**TÀI LIỆU ÔN TẬP CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT (JAVA)**

**1. ARRAY (MẢNG)**

**1.1. Mảng là gì?**

* **Mảng (Array)** là **một tập hợp các phần tử có cùng kiểu dữ liệu**, được lưu trữ **liên tục** trong bộ nhớ và được truy cập qua **chỉ số (index)**, bắt đầu từ 0.
* Ví dụ:  
  int[] arr = {10, 20, 30};
  + arr[0] = 10
  + arr[1] = 20
  + arr[2] = 30

**1.2. Đặc điểm nổi bật**

* **Kích thước cố định**: Đã khai báo là không thay đổi (nếu muốn thay đổi phải tạo mảng mới).
* **Truy cập nhanh**: Truy cập bất kỳ phần tử nào cũng chỉ tốn thời gian O(1).
* **Lưu trữ cùng kiểu dữ liệu**: Không trộn lẫn các loại dữ liệu.

**1.3. Khai báo và khởi tạo mảng trong Java**

java

Sao chép mã

// Khai báo mảng số nguyên 5 phần tử (tất cả khởi tạo mặc định = 0)

int[] arr = new int[5];

// Khởi tạo trực tiếp

int[] arr2 = {1, 2, 3, 4, 5};

// Khai báo mảng String

String[] names = new String[3];

names[0] = "An";

names[1] = "Bình";

names[2] = "Cường";

**1.4. Duyệt mảng**

* **Duyệt bằng for**

java

Sao chép mã

for(int i = 0; i < arr.length; i++) {

System.out.println(arr[i]);

}

* **Duyệt bằng for-each**

java

Sao chép mã

for(int x : arr) {

System.out.println(x);

}

**1.5. Một số thao tác cơ bản với mảng**

* **Cập nhật giá trị**

java

Sao chép mã

arr[2] = 100; // Đổi giá trị phần tử thứ 3 thành 100

* **Tìm độ dài mảng**

java

Sao chép mã

int n = arr.length;

* **Sao chép mảng**

java

Sao chép mã

int[] copy = Arrays.copyOf(arr, arr.length);

**2. Ưu điểm và nhược điểm của mảng**

**Ưu điểm**

* **Truy cập cực nhanh** (O(1))
* Tiết kiệm bộ nhớ (không lưu thêm thông tin gì ngoài giá trị phần tử)
* Đơn giản, dễ hiểu

**Nhược điểm**

* Kích thước cố định (không linh hoạt, nếu cần mở rộng phải tạo mảng mới)
* Thao tác chèn/xóa ở giữa mảng tốn kém (phải dịch chuyển các phần tử còn lại)
* Chỉ lưu trữ **cùng kiểu dữ liệu**

**3. Các dạng bài tập phổ biến về mảng**

Đây là **những bài tập kinh điển** từ cơ bản đến nâng cao – vừa để luyện code, vừa cực hay ra phỏng vấn:

**A. Dạng duyệt và thao tác cơ bản**

1. **Tính tổng, tìm max/min các phần tử trong mảng**
2. **Đếm số lượng phần tử chẵn/lẻ, số dương/số âm**
3. **Tìm phần tử lớn thứ hai, nhỏ thứ hai**
4. **Kiểm tra mảng có tăng dần/giảm dần không**
5. **Đảo ngược mảng**
6. **Sắp xếp mảng tăng dần/giảm dần**

**B. Dạng mảng nâng cao**

1. **Tìm phần tử xuất hiện nhiều nhất**
2. **Xóa phần tử tại vị trí chỉ định**
3. **Chèn phần tử vào vị trí bất kỳ**
4. **Gộp 2 mảng thành 1 mảng**
5. **Tách các số chẵn/lẻ thành 2 mảng mới**
6. **Loại bỏ phần tử trùng lặp**

**C. Dạng tư duy/phỏng vấn (Leetcode, HackerRank…)**

1. **Tìm cặp phần tử có tổng bằng K**
2. **Dãy con liên tiếp có tổng lớn nhất (Kadane’s Algorithm)**
3. **Tìm mảng con (subarray) có tổng bằng K**
4. **Tìm phần tử xuất hiện 1 lần (trong khi các phần tử khác xuất hiện 2 lần)**
5. **Di chuyển tất cả số 0 về cuối mảng (giữ nguyên thứ tự phần tử khác 0)**
6. **Merge mảng đã sắp xếp**
7. **Tìm phần tử majority (xuất hiện > n/2 lần)**
8. **Tìm độ dài dãy con tăng liên tiếp dài nhất**

**4. Một số ví dụ code điển hình**

**Ví dụ 1: Tìm tổng các phần tử mảng**

java

Sao chép mã

int[] arr = {1, 2, 3, 4, 5};

int sum = 0;

for(int x : arr) sum += x;

System.out.println("Tổng = " + sum);

**Ví dụ 2: Tìm phần tử lớn nhất**

java

Sao chép mã

int max = arr[0];

for(int i = 1; i < arr.length; i++) {

if(arr[i] > max) max = arr[i];

}

System.out.println("Max = " + max);

**Ví dụ 3: Đảo ngược mảng**

java

Sao chép mã

for(int i = 0, j = arr.length-1; i < j; i++, j--) {

int temp = arr[i]; arr[i] = arr[j]; arr[j] = temp;

}

**Ví dụ 4: Xóa phần tử tại vị trí k**

java

Sao chép mã

int k = 2; // Xóa phần tử tại chỉ số 2

for(int i = k; i < arr.length-1; i++) {

arr[i] = arr[i+1];

}

// Giảm size logic đi 1

**5. Tổng hợp các câu hỏi phỏng vấn hay gặp về mảng**

**Lý thuyết**

* Sự khác nhau giữa Array và ArrayList là gì?
* Tại sao nên dùng mảng thay vì List/Set/Map?
* Cách duyệt mảng đa chiều?

**Bài toán thực tế/phỏng vấn**

1. **Viết hàm đảo ngược mảng mà không dùng mảng phụ**
2. **Cho mảng số nguyên, tìm tất cả cặp có tổng bằng một số K**
3. **Di chuyển tất cả số 0 về cuối mảng**
4. **Kiểm tra mảng có phần tử trùng lặp hay không?**
5. **Tìm dãy con liên tiếp có tổng lớn nhất**
6. **Cho mảng đã sắp xếp, loại bỏ phần tử trùng lặp (in-place)**
7. **Tìm phần tử xuất hiện một lần (các phần tử khác xuất hiện 2 lần) – dùng XOR**
8. **Tìm số xuất hiện nhiều nhất trong mảng**

**6. Góc luyện tập – Thử thách tư duy**

**Một số câu đố phỏng vấn cực hay về mảng:**

1. **Reverse Array in-place**  
   *Viết code đảo ngược mảng mà chỉ dùng O(1) bộ nhớ phụ.*
2. **Sum Pair**  
   *Tìm xem trong mảng có cặp nào có tổng đúng bằng K không, độ phức tạp tối ưu là bao nhiêu?*
3. **Majority Element**  
   *Tìm phần tử xuất hiện nhiều hơn n/2 lần (nếu có). Có thuật toán O(n) không dùng HashMap không?*
4. **Move Zeros**  
   *Cho mảng chứa các số, hãy chuyển hết số 0 về cuối mà không thay đổi thứ tự các phần tử còn lại. Có thể làm O(n) không?*
5. **Kadane’s Algorithm**  
   *Tìm tổng lớn nhất của một dãy con liên tiếp trong mảng. Bạn có biết đây là thuật toán nổi tiếng Kadane?*
6. **Remove Duplicates from Sorted Array (Leetcode 26)**  
   *Loại bỏ trùng lặp trong mảng đã sắp xếp, trả về độ dài mới của mảng.*
7. **Rotate Array**  
   *Dịch phải mảng k vị trí (rotate right by k steps) – không dùng mảng phụ.*

**7. Các dạng nâng cao hơn (nếu muốn thử sức)**

* Mảng 2 chiều (ma trận): Duyệt hàng/cột, xoắn ốc, tìm đường đi ngắn nhất
* Bài toán sliding window (cửa sổ trượt)
* Dãy con tăng/giảm dài nhất

**8. Lời khuyên học mảng hiệu quả**

* Hiểu rõ **chỉ số (index) bắt đầu từ 0**
* Nắm vững **for – for-each – while** để duyệt mảng
* Tự luyện **tối thiểu 10 dạng bài tập** trên Leetcode/HackerRank về Array (tham khảo tag Leetcode Array)
* Gặp vấn đề về hiệu năng, hãy nghĩ tới **O(n)**, **O(n^2)**, hoặc thuật toán tối ưu như Hash, Two Pointer, Sliding Window

**I. Bài tập mảng cơ bản**

1. **Tính tổng các phần tử trong mảng**
2. **Tìm giá trị lớn nhất/nhỏ nhất trong mảng**
3. **Tìm phần tử lớn thứ hai/nhỏ thứ hai trong mảng**
4. **Đếm số lượng phần tử chẵn/lẻ**
5. **Đếm số lượng số dương/âm/0**
6. **Tính trung bình cộng của các phần tử**
7. **Đảo ngược mảng**
8. **Kiểm tra mảng có tăng/giảm dần không**
9. **Tìm tất cả các số nguyên tố trong mảng**
10. **Đếm số lần xuất hiện của một phần tử trong mảng**

**II. Bài tập thao tác trên mảng**

1. **Xóa phần tử tại vị trí chỉ định**
2. **Chèn phần tử vào vị trí bất kỳ**
3. **Loại bỏ tất cả phần tử trùng lặp trong mảng**
4. **Tách các số chẵn/lẻ thành 2 mảng mới**
5. **Gộp 2 mảng thành 1 mảng mới**
6. **Sắp xếp mảng tăng dần/giảm dần (không dùng hàm có sẵn)**
7. **Đổi chỗ hai phần tử tại 2 vị trí bất kỳ**
8. **Tìm phần tử xuất hiện nhiều nhất trong mảng**
9. **Xoay vòng mảng (Rotate array) sang phải/trái k vị trí**
10. **Di chuyển tất cả số 0 về cuối mảng**

**III. Bài toán tư duy – phỏng vấn quốc tế (Leetcode, FAANG)**

1. **Tìm cặp phần tử có tổng bằng K trong mảng**
2. **Tìm mảng con liên tiếp có tổng lớn nhất (Kadane’s Algorithm)**
3. **Tìm tất cả các cặp phần tử có hiệu bằng một giá trị K**
4. **Tìm phần tử xuất hiện duy nhất (các phần tử khác xuất hiện 2 lần)**
5. **Kiểm tra mảng có phần tử trùng lặp không (contains duplicate)**
6. **Tìm phần tử majority (xuất hiện nhiều hơn n/2 lần)**
7. **Tìm độ dài dãy con tăng liên tiếp dài nhất (Longest Increasing Subsequence)**
8. **Loại bỏ phần tử trùng lặp trong mảng đã sắp xếp (Remove Duplicates in-place)**
9. **Tìm dãy con liên tiếp có tổng bằng K**
10. **Sắp xếp mảng chứa 0, 1, 2 (Sort Colors/Dutch National Flag Problem)**
11. **Tìm median (giá trị trung vị) của mảng đã sắp xếp**
12. **Gộp 2 mảng đã sắp xếp thành một mảng đã sắp xếp (Merge Sorted Array)**
13. **Tìm phần tử thiếu đầu tiên dương (First Missing Positive)**
14. **Sinh tất cả các dãy con (Subsets) của mảng**
15. **Tìm 3 phần tử có tổng bằng 0 (3Sum Problem)**
16. **Tìm số lần xuất hiện của mỗi phần tử**
17. **Tìm số phần tử lớn hơn tất cả các phần tử đứng trước nó**
18. **Tìm phần tử có tần suất xuất hiện nhiều hơn 1 lần**
19. **Đếm số phần tử có giá trị lớn hơn trung bình cộng**
20. **Tìm độ dài mảng con liên tiếp có tổng nhỏ hơn hoặc bằng K**

**IV. Một số dạng bài tập mảng nâng cao, luyện tư duy**

1. **Tìm tổng lớn nhất của mảng con không kề nhau**
2. **Sliding Window: Tìm tổng lớn nhất của dãy con dài k**
3. **Two Pointer: Tìm cặp gần nhất có tổng nhỏ hơn hoặc bằng K**
4. **Tìm số xuất hiện lẻ trong mảng (dùng XOR)**
5. **Chia mảng thành hai phần có tổng bằng nhau (Partition Equal Subset Sum)**
6. **Tìm phần tử xuất hiện đúng một lần khi các phần tử khác xuất hiện ba lần**
7. **Tìm giá trị min/max của mảng 2 chiều**
8. **Xoay ma trận (Rotate matrix) 90 độ**
9. **Tìm đường chéo chính, phụ của ma trận vuông**
10. **Tìm tổng/tích của 2 hoặc 3 phần tử lớn nhất trong mảng**

**V. Bài tập thực tế/áp dụng vào sản phẩm**

1. **Tìm kiếm người dùng trong danh sách (theo id, tên)**
2. **Tính tổng giá sản phẩm trong giỏ hàng**
3. **Kiểm tra các email có trùng lặp không**
4. **Tìm ngày sinh viên có điểm cao nhất/lớn hơn trung bình**
5. **Tìm người lớn tuổi nhất/nhỏ tuổi nhất**
6. **Gộp danh sách từ nhiều nguồn (mảng)**
7. **Tách list thành các nhóm nhỏ kích thước k**

**Tài liệu luyện tập mở rộng (không bắt buộc)**

* Leetcode - Array Problems
* HackerRank - Array Practice
* GeeksforGeeks - Array Interview Questions

**Tổng kết**

* Mảng là nền tảng của mọi thuật toán và cấu trúc dữ liệu, hiểu kỹ mảng sẽ giúp bạn học tốt mọi chủ đề tiếp theo!
* Học thuộc và tự code lại ít nhất 10–20 bài tập trên, bạn sẽ **không còn sợ câu hỏi về mảng khi đi phỏng vấn**!

**2. LIST (DANH SÁCH)**

**1.1. Khái niệm List**

* **List** là một **interface** trong Java Collection Framework (JCF) dùng để lưu trữ **danh sách các phần tử có thứ tự** (có index, đánh số từ 0).
* List **cho phép chứa phần tử trùng lặp** (duplicate).
* List có thể mở rộng/thu nhỏ linh hoạt (kích thước động), khác với mảng (Array) có kích thước cố định.

**1.2. Các loại List phổ biến**

Trong Java, List có một số triển khai phổ biến:

* **ArrayList:**
  + Sử dụng mảng động bên trong.
  + Truy cập phần tử nhanh (O(1)), thêm/xóa ở cuối nhanh, thêm/xóa ở giữa chậm (phải dịch chuyển các phần tử).
  + Thường dùng nhất.
* **LinkedList:**
  + Sử dụng cấu trúc danh sách liên kết (linked list).
  + Thêm/xóa ở đầu/cuối nhanh (O(1)), truy cập theo index chậm (O(n)), thêm/xóa ở giữa nhanh hơn ArrayList trong một số trường hợp.
  + Có thể hoạt động như một Stack/Queue nhờ các method riêng.
* **Vector:**
  + Tương tự ArrayList nhưng **an toàn luồng** (thread-safe), ít dùng hiện nay.

**Tip:** Khi phỏng vấn, hầu hết dùng **ArrayList**, chỉ hỏi về LinkedList khi đề cập đến thao tác thêm/xóa ở đầu/cuối.

**1.3. Khai báo, khởi tạo, và các thao tác cơ bản với List**

**Khai báo & khởi tạo**

java

Sao chép mã

import java.util.\*;

List<String> list = new ArrayList<>();

// Hoặc

List<Integer> numbers = new LinkedList<>();

**Thêm, sửa, xóa, truy xuất**

java

Sao chép mã

list.add("An"); // Thêm cuối danh sách

list.add(1, "Bình"); // Thêm vào vị trí index 1

list.set(0, "Cường"); // Sửa phần tử ở vị trí 0

list.remove("An"); // Xóa theo giá trị

list.remove(1); // Xóa theo vị trí index

String name = list.get(0); // Lấy phần tử tại index 0

int size = list.size(); // Lấy số lượng phần tử

**Duyệt List**

java

Sao chép mã

// Cách 1: for thường

for (int i = 0; i < list.size(); i++) {

System.out.println(list.get(i));

}

// Cách 2: for-each

for (String s : list) {

System.out.println(s);

}

// Cách 3: Iterator

Iterator<String> it = list.iterator();

while (it.hasNext()) {

System.out.println(it.next());

}

**Một số method hữu ích khác**

java

Sao chép mã

list.contains("An"); // Kiểm tra phần tử tồn tại

list.indexOf("An"); // Vị trí xuất hiện đầu tiên

list.lastIndexOf("An");// Vị trí xuất hiện cuối cùng

list.clear(); // Xóa toàn bộ

list.isEmpty(); // Kiểm tra rỗng

Collections.sort(list); // Sắp xếp tăng dần

**1.4. Ưu điểm và nhược điểm**

| **Đặc điểm** | **ArrayList** | **LinkedList** |
| --- | --- | --- |
| Truy cập index | Nhanh (O(1)) | Chậm (O(n)) |
| Thêm/xóa ở giữa | Chậm (O(n)) | Nhanh hơn ở đầu/cuối |
| Kích thước | Động | Động |
| Hỗ trợ Thread-safe | Không | Không |
| Bộ nhớ | Tiết kiệm hơn | Tốn bộ nhớ hơn (cần con trỏ) |

**1.5. Khi nào nên dùng List?**

* Khi muốn lưu danh sách phần tử với số lượng có thể thay đổi linh hoạt.
* Khi cần truy xuất theo chỉ số.
* Khi cần cho phép trùng lặp phần tử.
* Khi thao tác chèn/xóa ở giữa không nhiều (ArrayList).

**2. Các dạng bài tập về List – CỰC SÁT PHỎNG VẤN**

**A. Cơ bản**

1. **Tạo danh sách, nhập/xuất các phần tử**
2. **Tìm phần tử lớn nhất/nhỏ nhất**
3. **Tìm tổng/trung bình cộng các phần tử số**
4. **Đếm số phần tử thỏa điều kiện (ví dụ: > 5, là số chẵn, chứa ký tự đặc biệt...)**
5. **Tìm vị trí xuất hiện đầu/ cuối của phần tử**
6. **Kiểm tra danh sách có chứa phần tử X không**

**B. Thao tác nâng cao**

1. **Sắp xếp danh sách tăng/giảm dần**
2. **Xóa tất cả phần tử trùng lặp trong List**
3. **Chèn phần tử vào vị trí bất kỳ**
4. **Xóa tất cả phần tử thỏa điều kiện (ví dụ: < 0)**
5. **Gộp 2 List thành 1 List**
6. **Chuyển List sang mảng, mảng sang List**
7. **Đảo ngược List**
8. **Lọc ra các phần tử duy nhất (không trùng lặp)**

**C. Dạng tư duy/phỏng vấn thực tế**

1. **Tìm tất cả cặp số có tổng bằng K**
2. **Tìm chuỗi dài nhất/ngắn nhất trong List**
3. **Đếm số lần xuất hiện của mỗi phần tử**
4. **Tìm phần tử xuất hiện nhiều nhất (most frequent)**
5. **Tìm List con liên tiếp có tổng lớn nhất (sublist)**
6. **Tách List thành nhiều List nhỏ kích thước K**
7. **Chuyển List thành Set để loại trùng**
8. **Gộp các List thành một List chung (flatten)**
9. **Kiểm tra List có là Palindrome không (đối xứng)**
10. **Xoay List sang phải/trái K vị trí**

**D. Một số bài tập nâng cao/phỏng vấn FAANG**

1. **Remove duplicates in-place from sorted List**
2. **Find missing number(s) in a List from 1...N**
3. **Rotate List without using extra List**
4. **Merge K sorted Lists**
5. **Check for cyclic references in LinkedList**
6. **Reverse a LinkedList (dùng ListNode)**

**3. Ví dụ code mẫu một số bài tiêu biểu**

**A. Xóa phần tử trùng lặp trong List**

java

Sao chép mã

List<Integer> list = Arrays.asList(1, 2, 3, 2, 4, 1, 5);

Set<Integer> unique = new HashSet<>(list);

List<Integer> noDup = new ArrayList<>(unique);

System.out.println(noDup);

**B. Tìm phần tử xuất hiện nhiều nhất**

java

Sao chép mã

List<Integer> list = Arrays.asList(1,2,2,3,3,3,4,5,3);

Map<Integer, Integer> countMap = new HashMap<>();

for(int x : list) {

countMap.put(x, countMap.getOrDefault(x, 0) + 1);

}

int maxCount = 0, res = list.get(0);

for(Map.Entry<Integer, Integer> entry : countMap.entrySet()) {

if(entry.getValue() > maxCount) {

maxCount = entry.getValue();

res = entry.getKey();

}

}

System.out.println("Most frequent: " + res);

**C. Đảo ngược List**

java

Sao chép mã

Collections.reverse(list);

**D. Chuyển List sang mảng**

java

Sao chép mã

List<Integer> list = new ArrayList<>();

Integer[] arr = list.toArray(new Integer[0]);

**4. So sánh List và Array (và các collection khác)**

| **Đặc điểm** | **Array** | **List (ArrayList)** | **Set** | **Map** |
| --- | --- | --- | --- | --- |
| Kích thước | Cố định | Động | Động | Động |
| Trùng lặp | Có | Có | Không | N/A (key: không) |
| Thứ tự | Có | Có | Không (HashSet) | Không |
| Truy cập | Theo chỉ số | Theo chỉ số | Không | Theo key |
| Thao tác thêm | Khó | Dễ | Dễ | Dễ |
| Loại dữ liệu | 1 loại | 1 loại | 1 loại | Key-Value |

**5. Ứng dụng thực tế của List**

* Quản lý danh sách sản phẩm, học sinh, user
* Kết quả tìm kiếm, lịch sử truy cập
* Thao tác dữ liệu động (thêm/xóa/sửa nhanh)
* Chuyển đổi dữ liệu giữa các hệ thống (ví dụ từ DB -> List -> xử lý -> trả về List)

**6. Lời khuyên học và luyện tập List hiệu quả**

* **Nắm chắc thao tác cơ bản (add, remove, get, set, size, contains, clear)**
* Tự code lại 10-20 bài tập cơ bản và nâng cao về List (như mục 2)
* Thử dùng cả ArrayList và LinkedList cho 1 số bài để so sánh hiệu năng
* Khi gặp bài phỏng vấn về List, luôn nghĩ đến việc tận dụng **các method của Collections** (sort, reverse, frequency, shuffle...)
* Rèn luyện thêm về **Generic** (List<String>, List<Integer>...) để code linh hoạt

**Tổng kết**

* **List** là cấu trúc dữ liệu linh hoạt, mạnh mẽ và cực kỳ phổ biến trong lập trình thực tế và phỏng vấn.
* Học kỹ về List giúp bạn tự tin xử lý mọi dạng đề về danh sách, cấu trúc dữ liệu động, và là nền tảng để học các cấu trúc khác (Set, Map, Stack, Queue...).
* **Đề xuất**: luyện tập toàn bộ các bài tập ở trên, tự thử code và tối ưu các bài khó hơn.

**3. STACK (NGĂN XẾP)**

**1.1. Stack là gì?**

* **Stack** (Ngăn xếp) là một cấu trúc dữ liệu tuyến tính **theo nguyên tắc LIFO** (Last In First Out – vào sau ra trước).
* Bạn chỉ có thể **thêm (push)** hoặc **lấy ra (pop)** phần tử **ở đầu ngăn xếp**.
* Hình dung dễ nhất: như một chồng đĩa, bạn chỉ lấy ra/đặt vào ở phía trên cùng.

**1.2. Đặc điểm nổi bật của Stack**

* **Chỉ thao tác trên 1 đầu (top)**
* Phần tử được lấy ra luôn là phần tử vừa mới được thêm vào gần nhất.
* Rất hữu ích khi giải các bài toán cần "nhớ lại" dữ liệu theo thứ tự ngược với thời điểm nhập vào.

**2. Ứng dụng thực tế của Stack**

* **Undo/Redo** trong các phần mềm soạn thảo văn bản, IDE, trình duyệt...
* **Kiểm tra biểu thức dấu ngoặc** đúng/sai (parentheses matching)
* **Duyệt cây/bài toán đệ quy** (có thể dùng stack thay cho call stack)
* **Tính giá trị biểu thức toán học** (postfix, prefix, infix)
* **Duyệt đồ thị DFS**
* **Chuyển đổi cơ số (thập phân sang nhị phân, v.v.)**
* **Reverse string, reverse list**
* **Back/Forward trên trình duyệt web**

**3. Các thao tác cơ bản với Stack (Java)**

Java có sẵn class Stack trong package java.util, hoặc bạn có thể dùng Deque (ArrayDeque) với vai trò stack (khuyến khích dùng ArrayDeque cho hiệu năng tốt hơn).

**Các thao tác chính:**

* **Push:** Thêm phần tử vào đầu stack
* **Pop:** Lấy ra phần tử đầu stack (và xóa nó)
* **Peek:** Xem phần tử đầu stack (không xóa)
* **isEmpty:** Kiểm tra stack rỗng không
* **size:** Số lượng phần tử trong stack

**Ví dụ code với Stack Java:**

java

Sao chép mã

import java.util.Stack;

Stack<Integer> stack = new Stack<>();

stack.push(10);

stack.push(20);

System.out.println(stack.peek()); // 20

System.out.println(stack.pop()); // 20 (bị xóa khỏi stack)

System.out.println(stack.isEmpty()); // false

**Ví dụ code với ArrayDeque (tốt hơn):**

java

Sao chép mã

import java.util.ArrayDeque;

import java.util.Deque;

Deque<Integer> stack = new ArrayDeque<>();

stack.push(10);

stack.push(20);

System.out.println(stack.pop()); // 20

System.out.println(stack.peek()); // 10

**4. Ưu điểm và nhược điểm**

**Ưu điểm**

* Đơn giản, dễ cài đặt
* Thao tác thêm/xóa nhanh (O(1))
* Lưu trữ lịch sử thao tác, rất tiện cho undo/redo hoặc đệ quy

**Nhược điểm**

* Chỉ thao tác ở 1 đầu (không truy cập tự do như list/array)
* Không thích hợp khi cần tìm kiếm/xóa phần tử bất kỳ

**5. Các dạng bài tập và câu đố phỏng vấn kinh điển về Stack**

**A. Dạng cơ bản**

1. **Push, pop, peek, kiểm tra rỗng**
2. **Đảo ngược mảng hoặc chuỗi bằng stack**
3. **Chuyển đổi cơ số (decimal -> binary) dùng stack**
4. **Kiểm tra dấu ngoặc trong biểu thức có hợp lệ không**
5. **Tính giá trị biểu thức hậu tố (postfix expression evaluation)**
6. **Tìm giá trị lớn nhất/nhỏ nhất trên stack**
7. **Reverse từng từ trong câu**

**B. Dạng ứng dụng/phỏng vấn (kinh điển)**

1. **Kiểm tra chuỗi palindrome bằng stack**
2. **Chuyển đổi biểu thức infix → postfix hoặc prefix**
3. **Thiết kế ngăn xếp hỗ trợ lấy min/max O(1) (Min Stack/Max Stack)**
4. **Kiểm tra dãy số có thể là kết quả pop hợp lệ của một stack không?**
5. **Thiết kế 2 stack dùng chung 1 mảng**
6. **Triển khai stack dùng 2 queue (hoặc ngược lại)**
7. **Kiểm tra chuỗi dấu ngoặc đa loại ((), {}, []) có hợp lệ không (Valid Parentheses - Leetcode)**
8. **Tính diện tích lớn nhất của histogram (Largest Rectangle in Histogram)**
9. **Tìm Next Greater Element cho từng phần tử trong mảng**
10. **Thiết kế stack hỗ trợ getMin, getMax O(1)**
11. **Reverse linked list dùng stack**

**C. Bài tập luyện tư duy**

1. **Kiểm tra chuỗi chứa số nhị phân hợp lệ (0101…)**
2. **Kiểm tra đường dẫn tuyệt đối (cd/../cd/../...) có hợp lệ không**
3. **Giải bài toán “hậu tố tiền tố” (postfix, prefix evaluation/conversion)**
4. **Valid Bracket Sequence (hợp lệ với nhiều loại ngoặc và nhiều lớp lồng nhau)**
5. **Thiết kế stack bằng linked list tự viết**
6. **Làm bộ đếm undo/redo (hai stack)**

**6. Một số ví dụ code mẫu minh họa**

**A. Đảo ngược chuỗi bằng Stack**

java

Sao chép mã

import java.util.Stack;

public class ReverseStringStack {

public static String reverse(String s) {

Stack<Character> stack = new Stack<>();

for (char c : s.toCharArray())

stack.push(c);

StringBuilder sb = new StringBuilder();

while (!stack.isEmpty())

sb.append(stack.pop());

return sb.toString();

}

}

**B. Kiểm tra dấu ngoặc đúng/sai**

java

Sao chép mã

public static boolean isValidParentheses(String s) {

Stack<Character> stack = new Stack<>();

for (char c : s.toCharArray()) {

if (c == '(') stack.push(c);

else if (c == ')') {

if (stack.isEmpty()) return false;

stack.pop();

}

}

return stack.isEmpty();

}

**C. Tính biểu thức hậu tố**

java

Sao chép mã

public static int evalPostfix(String expr) {

Stack<Integer> stack = new Stack<>();

String[] tokens = expr.split(" ");

for (String token : tokens) {

if ("+-\*/".contains(token)) {

int b = stack.pop(), a = stack.pop();

switch(token) {

case "+": stack.push(a + b); break;

case "-": stack.push(a - b); break;

case "\*": stack.push(a \* b); break;

case "/": stack.push(a / b); break;

}

} else {

stack.push(Integer.parseInt(token));

}

}

return stack.pop();

}

**7. Tài liệu tự luyện & lưu ý khi học Stack**

* **Leetcode Stack Tag:**  
  https://leetcode.com/tag/stack/
* **GeeksforGeeks Stack Interview Problems:**  
  https://www.geeksforgeeks.org/stack-data-structure/
* **Hackerrank Stack:**  
  https://www.hackerrank.com/domains/tutorials/10-days-of-javascript

**4. QUEUE (HÀNG ĐỢI)**

**1.1. Queue là gì?**

* **Queue (Hàng đợi)** là **cấu trúc dữ liệu tuyến tính** hoạt động theo nguyên tắc **FIFO (First In First Out – vào trước ra trước)**.
* Phần tử được thêm vào “đuôi hàng” (rear/tail) và được lấy ra từ “đầu hàng” (front/head).
* Dễ hình dung: xếp hàng mua vé, ai đến trước được phục vụ trước.

**1.2. Đặc điểm nổi bật**

* **FIFO**: phần tử vào trước sẽ ra trước, không thể lấy ra phần tử giữa hay cuối hàng đợi một cách trực tiếp.
* Thường dùng để **xử lý tác vụ theo thứ tự**, kiểm soát luồng dữ liệu.

**2. Ứng dụng thực tế của Queue**

* **Quản lý hàng chờ khách hàng**, job (in ấn, task scheduler…)
* **Lập lịch CPU** trong hệ điều hành, luồng vào/ra trong mạng máy tính
* **BFS (Breadth First Search)** trong đồ thị/cây
* **Buffer** dữ liệu khi truyền thông tin (I/O Buffer)
* **Giải quyết bài toán luồng/băng thông mạng**

**3. Các thao tác cơ bản với Queue trong Java**

**3.1. Khởi tạo và sử dụng**

Java không có lớp Queue riêng biệt, mà cung cấp **interface Queue** (và Deque) với các lớp cài đặt như **LinkedList**, **ArrayDeque**, **PriorityQueue**.

java

Sao chép mã

import java.util.\*;

Queue<Integer> queue = new LinkedList<>();

queue.offer(10); // Thêm phần tử vào cuối (enqueue)

queue.offer(20);

System.out.println(queue.poll()); // Lấy và xóa đầu hàng (dequeue): 10

System.out.println(queue.peek()); // Xem đầu hàng (không xóa): 20

System.out.println(queue.isEmpty()); // Kiểm tra rỗng: false

System.out.println(queue.size()); // Đếm phần tử: 1

**offer()**: Thêm vào cuối.  
**poll()**: Lấy và xóa đầu hàng.  
**peek()**: Xem phần tử đầu.  
**isEmpty()**, **size()**: Kiểm tra rỗng, lấy kích thước.

**3.2. Các loại queue phổ biến trong Java**

* **LinkedList**: cài đặt Queue đơn giản nhất, có thể làm cả Stack/Queue/Deque.
* **ArrayDeque**: hiệu suất cao hơn LinkedList, không bị giới hạn kích thước ban đầu.
* **PriorityQueue**: Queue có ưu tiên, phần tử có “thứ tự” sẽ được xử lý trước.

**4. Ưu điểm và nhược điểm của Queue**

**Ưu điểm**

* Đảm bảo thứ tự xử lý đúng thực tế (ai đến trước được xử lý trước)
* Dễ cài đặt, trực quan
* Hiệu quả khi cần xử lý luồng công việc tuần tự

**Nhược điểm**

* Không thể truy cập tự do phần tử ở giữa
* Không linh hoạt bằng List/Array nếu muốn thao tác phức tạp
* Khi đầy (queue tĩnh), phải xóa mới thêm được (với queue cỡ cố định)

**5. Các dạng bài tập, dạng toán phỏng vấn thường gặp về Queue**

**A. Dạng cơ bản**

1. **Khởi tạo, thêm (enqueue), lấy ra (dequeue), kiểm tra rỗng**
2. **In ra tất cả phần tử trong hàng đợi**
3. **Đảo ngược hàng đợi (Reverse Queue)**
4. **Tìm giá trị lớn nhất/nhỏ nhất trong hàng đợi**
5. **Kiểm tra số phần tử hiện có**
6. **Tìm phần tử đầu/cuối hàng đợi**
7. **Chuyển đổi giữa List và Queue**

**B. Dạng ứng dụng thực tế**

1. **Mô phỏng hàng chờ khách hàng/quầy giao dịch**
2. **Mô phỏng hệ thống in ấn (printer queue)**
3. **Quản lý luồng job (task scheduler)**
4. **Tính thời gian chờ trung bình của các tác vụ**
5. **Tìm tổng các giá trị trong hàng đợi**

**C. Dạng tư duy/phỏng vấn thuật toán**

1. **Duyệt đồ thị BFS bằng Queue**
2. **Thiết kế Queue bằng hai Stack**
3. **Thiết kế Stack bằng hai Queue**
4. **Đảo ngược K phần tử đầu hàng đợi**
5. **Tìm phần tử đầu tiên không lặp lại trong stream ký tự**
6. **Mô phỏng vòng tròn loại người (Josephus problem)**
7. **Tìm đường đi ngắn nhất trên ma trận/cây bằng BFS**
8. **Kiểm tra chuỗi Palindrome bằng Queue + Stack**
9. **Circular Queue (Queue vòng): cài đặt bằng mảng cố định**

**D. Dạng nâng cao/phỏng vấn FAANG**

1. **Sliding Window Maximum/Minimum bằng Queue hoặc Deque**
2. **LRU Cache bằng Queue + Map**
3. **Bài toán Producer-Consumer (dùng BlockingQueue)**
4. **Đếm số lượng đảo (bài toán lũy thừa BFS)**
5. **In ra tất cả số nhị phân từ 1 đến N bằng Queue**
6. **Cài đặt PriorityQueue**
7. **Tìm số lượng phần tử “đón đầu” trong luồng dữ liệu**

**6. Ví dụ code mẫu một số bài tiêu biểu**

**A. Đảo ngược hàng đợi bằng Stack**

java

Sao chép mã

public static void reverseQueue(Queue<Integer> queue) {

Stack<Integer> stack = new Stack<>();

while (!queue.isEmpty())

stack.push(queue.poll());

while (!stack.isEmpty())

queue.offer(stack.pop());

}

**B. Duyệt BFS trên cây dùng Queue**

java

Sao chép mã

Queue<TreeNode> queue = new LinkedList<>();

queue.offer(root);

while (!queue.isEmpty()) {

TreeNode node = queue.poll();

System.out.print(node.val + " ");

if (node.left != null) queue.offer(node.left);

if (node.right != null) queue.offer(node.right);

}

**C. Cài đặt Queue bằng hai Stack**

java

Sao chép mã

class MyQueue {

Stack<Integer> in = new Stack<>();

Stack<Integer> out = new Stack<>();

void enqueue(int x) { in.push(x); }

int dequeue() {

if (out.isEmpty())

while (!in.isEmpty())

out.push(in.pop());

return out.pop();

}

}

**D. Đảo ngược K phần tử đầu hàng đợi**

java

Sao chép mã

public static void reverseK(Queue<Integer> queue, int k) {

Stack<Integer> stack = new Stack<>();

for (int i = 0; i < k; i++)

stack.push(queue.poll());

while (!stack.isEmpty())

queue.offer(stack.pop());

int size = queue.size();

for (int i = 0; i < size - k; i++)

queue.offer(queue.poll());

}

**7. Tài liệu luyện tập tự học Queue**

* Leetcode Queue Tag
* GeeksforGeeks Queue Data Structure
* HackerRank Queue Exercises

**8. Lời khuyên luyện Queue hiệu quả**

* Hiểu rõ bản chất FIFO, luôn hình dung ví dụ xếp hàng thực tế
* Dùng thành thạo các thao tác cơ bản (enqueue, dequeue, peek, isEmpty)
* Vẽ bảng trạng thái từng bước với bài tập mô phỏng queue
* Luyện ít nhất 10–15 bài từ cơ bản đến ứng dụng thuật toán BFS, stack-queue kết hợp
* Đọc kỹ các bài về **circular queue**, **priority queue**, **queue dùng 2 stack**

**Tóm tắt kiến thức cần nhớ về Queue**

| **Đặc điểm** | **Stack** | **Queue** | **PriorityQueue** | **CircularQueue** |
| --- | --- | --- | --- | --- |
| Nguyên tắc | LIFO (vào sau ra trước) | FIFO (vào trước ra trước) | Phần tử có độ ưu tiên | FIFO, vòng tròn |
| Ứng dụng | Undo, reverse, back | Hàng chờ, BFS, job | Task scheduler, heap | Đệm vòng |
| Java class | Stack, Deque | Queue, LinkedList, ArrayDeque | PriorityQueue | Tự cài đặt bằng mảng |

**5. TREE (CÂY)**

**5.1. Kiến thức lý thuyết**

**1.1. Tree là gì?**

* **Tree (Cây)** là cấu trúc dữ liệu **phân cấp** gồm các **node** (nút), kết nối với nhau bằng các **cạnh** (edge).
* Một cây luôn có một **gốc (root)**, các node còn lại là **con** (child), **cha** (parent), **lá** (leaf - không có con).
* **Không có chu trình** (cycle), không có phần tử “nối ngược lên” gốc.

**Ví dụ thực tế**: Cây phả hệ, cây thư mục máy tính, menu phân cấp, cây quyết định...

**1.2. Một số thuật ngữ trong cây**

| **Thuật ngữ** | **Ý nghĩa** |
| --- | --- |
| Node (Nút) | Một phần tử trong cây |
| Root | Nút gốc (không có cha) |
| Parent | Nút cha |
| Child | Nút con |
| Leaf | Nút lá (không có con) |
| Subtree | Cây con |
| Height | Chiều cao cây |
| Depth/Level | Độ sâu (tính từ gốc đến node đó) |
| Sibling | Các nút có chung cha |

**1.3. Phân loại cây thường gặp**

* **Binary Tree (Cây nhị phân):** Mỗi node tối đa có 2 con (trái, phải).
* **Binary Search Tree (BST - Cây tìm kiếm nhị phân):** Nhánh trái nhỏ hơn gốc, nhánh phải lớn hơn gốc.
* **AVL Tree, Red-Black Tree:** Cây cân bằng, đảm bảo tốc độ tìm kiếm tốt.
* **Heap:** Cây nhị phân thỏa mãn tính chất Heap (Max Heap, Min Heap).
* **Trie (Prefix Tree):** Lưu trữ từ điển, tìm kiếm từ khóa.
* **N-ary Tree:** Mỗi node có nhiều con.

**1.4. Ứng dụng của cây**

* **Lưu trữ dữ liệu phân cấp:** menu, hệ thống file/thư mục, cây phả hệ
* **Cơ sở cho các thuật toán tìm kiếm, sắp xếp**
* **Áp dụng trong compiler, AI, machine learning (cây quyết định)**
* **Tìm kiếm nhanh (BST, Trie)**
* **Quản lý bộ nhớ, biểu thức toán học, logic**

**2. Các thao tác cơ bản trên Tree**

* **Tạo node, chèn (insert), xóa (delete), tìm kiếm (search)**
* **Duyệt cây (traversal):**
  + **Pre-order:** Gốc → Trái → Phải
  + **In-order:** Trái → Gốc → Phải (BST: ra dãy tăng dần)
  + **Post-order:** Trái → Phải → Gốc
  + **Level-order (BFS):** Duyệt theo tầng (dùng queue)

**Ví dụ code Java – định nghĩa node cây nhị phân:**

java

Sao chép mã

class Node {

int value;

Node left, right;

public Node(int value) {

this.value = value;

left = right = null;

}

}

**3. Tổng hợp thao tác và code mẫu trên Binary Tree**

**Khởi tạo cây:**

java

Sao chép mã

Node root = new Node(10);

root.left = new Node(5);

root.right = new Node(15);

**Duyệt cây In-order (trái – gốc – phải):**

java

Sao chép mã

void inOrder(Node node) {

if (node != null) {

inOrder(node.left);

System.out.print(node.value + " ");

inOrder(node.right);

}

}

**Chèn phần tử vào BST:**

java

Sao chép mã

Node insert(Node root, int value) {

if (root == null) return new Node(value);

if (value < root.value)

root.left = insert(root.left, value);

else

root.right = insert(root.right, value);

return root;

}

**Tìm kiếm phần tử trong BST:**

java

Sao chép mã

boolean search(Node root, int key) {

if (root == null) return false;

if (root.value == key) return true;

return key < root.value ? search(root.left, key) : search(root.right, key);

}

**4. Dạng bài tập – câu đố phỏng vấn về Tree**

**A. Dạng cơ bản (mọi lập trình viên cần biết)**

1. **Duyệt cây theo các thứ tự: Pre-order, In-order, Post-order, Level-order**
2. **Tính chiều cao (height) của cây**
3. **Đếm số node, số lá (leaf), số node có 2 con**
4. **Tìm node có giá trị lớn nhất/nhỏ nhất trong cây**
5. **Kiểm tra một cây có là cây nhị phân tìm kiếm (BST) không**
6. **Tìm đường đi từ gốc đến một node bất kỳ**
7. **Tính tổng giá trị các node**
8. **Đếm số node ở mức (level) K**
9. **Tìm ancestor chung thấp nhất (Lowest Common Ancestor – LCA)**

**B. Dạng nâng cao/tư duy/phỏng vấn FAANG**

1. **Chèn, xóa node trong BST**
2. **Kiểm tra cây có cân bằng không (Balanced Binary Tree)**
3. **Kiểm tra cây đối xứng (Symmetric Tree)**
4. **Kiểm tra cây giống nhau (Same Tree)**
5. **So sánh 2 cây có giống nhau không**
6. **Tìm đường đi dài nhất (Diameter of Binary Tree)**
7. **Tìm đường đi có tổng lớn nhất/nhỏ nhất**
8. **Chuyển cây sang danh sách liên kết (Flatten Binary Tree)**
9. **Invert/đảo ngược cây (Mirror Tree)**
10. **Đếm số đường đi có tổng bằng K**
11. **Convert BST thành mảng tăng dần**
12. **Xác định cây có chứa subtree cho trước không**
13. **Số lượng đường đi từ root tới leaf có tổng bằng số cho trước**
14. **Tìm khoảng cách giữa 2 node bất kỳ**
15. **Xác định cây là Complete/Full/Perfect Binary Tree**
16. **Đếm số node đơn lẻ (node chỉ có 1 con)**

**C. Dạng thực tế/ứng dụng**

1. **Duyệt cây thư mục máy tính (file system)**
2. **Áp dụng cây vào xử lý phân quyền, menu động**
3. **Sắp xếp thông tin dạng phân cấp**

**5. Ví dụ code một số bài tập tiêu biểu**

**A. Tính tổng các node lá của cây nhị phân**

java

Sao chép mã

int sumLeaf(Node node) {

if (node == null) return 0;

if (node.left == null && node.right == null)

return node.value;

return sumLeaf(node.left) + sumLeaf(node.right);

}

**B. Kiểm tra cây nhị phân tìm kiếm (BST)**

java

Sao chép mã

boolean isBST(Node node, int min, int max) {

if (node == null) return true;

if (node.value < min || node.value > max) return false;

return isBST(node.left, min, node.value - 1) &&

isBST(node.right, node.value + 1, max);

}

// Gọi: isBST(root, Integer.MIN\_VALUE, Integer.MAX\_VALUE)

**C. Duyệt cây theo mức (Level-order - BFS)**

java

Sao chép mã

void levelOrder(Node root) {

if (root == null) return;

Queue<Node> q = new LinkedList<>();

q.offer(root);

while (!q.isEmpty()) {

Node curr = q.poll();

System.out.print(curr.value + " ");

if (curr.left != null) q.offer(curr.left);

if (curr.right != null) q.offer(curr.right);

}

}

**6. Lưu ý luyện tập và tài liệu học Tree**

* Nên **vẽ cây ra giấy** để hiểu sâu thuật toán duyệt, thao tác chèn/xóa
* Học kỹ thao tác đệ quy (gần như mọi thuật toán trên tree đều dùng đệ quy!)
* Tìm hiểu thêm về BST, Heap, Trie, AVL nếu muốn đi sâu phỏng vấn nâng cao
* Tài liệu luyện tập:
  + Leetcode Tree Problems
  + GeeksforGeeks Tree
  + Visualizer (minh họa trực quan thuật toán trên cây)

**7. Tóm tắt kiến thức cần nhớ về Tree**

| **Loại cây** | **Mỗi node có mấy con?** | **Tính chất đặc biệt** | **Ứng dụng** |
| --- | --- | --- | --- |
| Binary Tree | Tối đa 2 | Không ràng buộc | Biểu thức, duyệt, AI |
| Binary Search Tree | Tối đa 2 | Trái < Gốc < Phải | Tìm kiếm nhanh |
| AVL, Red-Black Tree | Tối đa 2 | Tự cân bằng, giữ tốc độ tốt | DB, OS, tìm kiếm động |
| Heap | Tối đa 2 | Max/Min ở gốc | Cấp phát bộ nhớ, PriorityQ |
| Trie | Tùy ý | Lưu trữ theo tiền tố | Tìm kiếm từ khóa, autocomplete |
| N-ary Tree | Tùy ý | Không giới hạn số con | Menu, tổ chức, cây file |

**6. SET (TẬP HỢP)**

**6.1. Kiến thức lý thuyết**

**1.1. Set là gì?**

* **Set** là một **cấu trúc dữ liệu lưu trữ các phần tử duy nhất** (không cho phép trùng lặp).
* Không đảm bảo thứ tự các phần tử (tuỳ loại Set, xem thêm phía dưới).
* Thường dùng để **lọc trùng**, **tìm kiếm nhanh**, **lưu các giá trị duy nhất**.

**Ví dụ thực tế:** Danh sách email không trùng, số điện thoại duy nhất, tập ID sản phẩm…

**1.2. Đặc điểm nổi bật của Set**

* **Không chứa phần tử trùng lặp.**
* Có thể chứa tối đa 1 giá trị null (HashSet).
* Các thao tác thêm, xóa, kiểm tra phần tử rất nhanh (O(1) với HashSet).
* Không cho truy cập theo chỉ số (index).

**2. Các loại Set phổ biến trong Java**

| **Loại Set** | **Đặc điểm chính** | **Ứng dụng** |
| --- | --- | --- |
| **HashSet** | Nhanh nhất (dựa trên bảng băm), không có thứ tự | Lọc trùng, kiểm tra tồn tại, truy cập nhanh |
| **LinkedHashSet** | Lưu thứ tự thêm vào (insertion order) | Cần loại trùng + giữ thứ tự nhập ban đầu |
| **TreeSet** | Lưu các phần tử theo thứ tự tăng dần (sắp xếp) | Lọc trùng + duyệt theo thứ tự, toán học |

**Khi phỏng vấn, chủ yếu dùng HashSet hoặc TreeSet (khi cần sắp xếp).**

**3. Khai báo và thao tác cơ bản với Set trong Java**

java

Sao chép mã

import java.util.\*;

Set<Integer> set = new HashSet<>();

set.add(10);

set.add(20);

set.add(10); // Không thêm lại, vì đã có

set.remove(20); // Xóa phần tử

set.contains(10); // Kiểm tra tồn tại

set.size(); // Số phần tử

set.isEmpty(); // Kiểm tra rỗng

for (int x : set) {

System.out.println(x);

}

* **Thêm:** set.add(x)
* **Xóa:** set.remove(x)
* **Kiểm tra:** set.contains(x)
* **Xóa tất cả:** set.clear()

**4. Ứng dụng thực tế của Set**

* **Lọc danh sách khỏi các phần tử trùng lặp**  
  (ví dụ: email, username, mã đơn hàng…)
* **Kiểm tra nhanh một giá trị có tồn tại hay không**
* **Xử lý toán học tập hợp:** giao, hợp, hiệu (intersection, union, difference)
* **Tìm số lượng giá trị duy nhất**
* **Chuyển List/Array thành tập hợp giá trị không trùng**
* **Bài toán tần suất xuất hiện, đếm distinct**

**5. Các thao tác nâng cao với Set**

* **Union (Hợp):** set1.addAll(set2);
* **Intersection (Giao):** set1.retainAll(set2);
* **Difference (Hiệu):** set1.removeAll(set2);
* **Chuyển List sang Set:** new HashSet<>(list)
* **Chuyển Set sang List:** new ArrayList<>(set)

**6. Ưu nhược điểm của Set**

| **Ưu điểm** | **Nhược điểm** |
| --- | --- |
| Loại trùng lặp, tìm kiếm rất nhanh | Không truy cập được theo chỉ số |
| Đơn giản, dễ dùng | Không đảm bảo thứ tự (HashSet) |
| Toán học tập hợp mạnh mẽ | Không lưu phần tử trùng |

**7. Các dạng bài tập, câu hỏi phỏng vấn kinh điển về Set**

**A. Cơ bản**

1. **Loại bỏ phần tử trùng lặp trong List hoặc Array (distinct)**
2. **Kiểm tra xem một phần tử đã xuất hiện trong dãy chưa**
3. **Đếm số lượng giá trị khác nhau trong dãy (distinct count)**
4. **Tìm giao, hợp, hiệu của 2 tập hợp**
5. **Kiểm tra 2 dãy có phần tử chung không**

**B. Thao tác nâng cao/phỏng vấn thực tế**

1. **Kiểm tra mảng có phần tử trùng lặp không (Contains Duplicate)**
2. **Tìm phần tử đầu tiên bị lặp lại trong dãy**
3. **Tìm tất cả các giá trị xuất hiện nhiều hơn 1 lần**
4. **Cho 2 list, tìm các phần tử chung (intersection)**
5. **Cho nhiều list, tìm phần tử xuất hiện ở tất cả list**
6. **Kiểm tra một chuỗi có chứa tất cả ký tự chữ cái không (pangram)**
7. **Kiểm tra password có ký tự trùng hay không**
8. **Tìm số lượng từ khác nhau trong một văn bản**
9. **Tìm các giá trị chỉ xuất hiện đúng 1 lần (unique)**
10. **Tìm tất cả cặp số có tổng bằng K (dùng Set để tối ưu)**

**C. Dạng nâng cao/phỏng vấn quốc tế**

1. **Tìm dãy con (subarray) có tổng bằng K (Sum = K) dùng Set (prefix sum)**
2. **Bài toán sliding window tìm số lượng phần tử duy nhất trên mỗi cửa sổ**
3. **Tìm “missing number” từ 1...n (Missing Number Problem)**
4. **Tìm tất cả permutation (hoán vị) khác nhau**
5. **Kiểm tra tất cả các phần tử có khác nhau không (All Unique)**

**8. Một số ví dụ code tiêu biểu**

**A. Loại bỏ phần tử trùng lặp trong List**

java

Sao chép mã

List<Integer> list = Arrays.asList(1, 2, 2, 3, 4, 3, 5);

Set<Integer> set = new HashSet<>(list);

System.out.println(set); // Output: [1, 2, 3, 4, 5]

**B. Tìm phần tử trùng đầu tiên**

java

Sao chép mã

Set<Integer> seen = new HashSet<>();

for (int x : arr) {

if (seen.contains(x)) {

System.out.println("First duplicate: " + x);

break;

}

seen.add(x);

}

**C. Giao, hợp, hiệu của 2 tập hợp**

java

Sao chép mã

Set<Integer> a = new HashSet<>(Arrays.asList(1,2,3));

Set<Integer> b = new HashSet<>(Arrays.asList(2,3,4));

// Union

Set<Integer> union = new HashSet<>(a); union.addAll(b);

// Intersection

Set<Integer> inter = new HashSet<>(a); inter.retainAll(b);

// Difference

Set<Integer> diff = new HashSet<>(a); diff.removeAll(b);

System.out.println(union); // [1,2,3,4]

System.out.println(inter); // [2,3]

System.out.println(diff); // [1]

**9. Lưu ý học và luyện tập về Set**

* Hiểu rõ Set dùng cho bài toán cần **lưu trữ không trùng lặp**  
  (nếu cần giữ thứ tự: LinkedHashSet; nếu cần sắp xếp: TreeSet)
* Các bài toán tối ưu tìm kiếm, kiểm tra tồn tại đều nên nghĩ tới Set!
* Dùng Set để tăng hiệu năng khi bài toán yêu cầu check trùng nhiều lần.
* Khi phỏng vấn, **câu Contains Duplicate/Unique là “signature” của Set**.
* Luyện tập thao tác giữa List ↔ Set, và các phép toán hợp-giao-hiệu.

**10. Tài liệu tự luyện**

* Leetcode Set/HashSet Problems
* GeeksforGeeks Set
* [JavaDoc HashSet](https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html)

**Tóm tắt kiến thức cần nhớ về Set**

| **Đặc điểm** | **Array/List** | **Set (HashSet)** | **LinkedHashSet** | **TreeSet** |
| --- | --- | --- | --- | --- |
| Trùng lặp | Có | Không | Không | Không |
| Thứ tự | Có | Không | Có (theo thêm) | Có (sắp xếp) |
| Truy cập theo chỉ số | Có | Không | Không | Không |
| Hiệu năng | Trung bình | Rất nhanh | Nhanh | Trung bình |
| Phép toán tập hợp | Không | Có | Có | Có |

**7. MAP (BẢN ĐỒ, ÁNH XẠ KHÓA–GIÁ TRỊ)**

**1.1. Map là gì?**

* **Map** là một **cấu trúc dữ liệu ánh xạ (key-value)**, mỗi phần tử gồm **1 cặp khóa (key)** và **1 giá trị (value)**.
* **Khóa là duy nhất** (unique), không trùng lặp.
* Dễ hình dung như: Từ điển (từ → nghĩa), Bản đồ (id → đối tượng), danh bạ (số điện thoại → tên),...

**Ví dụ:** Map<String, Integer> ageMap = {"An"→20, "Bình"→21}

**1.2. Đặc điểm nổi bật**

* **Tìm kiếm, cập nhật, xóa theo khóa rất nhanh** (O(1) với HashMap)
* Mỗi khóa chỉ gắn với một giá trị (nếu thêm cùng khóa, giá trị mới ghi đè)
* **Không cho truy cập trực tiếp theo chỉ số (index)**
* Cho phép value trùng lặp, nhưng key không trùng

**2. Các loại Map phổ biến trong Java**

| **Loại Map** | **Đặc điểm chính** | **Ứng dụng** |
| --- | --- | --- |
| **HashMap** | Phổ biến nhất, rất nhanh (bảng băm), không thứ tự | Truy xuất dữ liệu nhanh, ánh xạ id, thống kê |
| **LinkedHashMap** | Giữ thứ tự thêm vào | Lưu cache, thống kê theo thứ tự xuất hiện |
| **TreeMap** | Các khóa luôn sắp xếp tăng dần (theo Comparable) | Sắp xếp, ánh xạ cần duyệt theo thứ tự tăng |
| **Hashtable** | An toàn luồng (thread-safe), ít dùng hiện nay | Thường thay bằng ConcurrentHashMap (đa luồng) |
| **ConcurrentHashMap** | Cho phép nhiều luồng truy cập đồng thời, cực mạnh | Hệ thống đa luồng, server |

**HashMap** là lựa chọn mặc định khi phỏng vấn hoặc đi làm.

**3. Khai báo và các thao tác cơ bản với Map**

java

Sao chép mã

import java.util.\*;

Map<String, Integer> ageMap = new HashMap<>();

ageMap.put("An", 20); // Thêm/ghi đè

ageMap.put("Bình", 21);

int age = ageMap.get("An"); // Lấy value theo key

ageMap.containsKey("An"); // Kiểm tra có khóa

ageMap.containsValue(21); // Kiểm tra có value

ageMap.remove("An"); // Xóa theo key

ageMap.size(); // Số cặp key-value

ageMap.isEmpty(); // Kiểm tra rỗng

// Duyệt qua tất cả cặp key-value:

for (Map.Entry<String, Integer> entry : ageMap.entrySet()) {

System.out.println(entry.getKey() + " - " + entry.getValue());

}

// Duyệt key:

for (String key : ageMap.keySet()) {

System.out.println(key);

}

// Duyệt value:

for (Integer value : ageMap.values()) {

System.out.println(value);

}

**4. Ứng dụng thực tế của Map**

* **Đếm tần suất xuất hiện (frequency)**: mỗi từ, ký tự, id, sản phẩm...
* **Ánh xạ đối tượng (ID → info, username → user object)**
* **Thống kê, gom nhóm (group by, histogram)**
* **Cache dữ liệu**
* **Tìm kiếm nhanh (O(1))**
* **Chuyển đổi dữ liệu dạng key–value**

**5. Các thao tác nâng cao với Map**

* **putIfAbsent(key, value):** Thêm nếu key chưa tồn tại
* **getOrDefault(key, defaultValue):** Lấy value hoặc giá trị mặc định nếu không có
* **replace(key, newValue):** Ghi đè value mới cho key
* **remove(key, value):** Xóa cặp nếu key và value đúng
* **forEach((k, v) -> ...):** Duyệt map với lambda (Java 8+)
* **groupingBy, partitioningBy (Stream API):** Gom nhóm/phân loại
* **Sắp xếp Map theo key/value (stream, TreeMap, List)**

**6. Ưu nhược điểm của Map**

| **Ưu điểm** | **Nhược điểm** |
| --- | --- |
| Truy xuất theo khóa cực nhanh | Không truy cập theo index |
| Lưu trữ cặp key-value linh hoạt | Không cho key trùng |
| Gom nhóm/thống kê mạnh mẽ | Không dùng khi cần thứ tự cụ thể\* |

(\*) Dùng LinkedHashMap/TreeMap nếu cần thứ tự hoặc sắp xếp.

**7. Các dạng bài tập, dạng toán phỏng vấn kinh điển về Map**

**A. Bài tập cơ bản**

1. **Đếm tần suất xuất hiện của phần tử (word count, character count)**
2. **Kiểm tra phần tử xuất hiện nhiều nhất (Most Frequent Element)**
3. **Tìm key theo value, value theo key**
4. **Đổi key và value cho nhau (reverse map)**
5. **Kiểm tra 2 mảng có phần tử chung không (dùng Map)**
6. **Tìm tất cả value trùng nhau**

**B. Bài tập thao tác phức tạp**

1. **Group by: Gom nhóm các phần tử theo một thuộc tính (vd: group student by class)**
2. **Merge 2 Map lại thành 1 Map tổng hợp**
3. **Tìm phần tử có value lớn nhất/nhỏ nhất**
4. **Đảo ngược map (chú ý value trùng)**
5. **So sánh 2 Map (bằng nhau về key, value)**
6. **Tìm value xuất hiện nhiều hơn n/2 lần (Majority Element)**
7. **Sắp xếp Map theo key hoặc theo value**
8. **Tìm 2 số trong mảng có tổng bằng k (Two Sum dùng Map)**
9. **Tạo Histogram từ một tập hợp**
10. **Đếm số lần xuất hiện của mỗi từ trong 1 file văn bản**
11. **Lập bản đồ id → đối tượng (studentId → Student object)**

**C. Dạng phỏng vấn FAANG**

1. **LRU Cache (Least Recently Used) dùng LinkedHashMap**
2. **Anagram group (gom nhóm chuỗi cùng ký tự)**
3. **Đếm cặp key–value thỏa mãn điều kiện (ví dụ: giá trị > 10)**
4. **Biến Map thành List, hoặc List thành Map (Stream API)**
5. **Check substring isomorphism, mapping pattern**
6. **HashMap dùng cho sliding window (count các window hợp lệ)**
7. **Design data structure hỗ trợ O(1) cho insert, remove, getRandom**

**8. Một số ví dụ code tiêu biểu**

**A. Đếm tần suất phần tử**

java

Sao chép mã

int[] arr = {1,2,2,3,4,2};

Map<Integer, Integer> freq = new HashMap<>();

for (int x : arr)

freq.put(x, freq.getOrDefault(x, 0) + 1);

System.out.println(freq); // {1=1, 2=3, 3=1, 4=1}

**B. Tìm phần tử xuất hiện nhiều nhất**

java

Sao chép mã

int maxCount = 0, res = 0;

for (Map.Entry<Integer, Integer> e : freq.entrySet()) {

if (e.getValue() > maxCount) {

maxCount = e.getValue();

res = e.getKey();

}

}

System.out.println("Most frequent: " + res);

**C. Gom nhóm theo thuộc tính**

java

Sao chép mã

class Student { String name; int grade; /\*...\*/ }

List<Student> students = ...;

Map<Integer, List<Student>> group = new HashMap<>();

for (Student s : students) {

group.computeIfAbsent(s.grade, k -> new ArrayList<>()).add(s);

}

**D. Tìm cặp số có tổng bằng K**

java

Sao chép mã

int[] arr = {2, 7, 11, 15}; int k = 9;

Map<Integer, Integer> map = new HashMap<>();

for (int i = 0; i < arr.length; i++) {

if (map.containsKey(k - arr[i])) {

System.out.println("Found: " + arr[i] + ", " + (k-arr[i]));

break;

}

map.put(arr[i], i);

}

**9. Lưu ý học và luyện Map hiệu quả**

* **Nắm chắc cách thao tác: put, get, containsKey, remove, duyệt entrySet**
* Dùng Map cho các bài toán “đếm”, gom nhóm, ánh xạ, lookup cực nhanh
* Khi bài toán cần tối ưu tốc độ tra cứu, hãy nghĩ đến Map
* Khi phỏng vấn, **các bài word count, group by, two sum, LRU cache, majority** đều dùng Map
* Thử luyện code bằng các class Map khác nhau để cảm nhận ưu/nhược điểm

**10. Tài liệu tự luyện**

* Leetcode Map/HashMap problems
* GeeksforGeeks Map in Java
* [JavaDoc Map](https://docs.oracle.com/javase/8/docs/api/java/util/Map.html)

**Tóm tắt so sánh Map và các Collection khác**

| **Đặc điểm** | **List/Array** | **Set (HashSet)** | **Map (HashMap)** |
| --- | --- | --- | --- |
| Trùng lặp | Có | Không | Key: Không, Value: Có |
| Thứ tự | Có | Không/tuỳ | Không/tuỳ (LinkedHashMap) |
| Truy cập | Theo index | Không | Theo key |
| Lưu cặp | Không | Không | Có (key-value) |
| Ứng dụng | Danh sách | Tập hợp | Tra cứu, thống kê, ánh xạ |