụng **LL** và **LL3.**

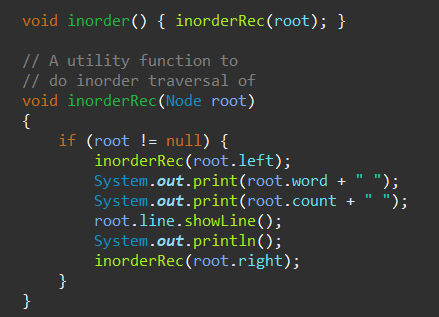
****

Tạo cấu trúc **Node** cho cây nhị phân (**Binary tree**)





Hàm **insert()** (hoặc **insertRec()**) để so sánh từng từ một ở cả lowercase và uppercase để xem từ đó có trong **root** chưa và thêm vào **root**.



Hàm **inorder()** để in kết quả ra màn hình



Hàm **main()** để check và lấy thông tin đầu vào cho **LL**, **LL3** và chính **LL2**.