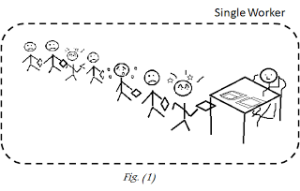
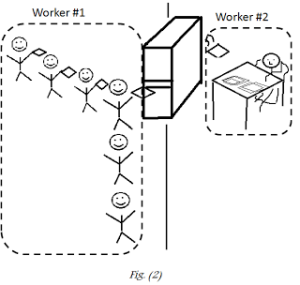
1. **MESSAGE QUEUE – Phạm Nhân**
2. **Producer–Consumer pattern**

Việc xếp hàng để thực hiện một hành vi nào đấy là rất phổ biến, ví dụ như xếp hàng mua vé tàu, người mua vé phải xếp hàng dài đợi chờ rất lâu để có thể mua vé tàu, vì việc này phải xử lý tuần tự, người này mua xong mới tới người khác. Trong kỹ thuật người ta gọi đây là xử lý Synchronized tức là xử lý đồng bộ.



Với một vài trường hợp như các tác vụ là độc lập không cần chờ nhau, thì việc này sẽ gây ra việc tắc nghẽn hệ thống vì task vụ trước phải sử lý xong với đến tác vụ sau.

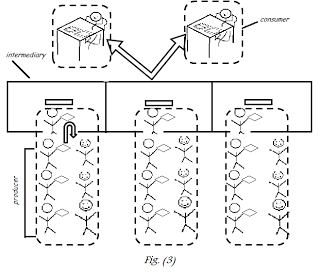
Để giải quyết tình huống này ta có thể xây dựng một hệ thống tiếp nhận yêu cầu liên tục mà không cần xử lý ngay, rồi dữ liệu sẽ được rút ra rồi xử lý dần dần như cái cabinet trong hình sau.



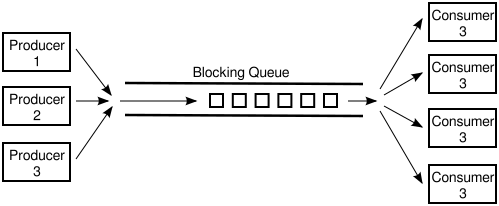
Trong kỹ thuật người ta gọi đây là xử lý bất đồng bộ **as-Synchronized**

Việc này sẽ giúp cho việc hệ thống có thể nhận yêu cầu liên tục từ “Customer” mà không cần chờ đợi, tác vụ sẽ được “Worker” xử lý dần cho đến khi các tác vụ được xử lý hết.

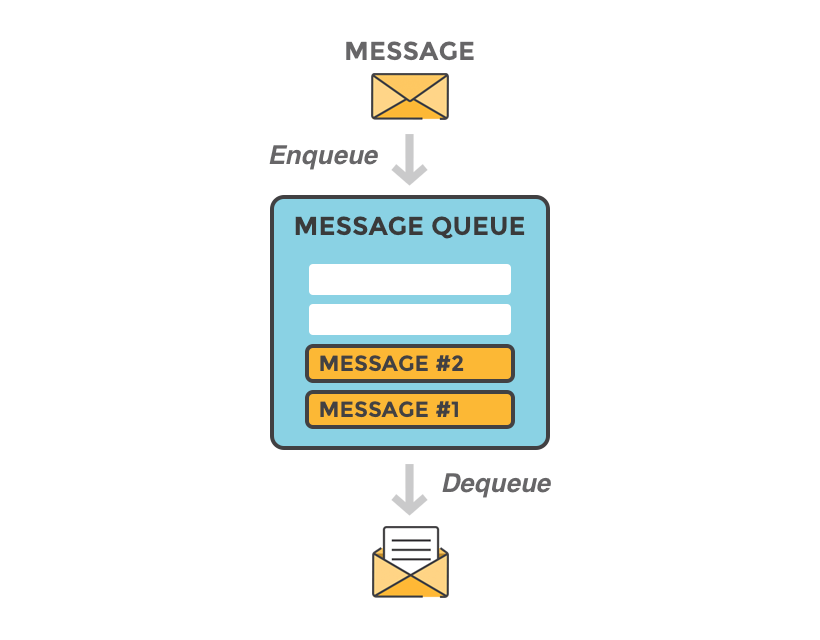
Nhưng sẽ có tình huống quá nhiều yêu cầu mà Worker không xử lý kịp, dẫn đến cabinet bị đầy và việc tiếp nhận sẽ bị chậm đi, để giải quyết việc này ta có thể bổ xung thêm nhiều Worker hoặc chia ra thành nhiều cabinet mỗi cabinet có một Worker.



Vần đề nêu trên trong lập trình người ta gọi là Producer–Consumer Problem. Producer ở đây là khách hàng, người gửi yêu cầu, còn Consumer ở đây được hiểu là Worker người xử lý yêu cầu của khách hàng, còn cái cabinet chưa yêu cầu được gọi là Queue.



1. **Khái niệm**

****

Message queue là một cơ chế trong lập trình và kiến trúc phần mềm, được sử dụng để truyền thông tin (thông điệp) giữa các thành phần của hệ thống mà không cần chúng tương tác trực tiếp với nhau [**2**](https://200lab.io/blog/message-queue-la-gi/?srsltid=AfmBOopg1xJux93ayTD3-SnaIc_HAPn5cWStm9vaEmR5-tbHSbmfrGEo). Nó hoạt động như một "hộp thư" cho phép các thành phần trong hệ thống gửi thông tin cho nhau [**1**](https://toidicodedao.com/2019/10/08/message-queue-la-gi-ung-dung-microservice/) một cách bất đồng bộ.

*Sở dĩ gọi nó là queue (hàng đợi) vì nó thực hiện việc lấy message theo cơ chế FIFO – First In First Out, tức đút vào trước thì rút ra trước.*

Lưu ý :

Message Queue (MQ) là hình thức giao tiếp giữa các dịch vụ.

Đây là **một hình thức giao tiếp**, nhưng không phải là giao tiếp trực tiếp (direct communication vd: HTTP, Grpc, …).

Thay vào đó, message queue hoạt động như một **trung gian** (middleware) để các dịch vụ trao đổi thông tin **gián tiếp**.

**Ví dụ**:

* Dịch vụ A (Producer) gửi thông điệp vào hàng đợi.
* Dịch vụ B (Consumer) lấy thông điệp từ hàng đợi để xử lý.
* Hai dịch vụ này không giao tiếp trực tiếp mà thông qua hàng đợi, nhưng đây vẫn được xem là giao tiếp giữa các dịch vụ vì thông tin được truyền tải.

1. **Các thành phần**

* **Producer**: Chương trình/service tạo ra thông tin, đưa thông tin vào message queue.
* **Consumer**:  Chương trình/service nhận message từ message queue và xử lý.
* **Message**: Thông tin được gửi đi (có thể là text, binary hoặc JSON).
* **Message queue**: Nơi lưu trữ tạm thời Message cho tới khi được Consumer lấy ra và xử lý.
* **Broker**: Xử lý Message và quản lý Message queue để đảm bảo Producer và Consumer truyền thông tin được cho nhau. Broker giúp định tuyến thông tin, quản lý tình trạng của hàng đợi, và đảm bảo rằng thông tin được chuyển giao đúng cách. => **Đây là một hệ thống trung gian (broker) để quản lý và chuyển tiếp thông điệp giữa các client.**
* **Channel**: Là cơ chế truyền thông tin giữa producer và consumer thông qua Message Queue. Channel đóng vai trò như một cầu nối để truyền thông điệp qua lại giữa các bên.

Chú ý : Một chương trình/service có thể **vừa là producer, vừa là consumer**

Note : Topic không phải là thành phần của message queue, nhưng nó là một khái niệm liên quan đến message queue và được sử dụng cùng với nó trong một số trường hợp cụ thể.

Phân biệt giữa Queue và Topic

1. **Queue** (Hàng đợi):

* Thiết kế cho giao tiếp một đối một (one-to-one)
* Tin nhắn được gửi đến một đích cụ và chỉ được đọc bởi một người tiêu dùng duy nhất
* Bảo đảm thứ tự xử lý theo FIFO (đầu tiên đến - đầu tiên được xử lý)
* Phù hợp cho việc phân phối công việc và giao tiếp điểm đến điểm [**1**](https://medium.com/version-1/queues-vs-topics-understanding-the-differences-in-messaging-frameworks-88861e2effa8)

1. **Topic** (Chủ đề):

* Thiết kế cho giao tiếp một đối nhiều (one-to-many)
* Tin nhắn được đăng lên một chủ đề và có thể được đọc bởi nhiều người đăng ký
* Cho phép phát sóng thông điệp đến tất cả các bên quan tâm
* Phù hợp cho kiến trúc dựa trên sự kiện và luồng dữ liệu thời gian thực [**1**](https://medium.com/version-1/queues-vs-topics-understanding-the-differences-in-messaging-frameworks-88861e2effa8)

Kết nối giữa Queue và Topic

* Một số hệ thống messaging cung cấp cả hai mô hình này:

+ Queue: Sử dụng cho giao tiếp một đối một

+ Topic: Sử dụng cho giao tiếp một đối nhiều và gửi thông điệp [**3**](https://www.ibm.com/topics/message-queues)

* Trong một số trường hợp, có thể sử dụng nhiều queue để phân phối tin nhắn đến nhiều đích hoặc sử dụng mô hình publish/subscribe với topic để phân phối tin nhắn [**3**](https://www.ibm.com/topics/message-queues)
* Topic không phải là thành phần của message queue, nhưng nó là một khái niệm liên quan và được sử dụng cùng với message queue trong các hệ thống messaging phức tạp hơn. Cả Queue và Topic đều là các thành phần quan trọng trong kiến trúc messaging, mỗi loại phục vụ cho các mục đích và kịch bản giao tiếp khác nhau.

1. **Phân loại**

Có thể chia thành 2 loại cơ bản là: **Message Base** và **Data Pipeline**

* So sánh:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

* Đối với loại “message base”: là những loại message queue truyền thống, thích hợp làm hệ thống trao đổi message giữa các service. **Việc đảm bảo mỗi consumer đều nhận được message và duy nhất một lần là quan trọng nhất**.
* Đối với loại “data-pipeline”, có cách lưu trữ message cũng như truyền tải message đến consumer hoàn toàn khác với hệ thống message queue truyền thống. Việc đảm bảo mỗi consumer đều phải nhận được message và duy nhất một lần không phải là ưu tiên số một, mà thay vào đó là khả năng lưu trữ message và tốc độ truyền tải message. Khi có message mới, consumer sẽ lựa chọn số lượng message mà mình muốn lấy, chính vì thế mà cùng một message consumer có thể nhận đi nhận lại nhiều lần.

**VD:** Một số loại Message base và Data Pipeline

A white background with black text

Description automatically generated

Ngoài ra, còn phân loại dựa trên một số tính năng, cơ chế hoạt động như:

1. Theo cơ chế hoạt động

**Blocking queue:** Mỗi khi một consumer lấy message, nó sẽ block cho đến khi nhận được message tiếp theo.

**Non-blocking queue**: Consumer sẽ kiểm tra xem có message mới chưa và xử lý nếu có.

1. Theo cách thức xử lý message:

**FIFO (First-In-First-Out):** Message được xử lý theo thứ tự đã nhập.

**Priority-based**: Message được xử lý dựa trên mức độ ưu tiên.

1. Theo tính chất của message

**Persistent**: Message được lưu trữ vĩnh viễn trên disk.

**Non**-**persistent**: Message chỉ được lưu trữ trong memory.

1. Theo khả năng đồng bộ

**Synchronous**: Producer và Consumer phải đồng bộ trong quá trình giao tiếp.

**Asynchronous**: Producer và Consumer không cần đồng bộ.

1. Theo kiến trúc

**Point-to-point**: Một message được gửi đến một đích cụ duy nhất.

**Publish-subscribe**: Một message được gửi đến nhiều đích.

1. Theo tính chất của broker

**Centralized: -** tập trung - Tất cả hàng đợi đều được quản lý bởi broker duy nhất

**Distributed**: - phân tán - Có thể được nhiều broker quản lý

1. Theo tính chất của message

**Typed :** Tin nhắn có cấu trúc đã được cấu hình, xác định trước

**Untyped:** Tin nhắn có cấu trúc linh hoạt.

1. Theo phương pháp lưu trữ

**Memory :** Lưu trữ trên RAM

**Disk :** Lưu trữ trên bộ nhớ

Chi tiết hơn :

**Point-to-Point Message Queues:** A diagram of a group of people

Description automatically generated

Trong mô hình này, các thông điệp được gửi từ producer đến một hàng đợi cụ thể.

**Mỗi thông điệp chỉ được tiêu thụ bởi một consumer**, đảm bảo quá trình xử lý độc quyền.

**Publish-Subscribe Message Queues:**

A diagram of a pub-sub model

Description automatically generated

Trong mô hình này, các thông điệp được các producer gửi tới các **exchanges** (**exchange** là các thành phần chịu trách nhiệm **chuyển tiếp thông điệp** từ producer đến một hoặc nhiều hàng đợi (queues) dựa trên các **quy tắc định tuyến)**.

Các exchanges phân phối thông điệp đến nhiều hàng đợi dựa trên các quy tắc định tuyến đã được xác định trước.

Nhiều consumer có thể đăng ký (subscribe) vào các hàng đợi và nhận thông điệp một cách độc lập.

**Delayed Message Queues:**

Các hàng đợi bị trì hoãn cho phép thông điệp được gửi đi sau một khoảng thời gian trì hoãn nhất định.

Các thông điệp ban đầu được giữ trong hàng đợi và chỉ được phát hành để tiêu thụ sau khi hết thời gian trì hoãn.

Mô hình này hữu ích khi cần tạo độ trễ giữa quá trình sản xuất và tiêu thụ thông điệp.

**Dead-Letter Queues:**

Hàng đợi dead-letter được sử dụng để xử lý các thông điệp không thể được xử lý hoặc gửi đi thành công.

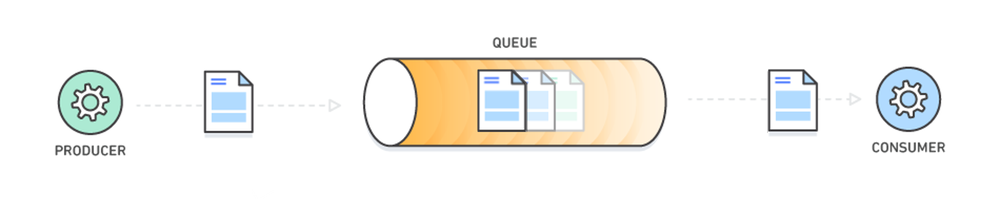
Các thông điệp gặp lỗi hoặc vượt quá giới hạn thử lại đã định trước sẽ được chuyển đến hàng đợi dead-letter để phân tích thêm.

Điều này giúp việc khắc phục sự cố và xác định các thông điệp gặp vấn đề.

A diagram of a letter

Description automatically generated

1. **Cách thức hoạt động**



**Bước 1:** Producer tạo ra thông điệp cần truyền đi. Thông tin này có thể là dữ liệu hoặc các thông tin bổ sung. Thông tin này sẽ được truyền vào Mesage queue thông qua channel và được lưu trữ tạm thời tại đây.

**Bước 2:** Consumer sẽ lấy thông điệp của Producer thông qua Message queue. Thông tin thường được được lấy theo cơ chế **FIFO (first in - first out)**, tuy nhiên vẫn có thể can thiệp vào cơ chế này bằng cách định ra các ưu tiên.

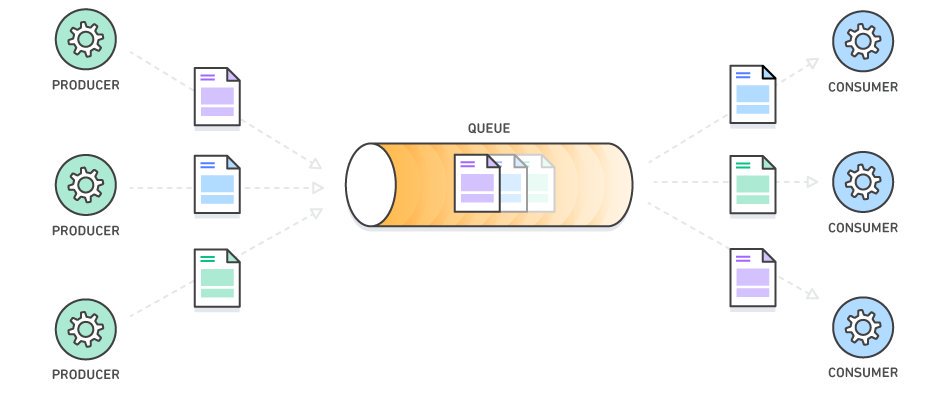
**Bước 3:** Sau khi Consumer lấy được thông tin sẽ tiếp tục xử lý và thực hiện các hành động tuỳ thuộc vào yêu cầu của hệ thống.

**--- Ví dụ :** Mình có một trang web cho phép người dùng tải từ Instagram, mình sẽ có các bộ phận sau:

* **Web service**: Là 1 producer. Nhận thông tin (url Instagram) từ phía người dùng, đưa thông tin này vào message queue
* **Processing Service**: Vừa là consumer vừa là producer. Service này đọc url Instagram từ message queue, bắt đầu tải file về và encode lại, lưu vào server. Sau khi encode xong, nó đưa url của file đã encode vào message queue
* **Uploading Service**:  Khi nhận được message từ processing server, nó sẽ upload các video đó lên Google Drive v…v

Quá trình này tạo ra một mô hình truyền và nhận thông tin linh hoạt và không đồng bộ giữa các thành phần của hệ thống. Producer và consumer **không cần biết về sự tồn tại của nhau**; thay vào đó, chúng tương tác thông qua hàng đợi được quản lý bởi Broker. Điều này giúp tăng tính mở rộng và giảm sự phụ thuộc giữa các thành phần trong hệ thống.

1. **Ứng dụng**



**Đảm bảo duration/recovery**: Do message đã được lưu trong queue, khi 1 service đang xử lý nhưng bị crash hoặc lỗi, ta không lo bị mất dữ liệu; vì có thể lấy message từ trong queue ra và chạy lại. Trong 1 hệ thống có nhiều consumer, nếu 1, 2 consume bị crash cũng không làm sụp toàn hệ thống.

**Phân tách hệ thống**: Giúp phân tách hệ thống thành nhiều service nhỏ hơn, mỗi service chỉ xử lý 1 chức năng nhất định.

**Hộ trợ rate limit, batching**: Trong nhiều trường hợp, năng lực xử lý hệ thống có hạn (chỉ có thể xử lý 300 đơn hàng/s). Với message queue, ta có thể dần dần lấy đơn hàng trong queue ra xử lý, không sợ thất lại. Hoặc thay vì mỗi lần gửi email mất thời gian lâu, ta có thể đợi message queue có yêu cầu gửi 200 email rồi gửi luôn 1 lượt.

**Dễ scaling hệ thống**: Vào giờ cao điểm, nhiều truy vấn, ta có thể tăng số lượng consumer lên để xử lý được nhiều messege hơn. Khi không cần ta có thể giảm lại.

**Message queue trong kiến trúc microservice:**

A diagram of a message

Description automatically generatedMessage queue là một kiến trúc tuyệt vời cho giao tiếp không đồng bộ. Vậy trong hệ thống microservice, khi mà ta cần các microservice vừa độc lập lại vừa có thể tương tác với nhau mà không cần ghép nối thì message queue là một lựa chọn hàng đầu.

Giao tiếp được thực hiện bằng cách gửi tin nhắn có chứa thông tin hoặc lệnh cần được xử lý. Người gửi là Producer. Những tin nhắn này sau đó được đưa vào trong một hàng đợi và được xử lý bởi một microservice khác ( Consumer). Sau đó, khi một tin nhắn được xử lý, nó sẽ bị xóa hoặc vô hiệu hóa, điều này đảm bảo rằng nó chỉ được xử lý một lần duy nhất.

1. **Ưu/Nhược điểm**

Ưu điểm của Message Queue

* **Bất đồng bộ**: Hỗ trợ truyền thông điệp giữa các thành phần mà không đòi hỏi chúng phải chờ đợi nhau.
* **Tính độc lập**: Producer và consumer không cần biết về sự tồn tại của nhau.
* **Xử lý lưu lượng cao**: Có thể xử lý lượng thông điệp lớn và đồng thời từ nhiều nguồn.
* **Đảm bảo giao tiếp tin cậy**: Thường có cơ chế đảm bảo trao đổi thông tin chính xác.
* **Giảm lỗi chồng chéo**: Loại bỏ trực tiếp kết nối giữa các thành phần.
* **Khả năng phục hồi**: Các thành phần hoạt động hoàn toàn độc lập với nhau .

Nhược điểm của Message Queue

* **Phức tạp hóa hệ thống**: Sử dụng Message Queue có thể làm phức tạp hóa hệ thống và tốn kém.
* **Độ trễ**: Việc trao đổi thông tin bất đồng bộ giữa các thành phần sẽ có một độ trễ nhất định.
* **Chi phí xử lý**: Message Queue sẽ tăng tải của hệ thống nếu phải xử lý lượng lớn thông tin.
* **Quản lý và theo dõi**: Khi hệ thống có nhiều hàng đợi, hoặc có nhiều Producer/consumer thì việc quản lý và theo dõi hoạt động của Message queue sẽ gặp nhiều khó khăn.
* **Khó xử lý đồng bộ**: Không phù hợp khi hệ thống cần xử lý đồng bộ giữa các service .

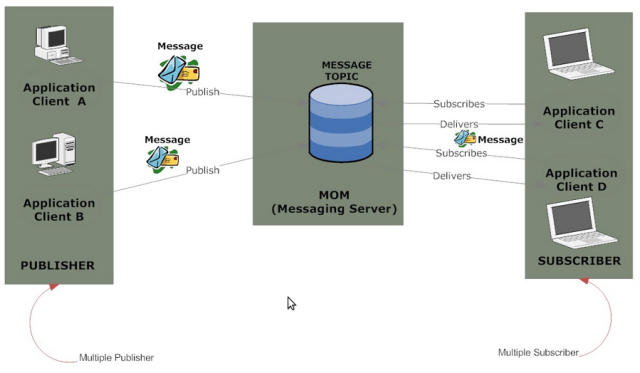
# **ACTIVE MQ – Phạm Nhân**

* 1. **Tại sao cần tới Message Queue, các khái niệm**

Để giải quyết vấn đề  Producer-Consumer ở mục đã đề cập trước đó, Active MQ được sinh ra nhằm mục đích cho phép:

* Producer gửi tin vào một queue mà không cần chờ đợi Consumer xử lý ngay
* Consumer lấy tin từ queue để xử lý dần mà không bị tắc nghẽn
* **Hỗ trợ JMS (Java Message Service)** : ActiveMQ là một trong những MOM (Message Oriented Middleware) phổ biến nhất và hỗ trợ đầy đủ các tính năng của JMS
* Tính linh hoạt cao: ActiveMQ có thể chạy độc lập hoặc được tích hợp vào các ứng dụng khác .Nó phù hợp để sử dụng trong môi trường server cũng như ứng dụng client-side.
* **ActiveMQ phù hợp khi cần:**
* Xây dựng hệ thống phân tán với nhiều thành phần cần trao đổi thông tin
* Giải quyết vấn đề đồng bộ hóa dữ liệu giữa các hệ thống
* Tạo hệ thống xử lý các yêu cầu độc lập không cần chờ đợi

- Điểm mạnh của ActiveMQ ta có thể dùng để gửi nhận message với nhiều hệ thông với các nền tảng khác nhau, đúng như tên gọi “**Hệ thống trung gian chuyển tải gói tin**”.



Các thành phần chính trong ActiveMQ

1. Producer/Publisher: Thành phần tạo và gửi tin (ActiveMQ-Client).
2. Broker trung gian hay Message Oriented Middleware (MOM) (ActiveMQ-Broker).
3. Consumer/Subcriber: Thành phần nhận tin từ Producer thông qua MOM (ActiveMQ-Client).

**https://lcdung.top/message-queue-la-gi/**

**Lưu ý: ACTIVE MQ**

* Là một hệ thống message broker đầy đủ, trong khi message broker chỉ là một khái niệm về chức năng trung gian chuyển tiếp tin nhắn giữa các thành phần của hệ thống.
* Hoạt động như một trung gian (broker) trong quá trình truyền tải tin nhắn giữa các thành phần của hệ thống, trong khi message broker chỉ ám chỉ chức năng này.
* Cung cấp nhiều tính năng như clustering, bảo mật, giám sát, trong khi message broker thường được sử dụng đơn giản hơn để lưu trữ và chuyển tiếp tin nhắn.
* Phù hợp cho các hệ thống phân tán lớn và phức tạp, trong khi message broker thường được sử dụng trong các ứng dụng đơn lẻ hoặc nhỏ hơn.
  1. **anc**