Họ tên : Lý Hoài Nam

Msv: 23010773

Lớp : Ứng dụng phân tán (N02)

Báo cáo về thuật toán Chord

1. Bài toán đặt ra:

Trong một hệ thống phân tán hoặc mạng P2P, dữ liệu (file, bản ghi, đối tượng, …) được lưu rải rác trên rất nhiều máy (node). Các yêu cầu cơ bản:

Tìm kiếm nhanh: Khi người dùng đưa vào một khóa (key), hệ thống cần biết dữ liệu đó được lưu ở node nào.

Phân phối công bằng: Dữ liệu phải được chia đều giữa các node để tránh quá tải cục bộ.

Thay đổi động: Khi node mới tham gia (join) hoặc rời khỏi hệ thống (leave/failure), hệ thống vẫn phải tiếp tục hoạt động, dữ liệu không bị mất và việc tái phân phối dữ liệu chỉ nên ảnh hưởng đến một phần nhỏ.

Không có trung tâm: Hệ thống hoạt động phi tập trung, không dựa vào server duy nhất để điều phối.

Đây chính là bài toán Distributed Hash Table (DHT) – xây dựng một bảng băm phân tán trên nhiều node.

1. Ứng dụng :

a) Ánh xạ key → node

Chord cung cấp một cơ chế để xác định node lưu trữ dữ liệu từ một khóa bất kỳ.

Trong phiên bản đơn giản (O(N)), việc tìm kiếm successor của một key chỉ cần duyệt tuyến tính các node trong vòng.

Ví dụ trong test case:

Node: {1, 5, 9, 12, 20, 33, 47, 60}, key = 18 → successor là node 20.

b) Lưu trữ phân tán

Mỗi node lưu một phần dữ liệu (store key–value).

put(key, value) → tự động lưu đúng nơi successor(key).

get(key) → tự động truy xuất đến node sở hữu key.

c) Thêm hoặc bớt node (động)

Khi node join, nó chỉ lấy lại dữ liệu trong khoảng (pred(node), node].

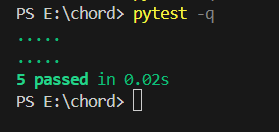
Khi node leave, nó chuyển dữ liệu của mình cho successor.

Điều này đảm bảo dữ liệu không bị mất và hệ thống tiếp tục hoạt động ổn định.

d) Khả năng mở rộng

Dù trong phiên bản O(N) lookup mất tuyến tính, khi mở rộng lên Chord chuẩn với finger table, lookup giảm xuống O(log N) – phù hợp với hàng nghìn hay hàng triệu node.

1. Minh chứng kết quả test case :

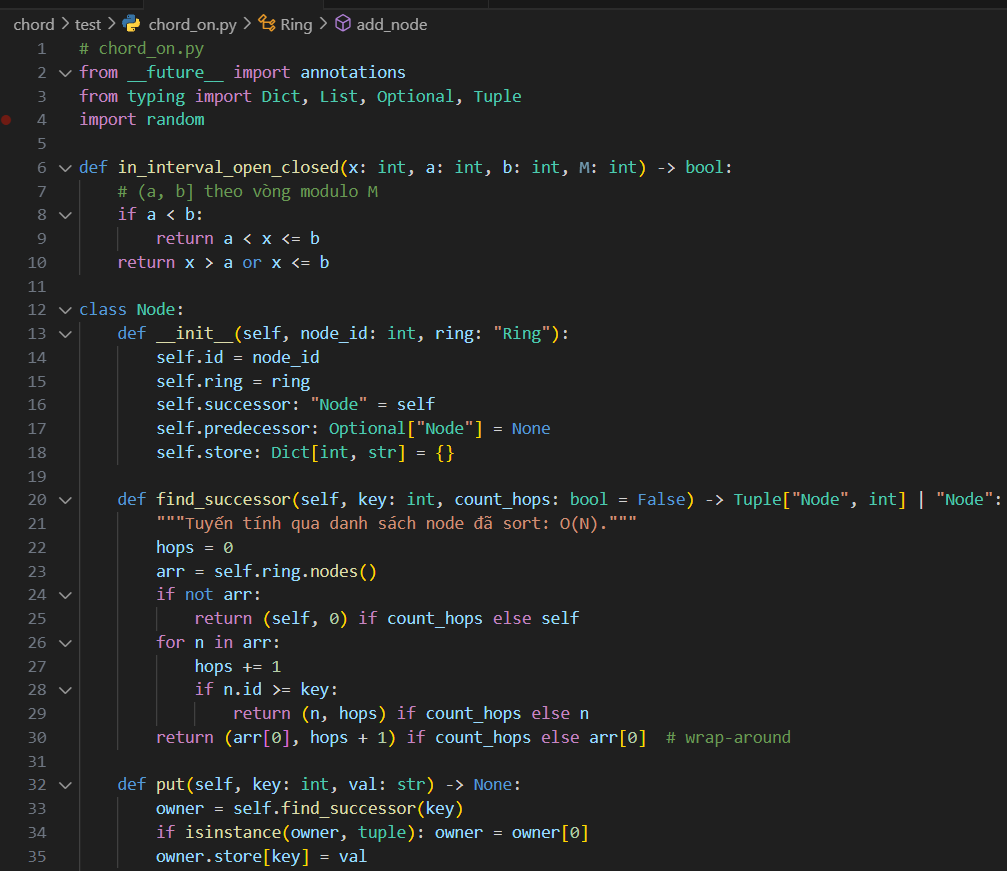


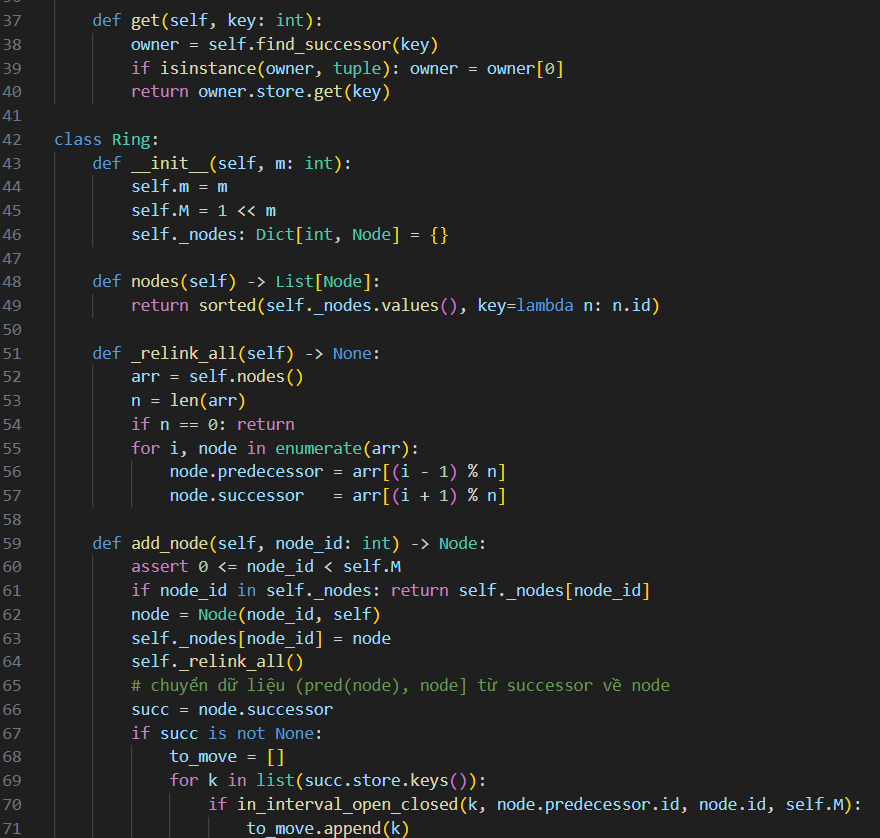
Tính toán :

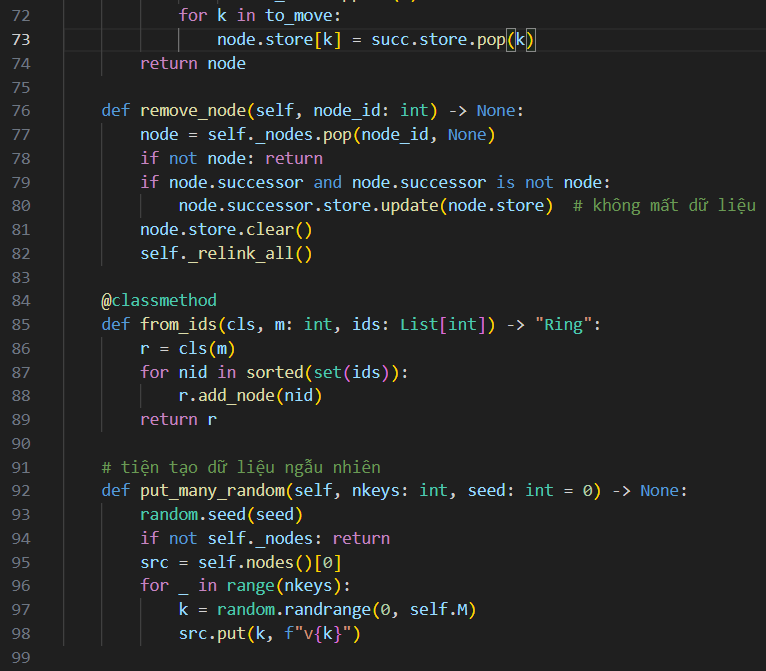
| **Tên test** | **Mục tiêu** | **Điều kiện tính Pass/Fail** | **Kết quả mong đợi** |
| --- | --- | --- | --- |
| test\_put\_get\_basic | Kiểm tra lưu trữ & truy xuất KV cơ bản | - put(3) → Node 5, put(18) → Node 20, put(62) → Node 1 (wrap). - get(3)="A", get(18)="B", get(62)="C", get(17)=None. | Dữ liệu lưu & lấy đúng successor. |
| test\_links\_ring\_invariant | Đảm bảo liên kết vòng đúng | Với mảng arr=nodes(): - arr[i].successor = arr[(i+1)%N]. - arr[i].predecessor = arr[(i-1)%N]. | Vòng successor–predecessor liền mạch. |
| test\_join\_moves\_boundary\_keys | Node join chỉ nhận key ở vùng biên | - Tổng số key trước = sau join. - Node mới chứa key thuộc (pred(node), node] (ví dụ: 11, 19 khi thêm node 20). | Không mất dữ liệu, key đúng chủ mới. |
| test\_leave\_keeps\_data\_and\_links | Node leave không làm mất dữ liệu, vòng vẫn liền | - Tổng số key trước = sau. - Node leave biến mất khỏi danh sách. - Các node còn lại có successor/predecessor đúng. | Dữ liệu bảo toàn, vòng hợp lệ. |
| test\_lookup\_hops\_linear\_bound | Đảm bảo lookup O(N) | - owner.id ∈ IDs. - 1 ≤ hops ≤ N+1 cho mọi key. | Số bước tìm kiếm bị chặn bởi N+1. |

ôCde

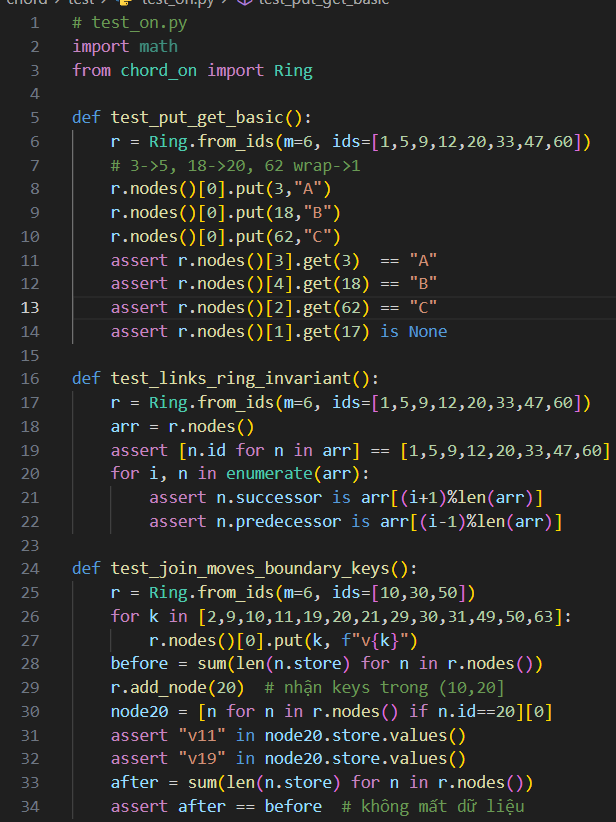
1. Code thực nghiệm :
   1. chord\_on.py:

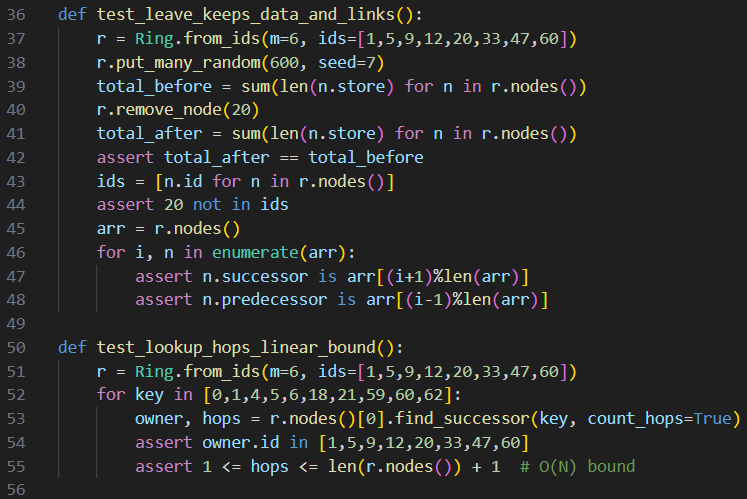






* 1. test\_on.py:





1. Link Git:

https://github.com/lyhoainamm/chord.git